BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Empowered Autonomous Institute Affiliated to Mumbai University)
Department Of Computer Engineering

Name	Mayur Solankar, Manish Jadhav, Vishesh Savani, Shreyansh Salvi
UID	2023301018, 2023301005, 2022300100, 2022300091
Subject	Distributed Computing
Experiment No.	8
Project title	Social Media System
Problem Statement	Implementation of Election algorithm
Objectives	Provides the dynamic selection of a leader node, enabling coordinated task management and efficient load balancing in the distributed environment of a social media platform.
Theory	**ELECTION ALGORITHM** **What is an Election Algorithm?** Election algorithms choose a process from a group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected by another processor. Election algorithm basically determines where a new copy of the coordinator should be restarted. Election algorithm assumes that every active process in the system has a unique priority number. The process with highest priority will be chosen as a new coordinator. There are 2 types of election algorithm : 1) Bully algorithm 2) Ring algorithm Here, we used the Bully algorithm. **-: Bully algorithm :-** There can be three types of messages that processes exchange with each other in the bully algorithm: 1. Election message: Sent to announce election. 2. OK (Alive) message: Responds to the Election message. 3. Coordinator (Victory) message: Sent by the winner of the election to announce the new coordinator. **Pros of the Bully algorithm:** ● Simple: The bully algorithm is easy to understand and implement.

	● Effective in small networks: The bully algorithm has low overhead in smaller distributed systems. ● Fault-tolerant: The bully algorithm can elect a new leader if the current leader fails. **Cons of the Bully algorithm:** ● Inefficient in large networks: The bully algorithm can introduce message overhead and delays in larger distributed systems. ● Risk of starvation: Lower-ranked nodes may never become leaders in some cases. ● Initialization challenges: The bully algorithm requires accurate Process rankings, which can be difficult to achieve in practice. ● Lack of preemption: The bully algorithm is non-preemptive meaning that the current leader cannot be preempted by a higher-ranked Process
Code	

```
class Node:
    def __init__(self, id):
        self.id = id
        self.is_leader = False
        self.is_active = True  # Node starts as active

    def down(self):
        """Simulate the node going down."""
        self.is_active = False
        print(f"Node {self.id} is down.")

    def up(self):
        """Simulate the node coming back up."""
        self.is_active = True
        print(f"Node {self.id} is up.")
def start_election(nodes, initiator):
    """Initiate the election process."""
    print(f"Node {initiator.id} is initiating the election.")

    # Find all higher ID nodes that are still active
    higher_nodes = [node for node in nodes if node.id > initiator.id and
node.is_active]

    if not higher_nodes:
        # No higher nodes, this node becomes the leader
        initiator.is_leader = True
        print(f"Node {initiator.id} becomes the leader.")
```

```
    else:
        # Send election messages to all higher nodes
        print(f"Node {initiator.id} sends election messages to higher nodes:
{[node.id for node in higher_nodes]}")
        for node in higher_nodes:
            start_election(nodes, node)
def check_leader(nodes):
    """Check if there is an active leader."""
    for node in nodes:
        if node.is_leader and node.is_active:
            return node
    return None
def simulate_failure(nodes):
    """Simulate the leader's failure and start a new election."""
    leader = check_leader(nodes)
    if leader:
        print(f"Leader {leader.id} has failed!")
        leader.down()  # Simulate leader failure
        # Start new election from the first available active node
        for node in nodes:
            if node.is_active:
                start_election(nodes, node)
                break
    else:
        print("No active leader to fail.")
def post_message(nodes, message):
    """Simulate posting a message handled by the leader."""
    leader = check_leader(nodes)
    if leader and leader.is_active:
        print(f"Leader {leader.id} is handling the post: '{message}'")
    else:
        print("Leader is down. Starting a new election...")
        simulate_failure(nodes)
        # After the new election, retry posting the message
        post_message(nodes, message)
if __name__ == "__main__":
    # Step 1: Create a list of nodes (simulated servers)
    nodes = [Node(id) for id in range(1, 6)]  # 5 nodes with IDs 1 to 5

    # Step 2: Initially, initiate election to select a leader
    print("Starting initial election...")
    start_election(nodes, initiator=nodes[0])

    # Step 3: Post a message through the leader
```

```
    print("\nPosting a message to the system...")
    post_message(nodes, "Hello from the leader!")

    # Step 4: Simulate leader failure and election
    print("\nSimulating leader failure...")
    simulate_failure(nodes)

    # Step 5: Post another message after leader failure
    print("\nPosting another message to the system...")
    post_message(nodes, "Message after leader failure!")
```

OUTPUT

```
PS C:\Users\vishe\OneDrive\Desktop\DC Codes> cd .\Election_Algo\
PS C:\Users\vishe\OneDrive\Desktop\DC Codes\Election_Algo> python experiment_9.py
Starting initial election...
Node 1 is initiating the election.
Node 1 sends election messages to higher nodes: [2, 3, 4, 5]
Node 2 is initiating the election.
Node 2 sends election messages to higher nodes: [3, 4, 5]
Node 3 is initiating the election.
Node 3 sends election messages to higher nodes: [4, 5]
Node 4 is initiating the election.
Node 4 sends election messages to higher nodes: [5]
Node 5 is initiating the election.
Node 5 becomes the leader.
Node 5 is initiating the election.
Node 5 becomes the leader.
Node 4 is initiating the election.
Node 4 sends election messages to higher nodes: [5]
Node 5 is initiating the election.
Node 5 becomes the leader.
Node 5 is initiating the election.
Node 5 becomes the leader.
Node 3 is initiating the election.
Node 3 sends election messages to higher nodes: [4, 5]
Node 4 is initiating the election.
Node 4 sends election messages to higher nodes: [5]
Node 5 is initiating the election.
Node 5 becomes the leader.
Node 5 is initiating the election.
Node 5 becomes the leader.
Node 4 is initiating the election.
Node 4 sends election messages to higher nodes: [5]
Node 5 is initiating the election.
Node 5 becomes the leader.
```

```
Node 5 is initiating the election.
Node 5 becomes the leader.

Posting a message to the system...
Leader 5 is handling the post: 'Hello from the leader!'

Simulating leader failure...
Leader 5 has failed!
Node 5 is down.
Node 1 is initiating the election.
Node 1 sends election messages to higher nodes: [2, 3, 4]
Node 2 is initiating the election.
Node 2 sends election messages to higher nodes: [3, 4]
Node 3 is initiating the election.
Node 3 sends election messages to higher nodes: [4]
Node 4 is initiating the election.
Node 4 becomes the leader.
Node 4 is initiating the election.
Node 4 becomes the leader.
Node 3 is initiating the election.
Node 3 sends election messages to higher nodes: [4]
Node 4 is initiating the election.
Node 4 becomes the leader.
Node 4 is initiating the election.
Node 4 becomes the leader.

Posting another message to the system...
Leader 4 is handling the post: 'Message after leader failure!'
PS C:\Users\vishe\OneDrive\Desktop\DC Codes\Election_Algo>
```

Conclusion:

By completing this experiment, we got to know about how the Bully Election Algorithm effectively manages leader selection in distributed systems, ensuring seamless task handling even when nodes fail. This showcases the importance of fault tolerance and system resilience in real-world applications like social media platforms.