Lab 1: Implementation of Election Algorithm: The Bully Algorithm

An algorithm for choosing a unique process to play a particular role is called an election algorithm. For example, in a variant of the central-server algorithm for mutual exclusion, the 'server' is chosen from among the processes p_i, (i = 1, 2,... N)that need to use the critical section. An election algorithm is needed for this choice. It is essential that all the processes agree on the choice. Afterwards, if the process that plays the role of server wishes to retire then another election is required to choose a replacement.

We say that a process calls the election if it takes an action that initiates a particular run of the election algorithm. An individual process does not call more than one election at a time, but in principle the N processes could call N concurrent elections. An important requirement is for the choice of elected process to be unique, even if several processes call elections concurrently. For example, two processes could decide independently that a coordinator process has failed, and both call elections.

Without loss of generality, we require that the elected process be chosen as the one with the largest identifier. The 'identifier' may be any useful value, as long as the identifiers are unique and totally ordered.

The **bully algorithm** allows processes to crash during an election, although it assumes that message delivery between processes is reliable. This algorithm assumes that the system is synchronous: it uses timeouts to detect a process failure. The bully algorithm also assumes that each process knows which processes have higher identifiers, and that it can communicate with all such processes. There are three types of message in this algorithm:

- an **election message** is sent to announce an election;
- an **answer message** is sent in response to an election message and
- a **coordinator message** is sent to announce the identity of the elected process – the new 'coordinator'.

A process begins an election when it notices, through timeouts, that the coordinator has failed. Several processes may discover this concurrently.

Algorithm: The process that knows it has the highest identifier can elect itself as the coordinator simply by sending a **coordinator message** to all processes with lower identifiers. On the other hand, a process with a lower identifier can begin an election by sending an **election message** to those processes that have a higher identifier and awaiting answer messages in response. If none arrives within time **T,** the process considers itself the coordinator and sends a **coordinator message** to all processes with lower identifiers announcing this. Otherwise, the process waits a further period **T`** for a **coordinator message** to arrive from the new coordinator. If none arrives, it begins another election.

If a process $\mathbf{p_i}$ receives a coordinator message, it sets its variable $\mathbf{elected_i}$ to the identifier of the coordinator contained within it and treats that process as the coordinator.

If a process receives an election message, it sends back an answer message and begins another election – unless it has begun one already.

When a process is started to replace a crashed process, it begins an election. If it has the highest process identifier, then it will decide that it is the coordinator and announce this to the other processes. Thus it will become the coordinator, even though the current coordinator is functioning. It is for this reason that the algorithm is called the 'bully' algorithm.

Source Code:

```
//created by Pukar Karki for Distributed Systems Laboratory Exercise 1
//first we include the necessary header files
#include <iostream>
#include <cstdlib>
//we define MAX as the maximum number of processes our program can simulate
#define MAX 20
//we declare list[MAX] to store the list status; 0 for dead and 1 for alive
//we declare n as the number of processes
//we declare coordinator to store the winner of election
int list[MAX], n, coordinator;
using namespace std;
//we declare the necessary functions
void bully();
void display();
int main()
{
  int i,j,fchoice;
  cout<<"Enter number of processes ";
  cin>>n;
  for(i=1;i<=n;i++)
  {
    cout<<"Enter Process "<<i<<" is alive or not(0/1) ";
    cin>>list[i];
    if(list[i])
      coordinator = i;
  }
  display();
  do
  {
    cout<<"1.BULLY ALGORITHM\n2.DISPLAY\n3.EXIT\n";
    cout<<"Enter your choice ";
    cin>>fchoice;
    switch(fchoice)
    {
      case 1:
```

```
        bully();
        break;

      case 2:
        display();
        break;

      case 3:
        exit(1);
        break;
    }
  } while (fchoice != 3);
  return 0;
}

void display()
{
  int i;
  //we display the processes, their status and the coordinator
  cout<<"Processes";
  for(i=1;i<=n;i++)
    cout<<"\t"<<i;
  cout<<endl<<"Alive    ";
  for(i=1;i<=n;i++)
    cout<<"\t"<<list[i];
  cout<<endl<<"COORDINATOR IS "<<coordinator<<endl;
}

void bully()
{
  int schoice, crash, activate, i, gid, flag, subcoordinator;
  do
  {
    cout<<"1.CRASH\n2.ACTIVATE\n3.DISPLAY\n4.EXIT\n";
    cout<<"Enter your choice ";
    cin>>schoice;
    switch (schoice)
    {
    case 1:
      //we manually crash the process to see if our implementation
      //can elect another leader
      cout<<"Enter process to crash ";
```

```
      cin>>crash;
      //if the process is alive then set its status to dead
      if(list[crash])
        list[crash] = 0;
      else
        cout<<"Process "<<crash<<" is already dead!"<<endl;
      do
      {
        //enter another process to initiate the election
        cout<<"Enter election generator id ";
        cin>>gid;
        if(gid == coordinator)
          cout<<"Enter a valid generator id"<<endl;
      } while( gid == coordinator);
      flag = 0;
      //if the coordinator has crashed then we need to find another leader
      if(crash == coordinator)
      {
        //the election generator process will send the message to all higher process
        for(i=gid+1;i<=n;i++)
        {
          cout<<"Message is sent from "<<gid<<" to "<<i<<endl;
          //if the higher process is alive then it will respond
          if(list[i])
          {
            subcoordinator = i;
            cout<<"Response is sent from "<<i<<" to "<<gid<<endl;
            flag = 1;
          }
        }
        //the highest responding process is selected as the leader
        if(flag == 1)
          coordinator = subcoordinator;
        //else if no higher process are alive then the election generator process
        //is selected as leader
        else
          coordinator = gid;
      }
      display();
      break;

    case 2:
```

```
    //enter process to revive
    cout<<"Enter Process ID to be activated ";
    cin>>activate;
    //if the entered process was dead then it is revived
    if(!list[activate])
    {
      list[activate]= 1;
    }
    else
    {
      cout<<"Process "<<activate<<" is already alive!"<<endl;
      break;
    }
    //if the highest process is activated then it is the leader
    if(activate == n)
    {
      coordinator = n;
      break;
    }
    flag = 0;
    //else, the activated process sends message to all higher process
    for(i=activate+1;i<=n;i++)
    {
      cout<<"Message is sent from "<<activate<<" to "<<i<<endl;
      //if higher process is active then it responds
      if(list[i])
      {
        subcoordinator = i;
        cout<<"Response is sent from "<<i<<" to "<<activate;
        flag = 1;
      }
    }
    //the highest responding process is made the leader
    if(flag == 1)
      coordinator = subcoordinator;
    //if no higher process respond then the activated process is leader
    else
      coordinator = activate;
    display();
    break;

  case 3:
```

```
      display();
      break;

    case 4:
      break;
    }
  } while (schoice!=4);
}
```

Output:

```
Enter number of processes 5
Enter Process 1 is alive or not(0/1) 1
Enter Process 2 is alive or not(0/1) 1
Enter Process 3 is alive or not(0/1) 1
Enter Process 4 is alive or not(0/1) 0
Enter Process 5 is alive or not(0/1) 1
Processes       1       2       3       4       5
Alive           1       1       1       0       1
COORDINATOR IS 5
1.BULLY ALGORITHM
2.DISPLAY
3.EXIT
Enter your choice 1
1.CRASH
2.ACTIVATE
3.DISPLAY
4.EXIT
Enter your choice 1
Enter process to crash 5
Enter election generator id 1
Message is sent from 1 to 2
Response is sent from 2 to 1
Message is sent from 1 to 3
Response is sent from 3 to 1
Message is sent from 1 to 4
Message is sent from 1 to 5
```

```
Processes       1       2       3       4       5
Alive           1       1       1       0       0
COORDINATOR IS 3
1.CRASH
2.ACTIVATE
3.DISPLAY
4.EXIT
Enter your choice 2
Enter Process ID to be activated 4
Message is sent from 4 to 5
Processes       1       2       3       4       5
Alive           1       1       1       1       0
COORDINATOR IS 4
1.CRASH
2.ACTIVATE
3.DISPLAY
4.EXIT
Enter your choice 4
1.BULLY ALGORITHM
2.DISPLAY
3.EXIT
Enter your choice 3
PS C:\Users\pukar>
```

Analysis:

Initially, among the five processes, process with ID:5 is the one with the highest ID, so it is selected as a leader. After that, the process with ID:5 crashes and since process with ID:4 is dead, process with ID:3 is selected as leader. After some time, the process with ID:4 activates and calls for election. Since, process with ID:5 is dead, the process with ID:4 is selected as leader.

Conclusion: Hence, in this lab, we successfully simulated the bully algorithm for leader election in distributed system.