

## Program – 7

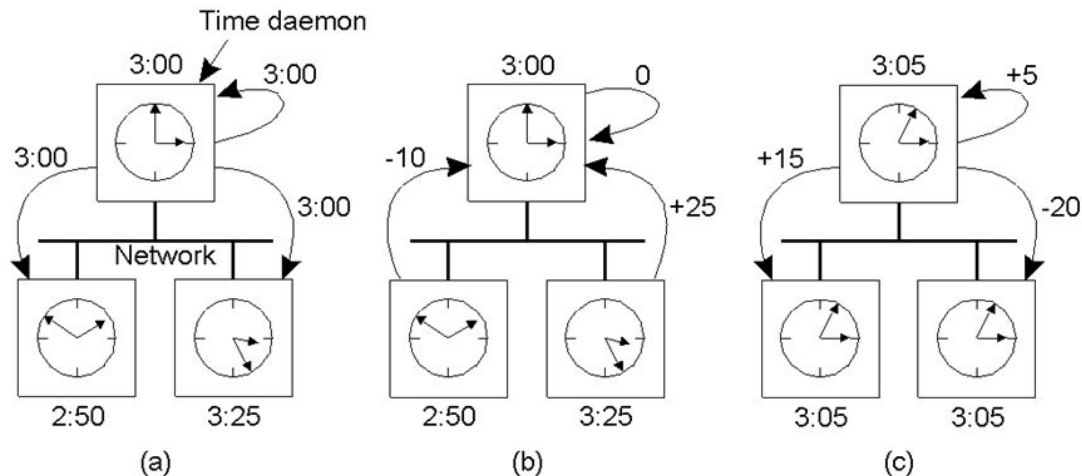
### AIM: Implement Berkeley Clock Synchronization

#### Introduction and Theory

The Berkeley algorithm is a method of clock synchronization in distributed computing which assumes no machine has an accurate time source.

- A master is chosen via an election process such as Chang and Roberts algorithm.
- The master polls the slaves who reply with their time in a similar way to Cristian's algorithm.
- The master observes the round-trip time (RTT) of the messages and estimates the time of each slave and its own.
- The master then averages the clock times, ignoring any values it receives far outside the values of the others.
- Instead of sending the updated current time back to the other process, the master then sends out the amount (positive or negative) that each slave must adjust its clock. This avoids further uncertainty due to RTT at the slave processes.

### The Berkeley Algorithm



- a) The time daemon asks all the other machines for their clock values
- b) The machines answer
- c) The time daemon tells everyone how to adjust their clock

11

# Program – 7

## Code

---

### Server

```
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;
```

## Program – 7

```
54 }
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");
```

## Program – 7

```
111     cl.sin_family = AF_INET;
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 n_proc = atoi(argv[2]);
132     int phase = atoi(argv[3]);
133     int procs[MPROC];
134     int times[MPROC];
135     int sock_id;
136     int avg = 0, diff = 0;
137     int new_time;
138     int itr, len, n, start_at;
139     char buff[ML], message[ML];
140     struct sockaddr_in from;
141     lamport_clock self_clock;
142     from.sin_family = AF_INET;
143     from.sin_addr.s_addr = htonl(INADDR_ANY);
144
145     for (itr = 0; itr < n_proc; itr += 1)
146         procs[itr] = atoi(argv[4 + itr]);
147
148     start_at = 1;
149     init(&self_clock);
150     tick(&self_clock, phase);
151
152     // 1. Create socket
153     printf("creating a node at %d %d \n", self, start_at);
154     sock_id = connect_to_port(self);
155     // getchar();
156     // 2. check is process is initiator
157     if (start_at)
158     {
159         for (itr = 0; itr < n_proc; itr++)
160         {
161             printf("Sending Poll: %d\n", itr);
162             send_poll(procs[itr], sock_id);
163         }
164         printf("POLLING DONE\n");
165     }
166     // 3. if not the initiator wait for someone else
167     while(TRUE)
```

## Program – 7

```
168     {
169         printf("\t - - - - -");
170     -\n\n");
171         sleep(2);
172         avg = 0;
173         tick(&self_clock, phase);
174         for (itr = 0; itr < n_proc; itr++)
175         {
176             memset(&from, 0, sizeof(from));
177             n = recvfrom(sock_id, (char *)buff, ML, MSG_WAITALL,
178 (struct sockaddr *)&from, &len);
179             buff[n] = '\0';
180             getpeername(procs[itr],
181                         (struct sockaddr*)&from, \
182
183 (socklen_t*)&from);
184             printf("Recieved time: %s from %d\n", buff, from);
185             new_time = atoi(buff);
186             times[itr] = new_time;
187             avg += new_time;
188         }
189         avg += self_clock.timer;
190         avg = avg / n_proc + 1;
191         for (itr = 0; itr < n_proc; itr++)
192         {
193             diff = times[itr] - avg;
194             printf("Sending update %d to %d\n", diff,
195 procs[itr]);
196             send_to_id(procs[itr], sock_id, diff);
197         }
198         for (itr = 0; itr < n_proc; itr++)
199         {
200             printf("Sending Poll: %d\n", itr);
201             send_poll(procs[itr], sock_id);
202         }
203         printf("\t - - - - -");
204     -\n\n");
205     }
```

## Client

```
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
```

## Program – 7

```
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 }
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);
```

## Program – 7

```
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;
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     announces completion by sending coord messages
124 */
125 int main(int argc, char* argv[])
126 {
127     // 0. Initialize variables
128     int self = atoi(argv[1]);
```

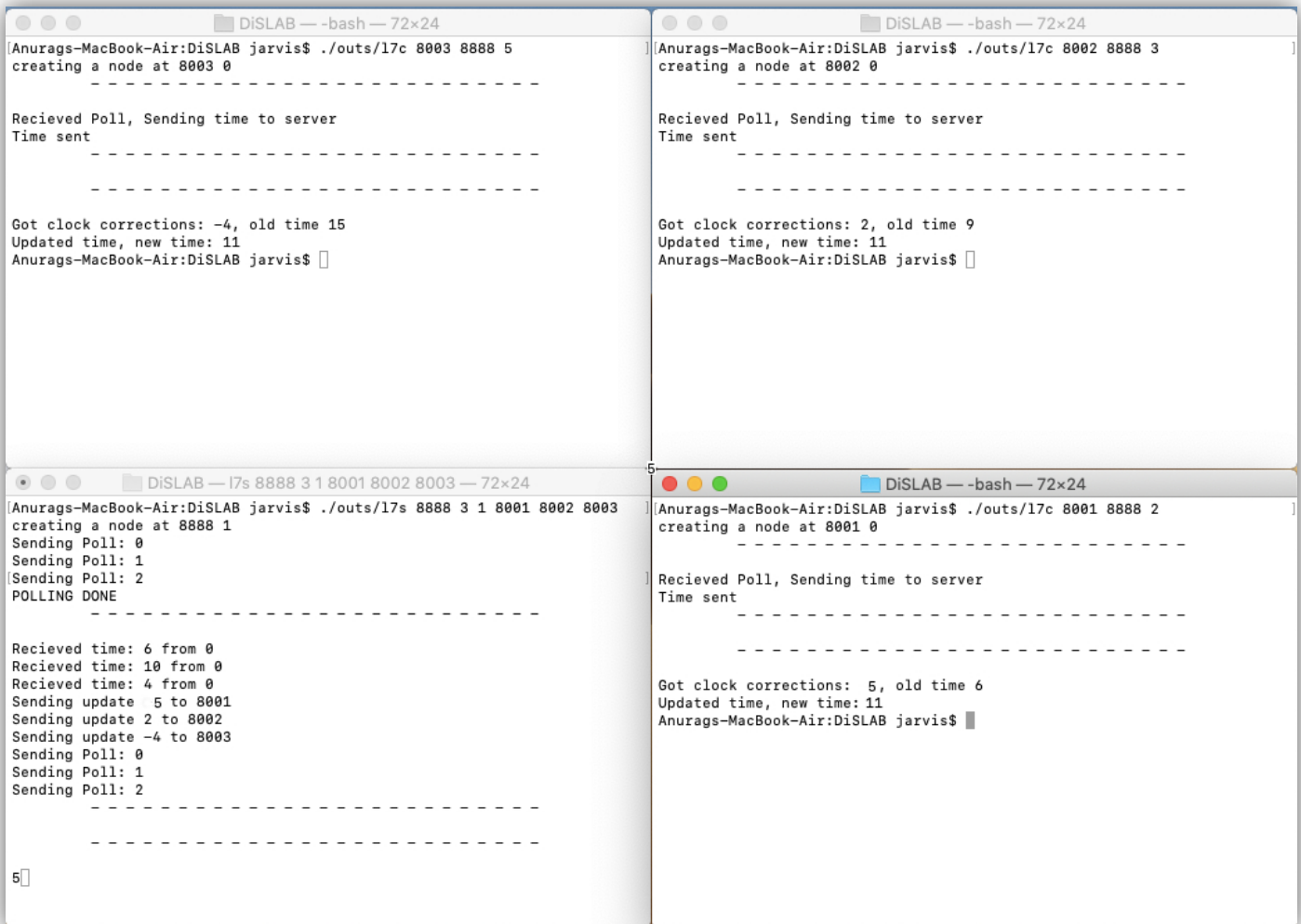
## Program – 7

```
129     int phase = atoi(argv[3]);
130     int server = atoi(argv[2]);
131     int times[MPROC];
132     int sock_id;
133     int avg = 0, diff = 0;
134     int new_time;
135     int itr, len, n, start_at;
136     char buff[ML], message[ML];
137     struct sockaddr_in from;
138     lamport_clock self_clock;
139     from.sin_family = AF_INET;
140     from.sin_addr.s_addr = htonl(INADDR_ANY);
141     init(&self_clock);
142     tick(&self_clock, phase);
143
144     // 1. Create socket
145     printf("creating a node at %d %d \n", self, start_at);
146     sock_id = connect_to_port(self);
147     // 3. if not the initiator wait for someone else
148     while(TRUE)
149     {
150         printf("\t - - - - - \n");
151         sleep(2);
152         avg = 0;
153         tick(&self_clock, phase);
154         memset(&from, 0, sizeof(from));
155         n = recvfrom(sock_id, (char *)buff, ML, MSG_WAITALL, (struct
156 sockaddr *)&from, &len);
157         buff[n] = '\0';
158         if (strcmp(buff, "POLL") == 0)
159         {
160             printf("Recieved Poll, Sending time to server\n");
161             send_to_id(server, sock_id, self_clock.timer);
162             printf("Time sent\n");
163         }
164         else
165         {
166             new_time = atoi(buff);
167             printf("Got clock corrections: %d, old time %d\n",
168 new_time, self_clock.timer);
169             update_clock(&self_clock, new_time);
170             printf("Updated time, new time: %d\n",
171 self_clock.timer);
172             exit(EXIT_SUCCESS);
173         }
174     }
175
176     printf("\t - - - - - \n");
177     -\n\n";
178 }
179 }
```



## Program – 7

### Results and Outputs:



```
DiSLAB -bash- 72x24
Anurags-MacBook-Air:DiSLAB jarvis$ ./outs/l7c 8003 8888 5
creating a node at 8003 0
-----

Recieved Poll, Sending time to server
Time sent
-----

Got clock corrections: -4, old time 15
Updated time, new time: 11
Anurags-MacBook-Air:DiSLAB jarvis$

DiSLAB -bash- 72x24
Anurags-MacBook-Air:DiSLAB jarvis$ ./outs/l7c 8002 8888 3
creating a node at 8002 0
-----

Recieved Poll, Sending time to server
Time sent
-----

Got clock corrections: 2, old time 9
Updated time, new time: 11
Anurags-MacBook-Air:DiSLAB jarvis$

DiSLAB -l7s 8888 3 1 8001 8002 8003 - 72x24
Anurags-MacBook-Air:DiSLAB jarvis$ ./outs/l7s 8888 3 1 8001 8002 8003
creating a node at 8888 1
Sending Poll: 0
Sending Poll: 1
Sending Poll: 2
POLLING DONE
-----

Recieved time: 6 from 0
Recieved time: 10 from 0
Recieved time: 4 from 0
Sending update 5 to 8001
Sending update 2 to 8002
Sending update -4 to 8003
Sending Poll: 0
Sending Poll: 1
Sending Poll: 2
-----

5

DiSLAB -bash- 72x24
Anurags-MacBook-Air:DiSLAB jarvis$ ./outs/l7c 8001 8888 2
creating a node at 8001 0
-----

Recieved Poll, Sending time to server
Time sent
-----

Got clock corrections: 5, old time 6
Updated time, new time: 11
Anurags-MacBook-Air:DiSLAB jarvis$
```

### Findings and Learnings:

1. We successfully implemented Berkeley Algorithm.