BHARATIYA VIDYA BHAVAN’S

SARDAR PATEL INSTITUTE OF TECHNOLOGY

(Empowered Autonomous Institute Affiliated to University of Mumbai)

[Knowledge is Nectar]

Department of Computer Engineering

Course - Distributed Computing (DC)

UID	2023800080 2023800071 2023800079
Name	Jeet Patel Soham Padalia Arya Patel
Class and Batch	TE Computer Science and Engineering - Batch B1
Date	31/10/25
Lab #	9
Aim	To implement **Bully** and **Ring Election Algorithms** in a distributed system to elect a new coordinator when the current one fails.
Objective	*1. To understand the concept of election algorithms in distributed systems.* *2. To study how Bully and Ring algorithms select a new coordinator.* *3. To simulate process coordination and failure recovery in a distributed environment.* *4. To analyze message passing and process priority in leader election.* *5. To compare the speed, fairness, and efficiency of both algorithms.*
Theory	*In distributed systems, multiple processes work together without a central controller. When the process acting as **a coordinator (leader)** fails, an **election algorithm** is used to select a new one.* ***1. Bully Algorithm:*** ○ *The process with the highest ID becomes the coordinator.* ○ *When a process detects the leader’s failure, it sends an ELECTION message to all higher-ID processes.* ○ *If no higher process replies, it declares itself as the coordinator and informs others.* ***2. Ring Algorithm:*** ○ *All processes are arranged in a **logical ring**.* ○ *When a coordinator fails, the process noticing it sends an ELECTION message around the ring.* ○ *Each process adds its ID, and the one with the highest ID is chosen as the new coordinator once the message completes the ring.* *These algorithms ensure **fault tolerance, coordination,** and **continuity** in distributed systems.*
Procedure:	***Bully:-*** *1. Assign a unique ID to each process (higher ID = higher priority).* *2. Start all processes and elect an initial coordinator (highest ID).*

3. When a process detects the coordinator has failed, it sends an ELECTION message to all processes with higher IDs.
4. If no higher process responds, the initiating process becomes the new coordinator and broadcasts a COORDINATOR message to all nodes.
5. If any higher process replies with OK, that higher process takes over the election.
6. Observe the output to confirm that the highest active process becomes the new coordinator.

Procedure for Ring Election Algorithm:

1. Arrange all processes in a logical ring and assign each a unique ID.
2. When a process detects that the coordinator has failed, it initiates an ELECTION message containing its ID and sends it to the next process in the ring.
3. Each process adds its own ID to the message and forwards it to the next node.
4. When the message completes the ring, the process with the highest ID is chosen as the new coordinator.
5. The coordinator sends a COORDINATOR message around the ring to inform all nodes.
6. Verify that all nodes recognize the same coordinator after election.

Implementation / Code 1

1A:
bully.json:

```
[
  { "pid": 1, "host": "127.0.0.1", "port": 6001 },
  { "pid": 2, "host": "127.0.0.1", "port": 6002 },
  { "pid": 3, "host": "127.0.0.1", "port": 6003 }
]
```

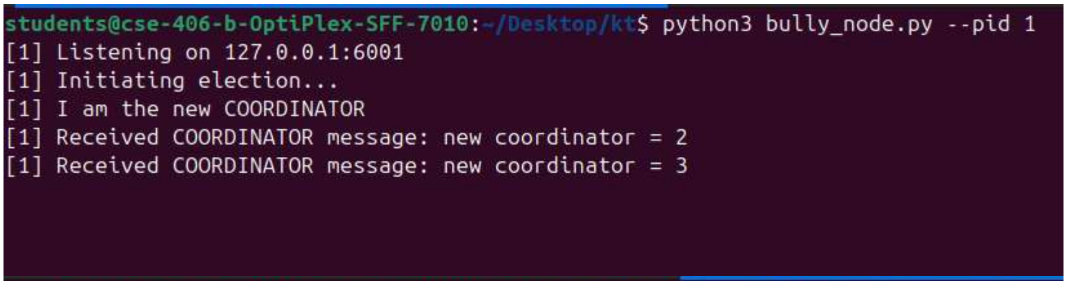
bully_node.py

```
import argparse
import json
import socket
import threading
import time

# Load nodes from json
with open('nodes.json') as f:
    nodes = json.load(f)

def get_node(pid):
    return next(n for n in nodes if n["pid"] == pid)

class BullyNode:
    def __init__(self, pid):
        self.pid = pid
        self.node_info = get_node(pid)
        self.coordinator = None
        self.alive = True
        self.lock = threading.Lock()
    def start(self):
        # Start server thread
        threading.Thread(target=self.server_thread, daemon=True).start()
        time.sleep(2)  # Wait for all nodes to start
        self.start_election()

        # Periodically check coordinator alive
        while self.alive:
            time.sleep(5)
            if self.coordinator is None or not self.ping(self.coordinator):
                print(f"[{self.pid}] Coordinator {self.coordinator} not responding -> starting
election")
                self.start_election()

    def server_thread(self):
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.bind((self.node_info['host'], self.node_info['port']))
        s.listen(5)
        print(f"[{self.pid}] Listening on {self.node_info['host']}:{self.node_info['port']}")

        while self.alive:
            conn, addr = s.accept()
            threading.Thread(target=self.handle_connection, args=(conn,),
daemon=True).start()

    def handle_connection(self, conn):
        msg = conn.recv(1024).decode()
        if msg.startswith("ELECTION"):
            sender_pid = int(msg.split()[1])
            print(f"[{self.pid}] Received ELECTION from {sender_pid}")
            # Reply OK if self.pid > sender_pid
            if self.pid > sender_pid:
                conn.sendall("OK".encode())
                # Start election
                self.start_election()
            else:
                # Ignore
                pass
        elif msg.startswith("COORDINATOR"):
            new_coord = int(msg.split()[1])
            with self.lock:
                self.coordinator = new_coord
            print(f"[{self.pid}] Received COORDINATOR message: new coordinator =
{new_coord}")
        conn.close()
```

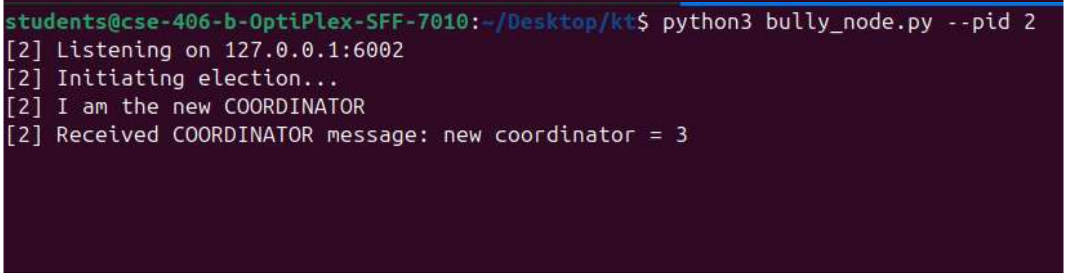
```
def start_election(self):
    print(f"[{self.pid}] Initiating election...")
    higher_nodes = [n for n in nodes if n['pid'] > self.pid]
    received_ok = False
    for node in higher_nodes:
        if self.send_message(node, f"ELECTION {self.pid}"):
            print(f"[{self.pid}] Received OK from {node['pid']}")
            received_ok = True

    if not received_ok:
        # I am the coordinator
        with self.lock:
            self.coordinator = self.pid
        print(f"[{self.pid}] I am the new COORDINATOR")
        # Broadcast coordinator message
        for node in nodes:
            if node['pid'] != self.pid:
                self.send_message(node, f"COORDINATOR {self.pid}")

def send_message(self, node, message):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(2)
        s.connect((node['host'], node['port']))
        s.sendall(message.encode())
        if message.startswith("ELECTION"):
            response = s.recv(1024).decode()
            if response == "OK":
                s.close()
                return True
        s.close()
    except:
        return False
    return False

def ping(self, pid):
    node = get_node(pid)
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(2)
        s.connect((node['host'], node['port']))
        s.close()
        return True
    except:
        return False
```

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--pid", type=int, required=True)
    args = parser.parse_args()

    node = BullyNode(args.pid)
    node.start()
```

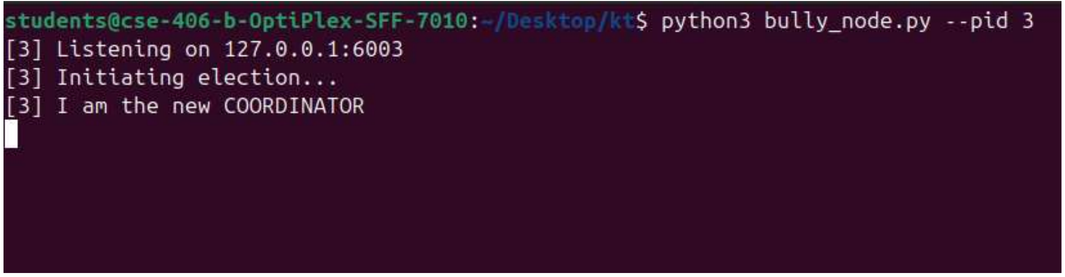
Output 1



```
students@cse-406-b-OptiPlex-SFF-7010:~/Desktop/kt$ python3 bully_node.py --pid 1
[1] Listening on 127.0.0.1:6001
[1] Initiating election...
[1] I am the new COORDINATOR
[1] Received COORDINATOR message: new coordinator = 2
[1] Received COORDINATOR message: new coordinator = 3
```



```
students@cse-406-b-OptiPlex-SFF-7010:~/Desktop/kt$ python3 bully_node.py --pid 2
[2] Listening on 127.0.0.1:6002
[2] Initiating election...
[2] I am the new COORDINATOR
[2] Received COORDINATOR message: new coordinator = 3
```



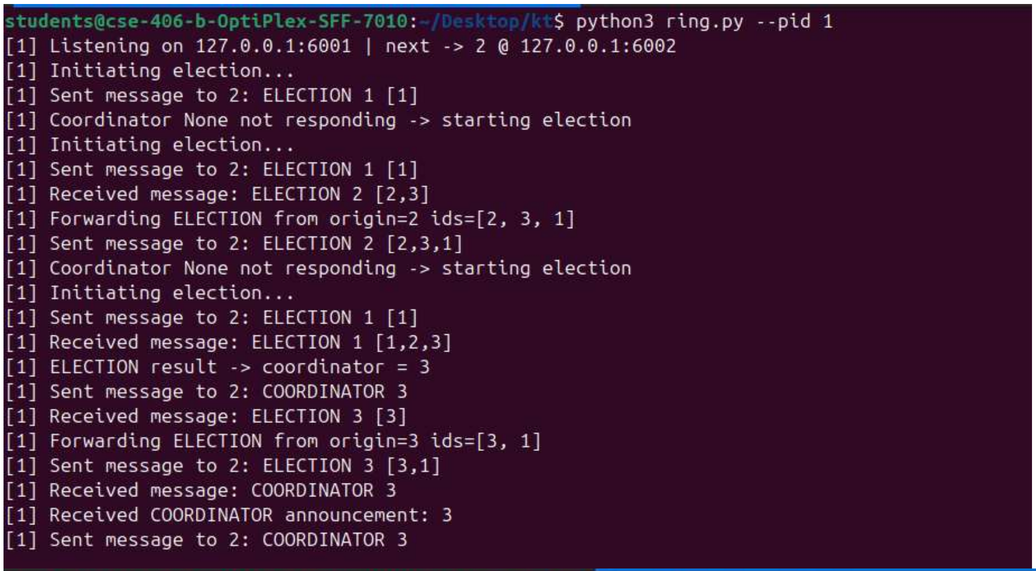
```
students@cse-406-b-OptiPlex-SFF-7010:~/Desktop/kt$ python3 bully_node.py --pid 3
[3] Listening on 127.0.0.1:6003
[3] Initiating election...
[3] I am the new COORDINATOR
```

Implementation/ Code 2

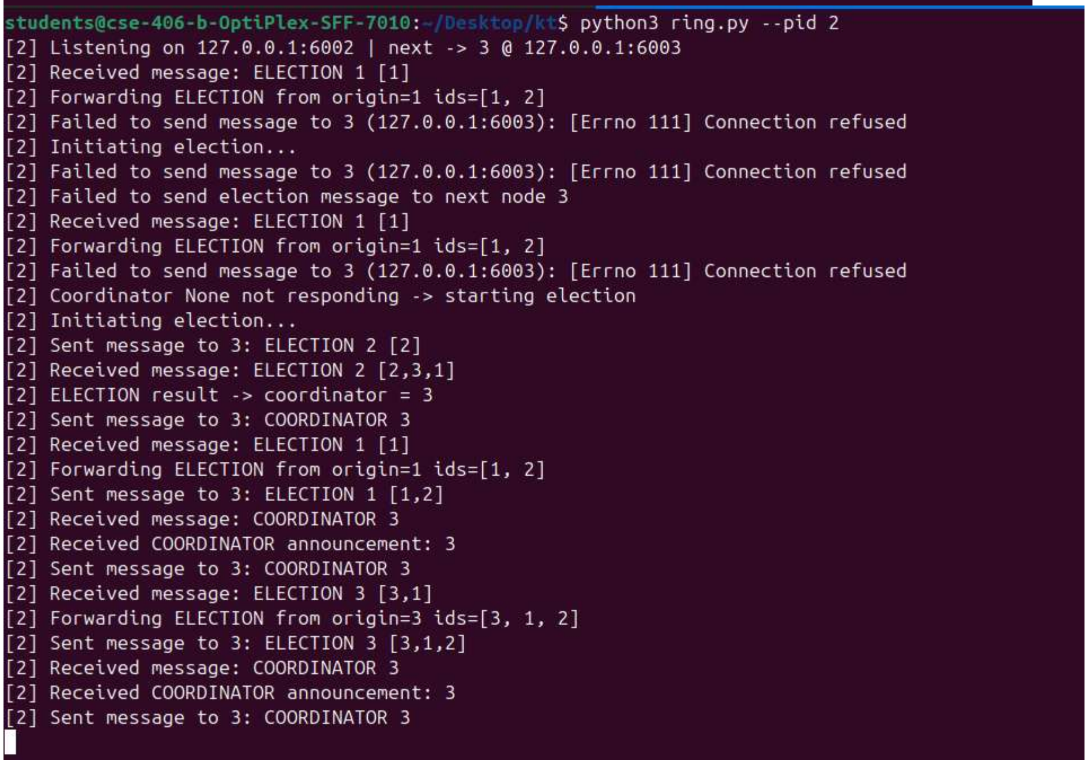
1B:

```
import argparse
import json
import socket
import threading
import time

with open('nodes.json') as f:
    nodes = json.load(f)

def get_node(pid):
    return next(n for n in nodes if n['pid'] == pid)

class RingNode:
    def __init__(self, pid):
        self.pid = pid
        self.node_info = get_node(pid)
        self.next_node = self.get_next_node()
        self.coordinator = None
        self.alive = True
        self.lock = threading.Lock()

    def get_next_node(self):
        idx = next(i for i, n in enumerate(nodes) if n['pid'] == self.pid)
        return nodes[(idx + 1) % len(nodes)]

    def start(self):
        threading.Thread(target=self.server_thread, daemon=True).start()
        time.sleep(5)  # Increased sleep to give time for all nodes to start

        self.start_election()

        while self.alive:
            time.sleep(5)
            if self.coordinator is None or not self.ping(self.coordinator):
                print(f"[{self.pid}] Coordinator {self.coordinator} not responding -> starting
election")
                self.start_election()

    def server_thread(self):
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s.bind((self.node_info['host'], self.node_info['port']))
        s.listen(5)
        print(f"[{self.pid}] Listening on {self.node_info['host']}:{self.node_info['port']} | next ->
{self.next_node['pid']} @ {self.next_node['host']}:{self.next_node['port']}")

        while self.alive:
            try:
                conn, addr = s.accept()
                threading.Thread(target=self.handle_connection, args=(conn,),
daemon=True).start()
            except Exception as e:
                print(f"[{self.pid}] Server accept error: {e}")

    def handle_connection(self, conn):
        try:
            msg = conn.recv(1024).decode()
            print(f"[{self.pid}] Received message: {msg}")
            if msg.startswith("ELECTION"):
                data = msg.split(maxsplit=2)
                origin = int(data[1])
                if len(data) > 2:
                    ids_str = data[2].strip("[]")
                    if ids_str:
                        ids = list(map(int, ids_str.split(",")))
                    else:
                        ids = []
                else:
                    ids = []

                if self.pid not in ids:
                    ids.append(self.pid)

                if origin == self.pid:
                    # Election message returned to origin, election done
                    coordinator = max(ids)
                    print(f"[{self.pid}] ELECTION result -> coordinator = {coordinator}")
                    with self.lock:
                        self.coordinator = coordinator
                    self.send_message(self.next_node, f"COORDINATOR {coordinator}")
                else:
                    print(f"[{self.pid}] Forwarding ELECTION from origin={origin} ids={ids}")
                    self.send_message(self.next_node, f"ELECTION {origin} [{','.join(map(str,
ids))}]")

            elif msg.startswith("COORDINATOR"):
                new_coord = int(msg.split()[1])
                with self.lock:
                    self.coordinator = new_coord
                print(f"[{self.pid}] Received COORDINATOR announcement: {new_coord}")
                if new_coord != self.pid:
                    self.send_message(self.next_node, msg)
        except Exception as e:
            print(f"[{self.pid}] Error handling connection: {e}")
        finally:
            conn.close()

    def start_election(self):
        print(f"[{self.pid}] Initiating election...")
        sent = self.send_message(self.next_node, f"ELECTION {self.pid} [{self.pid}]")
        if not sent:
            print(f"[{self.pid}] Failed to send election message to next node
{self.next_node['pid']}")

    def send_message(self, node, message):
        try:
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            s.settimeout(2)
            s.connect((node['host'], node['port']))
            s.sendall(message.encode())
            s.close()
            print(f"[{self.pid}] Sent message to {node['pid']}: {message}")
            return True
        except Exception as e:
            print(f"[{self.pid}] Failed to send message to {node['pid']}
({node['host']}:{node['port']}): {e}")
            return False

    def ping(self, pid):
        node = get_node(pid)
        try:
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            s.settimeout(2)
            s.connect((node['host'], node['port']))
            s.close()
            return True
        except Exception as e:
            print(f"[{self.pid}] Ping failed to {pid}: {e}")
            return False

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--pid", type=int, required=True)
    args = parser.parse_args()

    node = RingNode(args.pid)
    node.start()
```

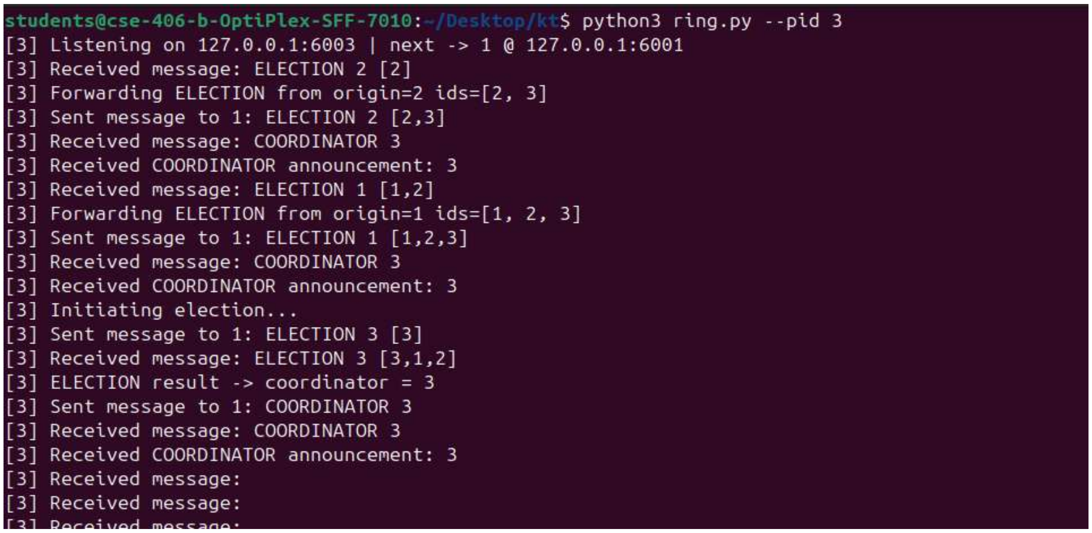
Output 2



```
students@cse-406-b-OptiPlex-SFF-7010:~/Desktop/kt$ python3 ring.py --pid 1
[1] Listening on 127.0.0.1:6001 | next -> 2 @ 127.0.0.1:6002
[1] Initiating election...
[1] Sent message to 2: ELECTION 1 [1]
[1] Coordinator None not responding -> starting election
[1] Initiating election...
[1] Sent message to 2: ELECTION 1 [1]
[1] Received message: ELECTION 2 [2,3]
[1] Forwarding ELECTION from origin=2 ids=[2, 3, 1]
[1] Sent message to 2: ELECTION 2 [2,3,1]
[1] Coordinator None not responding -> starting election
[1] Initiating election...
[1] Sent message to 2: ELECTION 1 [1]
[1] Received message: ELECTION 1 [1,2,3]
[1] ELECTION result -> coordinator = 3
[1] Sent message to 2: COORDINATOR 3
[1] Received message: ELECTION 3 [3]
[1] Forwarding ELECTION from origin=3 ids=[3, 1]
[1] Sent message to 2: ELECTION 3 [3,1]
[1] Received message: COORDINATOR 3
[1] Received COORDINATOR announcement: 3
[1] Sent message to 2: COORDINATOR 3
```



```
students@cse-406-b-OptiPlex-SFF-7010:~/Desktop/kt$ python3 ring.py --pid 2
[2] Listening on 127.0.0.1:6002 | next -> 3 @ 127.0.0.1:6003
[2] Received message: ELECTION 1 [1]
[2] Forwarding ELECTION from origin=1 ids=[1, 2]
[2] Failed to send message to 3 (127.0.0.1:6003): [Errno 111] Connection refused
[2] Initiating election...
[2] Failed to send message to 3 (127.0.0.1:6003): [Errno 111] Connection refused
[2] Failed to send election message to next node 3
[2] Received message: ELECTION 1 [1]
[2] Forwarding ELECTION from origin=1 ids=[1, 2]
[2] Failed to send message to 3 (127.0.0.1:6003): [Errno 111] Connection refused
[2] Coordinator None not responding -> starting election
[2] Initiating election...
[2] Sent message to 3: ELECTION 2 [2]
[2] Received message: ELECTION 2 [2,3,1]
[2] ELECTION result -> coordinator = 3
[2] Sent message to 3: COORDINATOR 3
[2] Received message: ELECTION 1 [1]
[2] Forwarding ELECTION from origin=1 ids=[1, 2]
[2] Sent message to 3: ELECTION 1 [1,2]
[2] Received message: COORDINATOR 3
[2] Received COORDINATOR announcement: 3
[2] Sent message to 3: COORDINATOR 3
[2] Received message: ELECTION 3 [3,1]
[2] Forwarding ELECTION from origin=3 ids=[3, 1, 2]
[2] Sent message to 3: ELECTION 3 [3,1,2]
[2] Received message: COORDINATOR 3
[2] Received COORDINATOR announcement: 3
[2] Sent message to 3: COORDINATOR 3
```



```
students@cse-406-b-OptiPlex-SFF-7010:~/Desktop/kt$ python3 ring.py --pid 3
[3] Listening on 127.0.0.1:6003 | next -> 1 @ 127.0.0.1:6001
[3] Received message: ELECTION 2 [2]
[3] Forwarding ELECTION from origin=2 ids=[2, 3]
[3] Sent message to 1: ELECTION 2 [2,3]
[3] Received message: COORDINATOR 3
[3] Received COORDINATOR announcement: 3
[3] Received message: ELECTION 1 [1,2]
[3] Forwarding ELECTION from origin=1 ids=[1, 2, 3]
[3] Sent message to 1: ELECTION 1 [1,2,3]
[3] Received message: COORDINATOR 3
[3] Received COORDINATOR announcement: 3
[3] Initiating election...
[3] Sent message to 1: ELECTION 3 [3]
[3] Received message: ELECTION 3 [3,1,2]
[3] ELECTION result -> coordinator = 3
[3] Sent message to 1: COORDINATOR 3
[3] Received message: COORDINATOR 3
[3] Received COORDINATOR announcement: 3
[3] Received message:
[3] Received message:
[3] Received message:
```

Conclusion

From this experiment, I learned how Bully and Ring Election Algorithms work in distributed systems to select a new coordinator when the current one fails.

Both algorithms ensure that the system continues to function even after a node crash, which is important for reliability and fault tolerance.

In the Bully Algorithm, the process with the highest ID quickly takes over as the coordinator, making it faster but with higher message overhead.

In contrast, the Ring Algorithm provides a more balanced and fair approach since every process gets a chance to participate, though it can take more time as messages circulate through the ring.

Overall, this lab helped me understand how leader election, process coordination, and failure recovery are managed practically in distributed systems. It also improved my understanding of inter-process communication and how algorithms maintain system stability after a failure.

References

1. A. S. Tanenbaum and M. Van Steen, *Distributed Systems: Principles and Paradigms*, 2nd Edition, Pearson Education, 2007.
2. G. Coulouris, J. Dollimore, T. Kindberg, *Distributed Systems: Concepts and Design*, 5th Edition, Pearson, 2012.
3. Course Lab Manual – *DC Experiment 9: Bully and Ring Election Algorithms*.
4. Online reference: GeeksforGeeks – *Bully and Ring Election Algorithms in Distributed Systems* (https://www.geeksforgeeks.org/).