Server

int clients[MAX_COUNT_CLIENTS]; *array to store file descriprots of clients.*

char is_active[MAX_COUNT_CLIENTS]; *array to store client's statuses.*

int count_active_clients; *to store the number of active clients.*

static inline int reserve_socket_cell() *to fill the clients and is_active array cells*

static inline void free_socket_cell(int cell) *to free the clients and is_active array cells*

static inline void notify_all(char *buffer, char message_len, int skip) *to send the message to every active client*

static void* client_handler(void * arg) *to get message from client and to notify all other clients.*

int main(int argc, char *argv[]) *to make socket and create threads with client handler for clients*

int clients[MAX_COUNT_CLIENTS]; *array to store file_id of clients.*

char is_active[MAX_COUNT_CLIENTS]; *array to store client's statuses.*

int count_active_clients; *to store the number of active clients.*

static inline int reserve_socket_cell()

mutex lock

Increment of count_active_clients

is_active[i] = 1, where i is the first free cell in is_active array

mutex unlock

static inline void free_socket_cell(int cell)

mutex_lock

is_active[cell] = 0

Decrement count_active_clients

mutex unlock

static inline void notify_all(char *buffer, char message_len, int skip) ***to send the message to every active client***

For every active client:

write(clients[i], &message_len, n) - first send message length

write(clients[i], buffer + message_len - n, n) – then send the message itself

Do not forget to check the return values from write functions to check the success.

static void* client_handler(void * arg) ***to get message from client and to notify all other clients.***

Here arg is for cell, so get current cell from the arg.
Create char buffer[256] for the message

In a cycle:

read(clients[cell], &message_len, 1) – to read the message length

bzero(buffer, 256)

char size = message_len;
n = read(clients[cell], buffer + message_len - size, size); - to read the message

Do not forget to read the read return values to check the success.
Break if error.

printf("Message get: %s\n", buffer)

notify_all(buffer, message_len, cell)

free_socket_cell(cell)

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

```
int sockfd, newsockfd;
uint16_t portno;
unsigned int clilen;
struct sockaddr_in serv_addr, cli_addr;
(void)argc;
(void)argv;

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

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

 if (argc != 2) {
 fprintf(stderr, "usage: %s port\n", argv[0]);
exit(0);    }
```

```
portno = (uint16_t) atoi(argv[1]);

/* Initialize socket structure */

bzero((char *) &serv_addr, sizeof(serv_addr));

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");
close(sockfd);
return 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 */

In a cycle

```
newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen)
```

Now check the newsockfd return value and check that count_active_clients is less than MAX_COUNT_CLIENTS

```
int cell = reserve_socket_cell()

clients[cell] = newsockfd

pthread_t thread_id

if (pthread_create(&thread_id, NULL, client_handler, is_active + cell) != 0) {
continue;       }

pthread_detach(thread_id); // не дано    }
```

Client

char is_input_mode – *flag to show the client is typing a message*

char force_read(int sockfd, char *buffer, int len) – *function to read from a socket*
char read_message(int sockfd, char *buffer) – *to read a message, using* force_read

char force_send(int sockfd, char *buffer, int len) – *function to write to a socket*
char send_message(int sockfd, char *nickname, char *text) – *to write a message in a format "[nickname] message", using* force_send

static void* server_handler(void * arg) – *to read a message from sever and to print it on the screen in format: time [nickname] message*

int main(int argc, char *argv[]) – *to create a socket, to connect to the server, ask the message from user, write this message to the socket and run server handler.*

char force_read(int sockfd, char *buffer, int len) – *function to read from a socket*
char read_message(int sockfd, char *buffer) – *to read a message, using* force_read

```
char force_read(int sockfd, char *buffer, int len)
        char size = len;
        int n = read(sockfd, buffer + len - size, size) – do not forget to check the read return value
```

```
char read_message(int sockfd, char *buffer) {

        force_read(sockfd, &len, 1)  - first to read the message length

        force_read(sockfd, buffer, len) – then read the message
```

Do not forget to check the read return values in both cases

static void* server_handler(void * arg) – ***to read a message from sever and to print it on the screen in format: time [nickname] message***

static void* server_handler(void * arg)
   int sockfd = *(int*)arg;
In a cycle:
      read_message(sockfd, buffer)

To print: time [nickname] message:

```
mutex lock
while (is_input_mode) {
     sleep(1) -  to thread wait for 1 second, while user typed a message
 }
time_t t = time(NULL);
struct tm* lt = localtime(&t);
printf("<%02d:%02d> %s", lt->tm_hour, lt->tm_min, buffer);
mutex unlock
```

char force_send(int sockfd, char *buffer, int len) – ***function to write to a socket***
char send_message(int sockfd, char *nickname, char *text) – ***to write a message in a format "[nickname] message", using*** **force_send**

char force_send(int sockfd, char *buffer, int len):
int n = len
int r = write(sockfd, buffer + len - n, n) – do not forget to check the return value to catch the possible errors

char send_message(int sockfd, char *nickname, char *text)
char len = strlen(nickname) + 3 + strlen(text) + 1;
force_send(sockfd, &len, 1)
force_send(sockfd, "[", 1)
force_send(sockfd, nickname, strlen(nickname))
force_send(sockfd, "] ", 2)
force_send(sockfd, text, strlen(text) + 1)

do not forget to check the return value to catch the possible errors

```
int main(int argc, char *argv[]) {
    int sockfd = 0;
    char *nickname = NULL;
    uint16_t portno = 0;
    struct sockaddr_in serv_addr = {};
    struct hostent *server = NULL;
    char buffer[256] = {};

    if (argc != 4) {
        fprintf(stderr, "usage: %s hostname port nickname\n", argv[0]);
        exit(0);    }
    portno = (uint16_t) atoi(argv[2]);

    /* Create a socket point */
    sockfd = socket(AF_INET, SOCK_STREAM, 0)  - do not forget to check the return value to catch the possible errors

    server = gethostbyname(argv[1]) - do not forget to check the return value to catch the possible errors

    nickname = argv[3]
```

```
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 */
  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    */

   pthread_t thread_id // to create a thread for server handler
   pthread_create(&thread_id, NULL, server_handler, &sockfd)
```

in a cycle:

strcmp(buffer, "m\n“) – to check the input value is “m”

strcmp(buffer, "exit\n") – to check the input value is “exit”

mutex lock

is_input_mode = 1 - flag to say user is writing a message

mutex_unlock

printf("Please enter the message: ")

fgets(buffer, 200, stdin)

is_input_mode = 0

/* Send message to the server */

send_message(sockfd, nickname, buffer)