Distributed Computing

Experiment – 5

Program to demonstrate Election Algorithm.	
Learning Objective:	To learn Election Algorithm.
Learning Outcome:	Implement techniques for Election Algorithms.
Course Outcome:	**CSL801.3**
Program Outcome:	(**PO**1) **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems. (**PO**2) **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences. **(PO**3) **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations. **(PO5) Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
Bloom's Taxonomy Level:	Analysis, Create
Theory:	**Election Algorithm -** The Election Algorithm is a fundamental concept in distributed computing that is used to elect a leader or coordinator among a group of distributed nodes or processes. This leader is responsible for coordinating activities, making decisions, or performing certain tasks within the distributed system. There are various algorithms designed for this purpose, each with its own advantages, disadvantages, and types. Here's an overview:

Types of Election Algorithms:

1. **Bully Algorithm:**

In this algorithm, the process with the highest priority (the "bully") takes over as the leader.
If a lower-priority process detects that the leader has failed, it initiates an election by sending messages to higher-priority processes.
The process with the highest priority that responds becomes the new leader.

2. **Ring Algorithm:**

In the ring-based algorithm, processes are arranged in a logical ring structure.
Each process communicates only with its immediate neighbors in the ring to determine the leader.
The election message circulates around the ring until it reaches the process with the highest ID, which becomes the leader.

3. **Token Ring Algorithm:**

Similar to the ring algorithm, but with the addition of a token that circulates among the processes.
The process holding the token is the leader, and if it fails, the token is passed to the next process in the ring.

4. **Randomized Algorithms:**

These algorithms introduce randomness into the election process to ensure fairness and prevent certain nodes from consistently being elected as leaders.
Randomized algorithms typically involve processes generating random numbers or making probabilistic decisions during the election.

Advantages of Election Algorithms:

- **Fault Tolerance:**

Election algorithms ensure that a new leader can be elected quickly in the event of the current leader failing or becoming unavailable.
This fault tolerance helps maintain system stability and continuity of operations.

- **Scalability:**

These algorithms can scale to large distributed systems with many nodes without significantly increasing complexity or overhead.

- **Decentralization:**

Election algorithms enable distributed systems to operate without relying on a central point of control.
This decentralization enhances resilience and reduces the risk of single points of failure.

Disadvantages of Election Algorithms:

- **Overhead:**

Election algorithms require communication and coordination among processes, which can introduce overhead and consume network resources.
In some cases, frequent elections can degrade system performance.

- **Complexity:**

Implementing and managing election algorithms can be complex, especially in large-scale distributed systems with dynamic node membership.

- **Potential for Inefficiency:**

Depending on the algorithm and system conditions, election processes may take time to complete, leading to delays in leadership transitions.

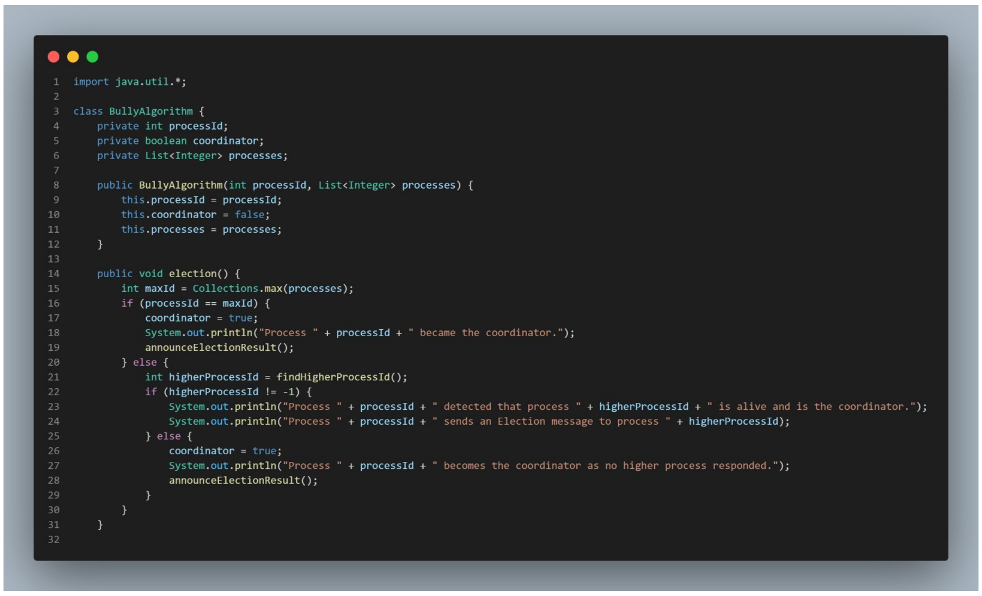
- **Election Anomalies:**

Some algorithms may exhibit anomalies such as the "split-brain" problem, where multiple leaders are elected concurrently due to network partitions or other issues.

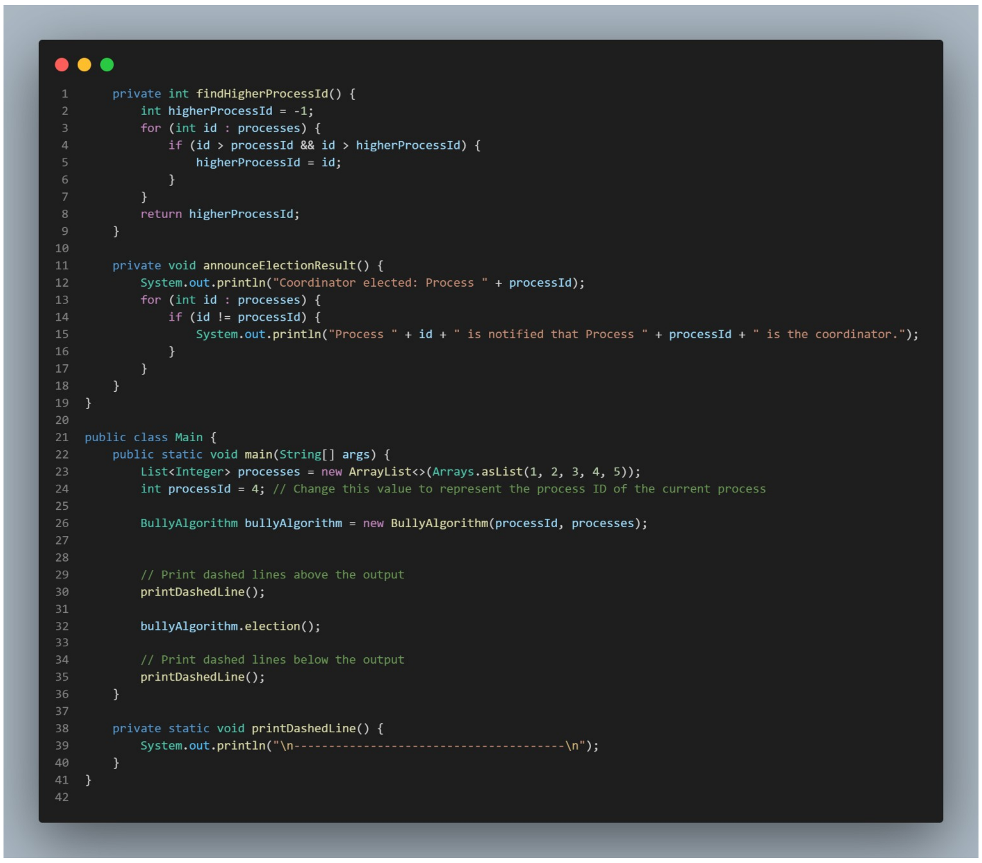
Algorithm :	Let's assume that P is a process that sends a message to the coordinator. 1. It will assume that the coordinator process is failed when it doesn't receive any response from the coordinator within the time interval T. 2. An election message will be sent to all the active processes by process P along with the highest priority number. 3. If it will not receive any response within the time interval T, the current process P elects itself as a coordinator. 4. After selecting itself as a coordinator, it again sends a message that process P is elected as their new coordinator to all the processes having lower priority. 5. If process P will receive any response from another process Q within time T: A) It again waits for time T to receive another response, i.e., it has been elected as coordinator from process Q. B) If it doesn't receive any response within time T, it is assumed to have failed, and the algorithm is restarted.

Outcome :	Program -
	
	`import java.util.*;`
	
	`class BullyAlgorithm {`
	`    private int processId;`
	`    private boolean coordinator;`
	`    private List<Integer> processes;`
	
	`    public BullyAlgorithm(int processId, List<Integer> processes) {`
	`        this.processId = processId;`
	`        this.coordinator = false;`
	`        this.processes = processes;`
	`    }`

```
    public void election() {
        int maxId = Collections.max(processes);
        if (processId == maxId) {
            coordinator = true;
            System.out.println("Process " + processId + " became the coordinator.");
            announceElectionResult();
        } else {
            int higherProcessId = findHigherProcessId();
            if (higherProcessId != -1) {
                System.out.println("Process " + processId + " detected that process " + higherProcessId + " is alive and is the coordinator.");
                System.out.println("Process " + processId + " sends an Election message to process " + higherProcessId);
            } else {
                coordinator = true;
                System.out.println("Process " + processId + " becomes the coordinator as no higher process responded.");
                announceElectionResult();
            }
        }
    }


    private int findHigherProcessId() {
        int higherProcessId = -1;
        for (int id : processes) {
            if (id > processId && id > higherProcessId) {
                higherProcessId = id;
            }
        }
        return higherProcessId;
    }

    private void announceElectionResult() {
        System.out.println("Coordinator elected: Process " +
processId);
        for (int id : processes) {
            if (id != processId) {
                System.out.println("Process " + id + " is notified that
Process " + processId + " is the coordinator.");
            }
        }
    }
}

public class Main {
    public static void main(String[] args) {
        List<Integer> processes = new ArrayList<>(Arrays.asList(1, 2,
3, 4, 5));
        int processId = 4; // Change this value to represent the process
ID of the current process

        BullyAlgorithm bullyAlgorithm = new BullyAlgorithm(processId,
processes);

        // Print dashed lines above the output

        printDashedLine();



        bullyAlgorithm.election();



        // Print dashed lines below the output

        printDashedLine();

    }



    private static void printDashedLine() {

        System.out.println("\n----------------------------------------\n");

    }

}
```

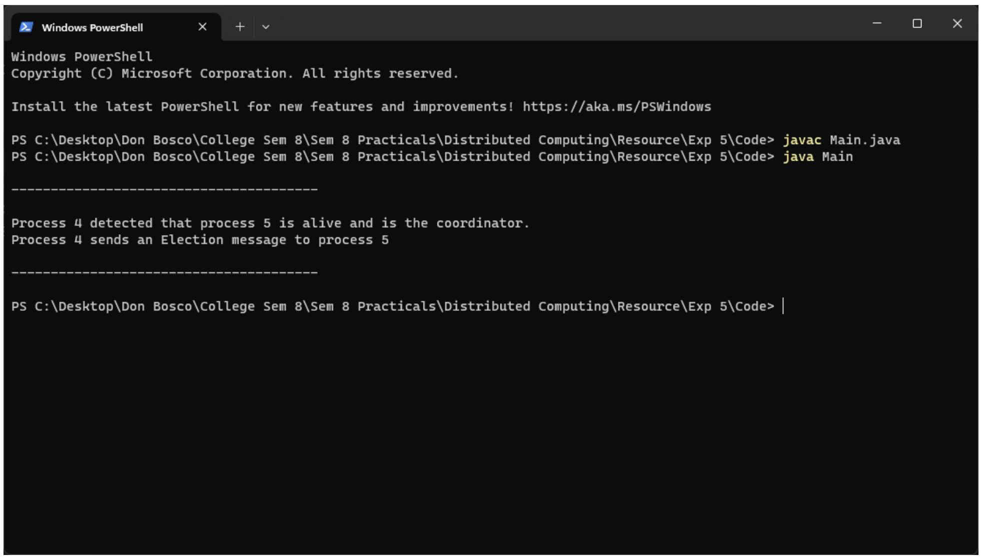


```java
import java.util.*;

class BullyAlgorithm {
    private int processId;
    private boolean coordinator;
    private List<Integer> processes;

    public BullyAlgorithm(int processId, List<Integer> processes) {
        this.processId = processId;
        this.coordinator = false;
        this.processes = processes;
    }

    public void election() {
        int maxId = Collections.max(processes);
        if (processId == maxId) {
            coordinator = true;
            System.out.println("Process " + processId + " became the coordinator.");
            announceElectionResult();
        } else {
            int higherProcessId = findHigherProcessId();
            if (higherProcessId != -1) {
                System.out.println("Process " + processId + " detected that process " + higherProcessId + " is alive and is the coordinator.");
                System.out.println("Process " + processId + " sends an Election message to process " + higherProcessId);
            } else {
                coordinator = true;
                System.out.println("Process " + processId + " becomes the coordinator as no higher process responded.");
                announceElectionResult();
            }
        }
    }

```



```java
    private int findHigherProcessId() {
        int higherProcessId = -1;
        for (int id : processes) {
            if (id > processId && id > higherProcessId) {
                higherProcessId = id;
            }
        }
        return higherProcessId;
    }

    private void announceElectionResult() {
        System.out.println("Coordinator elected: Process " + processId);
        for (int id : processes) {
            if (id != processId) {
                System.out.println("Process " + id + " is notified that Process " + processId + " is the coordinator.");
            }
        }
    }
}

public class Main {
    public static void main(String[] args) {
        List<Integer> processes = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5));
        int processId = 4; // Change this value to represent the process ID of the current process

        BullyAlgorithm bullyAlgorithm = new BullyAlgorithm(processId, processes);


        // Print dashed lines above the output
        printDashedLine();

        bullyAlgorithm.election();

        // Print dashed lines below the output
        printDashedLine();
    }

    private static void printDashedLine() {
        System.out.println("\n---------------------------------------\n");
    }
}
```

Output -



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Resource\Exp 5\Code> javac Main.java
PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Resource\Exp 5\Code> java Main

---------------------------------------

Process 4 detected that process 5 is alive and is the coordinator.
Process 4 sends an Election message to process 5

---------------------------------------

PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Resource\Exp 5\Code> |
```

Conclusion :	Thus, we studied Bully Algorithm through the experimentation and implementation of the Election Algorithm. We've gained insights into distributed systems' coordination mechanisms, where nodes autonomously elect a coordinator to manage the network, ensuring smooth operation and fault tolerance.

References:	https://www.javatpoint.com/bully-algorithm-in-java https://www.geeksforgeeks.org/election-algorithm-and-distributed-processing/ https://www.geeksforgeeks.org/bully-algorithm-in-distributed-system/

Rubrics for Assessment

Timely Submission	Late	Submitted just after deadline	On time Submission
	0 points	***1 points***	***2 points***
Understanding	Student is confused about the concept	Students has justifiably understood the concept	Students is very clear about the concepts
	0 points	***2 points***	***3 points***
Performance	Students has not performed the Experiment	Student has performed with help	Student has independently performed the experiment
	0 points	***2 points***	***3 points***
Development	Student struggles to write code	Student can write code with the requirement stated	Student can write exceptional code with his own ideas
	0 points	***2 points***	***3 points***
Overall Discipline in Lab	No Discipline maintained	Follows Rules	Sincerely performs the practical is always on time
	0 points	***1 points***	***4 points***