Experiment 5–Election Algorithm

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

Tools : Java

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 the currently 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).

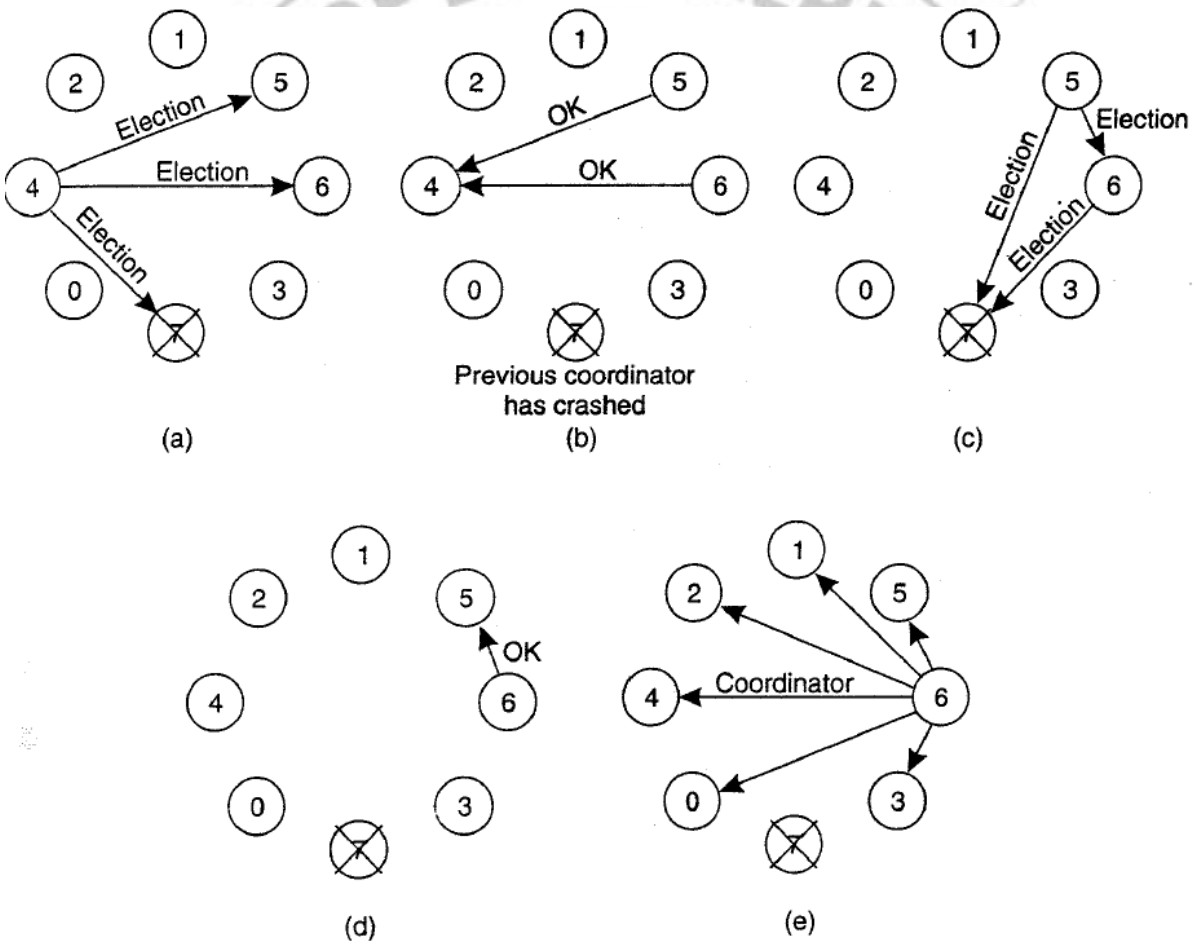

Figure 6-20. The bully election algorithm. (a) Process 4 holds an election. (b) Processes 5 and 6 respond, telling 4 to stop. (c) Now 5 and 6 each hold an election. (d) Process 6 tells 5 to stop. (e) Process 6 wins and tells everyone.

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.

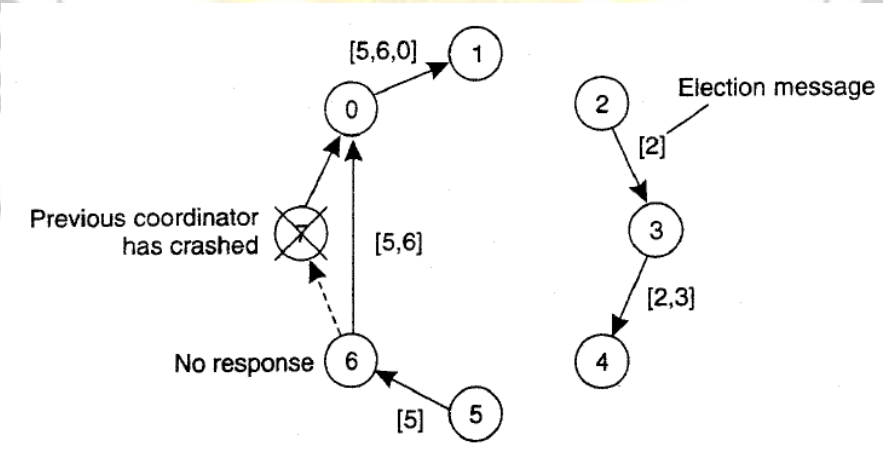

Figure 6-21. Election algorithm using a ring.

Result and Discussion: (Bully Algorithm)

Code:

```
import java.util.ArrayList;
import java.util.List;

class Process {
    private int id;
    private int priority;
    private boolean coordinator;

    public Process(int id, int priority) {
        this.id = id;
        this.priority = priority;
        this.coordinator = false;
    }

    public int getId() {
        return id;
    }

    public int getPriority() {
        return priority;
    }

    public boolean isCoordinator() {
        return coordinator;
    }

    public void setCoordinator(boolean coordinator) {
        this.coordinator = coordinator;
    }

    public void startElection(List<Process> processes) {
        System.out.println("Process " + id + " is starting election.");

        for (Process p : processes) {
            if (p.getPriority() > priority) {
                System.out.println("Process " + id + " sends election message to Process " +
p.getId());
                p.receiveElection();
            }
        }
    }

    public void receiveElection() {
```

```
        System.out.println("Process " + id + " receives election message.");
    }

    public void declareVictory(List<Process> processes) {
        Process newCoordinator = this;
        for (Process p : processes) {
            if (p.getPriority() > newCoordinator.getPriority()) {
                newCoordinator = p;
            }
        }
        for(Process p : processes){
            if(p!= newCoordinator){
                p.setCoordinator(false);
            }
        }
        newCoordinator.setCoordinator(true);
        System.out.println("Process " + newCoordinator.getId() + " is the new coordinator.");
    }

}

public class BullyAlgorithm {
    public static void main(String[] args) {

        List<Process> processes = new ArrayList<>();
        processes.add(new Process(1, 3));
        processes.add(new Process(2, 5));
        processes.add(new Process(3, 4));
        processes.add(new Process(4, 2));
        processes.add(new Process(5, 1));

        processes.get(0).setCoordinator(false);

        for (Process process : processes) {
            if (process.isCoordinator()) {
                System.out.println("Process " + process.getId() + " is already the coordinator.");
            } else {
                process.startElection(processes);
                break;
            }
        }

        processes.get(4).declareVictory(processes);

    }
}
```

Output:

```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   SEARCH ERROR

PS C:\Users\LAB-326\Desktop>  c:; cd 'c:\Users\LAB-326\Desktop'; & 'C:\Program Files\Java\jdk1.8.0_3
'BullyAlgorithm'
Process 1 is starting election.
Process 1 sends election message to Process 2
Process 2 receives election message.
Process 1 sends election message to Process 3
Process 3 receives election message.
Process 2 is the new coordinator.
PS C:\Users\LAB-326\Desktop>
```

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

Conclusion:

For Faculty Use

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