**PROGRAM – 8**

**Aim:** Implement Election Algorithm for Wireless Network.

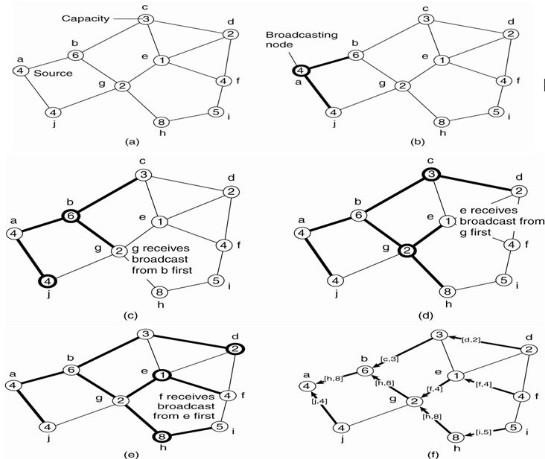
**Theory:**

# 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;

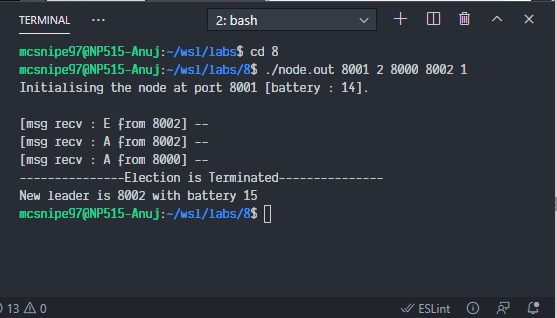
ELECTION\_FLAG = 0;

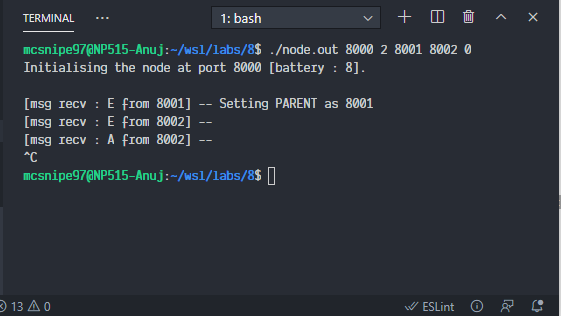
}

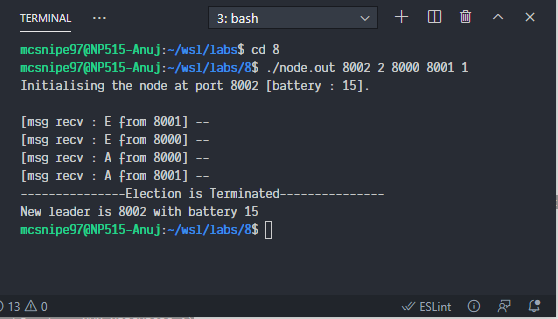
}

} } }

**Output:**



****

****

**Conclusion:**

We successfully implemented election in wireless networks.