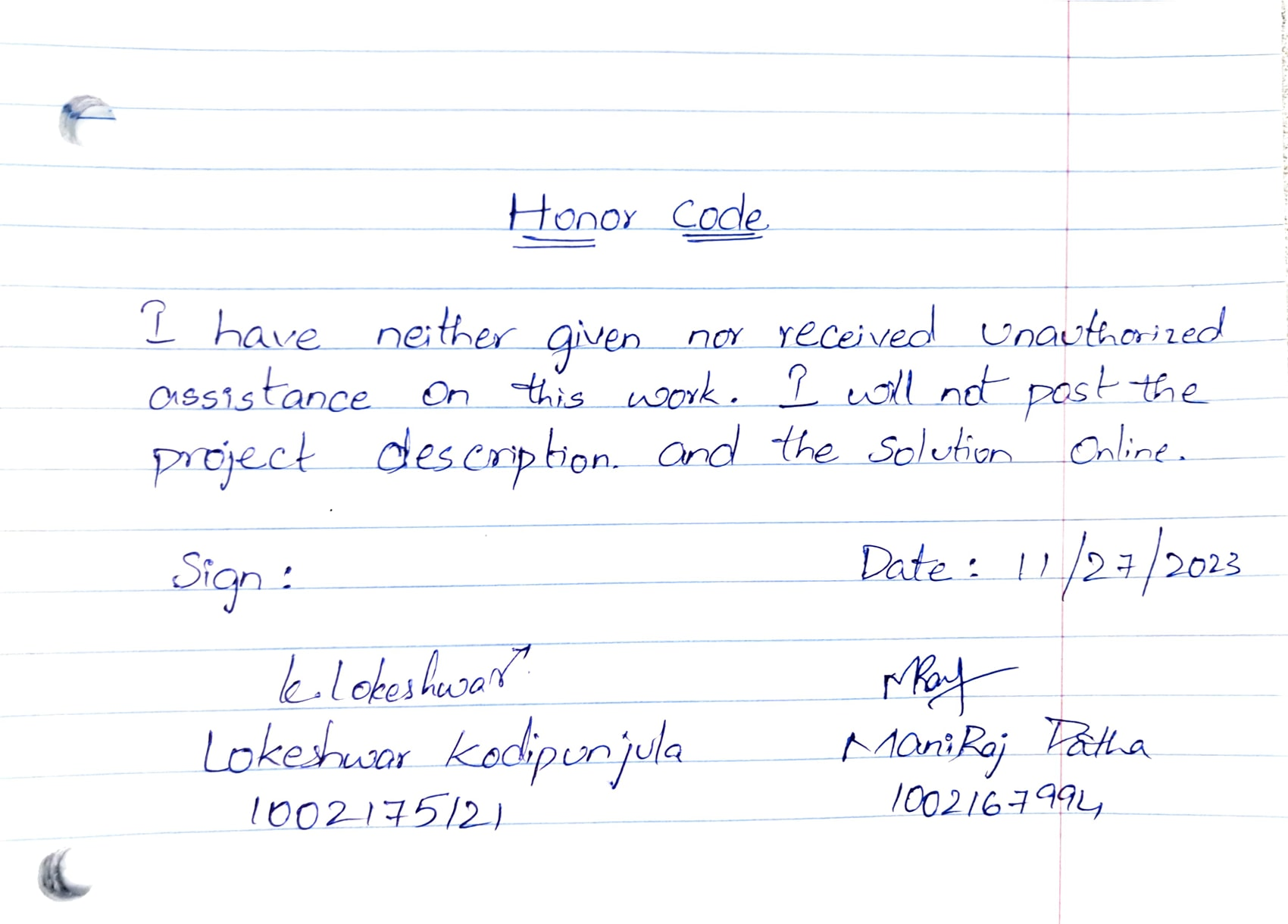
**CSE 5306 Project Assignment 2**

**Honor Code:**



**Description:**

This project implements a logical timestamp-based message multicasting system, where a group of processes communicate by multicasting messages to each other. Each message is timestamped with the sender's logical time, and the system ensures that messages are delivered in the same order across all processes. The key steps involve message multicasting, local queue management, acknowledgment handling, and application message delivery.

**Implementation of Program:**

This Java program implements a logical timestamp-based message multicasting system, simulating a group of processes that exchange messages. The system ensures that messages are consistently delivered in the same order across all processes. The core components include the Process, Message, Acknowledgment, Communication Thread, Delivery Thread, and the main entry point, Main class.

**Execution Flow:**

* The program starts by initializing processes, communication threads, and delivery threads based on the specified number (NUM\_OF\_PROCESSES).
* Communication threads periodically multicast messages to simulate message exchanges between processes.
* Processes receive messages, enqueue them in a priority queue, and multicast acknowledgments for the head of their message queue.
* Communication threads receive acknowledgments and update acknowledgment sets in processes.
* Delivery threads periodically deliver messages to the application when the necessary conditions are met.
* The simulation continues until interrupted, ensuring a controlled termination.

import java.util.HashSet;

import java.util.PriorityQueue;

import java.util.Set;

class Process {

    private int processId;

    private PriorityQueue<Message> queue;

    private Set<Acknowledgment> acknowledgments;

    private int eventCounter;

    public Process(int processId) {

        this.processId = processId;

        this.queue = new PriorityQueue<>();

        this.acknowledgments = new HashSet<>();

        this.eventCounter = 0;

    }

    public void multicast(String message, CommunicationThread[] threads) {

        this.eventCounter++;

        int eventId = this.eventCounter;

        for (int i = 0; i < threads.length; i++) {

            if (i != this.processId) {

                threads[i].queueMessage(this.processId, eventId, message);

            }

        }

    }

    public void receiveMessage(int senderId, int eventId, String message, CommunicationThread[] threads) {

        this.queue.add(new Message(eventId, senderId, message));

        this.multicastAck(eventId, threads);

    }

    private void multicastAck(int eventId, CommunicationThread[] threads) {

        for (int i = 0; i < threads.length; i++) {

            if (i != this.processId) {

                threads[i].queueAck(this.processId, eventId);

            }

        }

    }

    public void receiveAck(int senderId, int eventId) {

        this.acknowledgments.add(new Acknowledgment(eventId, senderId));

    }

    public void deliverMessages() throws InterruptedException {

        while (!this.queue.isEmpty()) {

            Message message = this.queue.poll();

            Acknowledgment acknowledgment = new Acknowledgment(message.getEventId(), message.getSenderId());

            if (this.acknowledgments.contains(acknowledgment)) {

                System.out.printf("%d: %d.%d: %s%n", this.processId, message.getEventId(), message.getSenderId(), message.getMessage());

                this.acknowledgments.remove(acknowledgment);

            }

        }

    }

    public int getProcessId() {

        return processId;

    }

    public int getEventCounter() {

        return eventCounter;

    }

}

class Message implements Comparable<Message> {

    private int eventId;

    private int senderId;

    private String message;

    public Message(int eventId, int senderId, String message) {

        this.eventId = eventId;

        this.senderId = senderId;

        this.message = message;

    }

    public int getEventId() {

        return eventId;

    }

    public int getSenderId() {

        return senderId;

    }

    public String getMessage() {

        return message;

    }

    @Override

    public int compareTo(Message other) {

        return Integer.compare(this.eventId, other.eventId);

    }

}

class Acknowledgment {

    private int eventId;

    private int senderId;

    public Acknowledgment(int eventId, int senderId) {

        this.eventId = eventId;

        this.senderId = senderId;

    }

    @Override

    public boolean equals(Object obj) {

        if (this == obj) return true;

        if (obj == null || getClass() != obj.getClass()) return false;

        Acknowledgment that = (Acknowledgment) obj;

        return eventId == that.eventId && senderId == that.senderId;

    }

    @Override

    public int hashCode() {

        return 31 \* eventId + senderId;

    }

}

class CommunicationThread extends Thread {

    private Process process;

    private CommunicationThread[] threads;

    private boolean exitEvent;

    public CommunicationThread(Process process, CommunicationThread[] threads) {

        this.process = process;

        this.threads = threads;

        this.exitEvent = false;

    }

    public void run() {

        while (!exitEvent) {

            try {

                Thread.sleep(2000);

                String message = String.format("Message from Process %d", this.process.getProcessId());

                int senderId = (this.process.getProcessId() + 1) % threads.length;

                this.process.multicast(message, threads); // Increment event counter internally

            } catch (InterruptedException e) {

                break;

            }

        }

    }

    public void queueMessage(int senderId, int eventId, String message) {

        this.process.receiveMessage(senderId, eventId, message, threads);

    }

    public void queueAck(int senderId, int eventId) {

        this.process.receiveAck(senderId, eventId);

    }

}

class DeliveryThread extends Thread {

    private Process process;

    private boolean exitEvent;

    public DeliveryThread(Process process) {

        this.process = process;

        this.exitEvent = false;

    }

    public void run() {

        while (!exitEvent) {

            try {

                this.process.deliverMessages();

                Thread.sleep(1000);  // Adjust the sleep duration as needed

            } catch (InterruptedException e) {

                // Handle the InterruptedException as needed

                break;

            }

        }

    }

    public void setExitEvent() {

        this.exitEvent = true;

    }

}

public class Main {

    private static final int NUM\_PROCESSES = System.getenv("NUM\_PROCESSES") != null ?

            Integer.parseInt(System.getenv("NUM\_PROCESSES")) : 3;

    private static CommunicationThread[] threads;

    private static Process[] processes;

    private static DeliveryThread[] deliveryThreads;

    public static void main(String[] args) {

        if (NUM\_PROCESSES >= 3) {

            processes = new Process[NUM\_PROCESSES];

            threads = new CommunicationThread[NUM\_PROCESSES];

            deliveryThreads = new DeliveryThread[NUM\_PROCESSES];

            for (int i = 0; i < NUM\_PROCESSES; i++) {

                processes[i] = new Process(i);

                threads[i] = new CommunicationThread(processes[i], threads);

                threads[i].start();

                deliveryThreads[i] = new DeliveryThread(processes[i]);

                deliveryThreads[i].start();

            }

            try {

                while (true) {

                    Thread.sleep(1000);

                }

            } catch (InterruptedException e) {

                System.out.println("Simulation terminated.");

                for (CommunicationThread thread : threads) {

                    thread.interrupt();

                }

                for (DeliveryThread thread : deliveryThreads) {

                    thread.setExitEvent();

                    thread.interrupt();

                }

            }

        } else {

            System.out.println("the no.of processes should be >= 3");

        }

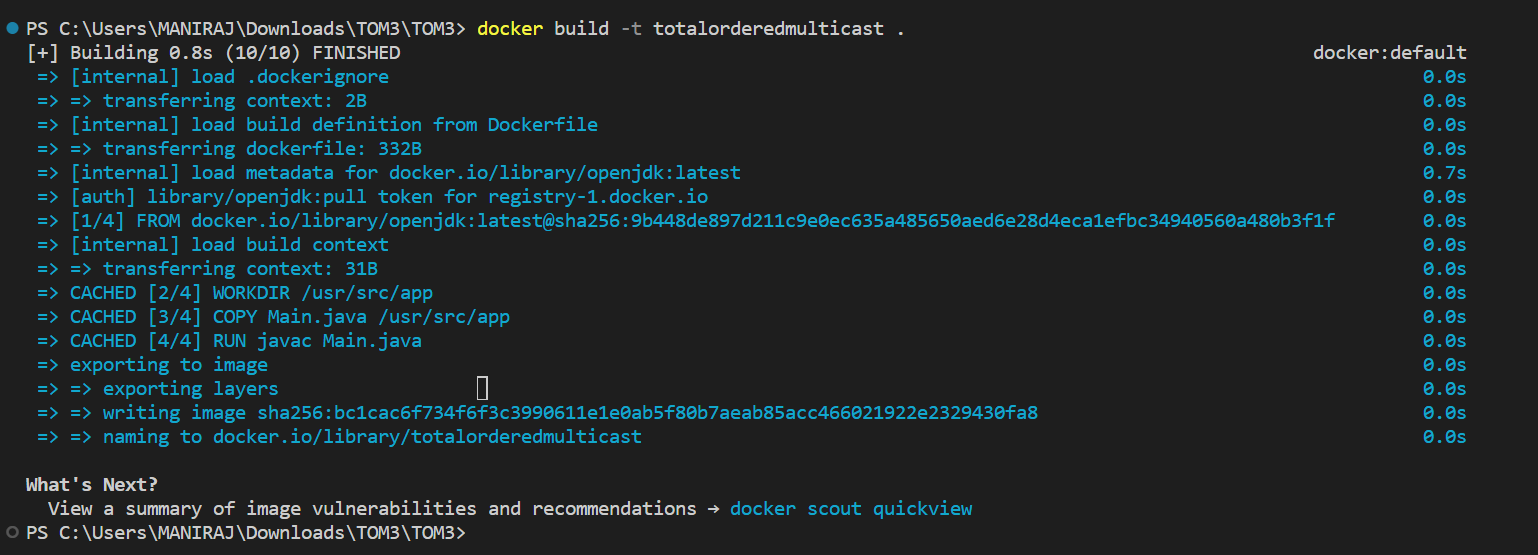
    }

}

**Execution:**

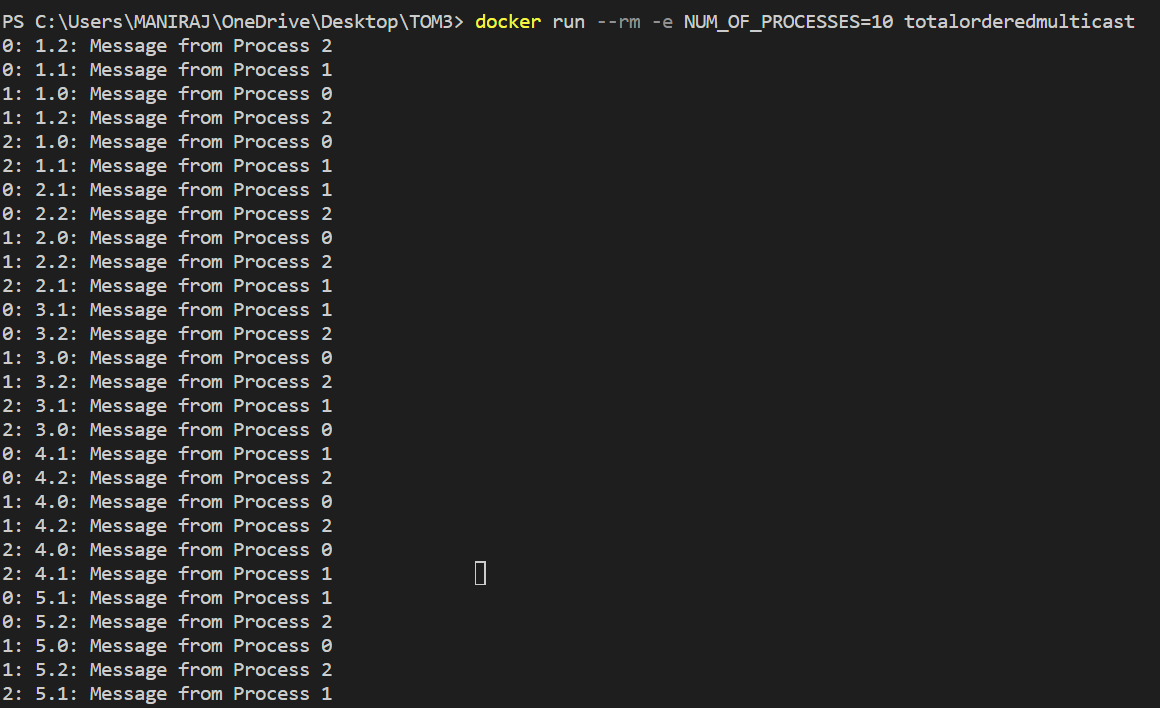
**Command: build "docker build -t totalordermulticast ."**

This command is used to build the docker image by name of totalorderedmulticast. Screenshot has been placed below.

****

**Command: docker run --rm -e NUM\_OF\_PROCESSES=10 totalordermulticast**

This commandis used to run a Docker container based on the "totalordermulticast" image.

****

**Name: Maniraj Patha**

**Student ID: 1002167994**

Contributed to the design and implementation of the Process class, focusing on initializing processes, managing message queues, and handling acknowledgments.

Collaborated on the Message and Acknowledgment classes, ensuring proper event ID comparisons and acknowledgment management.

**Name: Lokeshwar Kodipunjula**

**Student ID: 1002175121**

Collaborated on the Message and Acknowledgment classes, ensuring proper event ID comparisons and acknowledgment management.

Led the design of the Delivery Thread class, managing the delivery of messages to the application. Ensured synchronization and proper handling of interruptions.

**Both Worked together:**

Contributed to the main entry point, the Main class, orchestrating the initialization of processes, communication threads, and delivery threads. Managed the overall execution flow and termination of the simulation.

Collaborated on integrating individual components, resolving dependencies, and testing the entire system. Ensured that the simulation ran smoothly, identified potential issues, and iteratively refined the code and we have focused on adding comments, and documenting the codebase.

**Test Cases:**

**TS1: min no. of processes**

i/p: docker run -e NUM\_OF\_PROCESSES=2 totalordermulticast

o/p:

0: 1.1: Message from Process 1

0: 1.2: Message from Process 2

2: 1.0: Message from Process 0

1: 1.0: Message from Process 0

1: 1.2: Message from Process 2

2: 1.1: Message from Process 1

**TS2: Normal Execution**

i/p: docker run -e NUM\_OF\_PROCESSES=3 totalordermulticast

o/p:

0: 1.1: Message from Process 1

0: 1.2: Message from Process 2

2: 1.1: Message from Process 1

2: 1.0: Message from Process 0

1: 1.0: Message from Process 0

1: 1.2: Message from Process 2

0: 2.1: Message from Process 1

2: 2.1: Message from Process 1

2: 2.0: Message from Process 0

1: 2.2: Message from Process 2

1: 2.0: Message from Process 0

0: 3.1: Message from Process 1

2: 3.1: Message from Process 1

2: 3.0: Message from Process 0

1: 3.2: Message from Process 2

1: 3.0: Message from Process 0

.

.

.

.

**TS3: Just running the code**

i/p: docker run totalordermulticast

o/p:

0: 1.2: Message from Process 2

1: 1.0: Message from Process 0

1: 1.2: Message from Process 2

2: 1.1: Message from Process 1

2: 1.0: Message from Process 0

0: 2.2: Message from Process 2

0: 2.1: Message from Process 1

1: 2.0: Message from Process 0

1: 2.2: Message from Process 2

2: 2.0: Message from Process 0

2: 2.1: Message from Process 1

0: 3.2: Message from Process 2

0: 3.1: Message from Process 1

1: 3.0: Message from Process 0

1: 3.2: Message from Process 2

2: 3.0: Message from Process 0

2: 3.1: Message from Process 1

0: 4.2: Message from Process 2

0: 4.1: Message from Process 1

1: 4.0: Message from Process 0

1: 4.2: Message from Process 2

2: 4.0: Message from Process 0

2: 4.1: Message from Process 1

.

.

.

.

**TS4: large no. of processes**

i/p: docker run -e NUM\_OF\_PROCESSES=10 totalordermulticast

o/p:

0: 1.2: Message from Process 2

0: 1.1: Message from Process 1

2: 1.0: Message from Process 0

1: 1.0: Message from Process 0

1: 1.2: Message from Process 2

2: 1.1: Message from Process 1

0: 2.1: Message from Process 1

0: 2.2: Message from Process 2

1: 2.0: Message from Process 0

1: 2.2: Message from Process 2

2: 2.0: Message from Process 0

0: 3.1: Message from Process 1

0: 3.2: Message from Process 2

1: 3.0: Message from Process 0

1: 3.2: Message from Process 2

2: 3.0: Message from Process 0

0: 4.1: Message from Process 1

0: 4.2: Message from Process 2

1: 4.0: Message from Process 0

1: 4.2: Message from Process 2

2: 4.1: Message from Process 1

0: 5.1: Message from Process 1

0: 5.2: Message from Process 2

1: 5.0: Message from Process 0

1: 5.2: Message from Process 2

2: 5.1: Message from Process 1

2: 5.0: Message from Process 0

0: 6.2: Message from Process 2

0: 6.1: Message from Process 1

1: 6.2: Message from Process 2

1: 6.0: Message from Process 0

2: 6.0: Message from Process 0

2: 6.1: Message from Process 1

0: 7.1: Message from Process 1

1: 7.2: Message from Process 2

1: 7.0: Message from Process 0

2: 7.1: Message from Process 1

2: 7.0: Message from Process 0

0: 8.1: Message from Process 1

0: 8.2: Message from Process 2

1: 8.2: Message from Process 2

1: 8.0: Message from Process 0

2: 8.1: Message from Process 1

2: 8.0: Message from Process 0

0: 9.2: Message from Process 2

0: 9.1: Message from Process 1

1: 9.2: Message from Process 2

1: 9.0: Message from Process 0

2: 9.0: Message from Process 0

0: 10.2: Message from Process 2

0: 10.1: Message from Process 1

1: 10.2: Message from Process 2

1: 10.0: Message from Process 0

2: 10.1: Message from Process 1

2: 10.0: Message from Process 0

.

.

.

.

**TS5: running program and interrupt**

i/p: docker run -e NUM\_OF\_PROCESSES=5 totalordermulticast

o/p:

0: 1.2: Message from Process 2

0: 1.1: Message from Process 1

2: 1.0: Message from Process 0

2: 1.1: Message from Process 1

1: 1.0: Message from Process 0

1: 1.2: Message from Process 2

0: 2.2: Message from Process 2

0: 2.1: Message from Process 1

2: 2.1: Message from Process 1

2: 2.0: Message from Process 0

1: 2.2: Message from Process 2

1: 2.0: Message from Process 0

0: 3.2: Message from Process 2

0: 3.1: Message from Process 1

2: 3.1: Message from Process 1

2: 3.0: Message from Process 0

1: 3.2: Message from Process 2

1: 3.0: Message from Process 0

0: 4.2: Message from Process 2

2: 4.1: Message from Process 1

2: 4.0: Message from Process 0

1: 4.2: Message from Process 2

1: 4.0: Message from Process 0

0: 5.1: Message from Process 1

0: 5.2: Message from Process 2

2: 5.0: Message from Process 0

2: 5.1: Message from Process 1

1: 5.0: Message from Process 0

1: 5.2: Message from Process 2

^C%

**References:**

<https://cs.stackexchange.com/questions/23847/totally-ordered-multicast-with-lamport-timestamp>

<https://condor.depaul.edu/elliott/435/hw/programs/program-TotOrdered.html>

<https://slideplayer.com/slide/8289838/>

<https://stackoverflow.com/questions/36001200/totally-ordered-multicast-with-lamport-clocks-without-fifo>