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"

Кафедра вычислительных систем

**Курсовая работа  
 по дисциплине  
 Сетевое программирование**

**Разработка сетевого приложения «Чат».**

Мультипоточная реализация сервера с установлением

соединения с использованием библиотеки PTHREAD.

Выполнил: студент гр.ИП-213

Дмитриев Антон Александрович

ФИО студента

Новосибирск 2025 г.

# Содержание

[Содержание 2](#_Toc195812079)

[Постановка задачи 3](#_Toc195812080)

[Описание протокола TCP/IP 4](#_Toc195812081)

[Описание реализации 5](#_Toc195812082)

[Сервер 5](#_Toc195812083)

[Клиент 5](#_Toc195812084)

[Скан экрана работы программы 6](#_Toc195812085)

[Текст программы 7](#_Toc195812086)

[Server.cpp : 7](#_Toc195812087)

[Client.cpp : 8](#_Toc195812088)

[Список источников 11](#_Toc195812089)

# Постановка задачи

Целью данной лабораторной работы является разработка сетевого приложения «Чат» с поддержкой множественных клиентов. Требуется реализовать **многопоточный сервер** с использованием библиотеки PTHREAD, обеспечивающий обмен сообщениями между всеми подключёнными клиентами. Каждый клиент должен иметь возможность:

* установить соединение с сервером,
* отправлять текстовые сообщения,
* принимать сообщения, отправленные другими клиентами.

Сервер должен обрабатывать каждого клиента в отдельном потоке и обеспечивать широковещательную рассылку сообщений, исключая отправителя.

## 

# Описание протокола TCP/IP

В качестве транспортного уровня используется **протокол TCP** (Transmission Control Protocol), который обеспечивает:

* надёжную доставку данных,
* контроль порядка передачи,
* управление потоком и перегрузкой.

Основные этапы работы TCP-соединения:

1. **Установка соединения** (трёхстороннее рукопожатие):
   * клиент инициирует соединение (SYN),
   * сервер подтверждает (SYN-ACK),
   * клиент подтверждает (ACK).
2. **Обмен данными** — через вызовы send() и recv().
3. **Завершение соединения** — посредством FIN/ACK.

Протокол IP используется на сетевом уровне для маршрутизации пакетов между узлами. Адресация клиентов и сервера осуществляется через IPv4 (AF\_INET).

# ****Описание реализации****

## Сервер

Сервер создаёт TCP-сокет с помощью socket(AF\_INET, SOCK\_STREAM, 0) и привязывается к свободному порту через bind(). Для обработки клиентов используется многопоточность с применением библиотеки PTHREAD (через std::thread, использующий POSIX-совместимый механизм потоков).

#### Основные шаги:

1. **Ожидание подключений**:
   * listen() — установка сервера в режим ожидания;
   * accept() — принятие подключения клиента.
2. **Обработка клиента**:
   * Каждый клиент обслуживается в отдельном потоке;
   * Сервер принимает сообщения от клиента через recv();
   * Отправляет их всем другим клиентам через send();
   * Список клиентов хранится в std::vector<int> и синхронизируется с помощью std::mutex.
3. **Удаление клиента**:
   * При разрыве соединения — удаление клиента из общего списка и закрытие сокета.

#### Рассылка сообщений:

Функция broadcastMessage() отправляет сообщение каждому клиенту, кроме отправителя. Это реализует механизм группового чата.

## Клиент

Клиент подключается к серверу по IP-адресу и порту, заданным в аргументах командной строки.

#### Основные шаги:

1. **Установка соединения**:
   * socket() и connect() — создание TCP-соединения с сервером.
2. **Получение сообщений**:
   * В отдельном потоке вызывается recv(), который получает сообщения от сервера и выводит их на экран.
3. **Отправка сообщений**:
   * Пользователь вводит сообщения с клавиатуры;
   * Отправка производится через send().
4. **Завершение соединения**:
   * По команде /exit соединение разрывается, сокет закрывается.

# Скан экрана работы программы

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, Графическое программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

# Текст программы

## Server.cpp :

#include <iostream>

#include <vector>

#include <string>

#include <cstring>

#include <thread>

#include <mutex>

#include <algorithm>

#include <arpa/inet.h>

#include <unistd.h>

#include <netinet/in.h>

#include <sys/socket.h>

const int BUFFER\_SIZE = 1024;

std::vector<int> clients;

std::mutex clients\_mutex;

void broadcastMessage(const std::string& message, int senderSocket) {

std::lock\_guard<std::mutex> lock(clients\_mutex);

for (int client : clients) {

if (client != senderSocket) {

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

}

}

}

void handleClient(int clientSocket, sockaddr\_in clientAddr) {

char buffer[BUFFER\_SIZE];

while (true) {

int bytesReceived = recv(clientSocket, buffer, BUFFER\_SIZE - 1, 0);

if (bytesReceived <= 0) {

std::cout << "Клиент отключился: " << inet\_ntoa(clientAddr.sin\_addr) << ":" << ntohs(clientAddr.sin\_port) << std::endl;

break;

}

buffer[bytesReceived] = '\0';

std::string message = "[" + std::string(inet\_ntoa(clientAddr.sin\_addr)) + ":" + std::to\_string(ntohs(clientAddr.sin\_port)) + "] " + buffer;

std::cout << message << std::endl;

broadcastMessage(message, clientSocket);

}

// Удаление клиента

{

std::lock\_guard<std::mutex> lock(clients\_mutex);

clients.erase(std::remove(clients.begin(), clients.end(), clientSocket), clients.end());

}

close(clientSocket);

}

int main() {

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

if (listenerSocket < 0) {

std::cerr << "Ошибка создания сокета\n";

return 1;

}

sockaddr\_in serverAddr{};

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(0); // Автоматический выбор порта

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

if (bind(listenerSocket, (sockaddr\*)&serverAddr, sizeof(serverAddr)) < 0) {

std::cerr << "Ошибка привязки сокета\n";

close(listenerSocket);

return 1;

}

socklen\_t addrLen = sizeof(serverAddr);

getsockname(listenerSocket, (sockaddr\*)&serverAddr, &addrLen);

std::cout << "Сервер запущен на порту: " << ntohs(serverAddr.sin\_port) << std::endl;

listen(listenerSocket, 5);

while (true) {

sockaddr\_in clientAddr{};

socklen\_t clientLen = sizeof(clientAddr);

int clientSocket = accept(listenerSocket, (sockaddr\*)&clientAddr, &clientLen);

if (clientSocket < 0) {

std::cerr << "Ошибка подключения клиента\n";

continue;

}

{

std::lock\_guard<std::mutex> lock(clients\_mutex);

clients.push\_back(clientSocket);

}

std::cout << "Клиент подключился: " << inet\_ntoa(clientAddr.sin\_addr) << ":" << ntohs(clientAddr.sin\_port) << std::endl;

std::thread t(handleClient, clientSocket, clientAddr);

t.detach();

}

close(listenerSocket);

return 0;

}

## Client.cpp :

#include <iostream>

#include <string>

#include <cstring>

#include <thread>

#include <arpa/inet.h>

#include <unistd.h>

const int BUFFER\_SIZE = 1024;

void receiveMessages(int socket) {

char buffer[BUFFER\_SIZE];

int bytesReceived;

while ((bytesReceived = recv(socket, buffer, BUFFER\_SIZE - 1, 0)) > 0) {

buffer[bytesReceived] = '\0';

std::cout << "\n" << buffer << "\n> " << std::flush;

}

std::cout << "\n[Соединение с сервером потеряно]\n";

exit(0);

}

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

if (argc != 3) {

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

return 1;

}

const char\* serverIP = argv[1];

int serverPort = atoi(argv[2]);

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

if (clientSocket < 0) {

std::cerr << "Ошибка создания сокета\n";

return 1;

}

sockaddr\_in serverAddr{};

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(serverPort);

inet\_pton(AF\_INET, serverIP, &serverAddr.sin\_addr);

if (connect(clientSocket, (sockaddr\*)&serverAddr, sizeof(serverAddr)) < 0) {

std::cerr << "Ошибка подключения к серверу\n";

close(clientSocket);

return 1;

}

std::cout << "Подключено к чату. Введите сообщения (для выхода: /exit):\n";

std::thread recvThread(receiveMessages, clientSocket);

std::string message;

while (true) {

std::cout << "> ";

std::getline(std::cin, message);

if (message == "/exit") {

break;

}

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

}

close(clientSocket);

recvThread.join();

return 0;

}

# 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. Протоколы TCP/IP и разработка сетевых приложений : учеб. пособие / К.В. Павский ; Сиб. гос. ун-т телекоммуникаций и информатики. - Новосибирск : СибГУТИ, 2013. – 130c.
4. Дубаков, А. А. Сетевое программирование [Электронный ресурс] : учебное пособие / А. А. Дубаков. — Электрон. текстовые данные. — СПб. : Университет ИТМО, 2013. — 249 c. — 2227-8397. — Режим доступа: <http://www.iprbookshop.ru/68118.html> *Лицензия: до 01.10.2022*
5. Олифер, В. Г. Основы сетей передачи данных [Электронный ресурс] / В. Г. Олифер, Н. А. Олифер. — 2-е изд. — Электрон. текстовые данные. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 219 c. — 2227-8397. — Режим доступа: <http://www.iprbookshop.ru/73702.html> *Лицензия: до 23.01.2021*
6. Семенов, Ю. А. Алгоритмы телекоммуникационных сетей. Часть 1. Алгоритмы и протоколы каналов и сетей передачи данных [Электронный ресурс] / Ю. А. Семенов. — Электрон. текстовые данные. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 757 c. — 978-5-94774-706-5. — Режим доступа: <http://www.iprbookshop.ru/62806.html> *Лицензия: до 31.03.2020*