Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные системы и среды

К защите допустить:
И.О. Заведующего кафедрой информатики
_____ С. И. Сиротко

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

НИЗКОУРОВНЕВЫЙ РЕДАКТОР БЛОЧНОГО УСТРОЙСТВА УРОВНЯ СЕКТОРОВ (NCURCES)

БГУИР КП 1-40 04 01 026 ПЗ

Студент К. Ю. Фроленко

Руководитель Н. Ю. Гриценко

Нормоконтролёр Н. Ю. Гриценко

Минск 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире управление информационными потоками и обеспечение надежного хранения данных являются одними из ключевых задач для любой вычислительной системы. Эффективное управление информацией требует не только работы с высокоуровневыми программными средствами, но и понимания принципов функционирования устройств хранения данных на низком уровне. Блочные устройства, такие как жёсткие диски, *SSD* и другие носители, представляют собой основу для формирования файловых систем, и их корректная работа критически важна для обеспечения стабильности и безопасности информационных систем. Низкоуровневые операции с этими устройствами, такие как чтение и запись данных на уровне секторов, позволяют получить прямой доступ к «сырым» данным, что даёт возможность проводить детальную диагностику, восстановление данных и анализ внутренних структур хранения.

Особое значение данный подход приобретает в условиях, когда традиционные методы работы с файловыми системами оказываются недостаточными для решения сложных задач, связанных с анализом ошибок, восстановлением информации после сбоев или атаками злоумышленников. Прямой доступ к блочным устройствам позволяет специалистам не только выявлять скрытые неисправности аппаратных средств, но и оптимизировать алгоритмы работы систем резервного копирования, шифрования и восстановления данных. В последние годы наблюдается значительный рост объёмов обрабатываемой информации, что требует разработки новых методов анализа и управления данными. В этом контексте разработка специализированных утилит для работы с блочными устройствами становится актуальной не только для научных исследований, но и для практических задач в области системного администрирования и информационной безопасности.

Использование библиотеки *ncurses* для создания текстового интерфейса является логичным выбором для реализации подобных утилит. *Ncurses*, разработанная для *Unix*-подобных систем, обеспечивает высокую переносимость и гибкость при работе в терминальном режиме. Благодаря ей можно создать интуитивно понятный и легковесный интерфейс, который позволит специалистам быстро взаимодействовать с программой даже на системах с ограниченными ресурсами. Этот подход снижает требования к оборудованию и повышает оперативность работы с утилитой, что особенно важно при проведении диагностики в режиме реального времени.

Целью курсового проекта является разработка низкоуровневого редактора блочного устройства, позволяющего осуществлять просмотр и редактирование данных на уровне секторов с использованием текстового интерфейса на базе *ncurses*. Для достижения поставленной цели необходимо решить следующие задачи:

1 Анализ архитектуры вычислительной системы и особенностей работы блочных устройств. Изучить структуру современных вычислительных систем, особенности функционирования блочных устройств в Linux и методы доступа

к «сырым» данным, а также рассмотреть проблемы безопасности и устойчивости при работе с низкоуровневыми операциями.

2 Исследование возможностей библиотеки *ncurses*. Оценить функциональные возможности и ограничения *ncurses* для создания удобного и интуитивно понятного текстового интерфейса, рассмотреть способы оптимизации работы терминала при большом объёме отображаемых данных, а также изучить методы обработки пользовательского ввода в условиях ограниченных ресурсов.

3 Разработка алгоритмов работы с блочным устройством. Спроектировать и описать алгоритмы для чтения, редактирования и записи данных на уровне секторов, обеспечить возможность тестирования с использованием файлов-образов, а также проанализировать потенциальные риски и предложить методы их минимизации.

4 Проектирование архитектуры разрабатываемой программы. Определить основные модули программного обеспечения, описать их взаимодействие и структуру, разработать функциональные и блок-схемы алгоритмов, а также обосновать выбор структуры программы с точки зрения масштабируемости, модульности и удобства дальнейшей поддержки.

5 Реализация программного продукта. Написать программный код на языке *C/C++* в среде *Visual Studio Code*, провести комплексное тестирование и отладку созданного редактора, обеспечить документирование кода и подготовку его к дальнейшему расширению функционала, а также провести сравнительный анализ эффективности разработанных алгоритмов.

Реализация данного проекта позволит не только продемонстрировать практические навыки работы с низкоуровневыми операциями ввода-вывода, но и углубленно изучить вопросы взаимодействия программного обеспечения с аппаратными средствами. Успешное выполнение проекта даст возможность оценить эффективность выбранных технологий и архитектурных решений, а также подчеркнуть значимость специализированных инструментов для анализа, диагностики и восстановления данных. Кроме того, разработанный редактор может стать основой для дальнейших исследований и разработки более сложных систем, способных работать в режиме реального времени, что особенно важно в условиях постоянно растущих объёмов обрабатываемой информации и усложнения современных вычислительных систем.

1 АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

1.1 Структура и архитектура вычислительной системы

В данном проекте в качестве вычислительной системы выбрана операционная система *Linux*, дистрибутивом которой является *Ubuntu*. *Ubuntu* считается одним из самых стабильных и широко используемых дистрибутивов, обеспечивающих открытый доступ к аппаратным ресурсам через файловую систему [1]. Такой подход позволяет осуществлять прямое взаимодействие с блочными устройствами посредством специальных файлов, расположенных в системном каталоге. Возможность выполнять операции чтения и записи данных на уровне секторов даёт разработчику высокий контроль над операциями ввода-вывода, что является критически важным для разработки программ, работающих с «сырыми» данными носителя.

Модульная архитектура *Ubuntu* предусматривает разделение ядра системы, драйверов и прикладного программного обеспечения, что позволяет гибко настраивать систему и оптимизировать её работу под специфические задачи проекта. В рамках данной работы реализуется текстовый интерфейс на основе библиотеки *ncurses*, которая нативно поддерживается в *Linux* и позволяет создавать лёгкие и интуитивно понятные консольные приложения [2]. Для наглядного иллюстрирования принципов взаимодействия компонентов можно использовать готовые схемы из открытых источников. Например, блок-схема, демонстрирующая взаимодействие приложения, файловой системы и блочного устройства, представлена на Рисунке 1.1. Такие схемы помогают понять, как данные проходят от пользовательского ввода до непосредственного доступа к физическим устройствам, а также как осуществляется обработка информации на низком уровне.

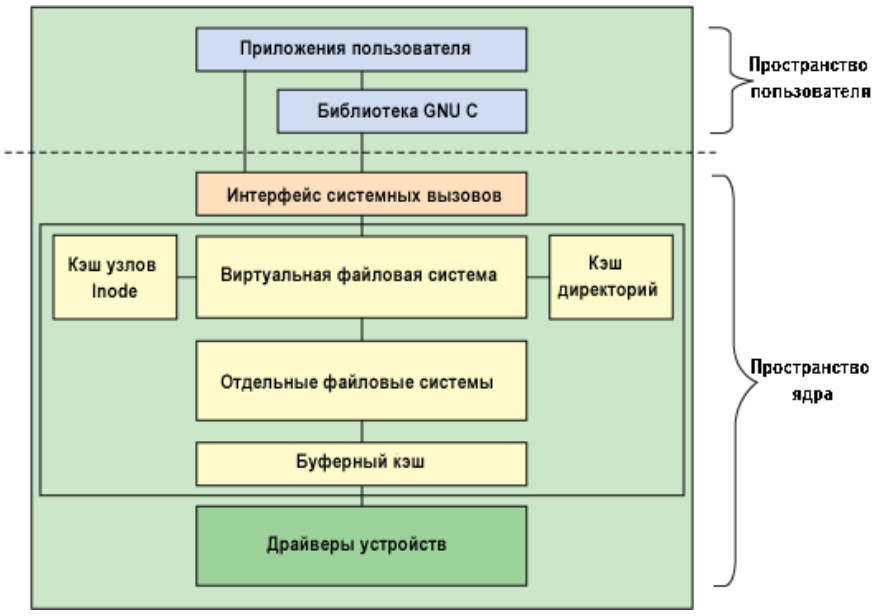

Рисунок 1.1 – Блок-схема взаимодействия приложения, файловой системы и блочного устройства

В данном разделе подробно описываются как теоретические основы работы с блочными устройствами, так и практические аспекты реализации доступа к данным, что позволяет создать надёжный фундамент для последующей разработки редактора. При этом особое внимание уделяется принципам взаимодействия на уровне секторов, методам кэширования и корректной обработке ошибок, а также вопросам безопасного тестирования с использованием псевдо-образов, позволяющих оттачивать навыки работы с «сырыми» данными без риска повреждения реальных носителей. Такой подход обеспечивает глубину понимания всех процессов, лежащих в основе низкоуровневого программирования, и формирует необходимую базу для дальнейшей реализации функционала редактора.

1.2 История, версии и достоинства

История развития операционных систем на базе *Linux* начинается в начале 1990-х годов, и за это время платформа претерпела значительные изменения, обеспечив себе репутацию высокостабильного, безопасного и производительного решения [3]. Важным фактором стала массовая поддержка сообщества разработчиков, которое активно вносило вклад в улучшение ядра и вспомогательных инструментов, благодаря чему *Linux* со временем вышла за рамки сугубо экспериментальной платформы и получила широкое признание в корпоративном сегменте, сфере высокопроизводительных вычислений и встраиваемых системах. Среди множества дистрибутивов особое место занимает *Ubuntu*, которая благодаря удобству установки, регулярным обновлениям и активному сообществу разработчиков получила широкое распространение в мире открытого программного обеспечения.

Преимущества *Ubuntu* заключаются в её открытости: доступ к исходному коду позволяет детально анализировать внутреннюю архитектуру системы, вносить необходимые модификации и оптимизировать её под конкретные задачи, что особенно важно при работе с низкоуровневым программированием [4]. Кроме того, *Ubuntu* обладает широким набором инструментов для диагностики и управления блочными устройствами. Использование таких утилит, как *fdisk*, *dd* и *losetup*, позволяет разработчику оперативно решать задачи по восстановлению и анализу данных. Подобная гибкость даёт возможность эффективно адаптировать систему как для исследовательских целей, так и для промышленной эксплуатации, обеспечивая надёжную платформу для разработчиков любых уровней.

Нативная поддержка текстовых интерфейсов посредством библиотеки *ncurses*, интегрированной в *Ubuntu*, даёт возможность создавать эффективные консольные приложения без необходимости обращения к сторонним библиотекам, что часто встречается в других операционных системах. Для иллюстрации эволюции платформы и преимуществ открытого программного обеспечения в *Linux* на Рисунке 1.2 приведена диаграмма, отражающая

ключевые этапы развития системы и демонстрирующая, как постепенно формировались её основные достоинства.

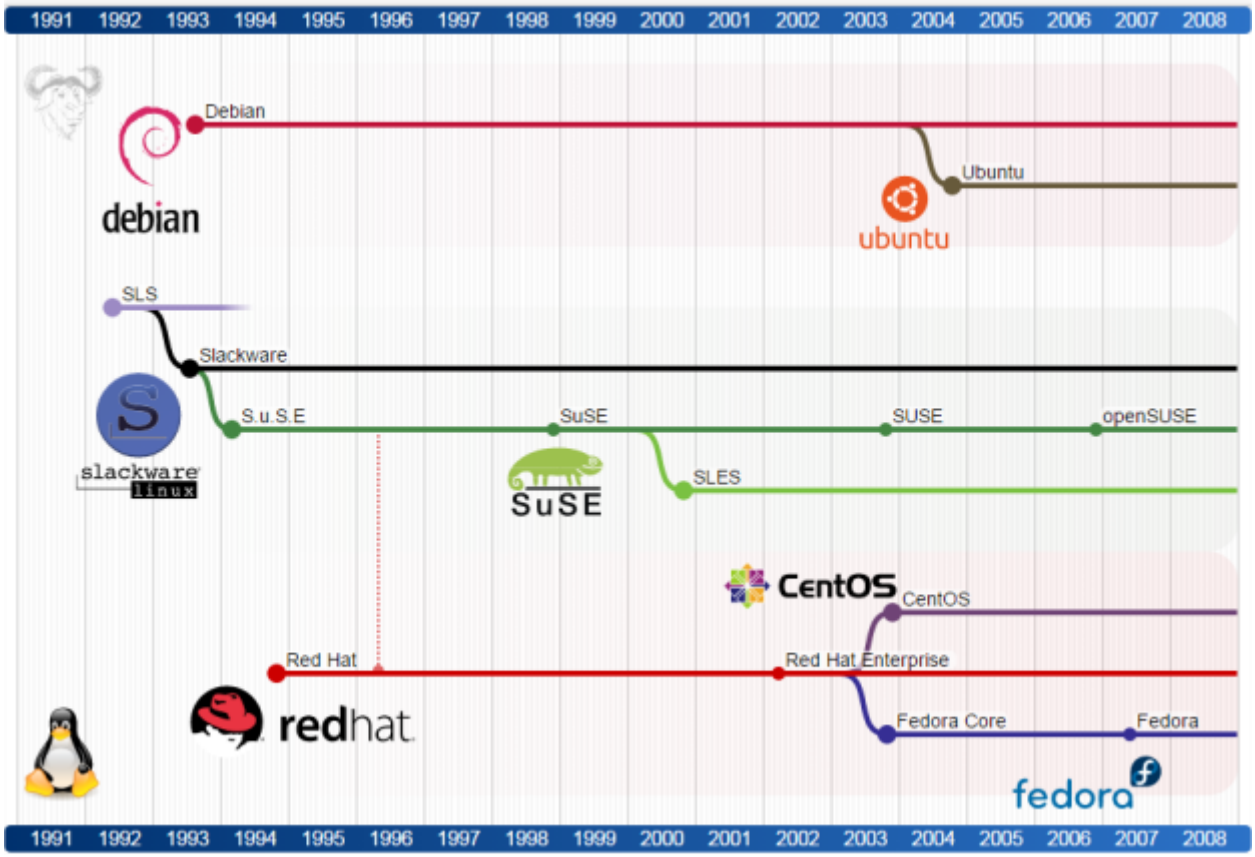

Рисунок 1.2 – Диаграмма развития *Linux* и ключевые этапы эволюции открытых операционных систем

Таким образом, богатая история *Ubuntu* и её технические преимущества делают эту систему надёжной базой для разработки проектов, требующих прямого доступа к аппаратным ресурсам и точного контроля над операциями ввода-вывода.

1.3 Обоснование выбора вычислительной системы

Выбор операционной системы *Linux*, дистрибутива *Ubuntu*, для реализации проекта обусловлен несколькими ключевыми факторами. Прежде всего, прямой доступ к блочным устройствам через файловую систему позволяет работать с данными на уровне секторов без лишних абстрактных слоёв, что существенно повышает точность и скорость выполнения операций. Кроме того, в *Ubuntu* доступны отлаженные средства разработки и отладки, такие как стандартные компиляторы и отладчики, что значительно упрощает процесс создания и оптимизации низкоуровневых приложений.

Нативная поддержка библиотеки *ncurses* является важным преимуществом при создании текстового интерфейса. В отличие от других операционных систем, где для обеспечения аналогичной функциональности

приходится прибегать к дополнительным решениям, *Linux* обеспечивает стабильную и эффективную работу консольных приложений «из коробки». Это позволяет сосредоточиться на реализации основной функциональности редактора без дополнительных затрат времени на настройку среды.

Открытая архитектура *Ubuntu* также играет существенную роль – возможность анализа и модификации исходного кода системы даёт разработчику гибкость в настройке платформы под специфические задачи проекта. Сравнительный анализ характеристик *Linux* и *Windows* представлен в Таблице 1.1, что наглядно демонстрирует преимущества выбранной системы для работы с блочными устройствами.

Таблица 1.1 – Сравнение *Linux* и *Windows* для работы с блочными устройствами

Параметр	*Linux*	*Windows*
Доступ к «сырым» устройствам	Прямой доступ через файловую систему	Доступ возможен, но требует дополнительных средств и прав
Набор системных вызовов	Полный, стандартизированный	Ограниченный, часто зависит от проприетарных решений
Поддержка текстовых интерфейсов	Нативная (*ncurses*)	Реализуется через дополнительные библиотеки (например, *PDCurses*)
Сообщество и документация	Обширное и активное	Меньше информации для задач низкоуровневой работы

Функциональные возможности, высокая степень контроля и удобство разработки в *Ubuntu* делают её оптимальным выбором для реализации данного проекта.

1.4 Анализ выбранной вычислительной системы

Подробный анализ показал, что *Ubuntu* обладает всеми необходимыми характеристиками для разработки низкоуровневого редактора блочного устройства. Система позволяет работать как с реальными блочными устройствами, так и с виртуальными образами, что обеспечивает возможность безопасного тестирования приложения без риска повреждения данных. Для обеспечения дополнительной безопасности и упрощения отладки в рамках данного проекта предусмотрено использование именно псевдо-образов (например, файлов или **.iso*), а не реальных устройств [5]. Разработка на языке *C/C++* в *Ubuntu*, с использованием стандартных компиляторов и библиотек,

способствует эффективному созданию и оптимизации программного кода. Надёжность и стабильность платформы гарантируют корректное выполнение низкоуровневых операций, а интеграция с *ncurses* обеспечивает создание удобного текстового интерфейса для конечного пользователя.

В таблице 1.2 представлен сравнительный анализ режимов работы с блочными устройствами, позволяющий увидеть преимущества использования виртуальных образов для тестирования по сравнению с работой с реальными устройствами.

Таблица 1.2 – Сравнительный анализ режимов работы с блочными устройствами

Режим работы	Преимущества	Недостатки
Работа с файловыми образами	Безопасность и возможность тестирования без риска повреждения данных	Может не учитывать все специфические характеристики реальных устройств
Работа с реальными устройствами	Полноценное тестирование и отражение реальных условий	Требует повышенных привилегий и несёт риск возникновения ошибок

Таким образом, *Ubuntu* является высокоэффективной и гибкой платформой, которая удовлетворяет всем требованиям проекта. Это позволяет создать надёжный и масштабируемый редактор, работающий как в условиях лабораторного тестирования, так и при эксплуатации с реальными аппаратными носителями.

1.5 Заключение

Подробный анализ архитектуры вычислительной системы демонстрирует, что операционная система *Linux*, дистрибутивом которой является *Ubuntu*, является оптимальной платформой для разработки низкоуровневого редактора блочного устройства. Прямой доступ к аппаратным ресурсам через файловую систему, широкий набор системных вызовов и наличие отлаженных средств разработки создают благоприятные условия для реализации программного обеспечения, способного работать с «сырыми» данными. Использование языка *C/C++* в сочетании с библиотекой *ncurses* даёт возможность создать компактное, высокопроизводительное и удобное приложение с текстовым интерфейсом.

Для повышения надёжности и упрощения отладки в проекте предусмотрена работа с псевдо-образами (файлами или **.iso*) вместо реальных устройств, что помогает избежать потенциальных ошибок и повреждения данных. Проведённый анализ подчёркивает высокую эффективность и перспективность выбранной вычислительной системы для дальнейшей разработки и оптимизации программного продукта.

2 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

2.1 Структура и архитектура платформы

Программная платформа, в рамках которой создаётся низкоуровневый редактор блочного устройства, включает в себя операционную систему *Ubuntu*, набор инструментов для сборки и отладки (*GCC*, *GDB*, *Make* или *CMake*), библиотеку *ncurses*, а также различные системные утилиты. Подобная конфигурация позволяет эффективно разрабатывать программы, взаимодействующие с аппаратными ресурсами напрямую, и обеспечивает доступ ко всем необходимым функциям ядра *Linux*. Благодаря модульной архитектуре, каждая часть платформы (компиляторы, библиотеки, системные вызовы) может быть обновлена или заменена без кардинальных изменений в остальных компонентах. Этот подход даёт разработчику высокую степень гибкости, что особенно важно при работе с «сырыми» данными, требующими тонкого контроля над операциями ввода-вывода [6].

Важной особенностью структуры является тесная интеграция *ncurses* и стандартных системных библиотек, позволяющая программно управлять терминалом. Приложение, написанное на *C/C++* и использующее *ncurses*, может выводить сложное текстовое меню, динамически обновлять части экрана, а также корректно обрабатывать ввод с клавиатуры. При этом взаимодействие с блочными устройствами осуществляется через *dev*, что означает прямой доступ к секторам, минуя высокоуровневые файловые *API* [7]. В результате разработчик может создавать одновременно удобные для пользования консольные интерфейсы и манипулировать физической структурой носителя, используя одну и ту же платформу.

2.2 История, версии и достоинства

Развитие программных инструментов для *Linux* тесно переплетается с историей свободных проектов *GNU* и идёт с 1980-х годов, когда появились первые версии *GCC*, *GDB* и *Make*. Со временем эти средства стали де-факто стандартом для *Unix*-подобных систем, так как позволяли компилировать и отлаживать программы на низком уровне без необходимости в проприетарных решениях [8]. Библиотека *ncurses*, эволюционировавшая из *curses*, заняла особое место, дав возможность создавать полноценные текстовые интерфейсы, не зависящие от графической среды. *Ubuntu*, появившаяся в 2004 году, упростила распространение этих инструментов, объединив их в системных репозиториях и предоставив удобный менеджер пакетов *apt* [9].

Достоинством выбранной платформы является стабильность и открытость, унаследованные от классических *Unix*-подходов. Компиляторы (*GCC, G++*), входящие в *GNU Toolchain*, обеспечивают поддержку множества языков и флагов оптимизации, а *GDB* позволяет пошагово отлаживать работу программы, вплоть до уровня регистров, что особенно ценно при возникновении критических ошибок на уровне «сырого» ввода-вывода [10].

Ncurses предоставляет богатый функционал для организации окон и обработки пользовательского ввода, при этом программы на её основе могут работать даже в режиме минимального окружения без графического сервера. Такое сочетание гибкости и простоты делает *Ubuntu* оптимальным выбором для проектирования низкоуровневых утилит [11].

2.3 Обоснование выбора платформы

Основанием для выбора именно *Ubuntu* и *GNU*-инструментов служит высокая доступность всех необходимых компонентов «из коробки». Разработчику не требуется искать сторонние библиотеки или вручную собирать компилятор, поскольку всё базовое окружение можно установить за несколько минут через стандартные репозитории [12]. В дополнение к этому, открытость кода, характерная для *Linux*, даёт возможность при необходимости анализировать или даже модифицировать внутренние механизмы ядра, что бывает нужно при отладке низкоуровневых операций.

Дополнительным аргументом выступает удобство тестирования. Работа с псевдо-образами (файлами, которые могут монтироваться как блочные устройства) не только повышает безопасность, но и позволяет легко имитировать различные сценарии использования, меняя параметры образа (размер, структуру, наличие разметки и т. д.) [13]. Разработанная утилита может последовательно читать и писать сектора, в то время как система воспринимает файл как полноценный диск. Это существенно ускоряет цикл «написание кода – тестирование – отладка», позволяя выявлять и исправлять ошибки без риска порчи реальных накопителей.

Нельзя не отметить и богатую документацию. Официальные руководства, руководства для программистов, страницы *man* и сайты сообществ подробно описывают все аспекты работы с системными вызовами, форматами бинарных файлов, а также сложные случаи настройки окружения [14]. Всё это делает платформу максимально прозрачной и дружелюбной к разработчику.

2.4 Анализ программного обеспечения для написания программы

Процесс написания низкоуровневого редактора блочного устройства включает несколько этапов. Сначала создаётся структура проекта (директории для исходных файлов, заголовков, скриптов сборки), а затем выбирается система сборки – *Make* или *CMake* – в зависимости от предпочтений команды. Исходный код, содержащий вызовы системных функций чтения и записи на блочном устройстве, компилируется *GCC*, который поддерживает целый ряд ключей оптимизации и предупреждений (например, *-O2, -Wall*), позволяющих на ранней стадии выявлять потенциальные ошибки.

Наиболее сложная часть разработки обычно связана с отладкой. Для этого используется *GDB*, дающий возможность останавливать выполнение программы, просматривать содержимое памяти и регистров, пошагово

анализировать логику. Если требуется посмотреть, какие именно системные вызовы совершает редактор, в ход идёт *strace* или *ltrace*. Первый отображает обращения к ядру *Linux* (например, чтение сектора), а второй помогает отслеживать вызовы библиотечных функций. Параллельно возможно использование инструментов монтирования псевдо-образов, чтобы проверять корректность работы с виртуальными дисками в безопасном режиме. В результате выстраивается чёткая методика: небольшой участок кода – сборка – тест на псевдо-образе – отладка; это ускоряет выявление логических ошибок и упрощает внесение правок.

Применение библиотеки *ncurses* придаёт редактору более удобный вид, чем традиционные консольные приложения с минимальным интерфейсом. Программа может выводить на экран текущее состояние загруженного сектора, цветом выделять изменённые байты, а также реагировать на команды пользователя без перезапуска (обновляя окно в реальном времени). Подобный подход облегчает ориентирование в массиве двоичных данных, что существенно экономит время при анализе и редактировании «сырой» структуры.

2.5 Заключение

Подводя итоги, можно сказать, что выбранная программная платформа на базе *Ubuntu*, *GNU*-инструментов и библиотеки *ncurses* представляет собой оптимальное решение для разработки низкоуровневого редактора блочного устройства. Платформа обеспечивает прямой доступ к аппаратным ресурсам, гибкость настройки, высокую стабильность и богатый набор средств для сборки и отладки. Использование псевдо-образов позволяет безопасно тестировать операции с секторами, а интерактивный текстовый интерфейс, реализованный с помощью *ncurses*, делает работу с приложением удобной и эффективной.

Кроме того, данная платформа предоставляет широкие возможности для дальнейшей интеграции с современными инструментами автоматизации и мониторинга. Например, разработчик может дополнительно настроить систему для динамического отображения логов и системных показателей в режиме реального времени, что позволяет не только ускорить отладку, но и повысить общую прозрачность работы приложения. Такой комплексный подход гарантирует, что итоговый продукт будет не только функциональным, но и легко масштабируемым, что крайне важно для проектов, связанных с критически важными операциями ввода-вывода.

3 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

3.1 Обоснование необходимости разработки

3.2 Использованные технологии программирования

3.3 Взаимосвязь архитектуры системы с программным обеспечением

3.4 Заключение

4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ

4.1 Введение в функциональные возможности

4.2 Описание основных функций программного обеспечения

4.3 Структура программного обеспечения

4.4 Обеспечение эффективности и оптимизации

4.5 Возможности дальнейшего расширения функциональности

4.6 Заключение

5 АРХИТЕКТУРА РАЗРАБАТЫВАЕМОЙ ПРОГРАММЫ

5.1 Общая структура программы

5.2 Описание функциональной схемы программы

5.3 Описание блок схемы алгоритма программы

5.4 Обработка неопределенных результатов

5.5 Заключение

ЗАКЛЮЧЕНИЕ

СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

[1] Ubuntu. Официальная документация [Электронный ресурс]. – Режим доступа: https://ubuntu.com/tutorials. – Дата доступа: 20.02.2025.

[2] GNU Ncurses. Ncurses Library [Электронный ресурс]. – Режим доступа: https://invisible-island.net/ncurses/announce.html. – Дата доступа: 20.02.2025.

[3] Linux Kernel Archives. История и развитие ядра Linux [Электронный ресурс]. – Режим доступа: https://www.kernel.org/category/about.html. – Дата доступа: 21.02.2025.

[4] The Linux Documentation Project. Руководства по Linux [Электронный ресурс]. – Режим доступа: https://www.tldp.org/. – Дата доступа: 21.02.2025.

[5] IBM Developer. Introduction to Linux block devices [Электронный ресурс]. – Режим доступа: https://developer.ibm.com/tutorials/. – Дата доступа: 21.02.2025.

[6] Freed. Linux from Scratch [Электронный ресурс]. – Режим доступа: https://www.linuxfromscratch.org/. – Дата доступа: 23.02.2025.

[7] man7.org. Linux man pages [Электронный ресурс]. – Режим доступа: https://man7.org/linux/man-pages/. – Дата доступа: 23.02.2025.

[8] Stack Exchange. Unix & Linux Q&A [Электронный ресурс]. – Режим доступа: https://unix.stackexchange.com/. – Дата доступа: 24.02.2025.

[9] Stallman R. The GNU Project [Электронный ресурс]. – Режим доступа: https://www.gnu.org/gnu/gnu-history.en.html. – Дата доступа: 24.02.2025.

[10] Open Source Initiative. About the Open Source Initiative [Электронный ресурс]. – Режим доступа: https://opensource.org/about. – Дата доступа: 24.02.2025.

[11] Visual Studio Code. Official Documentation [Электронный ресурс]. – Режим доступа: https://code.visualstudio.com/docs. – Дата доступа: 24.02.2025.

[12] Oracle. Linux System Calls Overview [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/en/operating-systems/linux/. – Дата доступа: 23.02.2025.

[13] CMake. Official Documentation [Электронный ресурс]. – Режим доступа: https://cmake.org/documentation/. – Дата доступа: 24.02.2025.

[14] FHS. Filesystem Hierarchy Standard [Электронный ресурс]. – Режим доступа: https://refspecs.linuxfoundation.org/fhs.shtml. – Дата доступа: 24.02.2025.

ПРИЛОЖЕНИЕ А
(обязательное)
Справка о проверке на заимствования

ПРИЛОЖЕНИЕ Б
(обязательное)
Листинг программного кода

ПРИЛОЖЕНИЕ В
(обязательное)
Функциональная схема алгоритма, реализующего программное средство

ПРИЛОЖЕНИЕ Г
(обязательное)
Блок схема алгоритма, реализующего программное средства

ПРИЛОЖЕНИЕ Д
(обязательное)
Графический интерфейс пользователя

ПРИЛОЖЕНИЕ Е
(обязательное)
Ведомость курсового проекта