Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Институт информатики и вычислительной техники

09.03.01 "Информатика и вычислительная техника"

профиль "Программное обеспечение средств вычислительной техники и автоматизированных систем"

**Курсовая работа**по дисциплине «Сетевое программирование»

**Разработка сетевого приложения «Чат».   
Асинхронный сервер с установлением соединения с использованием функции select**

Выполнил:

студент гр. ИП 214 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Шахов В. Г. /

ФИО студента

« 24 » апреля 2025 г.

Проверил

\_\_\_\_\_\_\_\_\_\_\_\_\_/ Павский К. В. /

ФИО преподавателя

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г. Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Новосибирск 2025 г.

Оглавление

[Задание 3](#_Toc193044041)

[Описание протокола 3](#_Toc193044042)

[Основные этапы работы TCP 3](#_Toc193044043)

[Преимущества и недостатки TCP 5](#_Toc193044044)

[Флаги TCP 5](#_Toc193044045)

[Описание реализации 6](#_Toc193044046)

[Работа программы 8](#_Toc193044047)

[Листинг 8](#_Toc193044048)

# **Задание**

В рамках курсовой работы необходимо написать асинхронный сервер с установлением соединения с использованием функции select. Реализация на языке C++ для Unix-подобных операционных систем.

# **Описание протокола**

TCP (Transmission Control Protocol) – это протокол управления передачей. Один из основных протоколов Интернета, который обеспечивает надежную передачу данных между узлами сети. Он работает на транспортном уровне модели OSI и обеспечивает установление, поддержку и завершение соединений, а также гарантирует надежность и порядок передачи данных.

## **Основные этапы работы TCP**

1. Установление соединения (Three-Way Handshake)

Перед тем как начать обмениваться данными, TCP устанавливает соединение между двумя узлами через трехстороннее рукопожатие (Three-Way Handshake):

* **Шаг 1:** Клиент отправляет серверу сегмент с флагом **SYN** (synchronize), который указывает на желание установить соединение. В этом сегменте указывается начальный номер последовательности (Sequence Number).
* **Шаг 2:** Сервер, получив запрос на соединение, отвечает клиенту сегментом с флагами **SYN** и **ACK** (acknowledge). Это подтверждение, что сервер готов к соединению, и также включает свой номер последовательности.
* **Шаг 3:** Клиент отправляет серверу подтверждение с флагом **ACK**, сообщая, что он получил ответ от сервера и готов к обмену данными.

После этого соединение считается установленным, и стороны могут начать обмен данными.

1. Передача данных

После установления соединения начинается передача данных. Этот процесс включает несколько важных этапов:

* **Нумерация байтов**: каждый байт данных, передаваемый по TCP, имеет уникальный номер. Этот номер используется для отслеживания, какие байты были отправлены и приняты.
* **Подтверждения (ACK)**: когда получатель получает данные, он отправляет подтверждение (ACK) отправителю, сообщая, что он принял данные. При этом в ACK содержится номер последнего принятого байта.
* **Окно скольжения (Sliding Window)**: этот механизм позволяет получателю указать отправителю, сколько данных он может передать без ожидания подтверждения. Это помогает предотвратить перегрузку сети.

1. Контроль потока и управление перегрузкой

* **Контроль потока:** с помощью окна скольжения TCP регулирует количество данных, которые могут быть отправлены без получения подтверждения. Это предотвращает переполнение приемной стороны.
* **Управление перегрузкой**: TCP использует различные алгоритмы для контроля за перегрузкой сети. Один из таких алгоритмов — это **медленный старт (Slow Start)**, который постепенно увеличивает скорость передачи данных до тех пор, пока не будет обнаружена перегрузка сети. Если это происходит, TCP уменьшает скорость передачи данных.

1. Коррекция ошибок

* **Проверка целостности**: Каждый сегмент данных TCP включает контрольную сумму (checksum), которая позволяет проверять целостность данных. Если получатель обнаруживает ошибку, он может запросить повторную передачу данных.
* **Повторная передача (Retransmission)**: если отправленный сегмент не был подтвержден в течение определенного времени (например, из-за потери пакета), то он будет автоматически отправлен повторно.

1. Закрытие соединения (Four-Way Handshake)

Когда одна из сторон завершает передачу данных, соединение должно быть закрыто. Для этого используется четырехсторонний процесс завершения соединения:

* **Шаг 1**: Сторона, которая инициирует завершение соединения, отправляет сегмент с флагом **FIN** (finish). Это сигнализирует о том, что эта сторона больше не будет отправлять данные.
* **Шаг 2**: Получатель подтверждает получение сегмента с флагом **ACK**, сообщая, что он принял запрос на завершение соединения.
* **Шаг 3**: Получатель может, в свою очередь, отправить свой сегмент с флагом **FIN**, если он также завершил передачу данных.
* **Шаг 4**: Инициатор завершения соединения подтверждает получение второго **FIN** сегмента, и соединение окончательно закрывается.

TCP гарантирует, что все данные будут доставлены в правильном порядке и без потерь:

* Если сегмент данных не был получен (например, из-за ошибки или потери в сети), он будет повторно отправлен.
* Если данные получены не в том порядке, в котором они были отправлены, TCP будет их переставлять.
* Все данные, передаваемые по TCP, сопровождаются контрольной суммой для проверки целостности.

## **Преимущества и недостатки TCP**

**Преимущества**:

* **Надежность**: гарантированная доставка данных и их целостность.
* **Порядок**: данные передаются в том порядке, в котором они были отправлены.
* **Контроль потока**: TCP предотвращает переполнение принимающей стороны.
* **Управление перегрузкой**: TCP может адаптировать скорость передачи данных, чтобы избежать перегрузки сети.

**Недостатки**:

* **Производительность**: из-за необходимости подтверждений и контроля за потоком, TCP может быть медленнее, чем без подтверждений (например, в случае с UDP).
* **Задержка**: время на установление соединения и подтверждения может добавить задержки при передаче данных.

## **Флаги TCP**

Каждый пакет TCP может содержать различные флаги, которые обозначают тип передаваемого пакета. Основные флаги:

* **URG –** указывает на срочность данных**;**
* **ACK – флаг пакета, содержащего подтверждение получения;**
* **PSH – флаг выталкивания, немедленная отсылка данных после считывания данных этого пакета;**
* **RST – флаг переустановки соединения;**
* **SYN – флаг синхронизации чисел последовательности;**
* **FIN – флаг окончания передачи со стороны отправителя**

# **Описание реализации**

Сервер чата реализован на языке C++ и использует сокеты для управления подключения клиентов. Он поддерживает многопользовательские комнаты, обмен сообщениями (в том числе и личными), а также происходит логирование.

1. Подключение клиентов

Сервер принимает входящие соединения через TCP-сокет:

* Создает серверный сокет.
* Устанавливается прослушивание порта.
* Добавляется в список отслеживаемых дескрипторов.
* В цикле select() ожидается активности на сокетах.
* Принимаются новые клиенты через accept().
* Отправляется запрос на ввод имени.

1. Аутентификация

При подключении клиент отправляет имя

* Сервер получает имя и проверяет на пустоту и уникальность
* Клиент получает приветственное сообщение.

1. Работа с select()

Для одновременной обработки нескольких клиентов используется select()

* Сервер создает множество отслеживаемых файловых дескрипторов fd\_set.
* В главный массив fd\_set добавляется серверный сокет.
* В основном цикле вызывается select(), который блокируется, пока не появится активность на каком-либо сокете.
* Когда новый клиент подключается, сервер вызывает accept() и добавляет новый сокет в fd\_set.
* Если активен клиентский сокет, сервер читает сообщения с помощью recv().
* Если клиент отключается, его сокет удаляется из множества fd\_set.

1. Работа с комнатами

* /create <name\_room> - сервер создает комнату, если ее нет. Создатель автоматически присоединяется к этой комнате.
* /join <name\_room> - команда для присоединения клиента к комнате. Сервер проверяет существование комнаты. После клиент подключается к комнате.
* /leave – команда выхода из комнаты. Клиент удаляется из списка комнаты.

1. Список комнат и пользователей

* /list – отображаемый список доступных комнат.
* /users – показывает всех участников комнаты.
* /users\_on\_server – выводит всех пользователей на сервере.

1. Обмен сообщениями

Клиенты могут отправлять сообщения в комнату.

* Сервер пересылает их всем участникам комнаты.
* Сообщения отображаются в формате [имя]: текст.

Также пользователи могут отправлять друг другу личные сообщения.

* /pm <user\_name> <message> - пользователь отправляет личное сообщение, которое увидит только получатель, если он существует. Сообщение приходит в виде: [Личное сообщение от <имя\_отправителя>]: <сообщение>.

1. Логирование

Реализовано логирование. Происходящее на сервере записывается в файл server\_log.txt в форме: [дата время] <Информация о том, что произошло на сервере в этот момент>.

Записывается почти всё: от того что сервер запущен, до того, кто и кому какое сообщение отправил.

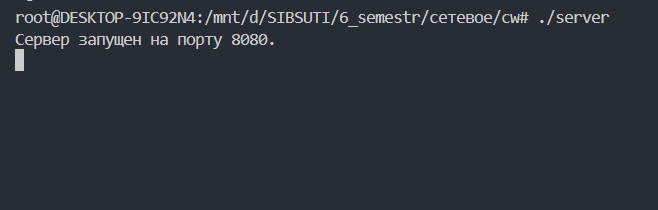
1. Завершение работы

Клиент может отключится командой /exit.

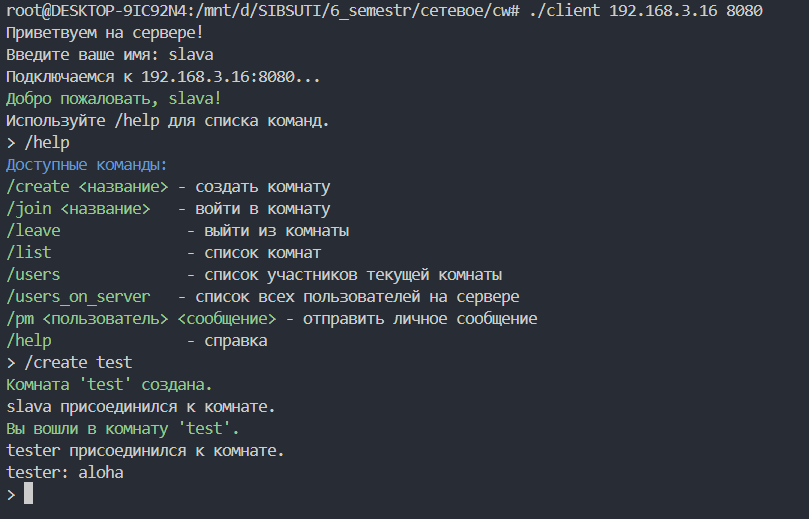
* Сервер удаляет его из комнаты.
* Если клиент разрывает соединение, он автоматически удаляется.

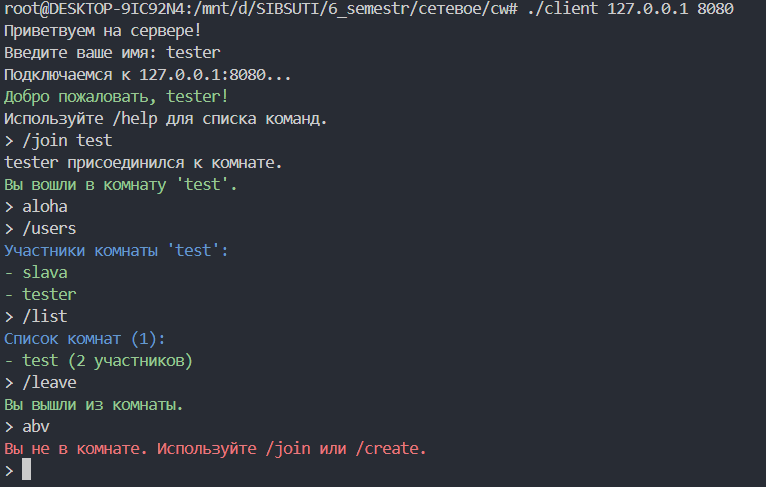
# **Работа программы**

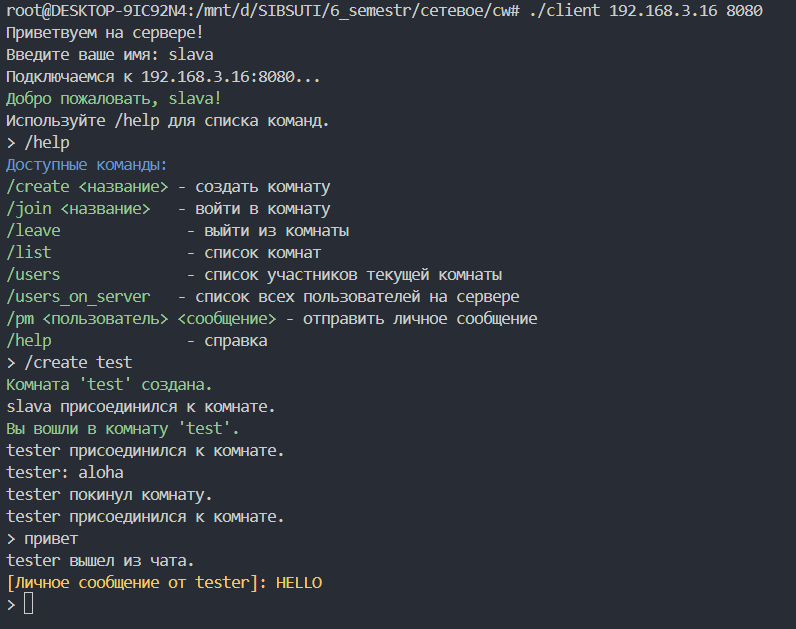
Запуск сервера:

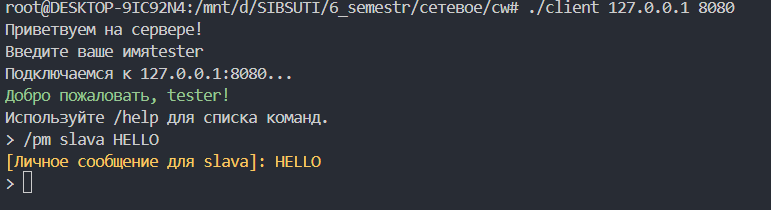


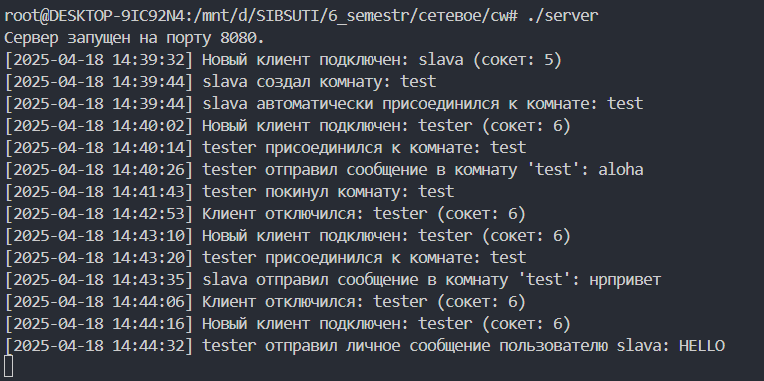
Присоединение клиента + основной функционал:

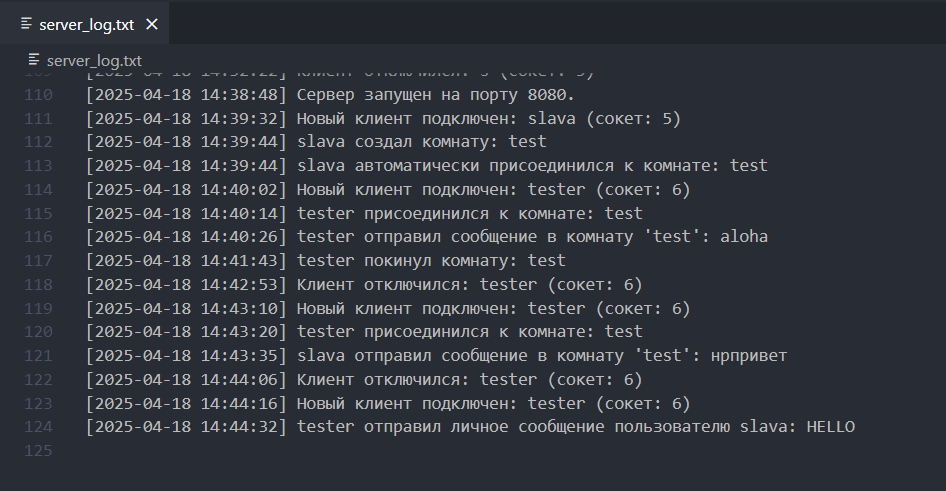












# **Листинг**

client.cpp:

#include <iostream>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <unistd.h>

#include <string.h>

#include <thread>

using namespace std;

void clearLine() {

    cout << "\r" << string(100, ' ') << "\r";

}

void readHandler(int sock) {

    char buffer[1024] = {0};

    while (true) {

        int valread = read(sock, buffer, sizeof(buffer)-1);

        if (valread > 0) {

            buffer[valread] = '\0';

            clearLine();

            cout << buffer;

            cout << "> " << flush;

        } else if (valread == 0) {

            cout << "\nСоединение с сервером разорвано\n";

            exit(0);

        } else {

            perror("Ошибка чтения");

            exit(1);

        }

    }

}

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

    if (argc != 3) {

        cerr << "Использование: " << argv[0] << " <IP> <порт>\n";

        return 1;

    }

    string server\_ip = argv[1];

    int server\_port = atoi(argv[2]);

    string name;

    cout << "Приветвуем на сервере!\n";

    cout << "Введите ваше имя: ";

    getline(cin, name);

    int sock = socket(AF\_INET, SOCK\_STREAM, 0);

    if (sock < 0) {

        perror("Ошибка создания сокета");

        return -1;

    }

    struct sockaddr\_in serv\_addr;

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

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

    if (inet\_pton(AF\_INET, server\_ip.c\_str(), &serv\_addr.sin\_addr) <= 0) {

        cerr << "Неверный адрес: " << server\_ip << endl;

        close(sock);

        return -1;

    }

    cout << "Подключаемся к " << server\_ip << ":" << server\_port << "..." << endl;

    if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

        perror("Ошибка подключения");

        close(sock);

        return -1;

    }

    send(sock, name.c\_str(), name.length(), 0);

    thread reader(readHandler, sock);

    reader.detach();

    string message;

    while (true) {

        cout << "> " << flush;

        getline(cin, message);

        if (message == "/exit") {

            break;

        }

        send(sock, message.c\_str(), message.length(), 0);

    }

    close(sock);

    return 0;

}

server.cpp:

#include <iostream>

#include <vector>

#include <string>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/select.h>

#include <cstring>

#include <algorithm>

#include <map>

#include <set>

#include <fstream>

#include <chrono>

#include <iomanip>

#define MAX\_CLIENTS 50

#define BUFFER\_SIZE 1024

using namespace std;

struct Client {

    int socket;

    string name;

    string current\_room;

};

struct Room {

    string name;

    set<int> clients;

};

map<string, Room> rooms;

map<int, Client> clients;

set<string> used\_names;*//для использованных имён*

ofstream log\_file("server\_log.txt", ios::app);

*//для получения текущей даты и времени в строковом формате*

string getCurrentTime() {

    auto now = chrono::system\_clock::now();

    auto time = chrono::system\_clock::to\_time\_t(now);

    stringstream ss;

    ss << put\_time(localtime(&time), "%Y-%m-%d %H:%M:%S");

    return ss.str();

}

void logMessage(const string& message) {

    if (log\_file.is\_open()) {

        log\_file << "[" << getCurrentTime() << "] " << message << endl;

    }

}

void sendToClient(int client\_socket, const string& message) {

    send(client\_socket, message.c\_str(), message.size(), 0);

}

void broadcastMessage(const string& room\_name, const string& message, int exclude\_socket = -1) {

    if (rooms.find(room\_name) == rooms.end()) return;

    for (int client\_socket : rooms[room\_name].clients) {

        if (client\_socket != exclude\_socket) {

            sendToClient(client\_socket, message);

        }

    }

}

vector<string> splitCommand(const string& command) {

    vector<string> args;

    string current;

    for (char c : command) {

        if (c == ' ') {

            if (!current.empty()) {

                args.push\_back(current);

                current.clear();

            }

        } else {

            current += c;

        }

    }

    if (!current.empty()) {

        args.push\_back(current);

    }

    return args;

}

void handleCommand(int client\_socket, const string& command) {

    Client& client = clients[client\_socket];

    vector<string> args = splitCommand(command);

    if (args.empty()) return;

    if (args[0] == "/create" && args.size() > 1) {

        string room\_name = args[1];

        if (rooms.find(room\_name) == rooms.end()) {

            rooms[room\_name] = {room\_name, {}};

            logMessage(client.name + " создал комнату: " + room\_name);

            cout << "[" << getCurrentTime() << "] " << client.name << " создал комнату: " << room\_name << endl;

            sendToClient(client\_socket, "\033[32mКомната '" + room\_name + "' создана.\033[0m\n");

            if (!client.current\_room.empty()) {

                rooms[client.current\_room].clients.erase(client\_socket);

                broadcastMessage(client.current\_room, client.name + " покинул комнату.\n");

            }

            client.current\_room = room\_name;

            rooms[room\_name].clients.insert(client\_socket);

            logMessage(client.name + " автоматически присоединился к комнате: " + room\_name);

            cout << "[" << getCurrentTime() << "] " << client.name << " автоматически присоединился к комнате: " << room\_name << endl;

            broadcastMessage(room\_name, client.name + " присоединился к комнате.\n");

            sendToClient(client\_socket, "\033[32mВы вошли в комнату '" + room\_name + "'.\033[0m\n");

        } else {

            sendToClient(client\_socket, "\033[31mКомната '" + room\_name + "' уже существует.\033[0m\n");

        }

        return;

    }

    else if (args[0] == "/join" && args.size() > 1) {

        string room\_name = args[1];

        if (rooms.find(room\_name) != rooms.end()) {

            if (!client.current\_room.empty()) {

                rooms[client.current\_room].clients.erase(client\_socket);

                broadcastMessage(client.current\_room, client.name + " покинул комнату.\n");

            }

            client.current\_room = room\_name;

            rooms[room\_name].clients.insert(client\_socket);

            logMessage(client.name + " присоединился к комнате: " + room\_name);

            cout << "[" << getCurrentTime() << "] " << client.name << " присоединился к комнате: " << room\_name << endl;

            broadcastMessage(room\_name, client.name + " присоединился к комнате.\n");

            sendToClient(client\_socket, "\033[32mВы вошли в комнату '" + room\_name + "'.\033[0m\n");

        } else {

            sendToClient(client\_socket, "\033[31mКомната '" + room\_name + "' не найдена.\033[0m\n");

        }

        return;

    }

    else if (args[0] == "/leave") {

        if (!client.current\_room.empty()) {

            string room\_name = client.current\_room;

            rooms[room\_name].clients.erase(client\_socket);

            logMessage(client.name + " покинул комнату: " + room\_name);

            cout << "[" << getCurrentTime() << "] " << client.name << " покинул комнату: " << room\_name << endl;

            broadcastMessage(room\_name, client.name + " покинул комнату.\n");

            client.current\_room = "";

            sendToClient(client\_socket, "\033[32mВы вышли из комнаты.\033[0m\n");

        } else {

            sendToClient(client\_socket, "\033[31mВы не находитесь в комнате.\033[0m\n");

        }

        return;

    }

    else if (args[0] == "/list") {

        string response = "\033[34mСписок комнат (" + to\_string(rooms.size()) + "):\033[0m\n";

        for (const auto& room : rooms) {

            response += "\033[32m- " + room.first + " (" + to\_string(room.second.clients.size()) + " участников)\033[0m\n";

        }

        sendToClient(client\_socket, response);

        return;

    }

    else if (args[0] == "/users") {

        if (!client.current\_room.empty()) {

            string response = "\033[34mУчастники комнаты '" + client.current\_room + "':\033[0m\n";

            for (int socket : rooms[client.current\_room].clients) {

                response += "\033[32m- " + clients[socket].name + "\033[0m\n";

            }

            sendToClient(client\_socket, response);

        } else {

            sendToClient(client\_socket, "\033[31mВы не находитесь в комнате.\033[0m\n");

        }

        return;

    }

    else if (args[0] == "/users\_on\_server") {

        string response = "\033[34mСписок пользователей на сервере:\033[0m\n";

        if (clients.empty()) {

            response += "\033[31mНа сервере нет подключенных пользователей.\033[0m\n";

        } else {

            for (const auto& client\_entry : clients) {

                const Client& c = client\_entry.second;

                string room = c.current\_room.empty() ? "Не в комнате" : c.current\_room;

                response += "\033[32m- " + c.name + " (комната: " + room + ")\033[0m\n";

            }

        }

        sendToClient(client\_socket, response);

        return;

    }

    else if (args[0] == "/help") {

        string help\_msg =

            "\033[34mДоступные команды:\033[0m\n"

            "\033[32m/create <название>\033[0m - создать комнату\n"

            "\033[32m/join <название>\033[0m   - войти в комнату\n"

            "\033[32m/leave\033[0m              - выйти из комнаты\n"

            "\033[32m/list\033[0m               - список комнат\n"

            "\033[32m/users\033[0m              - список участников текущей комнаты\n"

            "\033[32m/users\_on\_server\033[0m   - список всех пользователей на сервере\n"

            "\033[32m/pm <пользователь> <сообщение>\033[0m - отправить личное сообщение\n"

            "\033[32m/help\033[0m               - справка\n";

        sendToClient(client\_socket, help\_msg);

        return;

    }

    else if (args[0] == "/pm" && args.size() > 2) {

        string target\_name = args[1];

        string message = command.substr(command.find(' ', command.find(' ') + 1) + 1);

        bool user\_found = false;

        for (const auto& client\_entry : clients) {

            if (client\_entry.second.name == target\_name) {

                sendToClient(client\_entry.first, "\033[33m[Личное сообщение от " + client.name + "]: " + message + "\033[0m\n");

                sendToClient(client\_socket, "\033[33m[Личное сообщение для " + target\_name + "]: " + message + "\033[0m\n");

                logMessage(client.name + " отправил личное сообщение пользователю " + target\_name + ": " + message);

                cout << "[" << getCurrentTime() << "] " << client.name << " отправил личное сообщение пользователю " << target\_name << ": " << message << endl;

                user\_found = true;

                break;

            }

        }

        if (!user\_found) {

            sendToClient(client\_socket, "\033[31mПользователь с именем '" + target\_name + "' не найден.\033[0m\n");

        }

        return;

    }

    else {

        if (!client.current\_room.empty()) {

            string formatted\_msg = client.name + ": " + command + "\n";

            broadcastMessage(client.current\_room, formatted\_msg, client\_socket);

            logMessage(client.name + " отправил сообщение в комнату '" + client.current\_room + "': " + command);

            cout << "[" << getCurrentTime() << "] " << client.name << " отправил сообщение в комнату '" << client.current\_room << "': " << command << endl;

        } else {

            sendToClient(client\_socket, "\033[31mВы не в комнате. Используйте /join или /create.\033[0m\n");

        }

    }

}

int main() {

    int server\_fd, new\_socket;

    struct sockaddr\_in address;

    int opt = 1;

    int addrlen = sizeof(address);

    char buffer[BUFFER\_SIZE] = {0};

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

        perror("socket failed");

        exit(EXIT\_FAILURE);

    }

    if (setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR | SO\_REUSEPORT, &opt, sizeof(opt))) {

        perror("setsockopt");

        exit(EXIT\_FAILURE);

    }

    address.sin\_family = AF\_INET;

    address.sin\_addr.s\_addr = INADDR\_ANY;

    address.sin\_port = htons(8080);

    if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

        perror("bind failed");

        exit(EXIT\_FAILURE);

    }

    if (listen(server\_fd, 3) < 0) {

        perror("listen");

        exit(EXIT\_FAILURE);

    }

    logMessage("Сервер запущен на порту 8080.");

    cout << "Сервер запущен на порту 8080." << endl;

    fd\_set readfds;

    vector<int> client\_sockets(MAX\_CLIENTS, 0);

    while (true) {

        FD\_ZERO(&readfds);

        FD\_SET(server\_fd, &readfds);

        int max\_sd = server\_fd;

        for (int i = 0; i < MAX\_CLIENTS; i++) {

            int sd = client\_sockets[i];

            if (sd > 0) {

                FD\_SET(sd, &readfds);

            }

            if (sd > max\_sd) {

                max\_sd = sd;

            }

        }

        int activity = select(max\_sd + 1, &readfds, NULL, NULL, NULL);

        if ((activity < 0) && (errno != EINTR)) {

            perror("select error");

        }

        if (FD\_ISSET(server\_fd, &readfds)) {

            if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t\*)&addrlen)) < 0) {

                perror("accept");

                exit(EXIT\_FAILURE);

            }

            int valread = read(new\_socket, buffer, BUFFER\_SIZE);

            if (valread <= 0) {

                close(new\_socket);

                continue;

            }

            string client\_name(buffer, valread);

            client\_name.erase(remove(client\_name.begin(), client\_name.end(), '\n'), client\_name.end());

            client\_name.erase(remove(client\_name.begin(), client\_name.end(), '\r'), client\_name.end());

*// Проверка на уникальность имени*

            if (used\_names.count(client\_name) > 0) {

                string error\_msg = "\033[31mИмя '" + client\_name + "' уже занято. Пожалуйста, переподключитесь с другим именем.\033[0m\n";

                send(new\_socket, error\_msg.c\_str(), error\_msg.size(), 0);

                close(new\_socket);

                logMessage("Попытка подключения с уже занятым именем: " + client\_name);

                cout << "[" << getCurrentTime() << "] " << "Попытка подключения с уже занятым именем: " << client\_name << endl;

                continue;

            }

            clients[new\_socket] = {new\_socket, client\_name, ""};

            used\_names.insert(client\_name);

            for (int i = 0; i < MAX\_CLIENTS; i++) {

                if (client\_sockets[i] == 0) {

                    client\_sockets[i] = new\_socket;

                    logMessage("Новый клиент подключен: " + client\_name + " (сокет: " + to\_string(new\_socket) + ")");

                    cout << "[" << getCurrentTime() << "] " << "Новый клиент подключен: " << client\_name << " (сокет: " << new\_socket << ")" << endl;

                    sendToClient(new\_socket, "\033[32mДобро пожаловать, " + client\_name + "!\033[0m\nИспользуйте /help для списка команд.\n");

                    break;

                }

            }

        }

        for (int i = 0; i < MAX\_CLIENTS; i++) {

            int sd = client\_sockets[i];

            if (FD\_ISSET(sd, &readfds)) {

                int valread = read(sd, buffer, BUFFER\_SIZE);

                if (valread == 0) {

                    getpeername(sd, (struct sockaddr\*)&address, (socklen\_t\*)&addrlen);

                    logMessage("Клиент отключился: " + clients[sd].name + " (сокет: " + to\_string(sd) + ")");

                    cout << "[" << getCurrentTime() << "] " << "Клиент отключился: " << clients[sd].name << " (сокет: " << sd << ")" << endl;

                    if (!clients[sd].current\_room.empty()) {

                        broadcastMessage(clients[sd].current\_room, clients[sd].name + " вышел из чата.\n");

                        rooms[clients[sd].current\_room].clients.erase(sd);

                    }

                    used\_names.erase(clients[sd].name);

                    close(sd);

                    client\_sockets[i] = 0;

                    clients.erase(sd);

                } else {

                    buffer[valread] = '\0';

                    string message(buffer);

                    message.erase(remove(message.begin(), message.end(), '\n'), message.end());

                    message.erase(remove(message.begin(), message.end(), '\r'), message.end());

                    handleCommand(sd, message);

                }

            }

        }

    }

    return 0;

}

# **Список источников**

1. Павский К. В., Ефимов А. В. Разработка сетевых приложений (протоколы TCP/IP, клиент-сервер, PCAP, Boost.ASIO): Учебное пособие / Сибирский государственный университет телекоммуникаций и информатики. – Новосибирск, 2018. – 80 с.
2. Протоколы TCP/IP и разработка сетевых приложений: учеб. пособие / К.В. Павский; Сиб. гос. ун-т телекоммуникаций и информатики. - Новосибирск: СибГУТИ, 2013. – 130c.
3. IBM : сайт. – URL: https://www.ibm.com/docs/en/zos/2.5.0?topic=calls-select (дата обращения: 05.03.2025).
4. Пролетарский, А. В. Технологии TCP/IP в современных компьютерных сетях / А. В. Пролетарский, Е. А. Ромашкина, Е. В. Смирнова. – Москва : Издательство МГТУ им. Н.Э. Баумана, 2022. – 640 с. – ISBN 978-5-7038-5166-1.