**TITLE:**

Bully election and ring election algorithm

**AIM:**

To study and implement Bully Election Algorithm

**OBJECTIVES:**

1. To understand the working of election algorithms

2. To simulate bully algorithm

**THEORY:**

**Elections**

Election in distributed computing refers to the process of selecting a leader or coordinator node among a group of distributed nodes. This leader is responsible for coordinating activities, making decisions, and ensuring the integrity and consistency of the distributed system. Elections are crucial for fault tolerance, load balancing, and maintaining system stability in distributed environments. Various algorithms, such as the Bully Algorithm and the Ring Algorithm, are used to facilitate the election process, ensuring that a new leader is elected promptly in the event of failures or changes in the system topology.

**Bully Algorithm**

The Bully Algorithm is a leader election algorithm used in distributed systems. It allows nodes within a network to elect a coordinator or leader in a decentralized manner. The Bully Algorithm operates on the principle of higher priority.

Messages:

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 winner of the election to announce the new coordinator.

Steps:

1. Assume there are 6 Processes P0, P1, P2, P3, P4, P5 written in ascending order of their Process ID.
2. Suppose Process P2 sends a message to coordinator P5 and P5 does not respond in a desired time T (possible reason could be crash, down, etc.)
3. Then process P2, sends an election message to all processes with Process ID greater than P2 (i.e. P3, P4 & P5) and awaits a response from the processes.
4. If no one responds, P2 wins the election and become the coordinator.
5. If any of the processes with Process ID higher than 2 responds with OK, P2’s job is done and this Process will take over.
6. It then restarts and initiates an election message.
7. Process P4 responds to P3 with an OK message to confirm its alive state and Process P4 figures out that process 5 has crashed, and the new process with the highest ID is process 4.
8. The process that receives an election message sends a coordinator message if it is the Process with the highest ID (in this case it is P4).
9. If a Process which was previously down (i.e. P5) comes back, it holds an election and if it has the highest Process Id then it will become the new coordinator and sends message to all other processes.

**CODE:**

#include <iostream>

#include <list>

using *namespace* std;

// class to represent a single node

*class* Node

{

*private:*

    // Static variable for ID

    static *int* nextId;

*public:*

*int* id;

*bool* isAlive;

*bool* hasStartedElection;

*int* coordinator;

*bool* response;

    // constructor makes nodes alive, sets ID and sets no coordinator

    Node() : id(nextId++), isAlive(true), hasStartedElection(false), coordinator(-1), response(false) {}

*void* disableNode()

    {

        this->isAlive = false;

        cout << "Node " << this->id << " disabled." << endl;

    }

*void* enableNode()

    {

        this->isAlive = true;

        cout << "Node " << this->id << " enabled." << endl;

    }

*void* displayNode() const

    {

        cout << "Node ID: " << this->id << ", Alive: " << this->isAlive << ", Election Started: " << this->hasStartedElection << ", Coordinator: " << this->coordinator << endl;

    }

};

// init static variable

*int* Node::nextId = 1;

// bully algo

*void* startBullyElection(list<Node> &*nodes*, *int* *startingNodeId*)

{

    // Find the starting node

*auto* startingNode = *nodes*.end();

    for (*auto* it = *nodes*.begin(); it != *nodes*.end(); ++it)

    {

        if (it->id == *startingNodeId*)

        {

            startingNode = it;

            break;

        }

    }

    // if ahead of last node, return

    if (startingNode == *nodes*.end())

    {

        return;

    }

    // is node alive and not started election

    if (startingNode->isAlive && startingNode->hasStartedElection == false)

    {

        // cout<<startingNode->id<<endl;

*auto* temp = startingNode;

        ++temp;

        for (*auto* it = temp; it != *nodes*.end(); it++)

        {

            cout << startingNode->id << " sends election to " << it->id << endl;

            if (it->isAlive)

            {

                cout << it->id << " sends OK." << endl;

                startingNode->response = true;

            }

        }

        // if no response, the sender is the coordinator

        if (startingNode->response == false)

        {

            cout << "No response, " << startingNode->id << " is coordinator." << endl;

            for (*auto* it = *nodes*.begin(); it != *nodes*.end(); ++it)

            {

                it->coordinator = startingNode->id;

            }

            return;

        }

    }

    // recursive call to next node

    startBullyElection(*nodes*, *startingNodeId* + 1);

}

// ring algo

*void* startRingElection(list<Node> &*nodes*, *int* *startingNodeId*)

{

    // Find the starting node

*auto* startingNode = *nodes*.end();

    for (*auto* it = *nodes*.begin(); it != *nodes*.end(); ++it)

    {

        if (it->id == *startingNodeId*)

        {

            startingNode = it;

            break;

        }

    }

    // if ahead of last node, return

    if (startingNode == *nodes*.end())

    {

        return;

    }

*int* maxId = *startingNodeId*;

*bool* foundHigher = false;

*auto* currentIt = startingNode;

    do

    {

        // move to the next node, wrap to the beginning if at the end

        do

        {

            currentIt++;

            if (currentIt == *nodes*.end())

            {

                currentIt = *nodes*.begin();

            }

        } while (!currentIt->isAlive && currentIt->id != *startingNodeId*); // Skip dead nodes

        if (currentIt->id == *startingNodeId* && foundHigher)

        {

            // if made a full loop and found a higher ID, break

            break;

        }

        if (currentIt->id > maxId)

        {

            maxId = currentIt->id;

            foundHigher = true;

        }

        cout << "Node " << startingNode->id << " passes election to Node " << currentIt->id << endl;

        startingNode = currentIt; // move to next node

    } while (currentIt->id != *startingNodeId*); // loop back to the start

    // after completing the loop, maxId is the ID of the elected coordinator

    cout << "Node " << maxId << " is elected as coordinator." << endl;

    for (*auto* &node : *nodes*)

    {

        node.coordinator = maxId;

    }

}

*int* main()

{

    list<Node> nodes;

*int* choice;

    do

    {

        // menu

        cout << "\nMenu:\n"

             << "1. Add nodes\n"

             << "2. Disable node\n"

             << "3. Enable node\n"

             << "4. Display nodes\n"

             << "5. Start election\n"

             << "6. Exit\n"

             << "Enter your choice: ";

        cin >> choice;

        switch (choice)

        {

        case 1:

        {

            // add nodes

*int* n;

            cout << "Enter the number of nodes to add: ";

            cin >> n;

            for (*int* i = 0; i < n; ++i)

            {

                nodes.push\_back(Node());

                cout << "Node " << nodes.back().id << " added successfully." << endl;

            }

            break;

        }

        case 2:

        {

            // disable a node

*int* nodeId;

            cout << "Enter ID of the node to disable: ";

            cin >> nodeId;

*bool* found = false;

            for (*auto* &node : nodes)

            {

                if (node.id == nodeId)

                {

                    node.disableNode();

                    found = true;

                    break;

                }

            }

            if (!found)

            {

                cout << "Node not found." << endl;

            }

            break;

        }

        case 3:

        {

            // enable a node

*int* nodeId;

            cout << "Enter ID of the node to enable: ";

            cin >> nodeId;

*bool* found = false;

            for (*auto* &node : nodes)

            {

                if (node.id == nodeId)

                {

                    node.enableNode();

                    found = true;

                    break;

                }

            }

            if (!found)

            {

                cout << "Node not found." << endl;

            }

            break;

        }

        case 4:

            // display all nodes

            if (nodes.empty())

            {

                cout << "No nodes added yet." << endl;

            }

            else

            {

                cout << "Nodes:\n";

                for (const *auto* &node : nodes)

                {

                    node.displayNode();

                }

            }

            break;

        case 5:

        {

            // start election

            cout << "Choose election algorithm:\n"

                 << "1. Bully election algorithm\n"

                 << "2. Ring election algorithm\n"

                 << "Enter your choice: ";

*int* algoChoice;

            cin >> algoChoice;

            switch (algoChoice)

            {

            case 1:

            {

                // bully algo

*int* startingNode;

                cout << "Enter the starting node ID for the election: ";

                cin >> startingNode;

                startBullyElection(nodes, startingNode);

                // reset the election flags after election is over

                for (*auto* &node : nodes)

                {

                    node.hasStartedElection = false;

                    node.response = false;

                }

                break;

            }

            case 2:

                // ring algo

*int* startingNode;

                cout << "Enter the starting node ID for the election: ";

                cin >> startingNode;

                startRingElection(nodes, startingNode);

                // reset the election flags after election is over

                for (*auto* &node : nodes)

                {

                    node.hasStartedElection = false;

                    node.response = false;

                }

                break;

            default:

                cout << "Invalid choice." << endl;

                break;

            }

            break;

        }

        case 6:

            // Exit

            cout << "Exiting program." << endl;

            break;

        default:

            cout << "Invalid choice." << endl;

            break;

        }

    } while (choice != 6);

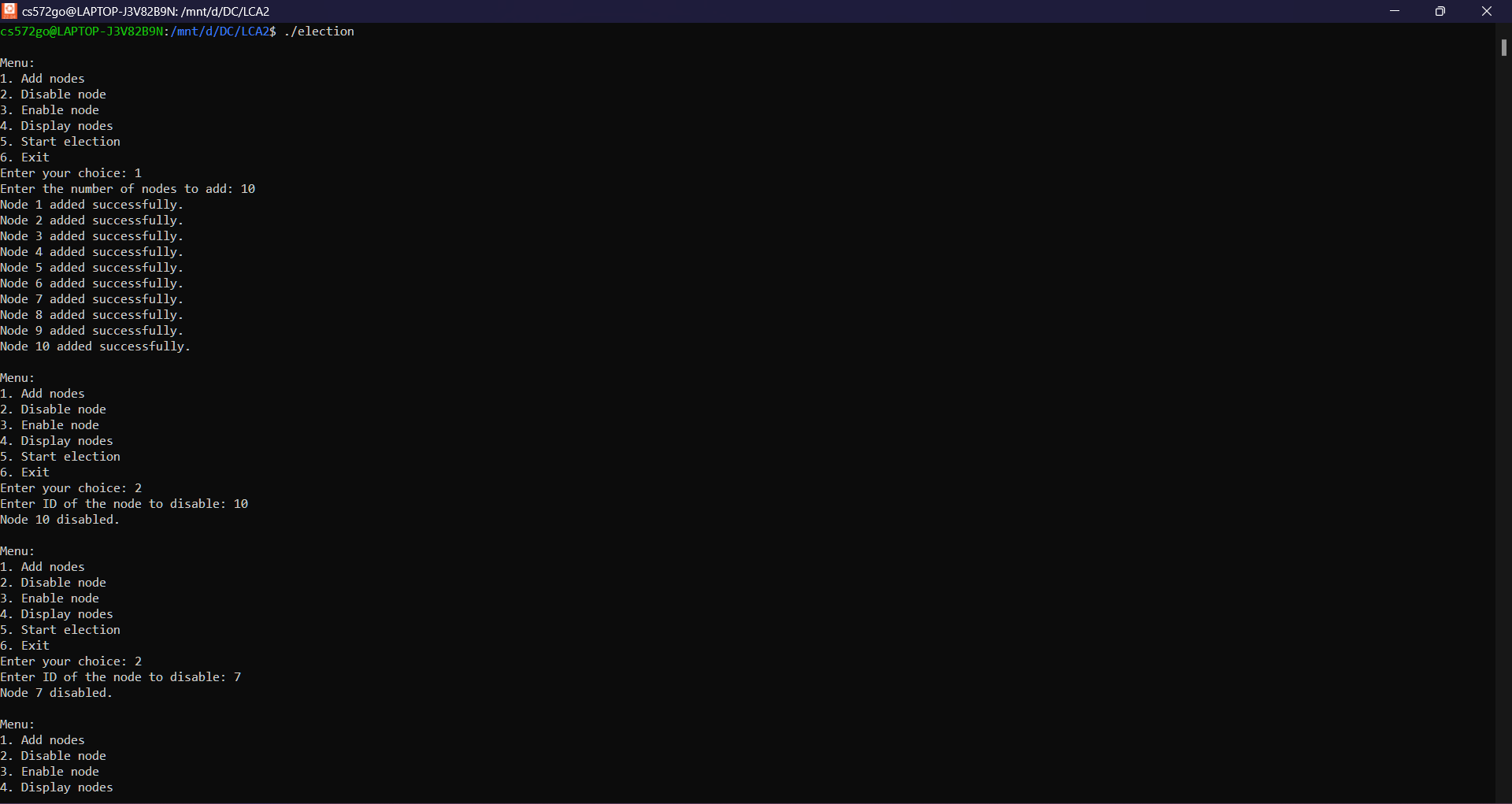
    return 0;

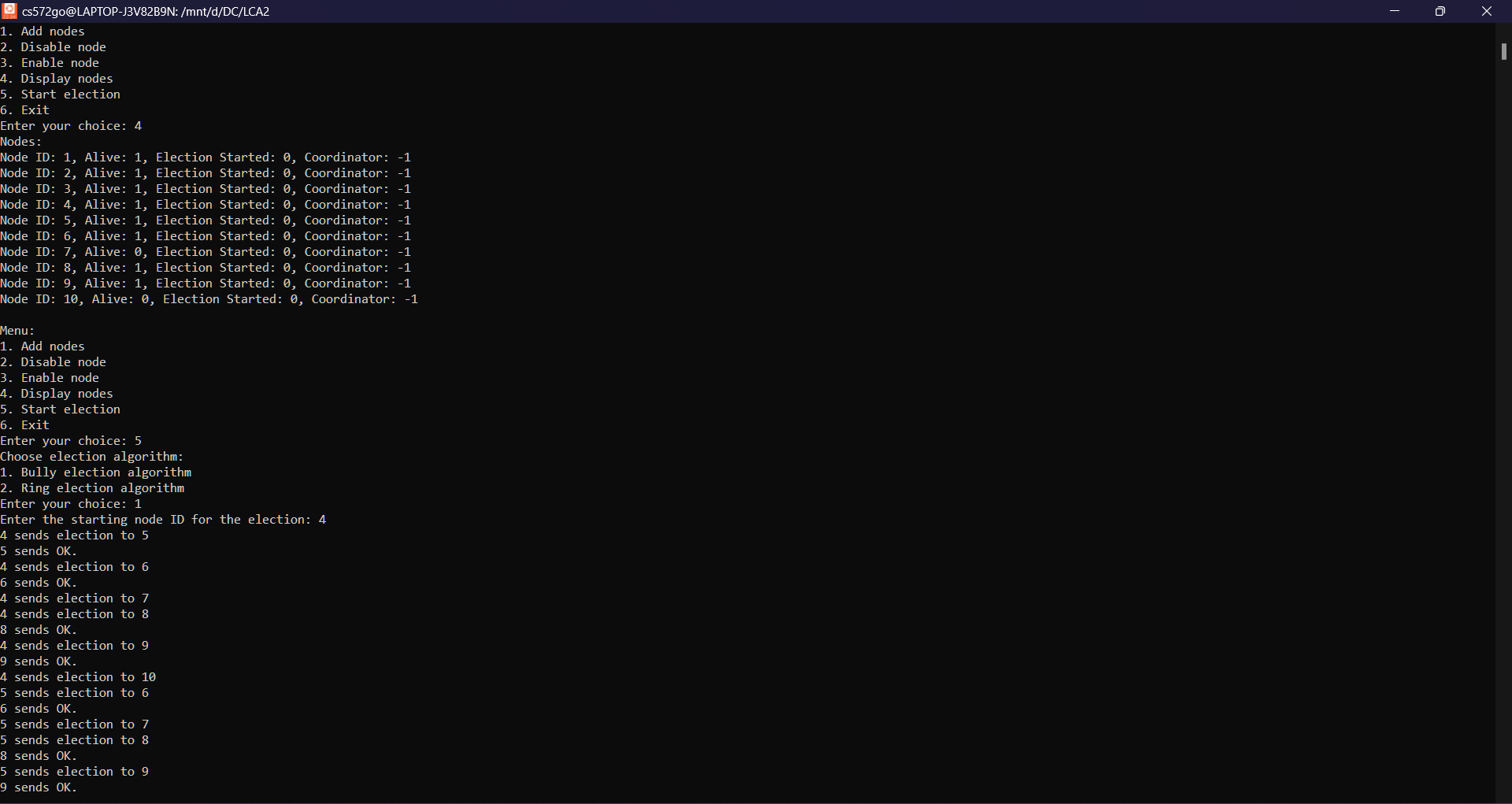
}

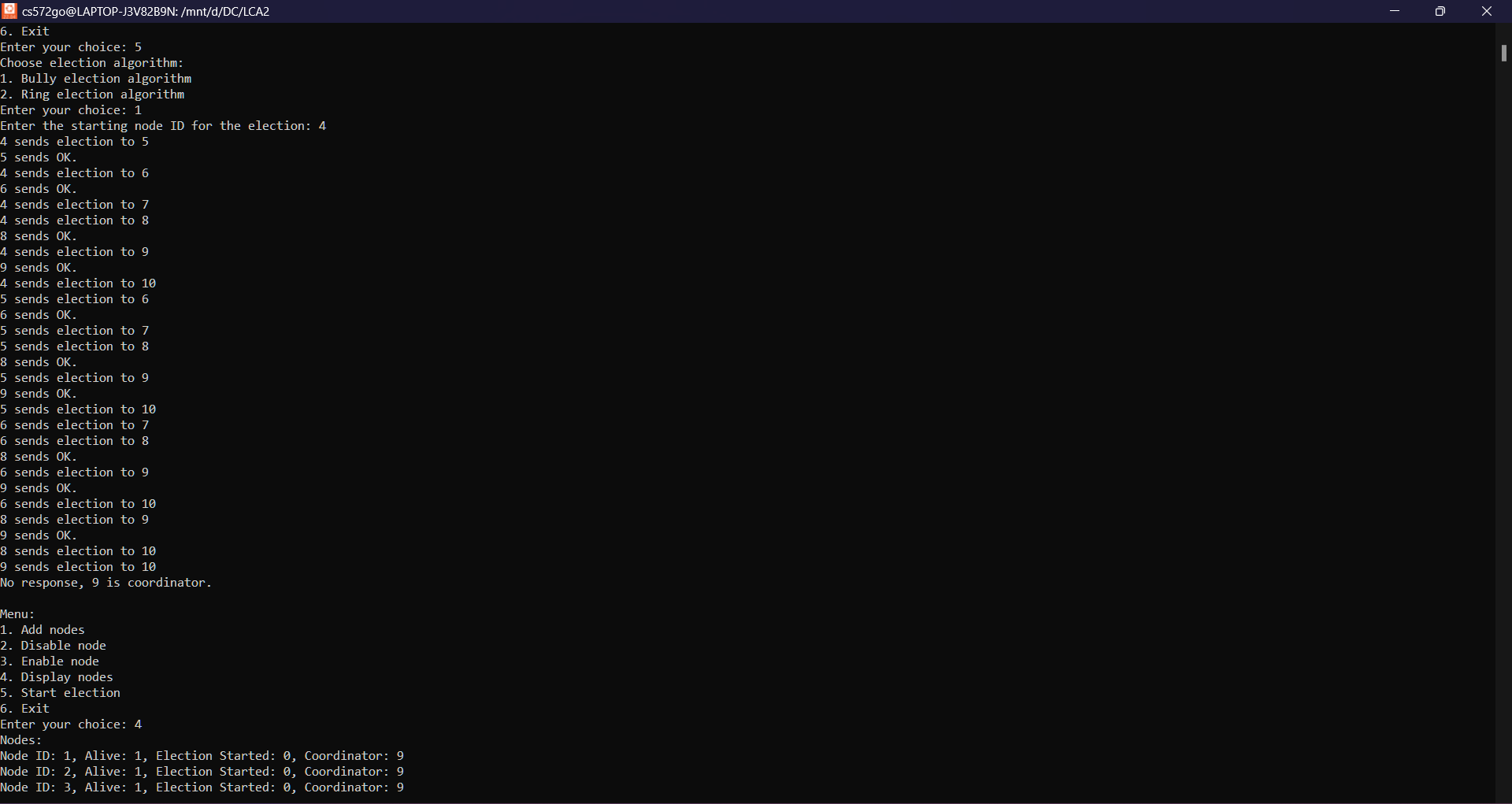
**INPUT**: 10 processes with IDs from 1 to 10. Fail node 10 and 7. Start election from node 4.

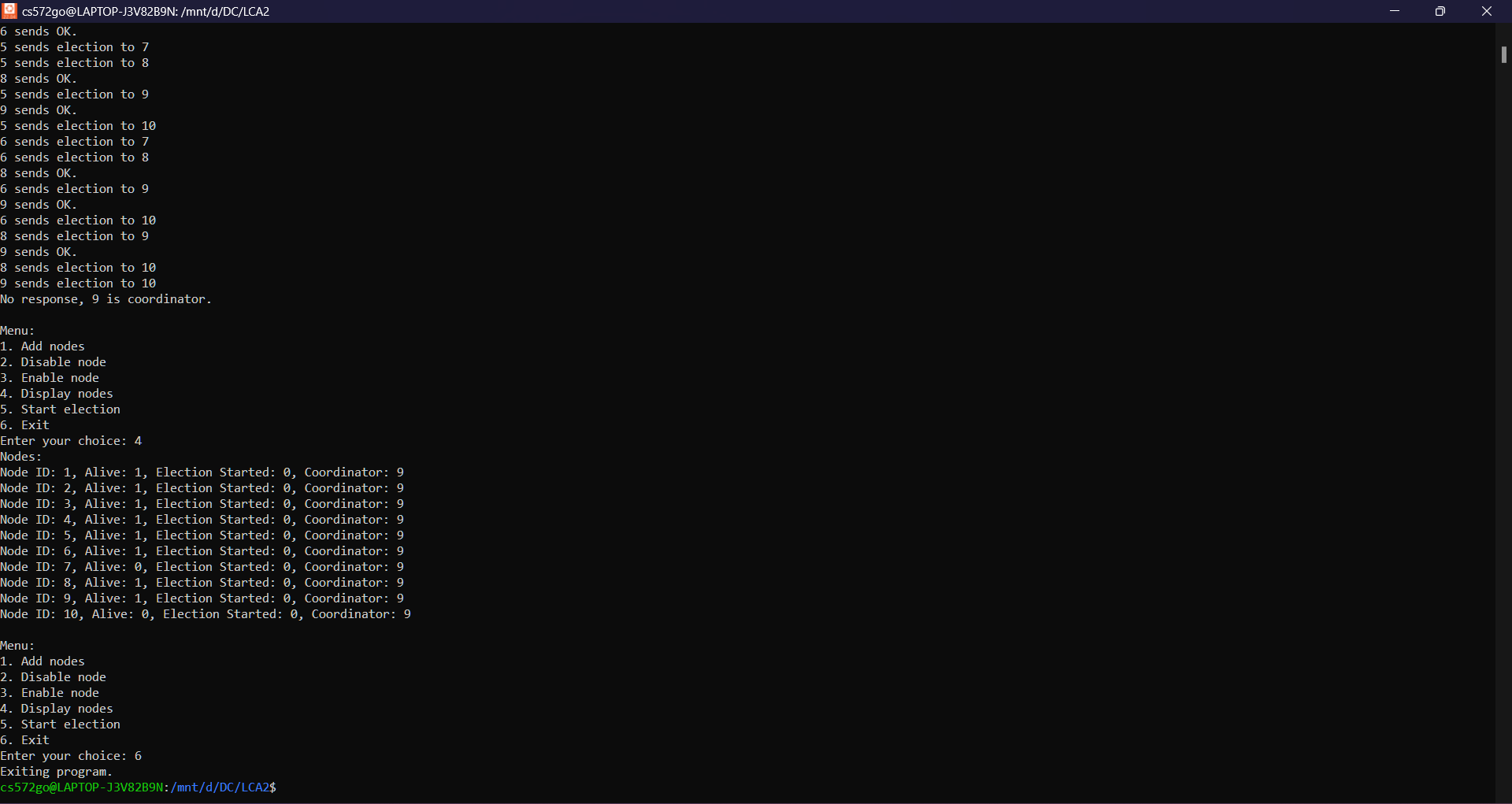
**OUTPUT:**

1. **Bully Algorithm**

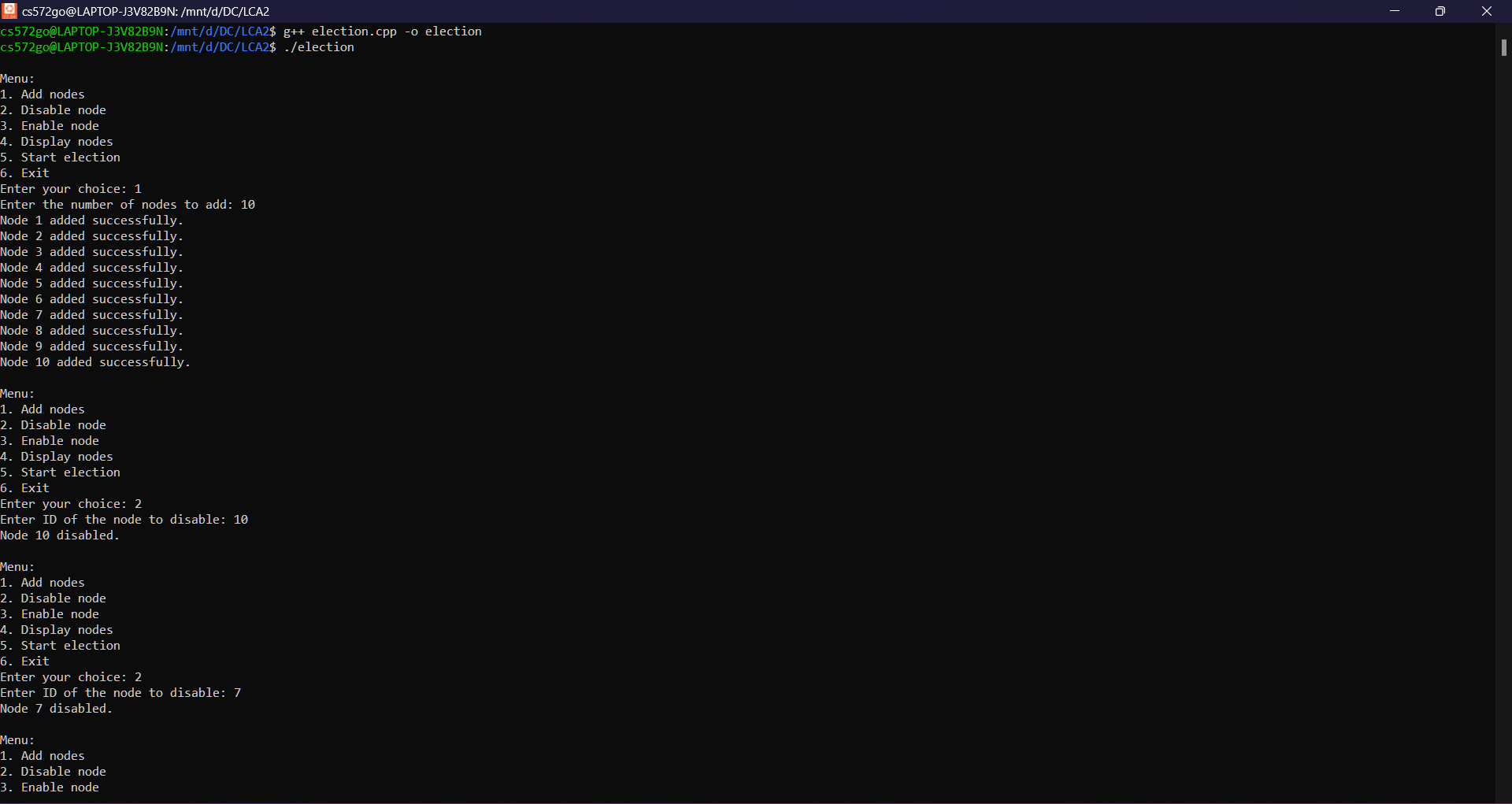
****

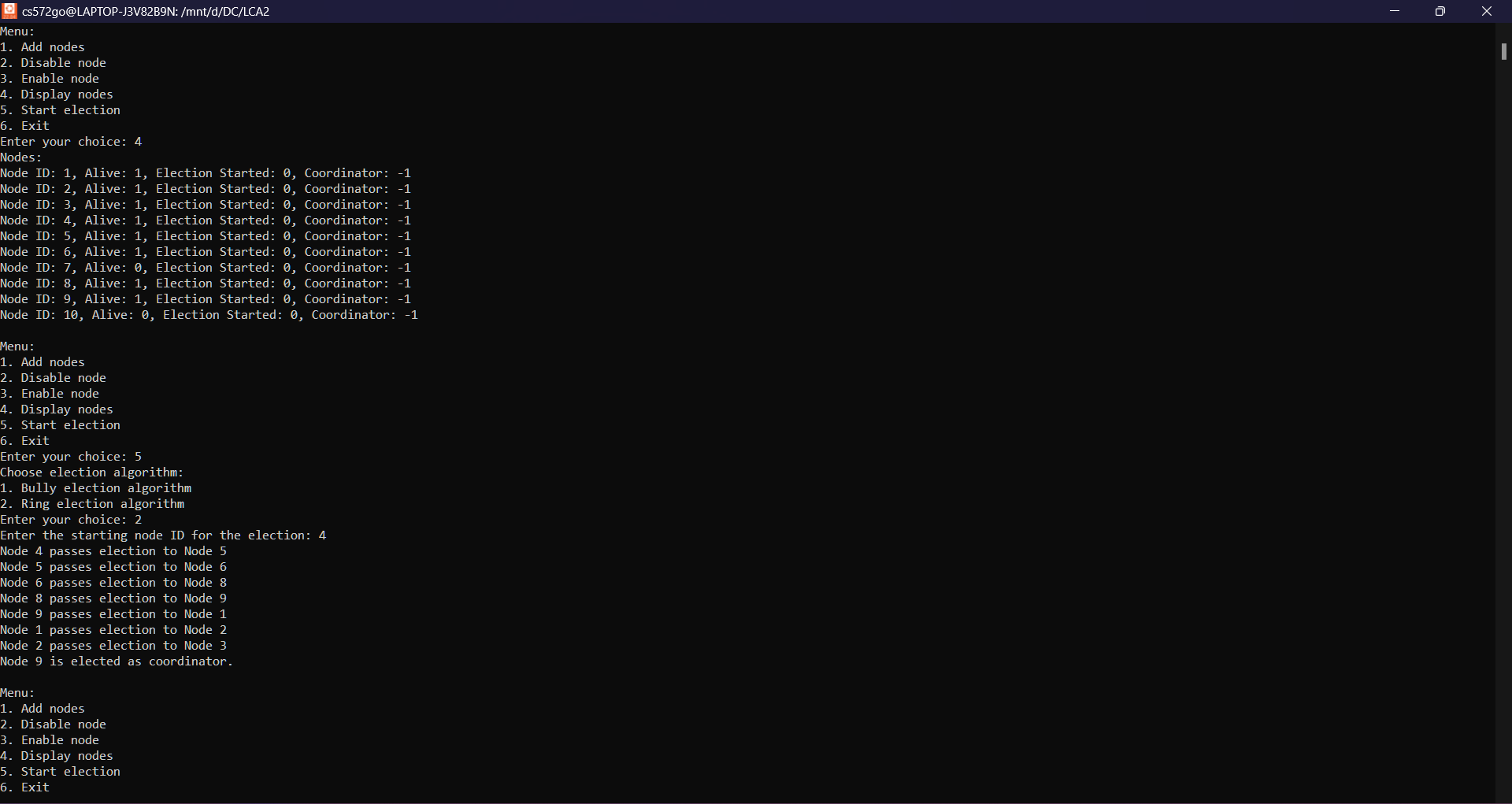
****

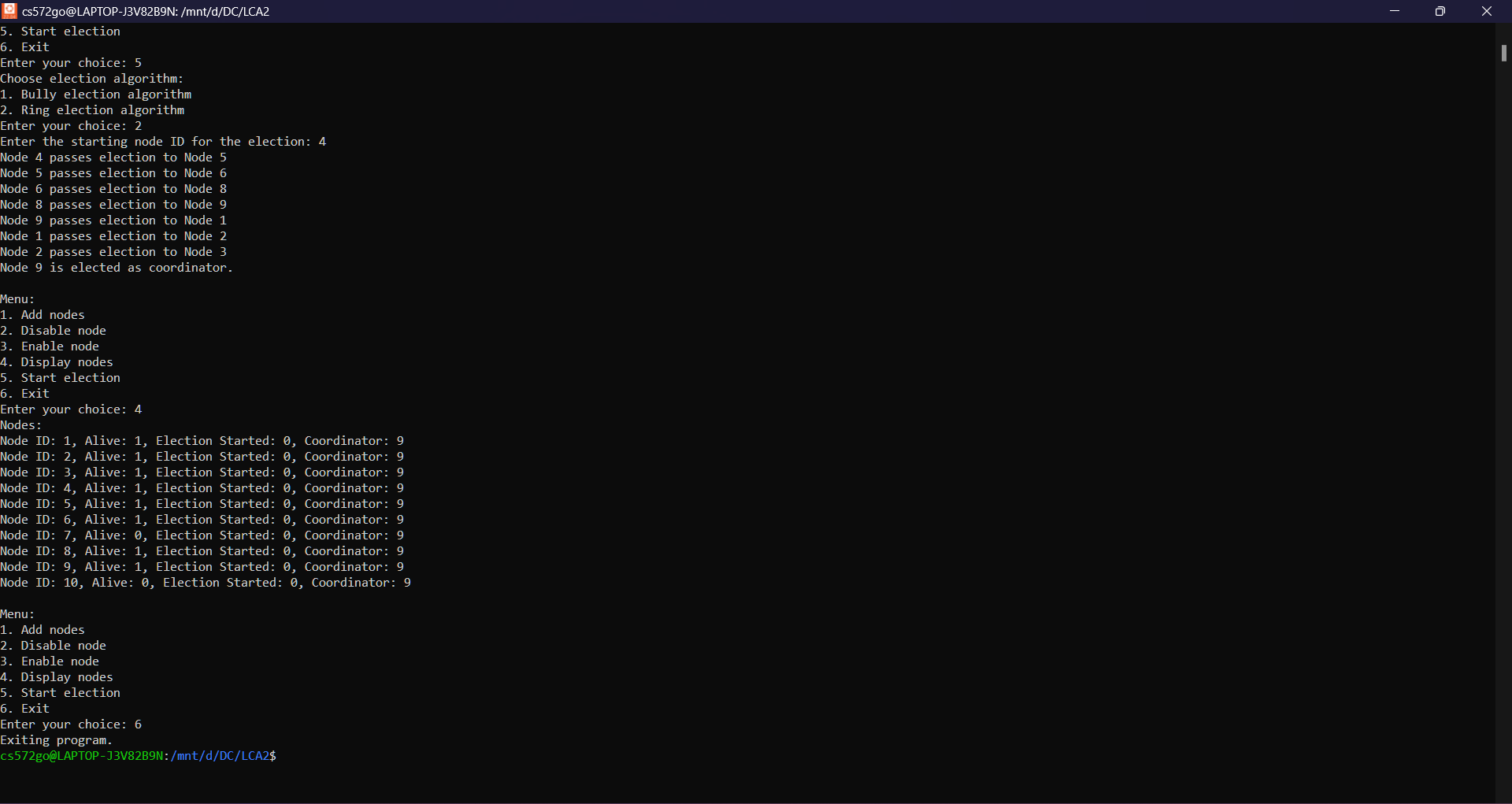
****

****

1. **Ring algorithm**

****

****

****

**PLATFORM:**

Windows:

VSCode

Windows Subsystem for Linux (Ubuntu 22.04 LTS)

**LANGUAGE:**

C++.

**CONCLUSION:**

Thus, bully algorithm is successfully implemented.

**FAQS**

1. What is the time complexity (best, average, worst) of bully algorithm?

Answer:

Best Case: O(1), when the highest process ID is the coordinator initially, requiring no election.

Average Case: O(n), considering a random distribution of processes across the network.

Worst Case: O(n^2), when the lowest process ID initiates an election, leading to a complete exchange of messages among all processes.

2. Why do we have to elect the coordinator process?

Answer:

Electing a coordinator process ensures system resilience and efficient task management in distributed systems. It establishes a single point of contact for coordinating actions, such as resource allocation, task scheduling, and fault management. By electing a coordinator, the system can maintain order and consistency, even in the presence of failures or network partitions. This process helps prevent conflicts, reduces message overhead, and facilitates efficient communication and decision-making within the system.

3. How did the name of "Bully" approach come up?

Answer:

The name "Bully" in the Bully Algorithm reflects the behaviour of nodes in a distributed system. When a lower-ranked node detects that the current coordinator has failed, it “bullies” higher-ranked nodes by initiating an election process. This aggressive behaviour is like a subordinate challenging higher-ranking node for leadership in the system. The term “bully” serves as a metaphor for this hierarchical interaction, where lower-ranked nodes assert themselves to take control in the absence of a coordinator.