Assignment No 6

Title: Implementation of Bully and Ring Algorithms for Leader Election in Distributed Systems

Objectives:

- To understand the importance of leader election in distributed systems.
- To learn the working principles of the **Bully** and **Ring** algorithms.
- To implement leader election mechanisms in a simulated distributed environment.
- To analyze and compare both algorithms in terms of communication complexity and efficiency.
- To simulate failure and recovery of processes and observe the dynamic leader re-election.

Problem Statement:

In distributed systems, it is often necessary to elect a leader (coordinator) to manage coordination tasks such as resource allocation, synchronization, and communication control. However, due to the decentralized nature of such systems and possible process failures, electing a reliable leader dynamically becomes a challenge.

This assignment requires students to implement two well-known leader election algorithms:

1. **Bully Algorithm** – where the process with the highest ID takes charge by sending election messages to all higher-ID processes.
2. **Ring Algorithm** – where processes are arranged in a logical ring and pass election messages around the ring until the coordinator is selected.

Students are to simulate multiple processes, detect process failures, and execute leader election accordingly.

Outcomes:

- Understanding of distributed coordination and the necessity of leader election.
- Hands-on experience in implementing and simulating leader election protocols.
- Ability to detect failures and handle dynamic process participation.
- Knowledge of message passing, process identifiers (IDs), and fault tolerance in distributed systems.
- Comparative analysis of Bully and Ring algorithms based on performance and suitability.

Theory:

1. Bully Algorithm:

The **Bully Algorithm**, proposed by Garcia-Molina, is a classic leader election algorithm used in distributed systems. It assumes that **each process has a unique ID** and that **processes**

know the IDs of all other processes. The goal is to elect the process with the **highest ID** as the leader.

Working:

- Any process can initiate the election if it notices the current leader has failed (e.g., no heartbeat or response).
- The initiating process sends an **ELECTION message** to all processes with higher IDs.
- If no process responds, the initiator becomes the coordinator and broadcasts a **COORDINATOR message** to all.
- If any higher-ID process responds with an **OK message**, it takes over the election by repeating the procedure.
- This continues until the process with the highest ID is elected.

Assumptions:

- Each process knows all other process IDs.
- Reliable message passing.
- The system allows crash and recovery.

Advantages:

- Quick termination once the highest-ID process is alive.
- Fair — the strongest (highest ID) becomes the leader.

Disadvantages:

- High message overhead in large systems.
- Requires knowledge of all nodes.

2. Ring Algorithm:

The **Ring Algorithm** arranges processes in a **logical ring topology** where each process knows only its **successor**. All communication is unidirectional (clockwise or counter-clockwise), and messages travel along the ring.

Working:

- A process that detects the coordinator failure initiates the election.
- It sends an **ELECTION message** with its own ID to its successor.
- Each process adds its own ID to the message and forwards it.
- When the message returns to the initiator, it selects the **highest ID** from the list as the new coordinator.
- Then, a **COORDINATOR message** is sent along the ring to inform all processes of the new leader.

Assumptions:

- Processes are logically arranged in a ring.

- Communication is one-way along the ring.
- Every process has a unique ID.

Advantages:

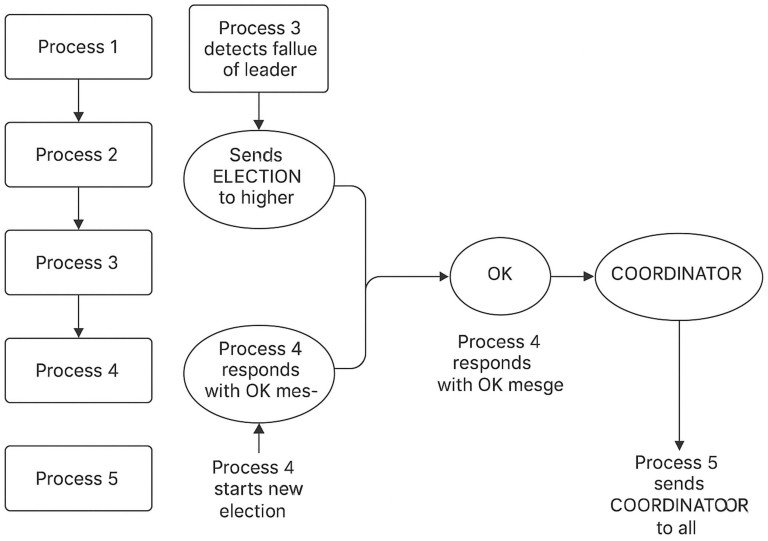
- Simple and does not require knowledge of all other processes.
- Equal opportunity for all processes.

Disadvantages:

- Slower than Bully in large networks.
- Message complexity is O(n), even when the highest ID is near the initiator.

Bully Algorithm – Working Diagram

Scenario: Process 3 initiates election; Process 5 is the highest ID.



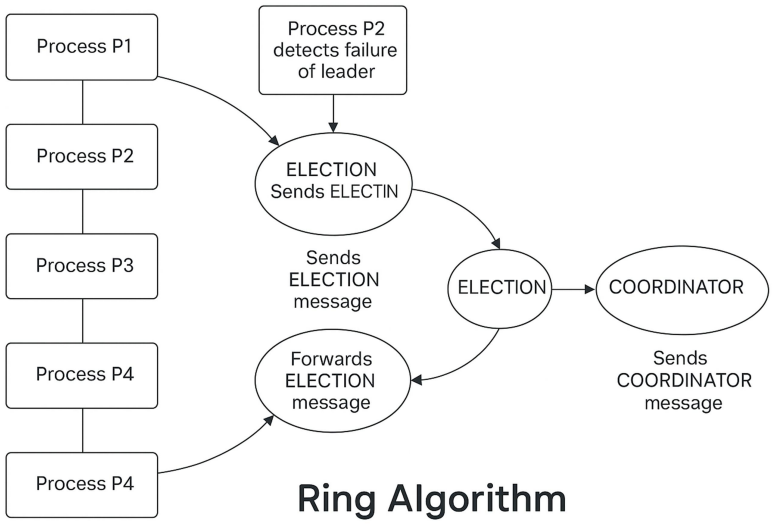
Message Flow Summary:

- Process 3 → ELECTION → 4, 5
- Process 4 → OK → 3
- Process 5 → OK → 3
- Process 5 (highest ID) declares itself as coordinator
- Process 5 → COORDINATOR → all

Ring Algorithm – Working Diagram

Scenario: Processes arranged in a ring; Process 2 starts election.

```
Logical Ring:
```




```
Step-by-Step Election:
1. Process 2 sends ELECTION with its ID to Process 3.
2. Each process appends its ID and forwards the message.
3. Message returns to Process 2.
4. Process 2 identifies highest ID (say, P5).
5. COORDINATOR message is passed through the ring.
```

Message Flow Summary:

- Process 2 → ELECTION (2) → 3 → 4 → 5 → 1 → back to 2
- Process 2 sees max ID = 5
- Process 2 → COORDINATOR (5) → 3 → 4 → 5 → 1 → back to 2

Software Requirements:

- **Programming Language:** Python / Java / C++
- **Python Libraries (if used):**
  - `socket` or `multiprocessing` for inter-process communication simulation
  - `threading` for concurrent behavior
  - `random` for simulating process failure or delay
  - `time` for simulation timing
- **IDE:** VS Code, PyCharm, Eclipse, etc.
- **Operating System:** Windows / Linux / macOS

Hardware Requirements:

- **Processor:** Intel i3 or better
- **RAM:** 4 GB or more
- **Storage:** Minimum 100 MB

- **Network (Optional for real deployment):** LAN or virtual network (if simulating on multiple devices)
- **Number of Systems:** Minimum 1 system (with multithreading/multiprocessing), or multiple for distributed deployment.

Conclusion: