PROGRAM – 8

Aim: Implement Election Algorithm for Wireless Network.

Theory:

Election Algorithms

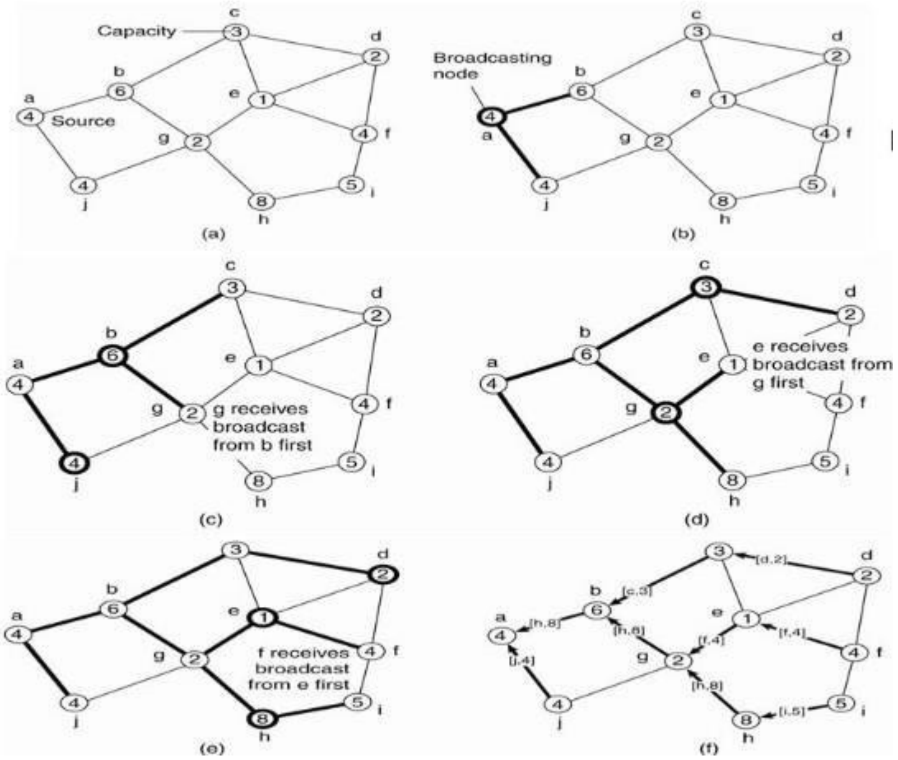
Election algorithms choose a process from group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected on other processor. Election algorithm basically determines where a new copy of coordinator should be restarted. Election algorithm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

Wireless Election process

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 neighbours.
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.

Salient Points

- 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.

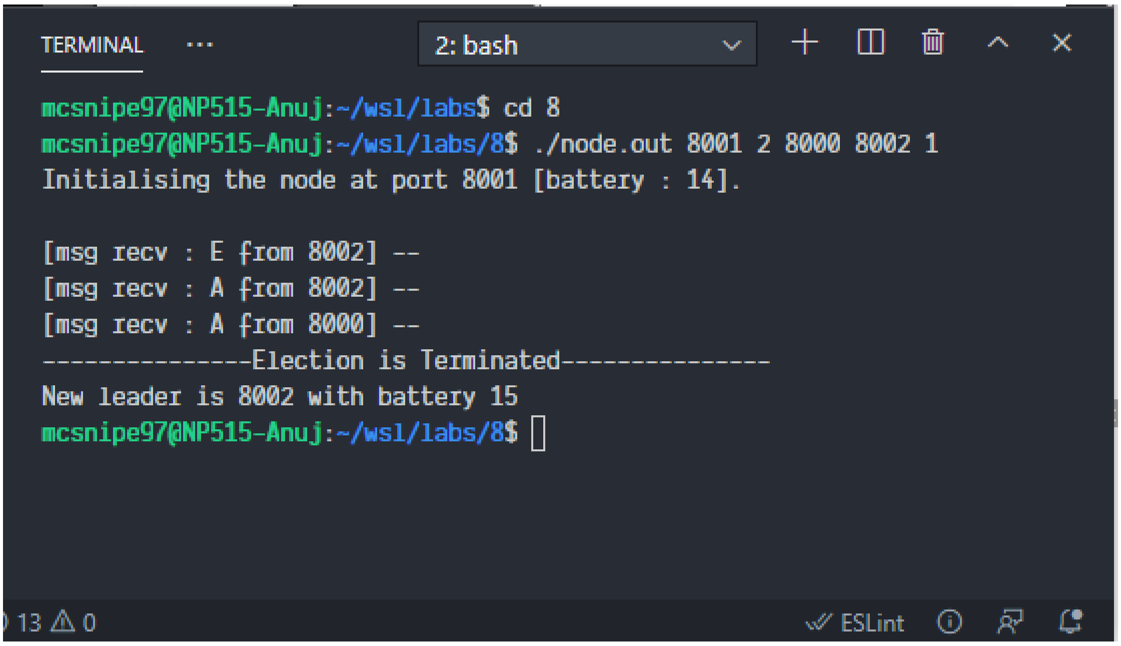


Code:

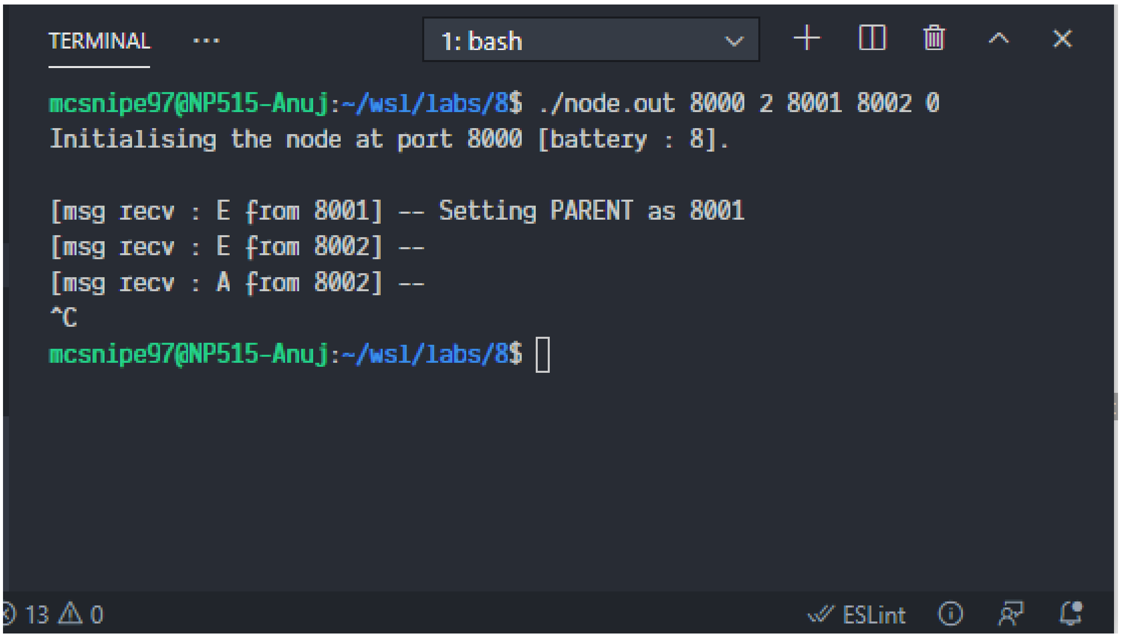
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netdb.h>
#include <time.h>

#define MAXLINE 1024
#define MAX_PROCESS 5

int ELECTION_FLAG = 0;

struct Message
{
    char type;    // E -> Election, A -> Election Ack
    int bestNode; // best node suited for Leader
    int bestNodeScore;
};

struct Candidate
{
    int port;
    int score;
};

int create_connection(int PORT)
{
    int sockfd;
    struct sockaddr_in servaddr;
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }
    int optval = 1;
    setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (const void *)&optval,
sizeof(int));
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET; // IPv4
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(PORT);
    if (bind(sockfd, (const struct sockaddr *)&servaddr,
             sizeof(servaddr)) < 0)
    {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
    return sockfd;
}

void send_message(int dest_port, int sockfd, struct Message newMsg)
{
    sleep(3);
    struct sockaddr_in client_addr;
    memset(&client_addr, 0, sizeof(client_addr));
    client_addr.sin_family = AF_INET; // IPv4
    client_addr.sin_addr.s_addr = INADDR_ANY;
    client_addr.sin_port = htons(dest_port);
    sendto(sockfd, (struct Message *)&newMsg, (1024 + sizeof(newMsg)), 0,
(struct sockaddr *)&client_addr, sizeof(client_addr));
}

void send_elec_ack(int port, int sockfd, struct Candidate bestDescendant)
{
    struct Message newMsg;
    newMsg.type = 'A';
    newMsg.bestNode = bestDescendant.port;
    newMsg.bestNodeScore = bestDescendant.score;
    send_message(port, sockfd, newMsg);
}

int main(int argc, char *argv[])
{

    srand(time(NULL));
    int MY_PORT = atoi(argv[1]);
    int NUM_NEIGHBORS = atoi(argv[2]);
    int NEIGHBORS[MAX_PROCESS];
    for (int i = 0; i < NUM_NEIGHBORS; i++)
        NEIGHBORS[i] = atoi(argv[3 + i]);
    int IS_INITIATOR = atoi(argv[3 + NUM_NEIGHBORS]);
    struct Message *temp = malloc(sizeof(struct Message)), newMsg;
    int myScore = (rand() % 100 + rand() % 100) % 100;
    int sockfd = create_connection(MY_PORT);
    printf("Initialising the node at port %d [battery : %d].\n", MY_PORT,
myScore);
    struct sockaddr_in recv_client_addr, send_client_addr;
    int len = sizeof(struct sockaddr_in), n;
    int PARENT = -1, eackCount = 0;
    struct Candidate bestDescendant = {MY_PORT, myScore};
    if (IS_INITIATOR)
    {
        ELECTION_FLAG = 1;
        newMsg.type = 'E';
        for (int i = 0; i < NUM_NEIGHBORS; i++)
        {
            send_message(NEIGHBORS[i], sockfd, newMsg);
        }
    }
    while (1)
    {
        printf("\n");
        n = recvfrom(sockfd, temp, sizeof(*temp), MSG_WAITALL, (struct
sockaddr *)&recv_client_addr, &len);
        printf("[msg recv : %c from %d] -- ", temp->type,
htons(recv_client_addr.sin_port));
        if (temp->type == 'E')
        {
            if (ELECTION_FLAG)
            {
                send_elec_ack(htons(recv_client_addr.sin_port), sockfd,
bestDescendant);
                continue;
            }
            PARENT = htons(recv_client_addr.sin_port);
            printf("Setting PARENT as %d", PARENT);
            ELECTION_FLAG = 1;
            int IS_LEAF = 1;
            for (int i = 0; i < NUM_NEIGHBORS; i++)
            {
                if (NEIGHBORS[i] != PARENT)
                {
                    newMsg.type = 'E';
                    send_message(NEIGHBORS[i], sockfd, newMsg);
                    IS_LEAF = 0;
                }
            }
            if (IS_LEAF)
            {
                send_elec_ack(htons(recv_client_addr.sin_port), sockfd,
bestDescendant);
            }
        }
        else if (temp->type == 'A')
        {
            eackCount++;
            if (bestDescendant.score < temp->bestNodeScore)
            {
                bestDescendant.score = temp->bestNodeScore;
                bestDescendant.port = temp->bestNode;
            }
            if (IS_INITIATOR)
            {
                if (eackCount == NUM_NEIGHBORS)
                {
                    printf("\n---------------Election is Terminated------
---------\n");
                    printf("New leader is %d with battery %d\n",
bestDescendant.port, bestDescendant.score);
                    exit(0);
                }
            }
            else
            {
                if (eackCount == NUM_NEIGHBORS - 1)
                {
                    send_elec_ack(PARENT, sockfd, bestDescendant);
                    eackCount = 0;
                    ELECTION_FLAG = 0;
                }
            }
        } } }
```

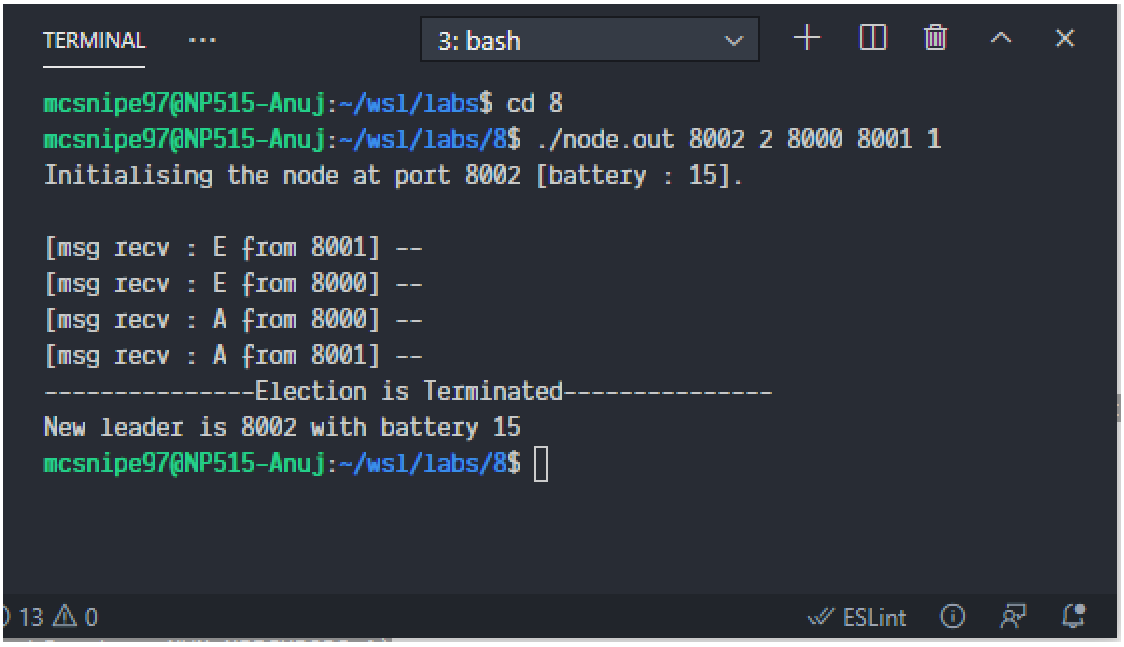
Output:



```
mcsnipe97@NP515-Anuj:~/wsl/labs$ cd 8
mcsnipe97@NP515-Anuj:~/wsl/labs/8$ ./node.out 8001 2 8000 8002 1
Initialising the node at port 8001 [battery : 14].

[msg recv : E from 8002] --
[msg recv : A from 8002] --
[msg recv : A from 8000] --
---------------Election is Terminated---------------
New leader is 8002 with battery 15
mcsnipe97@NP515-Anuj:~/wsl/labs/8$ 
```



```
mcsnipe97@NP515-Anuj:~/wsl/labs/8$ ./node.out 8000 2 8001 8002 0
Initialising the node at port 8000 [battery : 8].

[msg recv : E from 8001] -- Setting PARENT as 8001
[msg recv : E from 8002] --
[msg recv : A from 8002] --
^C
mcsnipe97@NP515-Anuj:~/wsl/labs/8$ 
```



```
mcsnipe97@NP515-Anuj:~/wsl/labs$ cd 8
mcsnipe97@NP515-Anuj:~/wsl/labs/8$ ./node.out 8002 2 8000 8001 1
Initialising the node at port 8002 [battery : 15].

[msg recv : E from 8001] --
[msg recv : E from 8000] --
[msg recv : A from 8000] --
[msg recv : A from 8001] --
---------------Election is Terminated---------------
New leader is 8002 with battery 15
mcsnipe97@NP515-Anuj:~/wsl/labs/8$ 
```

Conclusion:

We successfully implemented election in wireless networks.