

ASSIGNMENT No.6

TITLE:

Implement Bully and Ring algorithm for leader election

Problem Statement:

To develop Token Ring distributed algorithm for leader election.

Tools / Environment:

Java Programming Environment, JDK 1.8, Eclipse.

Related Theory:

Election Algorithm:

1. Many distributed algorithms require a process to act as a coordinator.
2. The coordinator can be any process that organizes actions of other processes.
3. A coordinator may fail.
4. How is a new coordinator chosen or elected?

Assumptions:

Each process has a unique number to distinguish them. Processes know each other's process number.

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 **COORDINATOR** 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.

Ring Algorithm:

Initiation:

1. When 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, 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: Message comes back to initiator. In the queue the **process with maximum ID Number wins**. Initiator announces the winner by sending another message around the ring.

Designing the solution:

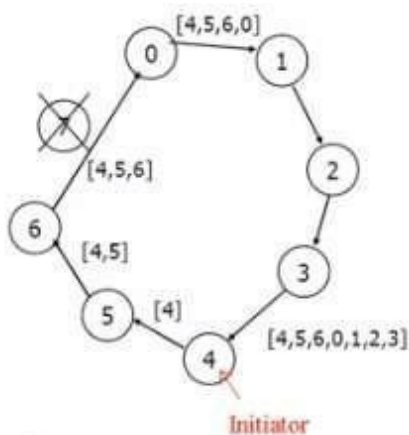
A. For Ring Algorithm

Initiation:

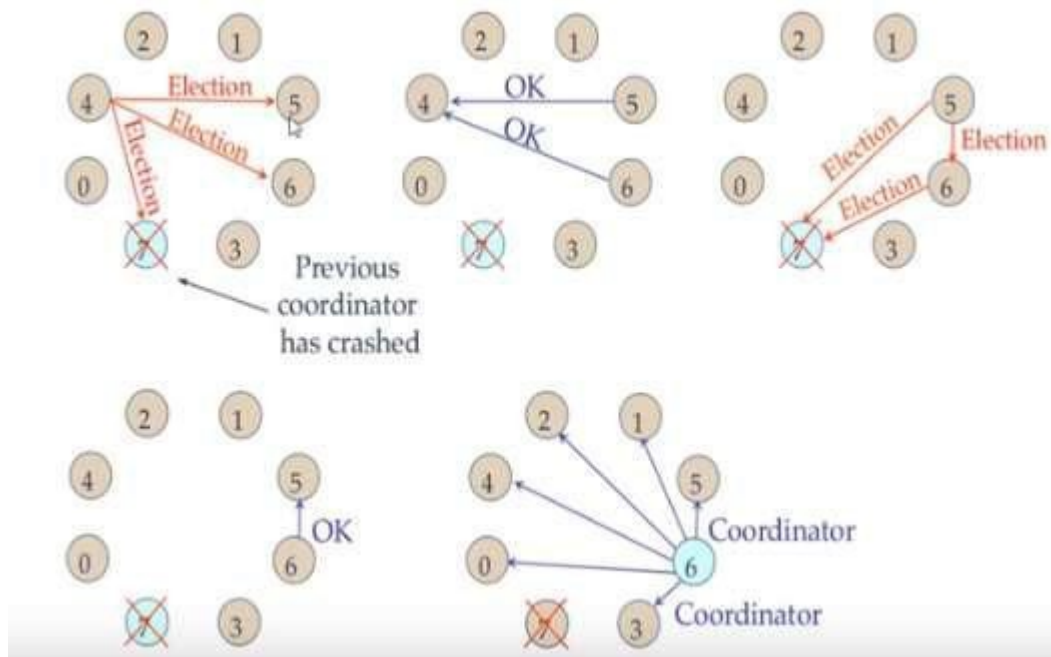
1. Consider the Process 4 understands that Process 7 is not responding.
2. Process 4 initiates the Election by sending "ELECTION" message to its successor (or next alive process) with its ID.

Leader Election:

3. Messages come back to initiator. Here the initiator is 4.
4. Initiator announces the winner by sending another message around the ring. Here the process with highest process ID is 6. The initiator will announce that Process 6 is Coordinator.



B. For Bully Algorithm:

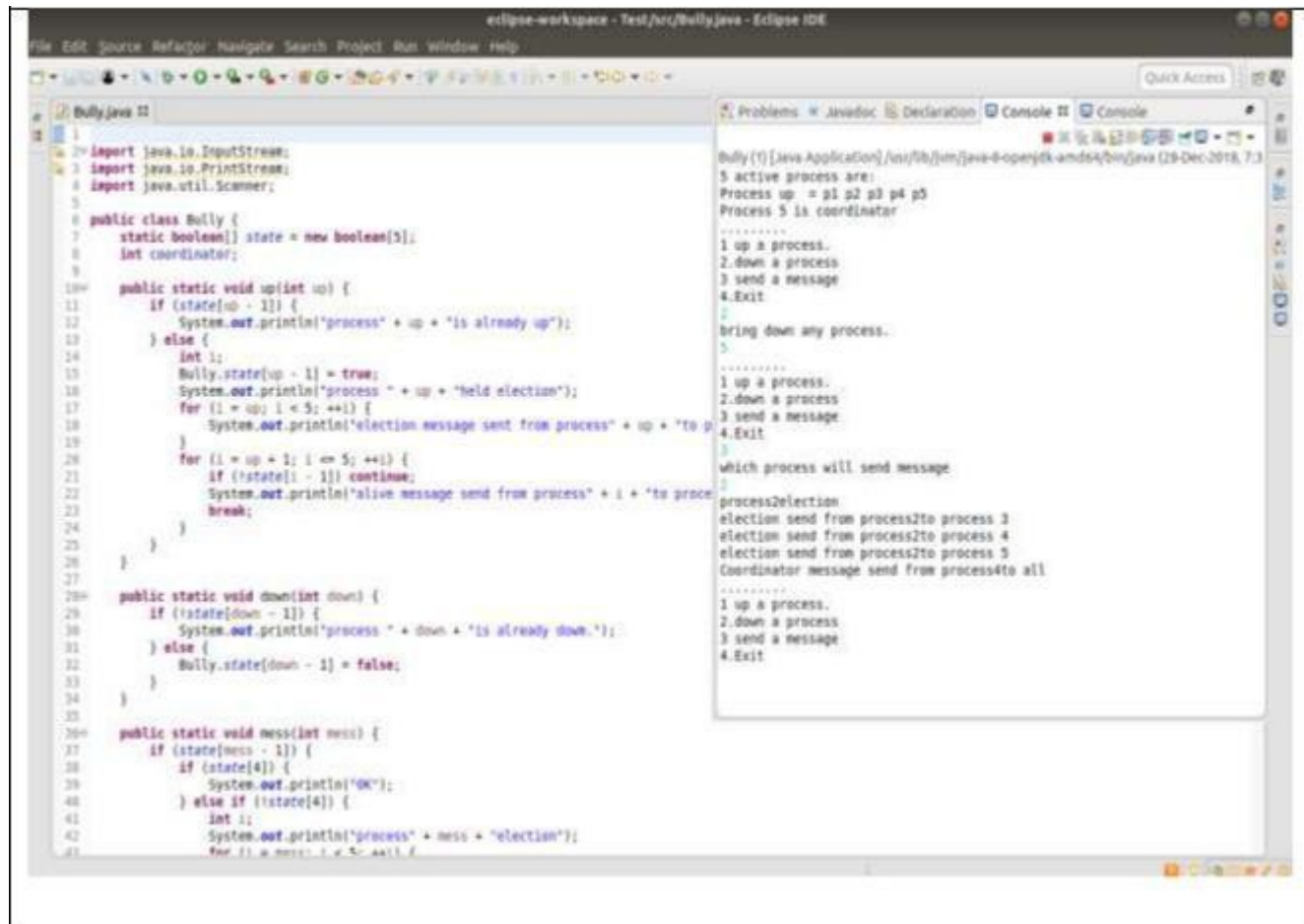


Implementing the solution:

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.Exit
7. Ask for initializing election process.
8. These inputs will be used by Ring Algorithm.

Writing the source code: Bully.java



The screenshot displays the Eclipse IDE interface. The main editor window shows the source code of `Bully.java`. The code implements a Bully algorithm for process coordination. It includes imports for `java.io.InputStream`, `java.io.PrintStream`, and `java.util.Scanner`. The `Bully` class has a static `boolean[] state` and a static `int coordinator`. It defines three main methods: `up(int up)`, `down(int down)`, and `mess(int mess)`. The `up` method handles process up events, including election messages and state updates. The `down` method handles process down events. The `mess` method handles messages from the coordinator. The console window on the right shows the output of the program, including the initial state of the processes and the sequence of messages sent during the election process.

```
1 import java.io.InputStream;
2 import java.io.PrintStream;
3 import java.util.Scanner;
4
5 public class Bully {
6     static boolean[] state = new boolean[5];
7     int coordinator;
8
9     public static void up(int up) {
10         if (state[up - 1]) {
11             System.out.println("process " + up + " is already up");
12         } else {
13             int i;
14             Bully.state[up - 1] = true;
15             System.out.println("process " + up + " held election");
16             for (i = up; i <= 5; ++i) {
17                 System.out.println("election message sent from process " + up + " to process " + i);
18             }
19             for (i = up + 1; i <= 5; ++i) {
20                 if (!state[i - 1]) continue;
21                 System.out.println("alive message sent from process " + i + " to process " + up);
22                 break;
23             }
24         }
25     }
26
27     public static void down(int down) {
28         if (!state[down - 1]) {
29             System.out.println("process " + down + " is already down.");
30         } else {
31             Bully.state[down - 1] = false;
32         }
33     }
34
35     public static void mess(int mess) {
36         if (state[mess - 1]) {
37             if (state[4]) {
38                 System.out.println("OK");
39             } else if (!state[4]) {
40                 int i;
41                 System.out.println("process " + mess + " election");
42                 for (i = mess; i <= 5; ++i) {
43                     System.out.println("election message sent from process " + mess + " to process " + i);
44                 }
45             }
46         }
47     }
48 }
```

Console Output:

```
Bully(1) [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (29-Dec-2018, 7:3)
5 active process are:
Process up = p1 p2 p3 p4 p5
Process 5 is coordinator
.....
1 up a process.
2 down a process
3 send a message
4.Exit
.....
bring down any process.
5.
.....
1 up a process.
2 down a process
3 send a message
4.Exit
.....
which process will send message
.....
process2election
election send from process2to process 3
election send from process2to process 4
election send from process2to process 5
Coordinator message send from process4to all
.....
1 up a process.
2 down a process
3 send a message
4.Exit
```

The screenshot shows the Eclipse IDE with the file 'Ring.java' open. The code implements a Ring election algorithm. It starts by importing 'java.util.Scanner'. The 'main' method takes an array of arguments. It initializes a scanner, prompts the user for the number of processes, and then for the IDs of each process. It creates an array of 'Process' objects (labeled as 'proc' in the code) and initializes them. Then, it sorts the processes based on their IDs. The console output shows the program running, with prompts for the number of processes (8) and their IDs (5, 1, 6, 2, 8). It then shows the process 8 selecting as co-ordinator, followed by a list of messages sent between processes (e.g., 'Process 8 send message to 5', 'Process 5 send message to 6', etc.), and finally, the program terminating.

```
1 import java.util.Scanner;
2
3 public class Ring {
4
5     public static void main(String[] args) {
6
7         // TODO Auto-generated method stub
8
9         int temp, i, j;
10        char str[] = new char[10];
11        Process proc[] = new Process[10];
12
13        // object initialisation
14        for (i = 0; i < proc.length; i++)
15            proc[i] = new Process();
16
17        // scanner used for getting input from console
18        Scanner in = new Scanner(System.in);
19        System.out.println("Enter the number of process : ");
20        int num = in.nextInt();
21
22        // getting input from users
23        for (i = 0; i < num; i++) {
24            proc[i].index = i;
25            System.out.println("Enter the id of process : ");
26            proc[i].id = in.nextInt();
27            proc[i].state = "active";
28            proc[i].f = 0;
29        }
30
31        // sorting the processes from on the basis of id
32        for (i = 0; i < num - 1; i++) {
33            for (j = 0; j < num - 1 - i; j++) {
34                if (proc[j].id > proc[j + 1].id) {
35                    temp = proc[j].id;
36                    proc[j].id = proc[j + 1].id;
37                    proc[j + 1].id = temp;
38                }
39            }
40        }
41    }
42
43 }
```

Console Output:

```
<terminated> Ring [1] [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java [28
Enter the number of process :
8
Enter the id of process :
5 1 6 2 8
process 8select as co-ordinator
1.election 2.quit
Enter the Process number who initialised election :
2
Process 8 send message to 5
Process 5 send message to 6
Process 6 send message to 8
process 8select as co-ordinator
1.election 2.quit
Program terminated ...
```

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.
- The output is associated in the above section.

Conclusion:

Election algorithms **are designed to choose a coordinator**. We have two election algorithms for two different configurations of distributed system. **The Bully** algorithm applies to system 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 process are unidirectional and every process can message to the process on its right only.