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

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

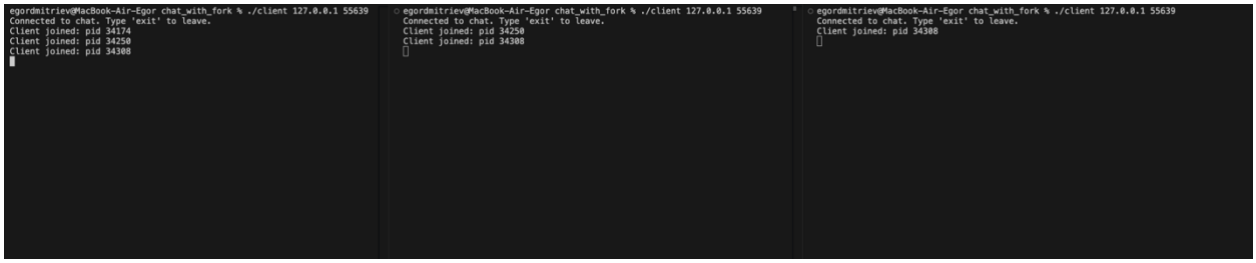
Клиент подключается к серверу через IP-адрес и порт. После успешного соединения вызывается fork():

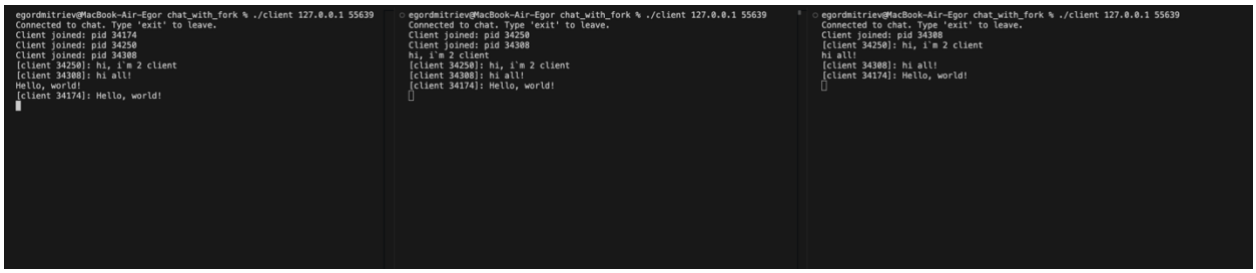
- в родительском процессе пользователь вводит сообщения, которые отправляются на сервер,
- в дочернем процессе клиент получает сообщения от сервера и выводит их в консоль.

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

```
egordmitriev@MacBook-Air-Egor chat_with_fork % ./server
Server started on port: 55639
```





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. – 130с.
4. Дубаков, А. А. Сетевое программирование [Электронный ресурс] : учебное пособие / А. А. Дубаков. — Электрон. текстовые данные. — СПб. : Университет ИТМО, 2013. — 249 с. — 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 с. — 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 с. — 978-5-94774-706-5. — Режим доступа: http://www.iprbookshop.ru/62806.html *Лицензия: до 31.03.2020*