

**Vidyavardhini’s College of Engineering & Technology**

Department of Computer Engineering Academic Year : 2024-25

|  |  |  |  |
| --- | --- | --- | --- |
| **Class:** | **BE** | **Semester:** | **VIII** |
| **Course Code:** | **CSL801** | **Course Name:** | **Distributed Computing Lab** |

|  |  |
| --- | --- |
| **Name of Student:** | **Pratima Dinkar Bombe** |
| **Roll No. :** | **07** |
| **Division:** | **-** |
| **Experiment No.:** | **05** |
| **Title of Experiment:** | **Ring Election Algorithm** |
| **Date of Submission:** | **11/02/2025** |
| **Date of Correction:** | **18/02/2025** |

Evaluation

|  |  |  |
| --- | --- | --- |
| **Performance Indicator** | **Max. Marks** | **Marks Obtained** |
| Performance | 5 |  |
| Understanding | 5 |  |
| Journal work and timely submission | 10 |  |
| Total | 20 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Performance Indicator** | **Exceed Expectations (EE)** | **Meet Expectations (ME)** | **Below Expectations (BE)** |
| Performance | 4-5 | 2-3 | 1 |
| Understanding | 4-5 | 2-3 | 1 |
| Journal work and timely submission | 8-10 | 5-8 | 1-4 |

**Checked by**

**Name of Faculty : Ms. Swati Varma**

**Signature :**

**Date :**

EXPERIMENT 5

**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 (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 and output**:

class RingProcess:

    def \_\_init\_\_(self, id):

        self.id = id

        self.coordinator = None

        self.next = None

    def start\_election(self):

        print(f"Process {self.id} starts an election.")

        active\_list = [self.id]

        self.send\_election\_message(active\_list)

    def send\_election\_message(self, active\_list):

        if self.next:

            print(f"Process {self.id} forwards election list {active\_list}.")

            self.next.receive\_election\_message(active\_list)

    def receive\_election\_message(self, active\_list):

        if self.id not in active\_list:

            active\_list.append(self.id)

            self.send\_election\_message(active\_list)

        else:

            new\_coordinator = max(active\_list)

            print(f"Process {self.id} elects {new\_coordinator} as the coordinator.")

            self.send\_coordinator\_message(new\_coordinator)

    def send\_coordinator\_message(self, new\_coordinator):

        if self.next:

            print(f"Process {self.id} informs that {new\_coordinator} is the new coordinator.")

            self.coordinator = new\_coordinator

            self.next.receive\_coordinator\_message(new\_coordinator)

    def receive\_coordinator\_message(self, new\_coordinator):

        if self.coordinator is None:

            self.coordinator = new\_coordinator

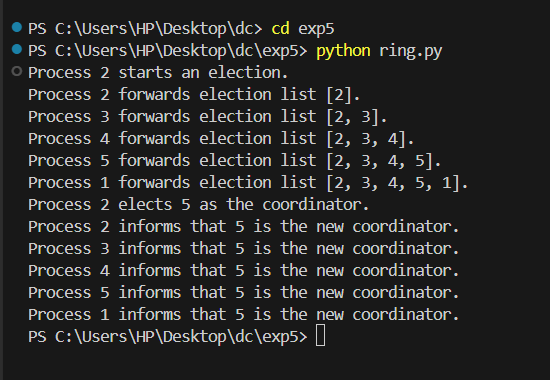
            self.send\_coordinator\_message(new\_coordinator)

processes = [RingProcess(i) for i in range(1, 6)]

for i in range(len(processes)):

    processes[i].next = processes[(i + 1) % len(processes)]

processes[1].start\_election()



**Conclusion**: In conclusion, election algorithms like the Bully Algorithm and Ring Algorithm play a crucial role in distributed systems by ensuring that a central coordinator is selected efficiently, even in the event of process failures. The Bully Algorithm is faster but relies on direct communication with higher-priority processes, whereas the Ring Algorithm is more structured and message-efficient, as it follows a circular path. Both methods help maintain system reliability and coordination, ensuring seamless operation in distributed environments.