Симулатор на драйвер за файлова система Ext2

Глава 1. Увод

1.1. Описание и идея на проекта

Идеята на проекта е да се създаде приложение на езика C++, което симулира основните функционалности на драйвер за файловата система Ext2. Програмата позволява на потребителя да взаимодейства с дисков образ, предварително форматиран с Ext2, като извършва операции като създаване на файлове и директории, изтриване, четене на съдържанието на файлове, запис и добавяне на съдържание към файлове, както и навигация в йерархичната структура на директориите. Този симулатор предоставя контролирана среда за изследване и работа с Ext2 структури без риск за реалната файлова система на потребителя.

1.2. Цел и задачи на разработката

Основната цел на разработката е придобиването на по-дълбоко практическо разбиране на вътрешните структури, организацията и принципите на работа на файловите системи под Linux на по-ниско ниво, с конкретен фокус върху файловата система Ext2.

За постигането на тази цел бяха поставени следните основни задачи:

1. **Проучване на спецификацията на Ext2:** Детайлно запознаване със структурите на файловата система Ext2, включително:
    - Суперблок (Superblock): съдържащ глобална информация за файловата система.
    - Дескриптори на блокови групи (Block Group Descriptors): описващи всяка блокова група.
    - Битови полета (Bitmaps): за проследяване на свободните блокове и индексни възли.
    - Индексни възли (Inodes): съдържащи метаданни за файловете и директориите.
    - Директорийни записи (Directory Entries): свързващи имената на файловете с техните индексни възли.
2. **Проектиране на софтуерна архитектура:** Създаване на модулна структура от класове на C++, които да представят основните компоненти на Ext2 и да капсулират логиката за работа с тях.
3. **Реализация на основни операции с дисковия образ:**
    - Четене и интерпретиране на метаданните на Ext2 (суперблок, дескриптори на групи, таблици на индексни възли) от предоставен дисков образ.
    - Запис на промените в метаданните обратно в дисковия образ.
4. **Имплементиране на команди за управление на файлове и директории:**
    - Извеждане на дървовидна структура на директориите (tree).

    - Четене на съдържанието на файлове (cat).
    - Създаване на нови празни файлове (touch).
    - Създаване на нови директории (mkdir).
    - Изтриване на файлове и директории (включително рекурсивно) (rm).
    - Запис на данни във файлове (презаписване и добавяне) (write, append).
5. **Разработка на потребителски интерфейс:** Създаване на прост команден интерпретатор (CLI), който позволява на потребителя да изпълнява имплементираните файлови операции.
6. **Разработка на помощни класове:** Имплементиране на собствени версии на основни структури от данни като динамичен низ (utils::string) и динамичен масив (utils::vector)

1.3. Структура на документацията

- **Глава 1 (Увод)** - представя описание, цел и задачи на проекта, както и структурата на самата документация.
- **Глава 2 (Преглед на предметната област)** - разглежда теоретичните основи на файловата система Ext2 и свързаните с нея концепции.
- **Глава 3 (Проектиране)** - описва общата архитектура на софтуерното решение и дизайна на основните компоненти.
- **Глава 4 (Реализация и тестване)** - се фокусира върху ключови моменти от имплементацията на класовете и функционалностите, както и подходите за тестване.
- **Глава 5 (Заключение)** обобщава постигнатите резултати и набелязва възможни насоки за бъдещо развитие на проекта

Глава 2. Преглед на предметната област

2.1. Основни дефиниции, концепции и алгоритми, които ще бъдат използвани

Файловата система Ext2 (Second Extended Filesystem) е една от класическите и влиятелни файлови системи в света на Linux. Въпреки че по-нови системи като Ext3, Ext4 и други предлагат допълнителни функционалности (като журналиране), разбирането на Ext2 е фундаментално за разбирането на много от техните основни принципи.

Основните компоненти и концепции на Ext2, релевантни за този проект, са:

1. **Дисков образ (Disk Image):**
    - В контекста на проекта, това е файл (build/ext2.img), който се третира като физическо блоково устройство. Той съдържа всички структури и данни на симулираната Ext2 файлова система.
2. **Блокове (Blocks):**
    - Файловата система разделя дисковото пространство на блокове с фиксиран размер (напр. 1KB, 2KB, 4KB). Блокът е най-малката единица за адресиране на данни. Размерът на блока се дефинира при създаването на файловата система и се съхранява в суперблока.
    - В проекта, класът Block служи като базов клас за представяне на структури, които се четат и записват от/в дисковия образ на определен офсет и с определен размер.
3. **Суперблок (Superblock):**
    - Това е труктура от данни, която съдържа глобална информация за файловата система. Обикновено се намира на фиксиран офсет от началото на дяла (1024 байта за Ext2).
    - Съдържа информация като: общ брой индексни възли (inodes) и блокове, брой свободни индексни възли и блокове, размер на блока, брой индексни възли и блокове на група, магическо число (0xEF53), състояние на файловата система и др.
    - В проекта е представен от класа SuperBlock.
4. **Блокови групи (Block Groups):**
    - Ext2 разделя дисковото пространство на блокови групи. Всяка група съдържа копие на суперблока и дескрипторите на групи (в някои групи), битови полета за блокове и inode-и, част от таблицата на inode-ите и блокове с данни.
5. **Дескриптор на блокова група (Block Group Descriptor - BGD):**
    - Всеки BGD описва своята група: указатели към битовите полета и таблицата на inode-ите, брой свободни блокове/inode-и, брой директории.
    - Съхраняват се в **Таблица на дескрипторите на блокови групи (BGDT)**.
    - В проекта са представени от класовете BlockGroupDescriptor и BlockGroupDescriptorTable.
6. **Индексен възел (Inode):**
    - Всеки файл/директория се представя от inode, съдържащ метаданни: тип, права, собственик, размер, времена (i_atime, i_mtime, i_ctime, i_dtime), брой връзки, указатели към блокове с данни (директни, еднократно, двукратно, трикратно индиректни).
    - В проекта е представен от класа Inode. Управляват се от InodeTable.

7. **Битови полета (Bitmaps):**
    - **Block Bitmap:** Всеки бит отговаря на блок с данни в групата (0 - свободен, 1 - зает).
    - **Inode Bitmap:** Всеки бит отговаря на inode в групата.
    - Използват се за заделяне/освобождаване на ресурси. Операциите с тях са в utils.
8. **Директории и Директорийни записи (Directory Entries):**
    - Директориите са файлове, съдържащи списък от LinkedDirectoryEntry структури.
    - Всеки запис свързва име с номер на inode и съдържа тип на файла, дължина на записа и името.
9. **Алгоритми за работа с данни:**
    - **Четене/Запис на файл:** Включва намиране на inode, работа с директни/индиректни указатели, заделяне/освобождаване на блокове чрез битови полета и актуализиране на метаданни.
    - **Създаване/Изтриване на файл/директория:** Включва операции с inode-и, блокове, директорийни записи в родителската директория и актуализиране на всички свързани метаданни (битови полета, броячи в суперблок и BGD).

2.2. Дефиниране на проблеми и сложност на поставената задача

- **Основни предизвикателства при реализацията:**
    - **Създаване и изтриване на файлове и директории**
    - **Управление на индиректните блокове**
    - **Работа с битови полета**.
    - **Рекурсивно изтриване на директории**
    - **Навигация в пътища**

2.3. Подходи и методи за решаване на поставените проблемите

- **Представяне на структурите:** Основните компоненти на Ext2 са представени чрез C++ класове (SuperBlock, Inode, Ext2 и др.), а по-простите дискови формати като структури (LinkedDirectoryEntry).
- **Алгоритми:** Използвани са линейно сканиране за търсене в битови полета, итеративно обхождане на директорийни записи, рекурсивни подходи за работа с индиректни блокове и опростена стратегия за локалност при заделяне на ресурси.
- **ООП и Design Patterns:**
    - **Капсулация, Наследяване, Полиморфизъм:** Стандартни ООП принципи. FileSystem е абстрактен базов клас, Ext2 е конкретна имплементация. Block е базов за дисковите структури.
    - **Композиция:** Ext2 съдържа SuperBlock, InodeTable, BlockGroupDescriptorTable.
    - **Template Method Pattern:** Класът Block дефинира общ алгоритъм за read()/write(), като конкретизацията на буфера се оставя на наследниците.
    - **Facade Pattern (в широк смисъл):** Класът Ext2 предоставя опростен интерфейс към сложната Ext2 подсистема.към сложната Ext2 подсистема.

2.4. Потребителски (функционални) изисквания и качествени

(нефункционални) изисквания

2.4.1. Функционални изисквания

Системата трябва да предоставя на потребителя следните основни функционалности чрез команден интерфейс (CLI):

- **Инициализация:** Системата трябва да може да зареди и интерпретира съществуващ дисков образ, форматиран с Ext2.
- **Навигация и преглед:**
  - Да позволява извеждане на дървовидната структура на директориите от зададен път (tree).
  - Да позволява извеждане на съдържанието на текстов файл (cat).
- **Управление на файлове:**
  - Да позволява създаването на нови, празни файлове (touch).
  - Да позволява запис на съдържание в нов или съществуващ файл, като презаписва старото съдържание (write).
  - Да позволява добавяне на съдържание към края на съществуващ файл (append).
  - Да позволява изтриването на файлове (rm).

- **Управление на директории:**
  - Да позволява създаването на нови директории (mkdir).
  - Да позволява изтриването на празни директории (rm).
  - Да позволява рекурсивното изтриване на директории и тяхното съдържание (rm -r).
- **Помощна информация:**
  - Да предоставя информация за наличните команди и тяхното използване (help).
- **Изход:**
  - Да позволява коректно прекратяване на работата на приложението (exit).

2.4.2. Качествени (нефункционални) изисквания

Предвид естеството на проекта като курсова разработка, не са били поставени строги количествени нефункционални изисквания. Въпреки това, при разработката са взети предвид следните качествени аспекти:

- **Коректност на операциите:** Основен приоритет е била коректната имплементация на файловите операции съгласно спецификацията на Ext2, така че файловата система да остава в консистентно състояние след всяка операция.
- **Леснота на използване (Usability):** Командният интерфейс (CLI) е проектиран да бъде интуитивен и да предоставя обратна връзка на потребителя за успешното или неуспешно изпълнение на командите, както и съобщения за грешки.

Глава 3. Проектиране

3.1. Обща архитектура – ООП дизайн

Основните класове в архитектурата на проекта са:

1. **FileSystem (абстрактен базов клас):**
    - **Предназначение:** Дефинира общия интерфейс за всички поддържани типове файлови системи. В конкретния проект той служи като основа, от която се наследява Ext2.
    - **Основни отговорности:** Декларира чисто виртуални функции (pure virtual functions) за основните операции, които един драйвер на файлова система трябва да предоставя (напр. tree, cat, mkdir, rm, touch, write_file_op).
    - Съхранява обща информация като пътя до дисковия образ (m_device_path).
2. **Ext2 (конкретен клас):**
    - **Предназначение:** Наследява FileSystem и имплементира специфичната логика за работа с файловата система Ext2. Това е централният клас, който оркестрира всички операции.
    - **Основни отговорности:**
        - Инициализация и зареждане на структурите на Ext2 от дисковия образ (load_ext2).
        - Имплементация на всички файлови операции, дефинирани в интерфейса FileSystem (напр. create_file, remove_file, read_file, write_file, resolve_path и т.н.).
        - Управление на основните компоненти на Ext2: суперблок, таблица на дескрипторите на блокови групи и таблица на индексните възли.
    - **Композиция:** Съдържа като членове обекти от класовете SuperBlock, BlockGroupDescriptorTable и InodeTable
3. **Block (абстрактен базов клас):**
    - **Предназначение:** Представлява абстракция на блок от данни на диска, който има определен размер (m_size) и офсет (m_offset) в дисковия образ.
    - **Основни отговорности:** Предоставя общ механизъм за четене (read) и запис (write) на съдържанието на блока от/във файла на дисковия образ. Конкретните данни, които се четат/записват, се определят от наследяващите класове чрез виртуалните методи get_fields_buffer_for_read() и get_fields_buffer_for_write() (принцип на Template Method Pattern).
4. **SuperBlock (наследява Block):**
    - **Предназначение:** Представя суперблока на Ext2 файловата система.
    - **Основни отговорности:** Съхранява полетата на суперблока (в структурата Fields), осигурява тяхното четене от и запис в дисковия образ. Предоставя методи за достъп до важна информация от суперблока
5. **BlockGroupDescriptor (наследява Block):**
    - **Предназначение:** Представя един дескриптор на блокова група.
    - **Основни отговорности:** Съхранява полетата на дескриптора (в структурата Fields), осигурява тяхното четене и запис.
6. **BlockGroupDescriptorTable (наследява Block, но по-скоро управлява колекция от BlockGroupDescriptor):**
    - **Предназначение:** Управлява таблицата с дескриптори на всички блокови групи.
    - **Основни отговорности:** Създава и съхранява масив от обекти

BlockGroupDescriptor. Осигурява четенето на цялата таблица от дисковия образ и записването на отделни модифицирани дескриптори. Предоставя достъп до конкретен BlockGroupDescriptor по неговия индекс.

7. **Inode (наследява Block):**
    - **Предназначение:** Представя един индексен възел (inode) на Ext2.
    - **Основни отговорности:** Съхранява полетата на inode-а (в структурата Fields), включително режим, собственици, размер, времена и указателите към блокове с данни. Осигурява четене и запис на inode-а. Съдържа методи за инициализация на нов inode (init).
8. **InodeTable (наследява Block, но управлява колекция от Inode):**
    - **Предназначение:** Управлява достъпа до таблиците с индексни възли. Тъй като таблицата на inode-ите може да е голяма и е разпределена между блоковите групи, този клас имплементира механизъм за зареждане "при поискване".
    - **Основни отговорности:** Зарежда inode таблицата само за конкретна блокова група, когато е необходим достъп до inode от тази група (load_group). Съхранява кеширана таблицата на последно заредената група. Предоставя достъп до конкретен Inode обект по неговия номер.
9. **Структури за данни (utils::string, utils::vector<T>):**
    - **utils::string:** Динамичен символен низ.
    - **utils::vector<T>:** Шаблонен клас за динамичен масив.
10. **ClientInterface:**
    - **Предназначение:** Осигурява потребителския интерфейс на приложението.
    - **Основни отговорности:** Парсва командите, въведени от потребителя, и извиква съответните методи на обекта Ext2. Управлява основния цикъл на приложението.
11. **Помощни структури и пространства от имена:**
    - **DirectoryEntry.h:** Дефинира структурата LinkedDirectoryEntry за представяне на директорийни записи.
    - **utils:: :** Съдържа помощни функции за работа с битови полета (is_bit_set, find_first_free_bit и др.), функции за разделяне на низове (split_path, split_str) и други.

Глава 4. Реализация, тестване

4.1. Реализация на класове

- **Block:** Методите read/write използват std::ifstream/std::fstream за дискови операции, като конкретните данни се предоставят от наследниците чрез get_fields_buffer_for_read/write().
- **InodeTable:** Методът get_inode управлява зареждането "при поискване" на inode таблицата за съответната блокова група чрез load_group, което намалява използваната памет. load_group инициализира всеки Inode в кеша с правилния му дисков офсет и размер.
- **Ext2:**
    - resolve_path: Итеративно обхожда компонентите на пътя, четейки директорийни записи.
    - create_file/remove_file: Комплексни операции, координиращи заделяне/освобождаване на inode-и и блокове, работа с битови полета и актуализация на всички свързани метаданни чрез commit_file или commit_deallocated_file.
    - read_file/write_file: Използват рекурсивни помощни функции (process_..._blocks, process_indirect_block_writes_recursive) за работа с директни и индиректни блокове. write_file прилага стратегия "освободи-всичко-и-запиши-наново".

4.2. Управление на паметта и алгоритми. Оптимизации.

- **Управление на паметта:**
    - utils::string и utils::vector<T> управляват динамично паметта си чрез new[]/delete[], с механизми за преоразмеряване.
    - Временни буфери за дискови I/O операции (uint8_t*) се заделят и освобождават коректно.
    - Буферите за битови полета, връщани от allocate_inode/allocate_block, се управляват от извикващия код.
- **Ключови алгоритми:**
    - **Разрешаване на път:** Линейно обхождане на компоненти.
    - **Заделяне/Освобождаване на ресурс:** Линейно търсене в битови полета, актуализация на броячи в BGD/SB.
    - **Работа с индиректни блокове:** Рекурсивно обхождане за четене, запис и освобождаване.
- **Оптимизации:**
    - Кеширане на inode група в InodeTable.
    - Опростена стратегия за локалност при заделяне на ресурси.

4.3. Планиране, описание и създаване на тестови сценарии

Тестването е извършено ръчно чрез CLI. Основни сценарии:

1. **Работа с директории/файлове:** mkdir, touch в директория, tree, rm -r.
2. **Операции с файлове:** touch, write (презаписване), cat, append, cat отново, rm.
3. **Работа с предварително създадени файлове:** Четене и модификация на файловете от makefile-a.

4. **Гранични случаи:** Опити за създаване на съществуващи елементи, изтриване на несъществуващи, работа с невалидни пътища. Проверката на състоянието се извършва основно чрез tree и cat.

Глава 5. Заключение

5.1. Обобщение на изпълнението на началните цели

Основната цел на проекта – придобиване на по-дълбоко практическо разбиране на вътрешните структури и принципите на работа на файловата система Ext2. Покрити са следните основни задачи:

- **Проучени и разбрани са основните структури на Ext2**
- **Проектирана и реализирана е обектно-ориентирана архитектура**
- **Имплементирани са основните операции за работа с дисковия образ**
- **Реализирани са ключови команди за управление на файлове и директории**
- **Разработен е функционален потребителски интерфейс (CLI)**
- **Създадени са помощни класове за основни структури от данни**
- **Управлението на паметта и ресурсите**

Проектът успешно симулира значителна част от поведението на реален Ext2 драйвер.

5.2. Насоки за бъдещо развитие и усъвършенстване

- **Имплементиране на пълна поддръжка за права за достъп**
- **Поддръжка на символни и твърди връзки**
- **По-сложни оптимизации при заделяне на блокове**
- **Фрагментация и дефрагментация**
- **Журналиране (Ext3 функционалност)**
- **Разширяване на набора от команди в CLI**
- **По-детайлно тестване**
- **Графичен потребителски интерфейс (GUI)**
- **Форматиране на празен дисков образ**

Използвана литература

Следните източници бяха използвани за справка по време на разработката на проекта и подготовката на тази документация:

1. "Ext2." *Wikipedia, The Free Encyclopedia*, Wikimedia Foundation, 1 май 2024 г., en.wikipedia.org/wiki/Ext2. Достъпен на 3 юни 2025 г.
2. Card, Remy, Theodore Ts'o, and Stephen Tweedie. "2. Ext2fs On-Disk Data Structures." *Design and Implementation of the Second Extended Filesystem*, Free Software Foundation, Inc., последна промяна 21 дек. 1999 г., www.nongnu.org/ext2-doc/ext2.html. Достъпен на 3 юни 2025 г.