ASSIGNMENT NO. 6

Problem Statement:

To implement Bully and Ring algorithm for leader election.

Tools / Environment:

Java Programming Environment, JDK 1.8, Eclipse Neon(EE).

Related Theory:

Many distributed algorithms require one process to act as coordinator, initiator, or otherwise perform some special role. In general, it does not matter which process takes on this special responsibility, but one of them has to do it. If all processes are exactly the same, with no distinguishing characteristics, there is no way to select one of them to be special. Consequently, we will assume that each process P has a unique identifier id(P). In general, election algorithms attempt to locate the process with the highest identifier and designate it as coordinator.

We also assume that every process knows the identifier of every other process. In other words, each process has complete knowledge of the process group in which a coordinator must be elected. What the processes do not know is which ones are currently up and which ones are currently down. The goal of an election algorithm is to ensure that when an election starts, it concludes with all processes agreeing on who the new coordinator is to be.

There are two types of Distributed Algorithms:
1. Bully Algorithm
2. Ring Algorithm

Bully Algorithm:

A. When a process, P, notices that the coordinator is no longer responding to requests, it initiates an election.
1. P sends an ELECTION message to all processes with higher numbers.
2. If no one responds, P wins the election and becomes a coordinator.
3. If one of the higher-ups answers, it takes over. P’s job is done.

B. When a process gets an ELECTION message from one of its lower-numbered colleagues:
1. Receiver sends an OK message back to the sender to indicate that he is alive and will take over.
2. Eventually, all processes give up apart of one, and that one is the new coordinator.
3. The new coordinator announces *its* victory by sending all processes a **CO-ORDINATOR** message telling them that it is the new coordinator.
C. If a process that *was* previously down comes back:
1. It holds an election.
2. If it happens to be the highest process currently running, it will win the election and take over the coordinators job.

“Biggest guy" always wins and hence the name bully algorithm.

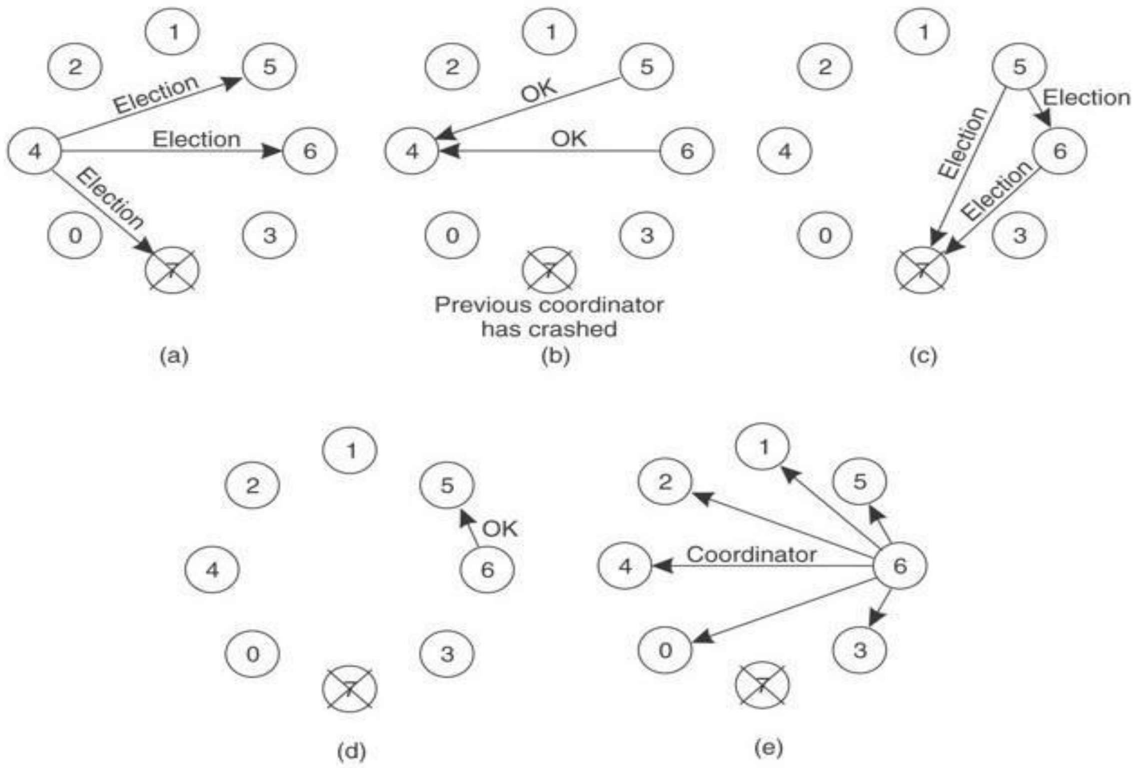


Figure 1: Bully Algorithm

Ring Algorithm:

Initiation:
1. A process notices that coordinator is not functioning:
2. Another process (initiator) initiates the election by sending "ELECTION" message (containing its own process number)

Leader Election:
3. Initiator sends the message to its successor (if successor is down, sender skips over it and goes to the next member along the ring, or the one after that, until a running process is located).
4. At each step, the sender adds its own process number to the list in the message.
5. When the message gets back to the process that started it all i.e Message comes back to initiator, the **process with maximum ID Number** in the queue **wins the Election**.
6. Initiator announces the winner by sending another message (Coordinator message) around the ring.

Implementing the solution:

For Ring Algorithm:

1. Creating Class for Process which includes
    i) State: Active / Inactive

ii) Index: Stores index of process.
iii) ID: Process ID

2. Import Scanner Class for getting input from Console
3. Getting input from User for number of Processes and store them into object of classes.
4. Sort these objects on the basis of process id.
5. Make the last process id as "inactive".
6. Ask for menu
   1.Election
   2.Quit
7. Ask for initializing the election process.
8. These inputs will be used by Ring Algorithm.

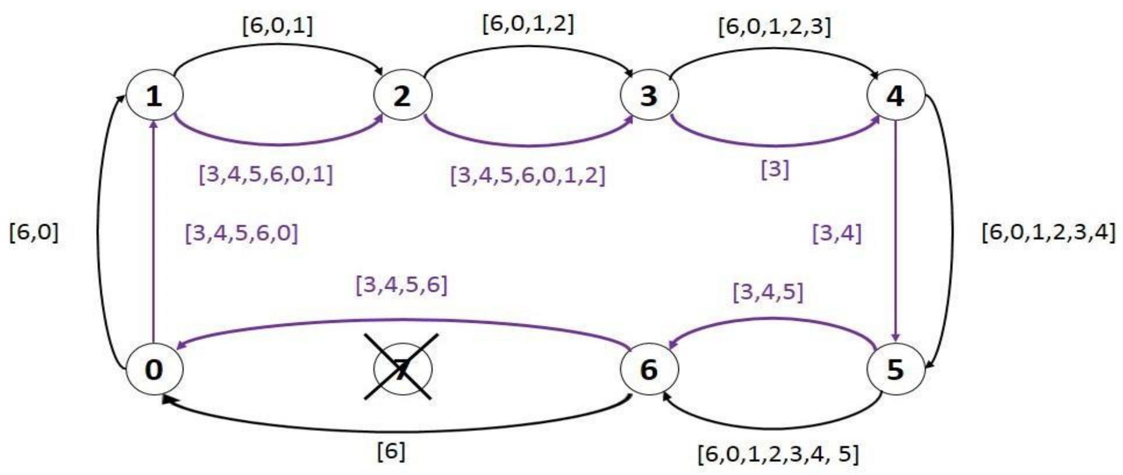


Figure 2: Ring Algorithm

Source code:

Ring.java

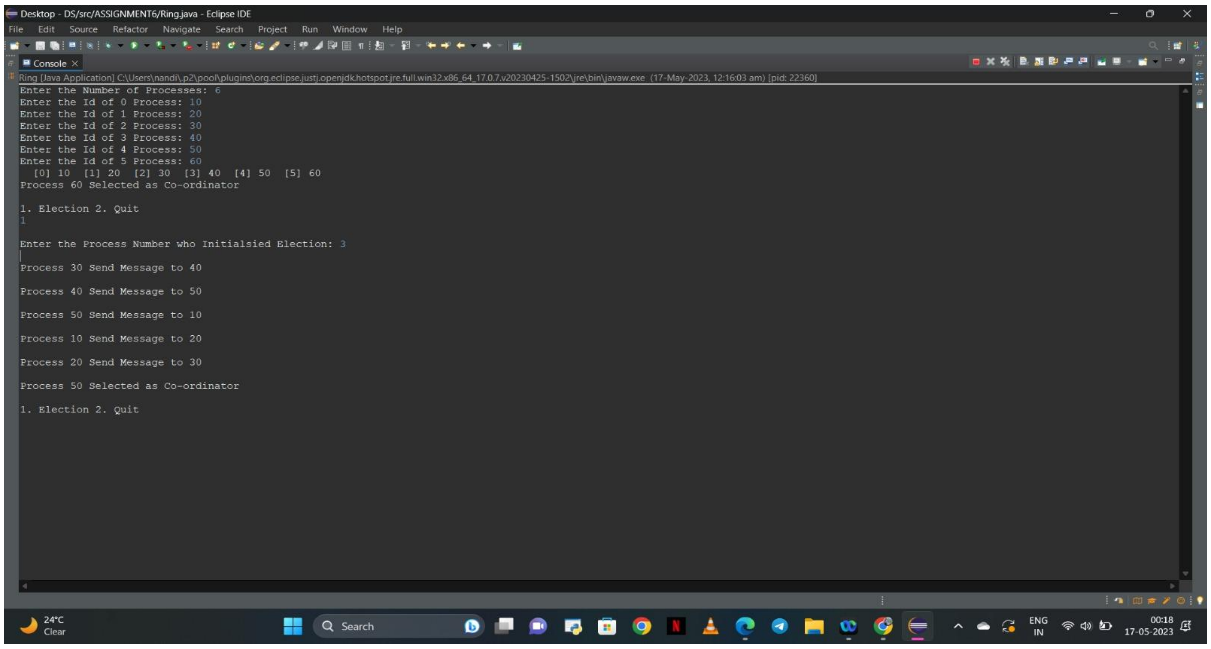
```
import java.util.Scanner;

public class Ring {

      public static void main(String[] args) {

            // TODO Auto-generated method stub

            int temp, i, j;
            char str[] = new char[10];
            Rr proc[] = new Rr[10];

// object initialisation
            for (i = 0; i < proc.length; i++)
                  proc[i] = new Rr();

// scanner used for getting input from console
            Scanner in = new Scanner(System.in);
            System.out.println("Enter the number of process : ");
            int num = in.nextInt();
```

```
// getting input from users
            for (i = 0; i < num; i++) {
                  proc[i].index = i;
                  System.out.println("Enter the id of process : ");
                  proc[i].id = in.nextInt();
                  proc[i].state = "active";
                  proc[i].f = 0;
            }


// sorting the processes from on the basis of id
            for (i = 0; i < num - 1; i++) {
                  for (j = 0; j < num - 1; j++) {
                        if (proc[j].id > proc[j + 1].id) {
                              temp = proc[j].id;
                              proc[j].id = proc[j + 1].id;
                              proc[j + 1].id = temp;
                        }
                  }
            }

            for (i = 0; i < num; i++) {
                  System.out.print("  [" + i + "]" + " " + proc[i].id);
            }

            int init;
            int ch;
            int temp1;
            int temp2;
            int ch1;
            int arr[] = new int[10];

            proc[num - 1].state = "inactive";

            System.out.println("\n process " + proc[num - 1].id + " select
as co-ordinator");

            while (true) {
                  System.out.println("\n 1.election 2.quit ");
                  ch = in.nextInt();

                  for (i = 0; i < num; i++) {
                        proc[i].f = 0;
                  }

                  switch (ch) {
                  case 1:
                        System.out.println("\n Enter the Process number who
initialsied election : ");
                        init = in.nextInt();
                        init--;
                        temp2 = init;
                        temp1 = init + 1;

                        i = 0;

                        while (temp2 != temp1) {
                              if ("active".equals(proc[temp1].state) &&
proc[temp1].f == 0) {
```

```
                                    System.out.println("\nProcess " +
proc[init].id + " send message to " + proc[temp1].id);
                                    proc[temp1].f = 1;
                                    init = temp1;
                                    arr[i] = proc[temp1].id;
                                    i++;
                              }
                              if (temp1 == num) {
                                    temp1 = 0;
                              } else {
                                    temp1++;
                              }
                        }

                        System.out.println("\nProcess " + proc[init].id + "
send message to " + proc[temp1].id);
                        arr[i] = proc[temp1].id;
                        i++;
                        int max = -1;


// finding maximum for co-ordinator selection
                        for (j = 0; j < i; j++) {
                              if (max < arr[j]) {
                                    max = arr[j];
                              }
                        }

// co-ordinator is found then printing on console
                        System.out.println("\n process " + max + "select as
co-ordinator");

                        for (i = 0; i < num; i++) {

                              if (proc[i].id == max) {
                                    proc[i].state = "inactive";
                              }
                        }
                        break;
                  case 2:
            System.out.println("Program terminated ...");
            return ;
                  default:
                        System.out.println("\n invalid response \n");
                        break;
                  }


            }
      }

}

class Rr {

      public int index;   // to store the index of process
      public int id;      // to store id/name of process
      public int f;
      String state;       // indiactes whether active or inactive state of
node

}
```

Compiling and Executing the solution:
1. Create Java Project in Eclipse
2. Create Package
3. Add class in package Ring.java.
4. Compile and Execute in Eclipse.

Output:



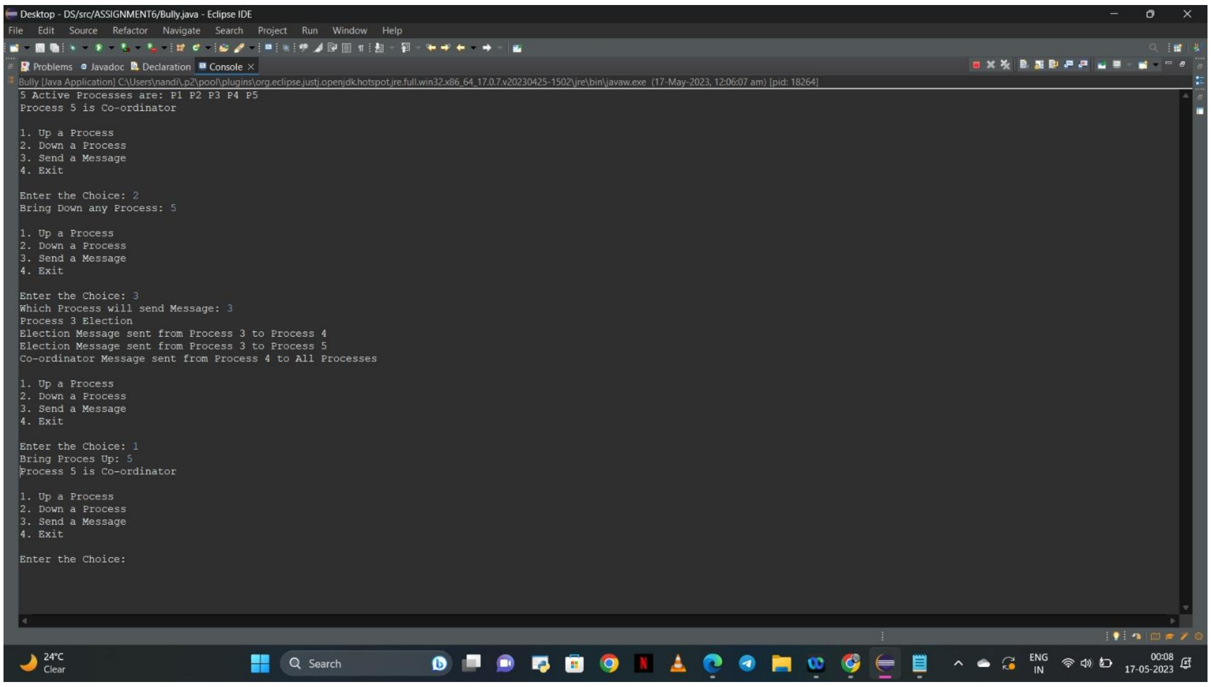
```
Enter the Number of Processes: 6
Enter the Id of 0 Process: 10
Enter the Id of 1 Process: 20
Enter the Id of 2 Process: 30
Enter the Id of 3 Process: 40
Enter the Id of 4 Process: 50
Enter the Id of 5 Process: 60
  [0] 10  [1] 20  [2] 30  [3] 40  [4] 50  [5] 60
Process 60 Selected as Co-ordinator

1. Election 2. Quit
1

Enter the Process Number who Initialsied Election: 3

Process 30 Send Message to 40

Process 40 Send Message to 50

Process 50 Send Message to 10

Process 10 Send Message to 20

Process 20 Send Message to 30

Process 50 Selected as Co-ordinator

1. Election 2. Quit
```

Bully.java

```
import java.io.InputStream;
import java.io.PrintStream;
import java.util.Scanner;

public class Bully {
    static boolean[] state = new boolean[5];
    int coordinator;

    public static void up(int up) {
        if (state[up - 1]) {
            System.out.println("process" + up + "is already up");
        } else {
            int i;
            Bully.state[up - 1] = true;
            System.out.println("process " + up + "held election");
            for (i = up; i < 5; ++i) {
                System.out.println("election message sent from process"
+  up + "to process" + (i + 1));
            }
            for (i = up + 1; i <= 5; ++i) {
                if (!state[i - 1]) continue;
                System.out.println("alive message send from process" + i +
"to process" + up);
                break;
            }
        }
    }

    public static void down(int down) {
        if (!state[down - 1]) {
            System.out.println("process " + down + "is already dowm.");
        } else {
            Bully.state[down - 1] = false;
        }
    }

    public static void mess(int mess) {
        if (state[mess - 1]) {
            if (state[4]) {
                System.out.println("0K");
            } else if (!state[4]) {
                int i;
                System.out.println("process" + mess + "election");
                for (i = mess; i < 5; ++i) {
                    System.out.println("election send from process" + mess
+ "to process " + (i + 1));
                }
                for (i = 5; i >= mess; --i) {
                    if (!state[i - 1]) continue;
                    System.out.println("Coordinator message send from
process" + i + "to all");
                    break;
                }
            }
        } else {
            System.out.println("Proccess" + mess + "is down");
        }
    }

    public static void main(String[] args) {
        int choice;
        Scanner sc = new Scanner(System.in);
        for (int i = 0; i < 5; ++i) {
            Bully.state[i] = true;
        }
        System.out.println("5 active process are:");
        System.out.println("Process up  = p1 p2 p3 p4 p5");
        System.out.println("Process 5 is coordinator");
        do {
            System.out.println(".........");
            System.out.println("1 up a process.");
            System.out.println("2.down a process");
            System.out.println("3 send a message");
            System.out.println("4.Exit");
            choice = sc.nextInt();
            switch (choice) {
                case 1: {
                    System.out.println("bring proces up");
                    int up = sc.nextInt();
```

```
                        if (up == 5) {
                            System.out.println("process 5 is co-ordinator");
                            Bully.state[4] = true;
                            break;
                        }
                        Bully.up(up);
                        break;
                    }
                    case 2: {
                        System.out.println("bring down any process.");
                        int down = sc.nextInt();
                        Bully.down(down);
                        break;
                    }
                    case 3: {
                        System.out.println("which process will send message");
                        int mess = sc.nextInt();
                        Bully.mess(mess);
                    }
                }
            } while (choice != 4);
        }
}
```

Output:



```
5 Active Processes are: P1 P2 P3 P4 P5
Process 5 is Co-ordinator

1. Up a Process
2. Down a Process
3. Send a Message
4. Exit

Enter the Choice: 2
Bring Down any Process: 5

1. Up a Process
2. Down a Process
3. Send a Message
4. Exit

Enter the Choice: 3
Which Process will send Message: 3
Process 3 Election
Election Message sent from Process 3 to Process 4
Election Message sent from Process 3 to Process 5
Co-ordinator Message sent from Process 4 to All Processes

1. Up a Process
2. Down a Process
3. Send a Message
4. Exit

Enter the Choice: 1
Bring Proces Up: 5
Process 5 is Co-ordinator

1. Up a Process
2. Down a Process
3. Send a Message
4. Exit

Enter the Choice:
```

Conclusion:

Election algorithms **are designed to choose a coordinator.** We have two election algorithms for two different configurations of distributed systems. **The Bully** algorithm applies to systems where every process can send a message to every other process in the system and **The Ring** algorithm applies to systems organized as a ring (logically or physically). In this algorithm we assume that the link between the processes are unidirectional and every process can message to the process on its right only.

Submitted By: Shreya Mahajan

Roll No.: 4345

Class: B.E.I.T.

Staff Name: Ms. M. A. Rane / Mr. K. V. Patil

Staff Signature:

Date: