Name : Ajay Shitkar
Roll No : 58
Experiment No. 05
Implement Election Algorithm
Date of Performance: 15/03/24
Date of Submission: 18/03/24

Aim: To Implement Election Algorithm

Objective: Develop a program to implement Implement Election Algorithm

Theory:

Election Algorithms:

- The coordinator election problem is to choose a process from among a group of processes on different processors in a distributed system to act as the central coordinator.
- An election algorithm is an algorithm for solving the coordinator election problem. By the nature of the coordinator election problem, any election algorithm must be a distributed algorithm.

(a) Bully Algorithm

Background: any process Pi sends a message to the current coordinator; if no response in T time units, Pi tries to elect itself as leader. Details follow:

Algorithm for process Pi that detected the lack of coordinator

1. Process Pi sends an "Election" message to every process with higher priority.
2. If no other process responds, process Pi starts the coordinator code running and sends a message to all processes with lower priorities saying "Elected Pi"
3. Else, Pi waits for T' time units to hear from the new coordinator, and if there is no response à start from step (l) again.

Algorithm for other processes (also called Pi)

If Pi is not the coordinator then Pi may receive either of these messages from Pj

if Pi sends "Elected Pj"; [this message is only received if i < j]

Pi updates its records to say that Pj is the coordinator.

Else if Pj sends "election" message (i > j)

Pi sends a response to Pj saying it is alive

Pi starts an election.

(b) Election In A Ring Ring Algorithm.

-assume that processes form a ring: each process only sends messages to the next process in the ring

- Active list: its info on all other active processes

- assumption: message continues around the ring even if a process along the way has crashed.
Background: any process Pi sends a message to the current coordinator; if no response in T time units, Pi initiates an election

1. initialize active list to empty.
2. Send an "Elect(i)" message to the right. + add it to the active list.

If a process receives an "Elect(j)" message

(a) this is the first message sent or seen initialize its active list to [i,j]; send "Elect(i)" + send "Elect(j)"

(b) if i !=j, add i to active list + forward "Elect(j)" message to active list

(c) otherwise (i = j), so process i has a complete set of active processes in its active list. => choose highest process ID + send "Elected (x)" message to neighbor

If a process receives "Elected(x)" message,

set coordinator to x

Example:

Suppose that we have four processes arranged in a ring: PI å P2 å P3 å P4 å PI ...
P4 is coordinator
Suppose PI + P4 crash
Suppose P2 detects that coordinator P4 is not responding
P2 sets active list to [ ]
P2 sends "Elect(2)" message to P3; P2 sets active list to [2]
P3 receives "Elect(2)"
This message is the first message seen, so P3 sets its active list to [2,3]
P3 sends "Elect(3)" towards P4 and then sends "Elect(2)" towards P4
The messages pass P4 + PI and then reach P2
P2 adds 3 to active list [2,3]
P2 forwards "Elect(3)" to P3
P2 receives the "Elect(2) message

P2 chooses P3 as the highest process in its list [2, 3] and sends an "Elected(P3)" message

P 3 receives the "Elect(3)" message

P 3 chooses P3 as the highest process in its list [2, 3] + sends an "Elected(P3)" message

Code and output:

Code :

```
import java. util.Scanner;

public class GFG {

        class Pro { int id;

                boolean

                act; Pro(int

                id)

                        this.id — id;

                        act = true;

        int TotalProcess; Pro[]

        process; public GFG() { }

        public                  void

        initialiseGFG()

                System.out.println("No of processes 5");
                 TotalProcess = 5; process = new

                 Pro[TotalProcess]; int i = O;

                 while (i < process.length) {

                        process[i] = new Pro(i);

        public void Election()
                 System.out.println("Process no '
                                                + process [FetchMaximum()].id
                                                 + " fails");
```

```
process[FetchMaximum()].act = false;

System.out.println("Election Initiated by 2");

int initializedProcess = 2;

int old — initializedProcess;

int newer = old + I ;

while (true) { if

(process[newer].act) {

System.out.println(

"Process " + process[old].id + "

pass Election(" + process[old].id

+ ") to" + process[newer].id);

old = newer;

newer = (newer + l) % TotalProcess;

if (newer — initializedProcess) {

break;

System.out.println("Process "

+ process

[FetchMaximum()].id + " becomes coordinator"); int coord

= process[FetchMaximum()].id;

old — coord; newer — (old + l)

% TotalProcess;

while (true) {

if (process[newer].act) {

System.out.println(

"Process " + process[old].id

+ " pass Coordinator(" + coord
```

```
                    + ") message to process '
                ,+ process[newer].id); old —
                    newer;
            newer = (newer + 1) % TotalProcess;
            if (newer coord) {
                    System.out.println("End Of Election ");
                    break;
public int FetchMaximum()
    int Ind = O; int maxld _ — _9999; int i = O; while
    (i < process.length) { if (process[i].act &&
    process[i].id > maxld) { maxld = process[i].id;
                    Ind — i;
    return Ind;
public static void main(String arg[])
    GFG object - new GFG();
    object.initialiseGFG();
    object.Election();
```

Output :

No of processes 5

Process no 4 fails

Election Initiated by 2

Process 2 pass Election(2) t03

Process 3 pass Election(3) too

Process O pass Election(O) tol

Process 3 becomes coordinator

Process 3 pass Coordinator(3) message to process O

Process O pass Coordinator(3) message to process I

Process 1 pass Coordinator(3) message to process 2

End Of Election

Conclusion:

Implementing an Election Algorithm is crucial for ensuring fault tolerance and leader selection in distributed systems. By employing algorithms like Bully Algorithm or Ring Algorithm, systems can dynamically elect a leader among nodes, facilitating coordination and decision-making. These algorithms prioritize scalability, fault tolerance, and efficiency, making them suitable for various distributed environments. However, careful consideration must be given to factors like network latency, node failures, and message delivery guarantees to ensure the robustness and reliability of the election process. Overall, implementing an Election Algorithm enhances the resilience and performance of distributed systems, contributing to their effectiveness in real-world applications.