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

«МНОГОПОТОЧНАЯ ПРОГРАММА ОБМЕНА ФАЙЛАМИ»

БГУИР КР 1-40 02 01 101 ПЗ

Студент Е.Д. Елиневский

Руководитель Л.П. Поденок

Минск 2025

Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой
(подпись)
_____ 2025 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту Елиневскому Егору Дмитриевичу

1. Тема проекта: *Многопоточная программа обмена файлами*

2. Срок сдачи студентом законченного проекта: *15 мая 2025 г.*

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

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке):

Введение. 1. Обзор методов и алгоритмов решения поставленной задачи. 2. Обоснование выбранных методов и алгоритмов. 3. Описание программы для программиста. 4. Описание алгоритмов решения задачи. 5. Руководство пользователя. Заключение. Список использованных источников. Приложения.

5. Перечень графического материала (с точными обозначениями обязательных чертежей и графиков):

1. *Схема алгоритма работы функции*
2. *Скриншоты работы программы*
3. *Ведомость документов*

6. Консультант по проекту (с обозначением разделов проекта) *Поденок Л. П.*

7. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

раздел 1 к 01.03. – 15%;

раздел 2, 3 к 01.04. – 50%;

раздел 4, 5 к 01.05. – 80%;

оформление пояснительной записки и графического материала к 15.05.2025 – 100%

Защита курсового проекта с 29.05 по 09.06.

Руководитель Л.П. Поденок

Задание принял к исполнению Е.Д. Елиневский

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире информационные технологии играют ключевую роль в обеспечении эффективного обмена данными между пользователями и устройствами. Особенно важным является быстрое и надежное перемещение файлов между узлами в распределённых системах и одноранговых (P2P) сетях. Традиционные клиент-серверные решения зачастую оказываются недостаточно гибкими или подвержены узким местам производительности, в то время как одноранговые приложения позволяют создать более масштабируемую и отказоустойчивую архитектуру.

В данной курсовой работе рассматривается разработка многопоточной P2P-программы обмена файлами, реализованной на языке программирования C++ с использованием стандартных библиотек и средств POSIX. Основная цель проекта — создание удобного и функционального консольного приложения, позволяющего пользователю:

1) Добавлять одноранговых участников (пиров),
2) Просматривать локальные файлы, доступные для отправки,
3) Выбирать пира для передачи данных,
4) Отправлять файлы и директории на указанный узел,
5) Получать файлы и автоматически сохранять их в выделенную директорию.

Ключевая особенность реализованной программы заключается в использовании многопоточности: один поток выполняет функции сервера и постоянно ожидает входящих соединений, в то время как основной поток предоставляет интерактивный пользовательский интерфейс. Это позволяет обеспечивать параллельность обработки операций, повышать отзывчивость и стабильность программы, не блокируя пользовательский ввод во время приёма данных.

Дополнительно реализована передача как отдельных файлов, так и целых директорий с сохранением структуры, что делает приложение удобным в реальных сценариях использования, например, при передаче проектов, архивов или мультимедийных библиотек.

Данная работа имеет как учебное, так и практическое значение. В процессе реализации рассматриваются важные аспекты многопоточного программирования, работы с файловой системой, сетевыми сокетами и взаимодействия между потоками, что позволяет глубже понять архитектуру распределённых приложений и принципы проектирования надёжного P2P-софта.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Обзор аналогов

Для качественного выполнения работы следует рассмотреть существующие аналоги программ, реализующих передачу файлов в модели "равный — равному" (P2P). Ниже приведены три решения, доступные для использования в ОС Linux.

BitTorrent — один из самых известных протоколов и клиентов для обмена файлами в P2P-среде. Его особенностью является разбиение файла на небольшие части и параллельная передача этих частей от множества пиров, что ускоряет процесс загрузки. BitTorrent применяется в крупных проектах для дистрибуции данных, однако требует наличия торрент-трекеров или DHT-сети для координации. Протокол активно используется, но слабо подходит для прямого обмена файлами между известными пользователями без промежуточных сервисов.

Syncthing — кроссплатформенное приложение с открытым исходным кодом, предназначенное для защищённой синхронизации файлов между устройствами. Оно реализует прямое P2P-соединение и использует TLS-шифрование для защиты данных. Syncthing автоматически определяет устройства в локальной сети, не требует серверов или ручного переноса файлов, но может быть избыточным для простого обмена файлами по запросу.

Croco File Exchange — менее известный, но простой P2P-решение с графическим интерфейсом, написанное на C++ и Qt. Поддерживает прямой обмен файлами между пользователями в локальной сети. Однако проект не получил широкого распространения, не поддерживает пересылку каталогов и редко обновляется, что ограничивает его применимость.

1.2 Постановка задачи

Программа должна быть разработана для операционных систем семейства Linux с использованием языка программирования C++ (с соблюдением стандарта C++17 или выше). В реализации предполагается использование системных вызовов POSIX, многопоточности (на основе std::thread или pthread), работы с файловой системой (<filesystem>) и сетевых сокетов (через sys/socket.h, netinet/in.h, arpa/inet.h и др.).

Основной целью проекта является создание P2P-программы для передачи файлов и папок между двумя пользователями по IP-соединению без привлечения центрального сервера. Особенностью утилиты является реализация собственного текстового протокола обмена управляющими сообщениями, что позволяет гибко управлять сеансом передачи и расширять функциональность без зависимости от сторонних решений.

Ключевые функциональные возможности программы:

1) Отправка и приём как отдельных файлов, так и целых папок с сохранением их структуры;

2) Ввод `IP`-адреса и порта подключения при запуске программы;

3) Отображение списка доступных файлов/папок для передачи;

4) Поддержка собственного текстового протокола для управления сеансом передачи;

5) Многопоточная обработка входящих и исходящих соединений;

Программа должна иметь простой и понятный текстовый интерфейс, удобный для использования в терминале. Также важно предусмотреть обработку ошибок (например, отсутствие файла, отказ в соединении, разрыв связи), чтобы обеспечить стабильную работу приложения в реальных условиях.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

После определения функциональных требований к P2P-программе обмена файлами, проект был логически разделён на функциональные модули. Такая модульная архитектура повышает масштабируемость и сопровождаемость системы, облегчает отладку и расширение функциональности. Ниже приводится описание ключевых компонентов системы.

2.1 Модуль обработки командной строки

Модуль отвечает за обработку команд, введённых пользователем в терминале. В нем будут определяться флаги команд, такие как добавление нового пира, выбор пира, просмотр файлов и отправка данных. Каждый введённый флаг будет вызывать соответствующий функциональный блок программы. При отсутствии введённых команд, программа будет предлагать пользователю ввести их заново. В данном случае, этот модуль также будет обрабатывать ввод порта для приема файлов и управление выбором пиров для отправки и получения данных.

2.2 Модуль многопоточной обработки клиентов

Этот модуль будет отвечать за многопоточную обработку клиентов, принимая подключения от пиров и распределяя задачи между потоками. Каждый клиент будет обслуживаться отдельным потоком, который будет обеспечивать отправку и получение данных между клиентом и сервером. Модуль будет использовать стандартную библиотеку потоков C++ для создания новых потоков и их управления. Это позволит параллельно обрабатывать несколько подключений, не блокируя основной поток программы.

2.3 Модуль передачи файлов

Модуль передачи файлов будет отвечать за отправку и получение данных между пирами. Он будет работать с файловой системой, определять, является ли передаваемый объект файлом или директорией, и поочередно передавать их по сети. В этом модуле также будет организована проверка наличия файлов или директорий перед отправкой и обработка ошибок в случае недоступности файлов. Модуль будет использовать сокеты для установления соединений и передачи данных.

2.4 Модуль логирования

Логирование является важной частью системы, поскольку оно будет отслеживать действия программы, такие как отправка и получение файлов,

успешность операций и возникающие ошибки. Модуль логирования будет записывать информацию о переданных и принятых файлах, размере переданных данных и любых возникающих ошибках. Для повышения производительности логирование можно будет осуществлять в отдельном потоке, чтобы не блокировать основной процесс передачи данных.

2.5 Модуль работы с пирами

Этот модуль будет отвечать за добавление и выбор пиров в списке. Пользователи смогут добавить нового пира с указанием `IP`-адреса и порта, а также выбрать активного пира для обмена данными. Этот модуль также будет отслеживать состояние пиров и поддерживать актуальный список доступных пиров для обмена данными.

2.6 Модуль работы с файловой системой

Модуль работы с файловой системой будет осуществлять чтение и запись файлов, включая создание директорий и обработку ошибок при доступе к файлам. В этом модуле также будет происходить рекурсивное перечисление файлов в каталоге и отправка их на другие пирами. Модуль будет использовать библиотеку `std::filesystem` для работы с файловыми путями, создания директорий и проверки существования файлов.

2.7 Модуль сервера

Этот модуль будет отвечать за создание и настройку серверного сокета, прослушивание входящих соединений и принятие запросов от клиентов. Сервер будет работать на указанном порте и обрабатывать несколько соединений с клиентами одновременно, создавая новые потоки для обработки каждого клиента.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Программа предоставляет следующие ключевые функции:

3.1 Обмен файлами между узлами

1) Отправка данных:
   - Возможность передачи как отдельных файлов, так и целых каталогов
   - Рекурсивная обработка вложенных папок
   - Индикация прогресса передачи для каждого файла
2) Прием данных:
   - Автоматическое создание необходимой структуры папок
   - Разделение приема файлов и директорий
   - Логирование принятых данных с указанием размера

3.2 Управление подключениями

1) Работа в режиме сервера:
   - Ожидание входящих подключений на указанном порту
   - Многопоточная обработка одновременных соединений
2) Работа в режиме клиента:
   - Установка исходящих соединений к указанным пирам
   - Автоматическое восстановление при разрыве соединения

3.3 Управление списком узлов

1) Хранение информации о пирах:
   - Поддержка списка доступных узлов (`IP` + порт)
   - Возможность выбора активного узла для отправки
2) Интерфейс управления:
   - Добавление новых узлов
   - Выбор активного узла из списка
   - Визуализация текущего выбора

3.4 Работа с файловой системой

1) Локальное хранилище:
   - "`shared`" - папка для файлов, доступных к отправке
   - "`received`" - папка для входящих файлов
2) Просмотр содержимого:
   - Рекурсивный обход директорий
   - Отображение(<тип элемента > <относительный путь> <размер>)

3.5 Пользовательский интерфейс

1) Интерактивное меню:
    - Добавление узлов (`add`)
    - Выбор активного узла (`select`)
    - Просмотр файлов (`list`)
    - Отправка данных (`send`)
    - Выход (`exit`)
2) Визуальное оформление:
    - Цветовое выделение различных типов информации
    - Четкое разделение на логические блоки
    - Индикация текущего состояния

3.6 Обработка ошибок

1) Типы обрабатываемых ошибок:
    - Проблемы с подключением
    - Ошибки работы с файловой системой
    - Некорректный пользовательский ввод
2) Механизмы обработки:
    - Цветовая индикация ошибок
    - Подсчет успешных/неудачных операций
    - Продолжение работы после ошибок

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

В данном разделе рассмотрены алгоритмы работы четырёх ключевых функций программы: функции серверного потока `server_thread`(), функции обработки клиента `handle_client`(), функции отправки данных `send_all`() и функции вывода списка файлов `list_files`().

4.1 Разработка схем алгоритмов

Схема алгоритма функции серверного потока `server_thread`(), которая создаёт сокет, переводит его в режим ожидания соединений и запускает обработчики клиентов, приведена в Приложении А.

Схема алгоритма функции добавления пира `add_peer()`, которая реализует интерактивное добавление нового пир-узла в программе, приведена в Приложении Б.

4.2 Разработка алгоритмов

Функция `send_all`()**:**
1) начало;
2) входные данные:
   - `ip` - `IP`-адрес получателя;
   - `port` - порт получателя;
   - `target` - имя файла/папки для отправки;
3) проверка существования целевого объекта в папке "`shared`";
4) если объект - файл:
   - добавление в список отправки;
5) если объект - папка:
   - рекурсивный обход содержимого;
   - добавление всех файлов в список отправки;
6) для каждого элемента списка:
   - установка `TCP`-соединения;
   - отправка типа данных ('`F`' - файл, '`D`' - папка);
   - отправка относительного пути;
   - для файлов - потоковая передача содержимого;
   - закрытие соединения;
7) вывод статистики по успешным/неудачным отправкам;
8) конец.

Функция `list_files`()**:**
1) начало;
2) входные данные:
   - `dir` - путь к сканируемой директории;

3) инициализация счетчиков файлов и папок;
4) рекурсивный обход директории;
5) для папки:
    - вывод синего индикатора;
    - вывод относительного пути;
    - инкремент счетчика папок;
6) для файла:
    - вывод индикатора;
    - вывод относительного пути и размера;
    - инкремент счетчика файлов;
7) вывод итоговой статистики;
8) конец.

5 РЕЗУЛЬТАТЫ РАБОТЫ

Функциональность программы была протестирована на одном компьютере, используя различные порты и три терминала: первый порт — 9000, второй — 9001, третий — 9002. Благодаря использованию TCP-сокетов разницы между локальным и сетевым тестированием не наблюдается.

5.1 Тестирование базового функционала

1) Передача одиночных файлов:
   - Успешно переданы:
   - Текстовые документы (10-100 КБ)
   - Изображения (1-5 МБ)
   - Видеофайлы (50-300 МБ)

Время передачи файла размером 244 МБ составило ~1 сек в локальной сети, результат передача представлен на рисунке 5.1.

2) Передача каталога:
   - Протестирована передача папки с тремя файлами

На рисунке 5.3 видно, что структура папок сохранилась полностью, пример сеанса передачи представлен на рисунке 5.2.

5.2 Тестирование интерфейса

1) Работа с меню:
   - Все команды (`add`, `select`, `list`, `send`) отрабатывают корректно
   - Цветовая индикация помогает быстро ориентироваться в статусе операций
2) Обработка ошибок:
   - Неверные IP-адреса
   - Занятые порты
   - Отсутствующие файлы
   - Сообщения об ошибках четко идентифицируют проблему

5.3 Сравнительный анализ

1) После передачи файлов и папок производилось сравнение:
   - Контрольные суммы файлов (md5)
   - Структура директорий
   - Права доступа
2) Во всех тестовых случаях:
   - Контрольные суммы совпали
   - Структура папок сохранилась
   - Время передачи соответствовало ожиданиям для данной сети

5.4 Выявленные ограничения

1) При передаче очень больших файлов (>2ГБ) рекомендуется:
   - Использовать проводное соединение
   - Увеличить размер буфера передачи

```
Введите команду: list
=== СОДЕРЖИМОЕ ПАПКИ shared ===
text.txt (0 bytes)
p2p (81720 bytes)
some
some/1.txt (7 bytes)
some/2.txt (7 bytes)
some/3.txt (7 bytes)
video.mp4 (256166922 bytes)

Итого: 1 папок, 6 файлов
=== ГЛАВНОЕ МЕНЮ ===
1. add    - Добавить нового пира
2. select - Выбрать пира
3. list   - Просмотр файлов
4. send   - Отправить файл/папку
5. exit   - Выход

Текущий пир: 0. 127.0.0.1:9000

Введите команду: send video.mp4
Введите имя файла/папки для отправки: === ОТПРАВКА НА 127.0.
0.1:9000 ===
✔ Файл video.mp4                    отправлен

Результат: 1 успешно, 0 с ошибками
```

Рисунок 5.1 – Отправка файла размером 244 мб.

```
Введите команду: list
=== СОДЕРЖИМОЕ ПАПКИ shared ===
text.txt (0 bytes)
p2p (81720 bytes)
some
some/1.txt (7 bytes)
some/2.txt (7 bytes)
some/3.txt (7 bytes)
video.mp4 (256166922 bytes)
```

```
Итого: 1 папок, 6 файлов
=== ГЛАВНОЕ МЕНЮ ===
1. add    - Добавить нового пира
2. select - Выбрать пира
3. list   - Просмотр файлов
4. send   - Отправить файл/папку
5. exit   - Выход

Текущий пир: 0. 127.0.0.1:9000

Введите команду: send
Введите имя файла/папки для отправки: some
=== ОТПРАВКА НА 127.0.0.1:9000 ===
✔ Файл some/1.txt                отправлен
✔ Файл some/2.txt                отправлен
✔ Файл some/3.txt                отправлен

Результат: 3 успешно, 0 с ошибками
```

Рисунок 5.2 – Отправка каталога с тремя файлами.

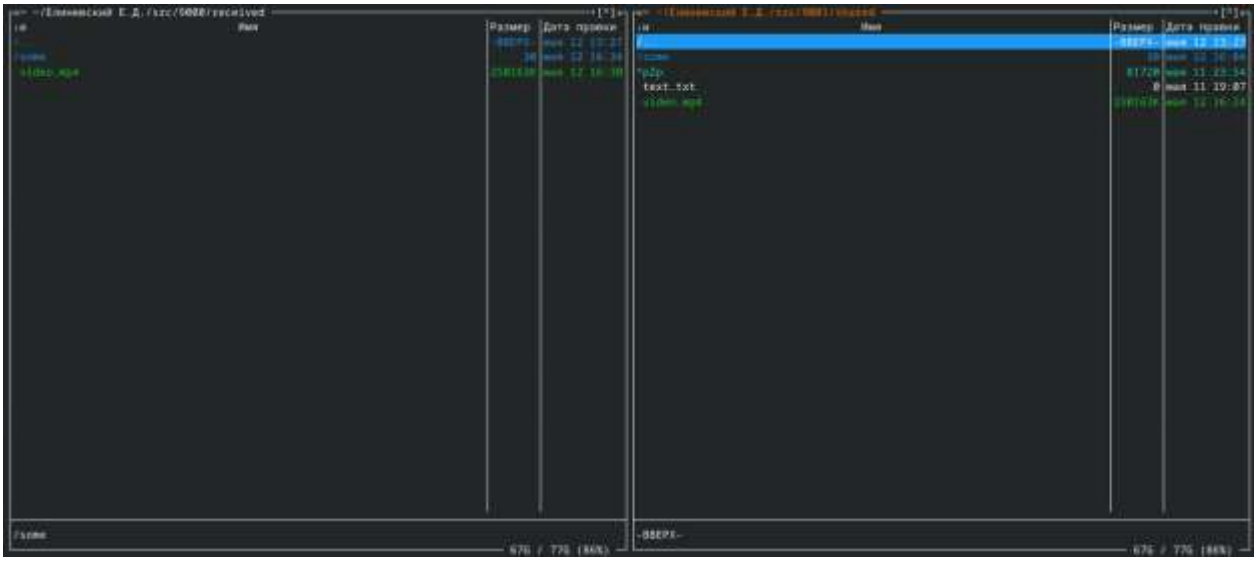


Рисунок 5.3 – Переданные файлы с порта 9001 на 9000.

ЗАКЛЮЧЕНИЕ

В ходе разработки программы для P2P-файлообмена был создан полнофункциональный сетевой инструмент, сочетающий возможности клиента и сервера в едином решении. Программа реализует устойчивый механизм передачи файлов и каталогов между узлами сети с поддержкой рекурсивной обработки вложенных структур.

Особое внимание было уделено созданию интуитивно понятного интерфейса с цветовым выделением различных типов операций, что значительно улучшает пользовательский опыт. Архитектура приложения построена на принципах многопоточности, где серверный компонент работает в фоновом режиме, обеспечивая бесперебойный прием входящих соединений без блокировки основного интерфейса.

Техническая реализация демонстрирует эффективное использование современных возможностей C++ (таких как filesystem API и потоки) в сочетании с традиционными сетевыми механизмами на основе BSD-сокетов. Программа корректно обрабатывает различные сценарии передачи данных, включая обработку ошибок подключения и контроль целостности передаваемой информации.

Разработанное решение представляет практическую ценность для обмена файлами в локальных сетях и служит хорошей основой для дальнейшего расширения функциональности. Перспективными направлениями развития могут стать реализация шифрования передаваемых данных, добавление механизма проверки контрольных сумм, создание графического интерфейса и внедрение технологии автоматического обнаружения соседних узлов в сети.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] C++ Programming Language / B. Stroustrup. – 4th ed. – Addison-Wesley, 2013. – 1368 p.[2] Boost C++ Libraries. – Электрон. ресурс. – Режим доступа: https://www.boost.org/
[3] Операционная система UNIX / А.М. Робачевский, А.А. Немнюгин, О.Л. Стесик. – 2-е изд. – СПб.: БХВ-Петербург, 2010. – 656 с.
[4] The Linux Programming Interface / M. Kerrisk. – San Francisco: No Starch Press, 2010. – 1552 p.
[5] Advanced Programming in the UNIX Environment / W. R. Stevens, S. A. Rago. – 3rd ed. – Addison-Wesley, 2013. – 1032 p.
[6] POSIX.1-2017 Standard / IEEE. – Электрон. ресурс. – Режим доступа: https://pubs.opengroup.org/onlinepubs/9699919799/

ПРИЛОЖЕНИЕ А

(обязательное)

Схема алгоритма функции `server_thread`()

ПРИЛОЖЕНИЕ Б

(обязательное)

Схема алгоритма функции `add_peer`()

ПРИЛОЖЕНИЕ В

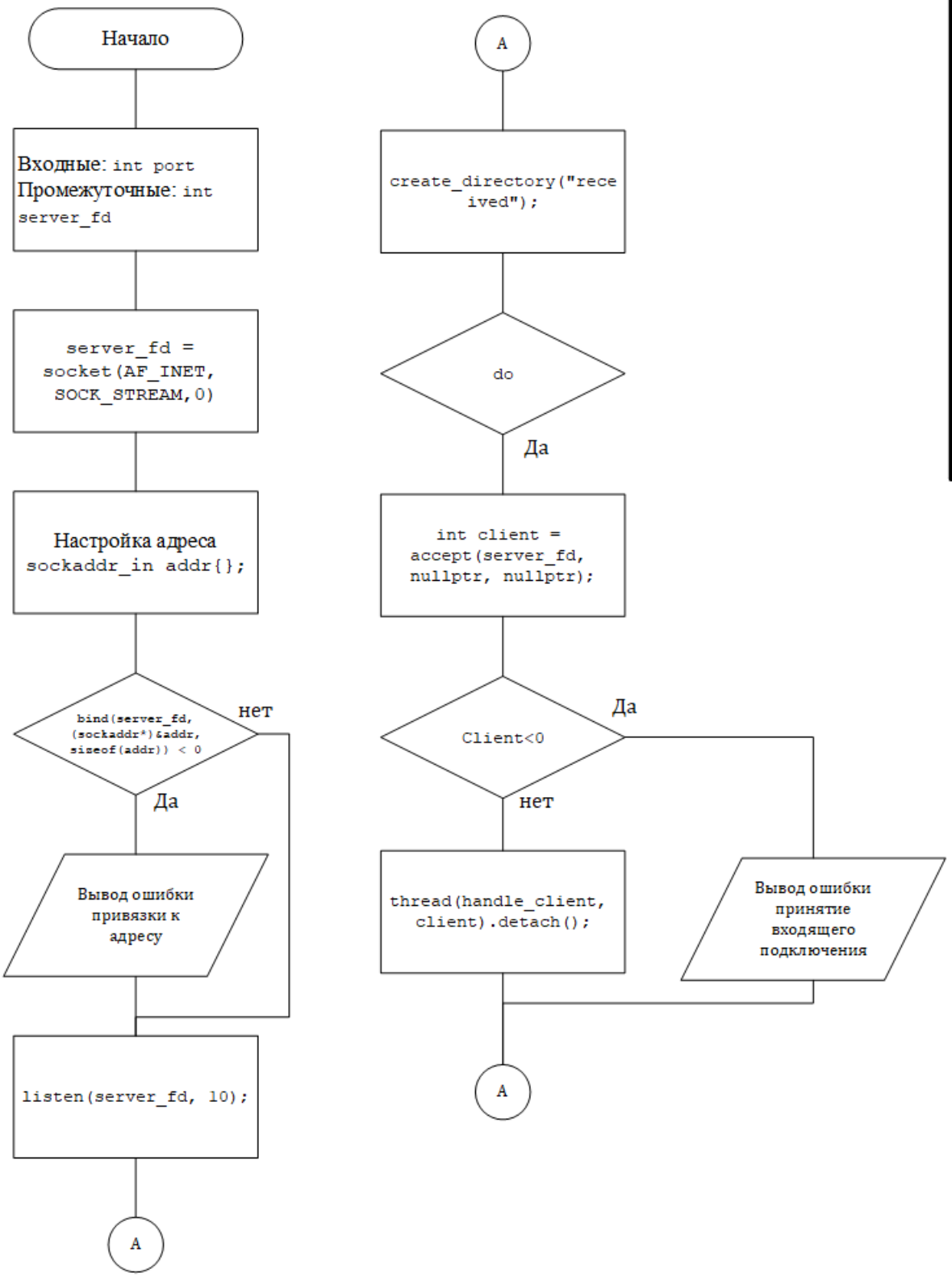
(обязательное)

Код программы

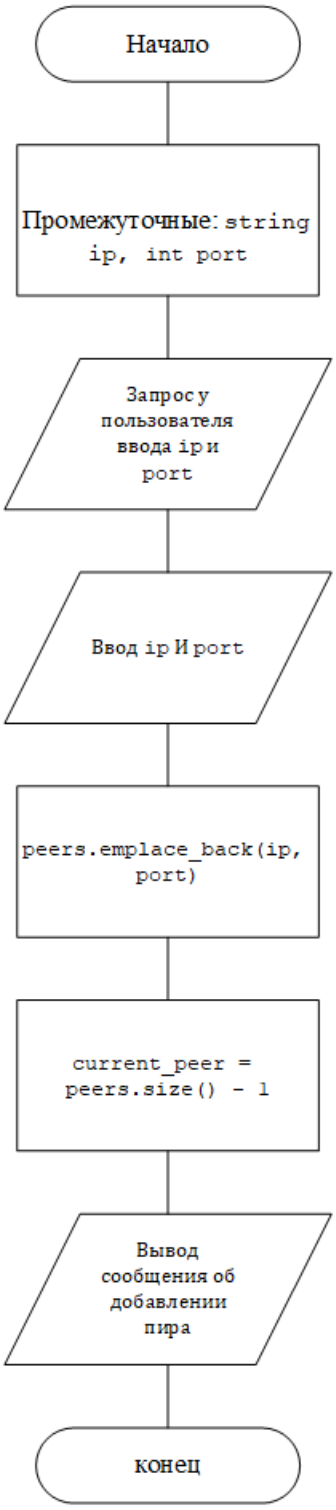
ПРИЛОЖЕНИЕ Г

(обязательное)

Ведомость документов



					ГУИР.6-05-0611-05.108 ПД1	
Изм	Лист	№ докум.	Подп.	Дата	Схема функции server_thread()	Лит. У / Масса / Масштаб
Разраб.		Елиневский				Лист / Листов 1
Пров.		Поденок				КИ (ВМСиС), гр.350501



Изм	Лист	№ докум.	Подп.	Дата
Разраб.		Елиневский		
Пров.		Поденок		

ГУИР.6-05-0611-05.108 ПД2

Схема функции `add_peer()`

Лит.	Масса	Масштаб
У		

Лист | Листов 1

КИ (ВМСиС), гр.350501

```
#include <iostream>
#include <filesystem>
#include <thread>
#include <fstream>
#include <vector>
#include <string>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <cstring>
#include <iomanip>

namespace fs = std::filesystem;
const int BUFFER_SIZE = 4096;

// ANSI цветовые коды
#define COLOR_RESET   "\033[0m"
#define COLOR_RED     "\033[31m"
#define COLOR_GREEN   "\033[32m"
#define COLOR_YELLOW  "\033[33m"
#define COLOR_BLUE    "\033[34m"
#define COLOR_MAGENTA "\033[35m"
#define COLOR_CYAN    "\033[36m"
#define COLOR_WHITE   "\033[37m"
#define COLOR_GRAY    "\033[90m"

int local_port;
std::vector<std::pair<std::string, int>> peers;
int current_peer = -1;

// ==== Вывод заголовка ====
void print_header(const std::string& title) {
    std::cout << COLOR_CYAN << "=== " << title << " ===" << COLOR_RE-
SET << "\n";
}

// ==== Просмотр файлов ====
void list_files(const std::string& dir) {
    print_header("СОДЕРЖИМОЕ ПАПКИ " + dir);

    size_t total_files = 0;
    size_t total_dirs = 0;

    for (const auto& entry : fs::recursive_directory_iterator(dir)) {
        if (fs::is_directory(entry)) {
            std::cout << COLOR_BLUE << " 🗀 " << COLOR_RESET
            << fs::relative(entry.path(), dir).string() << "\n";
            total_dirs++;
        } else {
            std::cout << " 🗎 " << fs::relative(entry.path(),
dir).string()
            << COLOR_GRAY << " (" << fs::file_size(entry) << " bytes)"
<< COLOR_RESET << "\n";
            total_files++;
        }
    }
```

```
    std::cout << COLOR_GREEN << "\nИтого: " << total_dirs << " папок,
"
    << total_files << " файлов" << COLOR_RESET << "\n";
}

// ==== Отправка файла или папки ====
void send_all(const std::string& ip, int port, const std::string& tar-
get) {
    std::string full_path = "shared/" + target;
    if (!fs::exists(full_path)) {
        std::cerr << COLOR_RED << "Ошибка: Файл или папка не найдены:
" << full_path << COLOR_RESET << "\n";
        return;
    }

    std::vector<fs::path> paths;
    if (fs::is_regular_file(full_path)) {
        paths.push_back(fs::path(target));
    } else {
        for (const auto& p : fs::recursive_directory_itera-
tor(full_path)) {
            paths.push_back(fs::relative(p.path(), "shared"));
        }
    }

    print_header("ОТПРАВКА НА " + ip + ":" + std::to_string(port));

    size_t success_count = 0;
    size_t fail_count = 0;

    for (const auto& rel_path : paths) {
        std::string full = "shared/" + rel_path.string();
        bool is_dir = fs::is_directory(full);

        int sock = socket(AF_INET, SOCK_STREAM, 0);
        if (sock < 0) {
            std::cerr << COLOR_RED << " ✗ Ошибка сокета для " <<
rel_path.string() << COLOR_RESET << "\n";
            fail_count++;
            continue;
        }

        sockaddr_in serv_addr{};
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_port = htons(port);
        inet_pton(AF_INET, ip.c_str(), &serv_addr.sin_addr);

        if (connect(sock, (sockaddr*)&serv_addr, sizeof(serv_addr)) <
0) {
            std::cerr << COLOR_RED << " ✗ Ошибка подключения для " <<
rel_path.string() << COLOR_RESET << "\n";
            close(sock);
            fail_count++;
            continue;
        }
```

```
        char type = is_dir ? 'D' : 'F';
        send(sock, &type, 1, 0);
        send(sock, rel_path.string().c_str(),
rel_path.string().size(), 0);
        send(sock, "\n", 1, 0);

        if (!is_dir) {
            std::ifstream in(full, std::ios::binary);
            char buffer[BUFFER_SIZE];
            while (in.read(buffer, sizeof(buffer)) || in.gcount()) {
                send(sock, buffer, in.gcount(), 0);
            }
        }

        close(sock);
        std::cout << COLOR_GREEN << " ✔ " << (is_dir ? "Папка " :
"Файл  ")
        << std::left << std::setw(40) << rel_path.string()
        << COLOR_RESET << " отправлен\n";
        success_count++;
    }

    std::cout << COLOR_YELLOW << "\nРезультат: " << success_count << "
успешно, "
    << fail_count << " с ошибками" << COLOR_RESET << "\n";
}

// ==== Обработка клиента ====
void handle_client(int client) {
    char type;
    if (recv(client, &type, 1, 0) <= 0) { close(client); return; }

    std::string rel_path;
    char ch;
    while (recv(client, &ch, 1, 0) > 0 && ch != '\n') {
        rel_path += ch;
    }

    std::string out_path = "received/" + rel_path;

    if (type == 'D') {
        fs::create_directories(out_path);
        std::cout << COLOR_BLUE << "[Сервер]  Принята папка: " <<
rel_path << COLOR_RESET << "\n";
    } else if (type == 'F') {
        fs::create_directories(fs::path(out_path).parent_path());
        std::ofstream out(out_path, std::ios::binary);
        char buffer[BUFFER_SIZE];
        ssize_t n;
        while ((n = recv(client, buffer, sizeof(buffer), 0)) > 0) {
            out.write(buffer, n);
        }
        std::cout << COLOR_GREEN << "[Сервер]  Принят файл: " <<
rel_path
```

```
        << COLOR_GRAY << " (" << fs::file_size(out_path) << " bytes)"
<< COLOR_RESET << "\n";
    }

    close(client);
}

// ==== Сервер ====
void server_thread(int port) {
    int server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd < 0) { perror("socket"); exit(EXIT_FAILURE); }

    sockaddr_in addr{};
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = INADDR_ANY;
    addr.sin_port = htons(port);

    if (bind(server_fd, (sockaddr*)&addr, sizeof(addr)) < 0) {
        perror("bind"); exit(EXIT_FAILURE);
    }

    listen(server_fd, 10);
    std::cout << COLOR_GREEN << "\n[Сервер] Запущен и слушает порт "
<< port << COLOR_RESET << "\n";

    fs::create_directory("received");

    while (true) {
        int client = accept(server_fd, nullptr, nullptr);
        if (client < 0) { perror("accept"); continue; }

        std::thread(handle_client, client).detach();
    }
}

// ==== Добавить нового пира ====
void add_peer() {
    print_header("ДОБАВЛЕНИЕ НОВОГО ПИРА");

    std::string ip;
    int port;
    std::cout << "IP адрес узла: ";
    std::cin >> ip;
    std::cout << "Порт узла: ";
    std::cin >> port;

    peers.emplace_back(ip, port);
    current_peer = peers.size() - 1;

    std::cout << COLOR_GREEN << "\nПир успешно добавлен!\n"
    << "Индекс: " << current_peer << "\n"
    << "Адрес: " << ip << ":" << port << COLOR_RESET << "\n";
}

// ==== Выбрать пира ====
void select_peer() {
```

```
    if (peers.empty()) {
        std::cout << COLOR_YELLOW << "Список пиров пуст. Используйте
команду 'add' для добавления." << COLOR_RESET << "\n";
        return;
    }

    print_header("ВЫБОР ПИРА");
    std::cout << "Текущий выбранный пир: ";
    if (current_peer == -1) {
        std::cout << COLOR_RED << "не выбран" << COLOR_RESET <<
"\n\n";
    } else {
        std::cout << COLOR_GREEN << current_peer << ". " << peers[cur-
rent_peer].first
        << ":" << peers[current_peer].second << COLOR_RESET << "\n\n";
    }

    std::cout << "Доступные пиры:\n";
    for (size_t i = 0; i < peers.size(); ++i) {
        std::cout << " " << (i == current_peer ? COLOR_GREEN ">" : "
") << COLOR_RESET
        << i << ". " << peers[i].first << ":" << peers[i].second <<
"\n";
    }

    std::cout << "\nВведите индекс пира (" << COLOR_YELLOW << "или -1
чтобы сбросить выбор" << COLOR_RESET << "): ";
    std::cin >> current_peer;

    if (current_peer < -1 || current_peer >= (int)peers.size()) {
        std::cout << COLOR_RED << "Неверный индекс!" << COLOR_RESET <<
"\n";
        current_peer = -1;
    } else if (current_peer == -1) {
        std::cout << COLOR_YELLOW << "Выбор пира сброшен" << COLOR_RE-
SET << "\n";
    } else {
        std::cout << COLOR_GREEN << "Выбран пир " << current_peer <<
": "
        << peers[current_peer].first << ":" << peers[cur-
rent_peer].second << COLOR_RESET << "\n";
    }
}

// ==== Главное меню ====
void print_menu() {
    print_header("ГЛАВНОЕ МЕНЮ");
    std::cout << " 1. " << COLOR_CYAN << "add" << COLOR_RESET << "
- Добавить нового пира\n";
    std::cout << " 2. " << COLOR_CYAN << "select" << COLOR_RESET << "
- Выбрать пира\n";
    std::cout << " 3. " << COLOR_CYAN << "list" << COLOR_RESET << "
- Просмотр файлов\n";
    std::cout << " 4. " << COLOR_CYAN << "send" << COLOR_RESET << "
- Отправить файл/папку\n";
```

```
    std::cout << " 5. " << COLOR_CYAN << "exit" << COLOR_RESET << "
- Выход\n";

    std::cout << "\nТекущий пир: ";
    if (current_peer == -1) {
        std::cout << COLOR_RED << "не выбран" << COLOR_RESET;
    } else {
        std::cout << COLOR_GREEN << current_peer << ". " << peers[cur-
rent_peer].first
        << ":" << peers[current_peer].second << COLOR_RESET;
    }
    std::cout << "\n";
}

// ==== Главная функция ====
int main() {
    std::cout << COLOR_CYAN << "\n=== P2P Файлообменник ===" <<
COLOR_RESET << "\n";
    std::cout << "Введите порт для приема файлов: ";
    std::cin >> local_port;

    std::thread(server_thread, local_port).detach();
    fs::create_directory("shared");

    while (true) {
        print_menu();

        std::string cmd;
        std::cout << "\nВведите команду: ";
        std::cin >> cmd;

        if (cmd == "exit"||cmd=="5") break;
        else if (cmd == "add"||cmd=="1") add_peer();
        else if (cmd == "select"||cmd=="2") select_peer();
        else if (cmd == "list"||cmd=="3") list_files("shared");
        else if (cmd == "send"||cmd=="4") {
            if (current_peer == -1) {
                std::cout << COLOR_RED << "Ошибка: не выбран пир!" <<
COLOR_RESET << "\n";
                continue;
            }
            std::string name;
            std::cout << "Введите имя файла/папки для отправки: ";
            std::cin >> name;
            send_all(peers[current_peer].first, peers[cur-
rent_peer].second, name);
        }
        else {
            std::cout << COLOR_RED << "Неизвестная команда. Попробуйте
снова." << COLOR_RESET << "\n";
        }
    }
    std::cout << COLOR_CYAN << "\nЗавершение работы..." << COLOR_RESET
<< "\n";
    return 0;
}
```

Обозначение	Наименование	Примечание
	Графические документы	
ГУИР.6-05-0611-05.108 ПД1	Схема алгоритма функции	А4
	server_thread. ПД1.	
ГУИР.6-05-0611-05.108 ПД2	Схема алгоритма функции add_peer. ПД2.	А4
	Текстовые документы	
БГУИР КП 6-05-0611-05 108 ПЗ	Пояснительная записка	22 с.

Изм.	Лист	№ докум.	Подп.	Дата	ГУИР.6-05-0611-05.108 Д1		
Разраб.		*Елиневский*			Ведомость документов	Лит	Лист	Листов
Пров.		*Поденок*				У		1
						КИ (ВМСиС), гр. 350501		