**Name: Shivangi Deshmukh**

**Roll No: 20**

**DS Assignment No: 4**

**Title: Implement Berkeley algorithm for clock synchronization.**

**Codes:**

**File name: Server.cpp**

#include <iostream>

#include <iomanip>

#include <cstdlib>

#include <unistd.h>

#include <stdio.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <netinet/in.h>

#include <string.h>

#include <arpa/inet.h>

#include <vector>

#include <cstdlib>

#include <ctime>

#define PORT 8080

using namespace std;

// function for string delimiter

vector<string> split(string s, string delimiter) {

    size\_t pos\_start = 0, pos\_end, delim\_len = delimiter.length();

    string token;

    vector<string> res;

    while ((pos\_end = s.find (delimiter, pos\_start)) != string::npos) {

        token = s.substr (pos\_start, pos\_end - pos\_start);

        pos\_start = pos\_end + delim\_len;

        res.push\_back (token);

    }

    res.push\_back (s.substr (pos\_start));

    return res;

}

int main(int argc, char \*argv[])

{

    // /\* deal with input arguments\*/

    // std::cout << "print arguments:\nargc == " << argc << '\n';

    // for(int ndx{}; ndx != argc; ++ndx) {

    //     std::cout << "argv[" << ndx << "] == " << argv[ndx] << '\n';

    // }

    // std::cout << "argv[" << argc << "] == "

    //           << static\_cast<void\*>(argv[argc]) << '\n';

    srand((unsigned int)time(NULL)); // avoid always same output of rand()

    float server\_local\_clock = rand() % 10; // range from 0 to 9

    vector<float> clients\_local\_clocks;

    printf("Sever starts. Server pid is %d \n", getpid());

    printf("Server local clock is %f \n\n", server\_local\_clock);

    // Socket Cite: https://www.geeksforgeeks.org/socket-programming-cc/?ref=lbp

    int server\_socket\_fd, new\_socket, valread;

    vector<int> client\_sockets;

    vector<string> client\_ips;

    vector<int> client\_ports;

    struct sockaddr\_in server\_address;

    server\_address.sin\_family = AF\_INET;    // IPv4

    server\_address.sin\_addr.s\_addr = INADDR\_ANY; // localhost

    server\_address.sin\_port = htons( PORT ); // 8080

    int opt = 1; // for setsockopt

    // Creating socket file descriptor (IPv4, TCP, IP)

    if ((server\_socket\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0)

    {

        perror("Server: socket failed");

        exit(EXIT\_FAILURE);

    }

    // Optional: it helps in reuse of address and port. Prevents error such as: “address already in use”.

    if (setsockopt(server\_socket\_fd, SOL\_SOCKET, SO\_REUSEADDR | SO\_REUSEPORT,

                                                  &opt, sizeof(opt)))

    {

        perror("Server: setsockopt");

        exit(EXIT\_FAILURE);

    }

    // Forcefully attaching socket to the port 8080

    if (bind(server\_socket\_fd, (struct sockaddr \*)&server\_address,

                                 sizeof(server\_address))<0)

    {

        perror("Server: bind failed");

        exit(EXIT\_FAILURE);

    }

    // Putting the server socket in a passive mode, waiting for the client to approach the server to make a connection

    // The backlog=7, defines the maximum length to which the queue of pending connections for sockfd may grow.

    // If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

    if (listen(server\_socket\_fd, 7) < 0)

    {

        perror("Server: listen");

        exit(EXIT\_FAILURE);

    }

    printf("Server: server is listening ...\n\nYou can open one or multiple new terminal windows now to run ./client\n");

    int clients\_ctr = 0;

    // Setting up buffer for receiving msg

    char recv\_buf[65536];

    memset(recv\_buf, '\0', sizeof(recv\_buf));

    int in\_client\_enough = 0;

    while ( in\_client\_enough == 0) { // block on accept() until positive fd or error

        struct sockaddr\_in client\_addr;

        socklen\_t length = sizeof(client\_addr);

        // Extracting the first connection request on the queue of pending connections for the listening socket (server\_socket\_fd)

        // Creates a new connected socket, and returns a new file descriptor referring to that socket

        if ((new\_socket = accept(server\_socket\_fd, (struct sockaddr \*)&client\_addr,

                        (socklen\_t\*)&length))<0)

        {

            perror("Server: accept");

            exit(EXIT\_FAILURE);

        }

        clients\_ctr ++;

        printf("\nYou have connected %d client(s) now.", clients\_ctr);

        // converting the network address structure src in the af address family into a character string.

        char client\_ip[INET\_ADDRSTRLEN] = "";

        inet\_ntop(AF\_INET, &client\_addr.sin\_addr, client\_ip, INET\_ADDRSTRLEN);

        printf("Server: new client accepted. client ip and port: %s:%d\n", client\_ip, ntohs(client\_addr.sin\_port));

        // store new client connection into array

        client\_sockets.push\_back(new\_socket);

        client\_ips.push\_back(client\_ip);

        client\_ports.push\_back(ntohs(client\_addr.sin\_port));

        printf("current connected clients amount is %d \n", int(client\_sockets.size()) );

        cout << "Do you have enought clients? (please input '1' for yes, '0' for no):" ;

        cin >> in\_client\_enough;

        if (in\_client\_enough == 0){

            cout << "OK. Please continute opening one or multiple new terminal windows to run ./client\n" << endl;

        }else if (in\_client\_enough != 1){

            cout << "Unrecognized input has been considered as 0. You can create one more client.\n" << endl;

            in\_client\_enough = 0;

        }

    }

    printf("\nClients creation finished! There are totally %d connected clients.\n", int(client\_sockets.size()) );

    printf("Asking all clients to report their local clock value ... \n\n\n");

    for (int i = 0; i < client\_sockets.size(); i++){

        // sending a message to client

        const char \*msg = "Hello from server, please tell me your local clock value.";

        send(client\_sockets[i] , msg , strlen(msg) , 0 );

        printf("Server: sent to client(%s:%d): '%s'\n", client\_ips[i].c\_str(), client\_ports[i], msg);

        // receiving

        while(recv(client\_sockets[i], recv\_buf, sizeof(recv\_buf), 0) > 0 ){

            printf("Server: recv from client(%s:%d): '%s' \n", client\_ips[i].c\_str(), client\_ports[i], recv\_buf);

            // convert char array to string

            string recv\_msg = string(recv\_buf);

            if (recv\_msg.find("Hello from client, my local clock value is") != string::npos){

                string substr\_after\_last\_space;

                vector<string> split\_str = split(recv\_msg, " ");

                substr\_after\_last\_space = split\_str[ split\_str.size() - 1 ];

                cout << "Server: received client local clock (string) is " << substr\_after\_last\_space << endl;

                float substr\_after\_last\_space\_f = stof(substr\_after\_last\_space);

                cout << "Server: received client local clock (float) is " << substr\_after\_last\_space\_f << endl;

                clients\_local\_clocks.push\_back(substr\_after\_last\_space\_f);

            }

            memset(recv\_buf, '\0', strlen(recv\_buf));

            break;

        }

    }

    printf("\n\n");

    // average clock values

    float all\_clock\_sum = server\_local\_clock;

    for (int i = 0; i < clients\_local\_clocks.size(); i++){

        all\_clock\_sum += clients\_local\_clocks[i];

    }

    float avg\_clock = all\_clock\_sum / (client\_sockets.size() + 1);

    // tell clients how to adjust

    for (int i = 0; i < client\_sockets.size(); i++){

        // prepare msg

        float offset = clients\_local\_clocks[i] - avg\_clock;

        string operation;

        if (offset >= 0){

            operation = "minus";

        }else{

            operation = "add";

            offset = 0 - offset;

        }

        string msg\_str = "From server, your clock adjustment offset is " + operation + " " + to\_string(offset);

        char msg\_char\_array[msg\_str.length() + 1];

        strcpy(msg\_char\_array, msg\_str.c\_str());

        // sending a message to client

        send(client\_sockets[i] , &msg\_char\_array , strlen(msg\_char\_array) , 0 );

        printf("Server: sent to client(%s:%d): '%s'\n", client\_ips[i].c\_str(), client\_ports[i], msg\_char\_array);

    }

    // adjust self

    server\_local\_clock += avg\_clock - server\_local\_clock;

    printf("\n\nServer new local clock is %f \n\n", server\_local\_clock);

    printf("Server: server stopped. \n");

    close(server\_socket\_fd);

    return 0;

}

**File name: Client.cpp**

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

#include <iostream>

#include <stdlib.h>     /\* srand, rand \*/

#include <cstdlib>

#include <ctime>

#include <vector>

#define PORT 8080

using namespace std;

// function for string delimiter

vector<string> split(string s, string delimiter) {

    size\_t pos\_start = 0, pos\_end, delim\_len = delimiter.length();

    string token;

    vector<string> res;

    while ((pos\_end = s.find (delimiter, pos\_start)) != string::npos) {

        token = s.substr (pos\_start, pos\_end - pos\_start);

        pos\_start = pos\_end + delim\_len;

        res.push\_back (token);

    }

    res.push\_back (s.substr (pos\_start));

    return res;

}

int main(int argc, char const \*argv[])

{

    srand((unsigned int)time(NULL)); // avoid always same output of rand()

    float client\_local\_clock = rand() % 10; // range from 0 to 9

    printf("Client starts. Client pid is %d \n", getpid());

    printf("Client local clock is %f \n\n", client\_local\_clock);

    int client\_socket\_fd, valread;

    char client\_read\_buffer[1024] = {0};

    struct sockaddr\_in server\_addr;

    server\_addr.sin\_family = AF\_INET;

    // server\_addr.sin\_addr.s\_addr = inet\_addr(argv[1]); // hardcode to 127.0.0.1

    server\_addr.sin\_port = htons(PORT);

    // Creating socket file descriptor (IPv4, TCP, IP)

    if ((client\_socket\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

    {

        printf("\n Client: Socket creation error \n");

        return -1;

    }

    // Converting IPv4 and IPv6 addresses from text to binary form,

    //   from character string src into a network

    //   address structure in the af address family, then copies the

    //   network address structure to dst.

    if(inet\_pton(AF\_INET, "127.0.0.1", &server\_addr.sin\_addr)<=0)

    {

        printf("\nClient: Invalid address/ Address not supported \n");

        return -1;

    }

    // Connecting server, return 0 with success, return -1 with error

    if (connect(client\_socket\_fd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0)

    {

        printf("\nClient: Connection Failed \n");

        return -1;

    }

    char server\_ip[INET\_ADDRSTRLEN]="";

    inet\_ntop(AF\_INET, &server\_addr.sin\_addr, server\_ip, INET\_ADDRSTRLEN);

    printf("Client: connected server(%s:%d). \n", server\_ip, ntohs(server\_addr.sin\_port));

    printf("\n\n");

    //

    // first round communicattion

    //

    // receiving form server

    valread = read( client\_socket\_fd , client\_read\_buffer, 1024);

    printf("Client: read: '%s'\n",client\_read\_buffer );

    // convert char array to string

    string recv\_msg = string(client\_read\_buffer);

    // reply according to what client receive

    if (strcmp(client\_read\_buffer, "Hello from server, please tell me your local clock value.") == 0) {

        // prepare msg

        string msg\_str = "Hello from client, my local clock value is " + to\_string(client\_local\_clock);

        char msg\_char\_array[msg\_str.length() + 1];

        strcpy(msg\_char\_array, msg\_str.c\_str());

        // sending a message to server

        send(client\_socket\_fd , &msg\_char\_array , strlen(msg\_char\_array) , 0 );

        printf("Client: sent message: '%s'\n", msg\_char\_array);

    }

    //

    // second round communicattion

    //

    // receiving form server

    valread = read( client\_socket\_fd , client\_read\_buffer, 1024);

    printf("Client: read: '%s'\n",client\_read\_buffer );

    // convert char array to string

    recv\_msg = string(client\_read\_buffer);

    if (recv\_msg.find("From server, your clock adjustment offset is") != string::npos){ // if latter is a substring of former

        string substr\_after\_lastbutone\_space;

        string substr\_after\_last\_space;

        vector<string> split\_str = split(recv\_msg, " ");

        substr\_after\_lastbutone\_space = split\_str[ split\_str.size() - 2 ];

        substr\_after\_last\_space = split\_str[ split\_str.size() - 1 ];

        cout << "Client: received local clock adjustment offset (string) is " << substr\_after\_lastbutone\_space << " " << substr\_after\_last\_space << endl;

        float substr\_after\_last\_space\_f = stof(substr\_after\_last\_space);

        cout << "Client: received local clock adjustment offset (float) is " << substr\_after\_lastbutone\_space << " " << substr\_after\_last\_space\_f << endl;

        char oper\_char\_array[substr\_after\_lastbutone\_space.length() + 1];

        strcpy(oper\_char\_array, substr\_after\_lastbutone\_space.c\_str());

        if (strcmp(oper\_char\_array, "add") == 0 ){

            client\_local\_clock += substr\_after\_last\_space\_f;

        }else if (strcmp(oper\_char\_array, "minus") == 0 ){

            client\_local\_clock -= substr\_after\_last\_space\_f;

        }

        printf("Client local clock is %f \n\n", client\_local\_clock);

    }

    close(client\_socket\_fd);

    return 0;

}

**Output**

**Terminal Output 1**

student@student-HP-Pro-3330-MT:~$ cd /documents/berk  
student@student-HP-Pro-3330-MT:~$ cd berk  
student@student-HP-Pro-3330-MT:~/berk$ make  
g++ server.cpp -o server -std=c++11  
g++ client.cpp -o client -std=c++11  
  
student@student-HP-Pro-3330-MT:~/berk$  
student@student-HP-Pro-3330-MT:~/berk$ ./server  
Sever starts. Server pid is 6329  
Server local clock is 3.000000  
  
Server: server is listening ...  
  
You can open one or multiple new terminal windows now to run ./client  
  
You have connected 1 client(s) now.Server: new client accepted. client ip and port: [127.0.0.1:54186](http://127.0.0.1:54186/#inbox/_blank)  
current connected clients amount is 1  
Do you have enought clients? (please input '1' for yes, '0' for no):1  
  
Clients creation finished! There are totally 1 connected clients.  
Asking all clients to report their local clock value …

**Terminal Output 2**

student@student-HP-Pro-3330-MT:~$ ./client  
bash: ./client: No such file or directory  
student@student-HP-Pro-3330-MT:~$ cd berk  
student@student-HP-Pro-3330-MT:~/berk$ ./client  
Client starts. Client pid is 6348  
Client local clock is 2.000000  
  
Client: connected server([127.0.0.1:8080](http://127.0.0.1:8080/#inbox/_blank)).  
  
  
Client: read: 'Hello from server, please tell me your local clock value.'  
Client: sent message: 'Hello from client, my local clock value is 2.000000'  
Client: read: 'From server, your clock adjustment offset is add 0.500000'  
Client: received local clock adjustment offset (string) is add 0.500000  
Client: received local clock adjustment offset (float) is add 0.5  
Client local clock is 2.500000