Experiment 5 –Election Algorithm

Learning Objective: Student should be able to develop a program for Election Algorithm

Tools : Java/Python

Theory:

Several distributed algorithms require that there be a coordinator process in the entire system that performs some type of coordination activity needed for the smooth running of other processes in the system. Two examples of such coordinator processes encountered

1. The coordinator in the centralized algorithm for mutual exclusion.
2. The central coordinator in the centralized deadlock detection algorithm.

Since all other processes in the system have to interact with the coordinator, they all must unanimously agree on who the coordinator is. Furthermore, if the coordinator process fails due to the failure of the site on which it is located, a new coordinator process must be elected to take up the job of the failed coordinator. Election algorithms are meant for electing a coordinator process from among the currently running processes in such a manner that at any instance of time there is a single coordinator for all processes in the system.

Election algorithms are based on the following assumptions:

1. Each process in the system has a unique priority number.
2. Whenever an election is held, the process having the highest priority number among thecurrently active processes is elected as the coordinator.
3. On recovery, a failed process can take appropriate actions to rejoin the set of active processes.

Therefore, whenever initiated, an election algorithm basically finds out which of the currently active processes has the highest priority number and then informs this to all other active processes.

The Bully Algorithm

As a first example, consider the bully algorithm . When any process notices that the coordinator is no longer responding to requests, it initiates an election. A process, P, holds an election as follows:

1. P sends an ELECTION message to all processes with higher numbers.
2. If no one responds, P wins the election and becomes coordinator.
3. If one of the higher-ups answers, it takes over. P's job is done.

At any moment, a process can get an ELECTION message from one of its lower-numbered colleagues. When such a message arrives, the receiver sends an OK message back to the sender to indicate that he is alive and will take over. The receiver then holds an election, unless it is already holding one. Eventually, all processes give up but one, and that one is the new coordinator. It announces its victory by sending all processes a message telling them that starting immediately it is the new coordinator

If a process that was previously down comes back up, it holds an election. If it happens to be the highest-numbered process currently running, it will win the election and take over the coordinator's job. Thus the biggest guy in town always wins, hence the name "bully algorithm."

In Fig. 6-20 we see an example of how the bully algorithm works. The group consists of eight processes, numbered from 0 to 7. Previously process 7 was the coordinator, but it has just crashed. Process 4 is the first one to notice this, so it sends ELECTION messages to all the processes higher than it, namely 5, 6, and 7. as shown in Fig. 6-20(a). Processes 5 and 6 both respond with OK, as shown in Fig. 6-20(b). Upon getting the first of these responses, 4 knows that its job is over. It knows that one of these bigwigs will take over and become coordinator. It just sits back and waits to see who the winner will be (although at this point it can make a pretty good guess).

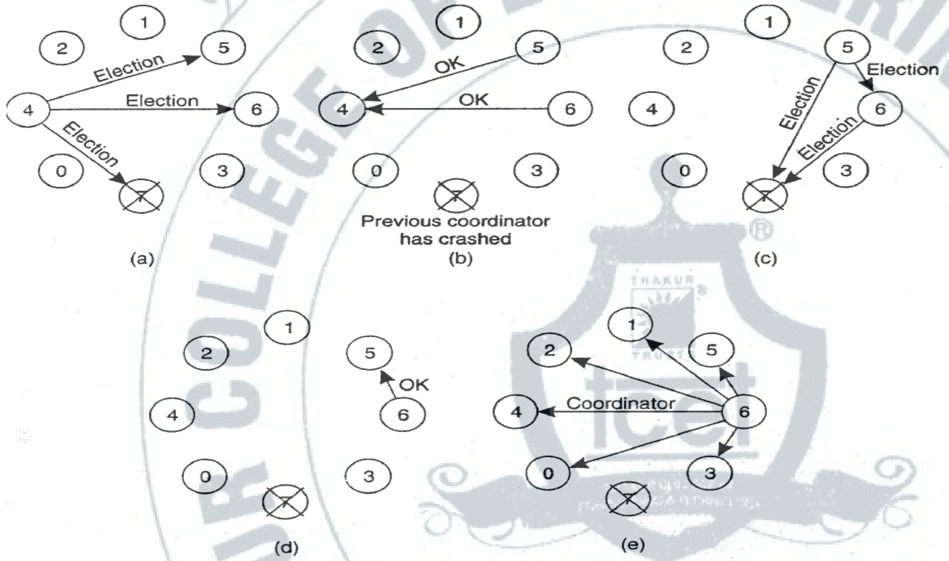


Figure 6-20. The bully election algorithm. (a) Process 4 holds an election. (b) Processes 5 and 6 respond, telling 4 to stop. (c) Now 5 and 6 each hold an election. (d) Process 6 tells 5 to stop. (e) Process 6 wins and tells everyone.

In Fig. 6-20(c), both 5 and 6 hold elections, each one only sending messages to those processes higher than itself. In Fig. 6-20(d) process 6 tells 5 that it will take over. At this point 6 knows that 7 is dead and that it (6) is the winner. If there is state information to be collected from disk or elsewhere to pick up where the old coordinator left off, 6 must now do what is needed. When it is ready to takeover, 6 announces this by sending a COORDINATOR message to all running processes.When 4 gets this message, it can now continue with the operation it was trying to do when it was discovered that 7 was dead, but using 6 as the coordinator this time. In this way the failure of 7 is handled and the work can continue.If process 7 is ever restarted, it will just send an others a COORDINATOR message and bully them into submission.

A Ring Algorithm

Another election algorithm is based on the use of a ring. Unlike some ring algorithms, this one does not use a token. We assume that the processes are physically or logically ordered, so that each process knows who its successor is. When any process notices that the coordinator is not functioning, it builds an ELECTION message containing its own process number and sends the message to its successor.

If the successor is down, the sender skips over the successor and goes to the next member along the ring. or the one after that, until a running process is located. At each step along the way, the sender adds its own process number to the list in the message effectively making itself a candidate to be elected as coordinator.

Eventually, the message gets back to the process that started it all. That process recognizes this event when it receives an incoming message containing its own process number. At that point, the message type is changed to COORDINATOR and circulated once again, this time to inform everyone else who the coordinator is (the list member with the highest number) and who the members of the new ring are. When this message has circulated once, it is removed and everyone goes back to work.

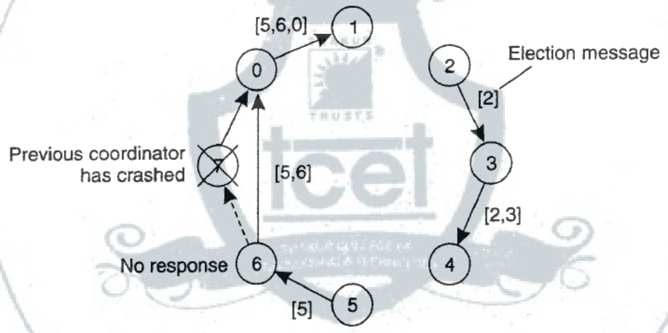


Figure 6-21. Election algorithm using a ring.

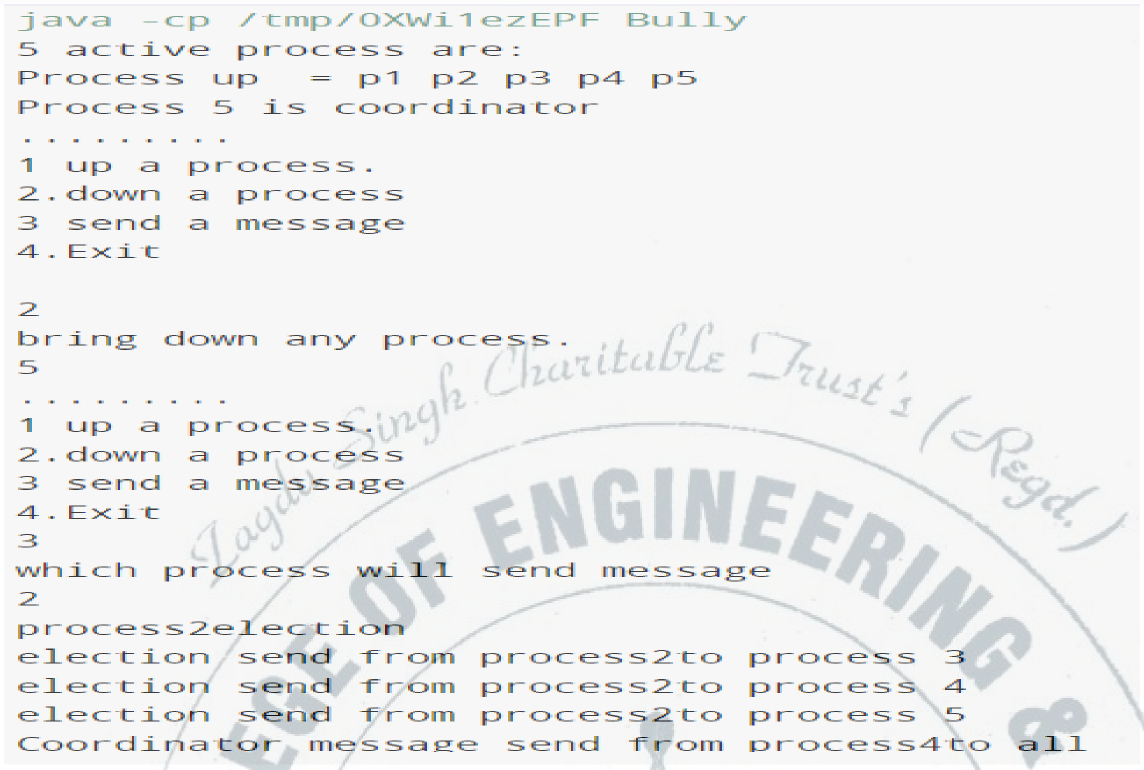
Result and Discussion:

BULLY ALGORITHM:

```
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 {
```

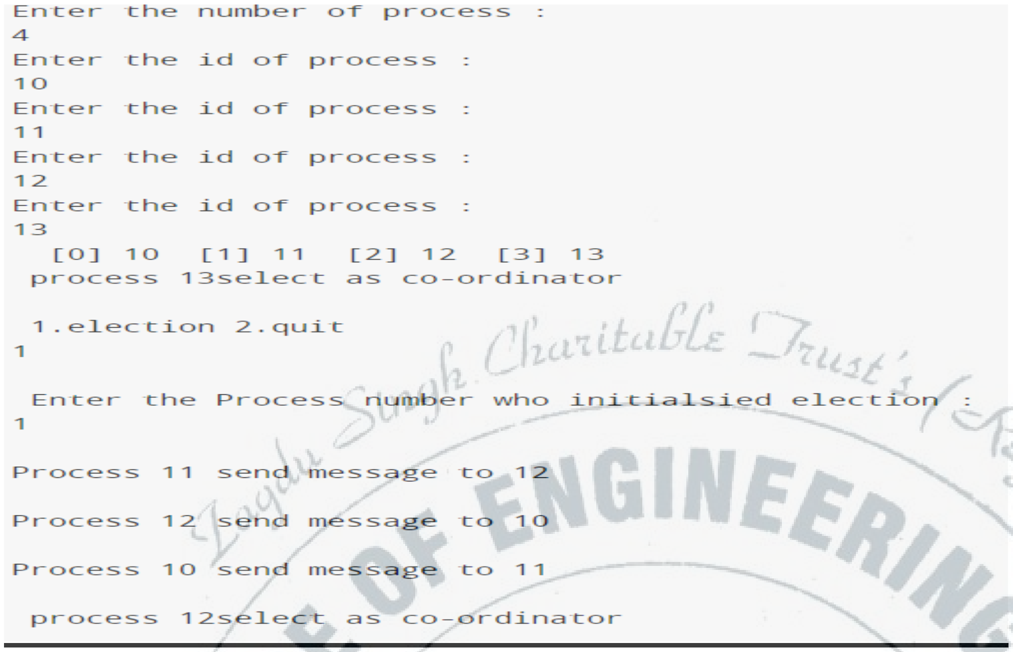
```
  Bully.state[down - 1] = false;
        }
    }

    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("Prccess" + 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:


```
java -cp /tmp/OXWi1ezEPF Bully
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

2
bring down any process.
5
.........
1 up a process.
2.down a process
3 send a message
4.Exit
3
which process will send message
2
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
```

....
RING ALGORITHM:

```
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();
          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 }
```



```
Enter the number of process :
4
Enter the id of process :
10
Enter the id of process :
11
Enter the id of process :
12
Enter the id of process :
13
  [0] 10   [1] 11   [2] 12   [3] 13
 process 13select as co-ordinator

 1.election 2.quit
1

 Enter the Process number who initialsied election :
1

Process 11 send message to 12

Process 12 send message to 10

Process 10 send message to 11

 process 12select as co-ordinator
```

Learning Outcomes: The student should have the ability to
LO1: Describe the Election Algorithms.

LO2: Write a Program to Demonstrate Election Algorithms.

Course Outcomes: Upon completion of the course students will be able to understand Election Algorithms

for Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completionof Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				