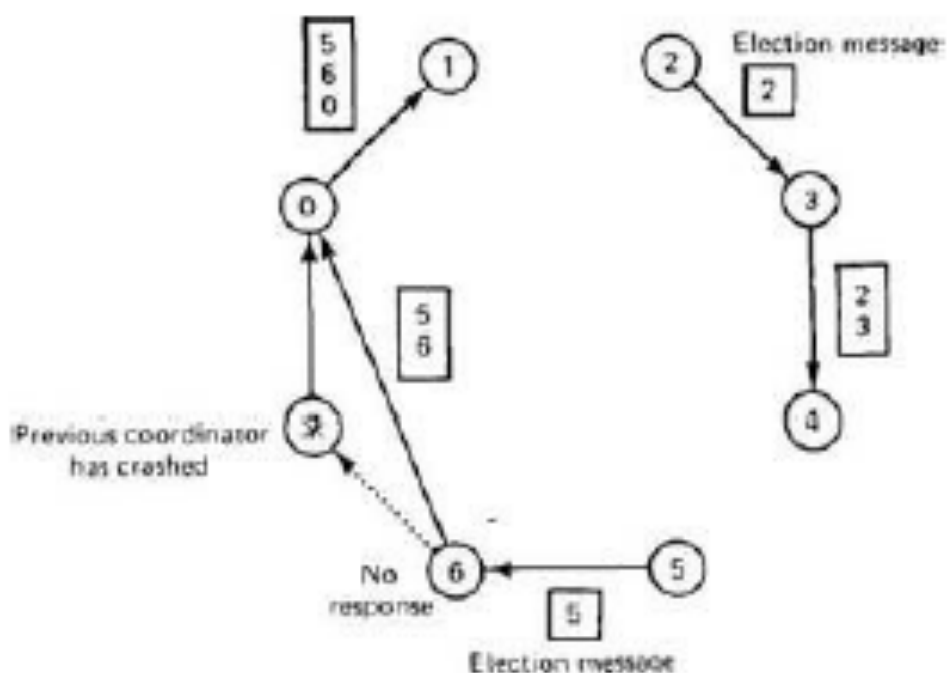


Program – 5

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

```
1  #include <sys/socket.h>
2  #include <netinet/in.h>
3  #include <arpa/inet.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <unistd.h>
7  #include <errno.h>
8  #include <string.h>
9  #include <sys/types.h>
10 #include <time.h>
11 #define MSG_CONFIRM 0
12 #define TRUE 1
13 #define FALSE 0
14 #define ML 1024
15 #define MPROC 32
16
17 /*
18      Function to create a new connection to port 'connect_to'
19      1. Creates the socket.
20      2. Binds to port.
21      3. Returns socket id
22 */
23
24 typedef struct lamport_clock{
25     int timer;
26 }lamport_clock;
27
28
29 void init(lamport_clock *clk)
30 {
31     clk->timer = 0;
32 }
33
34 void tick(lamport_clock *clk, int phase)
35 {
36     clk->timer += phase;
37 }
38
39 int str_to_int(char str[ML], int n)
40 {
41     int x = 0, i = 0, k;
42     printf("x: %d\n", x);
43     for (i = 0; i < n; i++)
44     {
45         k = atoi(str[i]);
46         x = x*10 + k;
47     }
48     return x;
49 }
50
51 void update_clock(lamport_clock *clk, int new_time)
52 {
53     clk->timer = clk->timer + new_time;
54 }
```

Program – 5

```
55
56 int connect_to_port(int connect_to)
57 {
58     int sock_id;
59     int opt = 1;
60     struct sockaddr_in server;
61     if ((sock_id = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
62     {
63         perror("unable to create a socket");
64         exit(EXIT_FAILURE);
65     }
66     setsockopt(sock_id, SOL_SOCKET, SO_REUSEADDR, (const void
67 *)&opt, sizeof(int));
68     memset(&server, 0, sizeof(server));
69     server.sin_family = AF_INET;
70     server.sin_addr.s_addr = INADDR_ANY;
71     server.sin_port = htons(connect_to);
72
73     if (bind(sock_id, (const struct sockaddr *)&server,
74 sizeof(server)) < 0)
75     {
76         perror("unable to bind to port");
77         exit(EXIT_FAILURE);
78     }
79     return sock_id;
80 }
81
82 /*
83     sends a message to port id to
84 */
85
86 void send_to_id(int to, int id, int diff)
87 {
88     struct sockaddr_in cl;
89     memset(&cl, 0, sizeof(cl));
90     char message[ML];
91     sprintf(message, "%d", diff);
92     cl.sin_family = AF_INET;
93     cl.sin_addr.s_addr = INADDR_ANY;
94     cl.sin_port = htons(to);
95
96     sendto(id, \
97         (const char *)message, \
98         strlen(message), \
99         MSG_CONFIRM, \
100         (const struct sockaddr *)&cl, \
101         sizeof(cl));
102 }
103
104
105 void send_poll(int to, int id)
106 {
107     struct sockaddr_in cl;
108     memset(&cl, 0, sizeof(cl));
109     char message[ML];
110     sprintf(message, "%s", "POLL");
111     cl.sin_family = AF_INET;
```

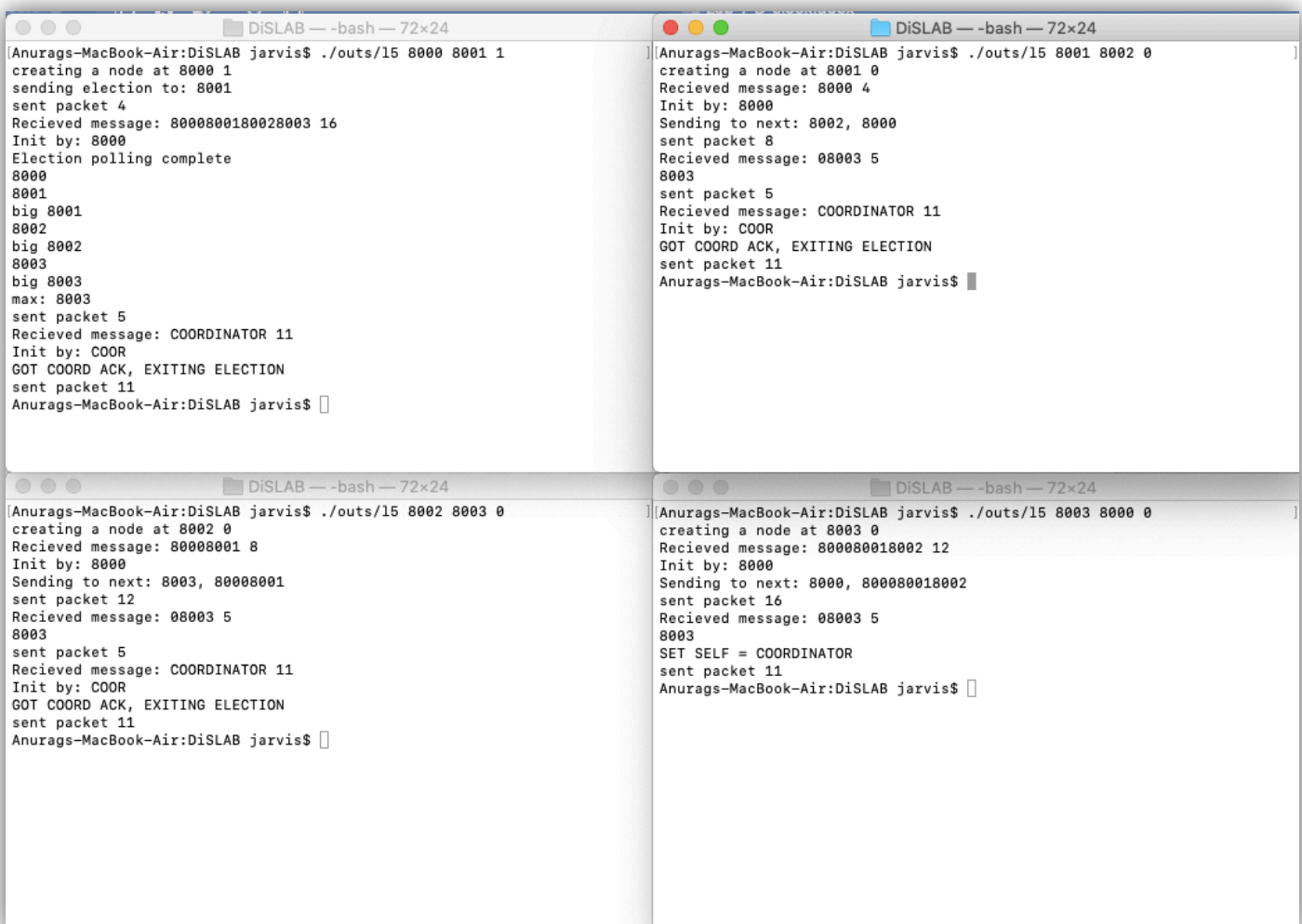
Program – 5

```
112     cl.sin_addr.s_addr = INADDR_ANY;
113     cl.sin_port = htons(to);
114
115     sendto(id, \
116           (const char *)message, \
117           strlen(message), \
118           MSG_CONFIRM, \
119           (const struct sockaddr *)&cl, \
120           sizeof(cl));
121 }
122
123 /*
124     announces completion by sending coord messages
125 */
126
127 int main(int argc, char* argv[])
128 {
129     // 0. Initialize variables
130     int self = atoi(argv[1]);
131     int phase = atoi(argv[3]);
132     int server = atoi(argv[2]);
133     int times[MPROC];
134     int sock_id;
135     int avg = 0, diff = 0;
136     int new_time;
137     int itr, len, n, start_at;
138     char buff[ML], message[ML];
139     struct sockaddr_in from;
140     lamport_clock self_clock;
141     from.sin_family = AF_INET;
142     from.sin_addr.s_addr = htonl(INADDR_ANY);
143     init(&self_clock);
144     tick(&self_clock, phase);
145     // 1. Create socket
146     printf("creating a node at %d %d \n", self, start_at);
147     sock_id = connect_to_port(self);
148     // 3. if not the initiator wait for someone else
149     while(TRUE)
150     {
151         printf("\t - - - - - \n");
152         sleep(2);
153         avg = 0;
154         tick(&self_clock, phase);
155         memset(&from, 0, sizeof(from));
156         n = recvfrom(sock_id, (char *)buff, ML, MSG_WAITALL, (struct
157 sockaddr *)&from, &len);
158         buff[n] = '\0';
159         if (strcmp(buff, "POLL") == 0)
160         {
161             printf("Recieved Poll, Sending time to server\n");
162             send_to_id(server, sock_id, self_clock.timer);
163             printf("Time sent\n");
164         }
165         else
166         {
167             new_time = atoi(buff);
168
```

Program – 5

```
169         printf("Got clock corrections: %d, old time %d\n",
170 new_time, self_clock.timer);
171         update_clock(&self_clock, new_time);
172         printf("Updated time, new time: %d\n",
173 self_clock.timer);
174         exit(EXIT_SUCCESS);
175     }
176     printf("\t - - - - - \n");
177     -\n\n");
178 }
179 }
```

Results and Outputs:



```
Anurags-MacBook-Air:DiSLAB jarvis$ ./outs/15 8000 8001 1
creating a node at 8000 1
sending election to: 8001
sent packet 4
Recieved message: 8000800180028003 16
Init by: 8000
Election polling complete
8000
8001
big 8001
8002
big 8002
8003
big 8003
max: 8003
sent packet 5
Recieved message: COORDINATOR 11
Init by: COOR
GOT COORD ACK, EXITING ELECTION
sent packet 11
Anurags-MacBook-Air:DiSLAB jarvis$
```

```
Anurags-MacBook-Air:DiSLAB jarvis$ ./outs/15 8001 8002 0
creating a node at 8001 0
Recieved message: 8000 4
Init by: 8000
Sending to next: 8002, 8000
sent packet 8
Recieved message: 08003 5
8003
sent packet 5
Recieved message: COORDINATOR 11
Init by: COOR
GOT COORD ACK, EXITING ELECTION
sent packet 11
Anurags-MacBook-Air:DiSLAB jarvis$
```

```
Anurags-MacBook-Air:DiSLAB jarvis$ ./outs/15 8002 8003 0
creating a node at 8002 0
Recieved message: 80008001 8
Init by: 8000
Sending to next: 8003, 80008001
sent packet 12
Recieved message: 08003 5
8003
sent packet 5
Recieved message: COORDINATOR 11
Init by: COOR
GOT COORD ACK, EXITING ELECTION
sent packet 11
Anurags-MacBook-Air:DiSLAB jarvis$
```

```
Anurags-MacBook-Air:DiSLAB jarvis$ ./outs/15 8003 8000 0
creating a node at 8003 0
Recieved message: 800080018002 12
Init by: 8000
Sending to next: 8000, 800080018002
sent packet 16
Recieved message: 08003 5
8003
SET SELF = COORDINATOR
sent packet 11
Anurags-MacBook-Air:DiSLAB jarvis$
```

Findings and Learnings:

1. We successfully implemented Ring Election Algorithm.