SOFE 4790U
Distributed Systems

Lecture CRN 43525

Jeremy Mark Tubongbanua
100849092

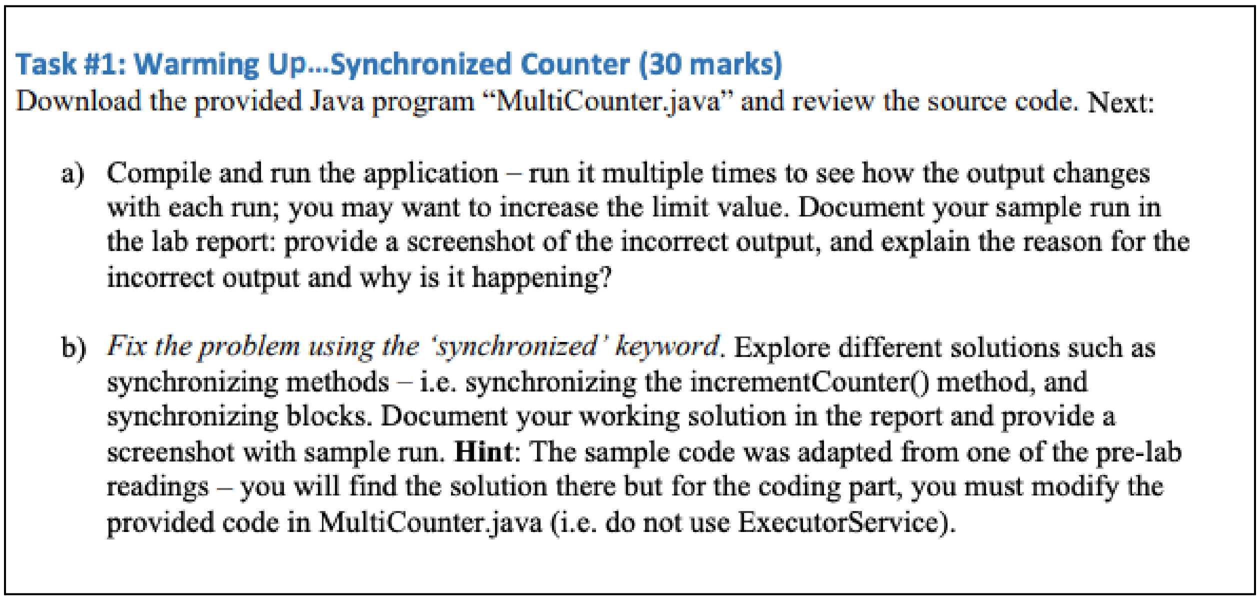
Lab 5

November 30, 2024

Table of Contents

Table of Contents......**2**
Task 1......**2**
Question A......2
Output......2
MultiCounter.java......4
Question B......5
MultiCounterSynchro.java......5
Output......6
Task 2......**8**
Task 3......**9**

Task 1



Task #1: Warming Up...Synchronized Counter (30 marks)
Download the provided Java program "MultiCounter.java" and review the source code. Next:

a) Compile and run the application – run it multiple times to see how the output changes with each run; you may want to increase the limit value. Document your sample run in the lab report: provide a screenshot of the incorrect output, and explain the reason for the incorrect output and why is it happening?

b) *Fix the problem using the 'synchronized' keyword*. Explore different solutions such as synchronizing methods – i.e. synchronizing the incrementCounter() method, and synchronizing blocks. Document your working solution in the report and provide a screenshot with sample run. **Hint**: The sample code was adapted from one of the pre-lab readings – you will find the solution there but for the coding part, you must modify the provided code in MultiCounter.java (i.e. do not use ExecutorService).

Question A

Running MultiCounter.java with a limit of 100 gives the following output, as shown in "Output".

The counter doesn't increment properly one-by-one, almost as if the counter is shared among the threads. That is what is incorrect about the code given during the lab.

Output

```
sofe4790u-lab5 javac */*.java
➜  sofe4790u-lab5 java task1.MultiCounter
```

```
Thread-2 counter : 0
Thread-2 counter : 1
Thread-2 counter : 2
Thread-1 counter : 0
Thread-1 counter : 4
Thread-0 counter : 0
Thread-2 counter : 3
Thread-1 counter : 5
Thread-1 counter : 8
Thread-1 counter : 9
Thread-1 counter : 10
Thread-1 counter : 11
Thread-1 counter : 12
Thread-0 counter : 6
Thread-0 counter : 14
Thread-0 counter : 15
Thread-0 counter : 16
Thread-2 counter : 7
Thread-2 counter : 18
Thread-2 counter : 19
Thread-2 counter : 20
Thread-2 counter : 21
Thread-1 counter : 13
Thread-1 counter : 23
Thread-1 counter : 24
Thread-1 counter : 25
Thread-1 counter : 26
Thread-1 counter : 27
Thread-1 counter : 28
Thread-0 counter : 17
Thread-2 counter : 22
Thread-2 counter : 31
Thread-2 counter : 32
Thread-2 counter : 33
Thread-2 counter : 34
Thread-2 counter : 35
Thread-2 counter : 36
Thread-2 counter : 37
Thread-2 counter : 38
Thread-1 counter : 29
Thread-1 counter : 40
Thread-1 counter : 41
Thread-1 counter : 42
Thread-1 counter : 43
Thread-1 counter : 44
Thread-1 counter : 45
Thread-1 counter : 46
Thread-0 counter : 30
Thread-2 counter : 39
Thread-2 counter : 49
Thread-2 counter : 50
Thread-1 counter : 47
Thread-1 counter : 52
Thread-0 counter : 48
Thread-2 counter : 51
Thread-2 counter : 55
Thread-2 counter : 56
Thread-2 counter : 57
Thread-1 counter : 53
```

```
Thread-1 counter : 59
Thread-1 counter : 60
Thread-0 counter : 54
Thread-2 counter : 58
Thread-2 counter : 63
Thread-1 counter : 61
Thread-1 counter : 65
Thread-1 counter : 66
Thread-0 counter : 62
Thread-0 counter : 68
Thread-0 counter : 69
Thread-0 counter : 70
Thread-0 counter : 71
Thread-0 counter : 72
Thread-0 counter : 73
Thread-0 counter : 74
Thread-0 counter : 75
Thread-2 counter : 64
Thread-1 counter : 67
Thread-1 counter : 78
Thread-1 counter : 79
Thread-1 counter : 80
Thread-1 counter : 81
Thread-1 counter : 82
Thread-1 counter : 83
Thread-0 counter : 76
Thread-0 counter : 85
Thread-0 counter : 86
Thread-0 counter : 87
Thread-0 counter : 88
Thread-0 counter : 89
Thread-2 counter : 77
Thread-1 counter : 84
Thread-1 counter : 92
Thread-1 counter : 93
Thread-0 counter : 90
Thread-0 counter : 95
Thread-2 counter : 91
Thread-1 counter : 94
Thread-0 counter : 96
Thread-0 counter : 99
Thread-2 counter : 97
Thread-1 counter : 98
```

MultiCounter.java

```
package task1;

public class MultiCounter extends Thread {

   private static int counter = 0;
   private static final int limit = 100;

   public static void main(String argv[]) throws InterruptedException {
      MultiCounter t1 = new MultiCounter();
      MultiCounter t2 = new MultiCounter();
```

```
        MultiCounter t3 = new MultiCounter();

        t1.start();
        t2.start();
        t3.start();

        t1.join();
        t2.join();
        t3.join();
    }

    public void run() {
        while (counter < limit) {
            incrementCounter();
        }
    }


    public void incrementCounter() {
        System.out.println(Thread.currentThread().getName() + " counter : " + counter);
        counter++;
    }

}
```

Question B

MultiCounterSynchro.java

```
public class MultiCounterSynchro extends Thread {

    private static int counter = 0;
    private static final int limit = 100;

    public static void main(String argv[]) throws InterruptedException {
        MultiCounterSynchro t1 = new MultiCounterSynchro();
        MultiCounterSynchro t2 = new MultiCounterSynchro();
        MultiCounterSynchro t3 = new MultiCounterSynchro();

        t1.start();
        t2.start();
        t3.start();

        t1.join();
        t2.join();
        t3.join();
    }

    public void run() {
        while (counter < limit) {
            incrementCounter();
        }
    }
```

```
   public void incrementCounter() {

      synchronized (this) {
         System.out.println(Thread.currentThread().getName() + " counter : " + counter);
         counter++;
      }

   }

}
```

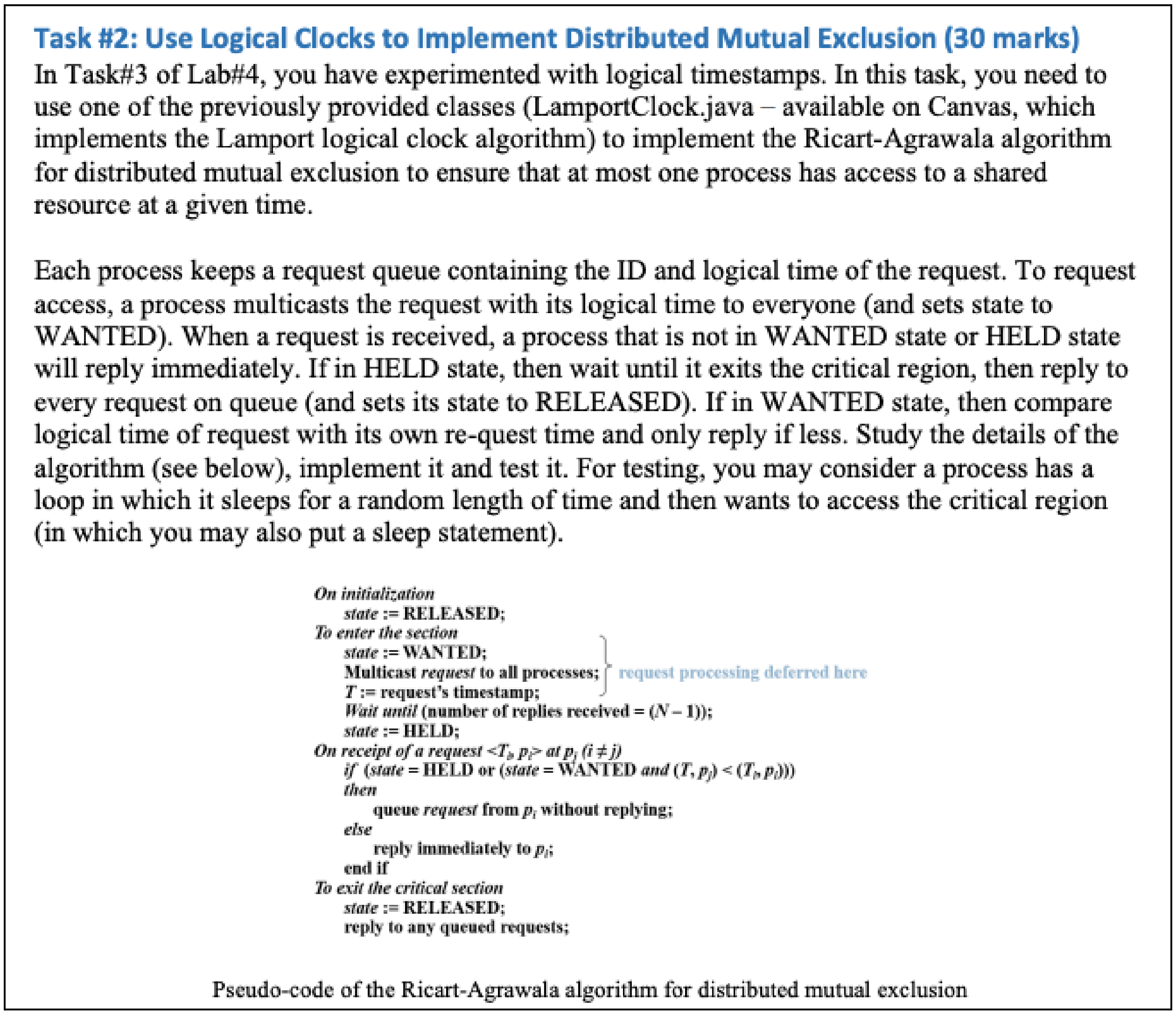
Output

```
sofe4790u-lab5 java task1.MultiCounterSynchro
Thread-2 counter : 0
Thread-2 counter : 1
Thread-0 counter : 0
Thread-0 counter : 3
Thread-0 counter : 4
Thread-0 counter : 5
Thread-0 counter : 6
Thread-0 counter : 7
Thread-0 counter : 8
Thread-0 counter : 9
Thread-0 counter : 10
Thread-1 counter : 0
Thread-1 counter : 12
Thread-2 counter : 2
Thread-0 counter : 11
Thread-0 counter : 15
Thread-0 counter : 16
Thread-1 counter : 13
Thread-1 counter : 18
Thread-1 counter : 19
Thread-1 counter : 20
Thread-1 counter : 21
Thread-2 counter : 14
Thread-0 counter : 17
Thread-0 counter : 24
Thread-0 counter : 25
Thread-0 counter : 26
Thread-0 counter : 27
Thread-1 counter : 22
Thread-0 counter : 28
Thread-0 counter : 30
Thread-2 counter : 23
Thread-0 counter : 31
Thread-0 counter : 33
Thread-0 counter : 34
Thread-1 counter : 29
Thread-2 counter : 32
Thread-2 counter : 37
Thread-2 counter : 38
Thread-2 counter : 39
Thread-2 counter : 40
```

```
Thread-2 counter : 41
Thread-0 counter : 35
Thread-2 counter : 42
Thread-2 counter : 44
Thread-2 counter : 45
Thread-1 counter : 36
Thread-0 counter : 43
Thread-0 counter : 48
Thread-0 counter : 49
Thread-2 counter : 46
Thread-1 counter : 47
Thread-1 counter : 52
Thread-1 counter : 53
Thread-0 counter : 50
Thread-0 counter : 55
Thread-0 counter : 56
Thread-2 counter : 51
Thread-2 counter : 58
Thread-2 counter : 59
Thread-2 counter : 60
Thread-1 counter : 54
Thread-0 counter : 57
Thread-2 counter : 61
Thread-2 counter : 64
Thread-2 counter : 65
Thread-1 counter : 62
Thread-0 counter : 63
Thread-0 counter : 68
Thread-0 counter : 69
Thread-2 counter : 66
Thread-2 counter : 71
Thread-2 counter : 72
Thread-2 counter : 73
Thread-2 counter : 74
Thread-1 counter : 67
Thread-1 counter : 76
Thread-1 counter : 77
Thread-1 counter : 78
Thread-1 counter : 79
Thread-1 counter : 80
Thread-1 counter : 81
Thread-0 counter : 70
Thread-0 counter : 83
Thread-0 counter : 84
Thread-0 counter : 85
Thread-0 counter : 86
Thread-0 counter : 87
Thread-0 counter : 88
Thread-0 counter : 89
Thread-0 counter : 90
Thread-0 counter : 91
Thread-2 counter : 75
Thread-1 counter : 82
Thread-1 counter : 94
Thread-0 counter : 92
Thread-0 counter : 96
Thread-2 counter : 93
Thread-1 counter : 95
Thread-0 counter : 97
```

```
Thread-2 counter : 98
Thread-1 counter : 99
```

Task 2



Task #2: Use Logical Clocks to Implement Distributed Mutual Exclusion (30 marks)

In Task#3 of Lab#4, you have experimented with logical timestamps. In this task, you need to use one of the previously provided classes (LamportClock.java – available on Canvas, which implements the Lamport logical clock algorithm) to implement the Ricart-Agrawala algorithm for distributed mutual exclusion to ensure that at most one process has access to a shared resource at a given time.

Each process keeps a request queue containing the ID and logical time of the request. To request access, a process multicasts the request with its logical time to everyone (and sets state to WANTED). When a request is received, a process that is not in WANTED state or HELD state will reply immediately. If in HELD state, then wait until it exits the critical region, then reply to every request on queue (and sets its state to RELEASED). If in WANTED state, then compare logical time of request with its own re-quest time and only reply if less. Study the details of the algorithm (see below), implement it and test it. For testing, you may consider a process has a loop in which it sleeps for a random length of time and then wants to access the critical region (in which you may also put a sleep statement).

```
On initialization
    state := RELEASED;
To enter the section
    state := WANTED;                              ⎫
    Multicast request to all processes;           ⎬ request processing deferred here
    T := request's timestamp;                     ⎭
    Wait until (number of replies received = (N – 1));
    state := HELD;
On receipt of a request <T_i, p_i> at p_j (i ≠ j)
    if (state = HELD or (state = WANTED and (T, p_j) < (T_i, p_i)))
    then
        queue request from p_i without replying;
    else
        reply immediately to p_i;
    end if
To exit the critical section
    state := RELEASED;
    reply to any queued requests;
```

Pseudo-code of the Ricart-Agrawala algorithm for distributed mutual exclusion

Output

```
sofe4790u-lab5 java task2.LamportClock
Enter Number of processes:
3
Enter your process id:
```

```
0

1. Request Access
2. Recieve Request
3. Exit
1
Process 0 is requesting critical section
Enter reply (Y/N) from Process 1:
Y
Enter reply (Y/N) from Process 2:
Y
Process 0 enter critical section
Process 0 releases critical section

1. Request Access
2. Recieve Request
3. Exit
2
Enter process id  and timestamp (space seperated):
1 5
Process 0 sends reply to Process 1

1. Request Access
2. Recieve Request
3. Exit
1
Process 0 is requesting critical section
Enter reply (Y/N) from Process 1:
N
Enter reply (Y/N) from Process 2:
Y

1. Request Access
2. Recieve Request
3. Exit
2
Enter process id  and timestamp (space seperated):
1 4
Process 0 sends reply to Process 1

1. Request Access
2. Recieve Request
3. Exit
```

```
1
Process 0 is requesting critical section
Enter reply (Y/N) from Process 1:
Y
Enter reply (Y/N) from Process 2:
Y
Process 0 enter critical section
Process 0 releases critical section

1. Request Access
2. Recieve Request
3. Exit
1
Process 0 is requesting critical section
Enter reply (Y/N) from Process 1:
N
Enter reply (Y/N) from Process 2:
N

1. Request Access
2. Recieve Request
3. Exit
2 4
```

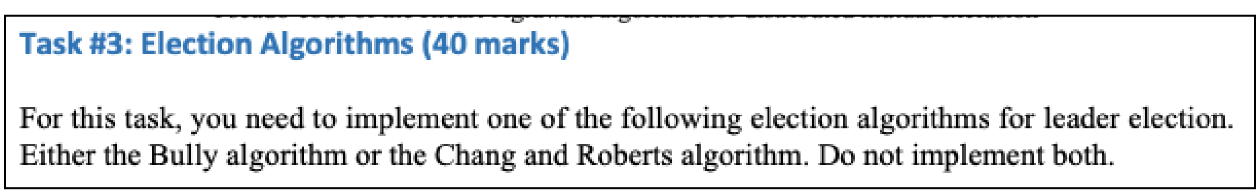
LamportClock.java

```java
package task2;

import java.util.*;

public class LamportClock {
   int c;
   String state;
   Queue<Request> requestQueue;
   int processID;
   int numProcesses;
   Set<Integer> replies;

   // Each process keeps an integer initially 0
   public LamportClock(int processID, int numProcesses) {
        this.c = 0;
        this.state = "RELEASED";
        this.requestQueue = new LinkedList<>();
        this.processID = processID;
        this.numProcesses = numProcesses;
        this.replies = new HashSet<>();
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter Number of processes: ");
        int totalProcesses = Integer.parseInt(scanner.nextLine());

        System.out.println("Enter your process id: ");
        int processID = Integer.parseInt(scanner.nextLine());

        LamportClock process = new LamportClock(processID, totalProcesses);

        while (true) {
            System.out.println("\n1. Request Access\n2. Recieve Request\n3.
Exit");
            int choice = Integer.parseInt(scanner.nextLine());
            if (choice==1) {
                process.requestAccess(scanner);
            }
            else if (choice == 2) {
                System.out.println("Enter process id  and timestamp (space
seperated): ");
                String[] input = scanner.nextLine().split(" ");
                int srcID = Integer.parseInt(input[0]);
                int timestamp = Integer.parseInt(input[1]);
                process.recieveRequest(srcID, timestamp);
            }
            else if (choice == 3) {
                System.out.println("Exiting...");
                break;

            }

        }
        scanner.close();
    }
```

```
   public int getValue() {
      return c;
   }

   // Internal even or local step
   public void localStep() {
      c = c + 1;
   }

   public void sendEvent() {
      c = c + 1;
   }

   public void receiveEvent(int src, int sentValue) {
      c = max(c, sentValue) + 1;
   }

   public int max(int a, int b) {
      return a > b ? a : b;
   }

   public synchronized void requestAccess(Scanner scanner){
      localStep();
      state = "WANTED";
      replies.clear();

      System.out.println("Process " + processID + " is requesting critical
section ");
      for(int i = 0; i < numProcesses; i++) {
         if (i != processID) {
            System.out.println("Enter reply (Y/N) from Process " + i + ":
");
            String reponse = scanner.nextLine().trim();
            if (reponse.equals("Y")){
               replies.add(i);
            }
         }
      }

      if (replies.size() == numProcesses - 1) {
         state = "HELD";
         System.out.println("Process " + processID + " enter critical
section");
```

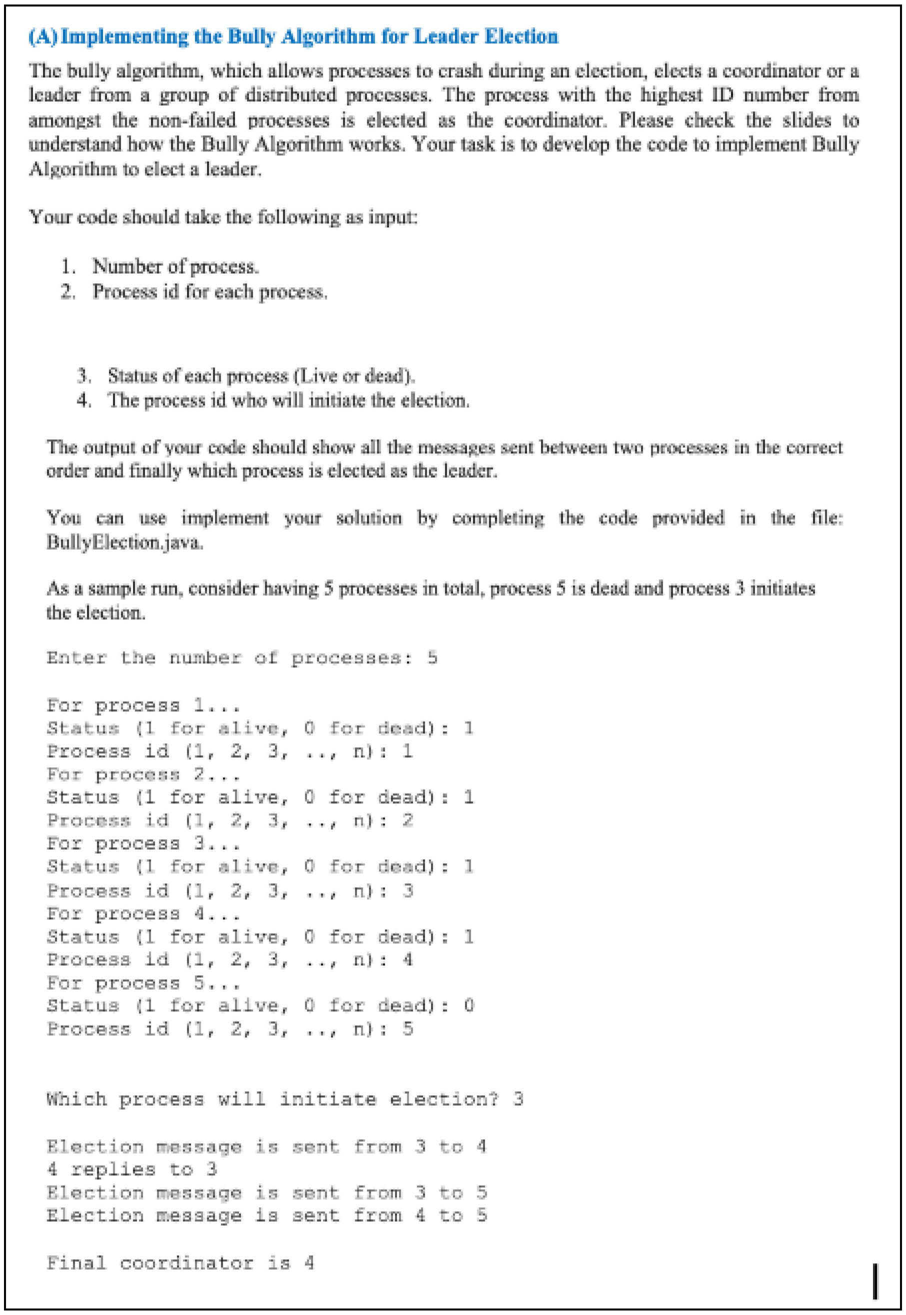
```
         try {
             Thread.sleep(2000);
         } catch (Exception e) {
            e.printStackTrace();
         }
         releaseAccess();
      }

   }

   public synchronized void releaseAccess() {
      state = "RELEASED";
      System.out.println("Process " + processID + " releases critical
section");
      while (!requestQueue.isEmpty()) {
         Request req = requestQueue.poll();
         sendReply(req.processID);
      }
   }

   public synchronized void recieveRequest(int srcProcessID, int timestamp)
{
      receiveEvent(srcProcessID, timestamp);
      Request currentRequest = new Request(processID, c);
      Request incomingRequest = new Request(srcProcessID, timestamp);

      if (!state.equals("HELD")&&
(state.equals("RELEASED")||compareRequests(incomingRequest, currentRequest)
< 0)) {
         sendReply(srcProcessID);

      }
      else {
         requestQueue.add(incomingRequest);
      }
   }

   public void sendReply(int targetProcessID) {
      System.out.println("Process " + processID + " sends reply to Process
"+ targetProcessID);

   }
   private int compareRequests(Request r1, Request r2) {
      if (r1.timestamp != r2.timestamp) {

         return Integer.compare(r1.timestamp, r2.timestamp);
      }
      return Integer.compare(r1.processID, r2.processID);

   }

   static class Request {
      int processID;
      int timestamp;

      public Request(int processID, int timestamp){
         this.processID = processID;
         this.timestamp = timestamp;
      }
   }



}
```

Task 3



Task #3: Election Algorithms (40 marks)

For this task, you need to implement one of the following election algorithms for leader election. Either the Bully algorithm or the Chang and Roberts algorithm. Do not implement both.



(A) Implementing the Bully Algorithm for Leader Election

The bully algorithm, which allows processes to crash during an election, elects a coordinator or a leader from a group of distributed processes. The process with the highest ID number from amongst the non-failed processes is elected as the coordinator. Please check the slides to understand how the Bully Algorithm works. Your task is to develop the code to implement Bully Algorithm to elect a leader.

Your code should take the following as input:

1. Number of process.
2. Process id for each process.
3. Status of each process (Live or dead).
4. The process id who will initiate the election.

The output of your code should show all the messages sent between two processes in the correct order and finally which process is elected as the leader.

You can use implement your solution by completing the code provided in the file: BullyElection.java.

As a sample run, consider having 5 processes in total, process 5 is dead and process 3 initiates the election.

```
Enter the number of processes: 5

For process 1...
Status (1 for alive, 0 for dead): 1
Process id (1, 2, 3, .., n): 1
For process 2...
Status (1 for alive, 0 for dead): 1
Process id (1, 2, 3, .., n): 2
For process 3...
Status (1 for alive, 0 for dead): 1
Process id (1, 2, 3, .., n): 3
For process 4...
Status (1 for alive, 0 for dead): 1
Process id (1, 2, 3, .., n): 4
For process 5...
Status (1 for alive, 0 for dead): 0
Process id (1, 2, 3, .., n): 5


Which process will initiate election? 3

Election message is sent from 3 to 4
4 replies to 3
Election message is sent from 3 to 5
Election message is sent from 4 to 5

Final coordinator is 4
```

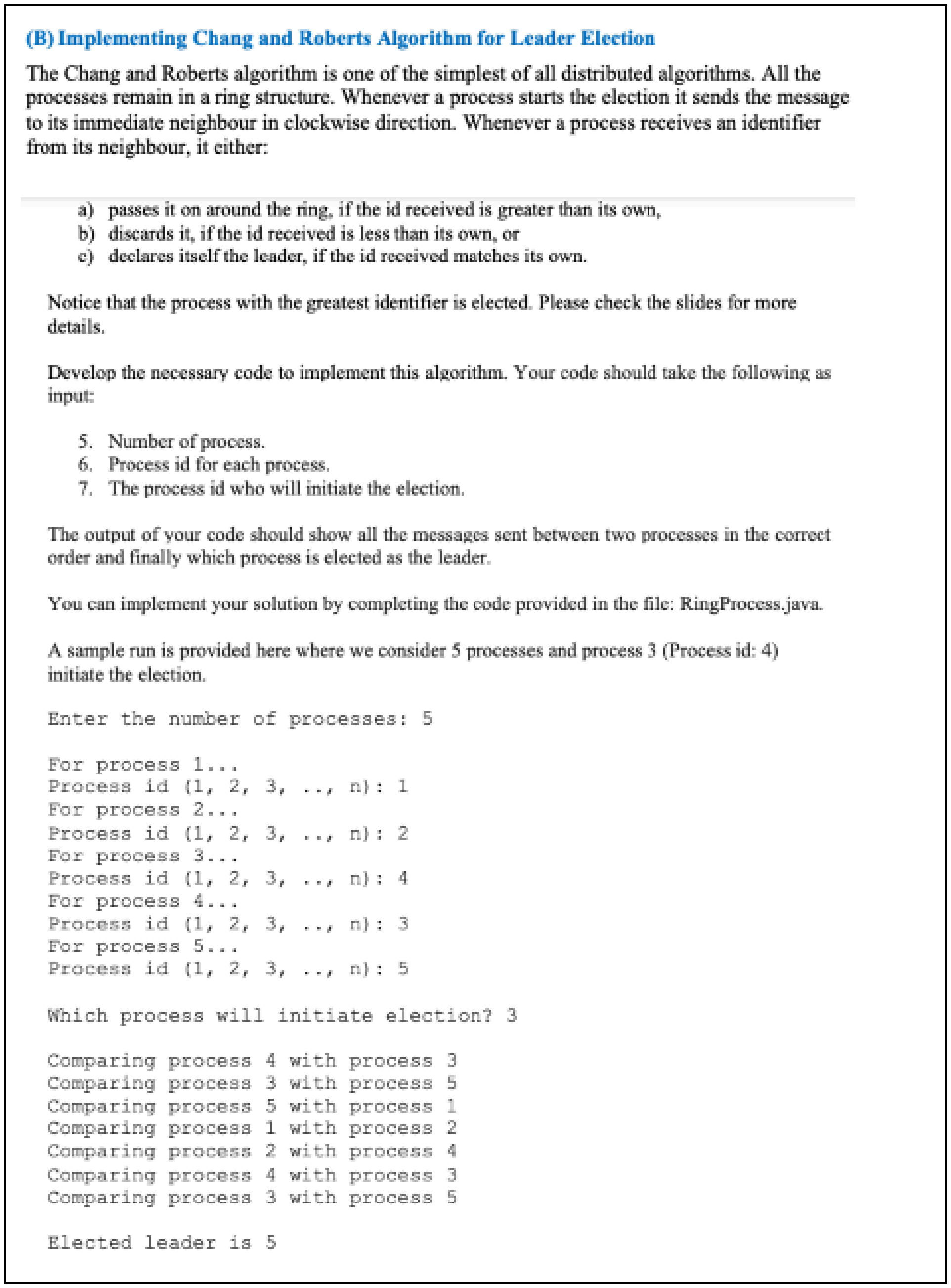
Output

```
  sofe4790u-lab5 java task3.BullyElection
Enter the number of processes: 5
For process 1...
Status (1 for alive, 0 for dead): 1
Process id (1, 2, 3, .., n): 11
For process 2...
```

```
Status (1 for alive, 0 for dead): 1
Process id (1, 2, 3, .., n): 22
For process 3...
Status (1 for alive, 0 for dead): 1
Process id (1, 2, 3, .., n): 33
For process 4...
Status (1 for alive, 0 for dead): 1
Process id (1, 2, 3, .., n): 44
For process 5...
Status (1 for alive, 0 for dead): 0
Process id (1, 2, 3, .., n): 55
Which process will initiate election? 3
Election initiated by process33
Process 33 sends election to Process 44
Election initiated by process44
Coordinator is Process 44
Coordinator is Process 44
Final coordinator is 44
```

BullyElection.java

```java
package task3;
import java.io.*;
import java.util.Scanner;

class BullyElection {
    static int n;
    static int pro[] = new int[100];
    static int sta[] = new int[100];
    static int co;

    public static void main(String args[])throws IOException
    {
        System.out.print("Enter the number of processes: ");
        Scanner in = new Scanner(System.in);
        n = in.nextInt();

        int i,j,k,l,m;

        for(i=0;i<n;i++)
        {
            System.out.println("For process "+(i+1)+"...");
            System.out.print("Status (1 for alive, 0 for dead): ");
            sta[i]=in.nextInt();
            System.out.print("Process id (1, 2, 3, .., n): ");
            pro[i] = in.nextInt();
        }

        System.out.print("Which process will initiate election? ");
        int ele = in.nextInt()-1;

        elect(ele);
        System.out.println("Final coordinator is "+co);
    }


    static void elect(int ele) {
        System.out.println("Election initiated by process" + pro[ele]);
        for (int i = ele + 1; i < n; i++) {
            if (sta[i]==1) {
                System.out.println("Process " + pro[ele] + " sends election
to Process " + pro[i]);
                elect(i);
            }
        }
        if (pro[ele]>co){
            co = pro[ele];
        }
        System.out.println("Coordinator is Process " + co);

    }
}
```



(B) Implementing Chang and Roberts Algorithm for Leader Election

The Chang and Roberts algorithm is one of the simplest of all distributed algorithms. All the processes remain in a ring structure. Whenever a process starts the election it sends the message to its immediate neighbour in clockwise direction. Whenever a process receives an identifier from its neighbour, it either:

a) passes it on around the ring, if the id received is greater than its own,
b) discards it, if the id received is less than its own, or
c) declares itself the leader, if the id received matches its own.

Notice that the process with the greatest identifier is elected. Please check the slides for more details.

Develop the necessary code to implement this algorithm. Your code should take the following as input:

5. Number of process.
6. Process id for each process.
7. The process id who will initiate the election.

The output of your code should show all the messages sent between two processes in the correct order and finally which process is elected as the leader.

You can implement your solution by completing the code provided in the file: RingProcess.java.

A sample run is provided here where we consider 5 processes and process 3 (Process id: 4) initiate the election.

```
Enter the number of processes: 5

For process 1...
Process id (1, 2, 3, .., n): 1
For process 2...
Process id (1, 2, 3, .., n): 2
For process 3...
Process id (1, 2, 3, .., n): 4
For process 4...
Process id (1, 2, 3, .., n): 3
For process 5...
Process id (1, 2, 3, .., n): 5

Which process will initiate election? 3

Comparing process 4 with process 3
Comparing process 3 with process 5
Comparing process 5 with process 1
Comparing process 1 with process 2
Comparing process 2 with process 4
Comparing process 4 with process 3
Comparing process 3 with process 5

Elected leader is 5
```

I decided to go with bully algorithm for this lab