# CSE-5306
# DISTRIBUTED SYSTEMS

**PROJECT – 2**
Date: 11/06/2023

**Submitted by:**

**Name**: Mohammad Shahedur Rahman ID: 1002157980

**Name**: Mehzad Hossain Setu ID:1002138330

I have neither given nor received unauthorized assistance on this work.

**Signed:**

| | |
|---|---|
| Signed: Mohammad Shahedur Rahman | Date: 11/06/2023 |
| Signed: Mehzad Hossain Setu | Date: 11/06/2023 |

## Table of Contents

## Introduction

The folders contain all the source codes of the programs separated by Assignment name and title. The Readme files in each folder contain instructions on how to run the programs for ready reference.

In this programming project, we have practiced programming related to Total Order Multicasting with Lamport's Algorithm, Vector Clock Algorithm, and Distributed Locking algorithm.

We have used the below-mentioned tools and programming language for this project 2.

1. Oracle VM Box 7.4.10
2. Ubuntu 22.04.3 LTS
3. Python 3.11
4. Other relevant packages.

# Assignment 1

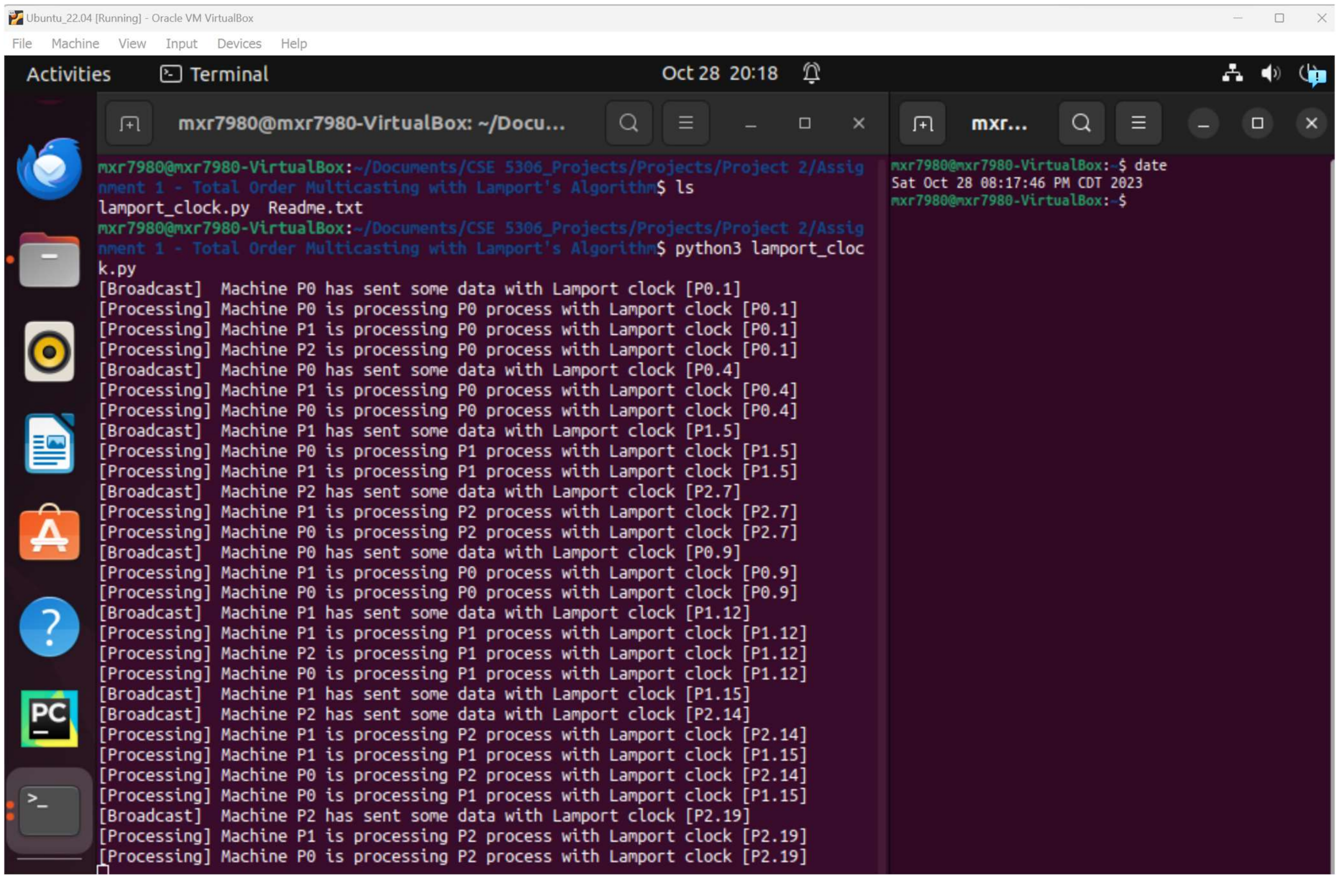
## Total Order Multicasting with Lamport's Algorithm

In this assignment, we were tasked to implement a Totally Ordered Multicasting system using Lamport's algorithm. We implemented the algorithm and simulated the output using Python 3.11. The program is implemented in a single file – lamport_clock.py

We employed UDP socket-based communication to facilitate communication between processes. A set number of processes are generated by our primary function (3, this can be easily changed to allow an arbitrary number of processes). We have defined the port numbers as well because, in our scenario, the number of processes is fixed. The Python Multiprocessing module is used to implement each process. The clock and machine operations are carried out by this module. To receive events and communicate with the other processes, a communication thread is also present.
To imitate the operation of a process, we've included a time delay in our software. The process increases its logical clock whenever an event happens and multicasts the event to all other running processes in the network. The other processes increase their logical clocks and add the event to a buffer queue when they receive an event. The message that the communication thread receives determines how it will respond.

- If the received message is an event –
    1. It updates its logical clock by comparing its own clock with the timestamp in the received event
    2. It puts the event in a priority queue, which is a heap with the event's timestamp as the comparison key.
    3. It broadcasts the acknowledgment for the event at the front of the queue.
- If the received message is an acknowledgment –
    1. It increments the acknowledgment count for that event
    2. If the event at the front of the queue has received acknowledgments from all other processes, then that event is popped from the heap and processed. Otherwise, it broadcasts the acknowledgment

The sample output of the program runtime is shown below:



Ubuntu_22.04 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Oct 28 20:18
mxr7980@mxr7980-VirtualBox: ~/Docu...
mxr...

```
mxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Projects/Projects/Project 2/Assig
nment 1 - Total Order Multicasting with Lamport's Algorithm$ ls
lamport_clock.py  Readme.txt
mxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Projects/Projects/Project 2/Assig
nment 1 - Total Order Multicasting with Lamport's Algorithm$ python3 lamport_cloc
k.py
[Broadcast]  Machine P0 has sent some data with Lamport clock [P0.1]
[Processing] Machine P0 is processing P0 process with Lamport clock [P0.1]
[Processing] Machine P1 is processing P0 process with Lamport clock [P0.1]
[Processing] Machine P2 is processing P0 process with Lamport clock [P0.1]
[Broadcast]  Machine P0 has sent some data with Lamport clock [P0.4]
[Processing] Machine P1 is processing P0 process with Lamport clock [P0.4]
[Processing] Machine P0 is processing P0 process with Lamport clock [P0.4]
[Broadcast]  Machine P1 has sent some data with Lamport clock [P1.5]
[Processing] Machine P0 is processing P1 process with Lamport clock [P1.5]
[Processing] Machine P1 is processing P1 process with Lamport clock [P1.5]
[Broadcast]  Machine P2 has sent some data with Lamport clock [P2.7]
[Processing] Machine P1 is processing P2 process with Lamport clock [P2.7]
[Processing] Machine P0 is processing P2 process with Lamport clock [P2.7]
[Broadcast]  Machine P0 has sent some data with Lamport clock [P0.9]
[Processing] Machine P1 is processing P0 process with Lamport clock [P0.9]
[Processing] Machine P0 is processing P0 process with Lamport clock [P0.9]
[Broadcast]  Machine P1 has sent some data with Lamport clock [P1.12]
[Processing] Machine P1 is processing P1 process with Lamport clock [P1.12]
[Processing] Machine P2 is processing P1 process with Lamport clock [P1.12]
[Processing] Machine P0 is processing P1 process with Lamport clock [P1.12]
[Broadcast]  Machine P1 has sent some data with Lamport clock [P1.15]
[Broadcast]  Machine P2 has sent some data with Lamport clock [P2.14]
[Processing] Machine P1 is processing P2 process with Lamport clock [P2.14]
[Processing] Machine P1 is processing P1 process with Lamport clock [P1.15]
[Processing] Machine P0 is processing P2 process with Lamport clock [P2.14]
[Processing] Machine P0 is processing P1 process with Lamport clock [P1.15]
[Broadcast]  Machine P2 has sent some data with Lamport clock [P2.19]
[Processing] Machine P1 is processing P2 process with Lamport clock [P2.19]
[Processing] Machine P0 is processing P2 process with Lamport clock [P2.19]
```

```
mxr7980@mxr7980-VirtualBox:~$ date
Sat Oct 28 08:17:46 PM CDT 2023
mxr7980@mxr7980-VirtualBox:~$
```

The order in which the machines process the message might change, but the clocks of all the machines are always synchronized to the same value.

# Assignment 2

## Vector Clock Algorithm

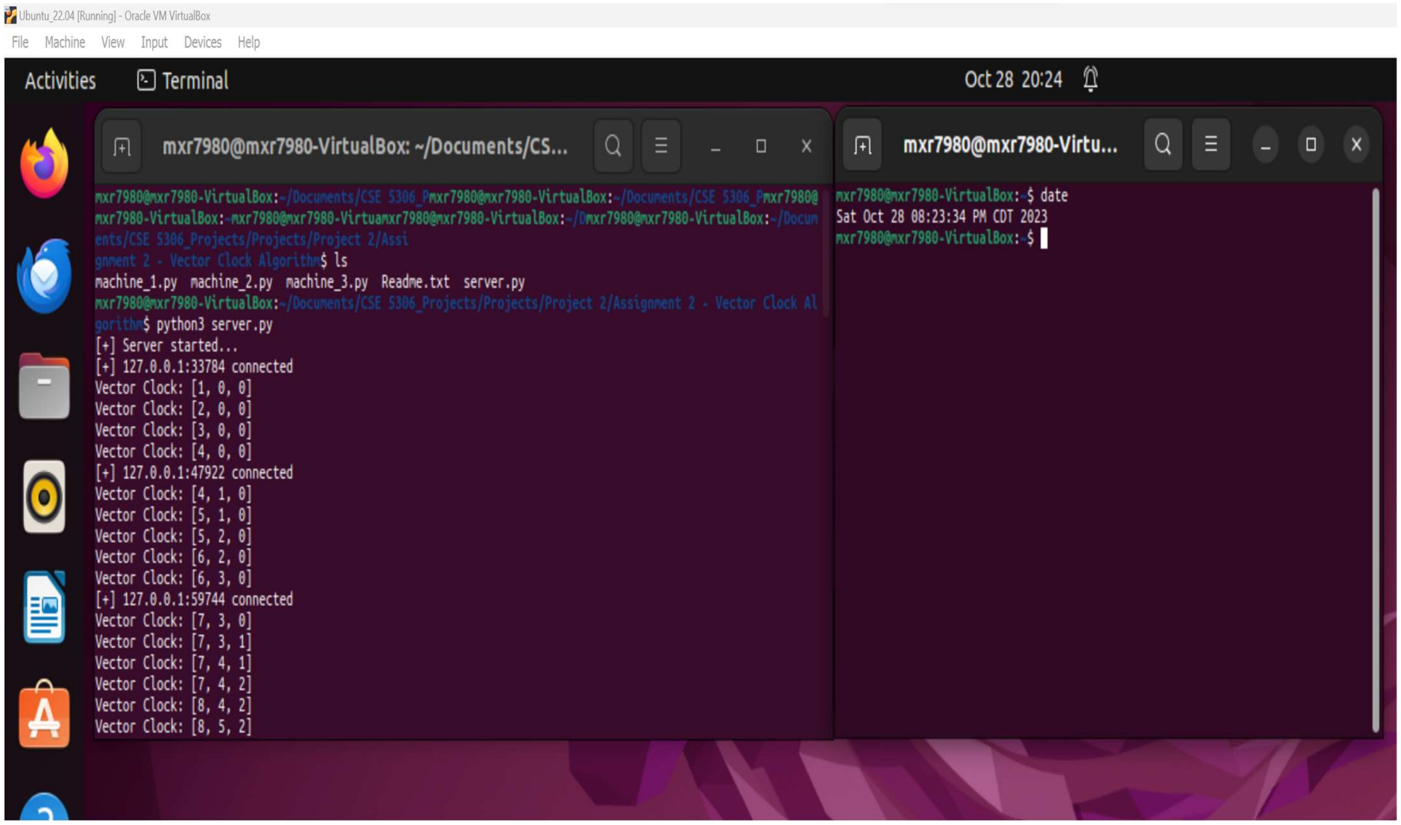
In this assignment, we were tasked with implementing the Vector Clock algorithm to enable causally ordered events in a distributed system. We implemented the algorithm and simulated the output using Python 3.11. The program is implemented in separate files:

- server.py
- machine_1.py
- machine_2.py
- machine_3.py

The three machines/processes that will communicate with each other are the files with the names machine_X.py which will communicate with each other through sockets using the server.py. In each process, we created two threads to communicate with each process. One thread is used to send the messages and the other is used to receive.

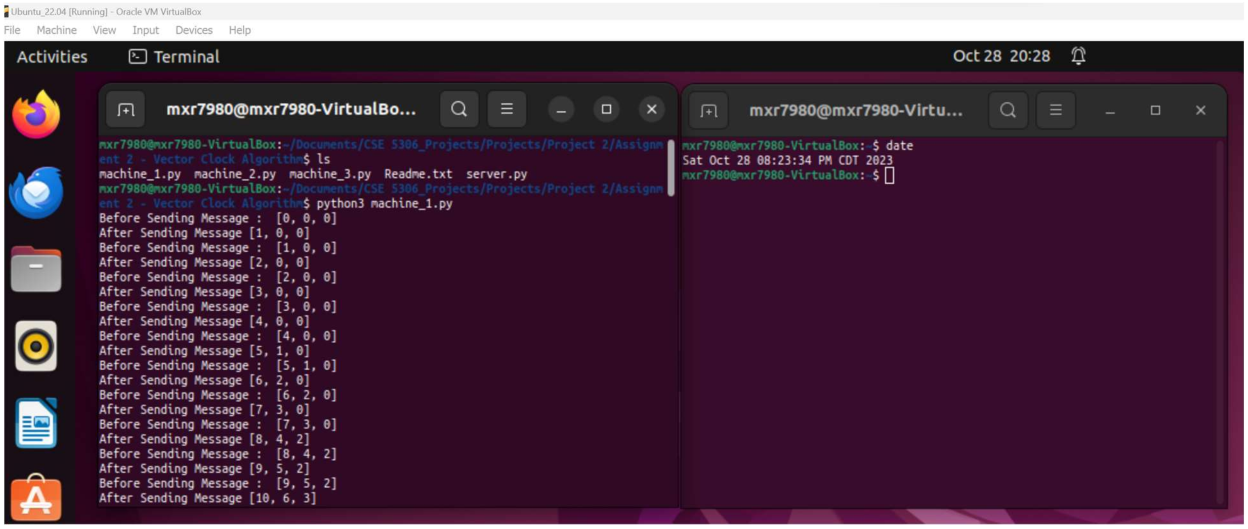
All clocks are initially set to zero, and before each event, the local clock is increased once. A process provides a copy of its own timestamp when it sends a message so that when a process receives the message, it can increment its own logical clock associated with that machine by comparing its version and the value received from the message and keeping the maximum of the two values. This is repeated for all machines.

In our implementation, before sending a message, we show the vector clock of each machine and increment the clock immediately after sending. The updated vector clock is also displayed in the terminal of each of the processes while the server shows the vector clocks from all the machines.


Ubuntu_22.04 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Oct 28 20:24
mxr7980@mxr7980-VirtualBox: ~/Documents/CS...
mxr7980@mxr7980-Virtu...

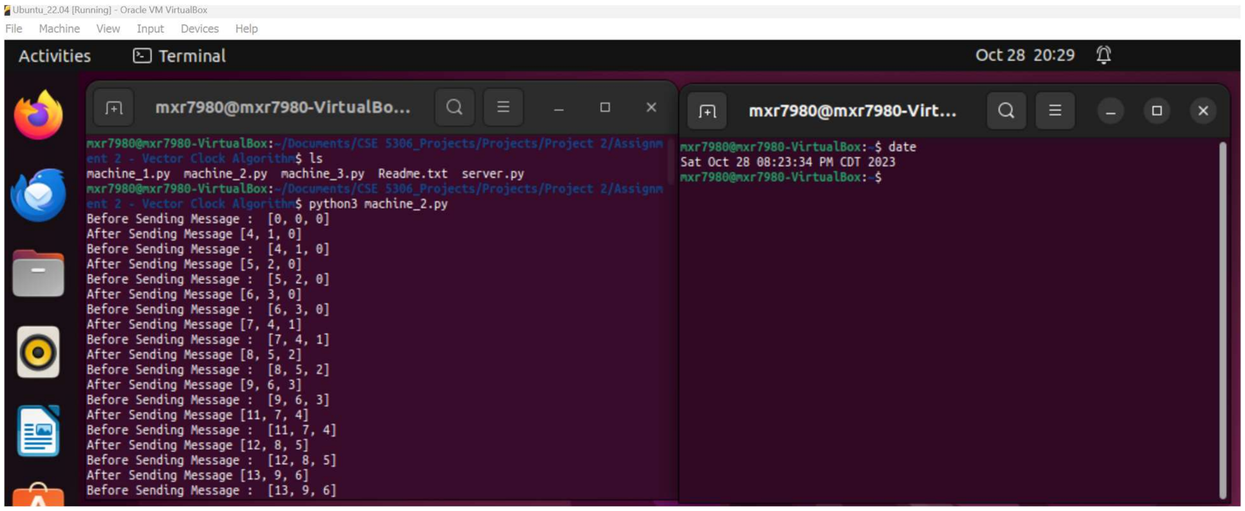
```
mxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Pmxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Pmxr7980@
mxr7980-VirtualBox:~mxr7980@mxr7980-Virtuamxr7980@mxr7980-VirtualBox:~/Dmxr7980@mxr7980-VirtualBox:~/Docum
ents/CSE 5306_Projects/Projects/Project 2/Assi
gnment 2 - Vector Clock Algorithm$ ls
machine_1.py  machine_2.py  machine_3.py  Readme.txt  server.py
mxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Projects/Projects/Project 2/Assignment 2 - Vector Clock Al
gorithm$ python3 server.py
[+] Server started...
[+] 127.0.0.1:33784 connected
Vector Clock: [1, 0, 0]
Vector Clock: [2, 0, 0]
Vector Clock: [3, 0, 0]
Vector Clock: [4, 0, 0]
[+] 127.0.0.1:47922 connected
Vector Clock: [4, 1, 0]
Vector Clock: [5, 1, 0]
Vector Clock: [5, 2, 0]
Vector Clock: [6, 2, 0]
Vector Clock: [6, 3, 0]
[+] 127.0.0.1:59744 connected
Vector Clock: [7, 3, 0]
Vector Clock: [7, 3, 1]
Vector Clock: [7, 4, 1]
Vector Clock: [7, 4, 2]
Vector Clock: [8, 4, 2]
Vector Clock: [8, 5, 2]
```

```
mxr7980@mxr7980-VirtualBox:~$ date
Sat Oct 28 08:23:34 PM CDT 2023
mxr7980@mxr7980-VirtualBox:~$ 
```


Ubuntu_22.04 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Oct 28 20:28
mxr7980@mxr7980-VirtualBo...
mxr7980@mxr7980-Virtu...

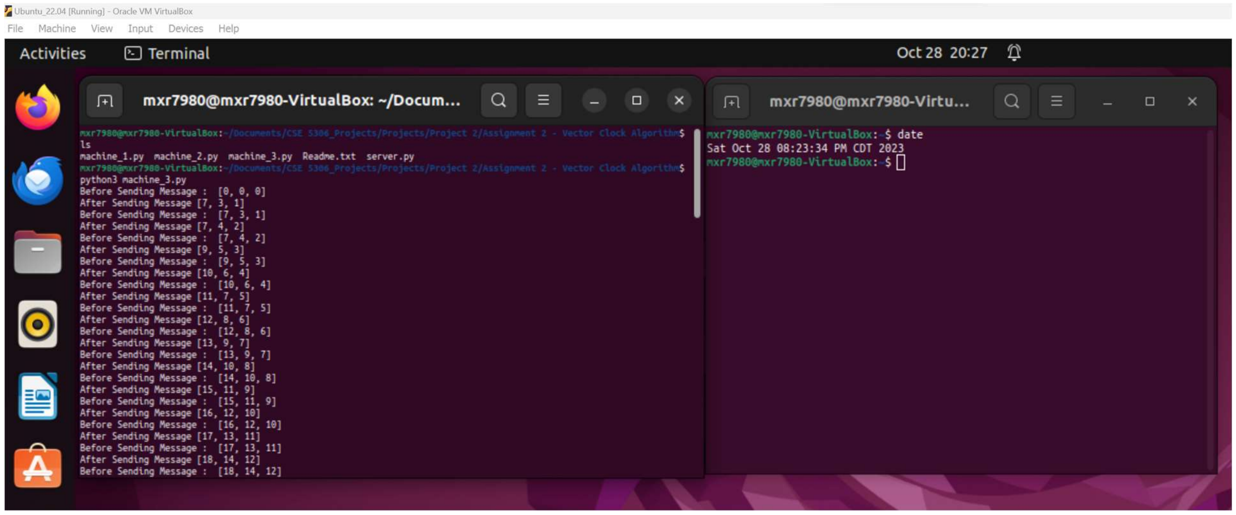
```
mxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Projects/Projects/Project 2/Assignm
ent 2 - Vector Clock Algorithm$ ls
machine_1.py  machine_2.py  machine_3.py  Readme.txt  server.py
mxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Projects/Projects/Project 2/Assignm
ent 2 - Vector Clock Algorithm$ python3 machine_1.py
Before Sending Message :  [0, 0, 0]
After Sending Message [1, 0, 0]
Before Sending Message :  [1, 0, 0]
After Sending Message [2, 0, 0]
Before Sending Message :  [2, 0, 0]
After Sending Message [3, 0, 0]
Before Sending Message :  [3, 0, 0]
After Sending Message [4, 0, 0]
Before Sending Message :  [4, 0, 0]
After Sending Message [5, 1, 0]
Before Sending Message :  [5, 1, 0]
After Sending Message [6, 2, 0]
Before Sending Message :  [6, 2, 0]
After Sending Message [7, 3, 0]
Before Sending Message :  [7, 3, 0]
After Sending Message [8, 4, 2]
Before Sending Message :  [8, 4, 2]
After Sending Message [9, 5, 2]
Before Sending Message :  [9, 5, 2]
After Sending Message [10, 6, 3]
```

```
mxr7980@mxr7980-VirtualBox:~$ date
Sat Oct 28 08:23:34 PM CDT 2023
mxr7980@mxr7980-VirtualBox:~$
```


Ubuntu_22.04 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Oct 28 20:29
mxr7980@mxr7980-VirtualBo...
mxr7980@mxr7980-Virt...

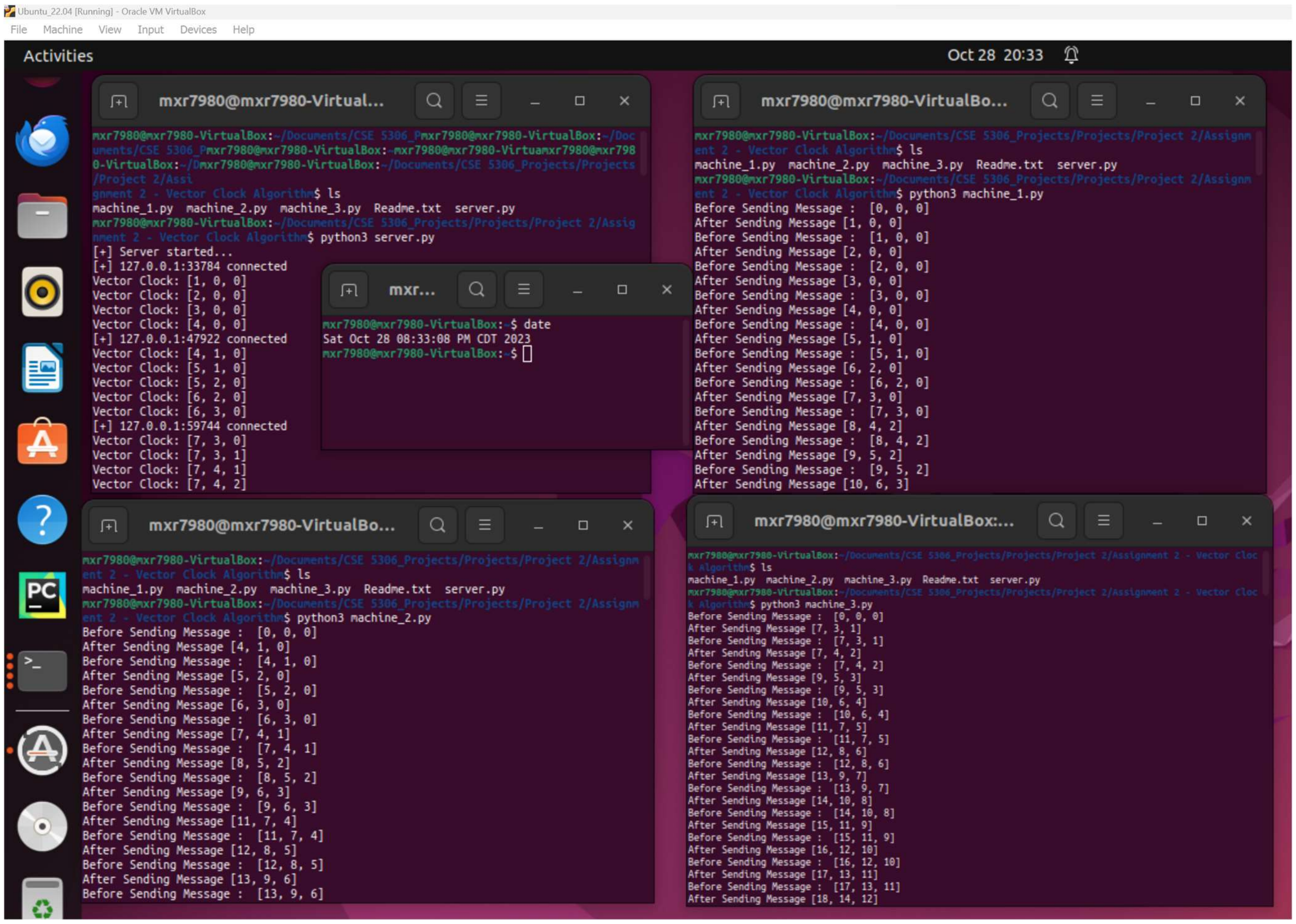
```
mxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Projects/Projects/Project 2/Assignm
ent 2 - Vector Clock Algorithm$ ls
machine_1.py  machine_2.py  machine_3.py  Readme.txt  server.py
mxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Projects/Projects/Project 2/Assignm
ent 2 - Vector Clock Algorithm$ python3 machine_2.py
Before Sending Message :  [0, 0, 0]
After Sending Message [4, 1, 0]
Before Sending Message :  [4, 1, 0]
After Sending Message [5, 2, 0]
Before Sending Message :  [5, 2, 0]
After Sending Message [6, 3, 0]
Before Sending Message :  [6, 3, 0]
After Sending Message [7, 4, 1]
Before Sending Message :  [7, 4, 1]
After Sending Message [8, 5, 2]
Before Sending Message :  [8, 5, 2]
After Sending Message [9, 6, 3]
Before Sending Message :  [9, 6, 3]
After Sending Message [11, 7, 4]
Before Sending Message :  [11, 7, 4]
After Sending Message [12, 8, 5]
Before Sending Message :  [12, 8, 5]
After Sending Message [13, 9, 6]
Before Sending Message :  [13, 9, 6]
```

```
mxr7980@mxr7980-VirtualBox:~$ date
Sat Oct 28 08:23:34 PM CDT 2023
mxr7980@mxr7980-VirtualBox:~$
```


Ubuntu_22.04 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Oct 28 20:27
mxr7980@mxr7980-VirtualBox: ~/Docum...
mxr7980@mxr7980-Virtu...

```
mxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Projects/Projects/Project 2/Assignment 2 - Vector Clock Algorithm$
ls
machine_1.py  machine_2.py  machine_3.py  Readme.txt  server.py
mxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Projects/Projects/Project 2/Assignment 2 - Vector Clock Algorithm$
python3 machine_3.py
Before Sending Message :  [0, 0, 0]
After Sending Message [7, 3, 1]
Before Sending Message :  [7, 3, 1]
After Sending Message [7, 4, 2]
Before Sending Message :  [7, 4, 2]
After Sending Message [9, 5, 3]
Before Sending Message :  [9, 5, 3]
After Sending Message [10, 6, 4]
Before Sending Message :  [10, 6, 4]
After Sending Message [11, 7, 5]
Before Sending Message :  [11, 7, 5]
After Sending Message [12, 8, 6]
Before Sending Message :  [12, 8, 6]
After Sending Message [13, 9, 7]
Before Sending Message :  [13, 9, 7]
After Sending Message [14, 10, 8]
Before Sending Message :  [14, 10, 8]
After Sending Message [15, 11, 9]
Before Sending Message :  [15, 11, 9]
After Sending Message [16, 12, 10]
Before Sending Message :  [16, 12, 10]
After Sending Message [17, 13, 11]
Before Sending Message :  [17, 13, 11]
After Sending Message [18, 14, 12]
Before Sending Message :  [18, 14, 12]
```

```
mxr7980@mxr7980-VirtualBox:~$ date
Sat Oct 28 08:23:34 PM CDT 2023
mxr7980@mxr7980-VirtualBox:~$
```



```
mxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Pmxr7980@mxr7980-VirtualBox:~/Doc
uments/CSE 5306_Pmxr7980@mxr7980-VirtualBox:~mxr7980@mxr7980-Virtuamxr7980@mxr798
0-VirtualBox:~/Dmxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Projects/Projects
/Project 2/Assi
gnment 2 - Vector Clock Algorithm$ ls
machine_1.py  machine_2.py  machine_3.py  Readme.txt  server.py
mxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Projects/Projects/Project 2/Assig
nment 2 - Vector Clock Algorithm$ python3 server.py
[+] Server started...
[+] 127.0.0.1:33784 connected
Vector Clock: [1, 0, 0]
Vector Clock: [2, 0, 0]
Vector Clock: [3, 0, 0]
Vector Clock: [4, 0, 0]
[+] 127.0.0.1:47922 connected
Vector Clock: [4, 1, 0]
Vector Clock: [5, 1, 0]
Vector Clock: [5, 2, 0]
Vector Clock: [6, 2, 0]
Vector Clock: [6, 3, 0]
[+] 127.0.0.1:59744 connected
Vector Clock: [7, 3, 0]
Vector Clock: [7, 3, 1]
Vector Clock: [7, 4, 1]
Vector Clock: [7, 4, 2]
```

```
mxr7980@mxr7980-VirtualBox:~$ date
Sat Oct 28 08:33:08 PM CDT 2023
mxr7980@mxr7980-VirtualBox:~$ 
```

```
mxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Projects/Projects/Project 2/Assignm
ent 2 - Vector Clock Algorithm$ ls
machine_1.py  machine_2.py  machine_3.py  Readme.txt  server.py
mxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Projects/Projects/Project 2/Assignm
ent 2 - Vector Clock Algorithm$ python3 machine_1.py
Before Sending Message :  [0, 0, 0]
After Sending Message [1, 0, 0]
Before Sending Message :  [1, 0, 0]
After Sending Message [2, 0, 0]
Before Sending Message :  [2, 0, 0]
After Sending Message [3, 0, 0]
Before Sending Message :  [3, 0, 0]
After Sending Message [4, 0, 0]
Before Sending Message :  [4, 0, 0]
After Sending Message [5, 1, 0]
Before Sending Message :  [5, 1, 0]
After Sending Message [6, 2, 0]
Before Sending Message :  [6, 2, 0]
After Sending Message [7, 3, 0]
Before Sending Message :  [7, 3, 0]
After Sending Message [8, 4, 2]
Before Sending Message :  [8, 4, 2]
After Sending Message [9, 5, 2]
Before Sending Message :  [9, 5, 2]
After Sending Message [10, 6, 3]
```

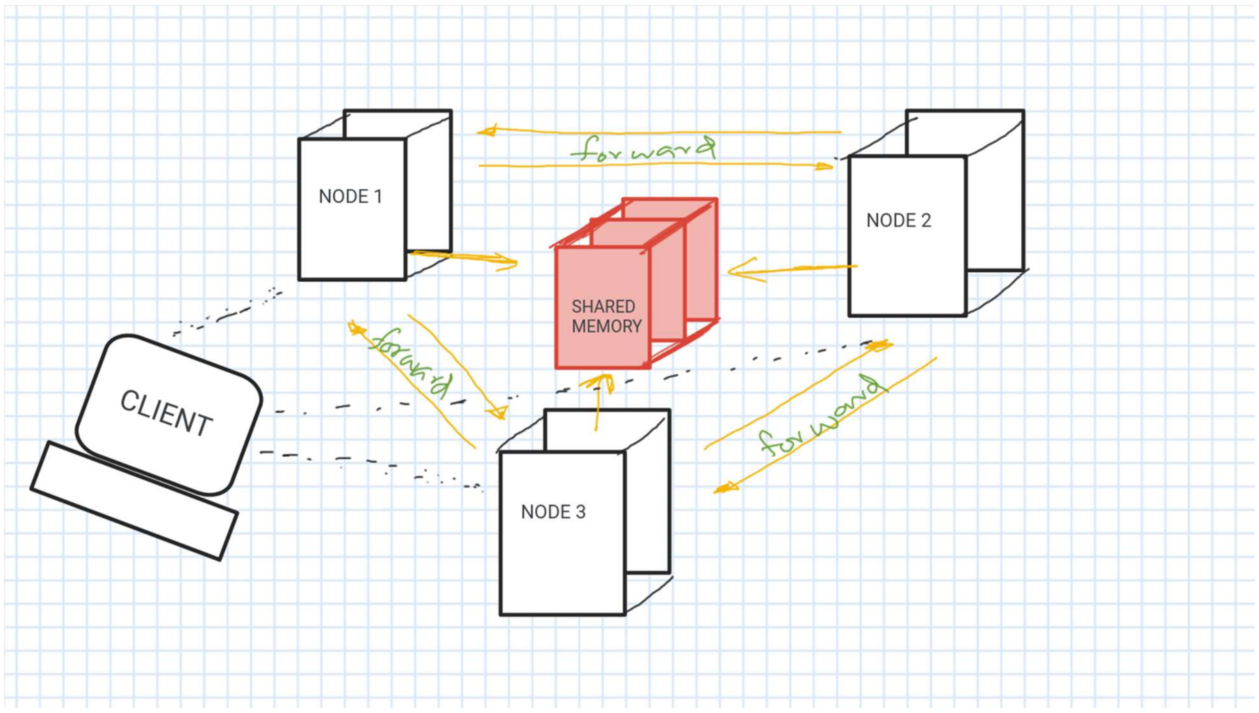
```
mxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Projects/Projects/Project 2/Assignm
ent 2 - Vector Clock Algorithm$ ls
machine_1.py  machine_2.py  machine_3.py  Readme.txt  server.py
mxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Projects/Projects/Project 2/Assignm
ent 2 - Vector Clock Algorithm$ python3 machine_2.py
Before Sending Message :  [0, 0, 0]
After Sending Message [4, 1, 0]
Before Sending Message :  [4, 1, 0]
After Sending Message [5, 2, 0]
Before Sending Message :  [5, 2, 0]
After Sending Message [6, 3, 0]
Before Sending Message :  [6, 3, 0]
After Sending Message [7, 4, 1]
Before Sending Message :  [7, 4, 1]
After Sending Message [8, 5, 2]
Before Sending Message :  [8, 5, 2]
After Sending Message [9, 6, 3]
Before Sending Message :  [9, 6, 3]
After Sending Message [11, 7, 4]
Before Sending Message :  [11, 7, 4]
After Sending Message [12, 8, 5]
Before Sending Message :  [12, 8, 5]
After Sending Message [13, 9, 6]
Before Sending Message :  [13, 9, 6]
```

```
mxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Projects/Projects/Project 2/Assignment 2 - Vector Cloc
k Algorithm$ ls
machine_1.py  machine_2.py  machine_3.py  Readme.txt  server.py
mxr7980@mxr7980-VirtualBox:~/Documents/CSE 5306_Projects/Projects/Project 2/Assignment 2 - Vector Cloc
k Algorithm$ python3 machine_3.py
Before Sending Message :  [0, 0, 0]
After Sending Message [7, 3, 1]
Before Sending Message :  [7, 3, 1]
After Sending Message [7, 4, 2]
Before Sending Message :  [7, 4, 2]
After Sending Message [9, 5, 3]
Before Sending Message :  [9, 5, 3]
After Sending Message [10, 6, 4]
Before Sending Message :  [10, 6, 4]
After Sending Message [11, 7, 5]
Before Sending Message :  [11, 7, 5]
After Sending Message [12, 8, 6]
Before Sending Message :  [12, 8, 6]
After Sending Message [13, 9, 7]
Before Sending Message :  [13, 9, 7]
After Sending Message [14, 10, 8]
Before Sending Message :  [14, 10, 8]
After Sending Message [15, 11, 9]
Before Sending Message :  [15, 11, 9]
After Sending Message [16, 12, 10]
Before Sending Message :  [16, 12, 10]
After Sending Message [17, 13, 11]
Before Sending Message :  [17, 13, 11]
After Sending Message [18, 14, 12]
```

- Vector clock on M1 is on the 1$^{st}$ index [M1, M2, M3]
- When messages from M1 is sent, its counter is increased [1, 0, 0]
- When messages from M2 is sent, its counter is increased [0, 1, 0]
- When messages from M3 is sent, its counter is increased [0, 0, 1]
- When a machine receives a message, it compares its index for that machine and updates its value by keeping the maximum of the two values, for example: M2's vector clock becomes [1, 1, 0] when it receives the message from M1.
- This pattern can be seen as more messages are sent, but each time the vector clocks are always in order.

# Assignment 3

## Distributed Locking Algorithm



A 3-node distributed system used to lock files saved in shared memory of all the nodes in the distributed system. When a node acquires a lock to a file, the counter is simply updated in the file and then the file gets closed. Using the principles of the token-ring algorithm, we have developed a mechanism to forward the transaction between the nodes in our 3-node distributed system. When a client connects a node to initiate a transaction, the node self-diagnoses its status on whether to process the transaction or to forward the transaction. From the above diagram, we see that the nodes are aligned in circular network topology. When the transaction reaches the node after getting passed over all the other nodes in its network to transact and not get transacted, it checks the transaction and redo the passing, until the transaction is finished. Clients can connect to any node to pass the transaction, but acquiring the lock is automatically managed by the nodes in the distributed system. The system is easily scalable and good to use for the simulation of a locking scheme mechanism. The files are stored in shared memory, which, in our case, is a common directory in a single machine emulating multiple processes as nodes of the distributed system.

During development, I have come across multiple ways of implementing the lock counter handling using configuration files and key-value maps at each node with continuous synchronization and others. Finally, I have concluded and implemented the counter value in the file itself. There are other useful keys similar to the counter-key in each file, such as the status key of the file, whether locked or unlocked, and the name-key of the file for better handling using nodes in our 3-node distributed system.

## Conclusion

This exercise has explored the basic ideas underlying Total Order Multicasting, Vector Clocks, and Distributed Locking. These algorithms are essential for the development of resilient, dependable, and optimized distributed systems. Total Order Multicasting ensures that all processes in a distributed system receive messages in the same order, regardless of the order in which the messages were sent. Lamport's Algorithm is a simple and efficient way to implement Total Order Multicasting. Vector Clocks provide a way to timestamp events in a distributed system in a way that is consistent across all processes. This is useful for tracking the causal relationships between events, and for detecting and resolving inconsistencies in distributed data. Distributed Locking algorithms allow processes in a distributed system to acquire and release locks on shared resources. This is important for preventing concurrent access to shared resources, which can lead to data corruption and other problems.