

Bully Election algorithm

Program – 4

AIM: Implement Bully Election Algorithm

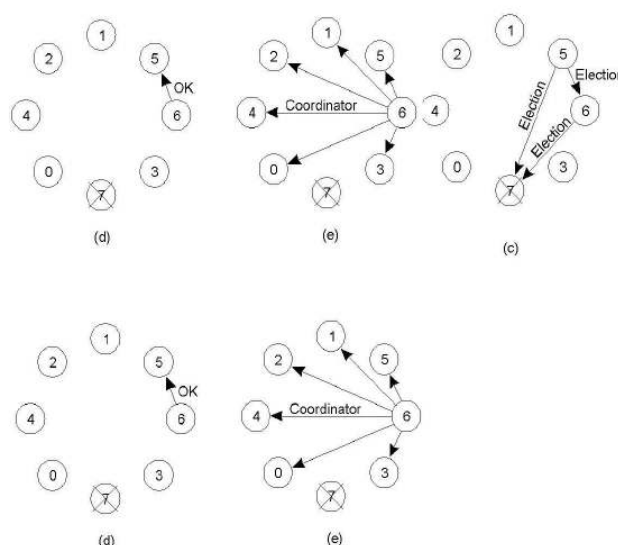
Introduction and Theory

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.

The Bully Election Process

1. P sends a message to the coordinator.
2. If coordinator does not respond to it within a time interval T, then it is assumed that coordinator has failed.
3. Now process P sends election message to every process with high priority number.
4. It waits for responses, if no one responds for time interval T then process P elects itself as a coordinator.
5. Then it sends a message to all lower priority number processes that it is elected as their new coordinator.
6. However, if an answer is received within time T from any other process Q,
 - (I) Process P again waits for time interval T' to receive another message from Q that it has been elected as coordinator.
 - (II) If Q doesn't respond within time interval T' then it is assumed to have failed and algorithm is restarted.



Program – 4

- Disadvantages
 - A large number of messages are sent, this can overload the system.
 - There may be cases in very large systems that multiple coordinators get elected.

Code

Node

```
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     Function to create a new connection to port 'connect_to'
18     1. Creates the socket.
19     2. Binds to port.
20     3. Returns socket id
21 */
22 int connect_to_port(int connect_to)
23 {
24     int sock_id;
25     int opt = 1;
26     struct sockaddr_in server;
27     if ((sock_id = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
28     {
29         perror("unable to create a socket");
30         exit(EXIT_FAILURE);
31     }
32     setsockopt(sock_id, SOL_SOCKET, SO_REUSEADDR, (const void
33 *)&opt, sizeof(int));
34     memset(&server, 0, sizeof(server));
35     server.sin_family = AF_INET;
36     server.sin_addr.s_addr = INADDR_ANY;
37     server.sin_port = htons(connect_to);
38
39     if (bind(sock_id, (const struct sockaddr *)&server,
40 sizeof(server)) < 0)
41     {
42         perror("unable to bind to port");
43         exit(EXIT_FAILURE);
44     }
45     return sock_id;
46 }
47 /*
48     sends a message to port id to
49 */
```

Program – 4

```
50 void send_to_id(int to, int id, char message[ML])
51 {
52     struct sockaddr_in cl;
53     memset(&cl, 0, sizeof(cl));
54
55     cl.sin_family = AF_INET;
56     cl.sin_addr.s_addr = INADDR_ANY;
57     cl.sin_port = htons(to);
58
59     sendto(id, \
60         (const char *)message, \
61         strlen(message), \
62         MSG_CONFIRM, \
63         (const struct sockaddr *)&cl, \
64         sizeof(cl));
65 }
66 /*
67     starts the election, returns 1 if it wins the round
68 */
69 int election(int id, int *procs, int num_procs, int self)
70 {
71     int itr;
72     char message[ML];
73     strcpy(message, "ELECTION");
74     int is_new_coord = 1; // assume you are the winner until you
75 lose
76     for (itr = 0; itr < num_procs; itr += 1)
77     {
78         if (procs[itr] > self)
79         {
80             printf("sending election to: %d\n",
81                 procs[itr]);
82             send_to_id(procs[itr], id, message);
83             is_new_coord = 0; // a proc with id > self
84 exists thus cannot be coord
85         }
86     }
87     return is_new_coord;
88 }
89 /*
90     announces completion by sending coord messages
91 */
92 void announce_completion(int id, int *procs, int num_procs, int
93 self)
94 {
95     int itr;
96     char message[ML];
97     strcpy(message, "COORDINATOR");
98
99     for (itr = 0; itr < num_procs; itr += 1)
100         if (procs[itr] != self)
101             send_to_id(procs[itr], id, message);
102 }
103
104 int main(int argc, char* argv[])
105 {
106     // 0. Initialize variables
```

Program – 4

```
107     int self = atoi(argv[1]);
108     int n_proc = atoi(argv[2]);
109     int procs[MPROC];
110     int sock_id, bully_id;
111     int itr, len, n, start_at;
112     char buff[ML], message[ML];
113     struct sockaddr_in from;
114
115     for (itr = 0; itr < n_proc; itr += 1)
116         procs[itr] = atoi(argv[3 + itr]);
117
118     start_at = atoi(argv[3 + n_proc]) == 1? TRUE : FALSE;
119
120     // 1. Create socket
121     printf("creating a node at %d %d \n", self, start_at);
122     sock_id = connect_to_port(self);
123     // getchar();
124     // 2. check is process is initiator
125
126     if (start_at == TRUE)
127     {
128         election(sock_id, procs, n_proc, self);
129     }
130
131     // 3. if not the initiator wait for someone else
132
133     while(TRUE)
134     {
135         memset(&from, 0, sizeof(from));
136         n = recvfrom(sock_id, (char *)buff, ML, MSG_WAITALL,
137 (struct sockaddr *)&from, &len);
138         buff[n] = '\0';
139         printf("Recieved messed: %s\n", buff);
140
141         if (!strcmp(buff, "ELECTION"))
142         {
143             strcpy(message, "E-ACK"); // send election
144 ack
145             sendto(sock_id,
146                 (const char *)message,
147                 strlen(message),
148                 MSG_CONFIRM,
149                 (const struct sockaddr *)&from,
150                 sizeof(from));
151
152             if (election(sock_id, procs, n_proc, self))
153             {
154                 announce_completion(sock_id, procs,
155 n_proc, self);
156                 printf("ANNOUNCING SELF AS NEW
157 COORD\n");
158             }
159         }
160         else if (!strcmp(buff, "E-ACK"))
161             continue; // nothing do, your job is done
162         else if (!strcmp(buff, "COORDINATOR"))
163             bully_id = from.sin_port;
```

Program – 4

Results and Outputs:

[illegible]

Figure 1 Election result

Findings and Learnings:

1. We successfully implemented Bully-Election Algorithm.