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 курсовой

работе на тему

РАЗРАБОТКА МНОГОПОТОЧНОГО TCP СЕРВЕРА

БГУИР КП 6-05-0611-05 220 ПЗ

Студент Д.А. Манухо

Руководитель Д.А. Жалейко

МИНСК 2025

Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет: ФКСиС. Кафедра: ЭВМ.

Специальность: 40 02 01 «Вычислительные машины, системы и сети».

Специализация: 400201-01 «Проектирование и применение локальных компьютерных сетей».

УТВЕРЖДАЮ

Заведующий кафедрой ЭВМ

\_\_\_\_\_\_\_\_\_ Б.В. Никульшин

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

ЗАДАНИЕ

по курсовому проекту студента

Манухо Даниила Алексеевича

1. Тема проекта: «Разработка многопоточного tcp cервера».
2. Срок сдачи студентом законченного проекта: 10 мая 2025 г.
3. Исходные данные к проекту: нет.
4. Содержание пояснительной записки (перечень подлежащих разработке вопросов):  
   Титульный лист.  
   Введение.  
   1. Обзор литературы.  
   2. Системное проектирование.   
   3.Функциональное проектирование.   
   4. Руководство пользователя.   
   Заключение.  
   Список используемых источников.  
   Приложения
5. Перечень графического материала (с точным указанием обязательных чертежей):
   1. Схема алгоритма.
   2. Диаграмма последовательностей.
   3. Диаграмма классов.

КАЛЕНДАРНЫЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапов  курсового проекта | Объем  этапа,  % | Срок выполнения этапа | Примечания |
| Выбор темы курсового проекта | 5 | 17.02 – 01.03 |  |
| Начальный этап ПЗ | 30 | 01.03 – 01.04 |  |
| Основная часть кода | 50 | 01.04 – 01.05 |  |
| Оформление пояснительной записки и графического материала | 15 | 01.05 – 10.05 | с выполнением  чертежа |
| Защита курсового проекта |  | 28.05 – 10.06 |  |

Дата выдачи задания: 1 февраля 2025 г.

Руководитель Д. А. Жалейко

ЗАДАНИЕ ПРИНЯЛ К ИСПОЛНЕНИЮ \_\_\_\_\_\_\_

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 5

1 ОБЗОР ЛИТЕРАТУРЫ 6

1.1 Игра как проект 6

1.2 Анализ существующих аналогов 6

1.3 Анализ использованных средств разработки.................................................7

1.3 Постановка задачи 7

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 8

2.1 Модуль управления состояниями 8

2.1 Модуль управления сущностями 9

2.1 Модуль управления обновления игрового контекста 9

**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 запросов клиента**

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 логирования и мониторинга**

Для отладки и анализа работы сервера реализован простой логирующий компонент. Он:

* Записывает информацию о подключениях, ошибках и завершении потоков;
* Сохраняет временные метки и идентификаторы клиентов;
* Может быть расширен до уровня журналирования по уровням (INFO, ERROR, DEBUG).
* Данный модуль облегчает диагностику проблем и контроль за состоянием системы в процессе эксплуатации.