//Client

```
#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;
}
```

//Server

```
#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;
}
```

- **OUTPUT :**