Федеральное агентство связи (Россвязь)
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра вычислительных систем

КУРСОВАЯ РАБОТА
По дисциплине «Сетевое программирование»

Тема: «Разработка сетевого приложения «Чат ». Сервер без установления соединения (протокол UDP).»

Выполнил:
Студент группы ИС-242
Пеалкиви Даниил Яковлевич
Проверил:
ассистент кафедры ВС
Невейко Артём Игоревич

Новосибирск, 2025

Оглавление

Постановка задачи **1**
Описание протокола **2**
Описание реализации **3**
Работа программы **4**
Листинг **5**
Список источников **15**

Постановка задачи

Разработать сетевое приложение «Чат» с использованием протокола UDP без установления соединения между клиентом и сервером. Реализовать следующие функции:

- передача сообщений между клиентами через сервер;
- поддержка приватных сообщений;
- авторизация пользователей;
- передача файлов между двумя клиентами через отдельное TCP-соединение.

Описание протокола

В данной работе используются **два** сетевых протокола:

1. **Протокол UDP**
   UDP используется для:

- авторизации клиента на сервере,
- отправки и приёма обычных и приватных сообщений,
- уведомления о начале TCP-соединения для передачи файла.

Формат сообщений

Сообщения между клиентом и сервером имеют следующую структуру (в текстовом виде):

[время] [имя_пользователя]: сообщение

Например:

[01:24:55] user1: Привет, как дела?

Приватное сообщение:

/pm user2 Привет! Это личное сообщение.

Команда передачи файла:

/file TCP.txt 5000

2. Протокол TCP

TCP применяется для передачи файлов напрямую между двумя клиентами. После получения команды **/file**, клиент-получатель запускает TCP-сервер на указанном порту и ожидает подключение. Отправитель подключается по этому порту и передает файл.

Пример сценария:

1. user1: /file TCP.txt 5000
2. user2: [Клиент] Автоматический запуск приёма файла на порту 5000
3. user1: запускается tcp_file_sender 127.0.0.1 5000 TCP.txt
4. user2: файл сохраняется как received_TCP

Формат передачи TCP:

TCP соединение передает файл в виде:

- заголовок: размер файла (целое число)
- содержимое файла (байты)

Описание реализации

1. Архитектура

Программа состоит из 4 компонентов:

1. udp_chat_server.cpp — сервер, принимает сообщения, пересылает клиентам.
2. udp_chat_client.cpp — клиент, поддерживает команды чата, авторизацию и обработку команд.
3. tcp_file_sender.cpp — отправка файла по TCP.
4. tcp_file_receiver.cpp — приём файла по TCP.

2. Авторизация

При подключении клиент отправляет логин и пароль:

```
std::cout << "Введите логин: ";
std::getline(std::cin, login);
std::cout << "Введите пароль: ";
std::getline(std::cin, password);
```

Сервер проверяет в хардкоденном списке:

```
std::map<std::string, std::string> users = {
    {"user1", "password1"},
    {"user2", "password2"},
    {"admin", "adminpassword"}
};
```

Если авторизация прошла — сервер отправляет приветствие.

3. Обычные и приватные сообщения

Клиент отправляет обычное сообщение:

```
std::string formatted = "[" + time_str + "] " + username + ": " + message;
std::cout << "Получено: " << formatted << std::endl;
```

Приватные — при вводе /pm <user> <сообщение>:

```
if (message.substr(0, 4) == "/pm ") {
```

Сервер получает и пересылает только указанному получателю.

4. Передача файла

Отправитель вводит в чате команду:

/file <имя_файла> <порт>

Пример:

/file TCP.txt 5000

После этого клиент:

- Отправляет сообщение другим участникам чата через UDP, сообщая о начале передачи файла.
- Автоматически запускает процесс передачи файла через TCP, выполнив команду:

```
std::string sender_command = "./tcp_file_sender 127.0.0.1 " + std::to_string(port) + " " + filename;
system(sender_command.c_str());
```

Получатель, получив сообщение о передаче, автоматически запускает серверную часть TCP (приемник), выполнив:

```
std::string command = "./tcp_file_receiver 127.0.0.1 " + std::to_string(port) + " received_" + filename;
```

Программа tcp_file_receiver.cpp ожидает входящее TCP-соединение, принимает файл и сохраняет его как received_TCP.

Работа программы

```
itzavangard@DESKTOP-SOMTGK4:~/NP/coursework$ ./udp_chat_server
UDP чат-сервер запущен на порту 12345
```

```
itzavangard@DESKTOP-SOMTGK4:~/NP/coursework$ ./udp_chat_client
Введите логин: user1
Введите пароль: password1
[Сервер] Привет, user1! Ты успешно авторизовался.
```

```
itzavangard@DESKTOP-SOMTGK4:~/NP/coursework$ ./udp_chat_client
Введите логин: user2
Введите пароль: password2
[Сервер] Привет, user2! Ты успешно авторизовался.
```

```
itzavangard@DESKTOP-SOMTGK4:~/NP/coursework$ ./udp_chat_client
Введите логин: user1
Введите пароль: password1
[Сервер] Привет, user1! Ты успешно авторизовался.
Привет, user2!
/pm user2 Привет, user2!
/file somefile.txt 5000
File sent successfully
```

```
itzavangard@DESKTOP-SOMTGK4:~/NP/coursework$ ./udp_chat_client
Введите логин: user2
Введите пароль: password2
[Сервер] Привет, user2! Ты успешно авторизовался.
[03:12:33] user1: Привет, user2!
[Приватное сообщение от user1]:  Привет, user2!
[03:15:49] user1: [Сервер] Перейдите на TCP для передачи файла: somefile.txt
[Клиент] Автоматический запуск приёма файла на порту 5000...
[TCP Receiver] Ожидание подключения на 127.0.0.1:5000...
[TCP Receiver] Файл получен и сохранён как received_TCP
```

```
itzavangard@DESKTOP-SOMTGK4:~/NP/coursework$ ./udp_chat_server
UDP чат-сервер запущен на порту 12345
Получено: [03:12:33] user1: Привет, user2!
Получено: [03:15:49] user1: [Сервер] Перейдите на TCP для передачи файла: somefile.txt
```

Листинг

udp_chat_server.cpp:

```
#include <iostream>
#include <map>
#include <vector>
#include <ctime>
#include <cstring>
#include <arpa/inet.h>
#include <unistd.h>
#include <sstream>

#define PORT 12345
#define BUFFER_SIZE 1024

std::string current_time() {
    time_t now = time(nullptr);
    char buf[64];
    strftime(buf, sizeof(buf), "%H:%M:%S", localtime(&now));
    return std::string(buf);
}

std::map<std::string, std::string> users = {
    {"user1", "password1"},
    {"user2", "password2"},
    {"admin", "adminpassword"}
};

int main() {
    int sockfd;
    char buffer[BUFFER_SIZE];
    struct sockaddr_in server_addr{}, client_addr{};
    socklen_t addr_len = sizeof(client_addr);

    std::map<std::string, sockaddr_in> clients;
    std::map<std::string, std::string> usernames;

    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
```

```
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);

    if (bind(sockfd, (const struct sockaddr *)&server_addr,
sizeof(server_addr)) < 0) {
        perror("bind failed");
        close(sockfd);
        exit(EXIT_FAILURE);
    }

    std::cout << "UDP чат-сервер запущен на порту " << PORT << std::endl;

    while (true) {
        memset(buffer, 0, BUFFER_SIZE);
        int n = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, (struct sockaddr
*)&client_addr, &addr_len);
        if (n <= 0) continue;
        buffer[n] = '\0';

        std::string client_ip = inet_ntoa(client_addr.sin_addr);
        uint16_t client_port = ntohs(client_addr.sin_port);
        std::string client_id = client_ip + ":" + std::to_string(client_port);

        std::string message(buffer);

        if (usernames.find(client_id) == usernames.end()) {
            std::string login, password;
            std::istringstream ss(message);
            ss >> login >> password;

            if (users.find(login) != users.end() && users[login] == password)
{
                usernames[client_id] = login;
                clients[client_id] = client_addr;
                std::string welcome = "[Сервер] Привет, " + login + "! Ты
успешно авторизовался.";
                sendto(sockfd, welcome.c_str(), welcome.size(), 0, (struct
sockaddr *)&client_addr, addr_len);
            } else {
```

```
                std::string error = "[Сервер] Неверный логин или пароль.";
                sendto(sockfd, error.c_str(), error.size(), 0, (struct
sockaddr *)&client_addr, addr_len);
            }
            continue;
        }

        std::string username = usernames[client_id];
        std::string time_str = current_time();

        if (message.substr(0, 4) == "/pm ") {
            std::string recipient = message.substr(4, message.find(' ', 4) -
4);
            std::string private_message = message.substr(message.find(' ', 4)
+ 1);

            bool found = false;
            for (const auto &pair : clients) {
                if (usernames[pair.first] == recipient) {
                    sendto(sockfd, private_message.c_str(),
private_message.size(), 0,
                           (const struct sockaddr *)&pair.second,
sizeof(pair.second));
                    found = true;
                    break;
                }
            }
            if (!found) {
                std::string error_msg = "[Сервер] Пользователь " + recipient +
" не найден.";
                sendto(sockfd, error_msg.c_str(), error_msg.size(), 0, (const
struct sockaddr *)&client_addr, addr_len);
            }
            continue;
        }

        std::string formatted = "[" + time_str + "] " + username + ": " +
message;
        std::cout << "Получено: " << formatted << std::endl;

        for (const auto &pair : clients) {
            if (pair.first != client_id) {
```

```
                sendto(sockfd, formatted.c_str(), formatted.size(), 0,
                       (const struct sockaddr *)&pair.second,
sizeof(pair.second));
            }
        }
    }

    close(sockfd);
    return 0;
}
```

udp_chat_client.cpp:

```
#include <iostream>
#include <thread>
#include <cstring>
#include <arpa/inet.h>
#include <unistd.h>
#include <sstream>

#define SERVER_PORT 12345
#define BUFFER_SIZE 1024

void receive_messages(int sockfd) {
    char buffer[BUFFER_SIZE];
    while (true) {
        memset(buffer, 0, BUFFER_SIZE);
        recv(sockfd, buffer, BUFFER_SIZE, 0);
        std::string msg(buffer);
        std::cout << msg << std::endl;

        if (msg.find("[Сервер] Перейдите на TCP для передачи файла:") !=
std::string::npos) {
            std::istringstream iss(msg);
            std::string skip, skip2, skip3, skip4, skip5, filename;
            iss >> skip >> skip2 >> skip3 >> skip4 >> skip5 >> filename;

            int port = 5000;
            std::string command = "./tcp_file_receiver 127.0.0.1 " +
std::to_string(port) + " received_" + filename;
```

```
            std::cout << "[Клиент] Автоматический запуск приёма файла на порту
" << port << "..." << std::endl;

            std::thread file_receiver_thread([command]() {
                system(command.c_str());
            });
            file_receiver_thread.detach();
        }
    }
}

int main() {
    int sockfd;
    struct sockaddr_in servaddr{};

    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(SERVER_PORT);
    servaddr.sin_addr.s_addr = INADDR_ANY;

    connect(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr));

    std::string login, password;
    std::cout << "Введите логин: ";
    std::getline(std::cin, login);
    std::cout << "Введите пароль: ";
    std::getline(std::cin, password);

    std::string auth_data = login + " " + password;
    send(sockfd, auth_data.c_str(), auth_data.size(), 0);

    std::thread receiver(receive_messages, sockfd);

    while (true) {
        std::string msg;
        std::getline(std::cin, msg);

        if (msg.substr(0, 4) == "/pm ") {
```

```
            std::stringstream ss(msg);
            std::string command, recipient, private_message;
            ss >> command >> recipient;
            std::getline(ss, private_message);
            private_message = "[Приватное сообщение от " + login + "]: " +
private_message;

            std::string pm_msg = "/pm " + recipient + " " + private_message;
            send(sockfd, pm_msg.c_str(), pm_msg.size(), 0);
            continue;
        }

        if (msg.substr(0, 5) == "/file") {
            std::stringstream ss(msg);
            std::string command, filename;
            int port;
            ss >> command >> filename >> port;

            std::string file_transfer_message = "[Сервер] Перейдите на TCP
для передачи файла: " + filename;
            send(sockfd, file_transfer_message.c_str(),
file_transfer_message.size(), 0);

            std::this_thread::sleep_for(std::chrono::seconds(2));
            std::string sender_command = "./tcp_file_sender 127.0.0.1 " +
std::to_string(port) + " " + filename;
            system(sender_command.c_str());
            continue;
        }

        send(sockfd, msg.c_str(), msg.size(), 0);
    }

    receiver.join();
    close(sockfd);
    return 0;
}
```

tcp_file_sender.cpp:

```cpp
#include <iostream>
#include <fstream>
#include <cstring>
#include <arpa/inet.h>
#include <unistd.h>

void send_file(const char *ip, int port, const char *filename) {
    int sockfd;
    struct sockaddr_in servaddr;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(port);
    inet_pton(AF_INET, ip, &servaddr.sin_addr);

    if (connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("Connection failed");
        close(sockfd);
        exit(EXIT_FAILURE);
    }

    std::ifstream file(filename, std::ios::binary);
    if (!file) {
        std::cerr << "Failed to open file\n";
        close(sockfd);
        return;
    }

    char buffer[1024];
    while (file.read(buffer, sizeof(buffer))) {
        send(sockfd, buffer, file.gcount(), 0);
    }
    send(sockfd, buffer, file.gcount(), 0);

    std::cout << "File sent successfully\n";
```

```
    file.close();
    close(sockfd);
}

int main(int argc, char *argv[]) {
    if (argc != 4) {
        std::cerr << "Usage: " << argv[0] << " <IP> <Port> <FileName>\n";
        return 1;
    }

    send_file(argv[1], std::stoi(argv[2]), argv[3]);
    return 0;
}
```

tcp_file_receiver.cpp:

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <arpa/inet.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    if (argc != 4) {
        std::cerr << "Usage: " << argv[0] << " <IP> <Port> <OutputFile>\n";
        return 1;
    }

    const char* ip = argv[1];
    int port = std::stoi(argv[2]);
    const char* filename = argv[3];

    int server_fd, new_socket;
    struct sockaddr_in address{};
    socklen_t addrlen = sizeof(address);

    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        return 1;
    }
```

```
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = inet_addr(ip);
    address.sin_port = htons(port);

    if (bind(server_fd, (struct sockaddr*)&address, sizeof(address)) < 0) {
        perror("bind failed");
        close(server_fd);
        return 1;
    }

    if (listen(server_fd, 1) < 0) {
        perror("listen");
        close(server_fd);
        return 1;
    }

    std::cout << "[TCP Receiver] Ожидание подключения на " << ip << ":" <<
port << "...\n";

    if ((new_socket = accept(server_fd, (struct sockaddr*)&address, &addrlen))
< 0) {
        perror("accept");
        close(server_fd);
        return 1;
    }

    std::ofstream outfile(filename, std::ios::binary);
    if (!outfile) {
        std::cerr << "Не удалось открыть файл для записи\n";
        close(new_socket);
        close(server_fd);
        return 1;
    }

    char buffer[1024];
    ssize_t bytes_read;
    while ((bytes_read = read(new_socket, buffer, sizeof(buffer))) > 0) {
        outfile.write(buffer, bytes_read);
    }

    std::cout << "[TCP Receiver] Файл получен и сохранён как " << filename <<
std::endl;
```

```
    outfile.close();
    close(new_socket);
    close(server_fd);
    return 0;
}
```

Makefile:

```
all: udp_chat_server udp_chat_client tcp_file_sender tcp_file_receiver

udp_chat_server: udp_chat_server.cpp
    g++ udp_chat_server.cpp -o udp_chat_server

udp_chat_client: udp_chat_client.cpp
    g++ udp_chat_client.cpp -o udp_chat_client -pthread

tcp_file_sender: tcp_file_sender.cpp
    g++ tcp_file_sender.cpp -o tcp_file_sender

tcp_file_receiver: tcp_file_receiver.cpp
    g++ tcp_file_receiver.cpp -o tcp_file_receiver
```

Список источников

1. Павский К. В Введение в разработку сетевых приложений (протоколы TCP/IP, клиент-сервер, PCAP): Учебное пособие / Сибирский государственный университет телекоммуникаций и информатики. – Новосибирск, 2020. – 91 с.
2. 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.
3. UDP: Электронный ресурс [https://ru.wikipedia.org/wiki/UDP]
4. TCP: Электронный ресурс [https://ru.wikipedia.org/wiki/TCP]