### main.c

```
#include "Chat.h"
#include "Utils/Utils.h"

int main(int argc, char *argv[]) {
    int connect_port = DEFAULT_PORT;
    int source_port = DEFAULT_PORT;
    char* connect_ip = NULL;
    char name[MAX_NAME_LENGTH] = "";

    parse_connect_adress(argc, argv, &connect_ip, &source_port);
    parse_source_port(argc, argv, &source_port);
    parse_name(argc, argv, (char *) &name);

    if (strcmp((char *) &name, "") == 0) {
        escape("Необходимо ввести имя: -name <имя>");
    }

    fflush(stdin);
    interface_init();

    struct sockaddr_in local_address;
    struct sockaddr_in buf_address;

    char buf_read[BUFLEN] = {0};
    char buf_send[BUFLEN] = {0};
    char buf_name[MAX_NAME_LENGTH] = {0};

    int buf_read_size = 0;
    int buf_send_size = 0;

    int sockfd = create_socket();

    bind_address(sockfd, &local_address, source_port);
    char* source_ip = inet_ntoa(local_address.sin_addr);

    update_info_box((char *) &name, source_ip, source_port);
    add_client(&buf_address, (char *) &name);

    sen_nonblock_flag(sockfd);
    sen_nonblock_flag(0);

    if (connect_ip != NULL) {
        create_adress(connect_ip, connect_port, &buf_address);

        sprintf((char *) &buf_send, "Подключаемся к %s:%d", connect_ip, connect_port);
        add_message((char *) &buf_send);

        connect_to_client(sockfd, &buf_address, (char *) &name);
    } else {
        add_message("Ждем подключения");
    }

    int timeToSendPing = SEND_PING_PAUSE;
    while (1) {
        unsigned int address_size = sizeof(local_address);
        while ((buf_read_size = socket_read(sockfd, (char *) &buf_read, &buf_address,
&address_size)) != -1) {
            if (is_equal_address(&local_address, &buf_address)) {
                continue;
            }
            int packet_id = get_packet_id((char *) &buf_read);

            struct Client* client = get_client(&buf_address);

            if (client == NULL && packet_id != PACKET_CONNECT_REQUES && packet_id !=
PACKET_CONNECT_ACCEPT) {
                continue;
            }
            if (client != NULL) {
                if (packet_id != PACKET_PING) {
                    create_simple_packet(PACKET_PING, (char *) &buf_send);
                    send_udp(sockfd, &buf_address, (char *) &buf_send, 1);
                }
            }

            buf_read[buf_read_size] = '\0';
            char* buf_ip = inet_ntoa(buf_address.sin_addr);
            int buf_port = ntohs(buf_address.sin_port);
            switch (packet_id) {
                case PACKET_CONNECT_REQUES:
                    if (!is_exist(&buf_address)) {
                        strcpy((char *) &buf_name, buf_read + 1);
                        add_client(&buf_address, (char *) &buf_name);
                        if (strcmp(buf_name, name) == 0) {
                            break;
                        }
                        update_client_box();
                        sprintf((char *) &buf_send, "Подключился клиент %s [%s:%d]", 
buf_name, buf_ip, buf_port);
                        add_message((char *) &buf_send);
                    }
                    buf_send_size = create_connect_accept_packet((char *) &buf_send, 
(char *) &name);
                    send_udp(sockfd, &buf_address, (char *) &buf_send, buf_send_size);
                    break;
                case PACKET_CONNECT_ACCEPT:
                    if (!is_exist(&buf_address)) {
                        strcpy((char *) &buf_name, buf_read + 1);
                        add_client(&buf_address, (char *) &buf_name);
                        update_client_box();

                        sprintf((char *) &buf_send, "Подключились к %s", buf_name);
                        add_message((char *) &buf_send);
                    }
                    break;
                case PACKET_PING:
                    client->isActive = PING_SKIP_TO_TIMEOUT;
                    break;
                case PACKET_TIMEOUT:
                    connect_to_client(sockfd, &buf_address, (char *) &name);
                    break;
                case PACKET_SEND_MESSAGE:
                    get_name(client, (char *) &buf_name);
                    if (strcmp(buf_name, name) == 0) {
                        break;
                    }
                    sprintf((char *) &buf_send, "%s: %s", buf_name, buf_read + 1);
                    add_message(buf_send);
                    break;
                case PACKET_REQUEST_USERS:
                    buf_send_size = create_list_users_packet((char *) &buf_send);
                    send_udp(sockfd, &buf_address, (char *) &buf_send, buf_send_size);
                    break;
                case PACKET_LIST_USERS:
                    buf_send_size = create_connect_request_packet((char *) &buf_send,
(char *) &name);
                    int count =  buf_read[1];
                    for (int i = 0; i < count; i++) {
                        memcpy(&(buf_address), buf_read + 2, sizeof(struct
sockaddr_in));
                        send_udp(sockfd, &buf_address, (char *) &buf_send,
buf_send_size);
                    }
                    break;
            }
        }
        static int size_input = 0;
        static char buf_input[100] = {0};
        while (read_input_box((char *) buf_input, &size_input) == 1) {
            if (strcmp(buf_input, "/quit") == 0) {
                close_socket(sockfd);
                interface_close();
                return 0;
            }
            sprintf((char *) &buf_send, "Вы: %s", buf_input);
            add_message((char *) &buf_send);
            create_message_packet((char *) &buf_send, (char *) &buf_input,
size_input);
            send_packet(sockfd, (char *) &buf_send, size_input + 1);
            memset(buf_input, 0, 100);
            size_input = 0;
        }

    }
    close_socket(sockfd);
    interface_close();
    return 0;
}
```

```
### Clients.h
#ifndef C_P2P_CHAT_CLIENTS_H
#define C_P2P_CHAT_CLIENTS_H

#include "Chat.h"

struct Client {
    int isActive;
    char name[MAX_NAME_LENGTH];
    struct sockaddr_in address;
};

extern struct Client clients[MAX_CLIENTS];
```

```
void add_client(const struct sockaddr_in* addr, const char* name);

struct Client* get_client(const struct sockaddr_in* addr);

int is_exist(const struct sockaddr_in* addr);

void remove_client(struct Client* client);

void get_name(const struct Client* client, char* name);

#endif //C_P2P_CHAT_CLIENTS_H
```

### Clients.c

```c
#include "Clients.h"

struct Client clients[MAX_CLIENTS] = {0};

void add_client(const struct sockaddr_in* addr, const char* name) {
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i].isActive <= 0) {
            memcpy(&(clients[i].address), addr, sizeof(struct sockaddr_in));
            strcpy((char *) &(clients[i].name), name);
            clients[i].isActive = PING_SKIP_TO_TIMEOUT;
            update_client_box();
            return;
        }
    }
}

struct Client* get_client(const struct sockaddr_in* addr) {
    for     (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i].isActive > 0) {
            // Сравниваем ip
            if (is_equal_address(addr, &(clients[i].address))) {
                return &(clients[i]);
            }
        }
    }
    return NULL;
}

int is_exist(const struct sockaddr_in* addr) {
    return get_client(addr) != NULL;
}

void remove_client(struct Client* client) {
    client->isActive = 0;
    update_client_box();
}

void get_name(const struct Client* client, char* name) {
    strcpy(name, (char *) &(client->name));
}

### Chat.h
#ifndef C_P2P_CHAT_CHAT_H
#define C_P2P_CHAT_CHAT_H

#include <curses.h>
#include <locale.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <arpa/inet.h>
#include <sys/ioctl.h>

#include "Config.h"
#include "Clients.h"
#include "Network/Packet.h"
#include "Network/Socket.h"
#include "Utils/Interface.h"

void escape(const char* error);
void connect_to_client(int sockfd, const struct sockaddr_in* addr, const char* name);
void send_packet(int sockfd, const char* buf, int buf_size);

#endif //C_P2P_CHAT_CHAT_H
```

```
#### Chat.c
#include "Chat.h"


void escape(const char* error) {
    printf("Error!!!\n");
    printf("%s\n", error);
    exit(EXIT_FAILURE);
}


void connect_to_client(int sockfd, const struct sockaddr_in* addr, const char* name) {
    char buf[100];
    int buf_size = 0;


    while (1) {
        buf_size = create_connect_request_packet((char *) &buf, name);
        send_udp(sockfd, addr, buf, buf_size);
        sleep(2);


        struct sockaddr_in buf_address = {0};
        unsigned int address_size = sizeof(struct sockaddr_in);
        while ((buf_size = socket_read(sockfd, (char *) &buf, &buf_address,
&address_size)) != -1) {
            buf[buf_size] = '\0';
            int packet_id = get_packet_id((char *) &buf);
            if (packet_id == PACKET_CONNECT_ACCEPT && is_equal_address(addr,
&buf_address)) {


                char buf_name[MAX_NAME_LENGTH * 2];
                strcpy((char *) &buf_name, buf + 1);
                add_client(&buf_address, (char *) &buf_name);
                update_client_box();


                sprintf((char *) &buf, "Подключились к %s", buf_name);
                add_message((char *) &buf);


                buf_size = create_simple_packet(PACKET_REQUEST_USERS, (char *) &buf);
                send_udp(sockfd, addr, buf, buf_size);
                return;
            }
        }
    }
}

void send_packet(int sockfd, const char* buf, int buf_size) {
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i].isActive > 0) {
            send_udp(sockfd, &(clients[i].address), buf, buf_size);
        }
    }
}


### Utils.h
#ifndef C_P2P_CHAT_UTILS_H
#define C_P2P_CHAT_UTILS_H

#include "../Chat.h"

void parse_connect_adress(int argc, char *argv[], char** ip, int* port);
void parse_source_port(int argc, char *argv[], int* port);
void parse_name(int argc, char *argv[], char* name);

#endif //C_P2P_CHAT_UTILS_H



### Utils.c
#include "Utils.h"

void parse_connect_adress(int argc, char *argv[], char** ip, int* port) {
    for (int i = 0; i < argc; i++) {
        if (strcmp(argv[i], "-connect") == 0 && (i + 2) <= argc) {
            *ip = argv[i + 1];
            *port = atoi(argv[i + 2]);

            if (*port == 0) {
                escape("Неправильно указан порт подключения\n");
            }
            return;
        }
    }
}

void parse_source_port(int argc, char *argv[], int* port) {
    for (int i = 0; i < argc; i++) {
        if (strcmp(argv[i], "-port") == 0 && (i + 1) <= argc) {
            *port = atoi(argv[i + 1]);

            if (*port == 0) {
                escape("Неправильно указан локальный порт\n");
            }
            return;
        }
    }
}

void parse_name(int argc, char *argv[], char* name) {
    for (int i = 0; i < argc; i++) {
        if (strcmp(argv[i], "-name") == 0 && (i + 1) <= argc) {
            strcpy(name, argv[i + 1]);
            return;
        }
    }
}
```

### Interface.h

```c
#ifndef C_P2P_CHAT_INTERFACE_H
#define C_P2P_CHAT_INTERFACE_H

#include "../Chat.h"

void interface_init();
void interface_close();
void update_client_box();
void update_info_box(const char* name, const char* ip, int port);
void add_message(const char* msg);
int read_input_box(char* buf, int* size);

#endif //C_P2P_CHAT_INTERFACE_H
```

### Interface.c

```c
#include "Interface.h"

static WINDOW* box_info = NULL;
static WINDOW* box_client = NULL;
static WINDOW* box_messages = NULL;
static WINDOW* box_input = NULL;

static char messages[16][126] = {{0}};

static void init_info_box() {
    box_info = newwin(5, 65, 0, 0);
    box(box_info, 0, 0);
    wrefresh(box_info);
}

static void init_client_box() {
    box_client = newwin(25, 15, 0, 65);
    box(box_client, 0, 0);
    mvwprintw(box_client, 1, 1, "   Клиенты   ");
    mvwprintw(box_client, 2, 0, "├─────────────┤");
    wrefresh(box_client);
}

static void init_message_box() {
    box_messages = newwin(17, 65, 5, 0);
    box(box_messages, 0, 0);
    mvwprintw(box_messages, 16, 0,
"│                                                               │");

    wrefresh(box_messages);
}

static void init_inpit_box() {
    box_input = newwin(3, 65, 22, 0);
    box(box_input, 0, 0);
    mvwprintw(box_input, 0, 0,
"├───────────────────────────────────────────────────────────────┤");;

    wrefresh(box_input);
}

void update_client_box() {
    wclear(box_client);
    box(box_client, 0, 0);
    mvwprintw(box_client, 1, 1, "   Клиенты   ");
    mvwprintw(box_client, 2, 0, "├─────────────┤");
    int position = 3;
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i].isActive > 0) {
            mvwprintw(box_client, position, 1, clients[i].name);
            position++;
        }
    }

    wrefresh(box_client);
```

```
}


static void update_message_box() {
    wclear(box_messages);
    box(box_messages, 0, 0);
    mvwprintw(box_messages, 16, 0,
"│                                                                │");
    for (int i = 0; i < 16; i++) {
        mvwprintw(box_messages, i + 1, 1, messages[i]);
    }

    wrefresh(box_messages);
}


void add_message(const char* msg) {
    for (int i = 1; i < 16; i++) {
        memset((char *) &messages[i - 1], ' ', sizeof(char) * 18);
        strcpy((char *) &messages[i - 1], (char *) &messages[i]);
    }
    strcpy((char *) &messages[15], msg);
    update_message_box();
}


void update_info_box(const char* name, const char* ip, int port) {
    wclear(box_info);
    box(box_info, 0, 0);
    mvwprintw(box_info, 2, 1, "  Ваш ник: ");
    mvwprintw(box_info, 2, 13, name);

    wrefresh(box_info);
}


void interface_init() {
    setlocale(LC_ALL,"");
    printf("\e[8;25;80;t");
    initscr();

    init_info_box();
    init_message_box();
    init_client_box();
    init_inpit_box();

    keypad(box_input, TRUE);
    echo();
    cbreak();      // disable line-buffering
    wtimeout(box_input, 1000 / TICK_PER_SECOND);  // wait 100 milliseconds for input
}

int read_input_box(char* buf, int* size) {
    int symbol = 0;
    while ((symbol = wgetch(box_input)) != ERR) {
        if (symbol == '\n') {
            for (int i = 0; i < *size; i++) {
                mvwprintw(box_input, 1, i + 1, " ");
            }
            return 1;
        }else if (symbol == KEY_BACKSPACE) {
            if (*size > 0) {
                mvwprintw(box_input, 1, (*size), " ");
                buf[--(*size)] = 0;
            }
        }else if (*size < 99) {
            buf[(*size)++] = (char) symbol;
        }
    }
    mvwprintw(box_input, 1, 1, (char *)buf);
    return 0;
}
```

```
void interface_close() {

    delwin(box_info);

    delwin(box_client);

    delwin(box_client);

    delwin(box_messages);


    endwin();

}


### Socket.h

#ifndef C_P2P_CHAT_SOCKET_H

#define C_P2P_CHAT_SOCKET_H


#include "../Chat.h"


int create_socket();


void close_socket(int sockfd);


void bind_address(int sockfd, struct sockaddr_in *addr, int port);


void send_udp(int sockfd, const struct sockaddr_in *addr, const char *buf, int
buf_size);


int socket_read(int sockfd, char *buf, struct sockaddr_in* addr, unsigned int
*addr_len);


void sen_nonblock_flag(int descriptor);


int is_equal_address(const struct sockaddr_in* first, const struct sockaddr_in*
second);


void create_adress(const char* ip, int port, struct sockaddr_in* addr);


#endif //C_P2P_CHAT_SOCKET_H
```

```
### Socket.c

#include "Socket.h"

int create_socket() {
    int sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (sockfd == -1) escape("Can't create socket");
    return sockfd;
}

void close_socket(int sockfd) {
    close(sockfd);
}

void bind_address(int sockfd, struct sockaddr_in *addr, int port) {
    memset((char *) addr, 0, sizeof(*addr));
    addr->sin_family = AF_INET;
    addr->sin_port = htons((unsigned short) port); // To network byte order
    addr->sin_addr.s_addr = htonl(INADDR_ANY);

    int result = bind(sockfd, (struct sockaddr *) addr, sizeof(*addr));
    if (result == -1) escape("Can't bind address");
}

void send_udp(int sockfd, const struct sockaddr_in *addr, const char *buf, int
buf_size) {
    sendto(sockfd, buf, (size_t) buf_size, 0, (struct sockaddr*) addr, sizeof(*addr));
}

int socket_read(int sockfd, char *buf, struct sockaddr_in* addr, unsigned int
*addr_len) {
    int recv_len = (int) recvfrom(sockfd, buf, BUFLEN, 0, (struct sockaddr*) addr,
addr_len);
    return recv_len;
}
```

```
void sen_nonblock_flag(int descriptor) {

    int flags = fcntl(descriptor, F_GETFL);

    flags |= O_NONBLOCK;

    fcntl(descriptor, F_SETFL, flags);

}


int is_equal_address(const struct sockaddr_in* first, const struct sockaddr_in*
second) {

    return  (first->sin_addr.s_addr == second->sin_addr.s_addr) &&

            (first->sin_port == second->sin_port);

}


void create_adress(const char* ip, int port, struct sockaddr_in* addr) {

    addr->sin_family = AF_INET;

    addr->sin_addr.s_addr = inet_addr(ip);

    addr->sin_port = htons(port);

}


### Packet.h

#ifndef C_P2P_CHAT_PACKET_H

#define C_P2P_CHAT_PACKET_H


#include "../Chat.h"


#define PACKET_CONNECT_REQUES '0'

#define PACKET_CONNECT_ACCEPT '1'

#define PACKET_PING '3'

#define PACKET_TIMEOUT '4'

#define PACKET_REQUEST_USERS '5'

#define PACKET_LIST_USERS '6'

#define PACKET_SEND_MESSAGE '7'


int get_packet_id(const char* data);

int create_simple_packet(char type, char* buf);
int create_connect_request_packet(char* buf, const char* name);
int create_connect_accept_packet(char* buf, const char* name);
int create_message_packet(char* buf_send, char* buf_input, int len_msg);
int create_list_users_packet(char* buf);

#endif //C_P2P_CHAT_PACKET_H

### Packet.c

#include "Packet.h"

int get_packet_id(const char* buf) {
    return buf[0];
}

int create_simple_packet(char type, char* buf) {
    buf[0] = type;
    return 1;
}

int create_connect_request_packet(char* buf, const char* name) {
    buf[0] = PACKET_CONNECT_REQUES;
    strcpy(buf + 1, name);
    return 1 + (int) strlen(name);
}

int create_connect_accept_packet(char* buf, const char* name) {
    buf[0] = PACKET_CONNECT_ACCEPT;
    strcpy(buf + 1, name);
    return 1 + (int) strlen(name);
}

int create_message_packet(char* buf_send, char* buf_input, int len_msg) {
    buf_send[0] = PACKET_SEND_MESSAGE;
    strcpy(buf_send + 1, buf_input);
    return 1 + len_msg;
}

int create_list_users_packet(char* buf) {
    buf[0] = PACKET_LIST_USERS;
    buf[1] = 0; // Кол-во клиентов

    int pos = 2;
    // Для всех клиентов
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i].isActive > 0) {
            buf[1]++;
            memcpy(buf + pos, &(clients->address), sizeof(struct sockaddr_in));
            pos += sizeof(struct sockaddr_in);
        }
    }
    return pos;
}
```