Assignment №1

1 Simple Messanger

You need to develop the simplest instant messaging system (chat, messenger)

1.1 Server

Must receive messages from clients and send messages to all connected clients. Use threads to work with multiple clients. The name of the binary file being executed must be **server**. The server must have a console interface in the parameters to which is transferred: port

1.2 Client

Each client must have own nickname, set by the user. When you receive a message from another client, the screen should display the time of receiving the message, the user-sender's nickname, and the text of the message. An example: {05:20} [John] Hi!

The client must have a console interface. The name of the **client** for binary file. The client accepts the work options through the command line arguments in the following order: server address, port, nickname.

To prevent the received (incoming) messages from interfering with the user's typing, it is suggested that a separate mode of sending a message, for example, when the **m** key is pressed, the user enters his message, new messages from other users are temporarily not displayed, after sending the message (by Enter) the mode is automatically turned off.

1.3 Protocol

Client -> Server

Nickname size	**Nick**	**Body size**	**Body**
4 bytes	Nickname size bytes	4 bytes	Body size bytes
Network format	-	Network format	-

Server -> Client

Nickname size	**Nick**	**Body size**	**Body**	**Date size**	**Date**
4 bytes	Nickname size bytes	4 bytes	Body size bytes	4 bytes	Data size bytes
Network format	-	Network format	-	Network format	-

1.4 Features that are not required of the server and the client

- Client registration, authorization, authentication
- More than one channel of communication
- Keeping your correspondence history
- Processing time zones
- Working with languages other than English

1.5 Requirements

- The server and the client must be written in the language of the C
- Make should be used as an build system
- The code must be successfully compiled and worked for Linux and MacOS
- Valgrind and Google Thread Sanitizer should not find errors in the code

server.c

```c
#include <stdio.h>
#include <stdlib.h>

#include <netdb.h>
#include <netinet/in.h>
#include <unistd.h>

#include <string.h>

int main(int argc, char *argv[]) {
    int sockfd, newsockfd;
    uint16_t portno;
    unsigned int clilen;
    char buffer[256];
    struct sockaddr_in serv_addr, cli_addr;
    ssize_t n;

    /* First call to socket() function */
    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    if (sockfd < 0) {
        perror("ERROR opening socket");
        exit(1);
    }

    /* Initialize socket structure */
    bzero((char *) &serv_addr, sizeof(serv_addr));
    portno = 5001;

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);

    /* Now bind the host address using bind() call.*/
    if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
        perror("ERROR on binding");
        exit(1);
    }

    /* Now start listening for the clients, here process will
       * go in sleep mode and will wait for the incoming connection
    */

    listen(sockfd, 5);
    clilen = sizeof(cli_addr);

    /* Accept actual connection from the client */
    newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);

    if (newsockfd < 0) {
        perror("ERROR on accept");
        exit(1);
    }

    /* If connection is established then start communicating */
    bzero(buffer, 256);
    n = read(newsockfd, buffer, 255);

    if (n < 0) {
        perror("ERROR reading from socket");
        exit(1);
    }

    printf("Here is the message: %s\n", buffer);

    /* Write a response to the client */
    n = write(newsockfd, "I got your message", 18);

    if (n < 0) {
        perror("ERROR writing to socket");
        exit(1);
    }

    return 0;
}
```

client.c

```c
#include <stdio.h>
#include <stdlib.h>

#include <netdb.h>
#include <netinet/in.h>
#include <unistd.h>

#include <string.h>

int main(int argc, char *argv[]) {
    int sockfd, n;
    uint16_t portno;
    struct sockaddr_in serv_addr;
    struct hostent *server;

    char buffer[256];

    if (argc < 3) {
        fprintf(stderr, "usage %s hostname port\n", argv[0]);
        exit(0);
    }

    portno = (uint16_t) atoi(argv[2]);

    /* Create a socket point */
    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    if (sockfd < 0) {
        perror("ERROR opening socket");
        exit(1);
    }

    server = gethostbyname(argv[1]);

    if (server == NULL) {
        fprintf(stderr, "ERROR, no such host\n");
        exit(0);
    }

    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy(server->h_addr, (char *) &serv_addr.sin_addr.s_addr, (size_t) server->h_length);
    serv_addr.sin_port = htons(portno);

    /* Now connect to the server */
    if (connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
        perror("ERROR connecting");
        exit(1);
    }

    /* Now ask for a message from the user, this message
       * will be read by server
    */

    printf("Please enter the message: ");
    bzero(buffer, 256);
    fgets(buffer, 255, stdin);

    /* Send message to the server */
    n = write(sockfd, buffer, strlen(buffer));

    if (n < 0) {
        perror("ERROR writing to socket");
        exit(1);
    }

    /* Now read server response */
    bzero(buffer, 256);
    n = read(sockfd, buffer, 255);

    if (n < 0) {
        perror("ERROR reading from socket");
        exit(1);
    }

    printf("%s\n", buffer);
    return 0;
}
```