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 функции fork.

Выполнил:

студент гр. ИП 214 Дмитриев Е. А.

ФИО студента

« 17 » апреля 2025 г.

Проверил

Павский К. В.

ФИО преподавателя

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

Новосибирск 2025 г.

# Содержание

[Содержание 2](#_Toc195803436)

[Постановка задачи 3](#_Toc195803437)

[Описание протокола TCP/IP 4](#_Toc195803438)

[Описание реализации 5](#_Toc195803439)

[Сервер 5](#_Toc195803440)

[Клиент 5](#_Toc195803441)

[Скан экрана работы программы 6](#_Toc195803442)

[Текст программы 7](#_Toc195803443)

[Server.cpp : 7](#_Toc195803444)

[Client.cpp : 11](#_Toc195803445)

[Список источников 13](#_Toc195803446)

# Постановка задачи

Целью данной работы является разработка многопроцессорного сетевого чат-сервера с поддержкой подключения нескольких клиентов. Все клиенты должны иметь возможность отправлять и получать сообщения друг от друга в режиме реального времени. Сервер должен реализовывать обработку каждого клиента в отдельном процессе с использованием системного вызова fork(), обеспечивая таким образом параллельную обработку соединений.

Основные требования к реализации:

* сервер работает по протоколу TCP,
* каждый подключившийся клиент может свободно общаться с другими клиентами,
* для каждого клиента сервер создаёт отдельный дочерний процесс,
* завершение соединения одним клиентом не должно влиять на работу других клиентов,
* клиент может в любой момент выйти из чата с помощью команды exit.

## 

# 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****

Система состоит из двух компонентов:

* **серверное приложение** (server.cpp),
* **клиентское приложение** (client.cpp).

## Сервер

Сервер создаёт TCP-сокет, привязывается к локальному адресу и начинает прослушивание порта. При подключении нового клиента вызывается fork(), создавая дочерний процесс, который обрабатывает коммуникацию с этим клиентом. Все процессы используют общий набор дескрипторов и обмениваются сообщениями через каналы (pipe), чтобы пересылать данные от одного клиента ко всем остальным. Основной процесс сервера следит за чтением из каналов и перенаправляет сообщения всем клиентам, кроме отправителя.

Также реализована функция reaper, которая обрабатывает завершение дочерних процессов (через сигнал SIGCHLD), чтобы избежать появления "зомби"-процессов.

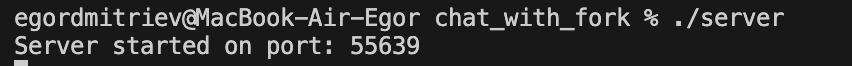
## Клиент

Клиент подключается к серверу через IP-адрес и порт. После успешного соединения вызывается fork():

* в родительском процессе пользователь вводит сообщения, которые отправляются на сервер,
* в дочернем процессе клиент получает сообщения от сервера и выводит их в консоль.

Обмен сообщениями происходит построчно. Добавлена автоматическая корректировка форматирования сообщений — каждое сообщение выводится с новой строки. Клиент завершает соединение по команде exit, при этом также завершается дочерний процесс клиента.

# Скан экрана работы программы

****

**Изображение выглядит как текст, снимок экрана, черный

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как снимок экрана, текст, черный

Контент, сгенерированный ИИ, может содержать ошибки.**

# Текст программы

## Server.cpp :

#include <iostream>

#include <vector>

#include <map>

#include <string>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <csignal>

#include <cstring>

#include <sys/wait.h>

#include <fcntl.h>

#define MAX\_CLIENTS 10

#define BUFFER\_SIZE 1024

struct ClientInfo {

int fd;

int pipe\_fd;

pid\_t pid;

};

std::vector<ClientInfo> clients;

std::map<pid\_t, int> pipe\_from\_pid;

void reap\_zombies(int) {

while (waitpid(-1, nullptr, WNOHANG) > 0);

}

void broadcast(const std::string& message, int exclude\_fd = -1) {

for (const auto& client : clients) {

if (client.fd != exclude\_fd) {

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

}

}

}

int main() {

signal(SIGCHLD, reap\_zombies);

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

if (listen\_fd < 0) {

perror("socket");

return 1;

}

sockaddr\_in server\_addr{};

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

server\_addr.sin\_port = 0;

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

if (bind(listen\_fd, (sockaddr\*)&server\_addr, sizeof(server\_addr)) < 0) {

perror("bind");

return 1;

}

socklen\_t len = sizeof(server\_addr);

getsockname(listen\_fd, (sockaddr\*)&server\_addr, &len);

std::cout << "Server started on port: " << ntohs(server\_addr.sin\_port) << std::endl;

listen(listen\_fd, MAX\_CLIENTS);

fd\_set readfds;

while (true) {

FD\_ZERO(&readfds);

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

int max\_fd = listen\_fd;

for (const auto& client : clients) {

FD\_SET(client.pipe\_fd, &readfds);

if (client.pipe\_fd > max\_fd) max\_fd = client.pipe\_fd;

}

if (select(max\_fd + 1, &readfds, nullptr, nullptr, nullptr) < 0) {

perror("select");

continue;

}

// Новое подключение

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

int client\_fd = accept(listen\_fd, nullptr, nullptr);

if (client\_fd < 0) {

perror("accept");

continue;

}

if (clients.size() >= MAX\_CLIENTS) {

std::string msg = "Server full. Try again later.\n";

send(client\_fd, msg.c\_str(), msg.size(), 0);

close(client\_fd);

continue;

}

int pipe\_fd[2];

if (pipe(pipe\_fd) < 0) {

perror("pipe");

close(client\_fd);

continue;

}

pid\_t pid = fork();

if (pid < 0) {

perror("fork");

close(client\_fd);

close(pipe\_fd[0]);

close(pipe\_fd[1]);

continue;

}

if (pid == 0) {

// child

close(pipe\_fd[0]);

char buffer[BUFFER\_SIZE];

while (true) {

int bytes = recv(client\_fd, buffer, BUFFER\_SIZE - 1, 0);

if (bytes <= 0) break;

buffer[bytes] = '\0';

std::string msg = "[client " + std::to\_string(getpid()) + "]: " + buffer;

write(pipe\_fd[1], msg.c\_str(), msg.size());

}

close(client\_fd);

close(pipe\_fd[1]);

exit(0);

} else {

// parent

close(pipe\_fd[1]);

fcntl(pipe\_fd[0], F\_SETFL, O\_NONBLOCK); // pipe read non-block

clients.push\_back({client\_fd, pipe\_fd[0], pid});

pipe\_from\_pid[pid] = pipe\_fd[0];

std::string welcome = "Client joined: pid " + std::to\_string(pid) + "\n";

broadcast(welcome);

}

}

// Чтение из pipe'ов клиентов

char buffer[BUFFER\_SIZE];

for (auto it = clients.begin(); it != clients.end(); ) {

if (FD\_ISSET(it->pipe\_fd, &readfds)) {

int bytes = read(it->pipe\_fd, buffer, BUFFER\_SIZE - 1);

if (bytes <= 0) {

std::string msg = "Client left: pid " + std::to\_string(it->pid) + "\n";

broadcast(msg);

close(it->fd);

close(it->pipe\_fd);

pipe\_from\_pid.erase(it->pid);

it = clients.erase(it);

continue;

}

buffer[bytes] = '\0';

broadcast(buffer);

}

++it;

}

}

close(listen\_fd);

return 0;

}

## Client.cpp :

#include <iostream>

#include <cstring>

#include <arpa/inet.h>

#include <unistd.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <signal.h>

#define BUFFER\_SIZE 1024

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

if (argc != 3) {

std::cerr << "Usage: " << argv[0] << " <Server IP> <Port>" << std::endl;

return 1;

}

const char\* server\_ip = argv[1];

int port = atoi(argv[2]);

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

if (sock < 0) {

perror("socket");

return 1;

}

sockaddr\_in server\_addr{};

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

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

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

perror("inet\_pton");

return 1;

}

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

perror("connect");

return 1;

}

std::cout << "Connected to chat. Type 'exit' to leave.\n";

pid\_t pid = fork();

if (pid == 0) {

// child — receive messages

char buffer[BUFFER\_SIZE];

while (true) {

int bytes = recv(sock, buffer, BUFFER\_SIZE - 1, 0);

if (bytes <= 0) break;

buffer[bytes] = '\0';

std::cout << buffer;

// Добавляем перенос строки, если в сообщении его не было

if (buffer[bytes - 1] != '\n') std::cout << std::endl;

}

std::cout << "Disconnected from server.\n";

close(sock);

exit(0);

} else {

// parent — send messages

std::string input;

while (true) {

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

if (input == "exit") break;

if (input.back() != '\n') input += '\n';

send(sock, input.c\_str(), input.size(), 0);

}

close(sock);

kill(pid, SIGKILL); // завершение дочернего процесса

}

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*