Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Операционные системы и системное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

РАЗРАБОТКА МНОГОПОТОЧНОГО TCP СЕРВЕРА

БГУИР КП 6-05-0611-05 220 ПЗ

Студент Д.А. Манухо

Руководитель Д.А. Жалейко

МИНСК 2025

Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой

\_\_\_\_\_\_\_\_ Б.В.Никульшин

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

ЗАДАНИЕ

по курсовому проектированию

Студенту Манухо Даниилу Алексеевичу

1. Тема проекта: Разработка многопоточного TCP сервера.

2. Срок сдачи студентом законченного проекта: *10 мая 2025 г.*

3. Исходные данные к проекту: *язык программирования C*

4. Содержание расчетно—пояснительной записки (перечень вопросов, которые

*Введение. 1. Обзор литературы. 2. Постановка задачи. 3. Системное проектирование. 4. Функциональное проектирование. 5. Разработка программных модулей. 6. Результат работы. Заключение. Список использованных источников. Приложения*

5. Перечень графического материала (с точным обозначением обязательных

чертежей и графиков):

1. *Схема структурная*
2. *Диаграмма классов*
3. *Ведомость документов*

6. Консультант по проекту *Жалейко Д. А.*

7. Календарный график работы над проектом на весь период проектирования

(с обозначением сроков выполнения и трудоемкости отдельных этапов):

*раздел 1, 2 к 06.03 — 5 %;*

*разделы 3, 4 к 07.04 — 30 %;*

*разделы 5, 6, 7, 8, 9 к 05.05 — 90 %;*

*оформление пояснительной записки и графического материала к 10.05 — 100%*

*Защита курсового проекта с 28.05 по 10.06.*

РУКОВОДИТЕЛЬ  *Д. А. Жалейко*

Задание принял к исполнению  *Д. А. Манухо*

**ВВЕДЕНИЕ**

Сетевые технологии играют важную роль в современной цифровой инфраструктуре, обеспечивая взаимодействие между компьютерами, мобильными устройствами и серверами в локальных и глобальных сетях. Одним из основополагающих элементов таких взаимодействий являются серверные приложения, принимающие и обрабатывающие клиентские запросы по сети. Среди наиболее распространённых протоколов передачи данных — TCP (Transmission Control Protocol), обеспечивающий надёжную доставку сообщений и гарантированную последовательность передачи.

Серверные приложения на базе TCP широко используются в различных сферах: от мессенджеров и многопользовательских игр до систем удалённого администрирования и обмена файлами. Эффективная реализация TCP-сервера требует понимания системного программирования, работы с сокетами, многозадачности и управления ресурсами операционной системы.

Данный курсовой проект посвящён разработке многопоточного TCP-сервера на языке программирования C. Целью проекта является создание надёжного серверного приложения, способного обрабатывать множественные клиентские подключения одновременно, обеспечивая устойчивость, масштабируемость и минимальное время отклика. Особое внимание в работе уделено реализации многопоточности с использованием POSIX-потоков (pthreads), организации безопасной работы с памятью и корректному завершению потоков и соединений.

Проект может быть использован в качестве основы для создания различных клиент-серверных систем, а также служит учебным примером разработки низкоуровневых сетевых приложений с учётом особенностей взаимодействия с операционной системой.

**1 ОБЗОР ЛИТЕРАТУРЫ**

**1.1 Сервер как проект**

Любое клиент-серверное приложение, в том числе и сетевая игра, состоит из трёх основных компонентов: серверной части, клиентской и графической. Данная курсовая работа ориентирована на реализацию серверной части, обеспечивающей стабильное взаимодействие между клиентами.

TCP-сервер выполняет ключевую роль посредника и координатора в сетевом взаимодействии, обеспечивая:

* Установление и поддержание соединений с множеством клиентов одновременно.
* Получение и интерпретацию входящих сетевых пакетов.
* Передачу ответных пакетов, отражающих текущее состояние игровой или логической сессии.
* Поддержание целостности и синхронизации данных между подключёнными клиентами.
* Обработку логики взаимодействия между пользователями (или между игроком и системой).

Для этого сервер использует механизмы многопоточности, позволяющие обрабатывать соединения параллельно, тем самым обеспечивая масштабируемость и отзывчивость.

**1.2 Анализ существующих аналогов**

В рамках сетевого программирования и игровых серверов существует множество реализаций, которые послужили источником вдохновения для построения архитектуры многопоточного сервера. Ниже рассмотрены два таких проекта, демонстрирующие подходы к сетевой логике и взаимодействию с клиентами.

**1.2.1 Minecraft Server**

Сервер Minecraft — один из самых известных игровых серверов, обеспечивающих мультиплеерную среду. Он управляет состоянием мира, обрабатывает перемещения игроков, действия с объектами и синхронизирует все изменения между участниками. Сервер разработан на Java, но существуют переписанные версии (например, C++-реализация Cuberite), которые демонстрируют гибкость серверной архитектуры. Особенности:

* Поддержка множества клиентов одновременно.
* Асинхронная передача сообщений.
* Обработка коллизий и игровых событий.
* Простота масштабирования благодаря модульности.

**1.2.2 Netcat и учебные TCP-серверы**

Простые реализации TCP-серверов, такие как netcat и учебные многопоточные серверы, иллюстрируют базовую суть TCP-соединений. Они дают понимание системных вызовов, работы с сокетами, многопоточности через pthreads и синхронизации ресурсов.

**1.3 Анализ использованных средств разработки**

В рамках проекта в качестве основного языка программирования выбран язык C, благодаря своей высокой производительности и низкоуровневому доступу к системным ресурсам.

Для реализации многопоточности применяется POSIX-библиотека pthread, позволяющая:

* Создавать и управлять потоками.
* Использовать мьютексы и условные переменные для синхронизации.
* Минимизировать конфликты при доступе к разделяемым ресурсам.

Работа с сетевыми соединениями осуществляется с использованием стандартных UNIX-сокетов: socket(), bind(), listen(), accept(), recv(), send() и др. Такой подход даёт полное понимание принципов сетевого взаимодействия на низком уровне и делает проект кроссплатформенным.

**1.4 Постановка задачи**

Анализ аналогов показывает, что несмотря на различие по сложности и функциональности, все TCP-серверы следуют базовым принципам приёма, обработки и ответа на запросы клиентов. Цель данной работы — разработка многопоточного TCP-сервера, способного обрабатывать множественные подключения одновременно.

Основные задачи проекта:

* Инициализация TCP-соединений и прослушивание заданного порта.
* Обработка клиентских запросов в отдельных потоках.
* Синхронизация доступа к общим ресурсам (например, состоянию сессии).
* Безопасное завершение потоков и закрытие соединений.
* Обработка сетевых ошибок и исключительных ситуаций.
* Логирование активности клиентов и основных событий сервера.

Этот проект служит основой для построения устойчивой серверной архитектуры и может быть расширен в сторону поддержки более сложных протоколов, авторизации клиентов и взаимодействия с базами данных.

**2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ**

Проект реализован на основе модульной архитектуры, при которой серверное приложение разделено на независимые компоненты, каждый из которых выполняет строго определённую функцию. Такой подход обеспечивает:

* Повышенную читаемость и удобство сопровождения кода;
* Изоляцию функциональных блоков, что снижает количество потенциальных ошибок;
* Возможность масштабирования и модификации отдельных компонентов без затрагивания всей системы.
* Модульная структура критически важна для серверных решений, так как позволяет эффективно отлаживать, расширять и оптимизировать систему при росте нагрузки или функционала. Ниже описаны основные модули TCP-сервера и их взаимодействие.

**2.1 Модуль инициализации и запуска сервера**

Этот компонент отвечает за начальную настройку сервера и запуск его основных подсистем.Создаёт TCP-сокет, настраивает его и привязывает к указанному порту; Запускает цикл прослушивания входящих соединений; Инициализирует основные структуры и системные ресурсы (например, мьютексы, логирование); Обрабатывает системные сигналы для корректного завершения работы. Фактически, это точка входа в приложение, управляющая запуском и завершением всех процессов сервера.

**2.2 Модуль обработки клиентских подключений**

Основная задача модуля — приём входящих соединений с использованием системного вызова accept() и передача управления каждому клиенту в отдельном потоке. Он создаёт новый поток (pthread\_create) для каждого подключившегося клиента; поддерживает пул активных соединений;

Контролирует корректное завершение и очистку ресурсов после завершения соединения. Модуль обеспечивает масштабируемость сервера и его способность обрабатывать множество клиентов параллельно.

**2.3 Модуль обработки сообщений**

Каждый поток, связанный с клиентом, использует данный модуль для интерпретации и обработки входящих сообщений. Этот компонент:

* Читает данные от клиента (recv);
* Проводит разбор протокола (в простейшем виде — строки команд);
* Вызывает соответствующие функции (например, эхо-ответ, передача файла, работа с базой данных, если применимо);
* Отправляет ответ клиенту (send).

Данный модуль логически изолирован от сетевого взаимодействия и может быть адаптирован под любой прикладной протокол.

**2.4 Модуль синхронизации и безопасности данных**

Поскольку сервер работает в многопоточном режиме, важную роль играет контроль доступа к разделяемым ресурсам. Этот модуль реализует:

Мьютексы (pthread\_mutex\_t) для защиты разделяемых структур (например, логов или общего хранилища состояния);

Условные переменные (pthread\_cond\_t) при необходимости синхронизации между потоками;

Минимизацию гонок и предотвращение взаимных блокировок.

Он обеспечивает корректную и безопасную работу всех параллельных компонентов сервера.

**2.5 Модуль обработки команд**

Для работы с сервером реализован простой командный режим. С помощью него клиенты могут взаимодействовать с сервером с помощью простых команд. Чтобы перейти в режим работы с командами, клиент должен аутентифицироваться, прежде чем перейти к работе с сервером.

**2.6 Модуль обработки файлов**

С помощью этого модуля можно загружать и выгружать различные данные(файлы) на сервер и с сервера. Это делает удобным использование его вместо стандартного ftp сервера. Чтобы перейти в режим работы с файлами, клиент должен аутентифицироваться, прежде чем перейти к работе с сервером.

**3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

В данном разделе описываются основные структуры данных, используемые при разработке многопоточного TCP-сервера с поддержкой чата, выполнения команд и передачи файлов. Эти структуры определяют способ хранения состояния клиентов, управления соединениями и обработки данных.

**3.1 Структуры клиентских сессий**

Для управления состоянием каждого подключенного клиента используется структура Client, которая содержит:

typedef struct {

int socket; // Сокет соединения с клиентом

char username[MAX\_USERNAME\_LEN]; // Имя пользователя (для чата)

bool authenticated; // Флаг аутентификации

} Client;

Для управления всеми подключенными клиентами используется структура ClientList:

typedef struct {

Client\* clients[MAX\_CLIENTS]; // Массив указателей на клиентов

int count; // Текущее количество клиентов

pthread\_mutex\_t mutex; // Мьютекс для потокобезопасности

} ClientList;

Ключевые функции для работы с клиентами:

* add\_client() - добавление нового клиента
* remove\_client() - удаление клиента
* broadcast\_message() - рассылка сообщений всем клиентам чата

**3.2 Структура сервера**

Сервер использует следующие глобальные переменные и константы:

#define PORT 8080 // Порт по умолчанию

#define MAX\_CLIENTS 100 // Максимальное число клиентов

#define BUFFER\_SIZE 1024 // Размер буфера сообщений

#define MAX\_USERNAME\_LEN 50 // Макс. длина имени пользователя

#define FILE\_DIR "files/" // Директория для файлов

Основные компоненты сервера:

* Главный сокет (server\_socket) для прослушивания подключений
* Пул потоков для обработки клиентов
* Глобальная структура client\_list для хранения подключений
* Мьютексы для синхронизации доступа к общим ресурсам

**3.3 Режимы работы сервера**

Сервер поддерживает три основных режима работы:

1. Чат:
   * Хранение истории сообщений в структуре:

typedef struct {

char message[BUFFER\_SIZE];

char username[MAX\_USERNAME\_LEN];

time\_t timestamp;

} ChatMessage;

* + Функции:
    - handle\_chat\_mode() - обработка чата
    - send\_chat\_history() - отправка истории новым клиентам

1. Выполнение команд:
   * Использование popen() для выполнения системных команд
   * Функция handle\_command\_mode() для обработки запросов
2. Передача файлов:
   * Работа с директорией FILE\_DIR
   * Функции:
     + handle\_file\_transfer\_mode() - основной обработчик
     + Загрузка/скачивание через системные вызовы read()/write()

**3.4 Сетевое взаимодействие**

Для работы с сетью используются:

* socket()/bind()/listen()/accept() - основные сокет-функции
* send()/recv() - передача данных
* select() (опционально) для мониторинга нескольких сокетов

**3.5 Потокобезопасность**

Для обеспечения потокобезопасности используются:

* pthread\_mutex\_t для защиты:
  + Списка клиентов
  + Доступа к базе данных SQLite
  + Истории чата
* Атомарные операции при работе с сокетами

**3.6 База данных**

Для аутентификации используется SQLite с таблицей:

CREATE TABLE users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

username TEXT NOT NULL UNIQUE,

password TEXT NOT NULL

);

Функции работы с БД:

* authenticate\_user() - проверка логина/пароля
* register\_user() - регистрация новых пользователей
* init\_database() - инициализация БД

**3.7 Дополнительные структуры**

* Временные файлы: создаются в FILE\_DIR с уникальными именами
* Буферизация: используется для передачи больших файлов
* Форматирование сообщений: единый формат для клиент-серверного взаимодействия

Данная структура обеспечивает:

1. Многопоточную обработку клиентов
2. Безопасный доступ к общим ресурсам
3. Гибкое управление различными режимами работы
4. Масштабируемость для добавления новых функций

**4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

В данном разделе рассмотрены алгоритмы работы ключевых функций, реализующих основной цикл обработки клиентских соединений и функциональность TCP-сервера с поддержкой чата, выполнения команд и передачи файлов.

4.1 Разработка схем алгоритмов

Схема алгоритма основного цикла сервера, реализованного в функции main(), приведена в приложении Б.

Схема алгоритма обработки клиентской сессии, реализованного в функции handle\_client(), приведена в приложении В.

4.2 Разработка алгоритмов

В основном цикле сервера main() происходит следующая последовательность действий:

1. Ожидание нового подключения:
   * Сервер блокируется на вызове accept(), ожидая входящего соединения
   * При поступлении соединения создается новый клиентский сокет
2. Создание клиентской сессии:
   * Выделяется память под структуру Client
   * Инициализируются поля сокета (новый дескриптор) и статуса аутентификации
   * Имя пользователя пока не заполнено
3. Запуск обработчика клиента:
   * Создается новый поток через pthread\_create()
   * Потоку передается указатель на структуру клиента
   * Поток отсоединяется (pthread\_detach) для автономной работы
4. Возврат к ожиданию:
   * Сервер немедленно возвращается к вызову accept()
   * Готов принимать следующее подключение, не дожидаясь завершения обработки предыдущего

Функция handle\_client(void\* arg) является ядром обработки клиентских соединений и реализует следующий алгоритм:

1. Инициализация сессии:
   * Получение структуры Client из параметра
   * Выделение буферов для обмена сообщениями
   * Установка таймаутов на сокет (опционально)
2. Процесс аутентификации:
   * Запрос учетных данных
   * Проверка через SQLite базу
   * Отказ при неудачной аутентификации
3. Выбор режима работы:
   * Отправка клиенту меню доступных режимов
   * Ожидание выбора (1-чат, 2-команды, 3-файлы)
   * Ветвление на соответствующий обработчик
4. Обработка режима чата:
   * Получение и рассылка сообщений
   * Ведение истории чата
   * Обработка команд (напр. /exit)
5. Обработка режима команд:
   * Безопасное выполнение shell-команд
   * Возврат результата клиенту
   * Валидация ввода
6. Обработка режима передачи файлов:
   * Поддержка команд upload/download
   * Проверка прав доступа к файлам
   * Бинарная передача с контролем целостности
7. Завершение сессии:
   * Удаление клиента из списка активных
   * Закрытие сокета
   * Освобождение ресурсов
   * Логирование завершения сессии