ASSIGNMENT 4

Name – Sarvesh Sanjay Shingare
Roll No – 144
PRN – 202101050031
Batch – B4

✸ **Problem Statement : Implement various Election Algorithms for Coordinator Selection in a distributed system.**

> Theory:

1. **Bully Algorithm**
   The Bully Algorithm works by allowing processes with higher IDs to take over as coordinator if they detect the current coordinator has failed.
2. If the coordinator does not respond to it within a time interval T, then it is assumed that coordinator has failed.
3. Now process P sends an election messages to every process with high priority number.
4. It waits for responses, if no one responds for time interval T then process P elects itself as a coordinator.
5. Then it sends a message to all lower priority number processes that it is elected as their new coordinator.
6. However, if an answer is received within time T from any other process Q,
   (I) Process P again waits for time interval T' to receive another message from Q that it has been elected as coordinator.
   (II) If Q doesn't responds within time interval T' then it is assumed to have failed and algorithm is restarted

Code :-

```
// Sarvesh Shingare
// 202101050031
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
class Process
{
public:
    int id;
    bool isAlive;
    Process(int id)
    {
        this->id = id;
        this->isAlive = true;
    }
};
class DistributedSystem
{
public:
    vector<Process> processes;
    int coordinatorId;
    DistributedSystem(vector<int> ids)
    {
        for (int id : ids)
        {
            processes.push_back(Process(id));
        }
        coordinatorId = -1;
    }
    void displayProcesses()
    {
        cout << "Processes: ";
        for (auto &process : processes)
        {
            if (process.isAlive)
            {
                cout << process.id << " ";
            }
        }
        cout << endl;
```

```
}
void crashProcess(int id)
{
    for (auto &process : processes)
    {
        if (process.id == id)
        {
            process.isAlive = false;
            cout << "Process " << id << " crashed." << endl;
            if (coordinatorId == id)
            {
                cout << "Coordinator " << id << " has crashed." << endl;
                startElection();
            }
            break;
        }
    }
}
void startElection()
{
    cout << "Election started." << endl;
    // Identify the process with the highest id that is still alive
    int highestId = -1;
    for (auto &process : processes)
    {
        if (process.isAlive && process.id > highestId)
        {
            highestId = process.id;
        }
    }
    if (highestId != -1)
    {
        coordinatorId = highestId;
        cout << "Process " << highestId << " is elected as the new
                                            coordinator." << endl;
    }
    else
    {
        cout << "No process available for election." << endl;
    }
}
void recoverProcess(int id)
{
```

```
        for (auto &process : processes)
        {
            if (process.id == id)
            {
                process.isAlive = true;
                cout << "Process " << id << " recovered." << endl;
                if (id > coordinatorId)
                {
                    startElection(); // Start election if the recovered process
                    has a higher id
                }
                break;
            }
        }
    }
    void displayCoordinator()
    {
        if (coordinatorId != -1)
        {
            cout << "Current Coordinator: Process " << coordinatorId << endl;
        }
        else
        {
            cout << "No coordinator is selected." << endl;
        }
    }
};
int main()
{
    vector<int> processIds = {1, 2, 3, 4, 5}; // Process IDs in the
    distributed system
        DistributedSystem ds(processIds);
    ds.displayProcesses();
    ds.startElection(); // Initial election
    ds.displayCoordinator();
    ds.crashProcess(5); // Simulate crash of the coordinator
    ds.displayCoordinator();
    ds.recoverProcess(5); // Simulate recovery of a crashed process
    ds.displayCoordinator();
    return 0;
}
```

Output :-

> theory

2. **The Ring Algorithm**

   This algorithm applies to systems organized as a ring(logically or physically). In this algorithm we assume that the link between the process are unidirectional and every process can message to the process on its right only. Data structure that this algorithm uses is active list, a list that has a priority number of all active processes in the system.

   Algorithm –

3. If process P1 detects a coordinator failure, it creates new active list which is empty initially. It sends election message to its neighbour on right and adds number 1 to its active list.
4. If process P2 receives message elect from processes on left, it responds in 3 ways:

   (I) If message received does not contain 1 in active list then P1 adds 2 to its active list and forwards the message.

   (II) If this is the first election message it has received or sent, P1 creates new active list with numbers 1 and 2. It then sends election message 1 followed by 2.

   (III) If Process P1 receives its own election message 1 then active list for P1 now contains numbers of all the active processes in the system. Now Process P1 detects highest priority number from list and elects it as the new coordinator.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
class Process
{
public:
    int id;
    bool isAlive;
    Process(int id)
    {
        this->id = id;
        this->isAlive = true;
    }
};
class RingDistributedSystem
{
public:
    vector<Process> processes;
    int coordinatorId;
    RingDistributedSystem(vector<int> ids)
    {
        for (int id : ids)
        {
            processes.push_back(Process(id));
        }
        coordinatorId = -1;
    }
    void displayProcesses()
    {
        cout << "Processes: ";
        for (auto &process : processes)
        {
            if (process.isAlive)
            {
                cout << process.id << " ";
            }
        }
        cout << endl;
    }
    void crashProcess(int id)
    {
        for (auto &process : processes)
```

```
    {
        if (process.id == id)
        {
            process.isAlive = false;
            cout << "Process " << id << " crashed." << endl;
            if (coordinatorId == id)
            {
                cout << "Coordinator " << id << " has crashed." << endl;
                startElection(id); // Start election by passing message
            }
            break;
        }
    }
}
void startElection(int initiatorId)
{
    cout << "Election started by process " << initiatorId << "." << endl;
    vector<int> electionList;
    // Logical Ring - Collect alive processes in election list
    for (auto &process : processes)
    {
        if (process.isAlive)
        {
            electionList.push_back(process.id);
        }
    }
    if (!electionList.empty())
    {
        // Elect the highest ID as coordinator
        int newCoordinator = *max_element(electionList.begin(),
                                          electionList.end());
        coordinatorId = newCoordinator;
        cout << "Process " << newCoordinator << " is elected as the
                                                new coordinator." << endl;
    }
    else
    {
        cout << "No process available for election." << endl;
    }
}
void recoverProcess(int id)
{
    for (auto &process : processes)
```

```
        {
            if (process.id == id)
            {
                process.isAlive = true;
                cout << "Process " << id << " recovered." << endl;
                if (coordinatorId == -1 || id > coordinatorId)
                {
                    startElection(id);
                }
                break;
            }
        }
    }
    void displayCoordinator()
    {
        if (coordinatorId != -1)
        {
            cout << "Current Coordinator: Process " << coordinatorId << endl;
        }
        else
        {
            cout << "No coordinator is selected." << endl;
        }
    }
};
int main()
{
    vector<int> processIds = {1, 2, 3, 4, 5}; // Process IDs in the
    distributed system
        RingDistributedSystem ds(processIds);
    ds.displayProcesses();
    ds.startElection(1); // Initial election started by process 1
    ds.displayCoordinator();
    ds.crashProcess(5); // Simulate crash of the coordinator
    ds.displayCoordinator();
    ds.recoverProcess(5); // Simulate recovery of a crashed process
    ds.displayCoordinator();
    return 0;
}
```

Output :-

```
PS E:\STUDY\Notes\college\DS\ds4> cd "e:\STUDY\Notes\college\DS\ds4\" ; if ($?) { g++ as.cpp -o as } ; if ($?) { .\as }
Processes: 1 2 3 4 5
Election started by process 1.
Process 5 is elected as the new coordinator.
Current Coordinator: Process 5
Process 5 crashed.
Coordinator 5 has crashed.
Election started by process 5.
Process 4 is elected as the new coordinator.
Current Coordinator: Process 4
Process 5 recovered.
Election started by process 5.
Process 5 is elected as the new coordinator.
Current Coordinator: Process 5
PS E:\STUDY\Notes\college\DS\ds4>
```