Experiment 9

AIM: Write a program to implement election algorithm for wireless network

THEORY:

Election algorithms choose a process from a group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected on another processor. Election algorithm basically determines where a new copy of the coordinator should be restarted. Election algorithms assume 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 the highest priority number. Then, this number is sent to every active process in the distributed system.

ALGORITHM

1. Any node can initiate the election.
2. When a node receives its first ELECTION message, it makes the sender as its parent.
3. After this it forwards the ELECTION to all its neighbors.
4. If a node already has set its parent, it simply acknowledges.
5. If a node is a leaf it sends its own priority otherwise it waits for its children to finish.
6. When a node has collected all values, it passes it on to its parent

SOURCE CODE:

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <time.h>
#include <string.h>
#define TRUE 1
#define FALSE 0
#define ML 1024
#define MPROC 32
typedef struct wireless_node
```

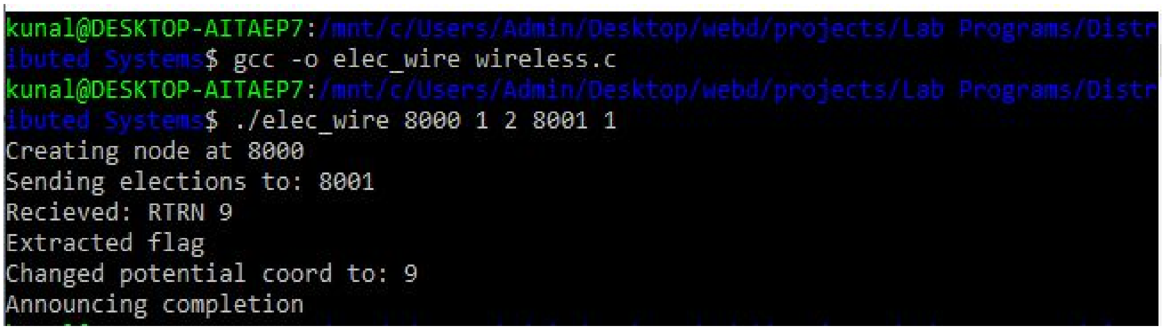
```
{
 int priority;
 int parent;
} wireless_node;
wireless_node w;
int max(int a, int b)
{
 return a >= b ? a : b;
}
int connect_to_port(int connect_to)
{
 int sock_id;
 int opt = 1;
 struct sockaddr_in server;
 if ((sock_id = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
 {
 perror("unable to create a socket");
 exit(EXIT_FAILURE);
 }
 setsockopt(sock_id, SOL_SOCKET, SO_REUSEADDR, (const void *)&opt,
sizeof(int));
 memset(&server, 0, sizeof(server));
 server.sin_family = AF_INET;
 server.sin_addr.s_addr = INADDR_ANY;
 server.sin_port = htons(connect_to);
 if (bind(sock_id, (const struct sockaddr *)&server,
 sizeof(server)) < 0)
 {
 perror("unable to bind to port");
 exit(EXIT_FAILURE);
 }
 return sock_id;
}
void send_to_id(int to, int from, char message[ML])
{
 struct sockaddr_in cl;
 memset(&cl, 0, sizeof(cl));
 cl.sin_family = AF_INET;
 cl.sin_addr.s_addr = INADDR_ANY;
 cl.sin_port = htons(to);
 sendto(
 from,
 (const char *)message,
 strlen(message),
 MSG_CONFIRM,
 (const struct sockaddr *)&cl,
 sizeof(cl));
}
void startElection(int id, int *procs, int num_procs, int self)
{
 int itr;
 char message[ML];
 sprintf(message, "%s %d", "ELEC", self);
 for (itr = 0; itr < num_procs; itr++)
 {
 if (procs[itr] != w.parent)
 {
 printf("Sending elections to: %d\n", procs[itr]);
 send_to_id(procs[itr], id, message);
 }
 }
}
void announce_completion(int self, int *procs, int num_procs, int coord)
{
 int itr;
 char message[ML];
 sprintf(message, "%s %d", "DONE", coord);
 for (itr = 0; itr < num_procs; itr++)
 {
 send_to_id(procs[itr], self, message);
 }
}
void propagate_completion(int self, int *procs, int num_procs, char M[ML])
{
 int itr;
 for (itr = 0; itr < num_procs; itr++)
 {
 send_to_id(procs[itr], self, M);
 }
}
int main(int argc, char *argv[])
{
 int self = atoi(argv[1]);
 int n_procs = atoi(argv[2]);
 int procs[MPROC];
 int sender, pcnt = 0, ecnt = 0;
 int sock_id, coord_id;
 int itr, n, start, ix;
 socklen_t len;
 char buffer[ML], flag[ML], p_id[ML], msg[256];
 struct sockaddr_in from;
 w.priority = atoi(argv[3]);
 w.parent = -1;
 coord_id = w.priority;
 for (itr = 0; itr < n_procs; itr += 1)
 procs[itr] = atoi(argv[4 + itr]);
 start = atoi(argv[4 + n_procs]) == 1 ? TRUE : FALSE;
 printf("Creating node at %d\n", self);
 sock_id = connect_to_port(self);
 if (start == TRUE)
 {
 startElection(sock_id, procs, n_procs, self);
 }
 while (TRUE)
 {
 if (start != TRUE && ecnt + 1 == n_procs)
 {
 sprintf(msg, "RTRN %d", coord_id);
 send_to_id(w.parent,
 sock_id,
msg);
 printf("Sending to parent %d\n", w.parent);
 }
 if (pcnt == n_procs)
 {
 if (start == TRUE)
 {
```

```
printf("Announcing completion\n");
announce_completion(sock_id, procs, n_procs,
coord_id);
exit(1);
}
else
{
sprintf(msg, "RTRN %d", coord_id);
send_to_id(w.parent,
sock_id,
msg);
printf("Sending to parent %d\n", w.parent);
}
}
memset(&from, 0, sizeof(from));
// printf("Tring read\n");
n = recvfrom(sock_id, (char *)buffer, ML, MSG_WAITALL,
(struct sockaddr *)&from, &len);
buffer[n] = '\0';
printf("Recieved: %s\n", buffer);
for (itr = 0; itr < 4; itr++)
{
// printf("%c %d\n", buffer[itr], itr);
flag[itr] = buffer[itr];
}
flag[itr] = '\0';
printf("Extracted flag \n");
if (strcmp(flag, "RTRN") == 0 || strcmp(flag, "DONE") == 0)
{
for (ix = 0, itr = itr + 1; itr < 6; itr++)
p_id[ix++] = buffer[itr];
}
else
{
for (ix = 0, itr = itr + 1; itr < 9; itr++)
p_id[ix++] = buffer[itr];
}
p_id[ix] = '\0';
sender = atoi(p_id);
```

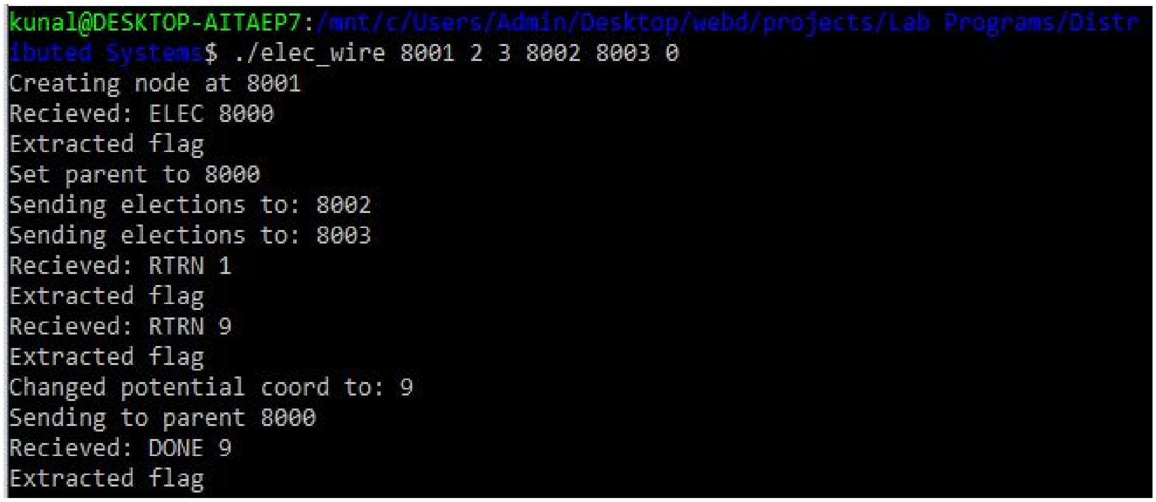
```
// printf("%s %d\n", flag, sender);
if (strcmp(flag, "ELEC") == 0)
{
if (w.parent == -1)
{
w.parent = sender;
printf("Set parent to %d\n", w.parent);
if (n_procs == 1 && procs[0] == w.parent)
pcnt++;
startElection(sock_id, procs, n_procs, self);
}
else
{
printf("Sending EACK to %d\n", sender);
send_to_id(sender, sock_id, "EACK 0000");
}
}
else if (strcmp(flag, "EACK") == 0)
{
ecnt += 1;
continue;
}
else if (strcmp(flag, "RTRN") == 0)
{
pcnt += 1;
if (w.priority < sender)
{
printf("Changed potential coord to: %d\n", sender);
coord_id = sender;
}
else
coord_id = max(coord_id, w.priority);
}
else if (strcmp(flag, "DONE") == 0)
{
if (w.priority != sender)
propagate_completion(sock_id, procs, n_procs,
buffer);
else
```

```
 printf("SET SELF AS CONTROLLER\n");
 exit(1);
 }
 // printf("Waiting\n");
 }
 return 0;
}
```

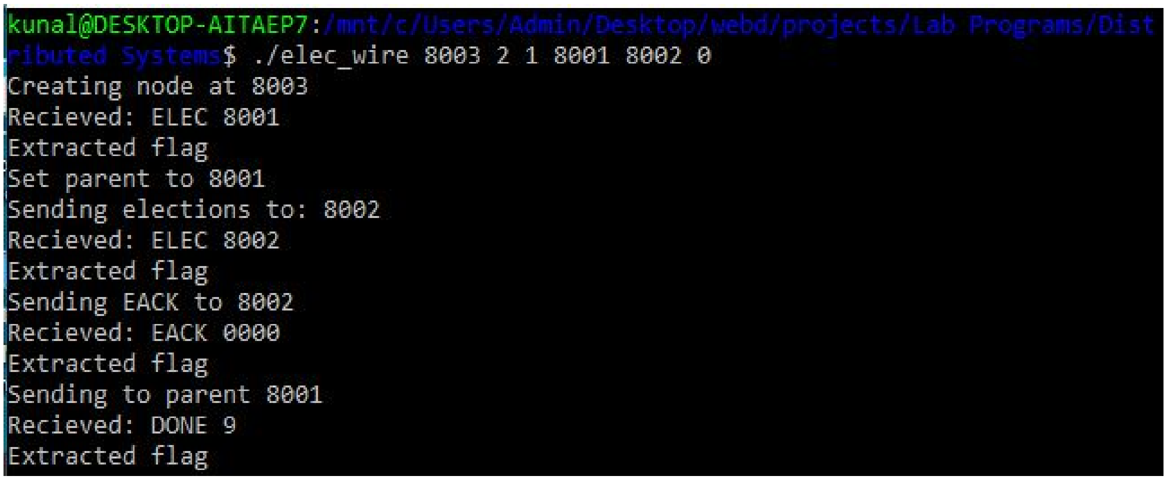
OUTPUT:



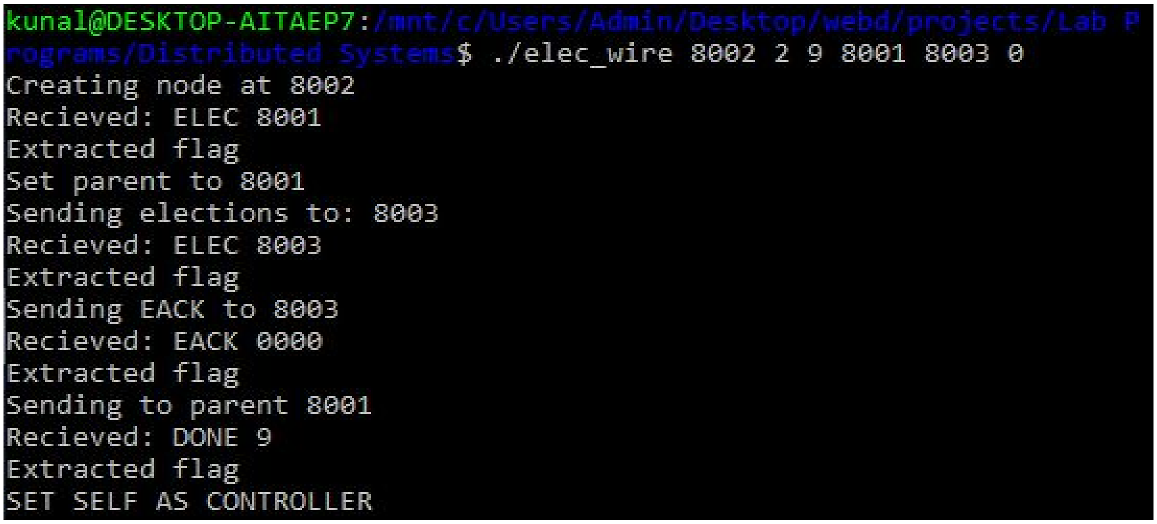
```
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/webd/projects/Lab Programs/Distr
ibuted Systems$ gcc -o elec_wire wireless.c
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/webd/projects/Lab Programs/Distr
ibuted Systems$ ./elec_wire 8000 1 2 8001 1
Creating node at 8000
Sending elections to: 8001
Recieved: RTRN 9
Extracted flag
Changed potential coord to: 9
Announcing completion
```



```
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/webd/projects/Lab Programs/Distr
ibuted Systems$ ./elec_wire 8001 2 3 8002 8003 0
Creating node at 8001
Recieved: ELEC 8000
Extracted flag
Set parent to 8000
Sending elections to: 8002
Sending elections to: 8003
Recieved: RTRN 1
Extracted flag
Recieved: RTRN 9
Extracted flag
Changed potential coord to: 9
Sending to parent 8000
Recieved: DONE 9
Extracted flag
```



```
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/webd/projects/Lab Programs/Dist
ributed Systems$ ./elec_wire 8003 2 1 8001 8002 0
Creating node at 8003
Recieved: ELEC 8001
Extracted flag
Set parent to 8001
Sending elections to: 8002
Recieved: ELEC 8002
Extracted flag
Sending EACK to 8002
Recieved: EACK 0000
Extracted flag
Sending to parent 8001
Recieved: DONE 9
Extracted flag
```



```
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/webd/projects/Lab P
rograms/Distributed Systems$ ./elec_wire 8002 2 9 8001 8003 0
Creating node at 8002
Recieved: ELEC 8001
Extracted flag
Set parent to 8001
Sending elections to: 8003
Recieved: ELEC 8003
Extracted flag
Sending EACK to 8003
Recieved: EACK 0000
Extracted flag
Sending to parent 8001
Recieved: DONE 9
Extracted flag
SET SELF AS CONTROLLER
```

LEARNING OUTCOMES:

We successfully implemented Election in wireless networks. At each point only, the best possible candidate is passed.Once the source gets the results back it can select the coordinator, which it then broadcasts.The messages are tagged with process IDs and in case of multiple ELECTIONS, only the one from a higher pid is entertained.