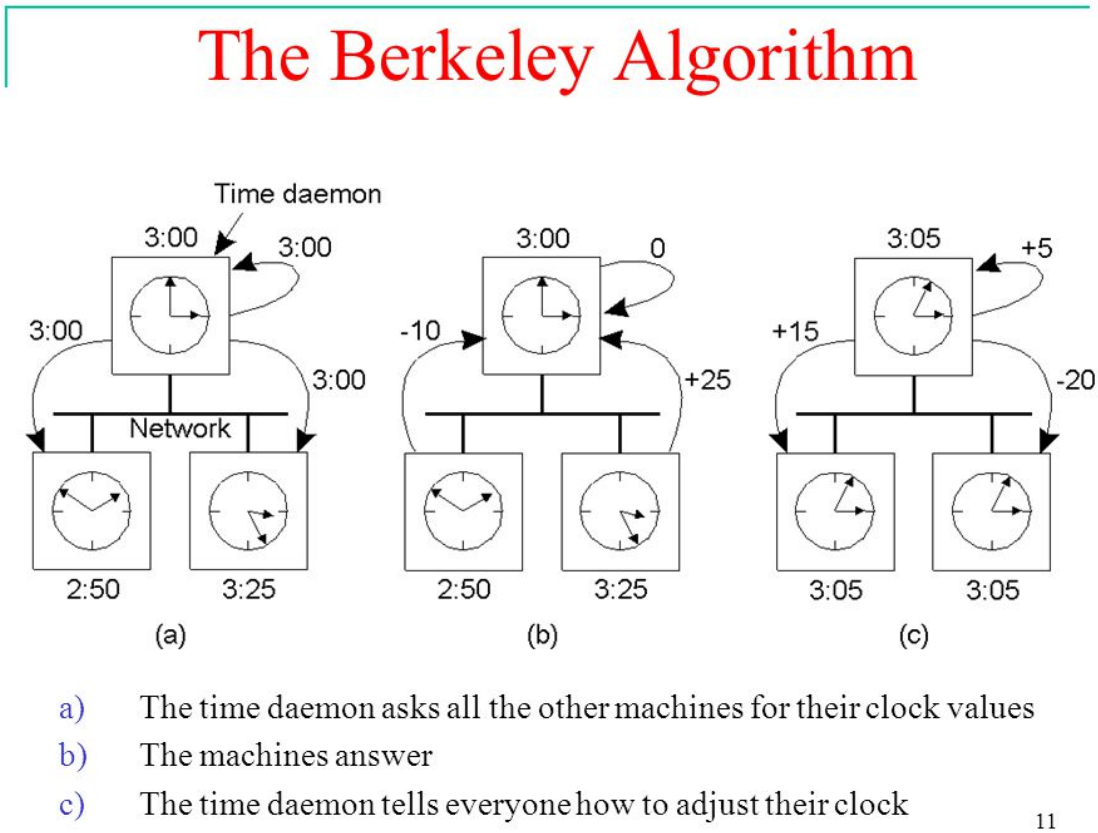
# Program – 7

## 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.



# Program – 7

## Code

**Server**

```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 – 7

```
}

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");
```

# Program – 7

```
        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));
}

/*
        announces completion by sending coord messages
*/

int main(int argc, char* argv[])
{
        // 0. Initialize variables
    int self = atoi(argv[1]);
        int n_proc = atoi(argv[2]);
    int phase = atoi(argv[3]);
        int procs[MPROC];
    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);

        for (itr = 0; itr < n_proc; itr += 1)
                procs[itr] = atoi(argv[4 + itr]);

        start_at = 1;
    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);
        // getchar();
        // 2. check is process is initiator
    if (start_at)
    {
         for (itr = 0; itr < n_proc; itr++)
         {
             printf("Sending Poll: %d\n", itr);
             send_poll(procs[itr], sock_id);
         }
         printf("POLLING DONE\n");
    }
        // 3. if not the initiator wait for someone else
        while(TRUE)
```

# Program – 7

```
        {
        printf("\t - - - - - - - - - - - - - - - - - - - - - - - - - -\n\n");
        sleep(2);
        avg = 0;
        tick(&self_clock, phase);
            for (itr = 0; itr < n_proc; itr++)
            {
                memset(&from, 0, sizeof(from));
                n = recvfrom(sock_id, (char *)buff, ML, MSG_WAITALL, (struct sockaddr *)&from, &len);
                buff[n] = '\0';
                getpeername(procs[itr],
                            (struct sockaddr*)&from, \

(socklen_t*)&from);
                printf("Recieved time: %s from %d\n", buff, from);
                new_time = atoi(buff);
                times[itr] = new_time;
                avg += new_time;
            }
            avg += self_clock.timer;
            avg = avg / n_proc + 1;
            for (itr = 0; itr < n_proc; itr++)
            {
                diff = times[itr] - avg;
                printf("Sending update %d to %d\n", diff, procs[itr]);
                send_to_id(procs[itr], sock_id, diff);
            }
            for (itr = 0; itr < n_proc; itr++)
            {
                printf("Sending Poll: %d\n", itr);
                send_poll(procs[itr], sock_id);
            }
        printf("\t - - - - - - - - - - - - - - - - - - - - - - - - - -\n\n");
        }
}
```

Client

```
#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
```

# Program – 7

```
#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;
}

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);
```

# Program – 7

```

        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;
        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]);
```

# Program – 7

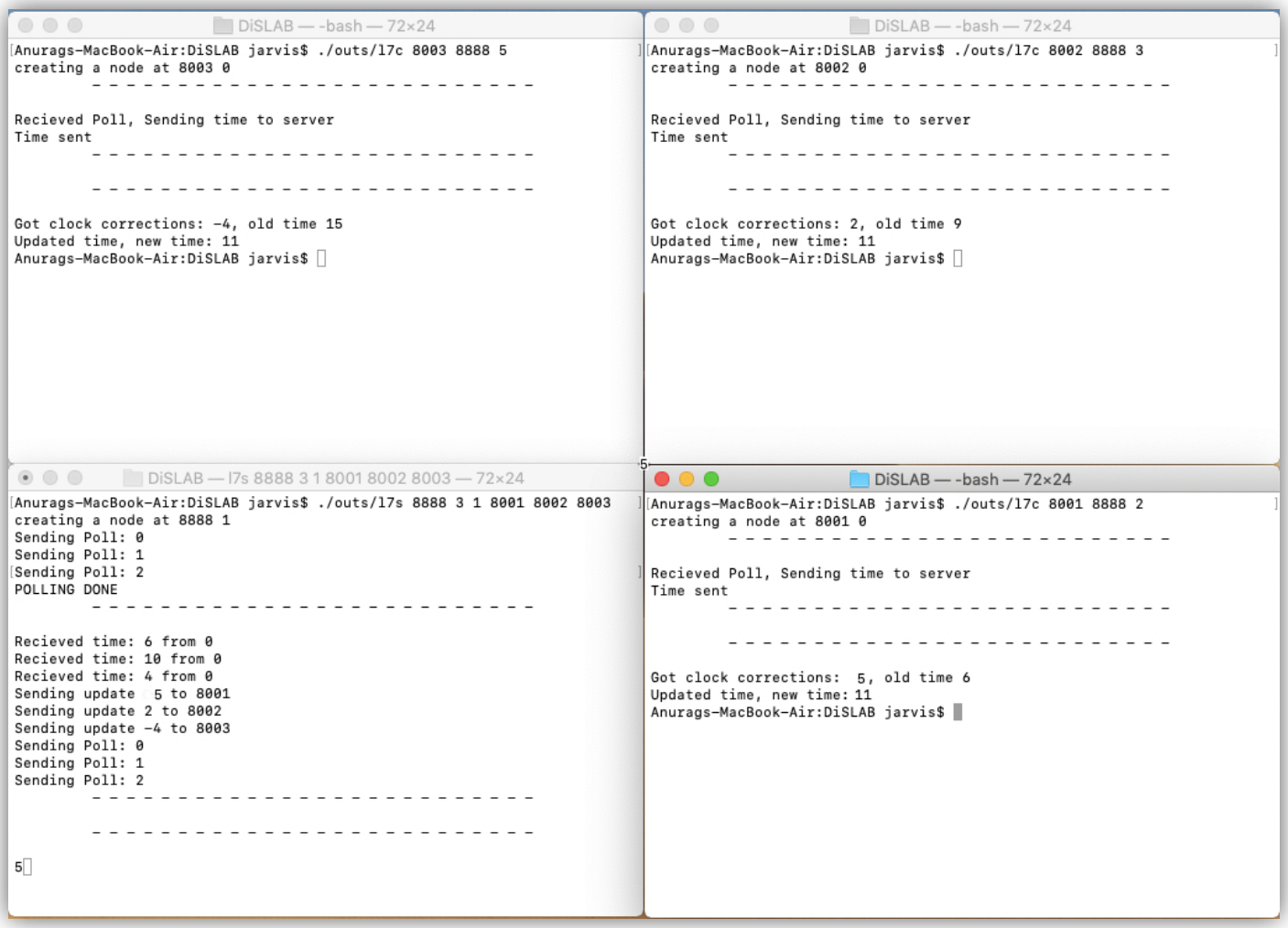
```
    int phase = atoi(argv[3]);
        int server = atoi(argv[2]);
    int times[MPROC];
        int sock_id;
    int avg = 0, diff = 0;
    int new_time;
        int itr, len, n, start_at;
        char buff[ML], message[ML];
        struct sockaddr_in from;
    lamport_clock self_clock;
    from.sin_family = 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);
            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");
        }
}
```

# Program – 7

## Results and Outputs:



```
DiSLAB — -bash — 72×24
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 — 72×24
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 — 72×24
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 — 72×24
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.