Name: Shivam Pandey	
Roll No.: 28	
Experiment No. 5	
Implement Election Algorithm	
Date of Performance: 22/02/2024	
Date of Submission: 07/03/2024	

Aim: To Implement Election Algorithm.

Objective: Develop a program to 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 (1) 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 i to 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 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: P1 à P2 à P3 à P4 à P1 …
P4 is coordinator
Suppose P1 + 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 + P1 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
P3 receives the "Elect(3)" message
P3 chooses P3 as the highest process in its list [2, 3] + sends an "Elected(P3)" message

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
struct Pro {
	int id;
	bool act;
	Pro(int id) {
		this->id = id;
		act = true;
	}
};
class Elect {
	public:
	int TotalProcess;
	std::vector<Pro> process;
	Elect() {}
	void initialiseElect() {
		std::cout << "No of processes 5" << std::endl;
		TotalProcess = 5;
		process.reserve(TotalProcess);
		for (int i = 0; i < process.capacity(); i++) {
			process.emplace_back(i);
		}
	}
	void Election() {
		std::cout << "Process no " << process[FetchMaximum()].id << " fails" <<
	std::endl;
		process[FetchMaximum()].act = false;
		std::cout << "Election Initiated by 2" << std::endl;
		int initializedProcess = 2;
		int old = initializedProcess;
		int newer = old + 1;
		while (true) {
			if (process[newer].act) {
			std::cout << "Process " << process[old].id << " pass Election(" <<
		process[old].id << ") to" << process[newer].id << std::endl;
			old = newer;
```

```
            }
            newer = (newer + 1) % TotalProcess;
            if (newer == initializedProcess) {
                break;
            }
        }
        std::cout << "Process " << process[FetchMaximum()].id << " becomes coordinator"
<< std::endl;
        int coord = process[FetchMaximum()].id;
        old = coord;
        newer = (old + 1) % TotalProcess;
        while (true) {
            if (process[newer].act) {
                std::cout << "Process " << process[old].id << " pass Coordinator("
            << coord << ") message to process " << process[newer].id << std::endl;
                old = newer;
            }
            newer = (newer + 1) % TotalProcess;
            if (newer == coord) {
                std::cout << "End Of Election " << std::endl;
                break;
            }
        }
}
int FetchMaximum() {
        int Ind = 0;
        int maxId = -9999;
        for (int i = 0; i < process.size(); i++) {
                if (process[i].act && process[i].id > maxId) {
                    maxId = process[i].id;
                    Ind = i;
                }
            }
            return Ind;
        }
};
int main() {
```

```
    Elect object;
    object.initialiseElect();
    object.Election();
    return 0;
}
```

Output:

No of processes 5
Process no 4 fails
Election Initiated by 2
Process 2 pass Election(2) to3
Process 3 pass Election(3) to0
Process 0 pass Election(0) to1
Process 3 becomes coordinator
Process 3 pass Coordinator(3) message to process 0
Process 0 pass Coordinator(3) message to process 1
Process 1 pass Coordinator(3) message to process 2
End Of Election

Conclusion: The implemented Ring Algorithm effectively demonstrates the process of electing a new coordinator in a distributed system when the current coordinator fails. By initiating an election process, each process communicates with its neighbors, forming an active list of processes. Through iterative message passing and comparison of process IDs, the algorithm successfully selects the process with the highest ID as the new coordinator. This ensures the continuity of coordination within the system even amidst failures, highlighting the robustness and reliability of the distributed election algorithm.