**AIM : To study and implement Bully Election Algorithm**

**OBJECTIVES :**  
1. To understand the working of election algorithms  
2. To simulate bully algorithm

**THEORY :**

Elections –

Elections are a crucial step in the Bully Algorithm's procedure for choosing a coordinator or leader for a distributed system. Elections are held in order to guarantee that a coordinator is always in place to oversee system functioning.

Bully Algorithm –

In distributed systems, the Bully Algorithm is a mechanism for leader elections in which several processes vie to choose a coordinator. A process starts an event when it notices that there isn't a coordinator or that it wants to take over as the coordinator. Election through contesting higher-priority processes (lower process IDs). When a timeout passes and no higher-priority process responds, the initiator takes over as coordinator. If not, it gives way to the procedure with a higher priority. This guarantees that a coordinator oversees the procedures in the event of a breakdown as well.

Types of messages -  
There are basically 3 types of messages :  
1. An election message to initiate the election  
2. A reply/response message given in response to the election message  
3. A coordinator message sent to inform other processes, the id of the coordinator process.

**INPUT :** Process IDS

**OUTPUT :** Selected Coordinator Process

**PLATFORM:** UNIX

**PROGRAMMING LANGUAGE :** C Language

**CONCLUSION:** Thus, bully algorithm is successfully implemented.  
FAQS  
**1. What is the time complexity (best,avg,worst)of bully algorithm?**

Ans.

* Best Case: O(1) in which there is no election required and the coordinator has the highest process ID at first.
* Average Case: In random process distribution across the network, this is usually O(n).
* Worst Case: O(n^2), when an election is started by the lowest ID and all processes exchange messages in full.

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

Ans. In distributed systems, we choose a coordinator process to make sure that nodes coordinate and communicate effectively. By centralising decision-making and minimising conflict, the coordinator improves system reliability and fault tolerance by facilitating tasks like resource allocation, distributed computation, and data consistency.

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

Ans. The "bully" approach in the distributed computing context derives its name from the behaviour exhibited by nodes in the system. When a node detects the failure of the current leader, it "bullies" its way to assume leadership by challenging other nodes through message exchanges to assert its dominance and become the new leader.

**Code :**

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

class node

{

public:

    int id;

    int coordinator;

    bool isAlive;

    bool electionStarted;

    bool message\_back;

    bool visited;

    node \*next = NULL;

    node()

    {

        this->isAlive = true;

        this->electionStarted = false;

    }

    node(int i, bool c)

    {

        this->id = i;

        this->coordinator = c ? i : -1;

        this->isAlive = true;

        this->electionStarted = false;

        this->message\_back = false;

        this->visited = false;

    }

    node \*startProcess(int total\_nodes)

    {

        int i\_d;

        bool isC;

        node \*p = NULL;

        node \*start = NULL;

        for (int i = 1; i <= total\_nodes; i++)

        {

            cout << "Enter id for process " << i << ": ";

            cin >> i\_d;

            cout << "Is process " << i\_d << " a coordinator (1/0): ";

            cin >> isC;

            node \*curr = new node(i\_d, isC);

            if (p == NULL)

            {

                start = curr;

            }

            else

            {

                p->next = curr;

            }

            p = curr;

        }

        return start;

    }

    void display\_processes(node \*n)

    {

        node \*head = n;

        while (head != NULL)

        {

            cout << "Process id : " << head->id << endl;

            cout << "Is coordinator : " << (head->coordinator != -1 ? "Yes" : "No") << endl;

            cout << "Election Started : " << (head->electionStarted ? "Yes" : "No") << endl;

            cout << "Is Alive : " << (head->isAlive ? "Yes" : "No") << endl

                 << endl;

            head = head->next;

        }

    }

    void kill\_node(int i\_d, node \*n)

    {

        node \*head = n;

        while (head != NULL)

        {

            if (head->id == i\_d)

            {

                head->isAlive = false;

            }

            head = head->next;

        }

    }

    void startElection\_bully(int total\_nodes, node \*start\_node, node \*coppy)

    {

        if (start\_node == NULL)

        {

            return;

        }

        node \*s = start\_node;

        if (!s->electionStarted)

        {

            s->electionStarted = true;

            for (int i = s->id; i < total\_nodes; i++)

            {

                cout << start\_node->id << " sent message to " << i + 1 << endl;

            }

            s = s->next;

            bool flag = false;

            while (s != NULL)

            {

                if (s->isAlive)

                {

                    cout << s->id << " sent ok back to " << start\_node->id << "\n";

                    flag = true;

                }

                s = s->next;

            }

            if (flag)

            {

                start\_node->message\_back = true;

            }

            if (!start\_node->message\_back)

            {

                cout << "New coordinator is " << start\_node->id << endl;

                while (coppy != NULL)

                {

                    coppy->coordinator = start\_node->id;

                    coppy = coppy->next;

                }

            }

            if (start\_node->next == NULL)

            {

                return;

            }

            s = start\_node;

            while (s->next != NULL)

            {

                if (s->next->isAlive)

                {

                    startElection\_bully(total\_nodes, s->next, coppy);

                }

                s = s->next;

            }

        }

    }

    void startElection\_ring(int total\_nodes, node \*start\_node)

    {

        int ans = INT\_MIN;

        int num\_visited = 0;

        node \*current\_node = start\_node;

        while (current\_node != nullptr && !current\_node->visited)

        {

            current\_node->visited = true;

            num\_visited++;

            ans = max(ans, current\_node->id);

            node \*next\_node = current\_node->next;

            while (next\_node != nullptr && !next\_node->isAlive)

            {

                next\_node = next\_node->next;

            }

            if (next\_node != nullptr && !next\_node->visited)

            {

                cout << current\_node->id << " Sent election e" << current\_node->id << " to " << next\_node->id << endl;

            }

            current\_node = next\_node;

        }

        node \*reset\_node = start\_node;

        while (reset\_node != nullptr)

        {

            reset\_node->visited = false;

            reset\_node = reset\_node->next;

        }

        cout << "New coordinator is " << ans << endl;

    }

};

int main()

{

    cout << "Election Algorithms (Bully and Ring) : " << endl;

    node x;

    node \*start\_node = NULL;

    int choice = 0;

    int num\_nodes = 0;

    do

    {

        cout << "Menu :" << endl;

        cout << "1. Add nodes" << endl;

        cout << "2. Disable a node" << endl;

        cout << "3. Display nodes" << endl;

        cout << "4. Start Bully algorithm election" << endl;

        cout << "5. Start Ring algorithm election" << endl;

        cout << "6. Exit" << endl;

        cin >> choice;

        switch (choice)

        {

        case 1:

            cout << "Number of nodes to be added : ";

            cin >> num\_nodes;

            start\_node = x.startProcess(num\_nodes);

            break;

        case 2:

            if (start\_node != NULL)

            {

                int node\_idx\_to\_kill;

                cout << "Enter node to disable: ";

                cin >> node\_idx\_to\_kill;

                x.kill\_node(node\_idx\_to\_kill, start\_node);

            }

            else

            {

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

            }

            break;

        case 3:

            if (start\_node != NULL)

            {

                x.display\_processes(start\_node);

            }

            else

            {

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

            }

            break;

        case 4:

            int node\_id;

            cout << "Enter node id that starts the election\n";

            cin >> node\_id;

            for (int i = 0; i < node\_id - 1; i++)

            {

                start\_node = start\_node->next;

            }

            x.startElection\_bully(num\_nodes, start\_node, start\_node);

            break;

        case 5:

            x.startElection\_ring(num\_nodes, start\_node);

            break;

        default:

            cout << "Enter valid choice" << endl;

            break;

        }

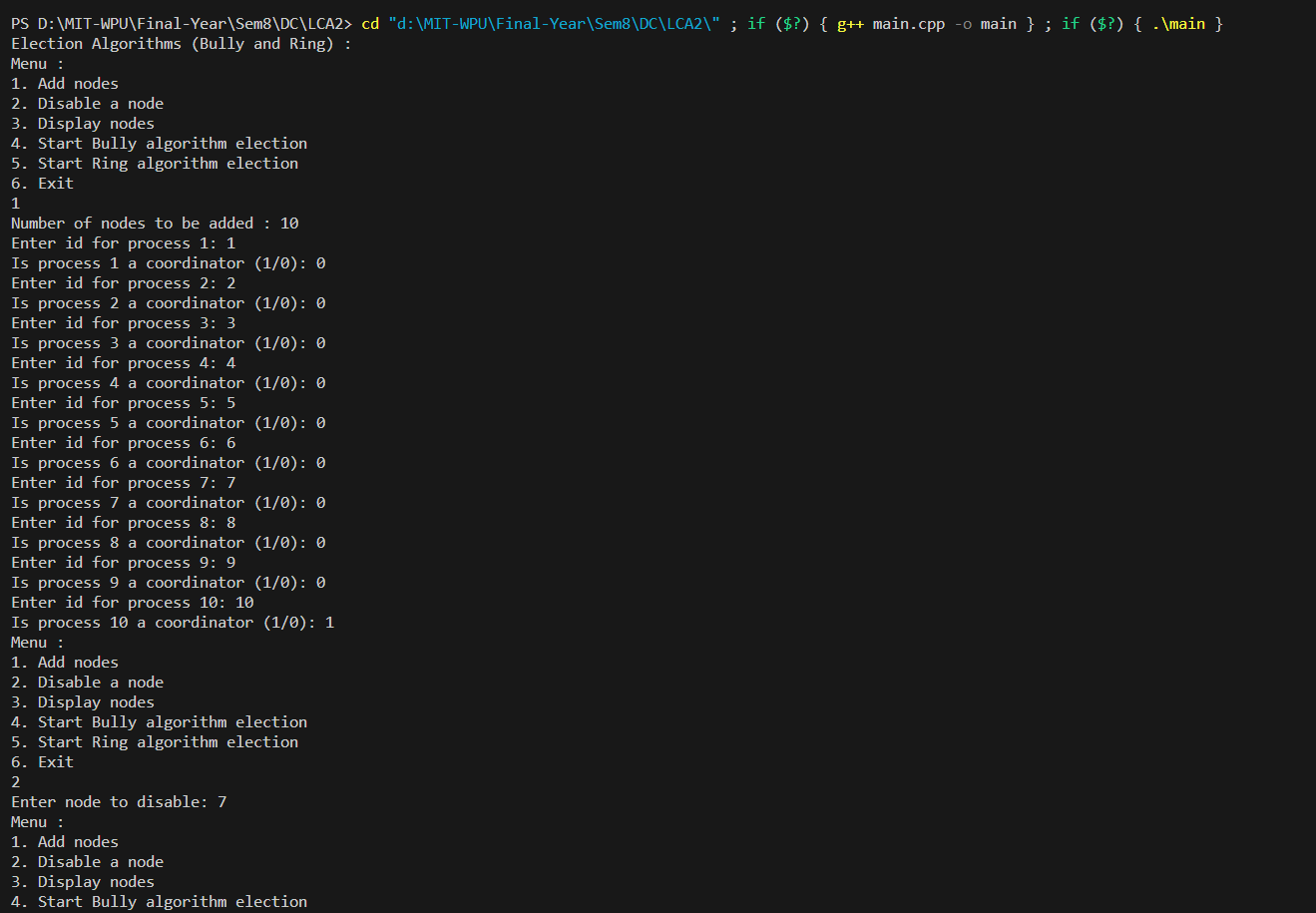
    } while (choice != 6);

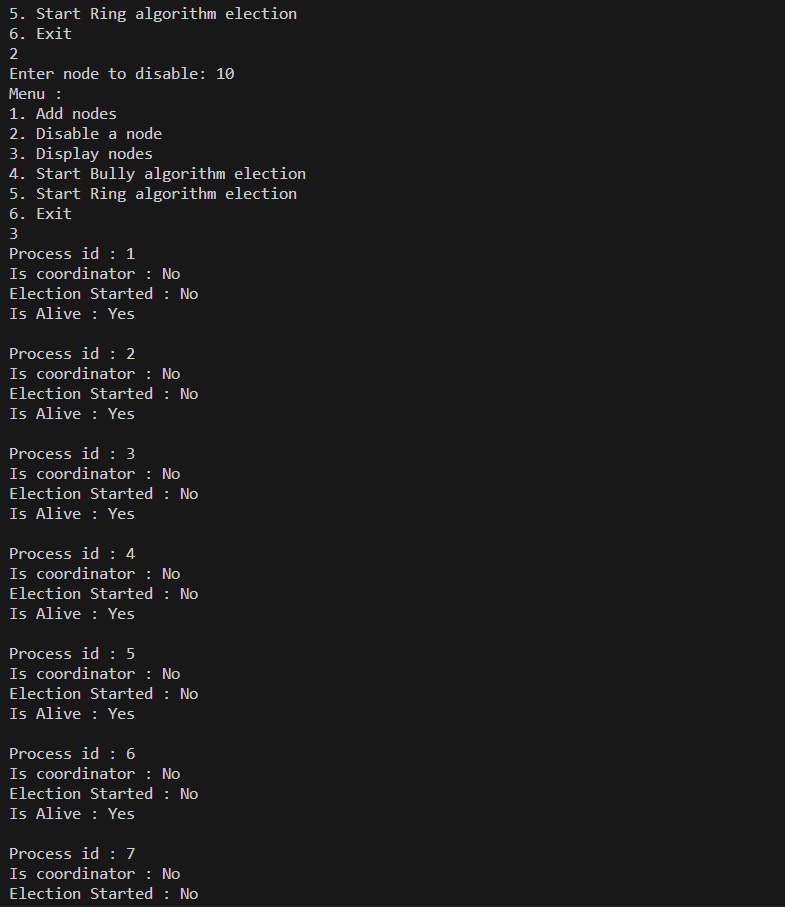
    return 0;

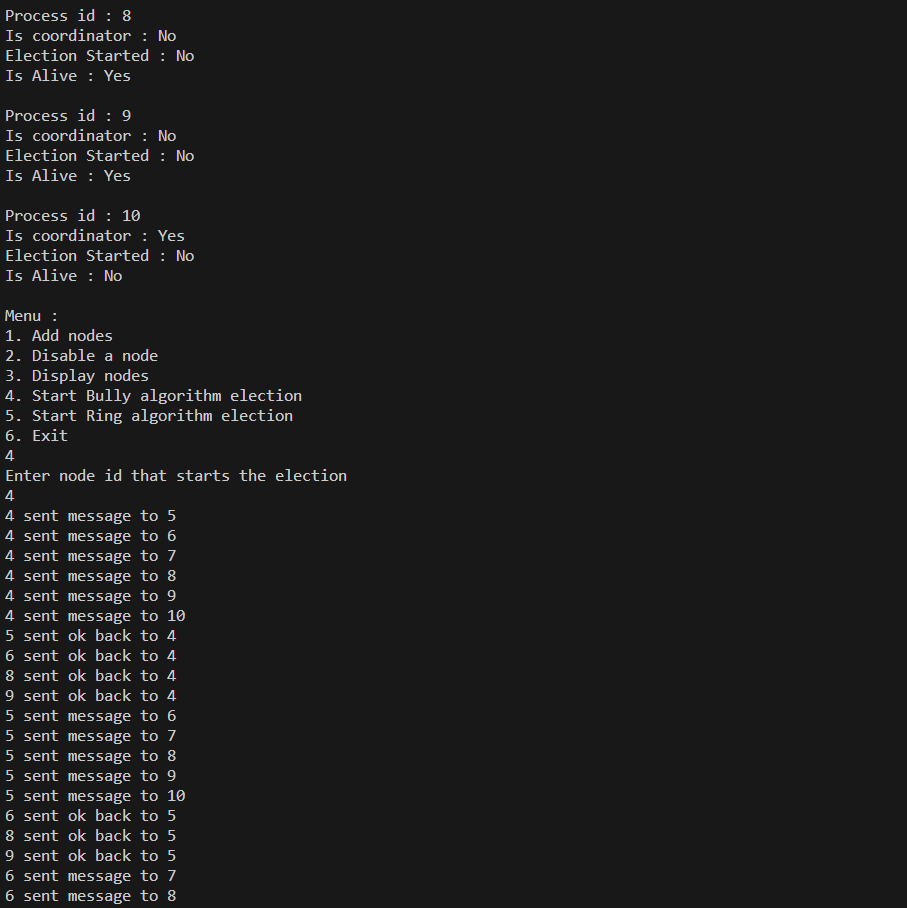
}

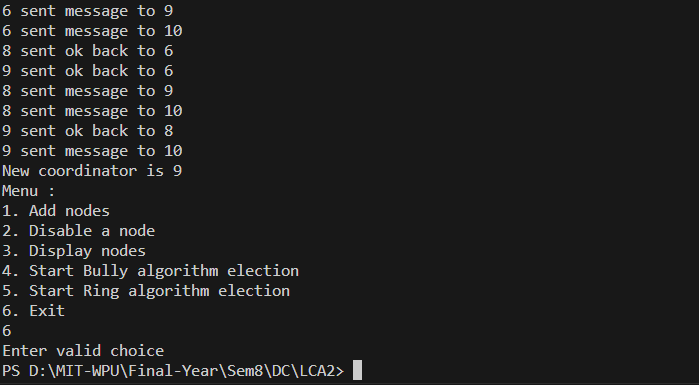
**Output :**

* **Bully Algorithm**









* **Ring Algorithm**

