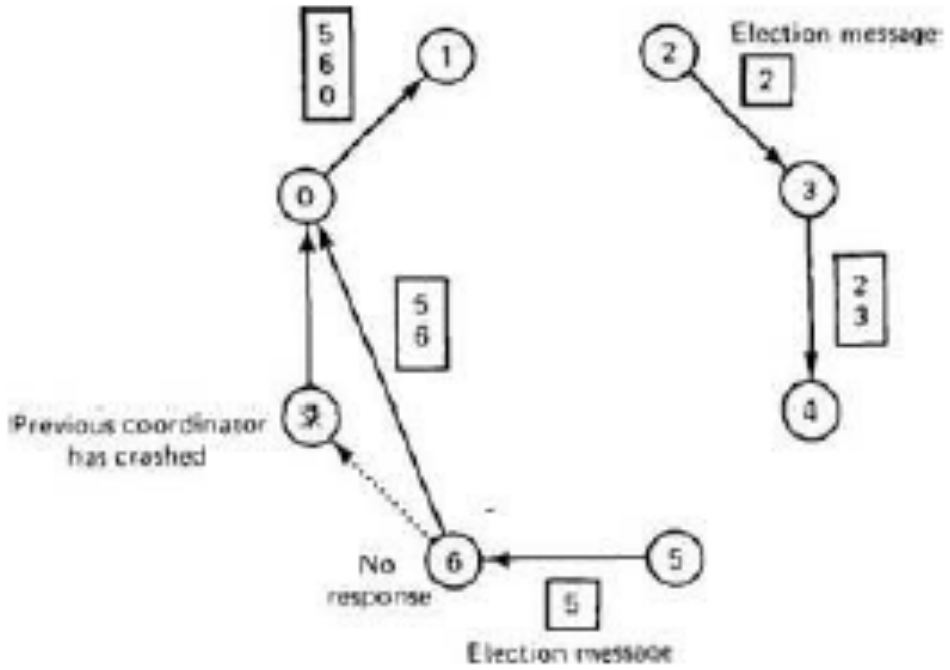
Program – 5

AIM: Implement Ring Election Algorithm

Introduction and 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. When any process notices that the coordinator is not functioning, it builds an ELECTION message containing its own process number and sends the message to its successor. If the successor is down, the sender skips over the successor and goes to the next member along the ring, or the one after that, until a running process is located. At each step, the sender adds its own process number to the list in the message. 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. When this message has circulated once, it is removed and everyone goes back to work.



Program – 5

Code

```c
#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>
#define MSG_CONFIRM 0
#define TRUE 1
#define FALSE 0
#define ML 1024
#define MPROC 32

/*
        Function to create a new connection to port 'connect_to'
        1. Creates the socket.
        2. Binds to port.
        3. Returns socket id
*/

typedef struct lamport_clock{
    int timer;
}lamport_clock;


void init(lamport_clock *clk)
{
    clk->timer = 0;
}

void tick(lamport_clock *clk, int phase)
{
    clk->timer += phase;
}

int str_to_int(char str[ML], int n)
{
    int x = 0, i = 0, k;
    printf("x: %d\n", x);
    for (i = 0; i < n; i++)
    {
        k = atoi(str[i]);
        x = x*10 + k;
    }
    return x;
}

void update_clock(lamport_clock *clk, int new_time)
{
    clk->timer = clk->timer + new_time;
}
```

Program – 5

```

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;
}

/*
        sends a message to port id to
*/

void send_to_id(int to, int id, int diff)
{
        struct sockaddr_in cl;
        memset(&cl, 0, sizeof(cl));
    char message[ML];
    sprintf(message, "%d", diff);
        cl.sin_family = AF_INET;
        cl.sin_addr.s_addr = INADDR_ANY;
        cl.sin_port = htons(to);

        sendto(id, \
                  (const char *)message, \
                   strlen(message), \
                   MSG_CONFIRM, \
                   (const struct sockaddr *)&cl, \
                   sizeof(cl));
}


void send_poll(int to, int id)
{
        struct sockaddr_in cl;
        memset(&cl, 0, sizeof(cl));
    char message[ML];
    sprintf(message, "%s", "POLL");
        cl.sin_family = AF_INET;
```

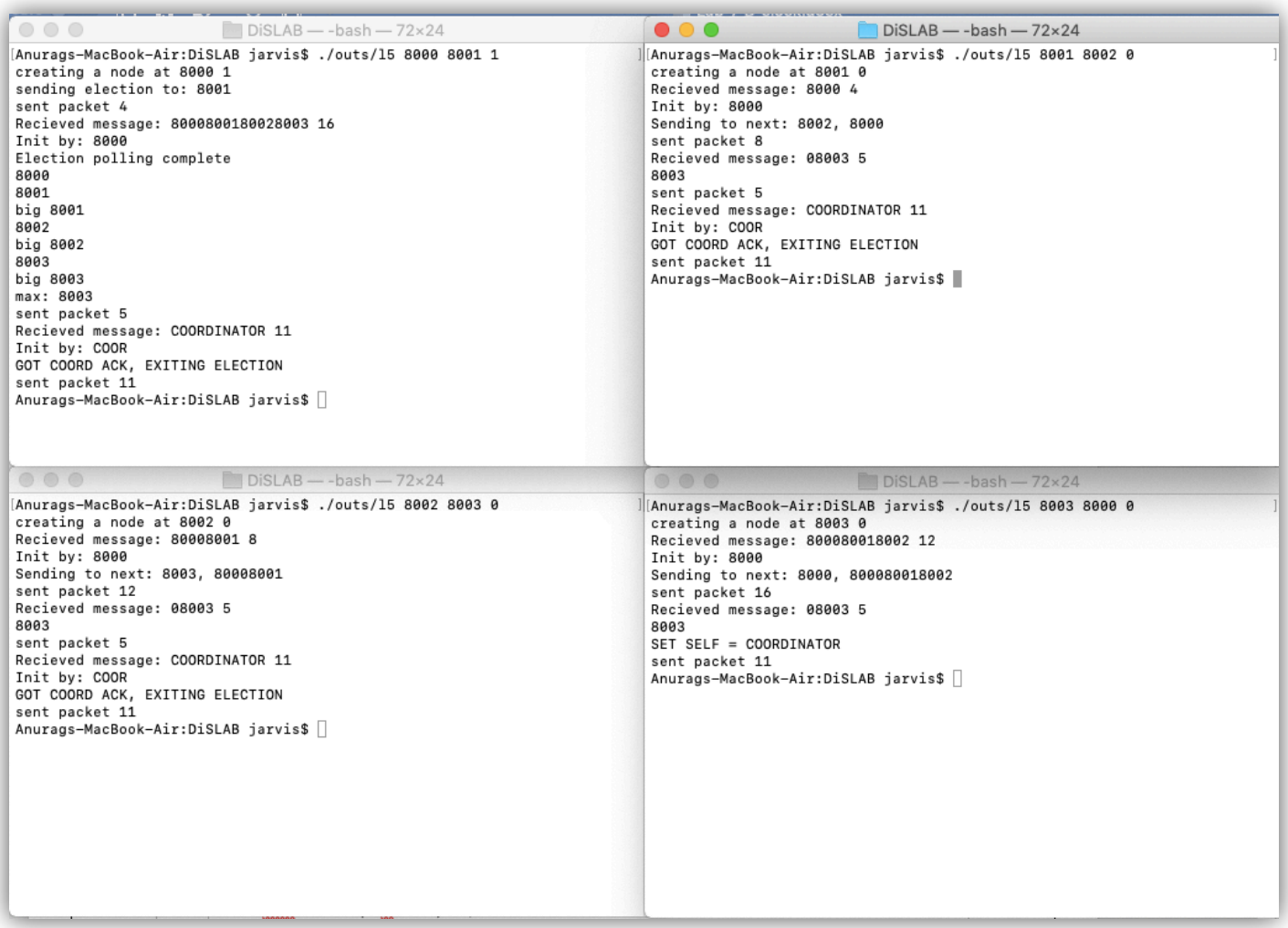
Program – 5

```
        cl.sin_addr.s_addr = INADDR_ANY;
        cl.sin_port = htons(to);

        sendto(id, \
                  (const char *)message, \
                   strlen(message), \
                   MSG_CONFIRM, \
                   (const struct sockaddr *)&cl, \
                   sizeof(cl));
}

/*
        announces completion by sending coord messages
*/

int main(int argc, char* argv[])
{
        // 0. Initialize variables
    int self = atoi(argv[1]);
    int phase = atoi(argv[3]);
        int server = atoi(argv[2]);
    int times[MPROC];
        int sock_id;
    int avg = 0, diff = 0;
    int new_time;
        int itr, len, n, start_at;
        char buff[ML], message[ML];
        struct sockaddr_in from;
    lamport_clock self_clock;
    from.sin_family = AF_INET;
    from.sin_addr.s_addr = htonl(INADDR_ANY);
    init(&self_clock);
    tick(&self_clock, phase);
        // 1. Create socket
        printf("creating a node at %d %d \n", self, start_at);
        sock_id = connect_to_port(self);
        // 3. if not the initiator wait for someone else
        while(TRUE)
        {
         printf("\t - - - - - - - - - - - - - - - - - - - - - - - - - -\n\n");
         sleep(2);
         avg = 0;
         tick(&self_clock, phase);
         memset(&from, 0, sizeof(from));
         n = recvfrom(sock_id, (char *)buff, ML, MSG_WAITALL, (struct sockaddr *)&from, &len);
         buff[n] = '\0';
         if (strcmp(buff, "POLL") == 0)
         {
             printf("Recieved Poll, Sending time to server\n");
             send_to_id(server, sock_id, self_clock.timer);
             printf("Time sent\n");
         }
         else
         {
             new_time = atoi(buff);
```

Program – 5

```
            printf("Got clock corrections: %d, old time %d\n",
new_time, self_clock.timer);
            update_clock(&self_clock, new_time);
            printf("Updated time, new time: %d\n",
self_clock.timer);
            exit(EXIT_SUCCESS);
        }
        printf("\t - - - - - - - - - - - - - - - - - - - - - - - - -
-\n\n");
        }
}
```

Results and Outputs:



Findings and Learnings:

1. We successfully implemented Ring Election Algorithm.