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

УТИЛИТА ДЛЯ ФОРМАТИРОВАНИЯ И ПРОВЕРКИ SFS

БГУИР КП 1-40 02 01 307 ПЗ

Студент: гр. 350503
Ганецкий В. В.

Руководитель:
Протько М. А.

Минск 2025

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет: КСиС. Кафедра: ЭВМ.
Специальность: 6-05-0611-05 «Компьютерная инженерия».
Профилизация: «Вычислительные машины, системы и сети».

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Б. В. Никульшин
«____» _____ 2025 г.

ЗАДАНИЕ
по курсовому проекту студента
Ганецкого Владимира Витальевича

1 Тема работы: «Утилита для форматирования и проверки SFS»

2 Срок сдачи студентом законченной работы: 31 марта 2025 г

3 Исходные данные к работе:
Операционная система: Linux; Язык программирования: C; Библиотека: ncurses; Цель: спроектировать программу для работы с файловой системой; Задачи: 1 Разработать минимально работоспособный аналог файловой системы. 2 Разработать функциональность для проверки и форматирования файловой системы. 3 Наладить управление серверным потоком для взаимодействия через сокеты. 4 Интегрировать шифрование для файлов.

4 Содержание пояснительной записки (перечень подлежащих разработке вопросов):
Введение. 1. Обзор литературы. 2. Системное проектирование. 3. Функциональное проектирование 4. Разработка программных модулей. 5. Программа и методика испытаний. 6. Руководство пользователя. Заключение. Список использованных источников.

5 Перечень графического материала (с точным обозначением обязательных чертежей и графиков)
5.1 Схема алгоритма метода `create_file`
5.2 Схема алгоритма метода `aes_cipher`
5.3 Схема алгоритма метода `check_incoming_requests`
5.4 Схема структурная

КАЛЕНДАРНЫЙ ПЛАН

Наименование этапов курсовой работы	Объем этапа, %	Срок выполнения этапа	Примечания
Подбор и изучение литературы	10	21.02 – 28.02	
Системное проектирование	20	01.03 – 15.03	
Функциональное проектирование	30	16.03 – 31.03	
Разработка программных модулей	20	01.04 – 15.04	
Программа и методика испытаний	5	16.04 – 30.04	
Руководство пользователя	5	01.05 – 09.05	
Оформление пояснительной записки и графического материала	10	10.05 – 22.05	
Защита курсового проекта		23.05 – 06.06	

Дата выдачи задания: 21 февраля 2025 г.

Руководитель Протько М. А.

ЗАДАНИЕ ПРИНЯЛ К ИСПОЛНЕНИЮ Ганецкий В. В.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Сейчас на любом устройстве, будь то компьютер или телефон, есть файловая система. Файловая система представляет собой систему организации и хранения данных на устройствах, которая позволяет эффективно использовать пространство носителя информации. Она отвечает за то, как данные размещаются в заданных им местах, как они будут распределяться по объему памяти, сколько дополнительной информации они будут содержать для их однозначной классификации и структуризации. Это позволяет операционной системе выгодно использовать пространство памяти и уменьшать нагрузку на рабочие элементы. Кроме того, неоспоримую пользу файловая система приносит для пользователя, предоставляя ему интерфейс для взаимодействия с информацией. Благодаря визуализации всех объектов, хранимых в устройстве, а также оформлении их в виде каталогов, пользователь может с легкостью находить файлы и различать их между собой, не вдаваясь в подробности их предназначения. Это существенно сокращает время на поиск информации и улучшает пользовательский опыт. Также файловая система может предоставлять расширенный функционал для работы с файлами, такой как разграничение доступа или шифрование файлов. Наличие такого рода функций позволяет улучшить безопасность данных и персонализировать представление объектов файловой системы.

С учетом сложности файловых систем, для каждой из них требуется утилита, позволяющая проверять корректность размещенных данных, их целостность и изменять структуру носителя информации, то есть осуществлять форматирование. Форматирование позволяет быстро очищать носитель от ненужной информации, давая возможность заново заполнять чистое пространство новыми данными, а также помогает вернуть структуру в рабочее состояние, если по какой-либо причине в файловой системе произошел сбой и работоспособность была нарушена. Также некоторые утилиты в таком случае позволяют не только отформатировать носитель информации, но и восстановить утраченные или поврежденные данные. Тема данного курсового проекта посвящена разработке именно такой утилите, которая позволит расширить работу SFS (simple file system) и защитить ее от сбоев.

1 ОБЗОР ЛИТЕРАТУРЫ

Для разработки данной утилиты была выбрана библиотека ncurses [1], предназначенная для управления вводом-выводом на терминал. Она упрощает работу, позволяя использовать готовые функции для создания консольного интерфейса, что дает возможность уделить больше времени на реализацию возможностей утилиты файловой системы. Более того, данная библиотека предоставляет преимущество в виде написания переносимого кода, так как ncurses сама управляет взаимодействием с аппаратными различиями терминалов.

В качестве языка программирования для реализации программного обеспечения был выбран C. Такой выбор обусловлен распространенностью языка в сфере написания операционных систем как средства, которое позволяет писать высокопроизводительный код с удобным уровнем абстракции, позволяющий эффективно создавать требуемый функционал, и при этом не требующее значительного потребления системных ресурсов [2]. С точки зрения темы проекта, это позволяет написать утилиту, быстро работающую с файловой системой и обладающей широким функционалом для взаимодействия с ней.

1.1 Анализ существующих аналогов

Учитывая большой потенциал для разработки мощного программного обеспечения, следует изучить имеющиеся аналоги, составить представление об их возможностях, недостатках, преимуществах и сформировать план по структуре собственной утилиты.

1.1.1 TestDisk

TestDisk (см. рисунок 1.1.1.1) позволяет восстанавливать загрузочные разделы, удаленные разделы, фиксировать поврежденные таблицы разделов и восстанавливать данные, а также создавать копии файлов с удаленных/недоступных разделов. При запуске приложения предоставляется список разделов жесткого диска, с которыми можно работать. Выбор доступных действий, осуществляемых в разделах, включает: анализ для корректировки структуры (и последующее восстановление, в случае обнаружения проблемы); изменение дисковой геометрии; удаление всех данных в таблице разделов; восстановление загрузочного раздела; перечисление и копирование файлов; восстановление удаленных файлов; создание снапшота раздела.

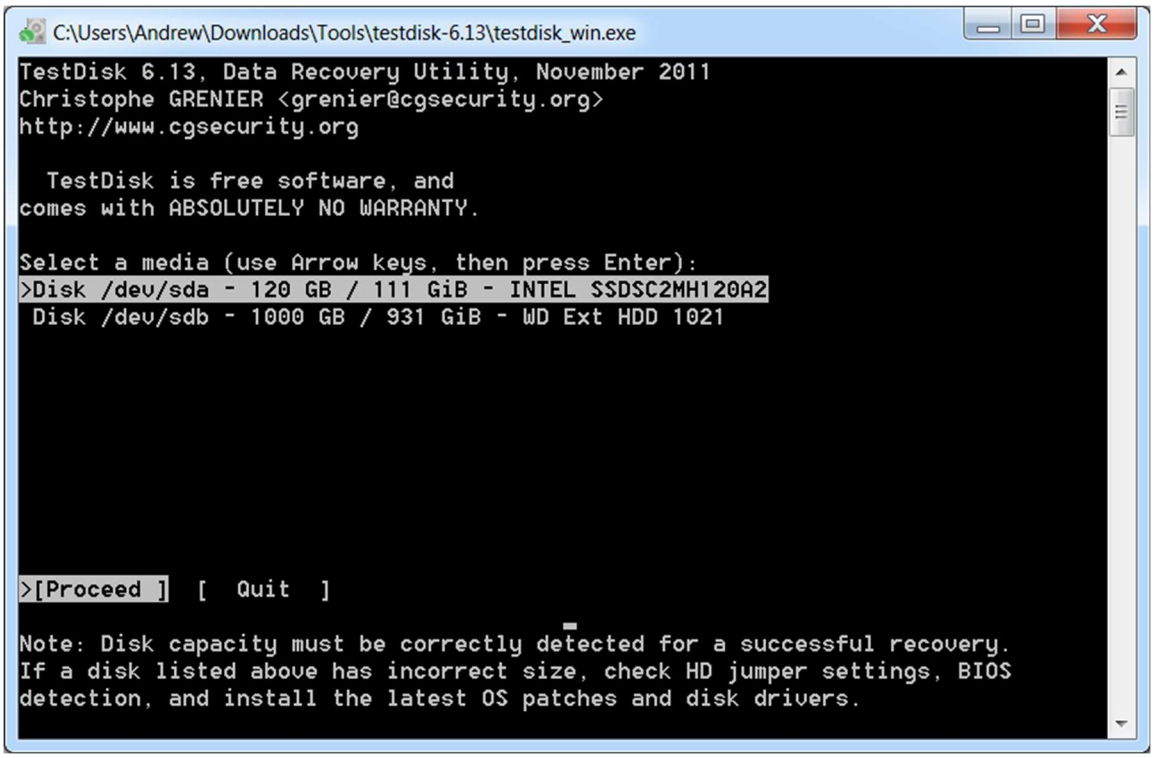

Рисунок 1.1.1.1 – скриншот программы TestDisk

1.1.2 EaseUS Partition Master

EaseUS Partition Master (см. рисунок 1.1.2.1) – инструмент для работы с разделами жесткого диска. Он позволяет создавать, перемещать, объединять, разделять, форматировать, изменяя их размер и расположение без потери данных. Также помогает восстанавливать удаленные или потерянные данные, проверять разделы, перемещать ОС на другой HDD/SSD и осуществлять другой функционал.

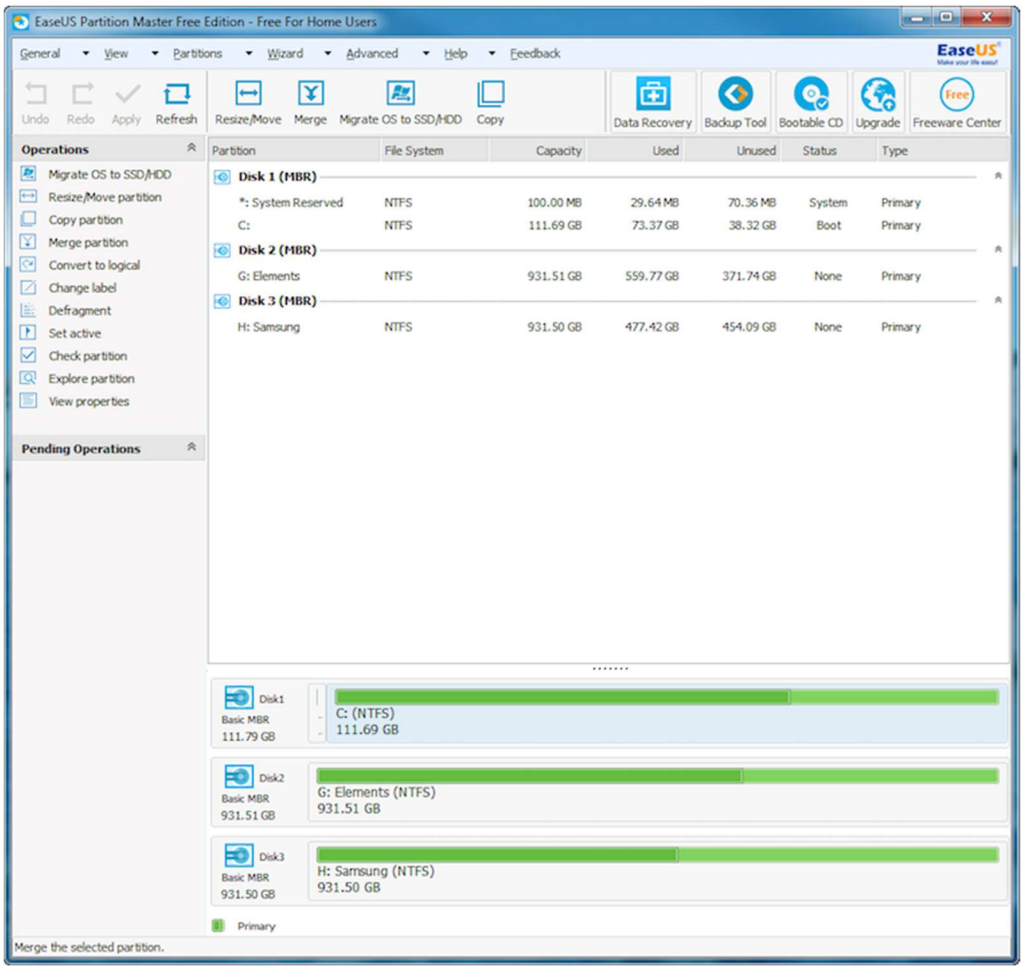

Рисунок 1.1.2.1 – скриншот программы EaseUS Partition Master

1.1.3 GParted

GParted (см. рисунок 1.1.3.1) – редактор дисковых разделов с открытым исходным кодом. Осуществляет эффективное и безопасное управление разделами (создание, удаление, изменение размера, перемещение, копирование, проверка) без потери данных. Данное приложение позволяет создавать таблицы разделов (MS-DOS или GPT), включать, отключать и изменять атрибуты, выравнивать разделы, восстанавливать данные с поврежденных разделов и выполнять другие важные функции.

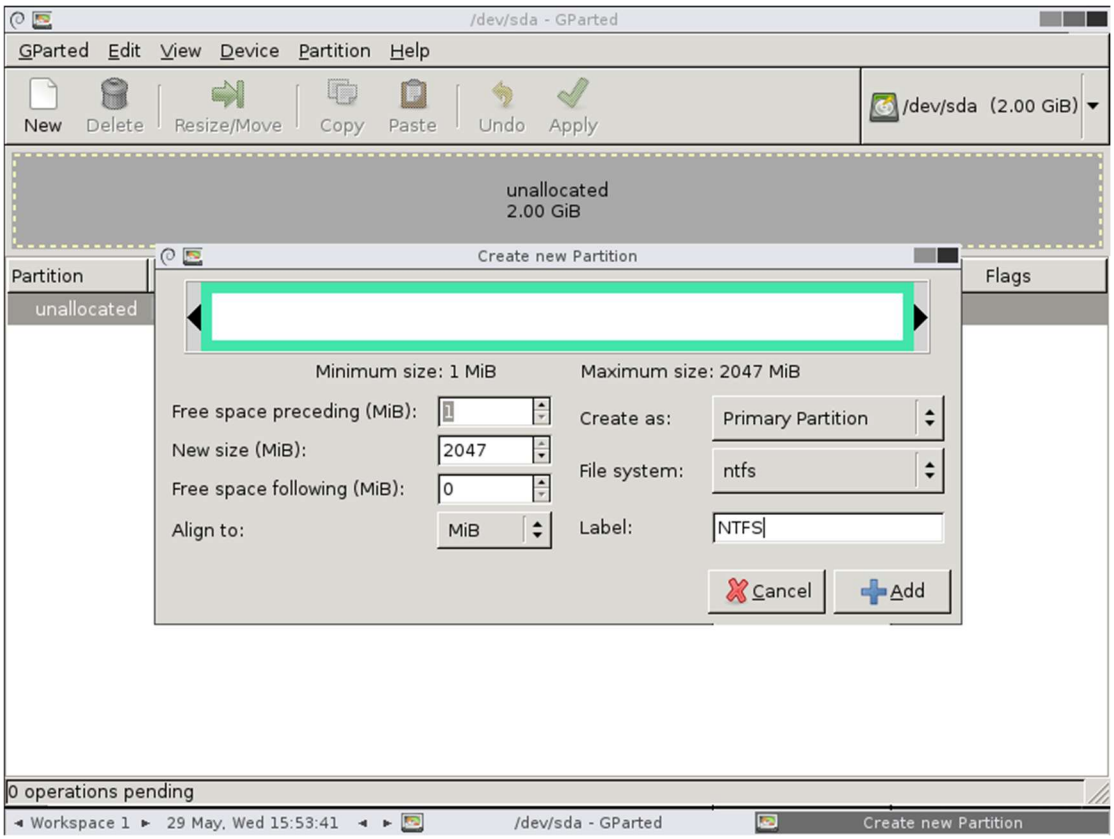

Рисунок 1.1.3.1 – скриншот программы GParted

1.2 Постановка задачи

По итогу рассмотрения аналогов можно сказать, что все они обладают разнообразным функционалом, а также большая часть из них имеет полноценный пользовательских интерфейс, что упрощает работу с данными программу. В рамках курсового проекта и выбранной темы реализации подобных функций и внешнего вида будет исчерпывающим, поэтому для реализации программного обеспечения были установлены следующие опорные пункты:

– Программа должна иметь минимальный пользовательских интерфейс для обеспечения удобного взаимодействия с приложением

– Программа должна предоставлять следующие функции: форматирование носителя, проверка файловой системы, шифрование файлов, восстановление повреждений

– Программа должна иметь документацию по принципу работы файловой системы и самой утилиты

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

После определения требований к функционалу разрабатываемого приложения его следует разбить на функциональные блоки. Такой подход упростит понимание проекта, позволит устранить проблемы в архитектуре, обеспечит гибкость и масштабируемость программного продукта в будущем путем добавления новых блоков.

2.1 Модуль файловой системы SFS

Данный модуль содержит основные компоненты файловой системы. В их число входят такие структуры, как `superblock`, `inode`, `dirent`. Данные элементы заключают в себе основную информацию о файловой системе, расположении файлов, их структуре и метаданных.

2.2 Модуль утилиты файловой системы SFS

Данный модуль содержит подмодули и функционал, отвечающий за проверку файловой системы на наличие повреждений, ее форматирование, журналирование и других действий, требуемых для поддержки корректной работы SFS.

2.3 Модуль сетевой передачи файлов

Данный модуль содержит функции, отвечающие за инициализацию протокола передачи файлов, формировании пакета и дальнейшей транспортировки требуемых данных получателю.

2.4 Модуль шифрования файлов

Данный модуль содержит реализацию алгоритма шифрования, который используется для шифрования данных в файлах и их расшифровке. В модуль включены структуры, которые требуются для полноценной работы алгоритма шифрования, функции, использующиеся для работы с данными файла, и методы, инкапсулирующие в себе совокупность процессов по обработке информации.

2.5 Модуль интерфейса

Данный модуль содержит реализацию внешнего вида приложения, которое позволяет пользователю легче воспринимать функционал, визуально

выделять важные действия или оповещать пользователя в более удобной форме, нежели простым выводом в терминал. За внешний вид отвечает библиотека ncurses, которая предоставляет различные возможности по кастомизации терминала и изменении вида вывода, что делает пользовательский опыт более насыщенным и влияет на восприятие программы в положительную сторону. Данный модуль использует функции модуля файловой системы для передачи параметров, введенных пользователем, и для получения результата выполнения с его последующим выводом на экран.

Программа была разбита на вышеприведенные модули в соответствие с функциями, выполняемыми внутри каждого из них: взаимодействие с файловой системой; использование дополнительных возможностей, предоставляемых утилитой файловой системы; работа с протоколом Интернета; шифрование данных; пользовательский интерфейс. Модуль файловой системы SFS использует возможности других модулей, получая через них доступ к содержащемуся внутри функционалу и предоставляя файлы для обработки.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описывается функционирование и структура разрабатываемого приложения.

Описание структур программы

3.1 superblock

Данная структура содержит информацию, необходимую для монтирования и управления работой файловой системы.

Поля:

- `uint32_t magic` (идентификатор файловой системы)
- `uint32_t block_size` (размер блока в байтах)
- `uint32_t total_blocks` (общее количество блоков памяти)
- `uint32_t free_blocks` (количество блоков, не содержащих записанной информации)
- `uint32_t total_inode` (общее количество inode в памяти)
- `uint32_t bitmap_blocks[TOTAL_BLOCKS]` (массив, содержащий информацию, занят блок или нет)
- `uint32_t bitmap_inode[TOTAL_INODE]` (массив, содержащий информацию, занят inode или нет)

3.2 inode

Данная структура представляет собой индексный дескриптор, в котором хранится метаинформация о стандартных файлах, каталогах и других объектах файловой системы.

Поля:

- `uint32_t type` (тип объекта)
- `uint32_t size` (размер данных в байтах)
- `uint32_t blocks[MAX_BLOCK_COUNT]` (массив номеров блоков данных, в которых хранится информация объекта)
- `time_t create_time` (время создания объекта)

3.3 dirent

Данная структура представляет объект в каталоге и содержит информацию для ее идентификации.

Поля:

- `char name[MAX_NAME_LEN]` (название объекта)
- `uint32_t inode_num` (номер inode, в котором хранится информация об объекте)

3.4 path_components

Данная структура представляет собой путь в файловой системе, разделенный на свои составляющие, то есть на объекты, из которых состоит данный путь.

Поля:

– `char** components` (массив строк, хранящий объекты пути)

– `int count` (количество объектов в пути)

3.5 file_request

Данная структура используется при сетевой передаче файлов и хранит в себе имя файла, который требуется передать, а также IP и дескриптор сокета отправителя для идентификации канала передачи.

Поля:

– `char filename[MAX_NAME_LEN]` (имя файла)

– `char sender_ip[INET_ADDSTRLEN]` (IP отправителя файла)

– `int socket_fd` (дескриптор сокета, требуемый для установки соединения)

3.6 content_buffer

Данная структура требуется для хранения информации, предоставляемой к выводу, в виде отдельных строк. Так как размер окна может быть меньше, чем общее количество строк к выводу, хранение данных в данной структуре позволяет эффективно распоряжаться имеющимся пространством на экране и использовать прокрутку для вывода только видимой части данных с учетом того, что интерфейс по умолчанию не поддерживает полосу прокрутки, и данный функционал требует собственной реализации.

Поля:

– `char** lines` (массив строк, в отдельности хранящий данные для вывода в заранее подготовленном виде)

– `int line_count` (общее количество строк)

– `int top_line` (первая видимая строка)

3.7 tab

Данная структура представляет собой вкладку, которая содержит в себе интерфейс, связанный с определенными функциями, объединенными общими объектами взаимодействия или принципом.

Поля:

– `WINDOW* win` (область, в которой располагается весь интерфейс)

– `PANEL* panel` (обертка для объекта win, позволяющая управлять областью по отношению к другим областям)

– `const char* title` (название вкладки)

– `void (*draw_content)(void)` (функция, отвечающая за отрисовку интерфейса вкладки)

– `void (*handle_mouse)(MEVENT*)` (функция, отвечающая за обработку нажатий по кнопкам, располагающимся внутри данной вкладки)

3.8 button

Данная структура представляет собой элемент интерфейса кнопку, которая служит для активации указанной функции при нажатии по ней.

Поля:

– `int x` (номер колонки, в котором располагается начало кнопки)

– `int y` (номер строки, в которой располагается начало кнопки)

– `int width` (ширина кнопки)

– `int height` (высота кнопки)

– `const char* label` (текст, расположенный внутри кнопки)

– `void (*action)(void)` (функция, которая вызывается при нажатии по кнопке)

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Содержание программных модулей

В следующих подразделах дается описание модулей приложения с уточнением деталей связи структур, принадлежащих указанному модулю, со структурами других модулей. Это позволяет лучше рассмотреть взаимодействие приложения со своими составными частями и понять, как один модуль использует функционал другого.

4.1.1 Модуль файловой системы SFS

Данный модуль представлен структурами, которые содержат информацию о файловой системе, ее объектах и метаданных, которые требуются для корректной работы функций взаимодействия с памятью. Основной структурой данного модуля является superblock, которая содержит основную информацию о файловой системе. Взаимодействие функций SFS в первую очередь происходит с объектами структуры inode. К данному модулю относятся такие функции, как создание файла, создание каталога, запись файла, чтение файла и удаление объектов. Он использует модуль шифрования файлов для использования алгоритма шифрования файлам с расширением .enc, что позволяет эффективно шифровать данные и безопасно передавать их по сети без опасения перехвата информации.

4.1.2 Модуль утилиты файловой системы SFS

Данный модуль является функциональным, предоставляя возможности для проверки и форматирования файловой системы. Он является неотъемлемой частью для корректной работы файловой системы, так как позволяет поддерживать бесперебойное взаимодействие всех частей памяти и не нарушать структуру блоков, Основными функциями данного модуля в роли проверки файловой системы являются check_inodes(), которая проверяет, все ли используемые inodes помечены как занятые в битовой карте; check_dirs(), которая проверяет, что каждый каталог содержит корректные записи dirent; check_duplicates(), которая ищет блоки, использующиеся несколькими inode одновременно; check_metadata(), проверяющая корректность метаданных в суперблоке; check_blocks(), удостоверяющаяся, что все используемые блоки помечены как занятые в битовой карте блоков. Для форматирования файловой системы используются следующие функции: clear_files_data(), которая очищает информацию во всех файлах; delete_all(), которая удаляет все объекты файловой системы, оставляя

только корневой каталог (без файлов); defragment(), отвечающая за перемещение информации между блоками таким образом, чтобы занятые блоки шли последовательно и между ними не было пустых блоков, что позволяет эффективнее использовать память и избегать фрагментации. Этот модуль связан с модулем самой файловой системы, так как он использует структуры, представляющие собой содержание SFS.

4.1.3 Модуль сетевой передачи файлов

Данный модуль отвечает за функционал, связанный с передачей файлов по сети с использованием сокетов. Он использует структуру FileRequest для хранения информации о файле, который требуется передать, и данные пользователя, который хочет этот файл отправить. Функция server_thread(), которая запускается в отдельном потоке в качестве фонового процесса для возможности постоянного прослушивания запросов без прерывания работы пользователя, производит создание сокета, его привязку к адресу пользователя и прослушиванию входящих запросов. При получении запроса на отправку файла данная функция добавляет новый запрос в очередь. Функция check_incoming_requests() проверяет очередь, в которую помещаются все запросы, пришедшие на сервер, выводит их, если очередь не пуста, и ждет согласие пользователя на скачивание имеющихся файлов. При согласии на адрес отправителя передается сигнал, что отправка файла разрешена, после чего начинается выделение памяти под новый файл и чтение данных из сокета. Если пользователь не согласился на получение файлов, то соответствующий сигнал также передается отправителю и все текущие запросы аннулируются. Для отправки файла используется функция send_file(), которая инициализирует сокет и соединяет его с адресом получателя, ищет указанный файл в файловой системе отправителя, после чего, если такой файл найден, отправляет имя файла получателю и ждет подтверждения на отправку в течение 10 секунд. Если за это время получатель не отправил сигнал или не согласился на получение файла, то соединение закрывается, иначе данные файла отправляются в файловую систему получателя. Этот модуль связан с модулем файловой системы, так как он использует функции по созданию файлов и чтению информации из них для корректного сбора данных и их записи на стороне получателя.

4.1.4 Модуль шифрования файлов

В этом модуле содержатся функции для шифрования и дешифрования информации в файлах, а само шифрование представлено алгоритмом AES (Advanced Encryption Standard). AES – это симметричный алгоритм блочного

шифрования. Для данного метода длина блока входных данных и состояния постоянна и равна 128 бит, а длина ключа может составлять 128, 192 или 256 бит. Модуль содержит в себе вспомогательные процедуры, которые используются для преобразования данных, и методы `aes_cipher(uint8_t* in, uint8_t* out, uint8_t* w)` и `aes_inv_cipher(uint8_t *in, uint8_t *out, uint8_t *w)`, которые отвечают за шифрование и дешифрование соответственно.

4.1.5 Модуль интерфейса

В этом модуле содержатся функции, которые отвечают за отрисовку визуальной составляющей программы и за отлавливание событий, вызываемых пользователем, таких как нажатие кнопки мыши или клавиатуры для дальнейшей обработки. Модуль состоит из разделенных элементов для удобного масштабирования и расширения функционала.

Базовый элемент интерфейса составляют вкладки, представленные структурой `tab`. Вкладки содержат в себе заголовок для производимых операций и кнопки, каждая из которых отвечает за свою функцию. Все кнопки содержат текст, однозначно определяющий их назначение. При нажатии по любой из кнопок появляется модальное окно, в котором происходит дальнейшее взаимодействие пользователя и программы. Такой подход позволяет не перерисовывать весь интерфейс конкретной вкладки, а использовать новое пространство, которое отображается поверх уже имеющихся визуальных элементов.

Главным элементов взаимодействия являются кнопки, представленные структурой button. Кнопки – это специальные области, выделенные определенным цветом, нажатие по которым вызывает определенную для них функцию. Для использования функций в паре с кнопками есть два механизма. Первый механизм заключается в передаче названия функции как параметра при создании кнопки. В таком случае, при нажатии по кнопке будет вызвана данная функция. Второй механизм использует обработчик, привязанный к определенной вкладке. Вначале определяется, в какой вкладке было произведено нажатие, после чего вызывается обработчик данной вкладки, в котором проверяются координаты нажатия и области каждой из кнопок. При обнаружении совпадения вызывается указанная в обработчике функция. Наличие двух подходов позволяет выставлять приоритетный вызов одной из двух функций, а также убирает необходимость в определении функции кнопки непосредственно во время инициализации кнопки, так как функцию можно определить в обработчике вкладки и оставить поле функции кнопки пустым.

Интерфейсом в основном используются два типа функций: функция-обработчик и функция-отрисовщик. Функции-обработчики, такие как

handle_files_mouse(MEVENT* mevent) или handle_network_mouse(), отвечают за определение, какая кнопка была нажата, и вызов соответствующей функции. Функции-отрисовщики, такие как draw_tools_tab() или draw_help_tab(), отрисовывают все визуальные элементы конкретной вкладки. Также для корректного отображения интерфейса используются функции register_button(int x, int y, int w, int h, const char *label, void (*action)(void)), которая инициализирует кнопку переданными в функцию параметрами и помещает ее в массив кнопок для предоставления доступа при будущих взаимодействиях, и switch_tab(int new_tab), которая принимает в качестве параметра номер вкладки и меняет текущую вкладку на требуемую, позволяя экономно использовать окно терминала для отображения функционала в виде логических групп и фокусируя внимание пользователя на необходимых ему действиях.

4.2 Разработка алгоритмов

В данном разделе рассматриваются методы, использующиеся в программе.

Метод server_thread ответственен за управление сокетом и входящих запросов. Он запускается в отдельном потоке и контролирует подключения, поступающие на IP адрес пользователя.

Метод delete_dir выполняет удаление каталога из файловой системы. Он получает путь к каталогу, разбирает его на компоненты, находит нужный каталог и проводит очистку его содержимого, осуществляя, помимо прочего, рекурсивное удаление других каталогов, содержащихся в исходном.

4.2.1 Метод server_thread

Шаг 1. Начало работы серверного потока
Шаг 2. Создаем TCP-сокет
Шаг 3. Инициализируем структуру sockaddr_in для адреса сервера
Шаг 4. Устанавливаем семейство адресов
Шаг 5. Задаем порт сервера
Шаг 6. Указываем, что сервер принимает подключения на все сетевые интерфейсы
Шаг 7. Привязываем сокет к адресу
Шаг 8. Если привязка не удалась, выводим ошибку и завершаем поток
Шаг 9. Переводим сокет в режим ожидания подключений
Шаг 10. Если вызов режима ожидания завершился ошибкой, выводим ошибку и завершаем поток

Шаг 11. Входим в бесконечный цикл для обработки клиентских подключений
Шаг 12. Объявляем структуру `sockaddr_in` для хранения адреса клиента
Шаг 13. Объявляем переменную `client_len` для хранения размера структуры адреса клиента
Шаг 14. Принимаем новое подключение
Шаг 15. Получаем имя файла от клиента и сохраняем в буфер
Шаг 16. Блокируем мьютекс для безопасного доступа к очереди запросов
Шаг 17. Проверяем, не переполнена ли очередь
Шаг 18. Если очередь не переполнена, копируем имя файла в массив `pending_requests`
Шаг 19. Преобразуем IP-адрес клиента в строку и сохраняем в `pending_requests`
Шаг 20. Сохраняем дескриптор клиентского сокета в `pending_requests`
Шаг 21. Увеличиваем счетчик запросов
Шаг 22. Если очередь переполнена, закрываем клиентский сокет
Шаг 23. Разблокируем мьютекс после работы с очередью запросов
Шаг 24. Возвращаемся к началу цикла для ожидания новых подключений

4.2.2 Метод delete_dir

Шаг 1. Разбираем переданный путь на компоненты
Шаг 2. Выводим имя удаляемой директории
Шаг 3. Если путь не содержит компонентов, возвращаем ошибку (-1)
Шаг 4. Находим `inode` родительской директории
Шаг 5. Если родительская директория не найдена, освобождаем память и возвращаем ошибку (-2)
Шаг 6. Читаем `inode` родительской директории в структуру `dir_inode`
Шаг 7. Читаем первый блок родительской директории в буфер
Шаг 8. Приводим буфер к типу `struct dirent*` для работы с записями директории
Шаг 9. Проходим по всем записям в родительской директории
Шаг 10. Если запись пуста и путь содержит более одного компонента, возвращаем ошибку (-3)
Шаг 11. Если имя записи совпадает с именем удаляемой директории, начинаем удаление
Шаг 12. Читаем `inode` удаляемой директории в

структуру `dir_inode_to_delete`
Шаг 13. Если удаляемая директория содержит блоки, читаем её первый блок в буфер
Шаг 14. Проходим по всем записям внутри удаляемой директории
Шаг 15. Если запись пуста, прерываем цикл
Шаг 16. Читаем `inode` текущего объекта (файла или подкаталога)
Шаг 17. Если объект – директория, рекурсивно удаляем её с помощью `delete_dir_inode()`
Шаг 18. Если объект – файл, удаляем его с помощью `delete_file_in_dir()`
Шаг 19. Очищаем `inode` объекта и записываем его на диск
Шаг 20. Очищаем `inode` удаляемой директории и записываем его на диск
Шаг 21. Сдвигаем оставшиеся записи в родительской директории, чтобы удалить пустой слот
Шаг 22. Если удаляемая директория содержала блоки, освобождаем её блок и очищаем его
Шаг 23. Записываем обновлённые записи родительской директории обратно на диск
Шаг 24. Освобождаем память, выделенную для компонентов пути

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Тестирование – это наблюдение за работой приложения в разных искусственно созданных ситуациях. Данные, полученные в ходе тестирования, важны при планировании последующей стратегии развития приложения. Это своего рода диагностика, которая влияет на дальнейшие действия и план разработки.

В данном разделе описаны все ключевые ошибки, которые могут возникнуть при использовании разработанного приложения. Было проведено тестирование и проработаны возможные решения данных моментов. После проверки на возникновение ошибок, приложение отработает корректно и не вызовет никаких нарушений в работе устройства.

5.1 Тестирование корректности пользовательского ввода

Каждое приложение, в котором есть ввод данных должно проверять входные данные на валидацию. В данном приложении проверяется корректность ввода имени и сообщения. Если введенная информация пустая, то на экране появится сообщение об ошибке и пользователю будет необходимо повторить ввод. Ниже приведены скриншоты ошибок, вызванные вводом пользователя (см рис. 5.1.1 – 5.1.6):

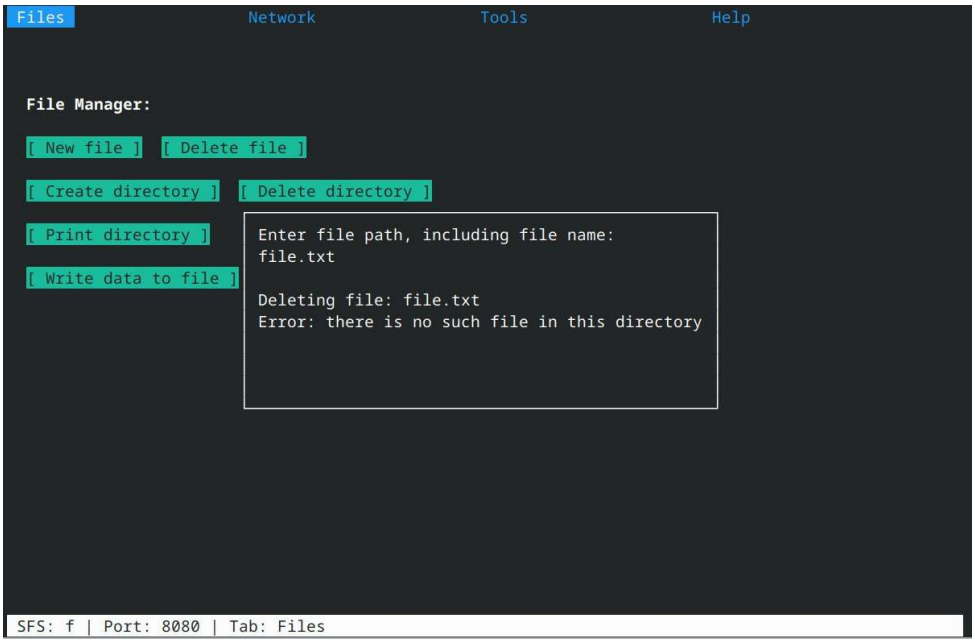

Рисунок 5.1.1 – отсутствие файла в указанной директории

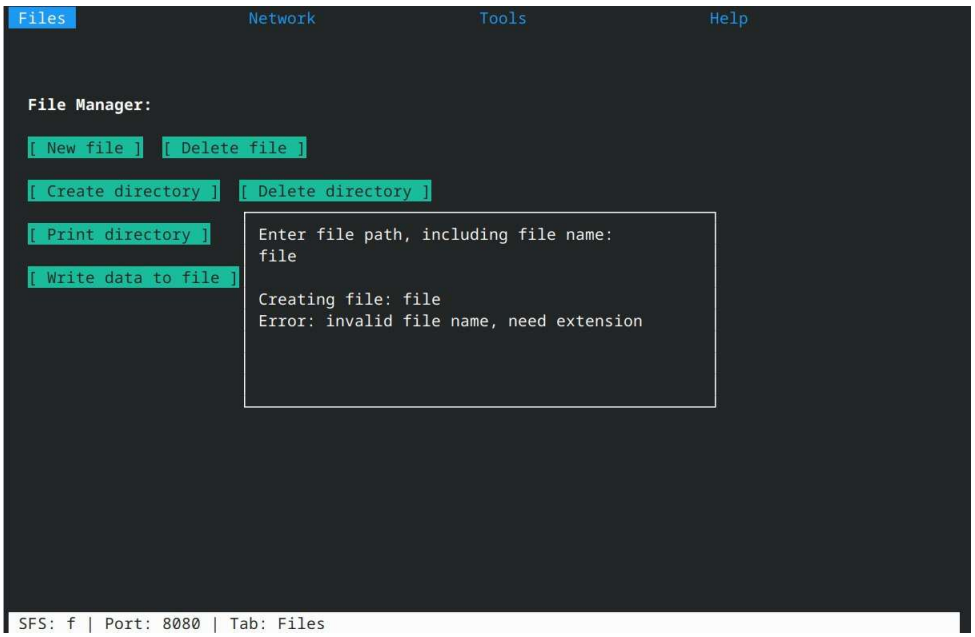

Рисунок 5.1.2 – некорректное имя файла

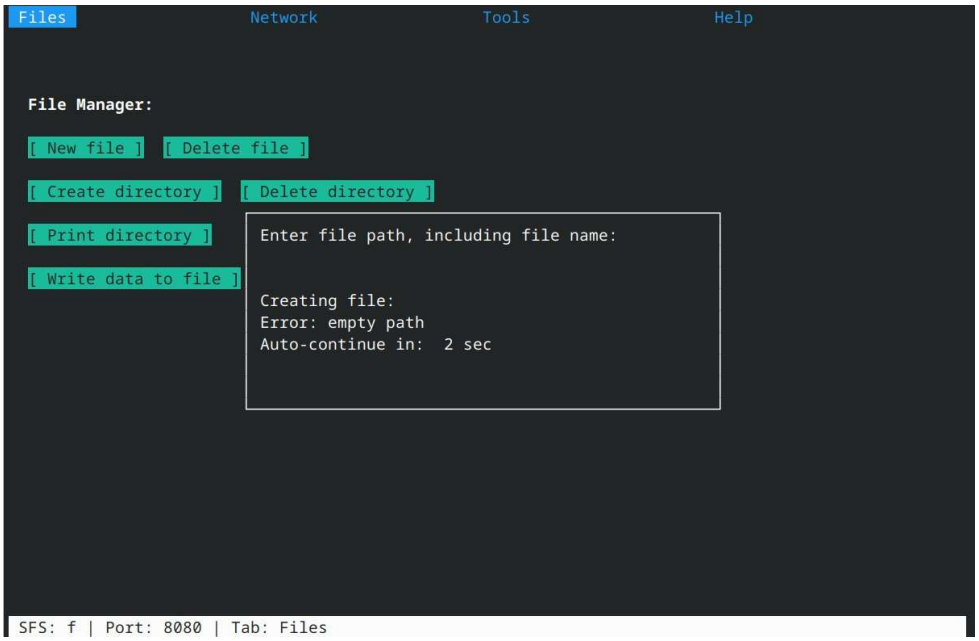

Рисунок 5.1.3 – пустой путь

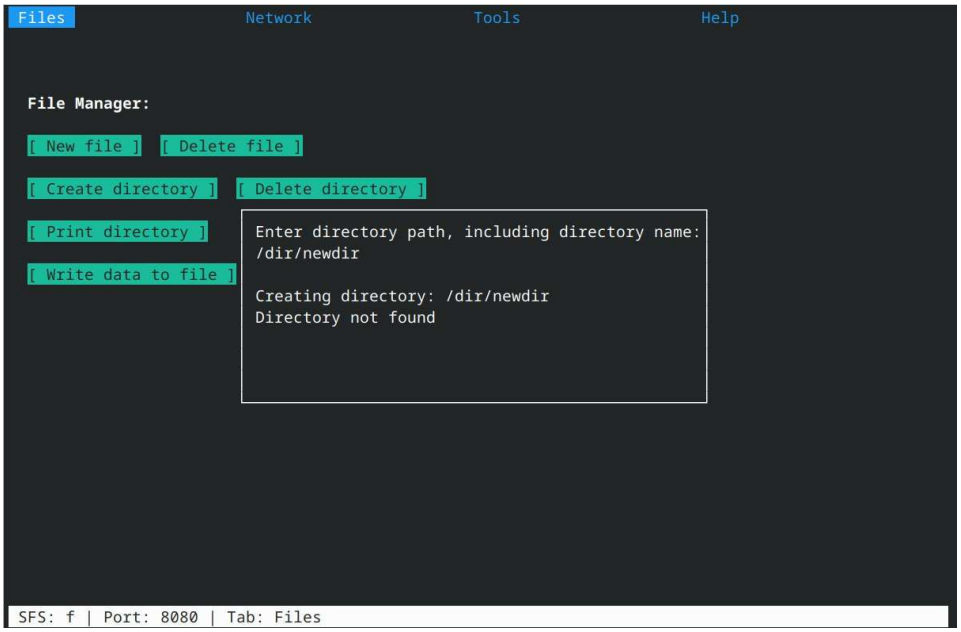

Рисунок 5.1.4 – отсутствие указанного каталога

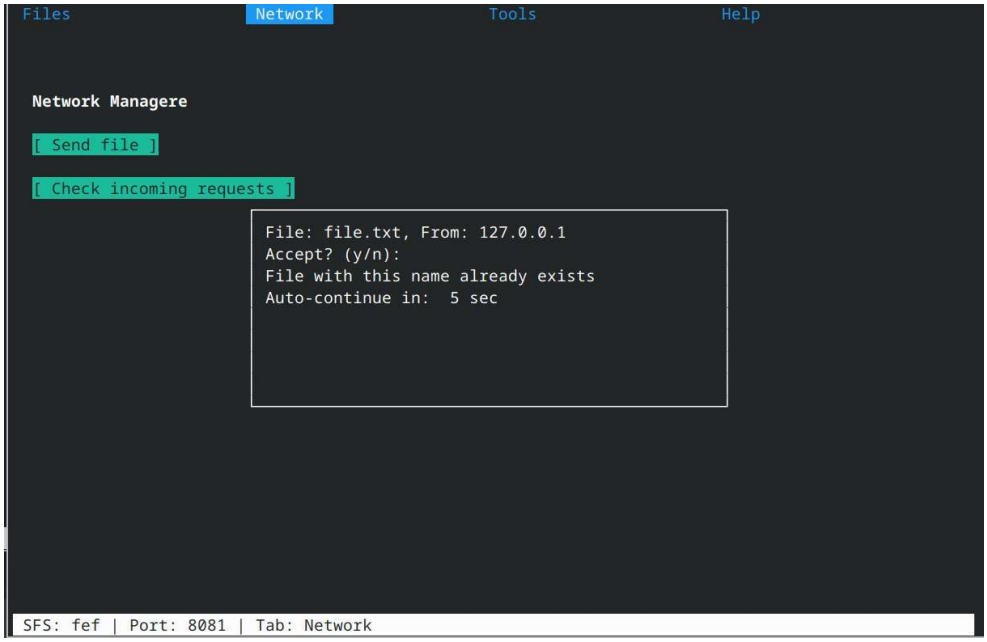

Рисунок 5.1.5 – файл с таким именем уже существует

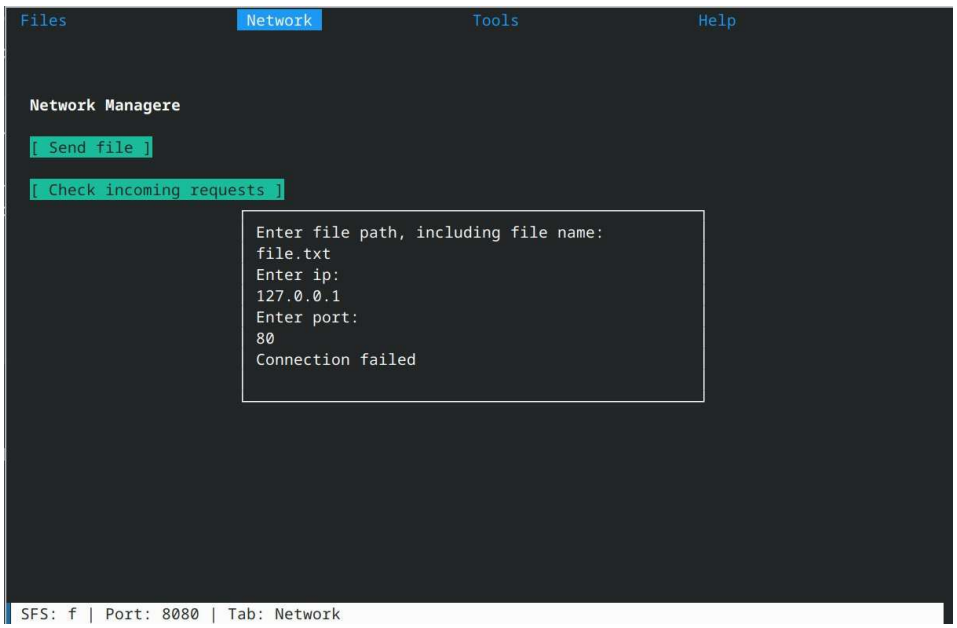

Рисунок 5.1.6 – некорректный ввод при отправке файла

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для запуска программы необходимо открыть файл “sfs.exe”, после чего появится окно программы.

Для того, чтобы начать пользоваться приложением, пользователю необходимо ввести начальные данные, в которые входят название образа файловой системы и порт, на котором будет запущен сервер по прослушиванию входящих запросов. После прохождения данного этапа появится основной интерфейс приложения.

Здесь можно будет увидеть четыре вкладки, в которых совмещен функционал, соответствующий названию каждой вкладки. При выборе одной из них интерфейс окна поменяется, отобразив кнопки с кратким описанием выполняемого действия. Также в самом низу окна для удобства отображается статус бар, на котором размещена информация, введенная пользователем при запуске программы.

Разберем возможности каждой из вкладки (Files, Network, Tools и Help).

6.1 Files

Данная вкладка (см. рисунок 6.1.1) по умолчанию открывается первой, поэтому в первую очередь пользователь видит функции данной вкладки, и это сделано неслучайно, так как здесь пользователь может совершать базовые действия над файловой системой. В их число входит создание и удаление файлов и каталогов, а также запись и вывод данных. В совокупности все эти функции позволяют полноценно взаимодействовать с файловой системой и дать объекты для использования функциям, которые расположены в других вкладках, поэтому раздел Files является основой программы.

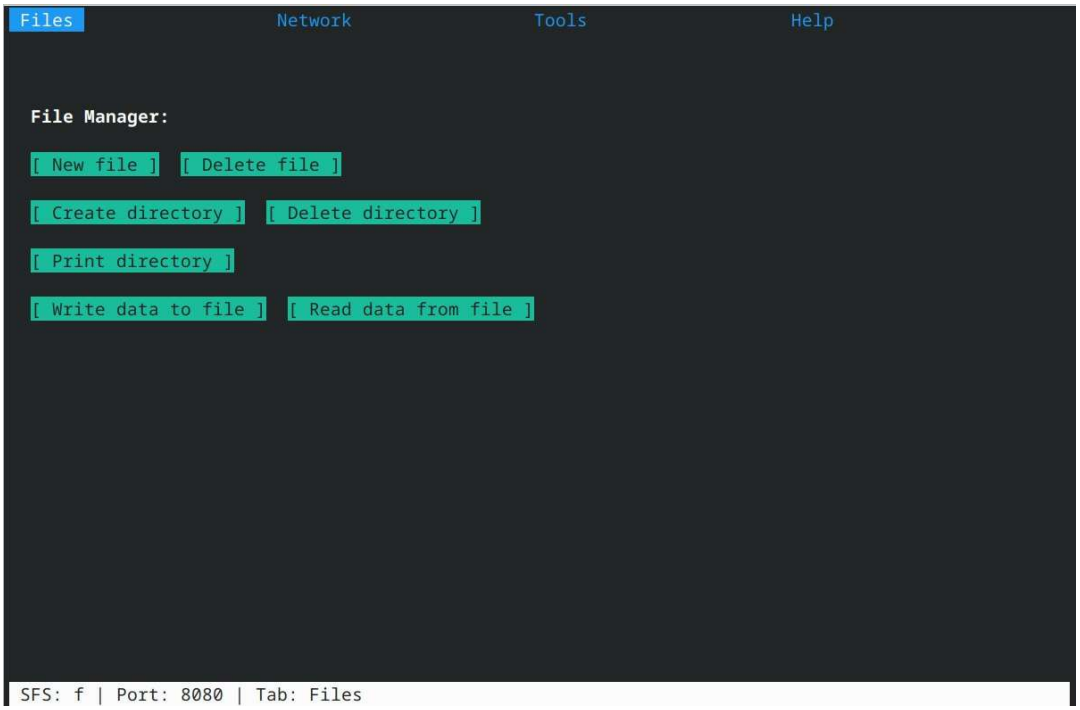

Рисунок 6.1.1 – интерфейс вкладки Files

6.2 Network

В этой вкладке (см. рисунок 6.2.1) пользователю предоставлено две кнопки. Кнопка Send file используется для инициализации отправки файла другому пользователю программы. При нажатии появляется диалоговое окно (см. рисунок 6.2.2), в котором требуется ввести путь к отправляемому файлу, IP адрес получателя и порт, на котором запущен его сервер. После ввода всех данных, начинается время ожидания, составляющее 10 секунд, за которое другой пользователь должен подтвердить или отклонить получение файла. Если время ожидания превышается, то в диалоговое окно выводится соответствующее сообщение (см. рисунок 6.2.3), и отправка файла завершается. Кнопка Check incoming requests служит для просмотра входящих запросов и их обработки. Если пользователю был отправлен запрос, то при нажатии по данной кнопке появится диалоговое окно (см. рисунок 6.2.4), в котором отобразится имя файла и IP адрес отправителя. При подтверждении файл будет скачен и помещен в корневой каталог, а при отклонении процесс получения файла завершится.

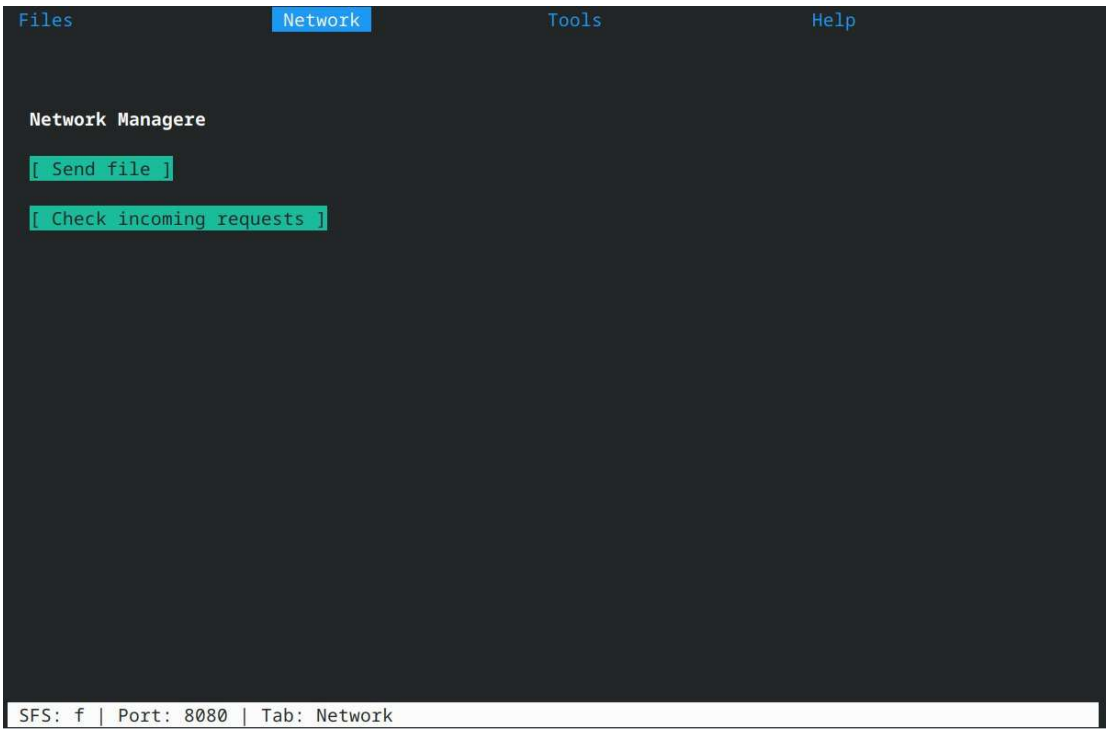

Рисунок 6.2.1 – интерфейс вкладки Network

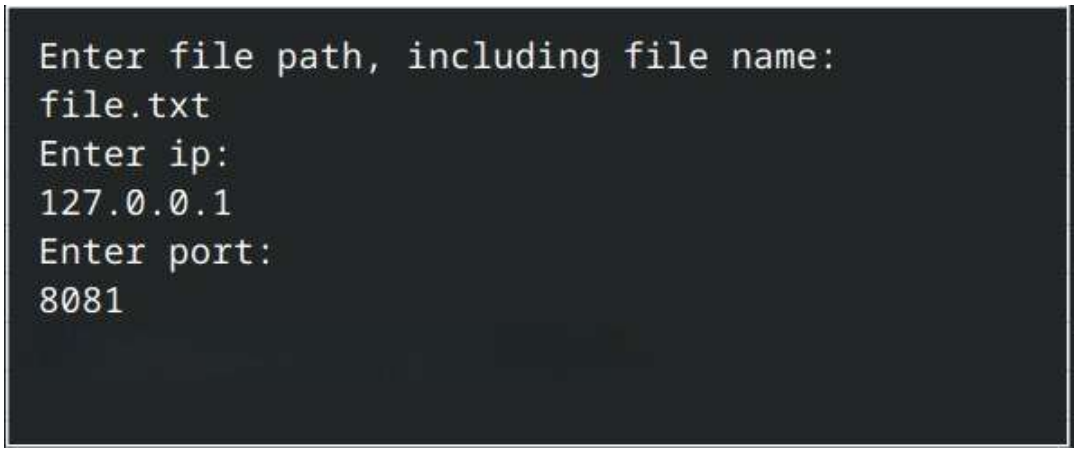

Рисунок 6.2.2 – диалоговое окно для отправки файла

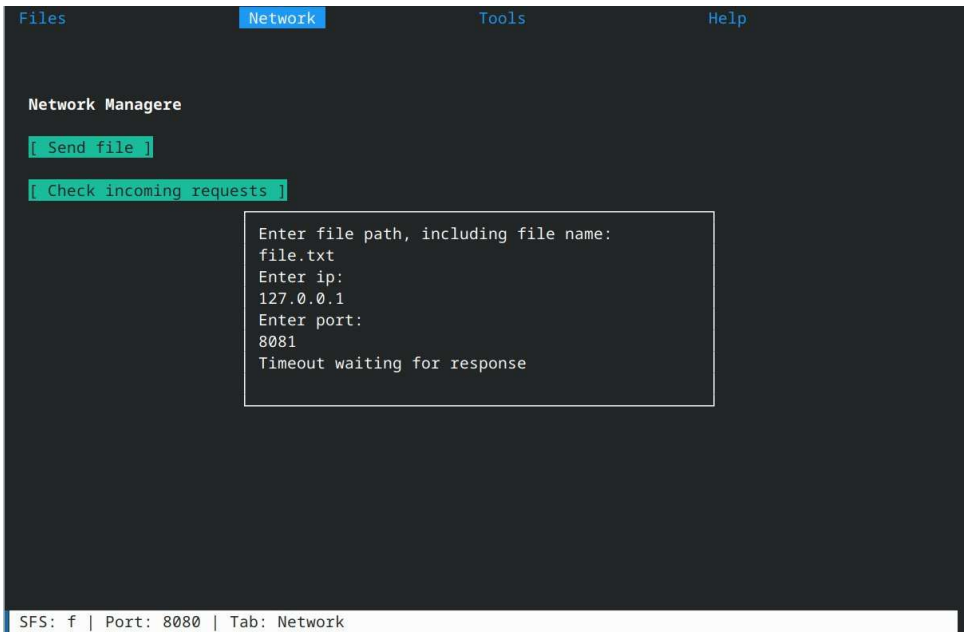

Рисунок 6.2.3 – сообщение о превышении времени ожидания

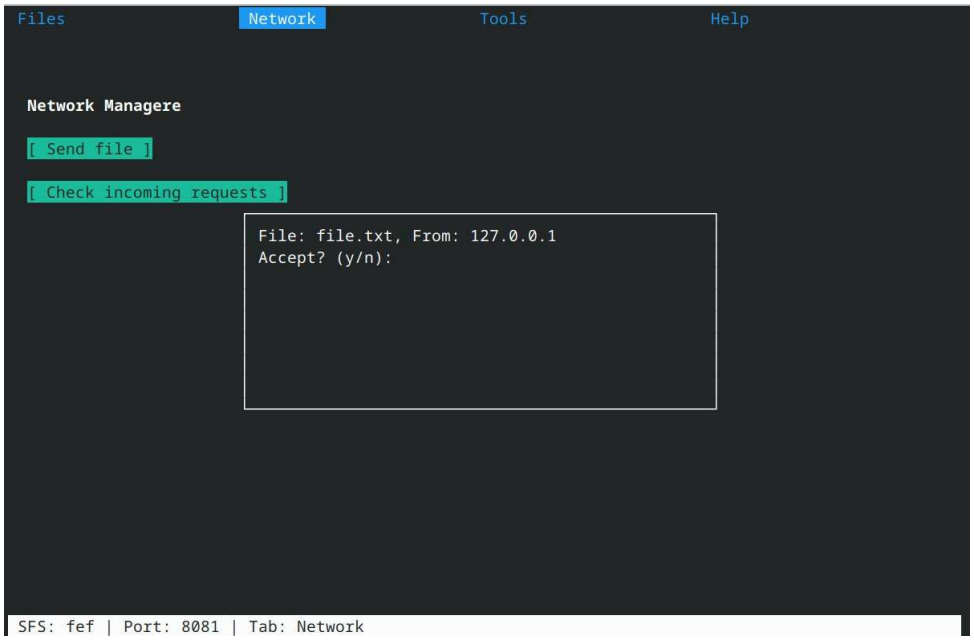

Рисунок 6.2.4 – диалоговое окно получения файла

6.3 Tools

В данной вкладке (см. рисунок 6.3.1) размещены функции по форматированию и проверке файловой системы. При проверке файловой системы появляется диалоговое окно, в котором отображаются результаты осуществленных действий, в число которых входит число исправленных и измененных блоков памяти, количество найденных дубликатов и общее свободное место в файловой системе. При дефрагментации файловой

системы в диалоговом окне выводится число блоков, которые были смещены для устранения фрагментации памяти и сохранения эффективного использования пространства. Также здесь доступна функция по очистке всех данных файлов (то есть вся информация, находящаяся в файлах, будет очищена, но сами файлы, ровно как и каталоги, останутся).

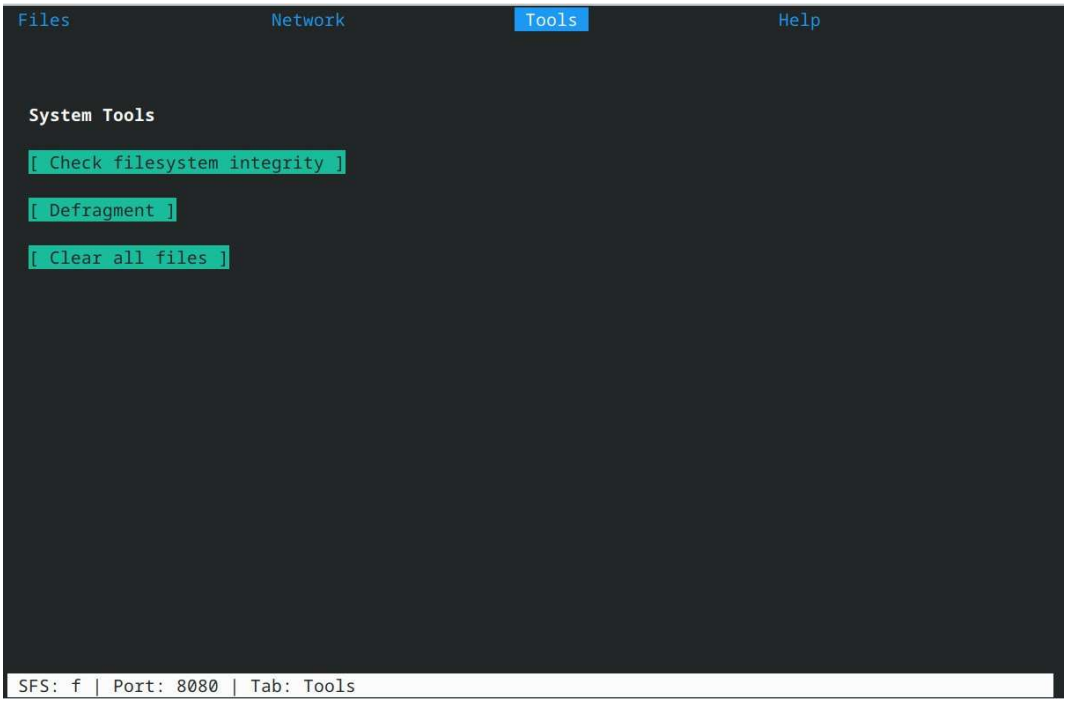

Рисунок 6.3.1 – интерфейс вкладки Tools

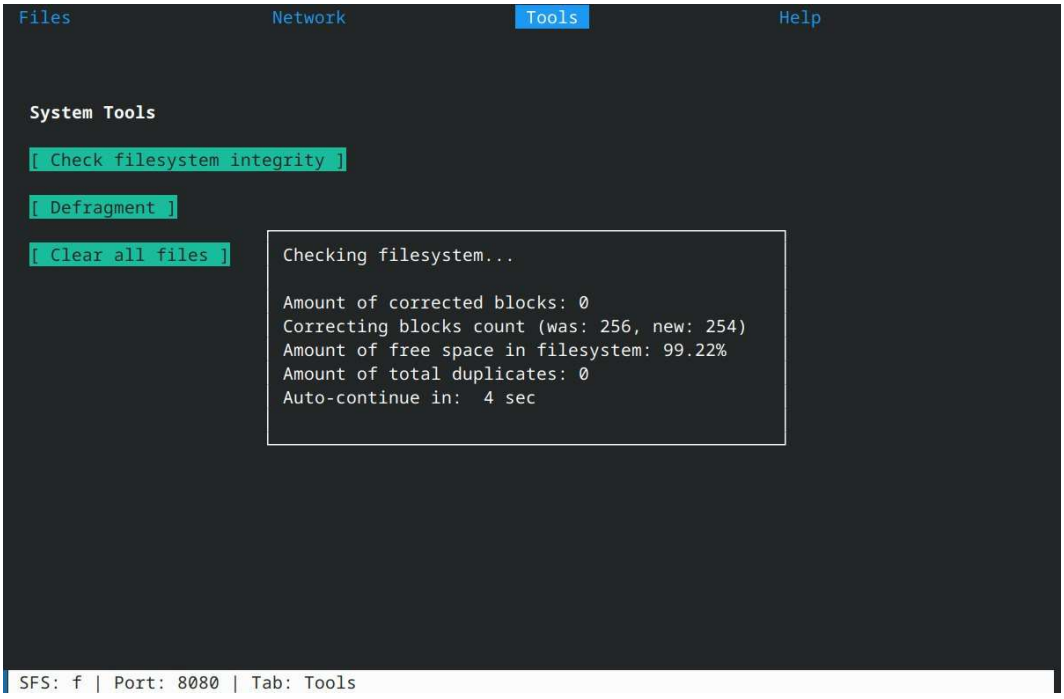

Рисунок 6.3.2 – проверка файловой системы

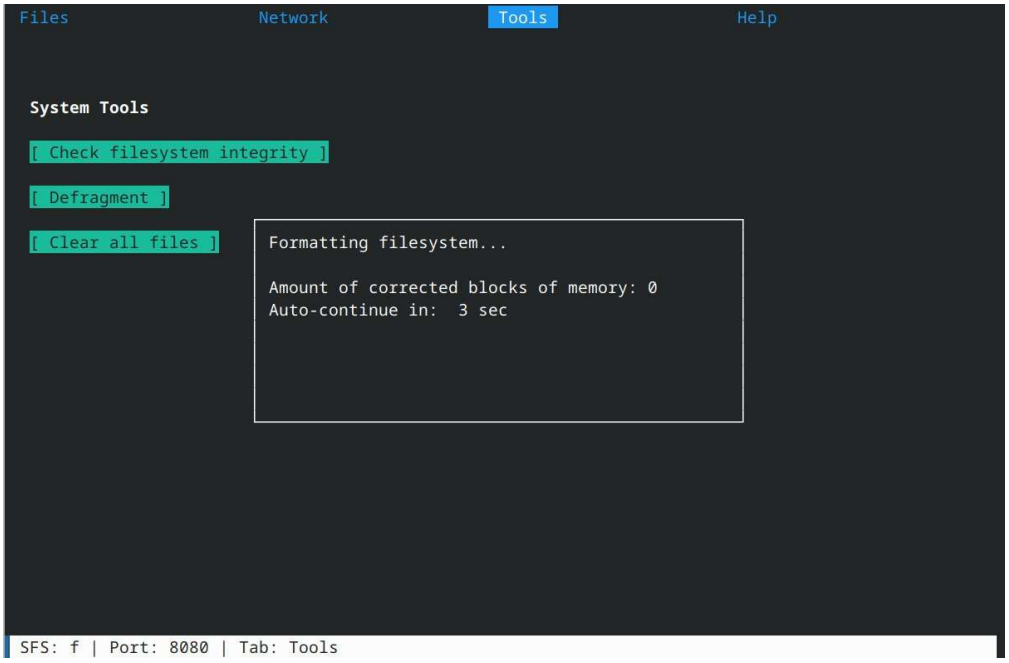

Рисунок 6.3.3 – дефрагментация файловой системы

6.4 Help

Данная вкладка (см. рисунок 6.4.1) носит информативный характер, предоставляю пользователю информацию о способе навигации по объектам файловой системы и другие полезные советы, которые помогают быстрее привыкнуть к программе и без проблем ориентироваться в ней при использовании.

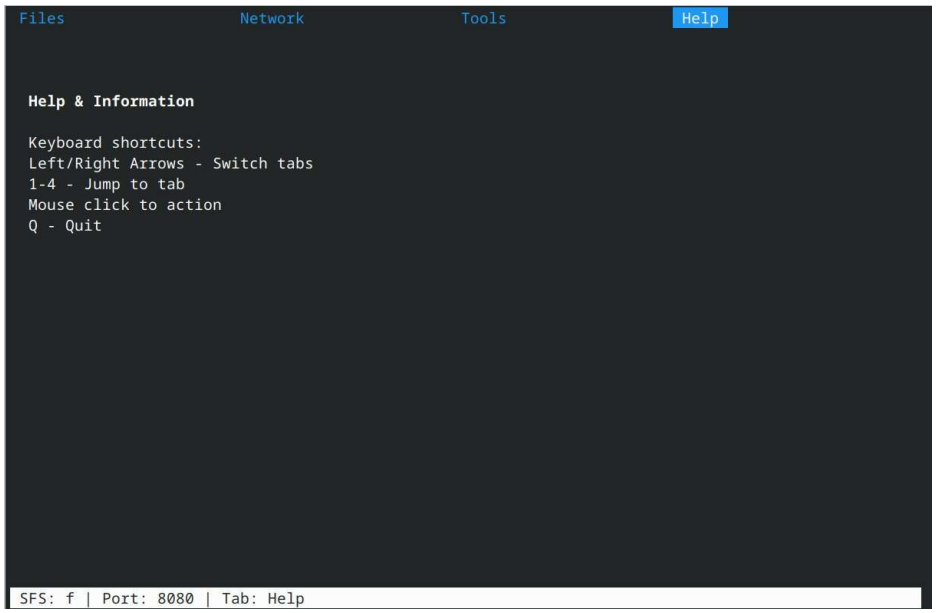

Рисунок 6.4.1 – интерфейс вкладки Help

ЗАКЛЮЧЕНИЕ

В результате работы над данным курсовым проектом была разработана программа, предоставляющая функционал для работы с файловой системой, ее объектами и дополнительными возможностями по модификации.

Для создания данного программного продукта были рассмотрены возможности библиотеки ncurses, с помощью которой был реализован программный интерфейс и осуществлено обработка пользовательских данных, углублены знания в создании архитектуры приложения для его дальнейшего улучшения и масштабирования. В ходе разработки были укреплены знания языка программирования C, который дает инструменты для тонкой работы с памятью и, следовательно, возможность написать программу для эффективного взаимодействия с ней.

Работа была разделена на такие этапы, как анализ существующих аналогов, литературных источников, постановка требований к проектируемому программному продукту, системное и функциональное проектирование, конструирование программного продукта, разработка программных модулей и тестирование проекта. После последовательного выполнения вышеперечисленных этапов разработки было получено исправно работающее приложение.

В дальнейшем планируется многогранное улучшение текущего функционала. Тема по созданию файловой системы огромна и включает в себя реализацию большого числа функций, многими из которых мы пользуемся повседневно, но даже не подозреваем об этом, ведь взаимодействие пользователя происходит лишь с интерфейсом, который позволяет вводить свои данные. Так, например, неотъемлемой частью файловой системы является работа с разными пользователями. Это учитывает, что с файлами может взаимодействовать несколько пользователей, и добавляет такое понятие, как права доступа. Хорошую оптимизацию при взаимодействии с файловой системой может дать кэширование, с использованием которого можно сохранять часто используемые файлы и при получении запроса на вывод данных из файла не начинать его поиск, а обращаться к кэшу и намного быстрее выводить его содержимое пользователю. Важной функцией является резервное копирование данных, так как в реальности не исключены случаи, когда файловая система повреждается и исправление поврежденной информации не представляется возможным, а резервное копирование позволяет восстановить данные в том виде, в котором они были до повреждения. На данный момент курсовой проект представляет собой решение, позволяющее легко расширять и использовать его в требуемых целях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Описание библиотеки ncurses [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/Ncurses. – Дата доступа: 30.04.2025

[2] Лав, Р. Linux. Системное программирование, 2-е издание / Р. Лав – СПб.: Питер, 2014. — 448 с.

Описание алгоритма AES [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/AES_(стандарт_шифрования). – Дата доступа: 30.04.2025.

СТП 01-2013. Дипломные проекты (работы): общие требования [Электронный ресурс] – Режим доступа: https://libeldoc.bsuir.by/bistream/123456789/2731/2/Rojnova_vich.pdf/. – Дата доступа: 30.04.2025.

ПРИЛОЖЕНИЕ А

(обязательное)

Схема алгоритма `create_file`

ПРИЛОЖЕНИЕ Б

(обязательное)

Схема алгоритма `check_incoming_requests`

ПРИЛОЖЕНИЕ В

(обязательное)

Схема алгоритма `aes_cipher`

ПРИЛОЖЕНИЕ Г

(обязательное)

Схема структурная

ПРИЛОЖЕНИЕ Д

(обязательное)

Ведомость документов

ПРИЛОЖЕНИЕ Е

(обязательное)

Листинг кода