Experiment 8

AIM: Write a program to implement the Ring election algorithm.

THEORY:

Another election algorithm is based on the use of a ring, but without a token. We assume that the processes are physically or logically ordered, so that each process knows who its successor is.
This algorithm applies to systems organized as a ring (logically or physically). In this algorithm we assume that the link between the processes is unidirectional and every process can message to the process on its right only. Data structure that this algorithm uses is an active list, a list that has the priority number of all active processes in the system.
The algorithm assumes that:

- Each process only sends messages to the next process in the ring
- Active list: its info on all other active processes
- Message continues around the ring even if a process along the way has crashed.

ALGORITHM

1. If process P1 detects a coordinator failure, it creates a new active list which is empty initially. It sends an election message to its neighbour on the right and adds number 1 to its active list.
2. If process P2 receives message elect from processes on left, it responds in 3 ways:
    a. (I) If a message received does not contain 1 in the active list then P1 adds 2 to its active list and forwards the message.
    b. (II) If this is the first election message it has received or sent, P1 creates a new active list with numbers 1 and 2. It then sends election message 1 followed by 2.
    c. (III) If Process P1 receives its own election message 1 then the active list for P1 now contains numbers of all the active processes in the system. Now Process P1 detects the highest priority number from the list and elects it as the new coordinator.

SOURCE CODE:

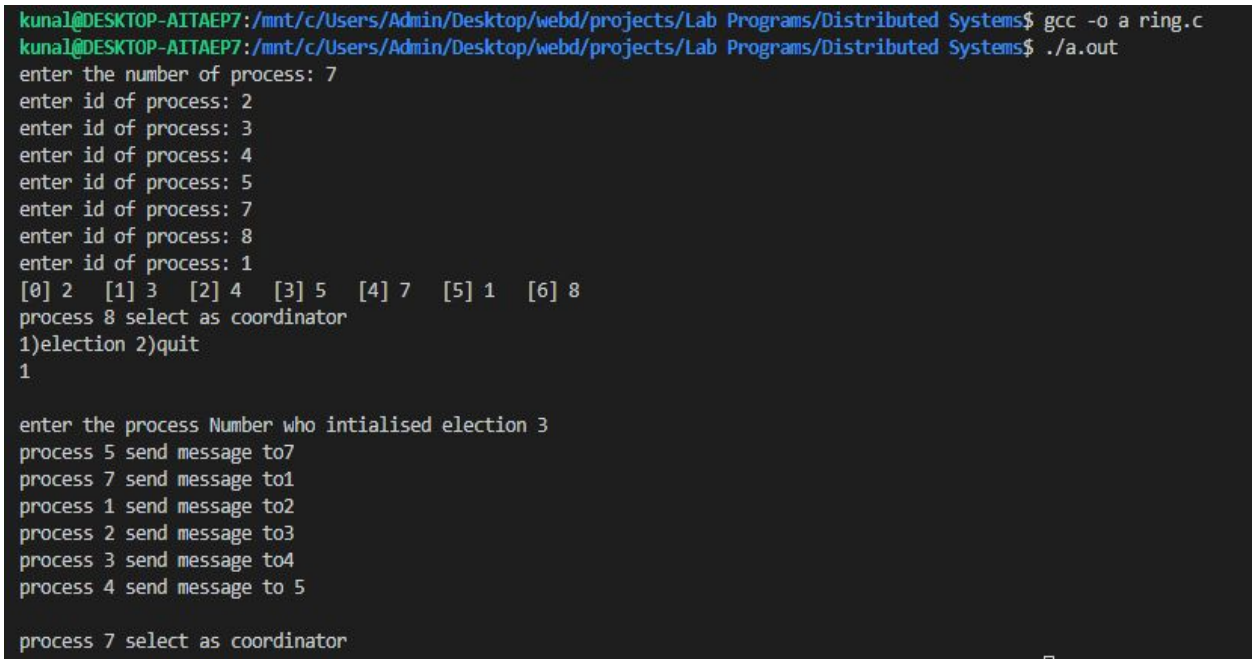
```
#include<string.h>
#include<stdio.h>
#include<stdlib.h>
using namespace std;
struct rr
```

```
{
 int index;
 int id;
 int f;
 char state[10];
}proc[10];
int i, j, k, m, n;
int main()
{
 int temp;
 char str[10];
 cout<<"enter the number of process: ";
 cin>>n;
 for(i=0;i<n;i++)
 {
 cout<<"enter id of process: ";
 cin>>proc[i].id;
 strcpy(proc[i].state,"active");
 proc[i].f=0;
 }
 for(j=0;j<n-1;j++)
 {
 if(proc[j].id>proc[j+1].id)
 {
 temp=proc[j].id;
 proc[j].id=proc[j+1].id;
 proc[j+1].id=temp;
 }
 }
 for(i=0;i<n;i++)
 printf("[%d] %d\t",i,proc[i].id);
 int init, ch, temp1, temp2;
 int arr[10];
 strcpy(proc[n-1].state,"inactive");
 cout<<"\nprocess "<<proc[n-1].id<<" select as coordinator";
 while(1)
 {
 cout<<"\n1)election 2)quit\n";
 scanf("%d",&ch);
```

```
for(i=0;i<n;i++)
proc[i].f=0;
switch(ch)
{
case 1:
cout<<"\nenter the process Number who intialised election";
scanf("%d",&init);
temp2=init;
temp1=init+1;
i=0;
while(temp2!=temp1)
{
if(strcmp(proc[temp1].state,"active")==0 && proc[temp1].f==0 )
{
cout<<"process "<<proc[init].id<<" send message to"<<proc[temp1].id<<"\n";
proc[temp1].f=1;
init=temp1;
arr[i]=proc[temp1].id; i++;
}
if(temp1==n)
temp1=0;
else
temp1++;
}
cout<<"process "<<proc[init].id<<" send message to "<<proc[temp1].id<<"\n";
arr[i]=proc[temp1].id;
i++;
int max=-1;
for(j=0;j<i;j++)
{
if(max<arr[j])
max=arr[j];
}
cout<<"\nprocess "<<max<<" select as coordinator\n";
for(i=0;i<n;i++)
{
if(proc[i].id==max)
strcpy(proc[i].state,"inactive");
}
```

```
 break;
 }
 break;
 }
 return 0;
}
```

OUTPUT:



LEARNING OUTCOMES:

Eventually, the message gets back to the process that started it all. That process recognizes this event when it receives an incoming message containing its own process number. At that point, the message type is changed to COORDINATOR and circulated once again, this time to inform everyone else who the coordinator is (the list member with the highest number) and who the members of the new ring are

Ring Election algorithm is very useful for implementing distributed systems. Here we have successfully implemented the algorithm.