

Министерство цифрового развития, связи и  
массовых коммуникаций Российской Федерации

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

**Отчет**  
по курсовой работе  
по дисциплине «Сетевое программирование»

Разработка сетевого приложения «Чат».  
Мультипоточная реализация сервера на базе протокола TCP; PTHREAD

Выполнил:

студент гр. ИС-142

«\_\_» мая 2024 г.

\_\_\_\_\_

/Григорьев Ю.В./

Проверил:

ассистент

«\_\_» мая 2024 г.

\_\_\_\_\_

/Третьяков Г.Н./

Оценка « \_\_\_\_\_ »

Новосибирск 2024

## **СОДЕРЖАНИЕ**

ПОСТАНОВКА ЗАДАЧИ	3
ОПИСАНИЕ ПРОТОКОЛА	3
ВЫПОЛНЕНИЕ РАБОТЫ	6
ДЕМОНСТРАЦИЯ РАБОТЫ	7
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	7
ПРИЛОЖЕНИЕ	8

## ПОСТАНОВКА ЗАДАЧИ

Требуется разработать сетевое приложение «Чат», используя библиотеку pthread для обработки запросов на сервере, для коммуникаций - протокол TCP. К чату подключаются пользователи со своими username с помощью программы-клиента, и могут писать в него сообщения, которые будут отображаться у всех пользователей в их клиентах.

**Дополнительное задание:** добавить вход в чат по паролю, заданному на сервере.

## ОПИСАНИЕ ПРОТОКОЛА

**TCP** (*Transmission Control Protocol*) - протокол транспортного уровня, который обеспечивает надежную и упорядоченную передачу данных между устройствами в сети.

### Особенности:

1. Надежная доставка данных: TCP гарантирует, что данные будут доставлены в назначенное место в правильном порядке и без потерь.
2. Управление потоком: TCP контролирует скорость передачи данных между отправителем и получателем, чтобы предотвратить перегрузку сети.
3. Установка соединения: Протокол TCP использует трехступенчатое установление соединения (three-way handshake) для установки связи между отправителем и получателем (request - accept - connect).
4. Управление ошибками: TCP включает в себя механизмы обнаружения и коррекции ошибок, чтобы обеспечить надежную передачу данных.
5. Подтверждение получения: Получатель отправляет подтверждения (ACK) отправителю для подтверждения приема данных.
6. Передача потоком (stream): Данные передаются как непрерывный поток байтов, а не в виде отдельных пакетов, как в UDP.

### Процесс передачи данных по TCP:

1. Установление соединения: Отправитель и получатель обмениваются специальными пакетами (сегментами) для установления соединения.

2. Передача данных: После установления соединения данные передаются в виде потока байтов от отправителя к получателю.
3. Подтверждение получения: Получатель отправляет подтверждения (ACK) для каждого полученного сегмента.
4. Управление потоком и ошибками: TCP использует механизмы управления потоком и контроля ошибок для регулирования скорости передачи данных и обеспечения надежности передачи.
5. Завершение соединения: После передачи данных соединение завершается по инициативе одной из сторон или обеих сторон, используя четырехступенчатый процесс завершения соединения (four-way handshake) (request - accept - finish - finish)

### **Примеры использования TCP:**

1. Веб-серверы и HTTP: TCP широко используется для передачи веб-страниц и другого контента через протокол HTTP.
2. Электронная почта (SMTP, POP3, IMAP): Почтовые серверы используют TCP для передачи электронных писем.
3. Файловые передачи (FTP): TCP используется для передачи файлов между клиентами и серверами FTP.
4. Удаленный доступ (SSH, Telnet): Протоколы для удаленного доступа используют TCP для обмена командами и данными между удаленными узлами.
5. Базы данных (MySQL, PostgreSQL, etc.): Базы данных используют TCP для соединения с клиентами и передачи данных.

### **Преимущества:**

1. Надежная доставка данных.
2. Управление потоком и ошибками.
3. Упорядоченная доставка данных.
4. Поддержка установления соединения и завершения.

### **Недостатки:**

1. Более высокая накладная нагрузка из-за механизмов контроля ошибок и управления потоком.

2. Медленнее, чем UDP из-за дополнительной сложности передачи данных.

### **TCP-заголовок:**

1. Source Port (Исходный порт): (16 бит)
2. Destination Port (Порт назначения): (16 бит)
3. Sequence Number (Порядковый номер): (32 бита)
4. Номер первого байта в сегменте данных. Используется для управления упорядоченной доставкой и перестройки данных на приемной стороне.
5. Acknowledgment Number (Номер подтверждения): (32 бита)  
Указывает на следующий ожидаемый байт, который получатель ожидает получить от отправителя. Используется для подтверждения приема данных.
6. Data Offset (Смещение данных): (4 бита)  
Размер заголовка TCP в 32-битных словах. Используется для определения начала данных в сегменте TCP.
7. Reserved (Зарезервировано): (6 бит)  
В настоящее время не используется и должно быть установлено в ноль.

### **8. Flags (Флаги): (6 бит)**

1. SYN (Synchronize): Для инициализации установки соединения.  
Когда клиент хочет установить соединение с сервером, он отправляет пакет с установленным флагом SYN. При получении такого пакета сервер отвечает пакетом с установленными флагами SYN и ACK, чтобы подтвердить получение и готовность к установке соединения.
2. ACK (Acknowledgment): Для подтверждения получения данных. Когда установлен этот флаг, это означает, что поле Acknowledgment Number содержит действительный номер следующего ожидаемого байта.
3. FIN (Finish): Для завершения соединения. Когда одна из сторон хочет закрыть соединение, она отправляет пакет с установленным флагом FIN. Получив такой пакет, другая сторона также отправляет пакет с установленным флагом FIN в ответ.
4. RST (Reset): Для сброса соединения. Когда сторона обнаруживает ошибку или неожиданное состояние, она отправляет пакет с установленным флагом RST, чтобы принудительно завершить соединение.

5. Window Size (Размер окна): (16 бит)

Размер окна приема, выраженный в байтах. Используется для управления потоком данных и объемом данных, которые отправитель может отправить до получения подтверждения.

6. Checksum (Контрольная сумма): (16 бит)

Используется для проверки целостности данных в сегменте TCP.

7. Urgent Pointer (Указатель на срочные данные): (16 бит)

Последний байт срочных данных в сегменте TCP. Используется, когда установлен соответствующий флаг флага URG.

## **ВЫПОЛНЕНИЕ РАБОТЫ**

Для выполнения работы были созданы файлы `client.cpp` и `server.cpp` (Приложение).

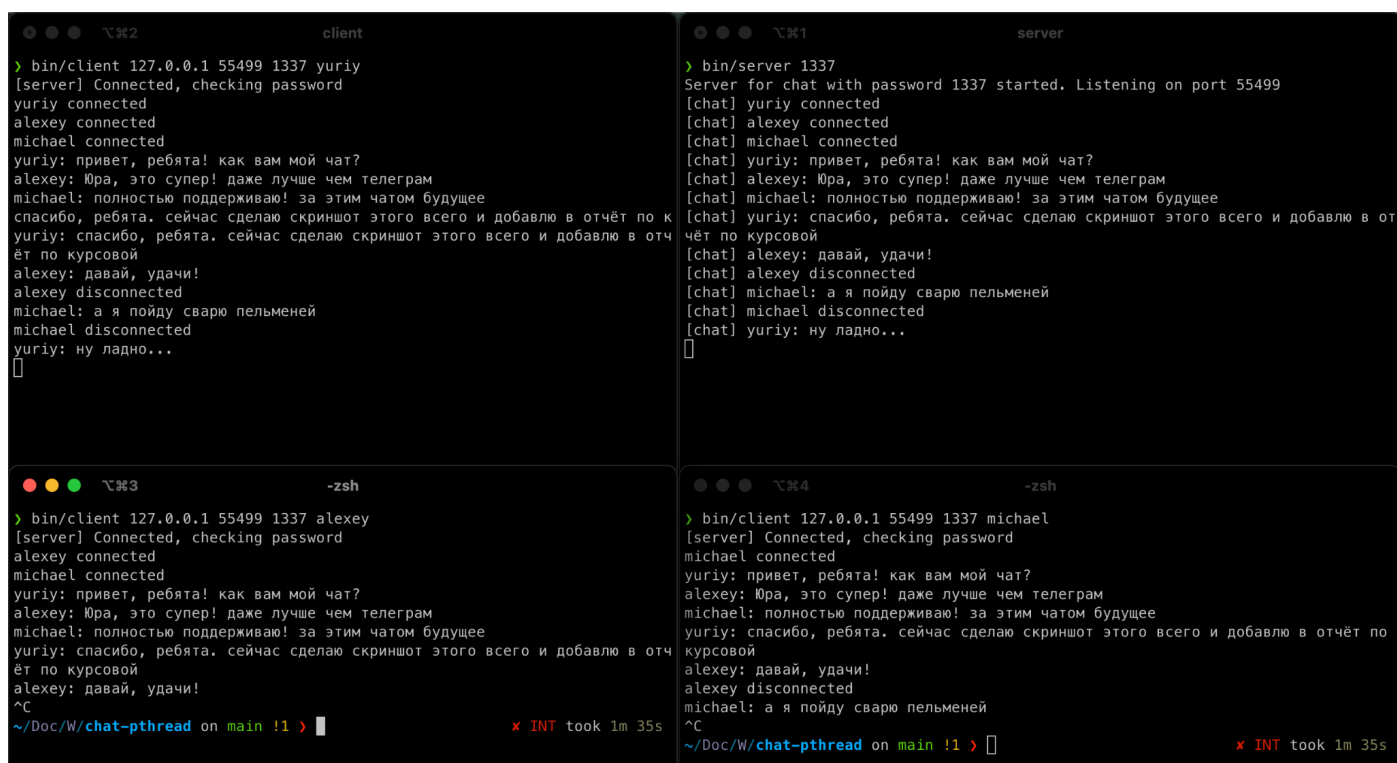
Программа-клиент работает следующим образом:

Пользователь подключается к чату по заданному IP, порту и паролю чата, также он должен указать в аргументах свой `username`, который отсылается серверу. После успешного подключения к серверу (о чем говорит соответствующее сообщение) в клиенте порождаются 2 потока также с помощью библиотеки `pthread` (как в серверной части), чтобы параллельно обрабатывать входящие от сервера и исходящие от клиента сообщения. Для передачи потокам своих задач были написаны функции `receiveMessages()` (слушает входящие сообщения) и `sendMessages()` (просит у пользователя ввод и отправляет серверу).

Программа-сервер делает следующее:

При запуске вводится только пароль чата. Создается сокет для TCP-подключений, выбирается и выводится в терминал свободный порт, на котором развернут сервер (который будет им прослушиваться). Далее при подключениях программ-клиентов и проверки их пароля на соответствие серверному, библиотекой `pthread` (функцией `pthread_create`) порождаются потоки для их обработки с помощью функции `handleClient`, также клиенты заносятся в динамический массив `clients`, где хранятся дескрипторы их сокетов в формате `int`. При получении сообщений от какого-либо клиента сервер рассылает их всем остальным. При подключении/отключении клиентов на сервере также выводятся соответствующие сообщения с указанным `username` пользователя.

## ДЕМОНСТРАЦИЯ РАБОТЫ



```
client
> bin/client 127.0.0.1 55499 1337 yuriy
[server] Connected, checking password
yuriy connected
alexey connected
michael connected
yuriy: привет, ребята! как вам мой чат?
alexey: Юра, это супер! даже лучше чем телеграм
michael: полностью поддерживаю! за этим чатом будущее
спасибо, ребята. сейчас сделаю скриншот этого всего и добавлю в отчёт по к
yuriy: спасибо, ребята. сейчас сделаю скриншот этого всего и добавлю в отч
ёт по курсовой
alexey: давай, удачи!
alexey disconnected
michael: а я пойду сварю пельменей
michael disconnected
yuriy: ну ладно...
[]

server
> bin/server 1337
Server for chat with password 1337 started. Listening on port 55499
[chat] yuriy connected
[chat] alexey connected
[chat] michael connected
[chat] yuriy: привет, ребята! как вам мой чат?
[chat] alexey: Юра, это супер! даже лучше чем телеграм
[chat] michael: полностью поддерживаю! за этим чатом будущее
[chat] yuriy: спасибо, ребята. сейчас сделаю скриншот этого всего и добавлю в от
чёт по курсовой
[chat] alexey: давай, удачи!
[chat] alexey disconnected
[chat] michael: а я пойду сварю пельменей
[chat] michael disconnected
[chat] yuriy: ну ладно...
[]

-zsh
> bin/client 127.0.0.1 55499 1337 alexey
[server] Connected, checking password
alexey connected
michael connected
yuriy: привет, ребята! как вам мой чат?
alexey: Юра, это супер! даже лучше чем телеграм
michael: полностью поддерживаю! за этим чатом будущее
yuriy: спасибо, ребята. сейчас сделаю скриншот этого всего и добавлю в отч
ёт по курсовой
alexey: давай, удачи!
^C
~/Doc/W/chat-pthread on main !1 > x INT took 1m 35s

-zsh
> bin/client 127.0.0.1 55499 1337 michael
[server] Connected, checking password
michael connected
yuriy: привет, ребята! как вам мой чат?
alexey: Юра, это супер! даже лучше чем телеграм
michael: полностью поддерживаю! за этим чатом будущее
yuriy: спасибо, ребята. сейчас сделаю скриншот этого всего и добавлю в отчёт по
курсовой
alexey: давай, удачи!
alexey disconnected
michael: а я пойду сварю пельменей
^C
~/Doc/W/chat-pthread on main !1 > x INT took 1m 35s
```

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Фейт С. TCP/IP: Архитектура, протоколы, реализация (включая IP версии 6 и IP Security). – М.: Лори, 2000. – 424 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. Протоколы TCP/IP и разработка сетевых приложений: учеб. пособие / К.В. Павский; Сиб. гос. ун-т телекоммуникаций и информатики. - Новосибирск: СибГУТИ, 2013. – 130с.

## ПРИЛОЖЕНИЕ

Исходный код программ:

### Файл **server.cpp**

```
#include <algorithm>
#include <arpa/inet.h>
#include <cstring>
#include <iostream>
#include <netinet/in.h>
#include <pthread.h>
#include <string>
#include <sys/socket.h>
#include <unistd.h>
#include <vector>

#define SERVER_ACCEPT '1'
#define SERVER_DENY '0'
#define MAX_CLIENTS 10

pthread_mutex_t clients_mutex = PTHREAD_MUTEX_INITIALIZER;

std::string prefix = "[chat] ";
std::string password;
std::vector<int> clients;

void *handleClient(void *arg) {
    int clientSocket = *((int *)arg);
    char buffer[1024];

    // Get password
    int bytesReceived = recv(clientSocket, buffer, 1024, 0);
    if (bytesReceived <= 0) {
        std::cerr << "Error receiving username\n";
        close(clientSocket);
        pthread_exit(nullptr);
    }
    std::string received_password(buffer, bytesReceived);

    // Check password
    char answer[1] = {SERVER_ACCEPT};
    if (received_password != password) {
        answer[0] = SERVER_DENY;
        send(clientSocket, answer, 1, 0);
        close(clientSocket);
        pthread_exit(nullptr);
    }
    send(clientSocket, answer, 1, 0);

    // Get username
    bytesReceived = recv(clientSocket, buffer, 1024, 0);
    if (bytesReceived <= 0) {
        std::cerr << "Error receiving username\n";
        close(clientSocket);
        pthread_exit(nullptr);
    }
    std::string username(buffer, bytesReceived);
```



```

// Send join message
pthread_mutex_lock(&clients_mutex);
std::string message = username + " connected";
for (int client : clients) {
    send(client, message.c_str(), message.size(), 0);
}
pthread_mutex_unlock(&clients_mutex);
std::cout << prefix << message << '\n';

while (true) {
    bytesReceived = recv(clientSocket, buffer, 1024, 0);

    // Client disconnected
    if (bytesReceived <= 0) {
        pthread_mutex_lock(&clients_mutex);
        auto it = std::find(clients.begin(), clients.end(), clientSocket);
        if (it != clients.end()) {
            clients.erase(it);
        }
        message = username + " disconnected";
        for (int client : clients) {
            send(client, message.c_str(), message.size(), 0);
        }
        pthread_mutex_unlock(&clients_mutex);
        std::cout << prefix << message << '\n';

        close(clientSocket);
        pthread_exit(nullptr);
    }

    // Send message to all other clients
    pthread_mutex_lock(&clients_mutex);
    for (int client : clients) {
        if (client != clientSocket) {
            send(client, buffer, bytesReceived, 0);
        }
    }
    pthread_mutex_unlock(&clients_mutex);

    // Show chat log on server
    buffer[bytesReceived] = '\0';
    std::cout << prefix << buffer << '\n';
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        std::cerr << "Usage: " << argv[0] << " <password>\n";
        return 1;
    }
    password = argv[1];

    int serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSocket == -1) {
        std::cerr << "Error creating socket\n";
        return 1;
    }
}

```

```

    struct sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(0);

    if (bind(serverSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) ==
        -1) {
        std::cerr << "Error binding socket\n";
        return 1;
    }

    socklen_t len = sizeof(serverAddr);
    getsockname(serverSocket, (struct sockaddr *)&serverAddr, &len);
    int port = ntohs(serverAddr.sin_port);
    std::cout << "Server for chat with password " << password
        << " started. Listening on port " << port << '\n';

    if (listen(serverSocket, MAX_CLIENTS) == -1) {
        std::cerr << "Error listening on socket\n";
        return 1;
    }

    while (true) {
        int clientSocket = accept(serverSocket, nullptr, nullptr);
        if (clientSocket == -1) {
            std::cerr << "Error accepting connection\n";
            continue;
        }

        pthread_mutex_lock(&clients_mutex);
        clients.push_back(clientSocket);
        pthread_mutex_unlock(&clients_mutex);

        pthread_t thread;
        if (pthread_create(&thread, nullptr, handleClient, (void *)&clientSocket) !=
            0) {
            std::cerr << "Error creating thread\n";
        }
    }
    close(serverSocket);
    return 0;
}

```

### Файл **client.cpp**

```

#include <arpa/inet.h>
#include <cstring>
#include <iostream>
#include <netinet/in.h>
#include <pthread.h>
#include <sys/socket.h>
#include <unistd.h>

#define SERVER_ACCEPT '1'

int clientSocket;
pthread_t receiveThread;
std::string username;

```

```

void *receiveMessages(void *arg) {
    char buffer[1024];
    while (true) {
        int bytesReceived = recv(clientSocket, buffer, 1024, 0);
        if (bytesReceived <= 0) {
            std::cerr << "Error receiving message from server\n";
            break;
        }
        buffer[bytesReceived] = '\0';
        std::cout << buffer << '\n';
    }
    close(clientSocket);
    pthread_exit(nullptr);
}

void *sendMessage(void *arg) {
    while (true) {
        std::string text;
        std::getline(std::cin, text);
        std::cout << "\033[A\33[2K\r";
        std::string message = username + ": " + text;
        if (send(clientSocket, message.c_str(), message.size(), 0) == -1) {
            std::cerr << "Error sending message\n";
            break;
        }
        std::cout << message << '\n';
    }
    pthread_cancel(receiveThread);
    close(clientSocket);
    pthread_exit(nullptr);
}

int main(int argc, char *argv[]) {
    if (argc != 5) {
        std::cerr << "Usage: " << argv[0]
            << " <server_ip> <port> <password> <username>\n";
        return 1;
    }
    const char *serverIp = argv[1];
    int PORT = atoi(argv[2]);
    std::string password = argv[3];
    username = argv[4];

    clientSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (clientSocket == -1) {
        std::cerr << "Error creating socket\n";
        return 1;
    }

    struct sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(PORT);
    inet_pton(AF_INET, serverIp, &serverAddr.sin_addr);

    if (connect(clientSocket, (struct sockaddr *)&serverAddr,
        sizeof(serverAddr)) == -1) {
        std::cerr << "Error connecting to server\n";
        return 1;
    }
}

```

```

std::cout << "[server] Connected, checking password\n";

// Send password
send(clientSocket, password.c_str(), password.size(), 0);
// Get answer (accept/deny)
char answer[1];
int received = recv(clientSocket, answer, 1, 0);
if (received <= 0) {
    std::cerr << "Error receiving answer from server\n";
    return 1;
}
if (answer[0] != SERVER_ACCEPT) {
    std::cerr << "[server] Wrong password\n";
    return 1;
}

// Send username
send(clientSocket, username.c_str(), username.size(), 0);

// Use pthreads to receive & send messages
pthread_create(&receiveThread, nullptr, receiveMessages, nullptr);
pthread_t sendThread;
pthread_create(&sendThread, nullptr, sendMessage, nullptr);
pthread_join(receiveThread, nullptr);
pthread_cancel(sendThread);
return 0;
}

```