Experiment 7

AIM: Write a program to implement Bully Election algorithm Message passing through procedural calls and return values.

THEORY:

Election algorithms choose a process from group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected on other processor. Election algorithm basically determines where a new copy of coordinator should be restarted.

Election algorithm assumes that every active process in the system has a unique priority number. The process with highest priority will be chosen as a new coordinator. Hence, when a coordinator fails, this algorithm elects that active process which has highest priority number. Then this number is send to every active process in the distributed system.

The algorithm assumes that:

- the system is synchronous.
- processes may fail at any time, including during execution of the algorithm.
- a process fails by stopping and returns from failure by restarting.

ALGORITHM

1. Suppose process P sends a message to the coordinator.
2. If coordinator does not respond to it within a time interval T, then it is assumed that coordinator has failed.
3. Now process P sends election message 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,
    a. (I) Process P again waits for time interval T' to receive another message from Q that it has been elected as coordinator.
    b. (II) If Q doesn't responds within time interval T' then it is assumed to have failed and algorithm is restarted. .

SOURCE CODE:

```
#include<bits/stdc++.h>
#include<stdio.h>
#include<stdlib.h>
using namespace std;
struct process
{
 int no;
 int priority;
 int active;
 struct process *next;
};
typedef struct process proc;
struct priority
{
 int pri;
 struct priority *next;
 proc *pp;
};
typedef struct priority pri;

pri* election_confirm(proc *head, pri *head1)
{
 proc *p1;
 pri *p2, *p3;
 p1 = head;
 while (p1->next != head)
 {
 if (p1->active == 1)
 {
 if (head1 == NULL)
 {
 head1 = (pri*) malloc(sizeof(pri));
 head1->pri = p1->priority;
 head1->next = NULL;
 head1->pp = p1;
 p2 = head1;
 }
 else
```

```
 {
 p3 = (pri*) malloc(sizeof(pri));
 p3->pri = p1->priority;
 p3->pp = p1;
 p3->next = NULL;
 p2->next = p3;
 p2 = p2->next;
 }
 p1 = p1->next;
 }
 else
 p1 = p1->next;
 }
 p3 = (pri*) malloc(sizeof(pri));
 p3->pri = p1->priority;
 p3->pp = p1;
 p3->next = NULL;
 p2->next = p3;
 p2 = p2->next;
 p3 = head1;
 return head1;
}

int election_start(pri *head)
{
 pri *p1;
 int max = -1,i = 0;
 p1 = head;
 while (p1 != NULL)
 {
 if (max < p1->pri && p1->pp->active == 1)
 {
 max = p1->pri;
 i = p1->pp->no;
 }
 p1 = p1->next;
 }
 return i;
}
```

```
int main()
{
 proc *head,*p1, *p2;
 int n, i, pr, maxpri, a, pid, max, o;
 char ch;
 head = p1 = p2 = NULL;
 printf("Enter how many process: ");
 scanf("%d", &n);
 for (i = 0; i < n; i++)
 {
if(i!=0)
   printf("\n");
 printf("Enter priority of process %d: ", i + 1);
 scanf("%d", &pr);
 printf("Is process with id %d is active ?(0/1) :", i + 1);
 scanf("%d", &a);
 if (head == NULL)
 {
 head = (proc*) malloc(sizeof(proc));
 if (head == NULL)
 { printf("\nMemory cannot be allocated");
 exit(0);
 }
 head->no = i + 1;
 head->priority = pr;
 head->active = a;
 head->next = head;
 p1 = head;
 }
 else
 {
p2 = (proc*) malloc(sizeof(proc));
 if (p2 == NULL)
 { printf("\nMemory cannot be allocated");
 exit(0);
 }
 p2->no = i + 1;
 p2->priority = pr;
 p2->active = a;
```

```
 p1->next = p2;
 p2->next = head;
 p1 = p2; }
 }
 printf("Enter the process id that invokes election algorithm: ");
 scanf("%d", &pid);
 p2 = head;
 while (p2->next != head)
 {
 if (p2->no == pid)
 {p2 = p2->next;
 break;
 }
 p2 = p2->next;
 }
 election_start(p2);
 printf("\nProcess with id %d has invoked election algorithm", pid);
 printf("\t\nElection message is sent to processes : ");
 while (p2->next != head)
 {
if (p2->no > pid)
 printf("%d, ", p2->no);
 p2 = p2->next;
 }
 printf("%d", p2->no);
 p2 = head;
 max = 0;
 while (1)
 {
if (p2->priority > max && p2->active == 1)
 max = p2->no;
 p2 = p2->next;
 if (p2 == head)
 break;
 }
 printf("\nProcess with the id %d is the co-ordinator\n", max);
return 0;
}
```

OUTPUT:

LEARNING OUTCOMES:

The safety property expected of leader election protocols is that every non-faulty process either elects a process Q, or elects none at all. Note that all processes that elect a leader must decide on the same process Q as the leader. The Bully algorithm satisfies this property (under the system model specified), and at no point in time is it possible for two processes in the group to have a conflicting view of who the leader is, except during an election. This is true because if it weren't, there are two processes X and Y such that both sent the Coordinator (victory) message to the group. This means X and Y must also have sent each other victory messages.