**LAB 7**

**Aim**: To implement Election Algorithms

**Lab Outcome**: Implement techniques for Election Algorithms.

**Theory**:

In distributed systems, election algorithms are used to elect a leader or coordinator

node among a group of nodes. The leader node is responsible for coordinating the

activities of other nodes, managing resources, and making decisions on behalf of the group.

There are various election algorithms used in distributed systems, and the choice of

algorithm depends on the requirements of the system, such as fault-tolerance, scalability,

and availability.

One commonly used election algorithm is the Bully algorithm, which is a centralized

algorithm. In this algorithm, the node with the highest priority becomes the leader, and

If the leader fails, the next highest priority node takes over.

Another algorithm is the Ring algorithm, which is a decentralized algorithm. In this

algorithm, nodes are arranged in a ring, and each node sends a message to its neighbor

indicating its willingness to become the leader. The node with the highest priority becomes

the leader.

A third algorithm is the Paxos algorithm, which is fault-tolerant and can handle network

failures. In this algorithm, nodes propose a value to be chosen as the leader, and if

the majority of nodes accept the proposal, the value is chosen as the leader.

These election algorithms ensure that a leader is elected in a fair and efficient manner,

and they provide fault-tolerance and availability in distributed systems.

Bully Algorithm

The bully Algorithm proposed by Garcia-Molina follows the following algorithm

● When a 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 the coordinator

3. If one of the higher-up’s answers, it takes over. P's job is done

● If a process can get an ELECTION message from one of its lower-numbered

colleagues.

The 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 does not give up 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 previously went down came 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."

Code :

**import java.util.ArrayList;**

**import java.util.Scanner;**

**class Process {**

**int processId;**

**boolean coordinator;**

**boolean active;**

**int priority;**

**public Process(int processId, boolean active, int priority) {**

**this.processId = processId;**

**this.coordinator = false;**

**this.active = active;**

**this.priority = priority;**

**}**

**public void startElection(ArrayList<Process> processes) {**

**for (Process p : processes) {**

**if (p.processId > this.processId && p.active) {**

**System.out.println("Election message is sent from " + this.processId + " to " + p.processId);**

**p.handleElection();**

**}**

**}**

**System.out.println("Final coordinator is " + this.processId);**

**this.coordinator = true;**

**announceCoordinator(processes);**

**}**

**public void handleElection() {**

**// Dummy method for handling election message**

**System.out.println("Process " + this.processId + " receives ELECTION message and sends OK message back");**

**}**

**public void announceCoordinator(ArrayList<Process> processes) {**

**for (Process p : processes) {**

**if (p.processId != this.processId && p.active) {**

**System.out.println("Process " + p.processId + " receives coordinator announcement from Process " + this.processId);**

**}**

**}**

**}**

**}**

**public class BullyAlgorithm {**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**// Get the number of processes from the user**

**System.out.print("Enter the number of processes: ");**

**int numProcesses = scanner.nextInt();**

**ArrayList<Process> processes = new ArrayList<>();**

**// Initialize processes**

**for (int i = 1; i <= numProcesses; i++) {**

**System.out.println("For process " + i + ":");**

**System.out.print("Status (1 for active, 0 for inactive): ");**

**int status = scanner.nextInt();**

**System.out.print("Priority: ");**

**int priority = scanner.nextInt();**

**processes.add(new Process(i, status == 1, priority));**

**}**

**// Find the process to initiate the election**

**int initiatingProcess;**

**do {**

**System.out.print("Which process will initiate the election? (Enter process id): ");**

**initiatingProcess = scanner.nextInt();**

**} while (initiatingProcess < 1 || initiatingProcess > numProcesses);**

**// Start election**

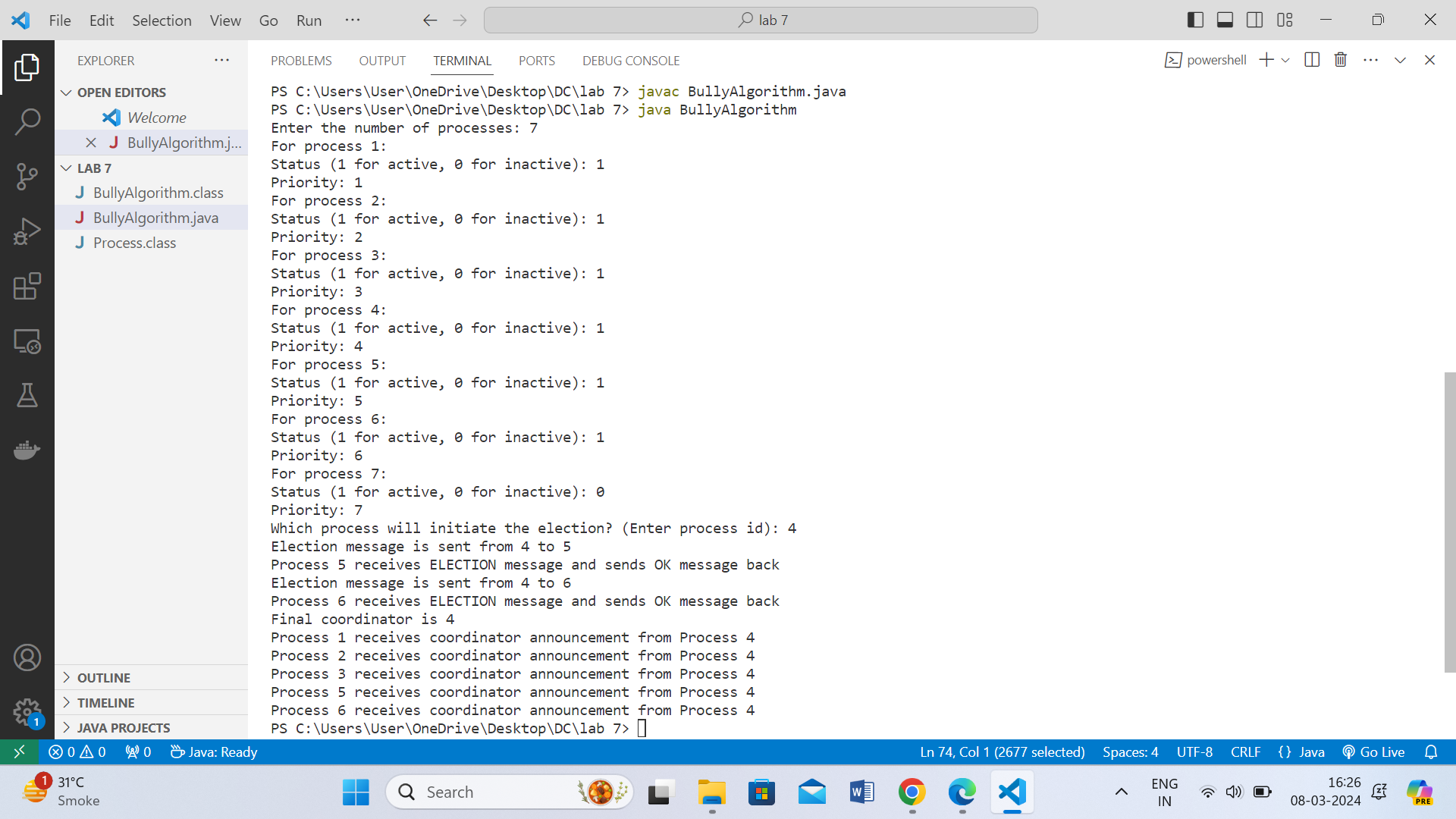
**processes.get(initiatingProcess - 1).startElection(processes);**

**scanner.close();**

**}**

**}**

Output :



**Conclusions** :

1. Understood and learnt the concept of Election Algorithms and significance of a

coordinator in a system.

2. Implemented the Bully Election Algorithm using Java.

3. The coordinator is elected using the Bully algorithm by considering the highest

process id in the network.

**Post Lab Questions:**

**1. What is the role of a coordinator in Distributed systems?**

In distributed systems, a coordinator plays a crucial role in coordinating and managing various activities and resources within the system. The role of a coordinator can vary depending on the specific application or system architecture, but some common roles include:

1. Resource Allocation: The coordinator may be responsible for allocating resources such as computing resources, storage, or network bandwidth among different processes or nodes in the system. This ensures efficient utilization of resources and prevents conflicts or contention.

2. Task Scheduling: In systems where tasks need to be executed or processed, the coordinator can schedule and distribute tasks among different nodes or processes based on various criteria such as workload, priority, or availability of resources. This helps in load balancing and optimizing system performance.

3. Transaction Management: In distributed databases or transactional systems, the coordinator oversees the execution of transactions across multiple nodes or databases. It ensures that transactions are executed atomically, consistently, and with durability guarantees, even in the presence of failures.

4. Consistency Maintenance: In distributed file systems or databases, the coordinator may enforce consistency among replicas or copies of data distributed across different nodes. It ensures that all replicas are updated and synchronized correctly, maintaining data integrity and coherence.

5. Fault Tolerance and Recovery: The coordinator may also play a role in fault tolerance mechanisms, such as leader election or replication management. It helps in detecting failures, initiating recovery procedures, and maintaining system availability and reliability.

6. Communication and Message Routing: In distributed messaging systems or communication networks, the coordinator can facilitate message routing, ensure reliable delivery, and manage communication protocols. It helps in establishing and maintaining communication channels among different nodes.

7. System Monitoring and Control: The coordinator may monitor the health and performance of the distributed system, collect system metrics and statistics, and make decisions or adjustments to optimize system behavior. It helps in maintaining system stability and responsiveness.

Overall, the coordinator acts as a central authority or controller in distributed systems, orchestrating various activities and ensuring that the system functions correctly, efficiently, and reliably.

**2. Compare Bully and Ring algorithms.**

| **Aspect** | **Bully Algorithm** | **Ring Algorithm** |
| --- | --- | --- |
| Basic Idea | Initiates an election when the coordinator fails. | Uses a token passing mechanism for coordination. |
| Process Participation | Every process can initiate an election. | Only one process starts the token. |
| Election Process | When coordinator fails, the highest-ranking active process starts the election. | Election starts with a predefined process, usually with the lowest ID. |
| Message Passing | Processes send election messages to higher-ranked processes. | Token is passed from one process to the next in a circular manner. |
| Coordination | Process with the highest ID becomes the coordinator. | Token holder coordinates activities. |
| Scalability | May not scale well for large numbers of processes due to frequent elections. | Scales well for large numbers of processes. |
| Failure Handling | Can handle coordinator failure and elect a new coordinator. | Not inherently designed for handling failures. |
| Complexity | Moderately complex due to election initiation and handling. | Relatively simple, primarily involving token passing. |
| Message Overhead | May result in higher message overhead due to multiple election messages. | Lower message overhead due to token passing. |
| Use Cases | Suitable for scenarios where a single coordinator needs to be elected in a distributed system. | Suitable for scenarios where processes need to communicate and coordinate in a cyclic manner. |