CSE 5306-004 DISTRIBUTED SYSTEM
Project 2

Team Members
Mehul Ganjude (1001990551)
Chaithra Kallumutlu Kushalappa (1001988073)

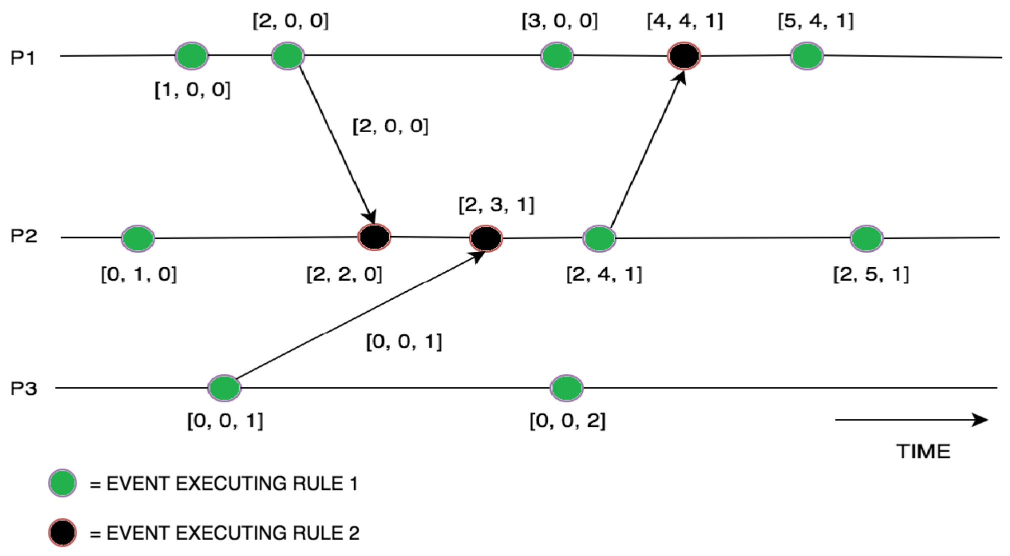
We have neither given nor received unauthorized assistance on this work.
Signed: Mehul Deorao Ganjude, Chaithra Kallumutlu Kushalappa
Date: 04/10/2022

Develop an n-node distributed system that implements vector clock. The distributed system uses a logical clock to timestamp messages sent/received between nodes. At first, each node should synchronize their logical clocks to the same initial value, based on which the ordering of events can be determined among the machines. Use the Berkeley Algorithm to obtain an initial agreement on the logical clocks.

Vector Clock

A vector clock is an algorithm for generating a partial ordering of events in a distributed system and detecting causality violations. Just as in a Lamport timestamps, inter process messages contain the state of the sending process's logical clock. A vector clock of a system of "N" processes is an array/vector of N logical clocks, one clock per process; a local "smallest possible values" copy of the global clock-array is kept in each process.

Example:
Consider a process (P) with a vector size N for each process: the above set of rules mentioned are to be executed by the vector clock:



The above example depicts the vector clocks mechanism in which the vector clocks are updated after execution of internal events, the arrows indicate how the values of vectors are sent in between the processes (P1, P2, P3).

To sum up, Vector clocks algorithms are used in distributed systems to provide a causally consistent ordering of events but the entire Vector is sent to each process for every message sent, in order to keep the vector clocks in sync.

What is clock synchronization algorithm?

Clock synchronization is a method of synchronizing clock values of any two nodes in a distributed system with the use of external reference clock or internal clock value of the node. During the synchronization, many factors effect on a network.

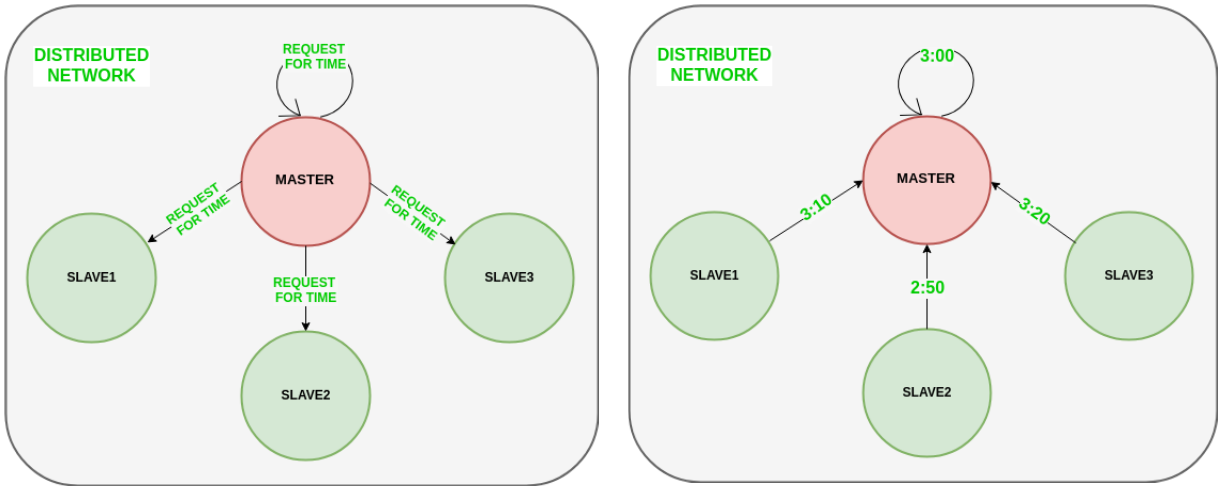
Berkeley's Algorithm

Berkeley's Algorithm is a clock synchronization technique used in distributed systems. The algorithm assumes that each machine node in the network either doesn't have an accurate time source or doesn't possess a UTC server.

Algorithm

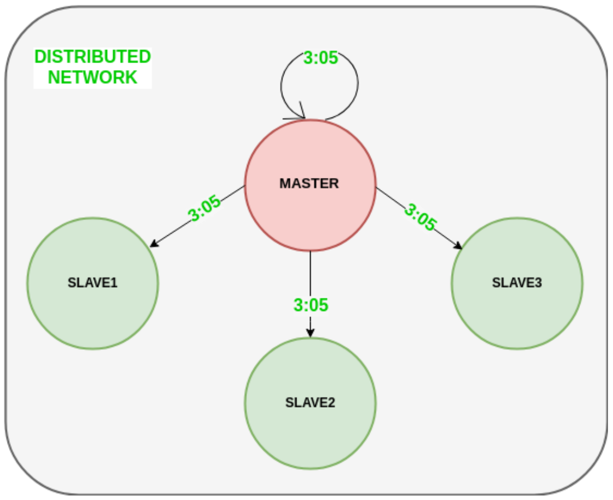
1. An individual node is chosen as the master node from a pool node in the network. This node is the main node in the network which acts as a master and the rest of the nodes act as slaves. The master node is chosen using an election process/leader election algorithm.
2. Master node periodically pings slaves nodes and fetches clock time.

The diagram on the left illustrates how the master sends requests to slave nodes and the diagram on the right illustrates how slave nodes send back time given by their system clock.



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

The diagram below illustrates the last step of Berkeley's algorithm.



We implemented the program by using java programming language. By initializing an empty dictionary having 3 processes and the IDE we used to be Microsoft visual studio code (or any IDE as Eclipse) with java language.

Steps to Execute the Program

1. Go to folder where file is stored.

2. Once you are in the same directory compile the Program,
   - ➢ javac -cp . project2/CoordinatorProcess.java

3. Once the compilation has been done successfully run the program,
   - ➢ java project2.CoordinatorProcess

4. Now you must enter the number of additional processes.
   Note: - It should be minimum 2.

5. Once you enter the number the coordinator process will wait for the other processes.

6. Open the terminal depending on the number of processes.

7. Again, go to the same folder where all files are present.

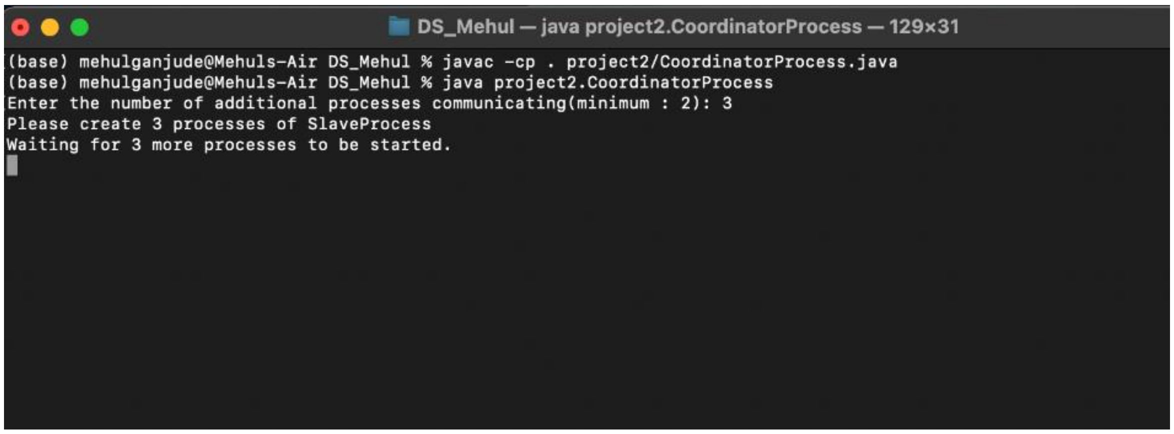
8. Compile the program
   - ➢ javac -cp SlaveProcess.java

9. Run java SlaveProcess

10. Check the desired output.

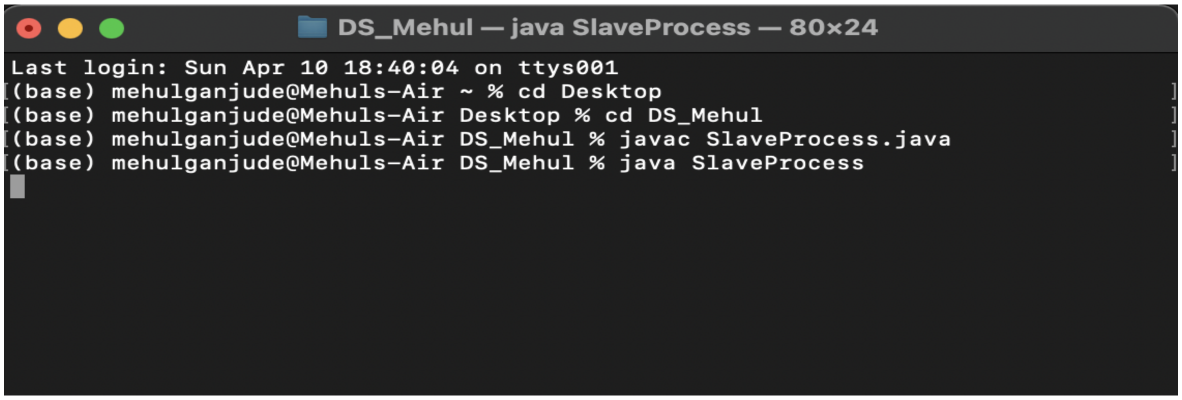
Screenshots

Coordinator Process
No of Communication=3
It will wait for other 3 process and will wait until 3 processes started.



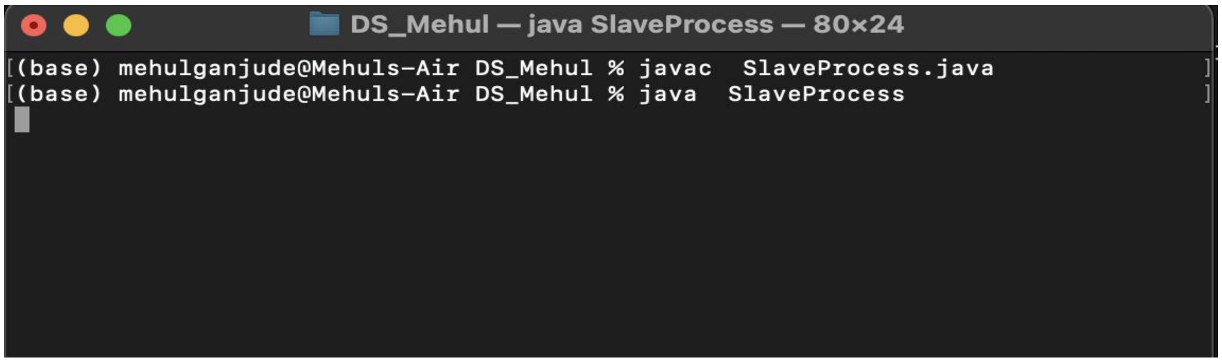
```
(base) mehulganjude@Mehuls-Air DS_Mehul % javac -cp . project2/CoordinatorProcess.java
(base) mehulganjude@Mehuls-Air DS_Mehul % java project2.CoordinatorProcess
Enter the number of additional processes communicating(minimum : 2): 3
Please create 3 processes of SlaveProcess
Waiting for 3 more processes to be started.
```

Process 1 started:



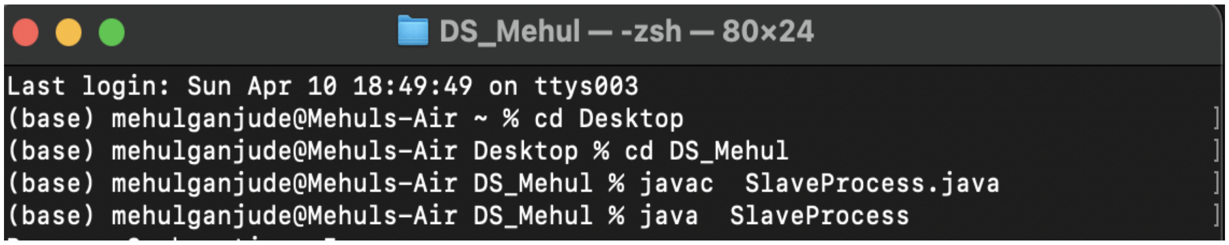
```
Last login: Sun Apr 10 18:40:04 on ttys001
(base) mehulganjude@Mehuls-Air ~ % cd Desktop
(base) mehulganjude@Mehuls-Air Desktop % cd DS_Mehul
(base) mehulganjude@Mehuls-Air DS_Mehul % javac SlaveProcess.java
(base) mehulganjude@Mehuls-Air DS_Mehul % java SlaveProcess
```

Process 2 Started:



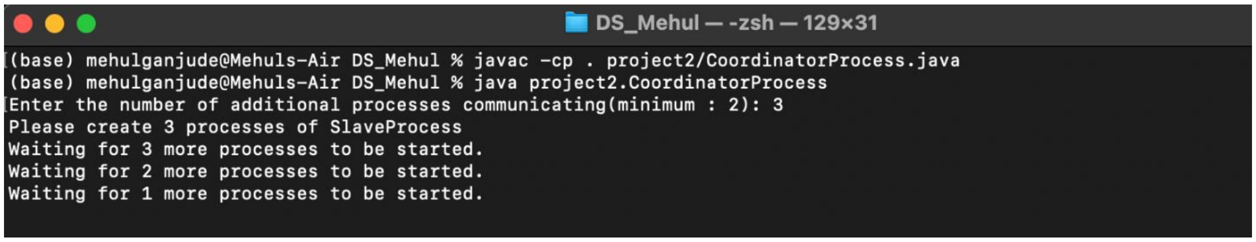
```
(base) mehulganjude@Mehuls-Air DS_Mehul % javac  SlaveProcess.java
(base) mehulganjude@Mehuls-Air DS_Mehul % java  SlaveProcess
```

Process 3 Started:



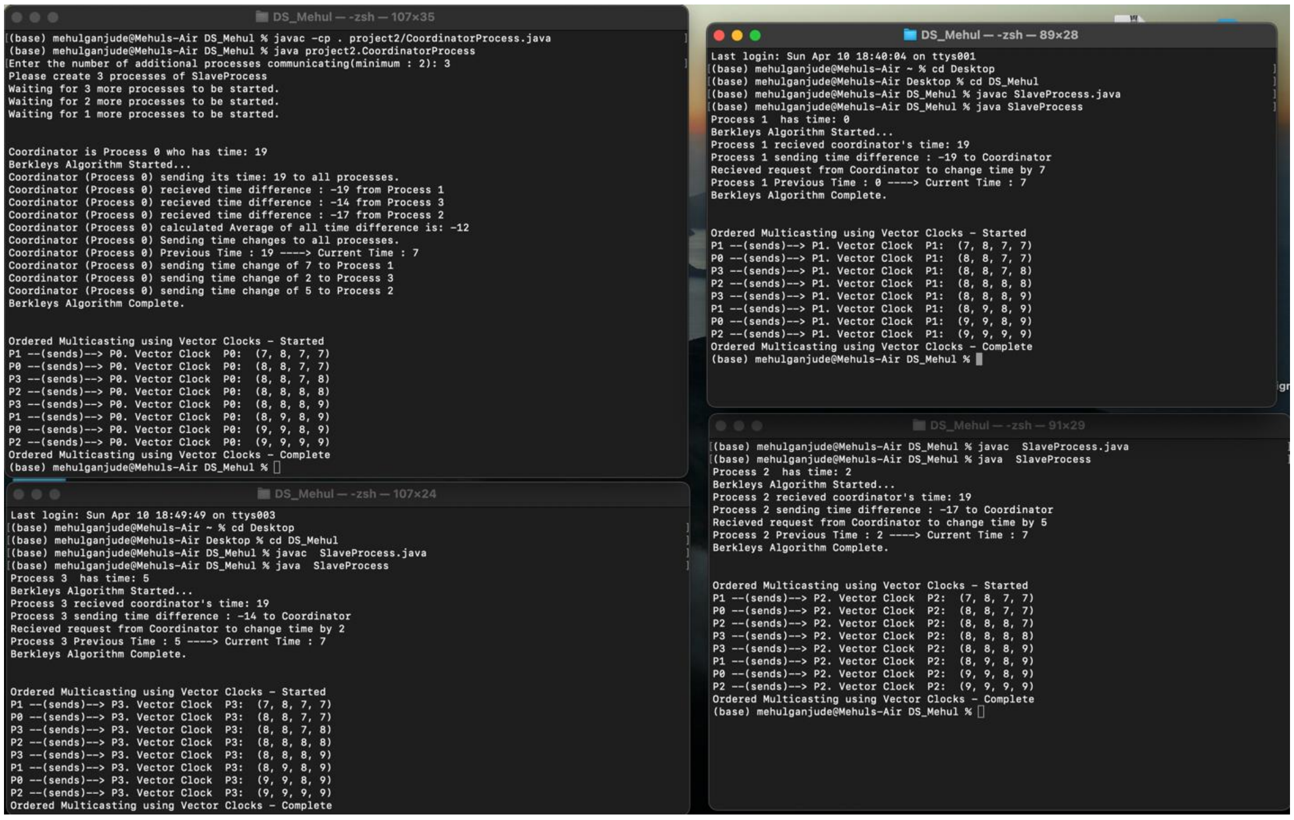
```
Last login: Sun Apr 10 18:49:49 on ttys003
(base) mehulganjude@Mehuls-Air ~ % cd Desktop
(base) mehulganjude@Mehuls-Air Desktop % cd DS_Mehul
(base) mehulganjude@Mehuls-Air DS_Mehul % javac  SlaveProcess.java
(base) mehulganjude@Mehuls-Air DS_Mehul % java  SlaveProcess
```

Coordinator process when all three processes has connected.



```
(base) mehulganjude@Mehuls-Air DS_Mehul % javac -cp . project2/CoordinatorProcess.java
(base) mehulganjude@Mehuls-Air DS_Mehul % java project2.CoordinatorProcess
Enter the number of additional processes communicating(minimum : 2): 3
Please create 3 processes of SlaveProcess
Waiting for 3 more processes to be started.
Waiting for 2 more processes to be started.
Waiting for 1 more processes to be started.
```

Final output when all the clock synchronization and send its time.



```
(base) mehulganjude@Mehuls-Air DS_Mehul % javac -cp . project2/CoordinatorProcess.java
(base) mehulganjude@Mehuls-Air DS_Mehul % java project2.CoordinatorProcess
Enter the number of additional processes communicating(minimum : 2): 3
Please create 3 processes of SlaveProcess
Waiting for 3 more processes to be started.
Waiting for 2 more processes to be started.
Waiting for 1 more processes to be started.


Coordinator is Process 0 who has time: 19
Berkleys Algorithm Started...
Coordinator (Process 0) sending its time: 19 to all processes.
Coordinator (Process 0) recieved time difference : -19 from Process 1
Coordinator (Process 0) recieved time difference : -14 from Process 3
Coordinator (Process 0) recieved time difference : -17 from Process 2
Coordinator (Process 0) calculated Average of all time difference is: -12
Coordinator (Process 0) Sending time changes to all processes.
Coordinator (Process 0) Previous Time : 19 ----> Current Time : 7
Coordinator (Process 0) sending time change of 7 to Process 1
Coordinator (Process 0) sending time change of 2 to Process 3
Coordinator (Process 0) sending time change of 5 to Process 2
Berkleys Algorithm Complete.


Ordered Multicasting using Vector Clocks - Started
P1 --(sends)--> P0. Vector Clock  P0:  (7, 8, 7, 7)
P0 --(sends)--> P0. Vector Clock  P0:  (8, 8, 7, 7)
P3 --(sends)--> P0. Vector Clock  P0:  (8, 8, 7, 8)
P2 --(sends)--> P0. Vector Clock  P0:  (8, 8, 8, 8)
P3 --(sends)--> P0. Vector Clock  P0:  (8, 8, 8, 9)
P1 --(sends)--> P0. Vector Clock  P0:  (8, 9, 8, 9)
P0 --(sends)--> P0. Vector Clock  P0:  (9, 9, 8, 9)
P2 --(sends)--> P0. Vector Clock  P0:  (9, 9, 9, 9)
Ordered Multicasting using Vector Clocks - Complete
(base) mehulganjude@Mehuls-Air DS_Mehul % 
```

```
Last login: Sun Apr 10 18:49:49 on ttys003
(base) mehulganjude@Mehuls-Air ~ % cd Desktop
(base) mehulganjude@Mehuls-Air Desktop % cd DS_Mehul
(base) mehulganjude@Mehuls-Air DS_Mehul % javac  SlaveProcess.java
(base) mehulganjude@Mehuls-Air DS_Mehul % java  SlaveProcess
Process 3  has time: 5
Berkleys Algorithm Started...
Process 3 recieved coordinator's time: 19
Process 3 sending time difference : -14 to Coordinator
Recieved request from Coordinator to change time by 2
Process 3 Previous Time : 5 ----> Current Time : 7
Berkleys Algorithm Complete.


Ordered Multicasting using Vector Clocks - Started
P1 --(sends)--> P3. Vector Clock  P3:  (7, 8, 7, 7)
P0 --(sends)--> P3. Vector Clock  P3:  (8, 8, 7, 7)
P3 --(sends)--> P3. Vector Clock  P3:  (8, 8, 7, 8)
P2 --(sends)--> P3. Vector Clock  P3:  (8, 8, 8, 8)
P3 --(sends)--> P3. Vector Clock  P3:  (8, 8, 8, 9)
P1 --(sends)--> P3. Vector Clock  P3:  (8, 9, 8, 9)
P0 --(sends)--> P3. Vector Clock  P3:  (9, 9, 8, 9)
P2 --(sends)--> P3. Vector Clock  P3:  (9, 9, 9, 9)
Ordered Multicasting using Vector Clocks - Complete
```

```
Last login: Sun Apr 10 18:40:04 on ttys001
(base) mehulganjude@Mehuls-Air ~ % cd Desktop
(base) mehulganjude@Mehuls-Air Desktop % cd DS_Mehul
(base) mehulganjude@Mehuls-Air DS_Mehul % javac SlaveProcess.java
(base) mehulganjude@Mehuls-Air DS_Mehul % java SlaveProcess
Process 1  has time: 0
Berkleys Algorithm Started...
Process 1 recieved coordinator's time: 19
Process 1 sending time difference : -19 to Coordinator
Recieved request from Coordinator to change time by 7
Process 1 Previous Time : 0 ----> Current Time : 7
Berkleys Algorithm Complete.


Ordered Multicasting using Vector Clocks - Started
P1 --(sends)--> P1. Vector Clock  P1:  (7, 8, 7, 7)
P0 --(sends)--> P1. Vector Clock  P1:  (8, 8, 7, 7)
P3 --(sends)--> P1. Vector Clock  P1:  (8, 8, 7, 8)
P2 --(sends)--> P1. Vector Clock  P1:  (8, 8, 8, 8)
P3 --(sends)--> P1. Vector Clock  P1:  (8, 8, 8, 9)
P1 --(sends)--> P1. Vector Clock  P1:  (8, 9, 8, 9)
P0 --(sends)--> P1. Vector Clock  P1:  (9, 9, 8, 9)
P2 --(sends)--> P1. Vector Clock  P1:  (9, 9, 9, 9)
Ordered Multicasting using Vector Clocks - Complete
(base) mehulganjude@Mehuls-Air DS_Mehul % 
```

```
(base) mehulganjude@Mehuls-Air DS_Mehul % javac  SlaveProcess.java
(base) mehulganjude@Mehuls-Air DS_Mehul % java  SlaveProcess
Process 2  has time: 2
Berkleys Algorithm Started...
Process 2 recieved coordinator's time: 19
Process 2 sending time difference : -17 to Coordinator
Recieved request from Coordinator to change time by 5
Process 2 Previous Time : 2 ----> Current Time : 7
Berkleys Algorithm Complete.


Ordered Multicasting using Vector Clocks - Started
P1 --(sends)--> P2. Vector Clock  P2:  (7, 8, 7, 7)
P0 --(sends)--> P2. Vector Clock  P2:  (8, 8, 7, 7)
P2 --(sends)--> P2. Vector Clock  P2:  (8, 8, 8, 7)
P3 --(sends)--> P2. Vector Clock  P2:  (8, 8, 8, 8)
P3 --(sends)--> P2. Vector Clock  P2:  (8, 8, 8, 9)
P1 --(sends)--> P2. Vector Clock  P2:  (8, 9, 8, 9)
P0 --(sends)--> P2. Vector Clock  P2:  (9, 9, 8, 9)
P2 --(sends)--> P2. Vector Clock  P2:  (9, 9, 9, 9)
Ordered Multicasting using Vector Clocks - Complete
(base) mehulganjude@Mehuls-Air DS_Mehul % 
```

Things learned:

- We have learned the working procedures of the vector clock algorithm.
- Performing the steps to identify when to deliver a message was a bit challenging in this assignment there were cases of events being missed due to starting the event sender thread before starting the event receiver thread.
- This issue was circumvented by starting 2 the event receiver thread before the event sender thread and putting a certain amount of delay between them.
- The most important thing we learned when doing this project is how to implement CPU scheduling and time complexity algorithms.
- How Berkeley's algorithm works, for clock synchronization technique used in distributed system.

Issues Encountered while Implementing the Project:

- The issue we encountered is with implementing the integers for the project and in the CPU, scheduling processes also with changing the vectors for later events in the program.
- Also printing the before and update time of the vector when it coordinates processes sends the command.
- How the synchronization between the vectors can be done by sending the message to each process and give time differences by each process.

References:

https://www.geeksforgeeks.org/berkeleys-algorithm/

https://www.geeksforgeeks.org/vector-clocks-in-distributed-systems/

http://mixu.net/vectorclock/

https://mittaltutorials.wordpress.com/2016/01/03/exp2-vector-clock/

https://github.com/jaymit123/MultiProcess-Communication

https://stackoverflow.com/questions/3061927/implementation-of-vector-clocks

https://github.com/DayuanTan/berkeley-algorithm-implementation

https://github.com/gsharma/vector-clock

https://www.sidmartinbio.org/what-is-berkeley-clock-synchronization-algorithm/