

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень та термінів.....	3
Вступ.....	4
1 Аналіз вимог програмного забезпечення.....	5
1.1 Загальні положення.....	5
1.2 Аналіз успішних ІТ-проектів.....	7
1.2.1 FUSE (Filesystem in Userspace).....	8
1.2.2 ZFS (Zettabyte File System).....	9
1.2.3 NTFS (New Technology File System).....	9
1.2.4 ext4 (Fourth Extended Filesystem).....	10
1.2.5 APFS (Apple File System).....	10
1.3 Порівняння існуючих програмних аналогів.....	11
1.4 Актуальність розробки власного програмного засобу.....	13
1.5 Аналіз вимог до програмного забезпечення.....	13
1.5.1 Функціональні вимоги.....	14
1.5.2 Трасування вимог.....	17
1.5.3 Нефункціональні вимоги.....	18
1.5.4 Постановка задачі.....	19
1.6 Висновки до розділу.....	19
2 Моделювання та конструювання програмного забезпечення.....	21
2.1 Моделювання та аналіз програмного забезпечення.....	21
2.2 Архітектура програмного забезпечення.....	24
2.3 Конструювання програмного забезпечення.....	25
2.3.1 Багатопотоковість, “Adapter” паттерн.....	25
2.3.2 Опис структур даних.....	29
2.3.2.1 Класи-моделі, “Composite” паттерн, std::variant.....	29
2.3.2.2 Класи-контролери, “Visitor” паттерн.....	33
2.3.2.3 Класи-представлення.....	45
2.4 Висновки до розділу.....	46

3 Аналіз якості та тестування програмного забезпечення.....	48
3.1 Аналіз якості пз.....	48
3.2 Опис процесів тестування.....	49
3.2.1 Функціональні вимоги.....	49
3.2.2 Нефункціональні вимоги.....	56
3.3 Опис контрольного прикладу.....	58
3.4 Висновки до розділу.....	60
4 Впровадження та супровід програмного забезпечення.....	62
4.1 Розгортання програмного забезпечення.....	62
4.2 Висновки до розділу.....	63
Висновки.....	64
Перелік посилань.....	66
Додаток А.....	75
Додаток Б.....	86
Додаток В.....	144
Додаток Г.....	150
Додаток Д.....	162
Додаток Е.....	163

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

ВФС — віртуальна файлова система.

FUSE — File System In User Space.

ОС — операційна система.

ВСТУП

Віртуальні файлові системи (ВФС) в сучасному інформаційному середовищі є ключовим елементом, що дозволяє ефективно та гнучко управляти доступом до даних та забезпечувати їхню цілісність. Вони становлять абстракцію над фізичними носіями даних та забезпечують єдиною точкою доступу до різноманітних джерел інформації. ВФС використовуються в різних областях, починаючи від операційних систем і закінчуючи розподіленими обчисленнями та хмарними сервісами.

З урахуванням швидкого розвитку сучасних технологій та зростання обсягів даних, виникає необхідність вдосконалення існуючих ВФС або розробки нових, які відповідають сучасним вимогам ефективності, безпеки та гнучкості. Актуальність написання нової файлової системи полягає у здатності відповісти на виклики, пов'язані з розширеним обсягом даних, розподіленими обчисленнями, вимогами до конфіденційності та високою швидкістю обробки інформації.

У даній курсовій роботі буде розглянуто концепцію віртуальних файлових систем, проведений аналіз існуючих рішень, та обґрунтована необхідність розробки нової файлової системи, яка відповідає вимогам сучасності та враховує актуальні тенденції в області обробки та збереження даних.

1 АНАЛІЗ ВИМОГ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У сучасному світі програмне забезпечення відіграє ключову роль у вирішенні різноманітних завдань та задач, спрямованих на оптимізацію робочих процесів, покращення взаємодії з користувачами та забезпечення безпеки та надійності. Процес розробки програмного забезпечення розпочинається з аналізу вимог, який визначає основні потреби та очікування від майбутнього продукту.

Аналіз вимог є етапом, на якому визначаються функціональні та нефункціональні вимоги до програмного забезпечення, що дозволяє визначити його ключові характеристики та особливості. Вірний та детальний аналіз вимог є вирішальною складовою у процесі розробки програм, оскільки від цього етапу залежить успішне втілення та функціонування програмного продукту.

У даній курсовій роботі висвітлено процес аналізу вимог до програмного забезпечення. Розглядаються загальні положення щодо визначення вимог, їхньої класифікації та впливу на подальший процес розробки. Детально розглядається аналіз успішних аналогів, а також порівняння їхніх характеристик та особливостей, що служить основою для формулювання вимог до нового програмного продукту.

1.1 Загальні положення

У даному розділі будуть розглянуті проблеми, які існують у площині створення віртуальних файлових систем та існуючі методи їх вирішення.

Віртуальна файлова система (ВФС) — це програмна або апаратна конструкція, яка надає інтерфейс для роботи з файловою системою. Це може бути шар абстракції між програмним забезпеченням та фізичними носіями даних, який дозволяє звертатися до файлів і папок, незалежно від їхнього розташування чи форматування зберігання.

ВФС може забезпечувати деякі додаткові функції, такі як шифрування, контроль доступу, кешування, компресія, відображення віддалених ресурсів тощо. Вона дозволяє програмам взаємодіяти з файловою системою, не враховуючи конкретні деталі роботи фізичних пристроїв чи мережевих ресурсів.

Основні переваги віртуальних файлових систем:

- абстракція від апаратних рішень: ВФС надають абстракцію від конкретних характеристик апаратного забезпечення, що дозволяє програмам працювати з файлами незалежно від фізичних пристроїв чи їхнього форматування; це спрощує взаємодію з даними та дозволяє легше переносити програми між різними системами;
- управління різноманітністю джерел даних: віртуальні файлові системи можуть об'єднувати дані з різних джерел, таких як локальні диски, мережеві пристрої, хмарні сховища тощо, створюючи зручний інтерфейс доступу до різноманітних ресурсів;
- додаткові функціональні можливості: ВФС можуть надавати додаткові функціональні можливості, такі як шифрування, контроль доступу, кешування, компресія та інші; це дозволяє розширювати можливості роботи з даними та забезпечувати додатковий рівень безпеки;
- підтримка віддалених ресурсів: ВФС можуть дозволяти доступ до віддалених ресурсів через мережу, що важливо в сучасному світі, де робота з віддаленими серверами та хмарними сховищами стає все більш поширеною;
- зручність розробки та тестування: розробка і тестування програм може бути спрощеною завдяки використанню ВФС; вони дозволяють емулювати різні сценарії взаємодії з файловою системою без прив'язки до конкретного обладнання чи мережевого середовища.

Віртуальна файлова система перехоплює системні виклики, пов'язані з операціями файлової системи. Ці виклики генеруються операційною системою або програмами, що намагаються взаємодіяти з файловою системою. ВФС взаємодіє з ядром операційної системи для обробки системних викликів і отримання доступу до ресурсів. Вона реєструється у ядрі як обробник файлових операцій та може взаємодіяти з іншими компонентами операційної системи. Розглянемо схему роботи на прикладі рисунку 2.1 за електронним джерелом [1].

Реалізація віртуальної файлової системи може зустрічати ряд проблем, з якими розробники повинні враховувати при створенні інтерфейсу. Деякі з основних проблем включають:

- продуктивність: віртуальні файлові системи можуть впливати на продуктивність, оскільки вони додають додатковий шар абстракції; ефективність роботи з файлами і папками повинна бути належним чином оптимізована;
- безпека: забезпечення безпеки даних і запобігання можливим атакам на віртуальну файлову систему є важливим завданням; це включає управління доступом, шифрування інформації та інші заходи безпеки;
- сумісність: важливо враховувати сумісність віртуальної файлової системи з різними операційними системами і програмами; розробка таких систем повинна враховувати різні стандарти та протоколи;
- відновлення та відміна змін: якщо стається помилка чи відмова в роботі, важливо мати ефективні механізми відновлення та відміни змін для запобігання втраті даних або пошкодженню файлової системи;
- масштабованість: при роботі з великою кількістю файлів та об'ємом даних, важливо забезпечити ефективну масштабованість віртуальної файлової системи.

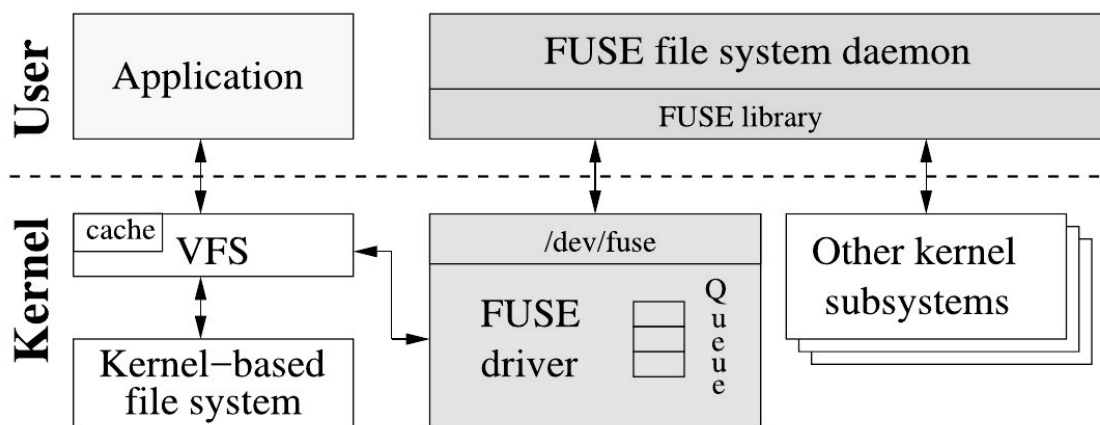


Рисунок 1.1 Схема роботи файлової системи

1.2 Аналіз успішних ІТ-проектів

У цьому розділі ми ретельно аналізуємо успішні ІТ-проекти, зосереджуючись на віртуальних файлових системах. Розглядаючи їх переваги та недоліки, ми визначимо ключові аспекти, які можуть визначати ефективні рішення в сучасному ІТ-середовищі. Це вивчення стане важливою основою для подальшого розгляду

власної віртуальної файлової системи.

Успішність віртуальних файлових систем визначається за кількома ключовими критеріями. Перш за все, це продуктивність, виміряна швидкістю та ефективністю роботи під високими навантаженнями. Надійність вказує на стійкість та можливість відновлення даних у разі виникнення збоїв. Масштабованість визначає здатність системи працювати ефективно при збільшенні обсягу даних. Безпека включає захист від несанкціонованого доступу та забезпечення конфіденційності. Інші важливі аспекти включають сумісність з іншими системами, здатність до інновацій та користувацький досвід, а також вартість власності, яка визначає витрати на утримання та розвиток файлової системи. Всі ці критерії враховуються для визначення та оцінки успішності віртуальних файлових систем у сучасному ІТ-середовищі.

1.2.1 FUSE (Filesystem in Userspace)

Проект розповсюджується як відкрите програмне забезпечення і не пов'язаний із конкретною компанією.

Основні функціональні можливості:

- FUSE [2] надає інтерфейс для створення віртуальних файлових систем у просторі користувача;
- дозволяє розробникам створювати власні файлові системи, не модифікуючи ядро операційної системи.

Переваги програмної системи:

- гнучкість: можливість реалізації різноманітних файлових систем без необхідності звертатися до ядра операційної системи;
- незалежність від ядра: робота в просторі користувача, що дозволяє уникнути проблем, пов'язаних з модифікацією ядра ОС.

Недоліки програмної системи:

- затримки: операції через простір користувача можуть призводити до затримок у виконанні;
- специфічність: реалізація віртуальних файлових систем від FUSE може вимагати додаткових зусиль для оптимізації.

1.2.2 ZFS (Zettabyte File System)

Компанія-виробник: Oracle Corporation [3] (раніше Sun Microsystems).

Основні функціональні можливості:

- ZFS [4] —розподілена файлова система, яка об'єднує файлову систему та об'єктне сховище даних;
- підтримка атомарних операцій, автоматичне виявлення та виправлення помилок, забезпечення високої надійності та ефективності.

Переваги програмної системи:

- висока ефективність: забезпечує швидкодію операцій завдяки кешуванню та інтелектуальному розподілу даних;
- надійність: автоматичне виявлення та виправлення помилок, захист від втрати даних.

Недоліки програмної системи:

- вимоги до обладнання: високі вимоги до обладнання можуть бути складні для задоволення на старіших або менш потужних системах;
- складність налаштувань: деяка складність у налаштуванні та конфігурації, що може вимагати досвіду.

1.2.3 NTFS (New Technology File System)

Компанія-виробник: Microsoft Corporation.

Основні функціональні можливості:

- NTFS [5] є файловою системою, розробленою для операційних систем сімейства Windows;
- підтримує розширені функції, такі як контроль доступу, журналювання та квоти.

Переваги програмної системи:

- безпека: забезпечує високий рівень безпеки через контроль доступу та аудит файлових операцій;
- журналювання: журнальна структура допомагає відновленню даних в разі помилок або аварій.

Недоліки програмної системи:

- портабельність: оскільки NTFS створена для платформи Windows, вона може бути менш сумісною з іншими операційними системами;
- обмеження функціональності: деякі функції можуть бути обмежені або не підтримуватися на інших операційних системах.

1.2.4 ext4 (Fourth Extended Filesystem)

Компанія-виробник: розробницьке співтовариство Linux.

Основні функціональні можливості:

- ext4 [6] є файловою системою для операційних систем сімейства Linux та є покращеною версією ext3;
- підтримує розширений розмір файлів та директорій, журналювання та відновлення файлової системи.

Переваги програмної системи:

- висока швидкодія: забезпечує високу продуктивність у великих файлах та директоріях;
- журналювання: журналований підхід допомагає відновленню даних після аварій.

Недоліки програмної системи:

- фрагментація: може виникати фрагментація файлової системи, особливо при роботі з великими обсягами даних;
- обмежені функції: у порівнянні з деякими іншими файловими системами, може бути менше розширених функцій.

1.2.5 APFS (Apple File System)

Компанія-виробник: Apple Inc.

Основні функціональні можливості:

- APFS [7] є файловою системою, розробленою для операційних систем Apple;
- підтримує розширений шифрування, снапшоти, оптимізацію для SSD та інші сучасні технології.

Переваги програмної системи:

- швидкодія на SSD: оптимізація для роботи на твердотільних накопичувачах,

забезпечуючи високу швидкодію;

- шифрування: вбудована підтримка шифрування для забезпечення конфіденційності даних.

Недоліки програмної системи:

- сумісність з іншими ОС: обмежена сумісність з операційними системами, що не є продуктами Apple;
- неідентифіковані недоліки: у деяких випадках може виникати несподівана поведінка, оскільки APFS є релятивно новою технологією.

1.3 Порівняння існуючих програмних аналогів

Розглянувши наявні успішні IT-проекти, складемо таблицю порівня.

Таблиця 1.1 Порівняння існуючих програмних аналогів

Критерії / Файлові Системи	FUSE	ZFS	NTFS	ext4	APFS
Назва продукту	Filesystem in Userspace	Zettabyte File System	New Technology File System	Fourth Extended Filesystem	Apple File System
Компанія- виробник	Проект розповсюдж ується як відкрите програмне забезпеченн я	Oracle Corporation (раніше Sun Microsystem s)	Microsoft Corporation	Розробниць ке співтоварис тво Linux	Apple Inc.
Основні функціонал ьні можливості	Створення віртуальних файлових систем у	Розподілена файлова система, журналюва	Розширений контроль доступу, журналюва	Розширений розмір файлів та директорій,	Оптимізація для SSD, розширене шифруванн

	просторі користувача	ння, автоматичн е виявлення та виправленн я помилок	ння, квоти	журналюва ння, відновлення	я
Переваги	Гнучкість в створенні різноманітн их файлових систем	Висока ефективніст ь, автоматичн е виявлення та виправленн я помилок	Безпека, журналюва ння	Висока швидкодія, журналюва ння	Швидкодія на SSD, вбудоване шифруванн я
Недоліки	Затримки через простір користувача	—Високі вимоги до обладнання, складність налаштуван ь	— Специфічні сть для платформи Windows	— Фрагментац ія, обмежені функції	Несумісніст ь з попереднім и версіями MacOS та IOS

Після розгляду успішних аналогів важливо відзначити, що багато з них характеризуються закритим кодом чи вимагають від розробників власної реалізації операцій, що часто може обмежувати гнучкість та розширюваність.

У цьому контексті власно-розроблена файлова система, заснована на FUSE бібліотеці, виділяється серед аналогів, пропонуючи низку вагомих переваг. Вона не лише гнучка завдяки стандартному інтерфейсу FUSE, але й має відкритий код, що сприяє активній участі розробників і прискорює виявлення та виправлення помилок. Додатково, забезпечення конфіденційності, висока швидкодія, цілісність даних у багатопотоковому середовищі, сумісність з Unix-подібними системами та

можливість розширення функціональності за допомогою кастомізації операцій роблять її важливим рішенням для сучасних вимог до файлових систем.

1.4 Актуальність розробки власного програмного засобу

Розробка власної віртуальної файлової системи (ВФС), реалізованої на основі FUSE, може бути актуальною з ряду причин:

- гнучкість та розширюваність: FUSE дозволяє створювати віртуальні файлові системи в просторі користувача без необхідності зміни ядра операційної системи; це забезпечує гнучкість та розширюваність у розробці, дозволяючи легко адаптувати ВФС до конкретних потреб проекту;
- сумісність з різними ОС: віртуальні файлові системи, розроблені за допомогою FUSE, можуть бути легко переносимі між різними операційними системами, оскільки FUSE підтримується на багатьох платформах (Linux, macOS, FreeBSD, інші); це робить розробку більш універсальною та ефективною;
- розвиток функціональності: розробка на основі FUSE дозволяє вам створювати власні файлові системи з розширеними функціональними можливостями, які можуть відповідати конкретним потребам користувачів чи проекту; це особливо корисно у випадках, коли існуючі ВФС не влаштовують за вимогами;
- експериментація та навчання: розробка ВФС на основі FUSE може слугувати відмінною можливістю для навчання та експериментації в області системного програмування; розуміння принципів роботи файлових систем та їх вплив на взаємодію з операційною системою може бути корисним для розробників.

1.5 Аналіз вимог до програмного забезпечення

Програмна система, що розробляється, спрямована на створення віртуальної файлової системи (ВФС) на основі FUSE, яка надає зручний та гнучкий інтерфейс для взаємодії з файловою структурою. Основною метою цього проекту є створення інструменту, який дозволяє користувачам легко маніпулювати віртуальною файловою структурою, використовуючи консольний інтерфейс.

Функціональні завдання та мета програмної системи сфокусовані на створенні віртуальної файлової системи (ВФС) на основі FUSE з метою надання користувачам зручного інтерфейсу для взаємодії з файловою структурою. Головною метою є забезпечення функціональних можливостей для створення та видалення об'єктів, отримання детальної інформації про атрибути файлів, читання та запису вмісту файлів, роботи з символьними посиланнями, а також навігації та взаємодії з директоріями. Це включає в себе можливість створення нових файлів та директорій, видалення об'єктів, отримання детальної інформації про атрибути файлів, читання та редагування їх вмісту, а також роботу з символьними посиланнями. Подальшою метою є створення зручного та функціонального інструменту для користувачів, який може бути використаний для проведення експериментів, навчання або вирішення конкретних завдань, забезпечуючи при цьому гнучкість та ефективність взаємодії з файловою системою на основі FUSE через консольний інтерфейс. Діграму використання можна побачити на рисунку 1.2:

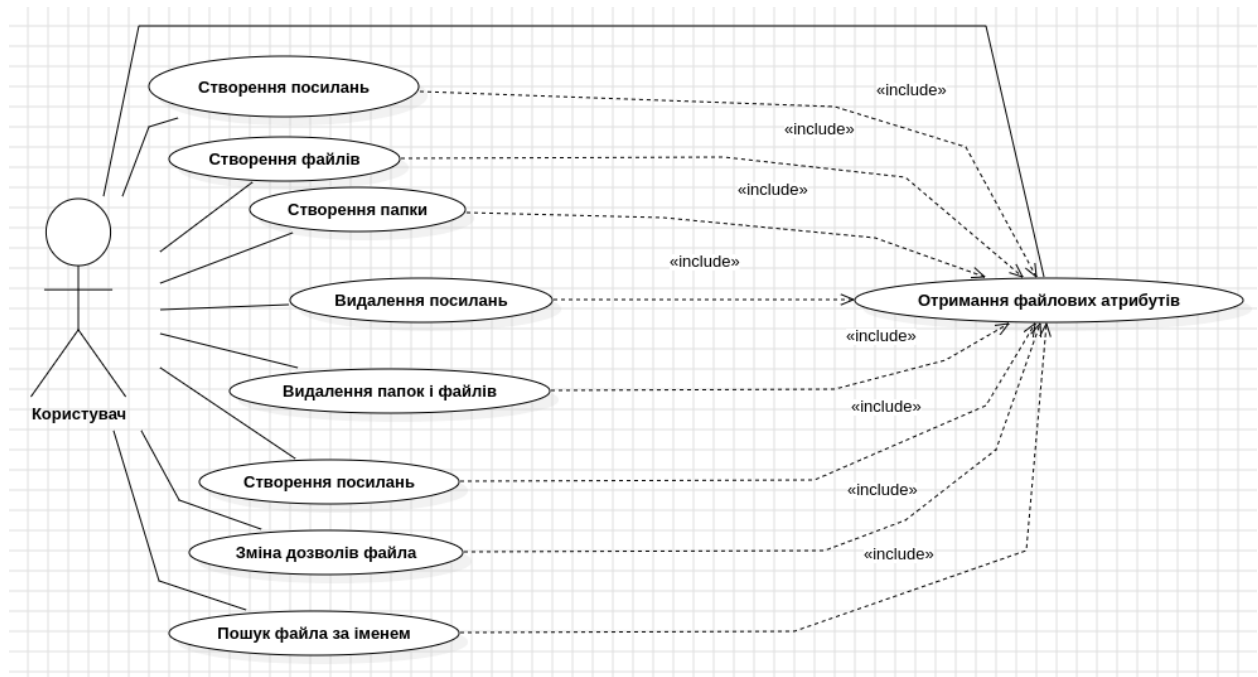


Рисунок 1.2 Діаграма використання

У таблицях, наведених у додатку А, можна переглянути варіанти використання файлової системи.

1.5.1 Функціональні вимоги

Функціональні вимоги — це конкретні функції чи сервіси, які система чи

програмне забезпечення повинні надавати для вирішення конкретних завдань та вимог користувачів. Ці вимоги описують очікувану функціональність та здатність системи виконувати певні операції. Функціональні вимоги зазвичай формулюються як конкретні задачі чи дії, які користувачі системи повинні мати можливість виконати.

Таблиця 1.2 Функціональні вимоги до до додатку

Номер	Назва	Опис
FR-1	Взяття файлових атрибутів (getattr) [8]	Система повинна надавати можливість користувачеві отримувати атрибути об'єктів (файлів та папок) у віртуальній файловій системі. Функціональність включає отримання інформації про ім'я об'єкта, його розмір, права доступу. Отримані атрибути повинні бути виведені у консоль або доступні для подальшого використання в інших частинах системи.
FR-2	Зчитування посилань (readlink) [9]	Система повинна забезпечити можливість користувачеві зчитувати вміст символьного посилання та виводити його у консоль.
FR-3	Створення файла (mknod) [10]	Система повинна дозволяти користувачеві створювати новий файл у віртуальній файловій системі та виводити підтвердження про успішне створення у консоль.
FR-4	Створення папки (mkdir) [11]	Система повинна дозволяти користувачеві створювати нову директорію у віртуальній файловій системі та виводити підтвердження про успішне створення у консоль.
FR-5	Видалення посилань (unlink) [12]	Система повинна дозволяти користувачеві видаляти посилання на об'єкти у віртуальній файловій системі та виводити підтвердження про успішне видалення у консоль.

FR-6	Видалення папок, посилань, файлів (rm) [13], (rmdir) [14]	Система повинна дозволяти користувачеві видаляти об'єкти (папки, посилання, файли) у віртуальній файловій системі та виводити підтвердження про успішне видалення у консоль.
FR-7	Створення soft-посилань (symlink) [15]	Система повинна дозволяти користувачеві створювати символічні посилання на об'єкти у віртуальній файловій системі та виводити підтвердження про успішне створення у консоль.
FR-8	Зміна дозволів файла (chmod) [16]	Система повинна забезпечувати можливість користувачеві змінювати права доступу до файлів у віртуальній файловій системі та виводити підтвердження про успішну зміну у консоль.
FR-9	Зчитування файла (read) [17]	Система повинна дозволяти користувачеві зчитувати вміст файлів у віртуальній файловій системі та виводити його у консоль.
FR-10	Редагування файла (write) [18]	Система повинна дозволяти користувачеві змінювати вміст файлів у віртуальній файловій системі та виводити підтвердження про успішне редагування у консоль.
FR-11	Зчитування папки (readdir) [19]	Система повинна дозволяти користувачеві зчитувати перелік об'єктів у директорії віртуальної файлової системи та виводити його у консоль.
FR-12	Пошук файла за іменем	Система повинна надавати можливість користувачеві здійснювати пошук файлу віртуальної файлової системи за його іменем. Функціональність включає можливість введення користувачем імені шуканого файлу, обробку цього запиту системою, виконання пошуку відповідного файлу та виведення інформації про

		нього у консоль.
--	--	------------------

Таблиця функціональних можливостей включає основні операції файлової системи та додаткову операцію швидкого пошуку.

1.5.2 Трасування вимог

Таблиця трасування вимог [20] є інструментом системного аналізу та управління проектами, який використовується для відстеження взаємозв'язків між різними елементами проекту. Основна мета цієї таблиці полягає в тому, щоб забезпечити зв'язок між вихідними вимогами до системи і конкретними елементами, які реалізують ці вимоги.

У контексті розробки програмного забезпечення, таблиця трасування вимог зазвичай включає в себе список вимог або функціональних можливостей, а також інші елементи, такі як сценарії використання (use-case'и), тестові випробування, компоненти системи тощо. Кожен елемент таблиці поєднується з конкретними вимогами, які він впроваджує чи тестує.

Таблиця 1.3 Матриця трасування вимог

UC\ FR	1	2	3	4	5	6	7	8	9	10	11	12
1	X											
2		X										
3			X									
4				X								
5					X							
6						X						
7							X					
8								X				
9									X			
10										X		

11											X	
12												X

Важливо відзначити, що таблиця трасування вимог для нашої файлової системи має спрощену структуру, оскільки кожен use-case повністю покривається однією функціональною вимогою.

1.5.3 Нефункціональні вимоги

Нефункціональні можливості вимоги — це характеристики системи чи програмного забезпечення, які не стосуються конкретної функціональності, але визначають якісні аспекти їхньої роботи та характеристики. Ці можливості визначають "якість" системи та включають такі аспекти, як продуктивність, надійність, безпека та інші.

Таблиця 1.4 Нефункціональні вимоги

Номер	Назва	Опис
NFR-1	Продуктивність	Система повинна забезпечувати відповідний рівень продуктивності для операцій пошуку файлів за іменем. Час відповіді системи не повинен перевищувати 1 секунду для більшості запитів.
NFR-2	Надійність	Система повинна бути стійкою до помилок та забезпечувати надійність при роботі з операціями. При виникненні помилок, система повинна надавати зрозумілі та інформативні повідомлення користувачеві.
NFR-3	Сумісність	Система повинна бути сумісною з існуючими програмами та UNIX операційними системами, що використовуються користувачами. Зокрема, результати пошуку мають бути виведені у форматі, який легко інтегрується з іншими

		програмами.
NFR-4	Безпека	Система повинна гарантувати конфіденційність інформації та захищати від несанкціонованого доступу під час операцій.
NFR-6	Зручність інтерфейсу	Користуватський інтерфейс системи повинен бути інтуїтивно зрозумілим та зручним для використання. Результати пошуку мають бути представлені чітко та лаконічно, а користувач повинен мати можливість взаємодіяти з ними без зайвих ускладнень.

Наведені в таблиці вимоги допомагають враховувати ключові аспекти, які забезпечують не тільки функціональну повноту, але й задовольняють якість програмного продукту.

1.5.4 Постановка задачі

Розробка може бути застосована під час розробки інших видів програмного забезпечення та у повсякденній взаємодії з файлами.

Цільова аудиторія для віртуальної файлової системи на основі FUSE включає розробників програмного забезпечення, системних адміністраторів, та інших ІТ-професіоналів, які використовують та інтегрують файлові системи у своїх проектах. Також, система має бути пристосована для кінцевих користувачів, які використовують віртуальну файлову систему для зручного управління та взаємодії з файлами.

Також програмне забезпечення має виконувати всі поставлені функціональні та нефункціональні вимоги.

1.6 Висновки до розділу

У даному відділі визначено формулювання задачі, розглянуто наявні аналоги файлових систем. Представлені альтернативи вважаються достатньо якісними, але вони мають вади, такі як закритий код або вимагають від користувача самостійно

реалізувати основні операції файлової системи.

Крім того, в першому розділі проведений аналіз вимог, побудовано таблицю для відповідного сценарію використання та кожної функціональної вимоги. Це дозволило отримати більш детальне уявлення про вимоги до мови програмування та технологій, які повинні бути використані.

У якості мови програмування було обрано C++[21], оскільки вона є достатньо гнучкою та надає можливість будувати високі абстракції над низькорівневим кодом.

У якості допоміжного засобу було обрано бібліотеку FUSE (Filesystem in User Space), оскільки вона вже реалізує спілкування із ядром операційної системи і надає інтерфейс для реалізації операцій файлової системи.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У сучасному інформаційному суспільстві програмне забезпечення виступає важливою складовою для вирішення складних завдань та досягнення стратегічних цілей в різних галузях. Процес розробки програмного забезпечення є складним та многозадачним, вимагаючи глибокого розуміння вимог користувачів, ефективних методів моделювання та конструювання.

Моделювання та конструювання програмного забезпечення становлять ключові етапи у життєвому циклі розробки програм, де визначаються концепції, архітектура та алгоритми, які лягають в основу майбутнього продукту. Ефективне моделювання дозволяє визначити оптимальні рішення та забезпечити гнучкість системи, а конструювання є процесом втілення цих концепцій у функціональний та надійний програмний продукт.

У даній курсовій роботі детально розглядається процес моделювання та конструювання програмного забезпечення. Аналізуються основні підходи до створення моделей, методи та інструменти, які використовуються у цьому процесі. Покладаючи акцент на важливість етапів моделювання та конструювання, робота пропонує висвітлити основні концепції та вирішення, що визначають успішну розробку програмного забезпечення в сучасному інформаційному середовищі.

2.1 Моделювання та аналіз програмного забезпечення

Використаємо BPMN (Business Process Model and Notation) [22] діаграму для опису бізнес-процесу. BPMN — це стандарт для моделювання бізнес-процесів, який надає уніфікований мовний засіб для спільного розуміння, аналізу та оптимізації бізнес-процесів всередині організації. Використовуючи символи та правила, BPMN дозволяє створювати графічні представлення процесів, що полегшує співпрацю між бізнес-аналітиками та розробниками програмного забезпечення.

Використання Бізнес-процесної мережі (BPNM) дозволяє досягти стандартизації в моделюванні бізнес-процесів. Цей міжнародний стандарт надає єдиний набір термінів та концепцій, що сприяє узгодженій роботі різних команд та організацій.

Використання BPMN спрощує керування проектами, надаючи можливість створювати, редагувати та вдосконалювати бізнес-процеси в єдиному середовищі. Це сприяє ефективній співпраці команд та управління змінами в організації.

Отримання зворотнього відгуку від ВФС є одним цільним бізнес-процесом, тому на рисунку 2.1 опишемо його за допомогою BPMN.

Опис зворотнього відгуку від ВФС:

- Запит на операції з файлами починається з виклику зовнішньої програми, яка звертається до файлової системи ядра операційної системи.
- Файлова система ядра ОС обробляє отриманий запит. Якщо шлях до файлу вказаний існуючою файловою системою, ядро файлова система передає управління цій файлової системі. У випадку, якщо шлях не знайдений або вказана файлова система не існує, ядро файлова система спробує самостійно обробити запит.
- Якщо файлова система ядра ОС визнає, що запит відноситься до віртуальної файлової системи, вона перенаправляє цей запит до віртуальної файлової системи для подальшої обробки.
- Віртуальна файлова система отримує запит і обробляє його відповідно до свого функціоналу. Це може включати доступ до реальних файлових ресурсів, взаємодію з іншими компонентами ядра операційної системи або виконання додаткових операцій, передбачених віртуальною файловою системою.
- Після обробки запиту віртуальна файлова система надсилає відповідь файлової системі ядра, яка в свою чергу передає цю відповідь зовнішній програмі. Зовнішня програма отримує результат операції, і обробка запиту завершується.

Моделювання віртуальної файлової системи використовуючи BPMN дозволяє чітко визначити її етапи та взаємодію компонентів. Це полегшує аналіз та оптимізацію, а також сприяє ефективній співпраці між учасниками розробки. Модель допомагає виявити можливі ризики та вдосконалити дизайн системи, що призводить до створення надійних та продуктивних рішень.

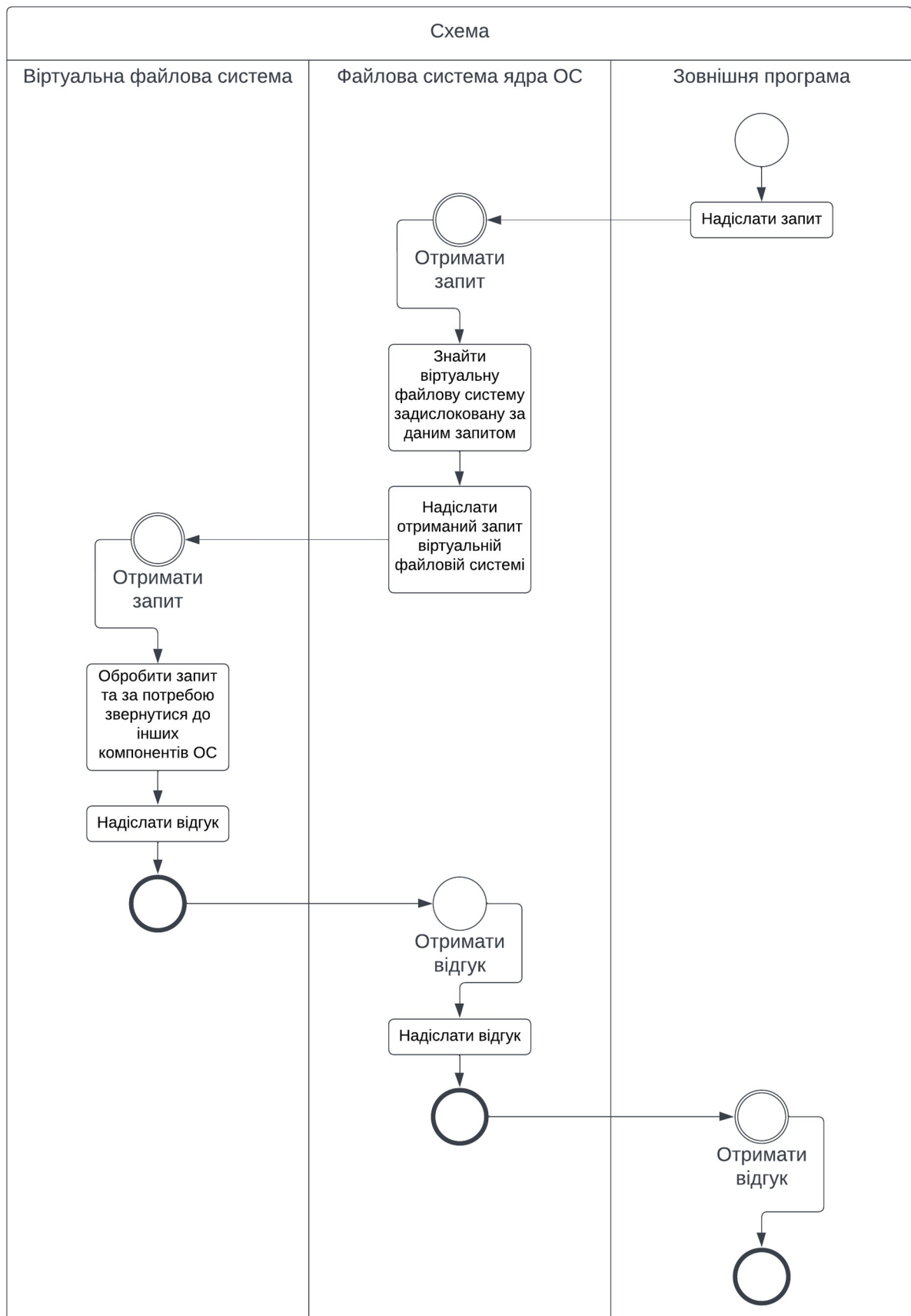


Рисунок 2.1 Схема бізнес-процесу отримання відгуку ВФС

2.2 Архітектура програмного забезпечення

В розробці віртуальної файлової системи, використання паттернів проектування стає важливим етапом для забезпечення ефективності, читабельності та легкості управління кодом. Один із таких паттернів - Model-View-Controller (MVC) [23] - дозволяє відокремити представлення, логіку та дані, роблячи архітектуру більш модульною та гнучкою.

Модель у віртуальній файловій системі представляє основні концепції, такі як звичайні файли, директорії та soft-посилання. Кожен з цих елементів відповідає реальній структурі файлової системи та забезпечує базовий функціонал для взаємодії з користувачем.

Представлення вирішує, як інформація буде відображатися для користувача. У цьому випадку, консольний інтерфейс, реалізований за допомогою C++ бібліотеки CLI11 [24], служить інструментом взаємодії. Він призначений для зручного введення команд та відображення результатів операцій.

Контролер виконує роль посередника між користувачем та моделлю. Він приймає команди від користувача через консоль та визначає, як ці команди повинні бути оброблені. Після цього він взаємодіє з моделлю для виконання необхідних операцій.

Для кращого розуміння архітектури побудуємо UML-діаграму компонентів. UML (Unified Modeling Language) [25] - це стандартний мовний інструментарій для моделювання об'єктно-орієнтованих систем. UML-діаграма компонентів використовується для візуалізації, спрощення та розуміння взаємодії компонентів програмної системи. Компонент в UML може представляти фізичний або логічний модуль програми.

Розглянемо схему архітектури:

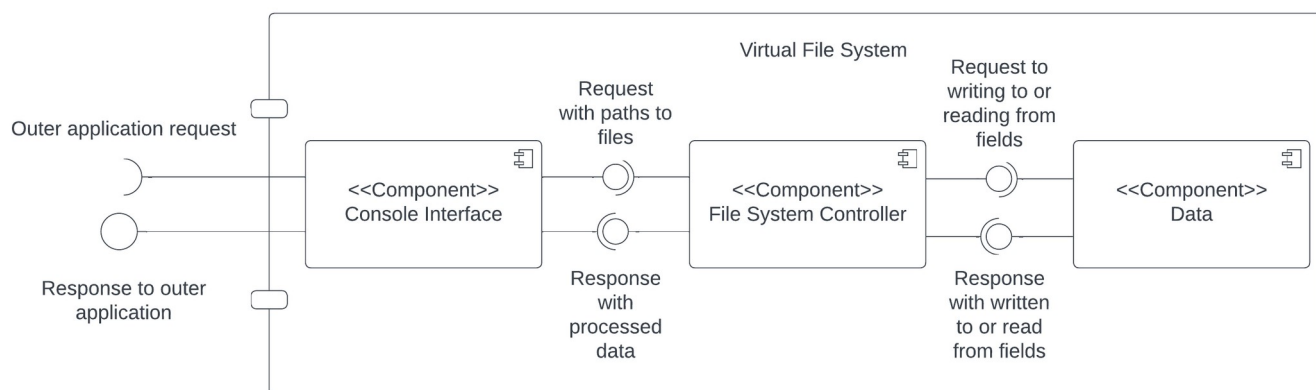


Рисунок 2.2 Діаграма компонентів ВФС

Архітектура, побудована за MVC паттерном, дозволяє зберігати код чистим, легко розширювати та модифікувати. Розподіл обов'язків між моделлю, представленням та контролером полегшує роботу над проектом, забезпечуючи оптимальну організацію та легкість управління.

2.3 Конструювання програмного забезпечення

Конструювання програмного забезпечення віртуальної файлової системи є критичним етапом у розробці, оскільки від цього залежить ефективність, безпека та функціональність продукту. У даному розділі ми детально розглянемо процес створення віртуальної файлової системи на основі FUSE (Filesystem in Userspace) та розглянемо ключові аспекти конструювання.

Конструювання віртуальної файлової системи має велике значення для досягнення успіху проекту. Цей розділ допоможе розробникам та інженерам краще зрозуміти етапи реалізації віртуальної файлової системи, враховуючи сучасні технології та підходи.

2.3.1 Багатопотоковість, “Adapter” паттерн

Одним з ключових аспектів при конструюванні віртуальної файлової системи є забезпечення ефективної багатопотоковості. Оскільки віртуальна файлова система може бути одночасно доступною для читання та запису з різних джерел, необхідно забезпечити конкурентний та безпечний доступ до даних.

Для цього застосуємо “Adapter” паттерн [26]. Основна ідея "Адаптер" полягає

в тому, щоб створити клас-посередник (адаптер), який конвертує інтерфейс одного класу у інтерфейс іншого класу. Це дозволяє об'єктам з різними інтерфейсами взаємодіяти один з одним без прямого залучення.

У зв'язку з цим, адаптер дозволяє:

- забезпечити сумісність інтерфейсів класів;
- дозволяти об'єктам працювати разом, навіть якщо їхні інтерфейси не сумісні напрямку.

Тобто потрібно побудувати такий клас, який є адаптером для забезпечення блокування читання-запису навколо об'єктів ВФС. Це дозволяє зручно взаємодіяти з об'єктами, надаючи їм властивості мутексів.

Як наслідок була розроблена міні-бібліотека `RwLock` (Read-Write Lock) [27]. Ця бібліотека містить шаблонний клас `TRwLock<T>`, що призначений для забезпечення конкурентного доступу до об'єктів типу `T`.

Таблиця 2.1 Методи класу `TRwLock<T>`

Назва методу	Сигнатура методу	Опис методу
Read	<code>TReadGuard<T> Read() const</code>	Забезпечує багатопотокове зчитування даних, надаючи <code>TRwLockReadGuard</code> , який автоматично відімкнеться при завершенні області дії. Дозволяє багатьом потокам одночасно читати дані без блокування один одного.
Write	<code>WriteGuard<T> Write()</code>	Забезпечує однопотоковий запис даних, надаючи <code>TRwLockWriteGuard</code> , який автоматично відімкнеться при завершенні області дії. Гарантує, що операції запису виконуються безпечно та без конфліктів в однопотоковому режимі.
TryRead	<code>std::optional<TRwLockTryRead</code>	Спроба багаточитального зчитування

	Guard<T>> TryRead() const	даних без блокування. Повертає TRwLockTryReadGuard, якщо операція успішна, або std::nullopt, якщо ресурс вже захоплений для запису.
TryWrite	std::optional<TRwLockTryWriteGuard<T>> TryWrite()	Спроба однопотокowego запису даних без блокування. Повертає TRwLockTryWriteGuard, якщо операція успішна, або std::nullopt, якщо ресурс вже захоплений для читання чи запису.

TrwLockGuardBase<T> є шаблоном базовим класом для усіх guard'ів. Кожен дочірній клас відповідає за спеціалізію деструктора, при виклику якого автоматично відпускається захоплений м'ютекс, що гарантує коректну роботу у багатопотоковому середовищі та уникнення блокування.

Таблиця 2.2 Методи класу TrwLockGuardBase

Назва методу	Сигнатура методу	Опис методу
GetPtr	Const T* GetPtr() const	Повертає константний покажчик на дані, що захоплені м'ютексом.
GetPtr	T* GetPtr()	Повертає мутабельний покажчик на дані, що захоплені м'ютексом.
operator->	Const T* operator->() const	Повертає константний покажчик на дані, що захоплені м'ютексом.
operator->	T* operator->()	Повертає мутабельний покажчик на дані, що захоплені м'ютексом.
operator*	Const T& operator*() const	Повертає константне посилання на дані, що захоплені м'ютексом.
operator*	T& operator*()	Повертає мутабельне посилання на дані, що захоплені м'ютексом.

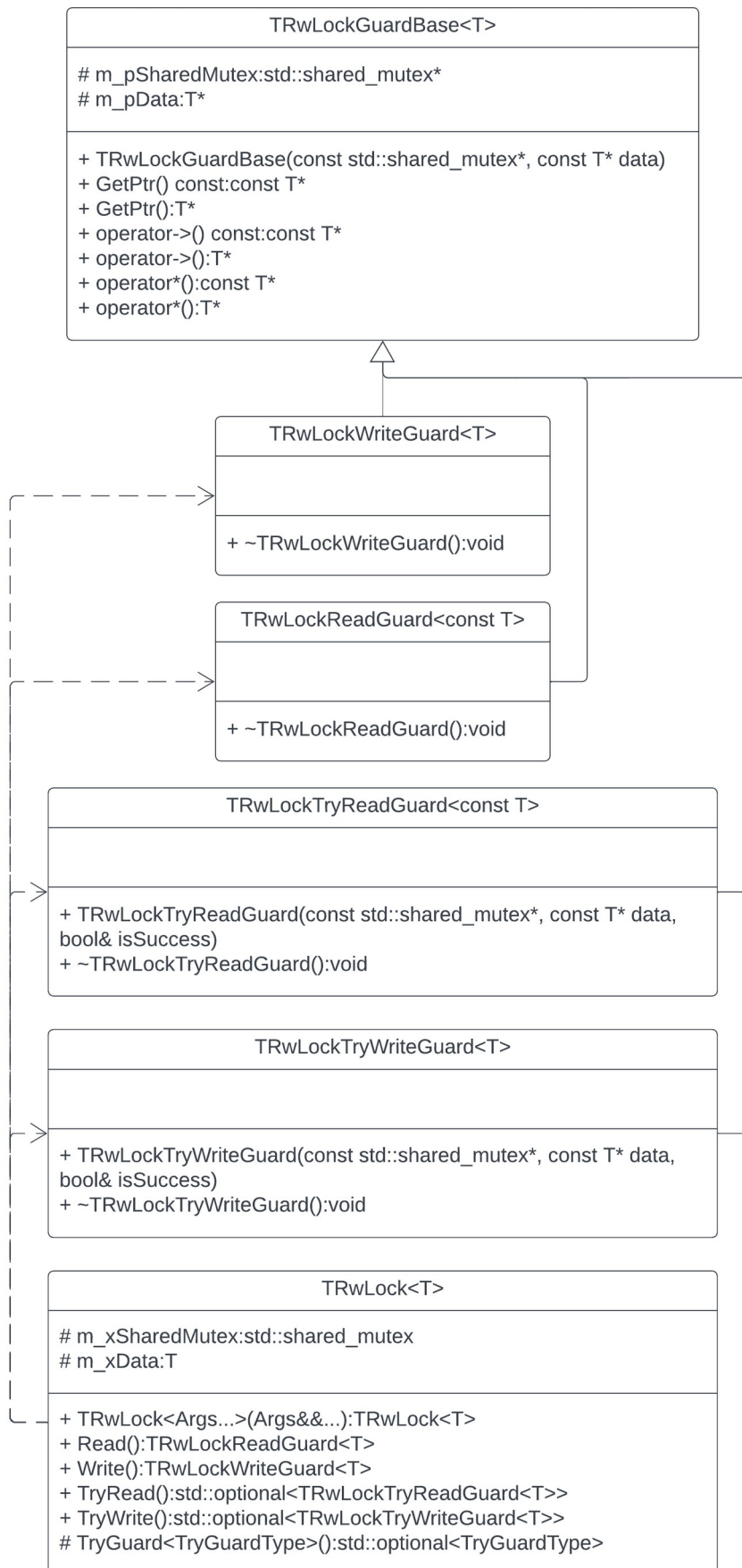


Рисунок 2.3 Діаграма класів `RwLock`

2.3.2 Опис структур даних

В даному розділі буде проведено детальний аналіз структури даних, яка застосовується у віртуальній файловій системі. У фокусі уваги - проєктовані класи та їх взаємодія, методи, що забезпечують роботу з файловою системою. Також буде розглянуто використання ключових паттернів проєктування, які покращують структуру та забезпечують гнучкість системи.

Детальний огляд реалізації класів і їхніх методів дозволить виявити сильні та слабкі сторони структури даних, аналізувати оптимальність вибраних рішень та вирішувати проблеми, які можуть виникнути при реалізації віртуальної файлової системи.

2.3.2.1 Класи-моделі, “Composite” паттерн, std::variant

Розглянемо звичайний Composite паттерн [28]. Цей шаблон проєктування відноситься до структурних патернів і дозволяє клієнтам обробляти окремі об'єкти та їхні композиції (групи об'єктів) однаковою чином. У цьому шаблоні створюється ієрархія класів, що представляють як окремі об'єкти, так і їхні композиції, де обидва типи об'єктів реалізують спільний інтерфейс. Таким чином, клієнт може взаємодіяти з кожним об'єктом (примітивним або складним) через спільний інтерфейс, що спрощує обробку об'єктів в ієрархії.

Розглянемо UML-діаграму моделей файлової системи на прикладі віртуальної файлової системи на рисунку 2.4. Зліва можна побачити звичайний Composite, справа — Composite у поєднанні з TRwLock:

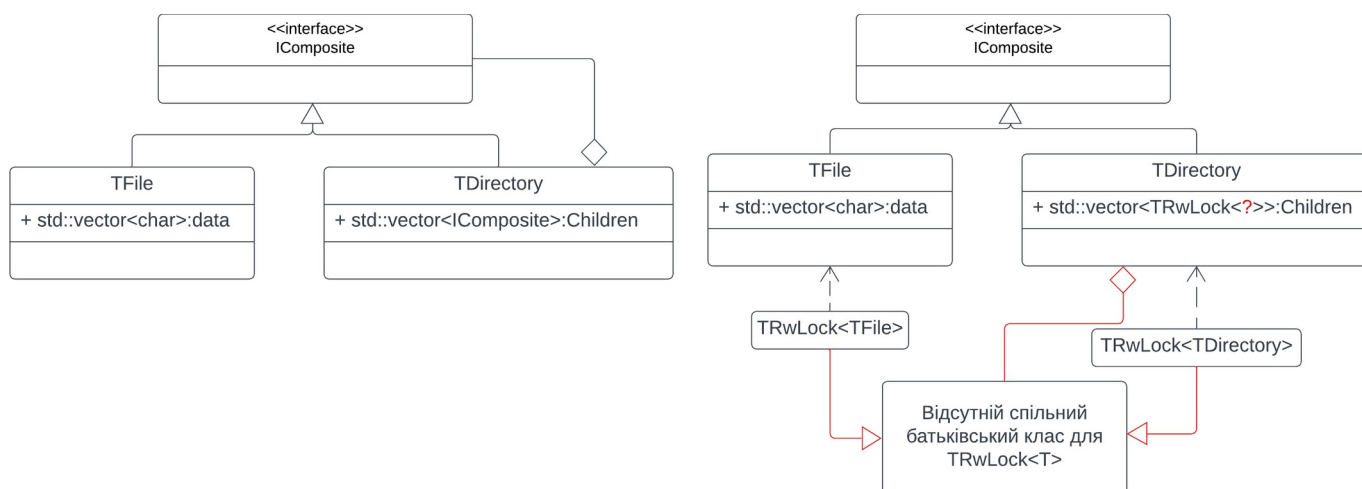


Рисунок 2.4 Composite шаблон на прикладі ВФС

Як бачимо, `TRwLock<TFile>` та `TRwLock<TDirectory>` не мають спільного батьківського класа, а тому не можуть бути покладені разом у контейнері.

Щоб вирішити проблему, зазначену у попередньому пункті, потрібно представити певний варіантивний тип, який одночасно може представляти класи, які не мають спільного походження. Для цього у C++17 був представлений `std::variant` [29].

`std::variant` — це шаблон класу у стандартній бібліотеці C++, який надає безпечний спосіб працювати з альтернативними типами даних (здебільшого використовується для об'єднань типів). Цей контейнер дозволяє представляти об'єкт одного з кількох можливих типів.

Основні характеристики `std::variant`:

- безпека типізації: компілятор гарантує, що тільки один із типів визначених у `std::variant` використовується в конкретний момент часу;
- обіймає варіанти: `std::variant` може об'єднувати типи, що визначають варіанти об'єкта;
- безпечні операції отримання значення: можна безпечно отримувати значення з `std::variant` за допомогою `std::get` або методу `std::visit`, що використовується для обробки всіх можливих типів;
- висока продуктивність: зазвичай `std::variant` ефективно реалізований і не вносить значних витрат на продуктивність.

З огляду на `std::variant`, складемо діаграму класів та наведемо опис до кожного з них.

Таблиця 2.3 Опис класів-моделей ВФС

Назва	Опис
<code>AWeakRWLock<T></code>	Синонім для <code>std::weak_ptr<TRwLock<T>></code> . Надає слабку (weak) власність до блокування читання-запису, пов'язаного з об'єктом типу <code>TRwLock<T></code> .
<code>ASharedRWLock<T></code>	Синонім для <code>std::shared_ptr<TRwLock<T>></code> . Надає спільну (shared) власність до блокування читання-запису, пов'язаного з об'єктом типу

	TRwLock<T>.
TFile<ParentType>	Шаблонний базовий клас, описуючи загальні атрибути кожного файлу у віртуальній файловій системі (ВФС). Параметр шаблону ParentType вказує на батьківський клас.
AWeakRwLock<TDirectory>	Спеціалізація для TDirectory. Синонім для спільної (shared) власності до блокування читання-запису, пов'язаного з TDirectory.
ASharedRwLock<TRegularFile>	Спеціалізація для TRegularFile. Синонім для спільної (shared) власності до блокування читання-запису, пов'язаного з TRegularFile.
ASharedRwLock<TLink>	Спеціалізація для TLink. Синонім для спільної (shared) власності до блокування читання-запису, пов'язаного з TLink.
ASharedFileVariant	Синонім для std::variant<ASharedRwLock<TDirectory>, ASharedRwLock<TRegularFile>, ASharedRwLock<TLink>>. Варіант для представлення об'єктів різних типів.
TRegularFile	Дочірній клас TFile<Tdirectory> для звичайних файлів. Спеціалізація для TRegularFile. Зберігає вектор байтів.
TLink	Дочірній клас TFile<Tdirectory> для soft-посилань. Спеціалізація для TLink.
TDirectory	Дочірній клас TFile<Tdirectory> для директорій. Спеціалізація для TDirectory. Містить в собі вектор ASharedFileVariant.

Такі статичні методи New для класів TRegularFile, TLink, та TDirectory були введені для зручності створення відповідних об'єктів, обгортаних у власність за допомогою ASharedRwLock<T>. Це дозволяє спростити процес конструювання об'єктів і забезпечити власність блокування читання-запису навколо них.

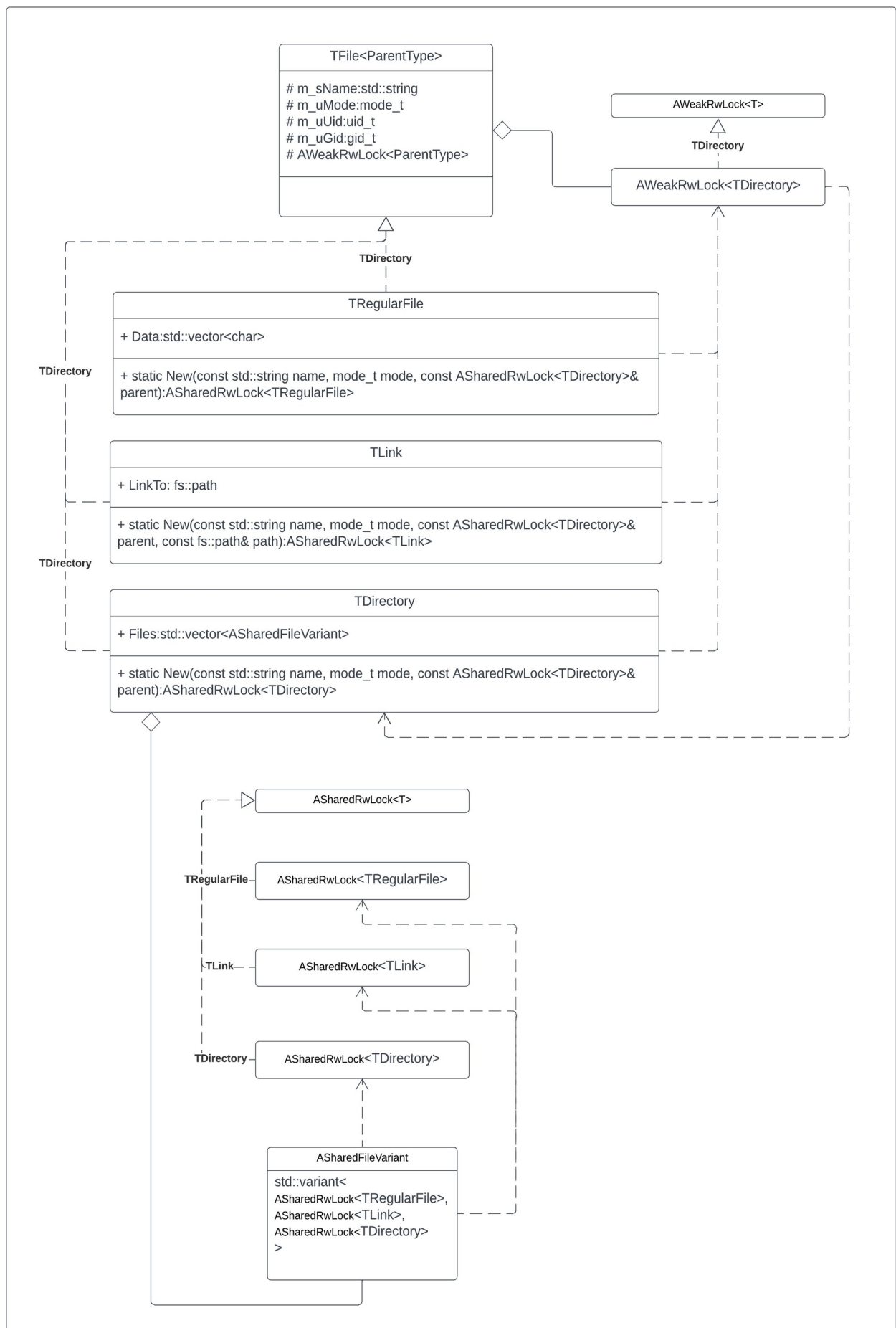


Рисунок 2.5 Діаграма класів моделей ВФС з використанням `std::variant`

2.3.2.2 Класи-контролери, “Visitor” паттерн

Visitor (відвідувач) [30] є поведінковим паттерном проєктування, який дозволяє визначити нову операцію без зміни класів об'єктів, над якими вона виконується.

У контексті віртуальної файлової системи, Visitor паттерн може бути корисним для виконання різноманітних операцій над різними типами файлів. Наприклад, операції, які можна виконувати над директорією, можуть бути відмінними від тих, що можна виконати над звичайним файлом чи символьним посиланням.

Переваги використання паттерна Visitor у ВФС:

- Розділення операцій та класів об'єктів: Забезпечує чітку структуру коду, розділяючи операції над об'єктами від їхньої структури.
- Легкість додавання нових операцій: Дозволяє додавати нові операції, не змінюючи класи об'єктів.
- Розширюваність системи: Дозволяє легко розширювати функціональність ВФС, додаючи нові класи Visitor та реалізації методів.

Мова C++ дозволяє набагато ефективніше імплементувати Visitor паттерн без наслідування всіх операцій від спільного інтерфейсу IVisit навідміну від C# чи Java. Для цього потрібно звернутися до лекції Клауса Іглбергера на CppCon 2022 [31], де він розповідає про використання `std::visit` [32] як основного інструмента для реалізації цього паттерна.

`std::visit` — це функція, яка дозволяє виконувати операції над об'єктами, які можуть мати різні типи в залежності від об'єкту-варіанта (`std::variant`). Вона використовується для реалізації статичного поліморфізму, що відбувається на етапі компіляції.

Розглянемо основні аспекти `std::visit`:

- динамічний вибір операції: функція `std::visit` дозволяє вибирати конкретну операцію для виконання залежно від типу об'єкта-варіанта;
- статичний поліморфізм: статичний поліморфізм вказує на те, що операції визначаються на етапі компіляції, а не викликаються віртуально під час виконання програми;

- використання `std::variant`: `std::visit` часто використовується для операцій над об'єктами типу `std::variant`, який представляє альтернативи (різні типи) об'єкта; за допомогою `std::visit` можна здійснювати дії над кожним з можливих типів;
- зручний спосіб реалізації відвідувача: `std::visit` використовується для реалізації відвідувача для обробки різних типів об'єктів, об'єднаних варіантом;
- інтеграція з іншими стандартними функціональними можливостями: `std::visit` добре інтегрується з іншими стандартними функціональними можливостями мови, такими як `lambda`-вирази.

Розглянувши методи реалізації Visitor паттерна, спроектуємо класи-контролери для ВФС. Для цього опишемо таблиці класів і методів.

Розглянемо ієрархію класів, що відповідають за вписування атрибутів файлів.

Таблиця 2.4 Класи для зміни атрибутів файлів

Назва	Опис
<code>TSetInfoParameterMixin<ParamType></code>	Змішуючий клас, який додає функціональність перевантаження оператора <code>()</code> дочірнім типам. Параметризується типом <code>ParamType</code> .
<code>TSetInfoParameterGeneralMixin<ParamType, DerivedType></code>	Дочірній клас, який вимагає від свого дочірнього типу почергового перевантаження оператора <code>()</code> .
<code>TSetInfoName</code>	Дочірній клас, який відповідає за запис імені файла. Параметризується типами <code>std::string</code> , <code>TSetInfoName</code>
<code>TSetInfoGid</code>	Дочірній клас, який відповідає за запис <code>group id</code> файла. Параметризується типами <code>gid_t</code> , <code>TSetInfoGid</code>
<code>TSetInfoUid</code>	Дочірній клас, який відповідає за запис <code>user id</code> файла. Дочірній клас, який відповідає за запис <code>group id</code> файла. Параметризується типами <code>uid_t</code> , <code>TsetInfoUid</code> .
<code>TSetInfoMode</code>	Дочірній клас, який відповідає за запис режимів доступу файла. Параметризується типами <code>mode_t</code> , <code>TSetInfoMode</code> .

TSetInfoParent	Дочірній клас, який відповідає за запис батька файла. Параметризується типом ASharedRwLock<Tdirectory>.
----------------	---

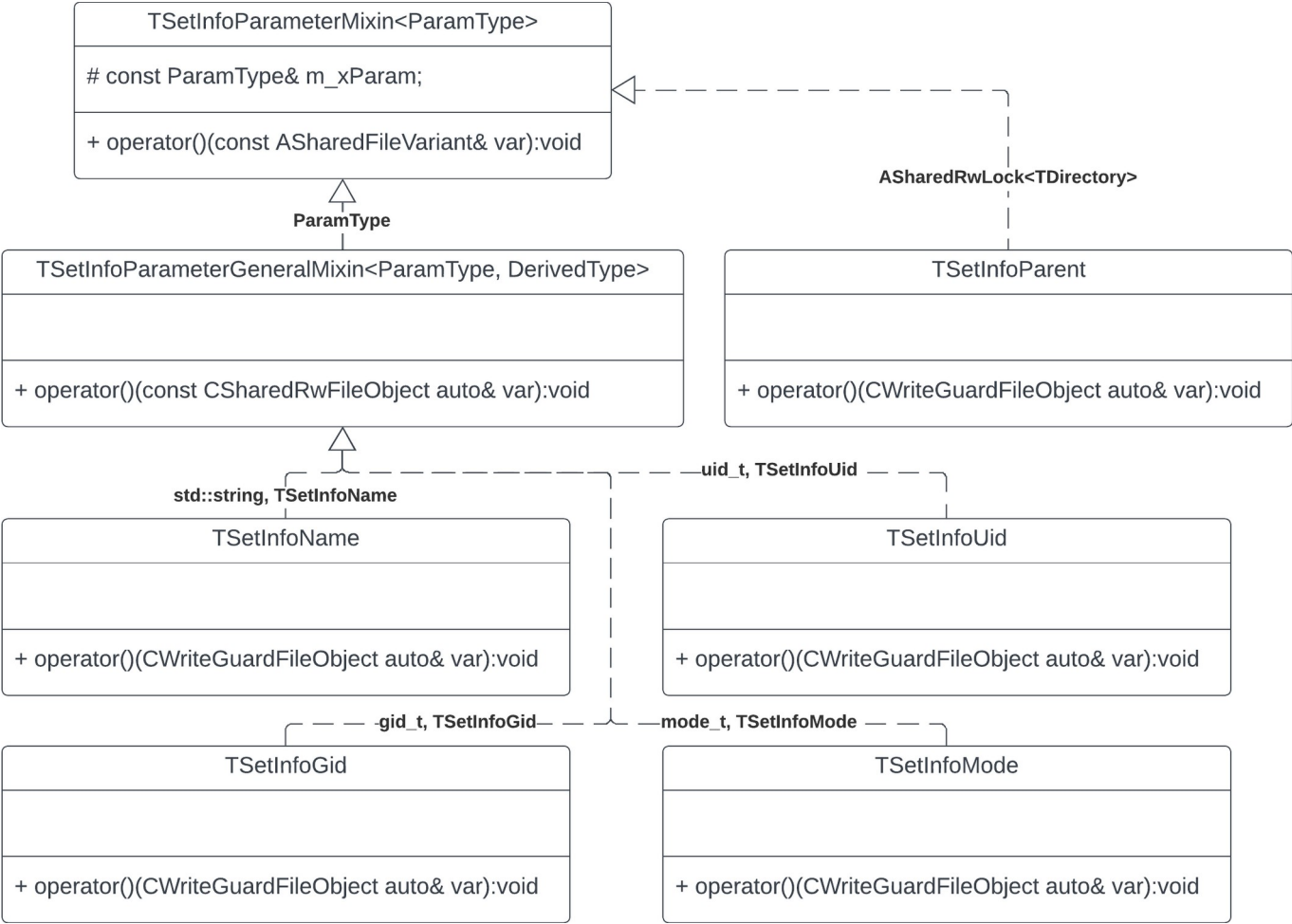


Рисунок 2.6 Діаграма класів для зміни атрибутів файлів

Розглянемо ієрархію класів, що відповідають за зчитування атрибутів файлів.

Таблиця 2.5 Класи для зчитування атрибутів файлів

Назва	Опис
TGetFileParameter<ParamType, Derived>	Змішуючий клас, який додає функціональність перевантаження оператора () дочірнім типам. Параметризується типом ParamType та дочірнім типом Derived.
TGetInfoName	Дочірній клас TGetFileParameter, який відповідає за зчитування імені файла. Параметризується типами std::string, TGetInfoName.

TGetInfoGid	Дочірній клас TGetFileParameter, який відповідає за зчитування group id файла. Параметризується типами gid_t, TGetInfoGid.
TGetInfoUid	Дочірній клас TGetFileParameter, який відповідає за зчитування user id файла. Параметризується типами uid_t, TGetInfoUid.
TGetInfoMode	Дочірній клас TGetFileParameter, який відповідає за зчитування режимів доступу файла. Дочірній клас TGetFileParameter, який відповідає за зчитування user id файла. Параметризується типами mode_t, TGetInfoMode.
TGetInfoParent	Дочірній клас TGetFileParameter, який відповідає за зчитування батька файла. Параметризується типами AsharedRwLock<Tdirectory>, TGetInfoParent.

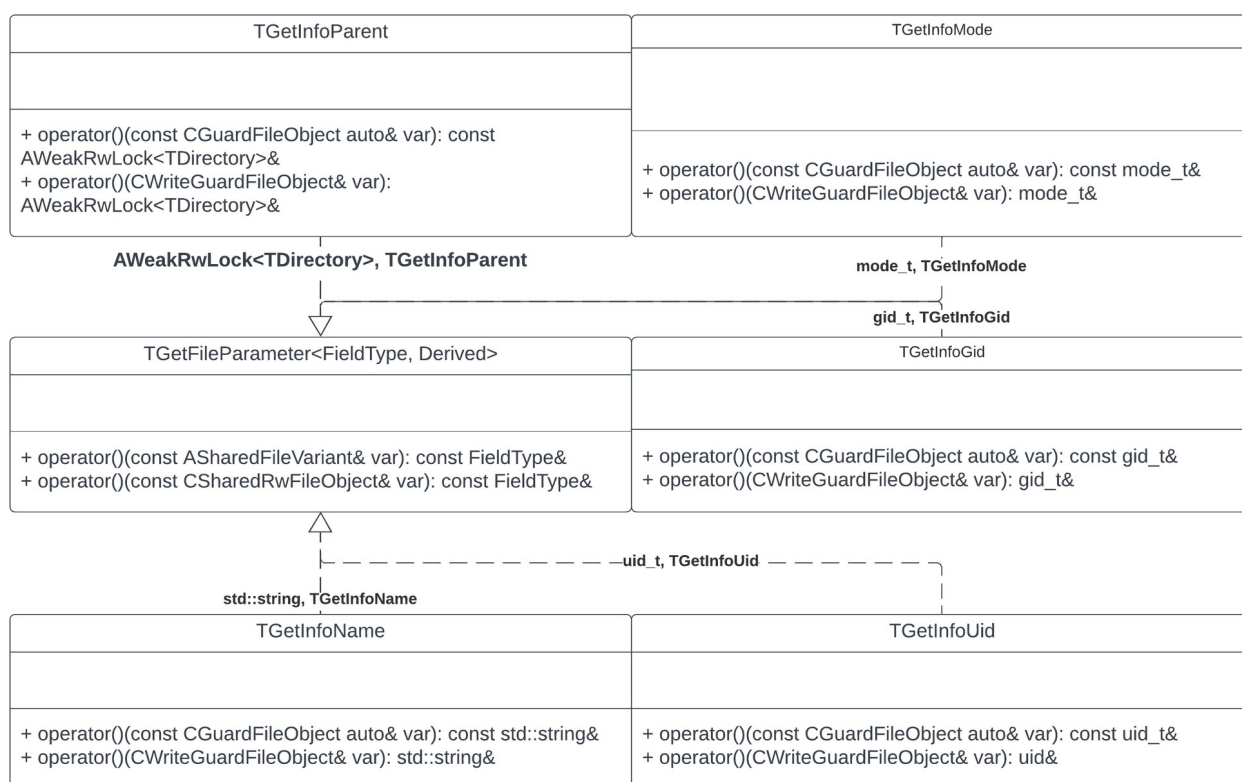


Рисунок 2.7 Діаграма класів для зчитування атрибутів файлів

Розглянемо клас TReadDirectory, що відповідає за зчитування директорії у віртуальній файловій системі.

Таблиця 2.6 Методи та поля класу TReadDirectory

Назва	Тип	Сигнатура	Опис
TReadDirectory	Конструктор	TReadDirectory(const fs::path& path, void* buffer, fuse_fill_dir_t filler)	Приймає шлях, буфер та функцію-заповнювач буфера.
Operator()	Публічний метод	void operator()()	Перевантажений оператор виклику функції, щоб мати можливість вважати об'єкт даного класу за функцію.
DoReadDir	Захищений метод	void DoReadDir(const ASharedRwLock<T Directory>& var)	Відповідає за зчитування директорії.
DoReadDir	Захищений метод	void DoReadDir(const ASharedRwLock<T Link>& var)	Намагається зчитати директорію за шляхом, що зберігає посилання. Якщо не знаходить, то кидає помилку TFSException.
DoReadDir	Захищений метод	void DoReadDir(const ASharedRwLock<T RegularFile>& var)	Існує для того, щоб викинути помилку, якщо варіант зберігає файл.
FillerBuffer	Захищений метод	void FillerBuffer(const std::string_view& name)	Заповнює певне ім'я файлу директорії.
FillerDirectory	Захищений метод	void FillerDirectory(const ASharedRwLock<T Directory>& dir)	Заповнює усі імена файлів, що знаходяться в директорії.
m_pPath	Захищене поле	const fs::path&	Зберігає шлях, за яким має

		m_pPath	бути директорія.
m_pBuffer	Захищене поле	void* m_pBuffer	Зберігає буфер, у який буде записано імена файлів директорії.
m_xFiller	Захищене поле	fuse_fill_dir_t m_xFiller	Показчик на функцію-заповнювач буфера, що надає бібліотека FUSE.

Потрібно звернути увагу, що через специфіку мови C++ простори імен(namespaces), які складаються лише з функцій, будуть вважатися класами, що мають лише публічні статичні методи.

Розглянемо простір імен NSFindFile, що відповідає за знаходження файла за іменем.

Таблиця 2.7 Функції простору імен NSFindFile

Назва	Сигнатура	Опис
Find	ASharedFileVariant Find(const fs::path& path)	Функція, що знаходить файл за заданим шляхом. Повертає варіативний тип файла. Викидає TFSException, якщо за заданим шляхом файл відсутній.
AddToNameHash	void AddToNameHash(const fs::path& path)	Функція, що додає шлях до хешу шляхів для швидкого знаходження файла за іменем.
RemoveFromNameHash	void RemoveFromNameHash(const fs::path& path)	Функція, що видаляє шлях з хеша шляхів.
FindByName	const std::set<fs::path>& FindByName(const std::string& name)	Функція, що знаходить хешовані шляхи за заданим ім'ям.
FindDir	ASharedRwLock<TDirector y> FindDir(const fs::path&	Функція, що знаходить директорію за заданим шляхом. Повертає

	path)	ASharedRwLock<TDirectory>. Викидає TFSException, якщо директорія відсутня.
FindLink	ASharedRwLock<TLink> FindLink(const fs::path& path)	Функція, що знаходить soft-посилання за заданим шляхом. Повертає ASharedRwLock<TLink>. Викидає TFSException, якщо soft-посилання відсутнє.
FindRegularFile	ASharedRwLock<TRegularFile> FindRegularFile(const fs::path& path)	Функція, що знаходить регулярний файл за заданим шляхом. Повертає ASharedRwLock<TRegularFile>. Викидає TFSException, якщо регулярний файл відсутній.

Розглянемо простір мен NSFileType, що визначає маску файла у UNIX.

Таблиця 2.8 Функції простору імен NSFileType

Назва	Сигнатура	Опис
Get (перевантаження для ASharedFileVariant)	constexpr NFileType Get(const ASharedFileVariant& var)	Функція, що визначає маску файла у UNIX з об'єкта типу ASharedFileVariant. Використовується для отримання типу файла, який зберігається у варіативному типі у віртуальній файловій системі.
Get (перевантаження для CSharedRwFileObject)	constexpr NFileType Get(const CSharedRwFileObject auto& var)	Функція, що визначає маску файла у UNIX з об'єкта, який відповідає концепту CSharedRwFileObject (зазвичай це спільний покажчик на об'єкт файлу). Використовується для отримання типу файла за

		допомогою концепту та шаблонного параметра.
Get (перевантаження для CGuardFileObject)	constexpr NFileType Get(const CGuardFileObject &var)	Функція, що визначає маску файлу у UNIX з об'єкта, який відповідає концепту CGuardFileObject (зазвичай це об'єкт-гвард для операцій з файлами). Використовується для отримання типу файлу за допомогою концепту та шаблонного параметра.

Маємо простір імен NSFileAttributes, який відповідає за зчитування атрибутів файлів у буфер, що наданий операційною системою. Має лише одну функцію Get, що приймає наданий буфер та заповнює його інформацією, про наданий файл.

Маємо простір імен NSDeleteFile, який має лише одну функцію Delete, що відповідає за видалення файлу з його директорії.

Розглянемо клас TFileSystem, що відповідає за реалізацію основних операцій ВФС [33].

Таблиця 2.9 Опис методів та полів класу TFileSystem

Назва	Тип	Сигнатура	Опис
Init	Публічний статичний метод	static int Init(int argc, char *argv[])	Статичний метод, що ініціалізує файлову систему з використанням аргументів командного рядка argc та argv.
GetAttr	Захищений статичний метод	static int GetAttr(const char* path, struct stat* st,	Захищений метод, що повертає атрибути файлу.

		struct fuse_file_info* fi)	
ReadLink	Захищений статичний метод	static int ReadLink(const char* path, char* buffer, size_t size)	Захищений метод, що зчитує зв'язану символьну ланку.
MkNod	Захищений статичний метод	static int MkNod(const char* path, mode_t mode, dev_t rdev)	Захищений метод, що створює файловий вузол.
MkDir	Захищений статичний метод	static int MkDir(const char* path, mode_t mode)	Захищений метод, що створює директорію.
Unlink	Захищений статичний метод	static int Unlink(const char* path)	Захищений метод, що видаляє файл.
Rmdir	Захищений статичний метод	static int Rmdir(const char* path)	Захищений метод, що видаляє директорію.
SymLink	Захищений статичний метод	static int SymLink(const char* target_path, const char* link_path)	Захищений метод, що створює символьне посилання.
ChMod	Захищений статичний метод	static int ChMod(const char* path, mode_t mode, struct fuse_file_info *fi)	Захищений метод, що змінює права доступу до файла.
Read	Захищений статичний метод	static int Read(const char* path, char* buffer, size_t size,	Захищений метод, що здійснює операцію читання з

		off_t offset, struct fuse_file_info *fi)	файла.
Write	Захищений статичний метод	static int Write(const char* path, const char* buffer, size_t size, off_t offset, struct fuse_file_info *info)	Захищений метод, що здійснює операцію запису у файл.
ReadDir	Захищений статичний метод	static int ReadDir(const char* path, void* buffer, fuse_fill_dir_t filler, off_t offset, struct fuse_file_info *fi, enum fuse_readdir_flags flags)	Захищений метод, що здійснює операцію читання директорії.
RootDir	Публічний статичний метод	static const ASharedRwLock<T Directory>& RootDir()	Публічний статичний метод, що повертає кореневу директорію.
FindByNameThread	Захищений статичний метод	static void FindByNameThread()	Захищений статичний метод, що викликається з іншого потоку та відповідає за знаходження файла за іменем.
s_pRootDir	Захищене статичне поле	static ASharedRwLock<T	Захищене статичне поле, яке

		Directory> s_pRootDir	представляє кореневу директорію.
FifoPath	Захищене статичне поле	static fs::path FifoPath	Захищене статичне поле, яке є шляхом до fifo-файлу, що відповідає за комунікацію між зовнішньою програмою та ВФС.

Клас `TFSException` визначає виняток для ситуацій, пов'язаних з операціями файлової системи, такими як спроба роботи з неіснуючим файлом чи директорією. Перевизначає віртуальний метод `what()` для повернення зрозумілого текстового повідомлення про виняток та надає метод `Type()` для отримання типу винятку. Конструктори приймають шлях або рядок (як `std::string_view`), який стосується проблемного об'єкта, та тип винятку.

Таблиця 2.10 Опис методів та полів класу `TFSException`

Назва	Тип	Сигнатура	Опис
<code>TFSException</code>	Конструктор	<code>TFSException(fs::path ::iterator begin, fs::path::iterator end, NFSExceptionType type)</code>	Конструктор класу <code>TFSException</code> , який приймає ітератори на початок та кінець шляху та тип винятку. Створює виняток для неіснуючого шляху в системі файлової системи.
<code>TFSException</code>	Конструктор	<code>TFSException(const fs::path& path, NFSExceptionType type)</code>	Конструктор класу <code>TFSException</code> , який приймає об'єкт <code>fs::path</code> та тип винятку. Створює

			виняток для неіснуючого шляху в системі файлової системи.
TFSException	Конструктор	TFSException(const std::string_view& path, NFSExceptionType type)	Конструктор класу TFSException, який приймає рядок (std::string_view) та тип винятку. Створює виняток для неіснуючого шляху в системі файлової системи.
what	Публічний віртуальний метод	virtual const char* what() const noexcept override	Перевизначений віртуальний метод what(). Повертає константний рядок, що містить повідомлення про виняток.
Type	Публічний метод	[[nodiscard]] NFSExceptionType Type() const	Невіртуальний метод, який повертає тип винятку (NFSExceptionType).
UpdateMessage	Захищений метод	void UpdateMessage(const std::string_view& path, NFSExceptionType type)	Захищений метод, який оновлює повідомлення про виняток на основі заданого шляху та типу винятку. Використовується при конструюванні винятку для неіснуючого шляху.
m_xType	Захищене поле	NFSExceptionType	Тип винятку (NFSExceptionType).
m_sMessage	Захищене поле	std::string	Рядок, що містить повідомлення про виняток.

2.3.2.3 Класи-представлення

Клас `TFileSystemClientCLI` є підкласом `CLI::App` і використовується для обробки командного рядка для клієнта файлової системи. Має конструктор для ініціалізації об'єкта класу та статичний метод `FindByName` для знаходження файлу за іменем через канал. Також містить захищений метод `Process`, який обробляє введені дані, та поля для зберігання шляху до каналу, імені файлу та повідомлення про помилку.

Таблиця 2.11 Методи та поля класу `TfileSystemClientCLI`

Назва	Тип	Сигнатура	Опис
<code>TFileSystemClientCLI</code>	Конструктор	<code>TFileSystemClientCLI()</code>	Конструктор класу <code>TFileSystemClientCLI</code> . Ініціалізує об'єкт класу.
<code>FindByName</code>	Статичний публічний метод	<code>template<unsigned long BufferSize> static void FindByName(const fs::path& pipePath, const std::string& fileName, std::array<char, BufferSize>& buffer)</code>	Статичний метод для знаходження файлу за іменем. Приймає шлях до каналу (<code>pipePath</code>), ім'я файлу (<code>fileName</code>) та буфер фіксованого розміру (<code>buffer</code>). Записує ім'я файлу в канал, а потім зчитує результат з каналу в буфер. Викидає виняток у випадку неможливості відкриття каналу.
<code>Process</code>	Захищений метод	<code>void Process() const</code>	Захищений метод, що обробляє введені дані. Реалізація відсутня в даному контексті.
<code>m_xPipePath</code>	Захищене поле	<code>fs::path</code>	Об'єкт класу <code>fs::path</code> , який представляє шлях

			до каналу.
m_sFileName	Захищене поле	std::string	Рядок, що містить ім'я файлу.
s_sError	Захищене поле	constexpr std::string_view	Константний рядок, що містить повідомлення про помилку "Can not open the pipe for writing". Використовується при виникненні винятку.

TFileSystemCLI - це клас, який представляє інтерфейс командного рядка для взаємодії з файловою системою. Конструктор класу ініціалізує об'єкт та встановлює параметри для роботи з бібліотекою CLI11. У конструкторі додаються різні параметри та прапорці для коректного використання файлової системи, такі як режим фонового процесу, відключення підтримки багатьох потоків та виведення налагоджувальних повідомлень.

2.4 Висновки до розділу

В розділі "Конструювання програмного забезпечення" віртуальної файлової системи (ВФС) було представлено процес проєктування та реалізації ключових компонентів системи. Основні аспекти включали архітектуру файлової системи, многопоточність, використання шаблонів проєктування, а також побудову класів-моделей, -контролерів, -представлень.

Зокрема, була розглянута архітектура системи, побудована за паттерном Model-View-Controller (MVC), де моделями є файли, директорії та посилання, контролером — файлова система, а представленням — консольний інтерфейс, реалізований за допомогою C++ бібліотеки CLI11. Ця архітектура дозволяє відокремлювати логіку, представлення та дані, що полегшує розширення та підтримку коду.

Також, було визначено та реалізовано необхідні класи для представлення файлів та директорій, їхніх атрибутів та забезпечено їх взаємодію за допомогою

шаблонів та роботи зі зв'язаними м'ютексами для багатопотоковості.

Додатково, була створена міні-бібліотека `RwLock` (Read-Write Lock), яка включає в себе реалізацію м'ютексів для забезпечення безпечного доступу до даних у багатопотоковому середовищі.

У розділі також висвітлено важливість розробки інтерфейсу користувача для взаємодії з ВФС. Для цього був реалізований консольний інтерфейс, який дозволяє користувачеві легко використовувати основні функції ВФС.

Отже, результатом цього розділу є створення основної структури та компонентів ВФС, які дозволяють ефективно управляти файлами та директоріями в багатопотоковому середовищі та надають зручний інтерфейс для користувача.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У розділі ми детально розглянемо якість та стабільність розробленого програмного продукту, а також проведемо процес тестування для впевненості в його надійності та ефективності.

3.1 Аналіз якості пз

Аналіз якості ПЗ було проведено з допомогою github-застосунку CCCC (C and C++ Code Counter) [34], який автоматично генерує звіт якості коду за різними метриками.

NOM (Number of modules) — це кількість нетривіальних модулів, ідентифікованих аналізатором. Нетривіальні модулі включають всі класи та будь-який інший модуль, для якого визначені функції-члени. Загалом кількість окремих модулів, для коду ідентифіковано 40. Ця метрика важлива для приблизної оцінки складності проекту, оскільки чим більший проект, тим більше файлів, тим складніше його підтримувати.

LOC (Lines of Code) [35] — кількість непорожніх рядків вихідного коду без коментарів, підрахованих аналізатором. Загалом кількість рядків коду оцінена в 1696 рядки.

COM (Lines of Comments) — кількість рядків коментарів, визначених аналізатором. Загалом кількість рядків коментарів оцінена в 4 рядки. Ця метрика може бути досить важливою для великої кодової бази, однак у даній курсовій роботі автор намагався писати самодокументований код, за яким особа, що читає може зрозуміти сенс написаного.

MVG (McCabe's Cyclomatic Complexity) [36] — міра складності рішення для функцій, які складають програму. Суворе визначення цієї міри полягає в тому, що це кількість лінійно незалежних маршрутів через спрямований ациклічний граф, який відображає потік керування підпрограмою. Аналізатор підраховує це, записуючи кількість різних результатів рішення, що містяться в кожній функції, що дає гарне наближення до формально визначеної версії міри. Загалом аналізатор показує 125.

Загальний звіт можна побачити на рисунку 3.1.

Metric	Tag	Overall
Number of modules	NOM	40
Lines of Code	LOC	1696
McCabe's Cyclomatic Number	MVG	125
Lines of Comment	COM	4

Рисунок 3.1 Звіт якості коду CCCC

3.2 Опис процесів тестування

Тестування віртуальної файлової системи (ВФС) було проведено з використанням фреймворку для тестування googletest[37]. Вибір googletest був обумовлений його визнаною ефективністю, гнучкістю та набором зручних інструментів для написання та виконання тестів.

Однією з основних переваг googletest є його зручний і легкий у використанні синтаксис для написання тестів. Тестові кейси та перевірки можуть бути легко створені і організовані за допомогою зрозумілих механізмів структури тестових класів та макросів, що полегшує процес написання і управління тестами.

Ще однією важливою перевагою є можливість паралельного виконання тестів. Це забезпечує швидший оборот часу при великій кількості тестових кейсів, що сприяє ефективності та розподілу ресурсів при тестуванні.

Розпочнемо аналіз якості програмного забезпечення з відповідності нефункціональним вимогам.

3.2.1 Функціональні вимоги

У даному пункті будуть проаналізовані усі функціональні вимоги, що були висунуті для ВФС, щоб визначити їхню повноту, конкретність, узгодженість з бізнес-потребами та здатність задовольняти очікувані вимоги користувачів.

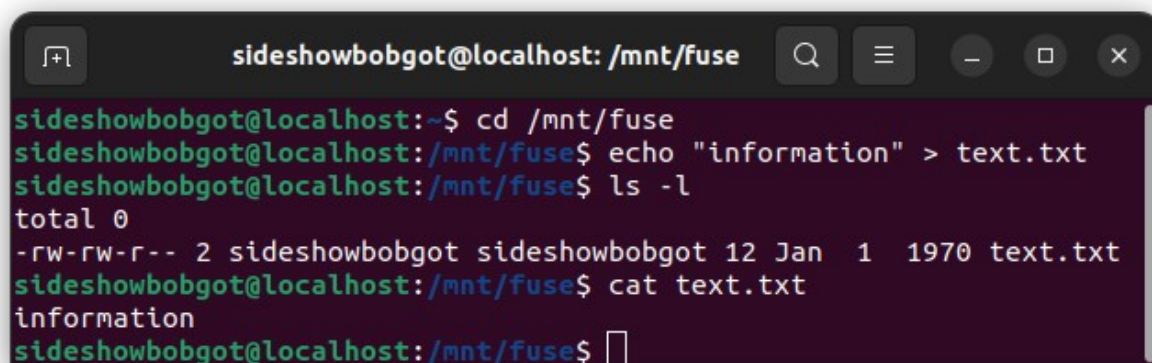
Перевіримо працездатність функцій write, read, mknod, getattr, open[38]. Процес показано в таблиці 3.1.

Таблиця 3.1 Тестування функцій mknod, write, read, open, getattr

Тест	Перевірка роботи з регульним файлом регулярний файл
------	---

Номер тесту	1
Початковий стан	Користувач знаходиться в консолі
Вхідні дані	Ім'я файлу text. txt, Текст “information”
Опис проведення тесту	Створити файл та вписати в нього інформацію з допомогою команди echo[39] та зчитати за допомогою команди cat[40], перевірити атрибути з допомогою команди ls[41].
Очікуваний результат	При зчитування файлу виведеться аналогічний текст, що був вписаний, при взятті атрибутів на місці позначки типу знаходиться “-”.
Фактичний результат	При зчитування файлу виводиться аналогічний текст, що був вписаний

Як можна побачити на рисунку 3.1, регулярний файл був успішно створений; перевірили, що рядок був вписаний; перевірили, що і справді тип файлу має префікс “-”.



```
sideshowbobgot@localhost: /mnt/fuse
sideshowbobgot@localhost:~$ cd /mnt/fuse
sideshowbobgot@localhost:/mnt/fuse$ echo "information" > text.txt
sideshowbobgot@localhost:/mnt/fuse$ ls -l
total 0
-rw-rw-r-- 2 sideshowbobgot sideshowbobgot 12 Jan  1  1970 text.txt
sideshowbobgot@localhost:/mnt/fuse$ cat text.txt
information
sideshowbobgot@localhost:/mnt/fuse$
```

Рисунок 3.2 Ручна перевірка працездатності функцій write, read, mknod, getattr, open

Перевіримо працездатність функцій symlink, readlink. Процес показано в таблиці 3.2.

Таблиця 3.2 Тестування функцій symlink, getattr, readlink

Тест	Перевірка роботи з soft-посиланням
Номер тесту	2
Початковий стан	Користувач знаходиться в консолі

Вхідні дані	Файл з ім'ям “linked”, ім'я посилання “link”
Опис проведення тесту	Створити soft-посилання на файл, перевірити атрибути з допомогою команди ls.
Очікуваний результат	При зчитуванні атрибутів файлів посилання буде мати префікс “l” та буде вивдена стрілочка на шлях до реального файла.
Фактичний результат	При зчитуванні атрибутів файлів посилання має префікс “l” та вивдена стрілочка на шлях до реального файла.

Як можна побачити на рисунку 3.3, тестовий файл успішно створений; створення посилання на файл коректно відпрацювало, і побачили, що і справі посилання показує шлях до файла.

```
sideshowbobgot@localhost: /mnt/fuse
sideshowbobgot@localhost:/mnt/fuse$ touch linked
touch: setting times of 'linked': Function not implemented
sideshowbobgot@localhost:/mnt/fuse$ ln -s linked link
sideshowbobgot@localhost:/mnt/fuse$ ls -l
total 0
lrwxrwxr-x 2 sideshowbobgot sideshowbobgot 6 Jan  1  1970 link -> linked
-rw-rw-r-- 2 sideshowbobgot sideshowbobgot 0 Jan  1  1970 linked
sideshowbobgot@localhost:/mnt/fuse$
```

Рисунок 3.3 Ручна перевірка працездатності функцій symlink, getattr, readlink

Перевіримо працездатність функцій mkdir, unlink, rm, rmdir, opendir[41], readdir[30]. Процес показано в таблиці 3.3.

Таблиця 3.3 Тестування функцій mkdir, unlink, rmdir, opendir, readdir

Тест	Перевірка роботи з директорією
Номер тесту	3
Початковий стан	Користувач знаходиться в консолі
Вхідні дані	Директорія з ім'ям “dir”, внутрішній файл “indirFile”
Опис проведення тесту	Створити директорію з відповідним ім'ям з допомогою команди mkdir; створити всередині неї файл з допомогою команди touch[43]; перевірити, що префікс директорії “d”, а внутрішнього файла “-”, видалити внутрішній файл з

	допомогою команди <code>rm</code> , видалити директорію.
Очікуваний результат	Створиться директорія, всередині неї буде присутній файл, вони будуть відповідно мати типи “d” та “-”. При видаленні файла, його не залишиться в директорії. При видаленні директорії її не залишиться у корневій папці.
Фактичний результат	Створилася директорія, всередині неї файл, вони мають типи “d” та “-”. При видаленні файла, його не залишається в директорії. При видаленні директорії її не залишається у корневій папці.

Як можна побачити на рисунку 3.4, створення директорії відпрацювало успішно; коректно відпрацювало створення файла всередині директорії.

```

sideshowbobgot@localhost: /mnt/fuse
sideshowbobgot@localhost:/mnt/fuse$ mkdir dir
sideshowbobgot@localhost:/mnt/fuse$ ls -l
total 0
drwxrwxr-x 2 sideshowbobgot sideshowbobgot 0 Jan  1  1970 dir
sideshowbobgot@localhost:/mnt/fuse$ cd dir
sideshowbobgot@localhost:/mnt/fuse/dir$ touch indirFile
touch: setting times of 'indirFile': Function not implemented
sideshowbobgot@localhost:/mnt/fuse/dir$ ls -l
total 0
-rw-rw-r-- 2 sideshowbobgot sideshowbobgot 0 Jan  1  1970 indirFile
sideshowbobgot@localhost:/mnt/fuse/dir$ rm indirFile
sideshowbobgot@localhost:/mnt/fuse/dir$ ls -l
total 0
sideshowbobgot@localhost:/mnt/fuse/dir$ cd ..
sideshowbobgot@localhost:/mnt/fuse$ rm -r dir
sideshowbobgot@localhost:/mnt/fuse$ ls -l
total 0
sideshowbobgot@localhost:/mnt/fuse$

```

Рисунок 3.4 Ручна перевірка працездатності функцій `mkdir`, `unlink`, `rmdir`, `opendir`, `readdir`

Перевіримо працездатність функцій `chmod`, `access`[44] для регулярного файла. Процес показано в таблиці 3.4.

Таблиця 3.4 Тестування функцій `access`, `chmod` для регулярних файлів

Тест	Перевірка прав доступу до файла
------	---------------------------------

Номер тесту	4
Початковий стан	Користувач знаходиться в консолі
Вхідні дані	Ім'я файлу "accessFile", команда для копіювання mkdir
Опис проведення тесту	Створити файл з відповідним ім'ям, надати йому дозволи 777, перевірити, що можна вписувати та читати, забрати права доступу 555 вписування; перевірити, що не можна вписати у файл; забрати права доступу зчитування; перевірити, що не можна зчитати з файлу; скопіювати вказану стандартну команду; перевірити, що команда виконується; забрати дозволи виконання 666; перевірити, що неможливо виконати дану команду.
Очікуваний результат	Створиться відповідний файл та скопіюється відповідна команда, після забрання відповідних дозволів неможливо буде виконати відповідні команди.
Фактичний результат	Був створений відповідний файл та скопійована відповідна команда, після забрання відповідних дозволів неможливо виконати відповідні команди.

З рисунка 3.5 бачимо, що при наданні файлу усі дозволів в нього можна вписати та зчитати з нього. При прибиранні дозволу на вписування бачимо, що і справді у файл нічого не можна вписати. При прибиранні дозволу на зчитування бачимо, що і справді файл не можна зчитати. Далі протестуємо виконання бінарного файлу. Для цього з директорії /bin[45], що зберігає основні команди Unix-подібних систем, скопіюємо регулярний файл mkdir у директорию, куди приєднана ВФС. Бачимо, що при наданні всіх дозволі команда mkdir успішно відпрацьовує. G\ Приберемо дозволи на виконання бінарного файлу, бачимо, що повертається код помилки про відсутність прав доступу.

```
sideshowbobgot@localhost: /mnt/fuse
sideshowbobgot@localhost:/mnt/fuse$ touch accessFile && chmod 777 accessFile && ls -l
touch: setting times of 'accessFile': Function not implemented
sideshowbobgot@localhost:/mnt/fuse$ echo "information" > accessFile && cat accessFile
information
sideshowbobgot@localhost:/mnt/fuse$ chmod 555 accessFile && echo "newInformation" > accessFile
bash: echo: write error: Permission denied
sideshowbobgot@localhost:/mnt/fuse$ chmod 333 accessFile && cat accessFile
cat: accessFile: Operation not permitted
sideshowbobgot@localhost:/mnt/fuse$ cp /bin/mkdir mkdir && chmod 777 mkdir && ./mkdir someDir
sideshowbobgot@localhost:/mnt/fuse$ chmod 666 mkdir && ./mkdir someDir2 && ls -l
bash: ./mkdir: Permission denied
sideshowbobgot@localhost:/mnt/fuse$ ls -l
total 0
--wx-wx-wx 2 sideshowbobgot sideshowbobgot 12 Jan 1 1970 accessFile
-rw-rw-rw- 2 sideshowbobgot sideshowbobgot 68096 Jan 1 1970 mkdir
drwxrwxr-x 2 sideshowbobgot sideshowbobgot 0 Jan 1 1970 someDir
sideshowbobgot@localhost:/mnt/fuse$
```

Рисунок 3.5 Ручна перевірка працездатності функцій access, chmod на регулярному файлі

Аналогічно перевірямо працездатність access та chmod на soft-посиланні за таблицею 3.5.

Таблиця 3.5 Тестування функцій access, chmod для soft-посилань

Тест	Перевірка прав доступу до soft-посилання
Номер тесту	5
Початковий стан	Користувач знаходиться в консолі
Вхідні дані	Ім'я файла “accessFile”, команда для копіювання mkdir, ім'я посилання “accessLink”, посилання на команду “mkdirLink”
Опис проведення тесту	Створити файл з відповідним ім'ям; створити на нього посилання з допомогою команди ln[46]; надати йому дозволи 777; перевірити, що можна вписувати та читати, забрати права доступу 555 вписування; перевірити, що не можна вписати у файл; забарати права доступу зчитування; перевірити, що не можна зчитати з файла; скопіювати вказану стандартну команду; перевірити, що команда виконується; забрати дозволи виконання 666; перевірити, що неможливо виконати дану команду.
Очікуваний результат	Створиться відповідний файл та посилання, скопіюється

	відповідна команда, після забрання відповідних дозволів у посилань неможливо буде виконати відповідні команди.
Фактичний результат	Був створений відповідний файл та посилання, та скойована відповідна команда, після забрання відповідних дозволів неможливо виконати відповідні команди.

З рисунка 3.6 бачимо, що при зверненні `chmod` до посилання, міняються дозволи самого файла, тобто відбувається розіменовування.

```
sideshowbobgot@localhost: /mnt/fuse
sideshowbobgot@localhost:/mnt/fuse$ touch accessLinkFile
touch: setting times of 'accessLinkFile': Function not implemented
sideshowbobgot@localhost:/mnt/fuse$ ln -s accessLinkFile accessLink && chmod 777 accessLink && ls -l
total 0
lrwxrwxrwx 2 sideshowbobgot sideshowbobgot 14 Jan 1 1970 accessLink -> accessLinkFile
-rwxrwxrwx 2 sideshowbobgot sideshowbobgot 0 Jan 1 1970 accessLinkFile
sideshowbobgot@localhost:/mnt/fuse$ echo "information" > accessLink && cat accessLink
information
sideshowbobgot@localhost:/mnt/fuse$ chmod 555 accessLink && echo "newInformation" > accessLink
bash: echo: write error: Permission denied
sideshowbobgot@localhost:/mnt/fuse$ chmod 333 accessLink && cat accessLink
cat: accessLink: Operation not permitted
sideshowbobgot@localhost:/mnt/fuse$ cp /bin/mkdir mkdir && ln -s mkdir mkdirLink && chmod 777 mkdirLink && ./mkdirLink someDir
sideshowbobgot@localhost:/mnt/fuse$ chmod 666 mkdirLink && ./mkdirLink someDir2
bash: ./mkdirLink: Permission denied
sideshowbobgot@localhost:/mnt/fuse$ ls -l
total 0
lrwxrwxrwx 2 sideshowbobgot sideshowbobgot 14 Jan 1 1970 accessLink -> accessLinkFile
--wx--wx--wx 2 sideshowbobgot sideshowbobgot 12 Jan 1 1970 accessLinkFile
-rw-rw-rw- 2 sideshowbobgot sideshowbobgot 68096 Jan 1 1970 mkdir
lrwxrwxrwx 2 sideshowbobgot sideshowbobgot 5 Jan 1 1970 mkdirLink -> mkdir
drwxrwxr-x 2 sideshowbobgot sideshowbobgot 0 Jan 1 1970 someDir
sideshowbobgot@localhost:/mnt/fuse$
```

Рисунок 3.6 Ручна перевірка працездатності функцій `access`, `chmod` на `soft`-посиланні

Перевірямо працездатність `access` та `chmod` на директорії за тиблицею 3.6.

Таблиця 3.6 Тестування функцій `access`, `chmod` для директорії

Тест	Перевірка прав доступу до директорії
Номер тесту	6
Початковий стан	Користувач знаходиться в консолі
Вхідні дані	Ім'я директорії "dir"
Опис проведення тесту	Створити директорію з відповідним ім'ям; надати усі дозволи 777; перевірити чи можна створити піддиректорію та зчитати директорію; забрати права

	виконання директорії; перевірити, що неможливо
Очікуваний результат	Створиться відповідна директорія, після забрання відповідних дозволів у посилань неможливо буде виконати відповідні команди.
Фактичний результат	Була створена відповідна директорія, після забрання відповідних дозволів неможливо виконати відповідні команди.

З рисунка 3.7 бачимо, що при наданні директорії усіх дозволів, ми можемо в неї створювати файли, зчитувати її, та рухатися вниз по ієрархії. При прибиранні дозволу на вписування директорії бачимо, що і справді всередині неї не можна створювати нові файлид. При прибиранні прав на зчитування директорії і справді не можна подивитися, що знаходиться всередині неї. При прибиранні прав на виконання директорії і справді можна побачити, що знаходиться всередині неї, але не можна рухатися вниз по ієрархії.

```
sideshowbobgot@localhost: /mnt/fuse
sideshowbobgot@localhost:/mnt$ cd fuse
sideshowbobgot@localhost:/mnt/fuse$ mkdir dir && chmod 777 dir && mkdir
dir/subDir && ls -l
total 0
drwxrwxrwx 2 sideshowbobgot sideshowbobgot 0 Jan  1 1970 dir
sideshowbobgot@localhost:/mnt/fuse$ chmod 666 dir && ls -l dir
ls: cannot access 'dir/subDir': Permission denied
total 0
?????????? ? ? ? ? ?      ? subDir
sideshowbobgot@localhost:/mnt/fuse$ chmod 555 dir && mkdir dir/subDir2
mkdir: cannot create directory 'dir/subDir2': Permission denied
sideshowbobgot@localhost:/mnt/fuse$ chmod 333 dir && ls -l dir
ls: cannot open directory 'dir': Operation not permitted
sideshowbobgot@localhost:/mnt/fuse$
```

Рисунок 3.7 Ручна перевірка працездатності функцій access, chmod на директорії

3.2.2 Нефункціональні вимоги

Для початку оцінимо продуктивність, а саме швидкість пошуку файлів за їхнім іменем. Для цього напишемо юніт-тест скрипт, що заповнить ВФС директоріями від A-Z, потім всередині кожної директорії іще раз створить

директорії від A-Z, повторить цю операцію 4 рази, тобто у результаті будемо мати $28^4 = 614656$ директорій. Під час тестування, скажемо ВФС знайти усі повні шляхи директорій, які мають ім'я Н.

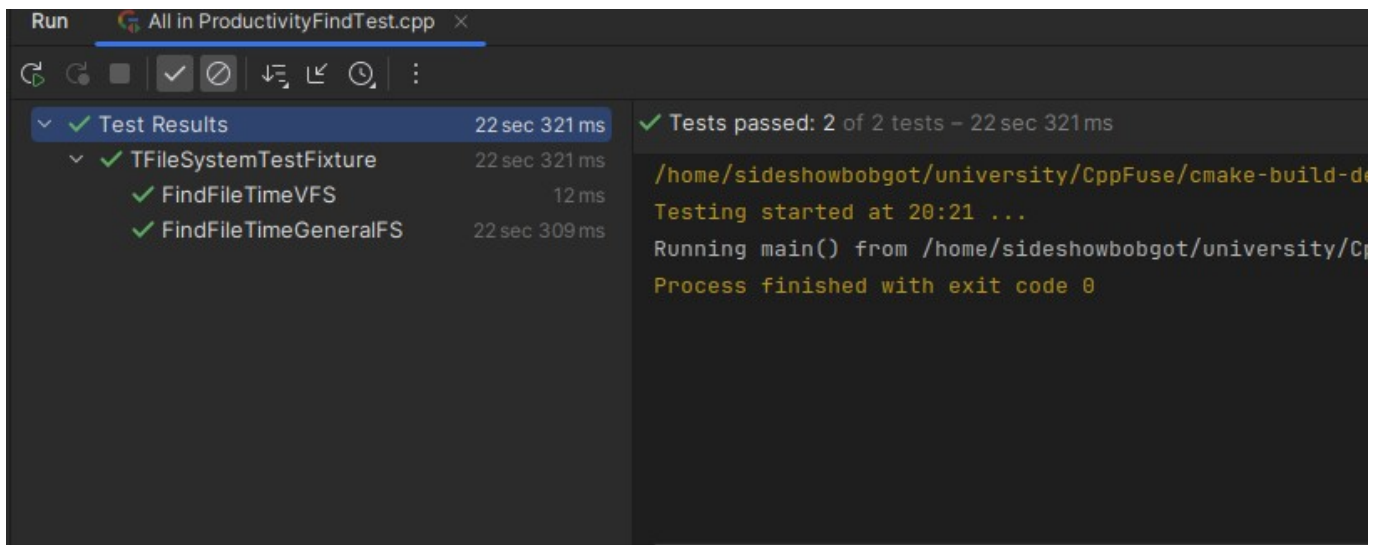


Рисунок 3.8 Результати пошуку

Бачимо, що ВФС знайшла усі файли з іменем “Н” за 12 мілісекунд, натомість файлова система Linux віднайшла їх за 22 секунди 309 мілісекунд. Прискорення у 1859 разів.

Безпека була протестована разом із операціями `chmod` та `access` у попередньому пункті. ВФС захищає несанкціонований доступ до файлів.

Оскільки тестування проводилося у багатопотоковому режимі і не було виявлено жодних критичних помилок, то система проходить перевірку надійності.

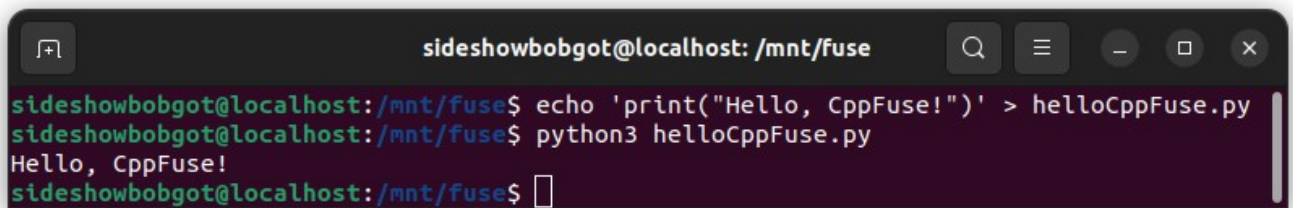
До того ж оскільки ВФС побудована в оболонці FUSE, то вона виводить усі помилки та приймає операцій як стандартні в UNIX-операційних системах, що покращує зручність інтерфейсу в цілому.

Для прикладу сумісності вже було показано, що система може використовувати стандартні команди взаємодії з файлами, як-от:

- `mkdir` — створення директорій;
- `mknod` — створення регулярних файлів;
- `ls` — зчитування директорії;
- `rm` — видалення файлів;
- `ln` — створення посилань;

- cat — зчитування і вписування файлів.

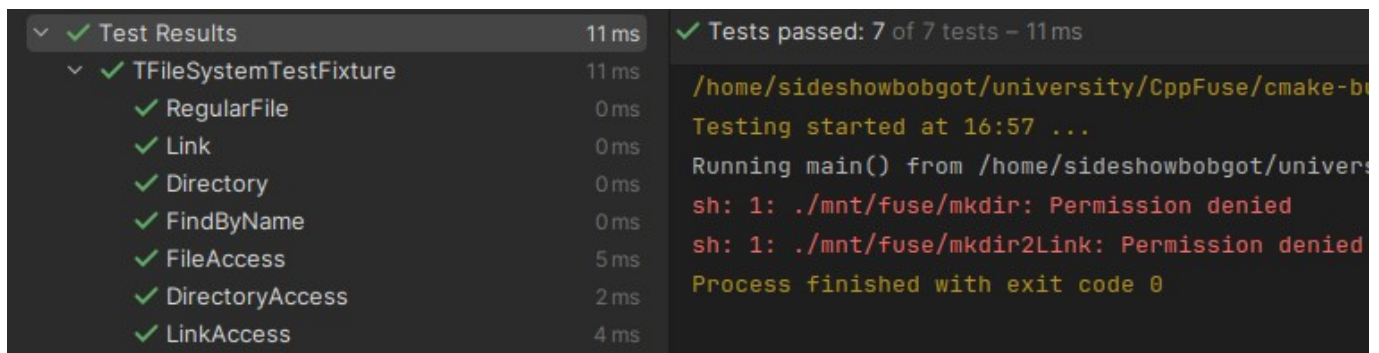
Однак покажемо, що ВФС здатна обробляти, наприклад, скрипти мови Python[47]. Запишемо у файл “print(‘Hello, CppFuse!’)” та виконаємо даний скрипт.



```
sideshowbobgot@localhost: /mnt/fuse
sideshowbobgot@localhost:/mnt/fuse$ echo 'print("Hello, CppFuse!")' > helloCppFuse.py
sideshowbobgot@localhost:/mnt/fuse$ python3 helloCppFuse.py
Hello, CppFuse!
sideshowbobgot@localhost:/mnt/fuse$
```

Рисунок 3.9 Виконання скрипта Python всередині ВФС

Детальний опис та аналіз процесів тестування наведено в «Програма та методика тестування» курсової роботи.



Test Results	Duration
Test Results	11 ms
TFFileSystemTestFixture	11 ms
RegularFile	0 ms
Link	0 ms
Directory	0 ms
FindByName	0 ms
FileAccess	5 ms
DirectoryAccess	2 ms
LinkAccess	4 ms

Tests passed: 7 of 7 tests – 11 ms

```
/home/sideshowbobgot/university/CppFuse/cmake-b...
Testing started at 16:57 ...
Running main() from /home/sideshowbobgot/univers...
sh: 1: ./mnt/fuse/mkdir: Permission denied
sh: 1: ./mnt/fuse/mkdir2Link: Permission denied
Process finished with exit code 0
```

Рисунок 3.10 Результат виконання unit-тестів

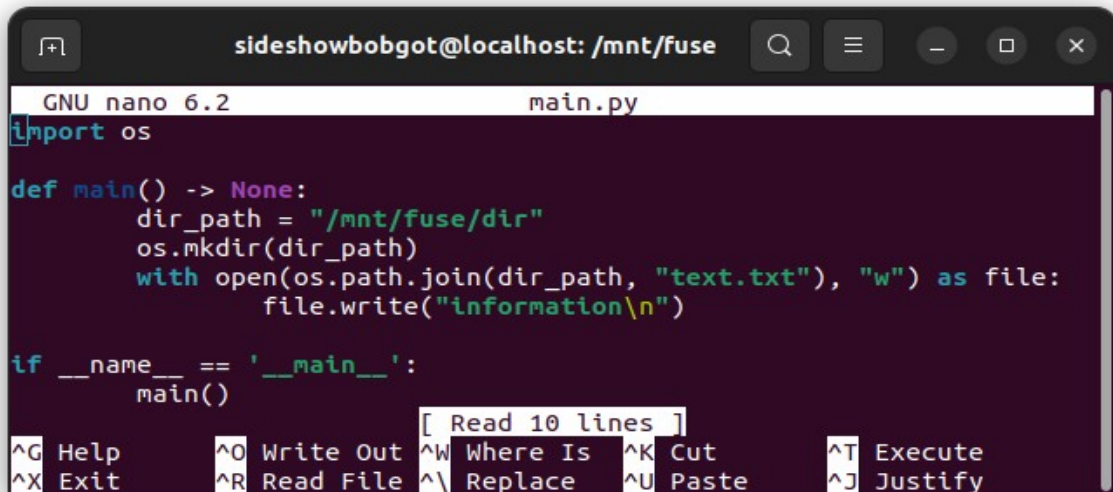
3.3 Опис контрольного прикладу

У попередньому розділі було протестовано усі реалзовані операції для файлової системи. Тому в описі контрольного прикладу будемо намагатися відтворити взаємодію користувача з файловою системою.

Для цього на рисунку 3.11 відкриємо файл у консольному текстовому редакторі Nano [47], напишемо скрипт на Python, що створить всередині файлової системи директорію з назвою “dir” та файл “text” з певним текстом. Результат виконання скрипта наведено на рисунку 3.12. Як бачимо ВФС правильно обробила системні виклики, що призвело до правильності виконання скрипта.

Створимо в декількох різних директоріях також файли з назвою text та викличемо скрипт пошуку на рисунку 3.13.

Створимо посилання на dir/text та видалимо всі файли з ВФС на рисунку 3.14.



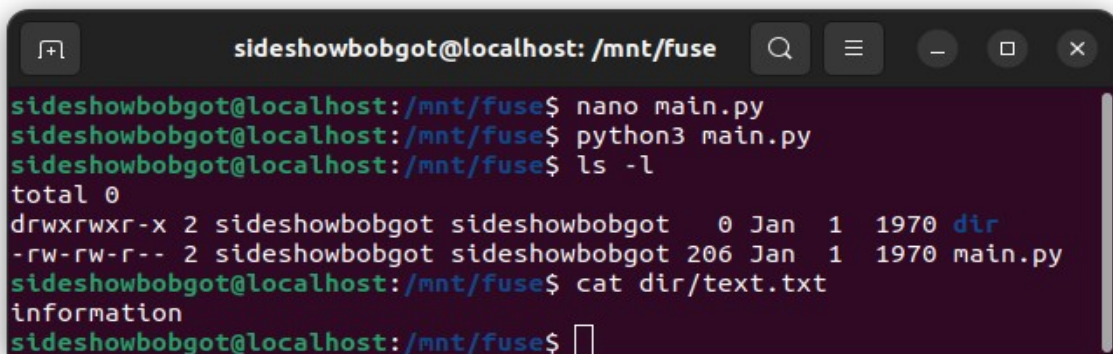
```
sideshowbobgot@localhost: /mnt/fuse
GNU nano 6.2 main.py
import os

def main() -> None:
    dir_path = "/mnt/fuse/dir"
    os.mkdir(dir_path)
    with open(os.path.join(dir_path, "text.txt"), "w") as file:
        file.write("information\n")

if __name__ == '__main__':
    main()

[ Read 10 lines ]
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify
```

Рисунок 3.11 Написання скрипта за допомогою Nano всередині ВФС



```
sideshowbobgot@localhost: /mnt/fuse
sideshowbobgot@localhost:/mnt/fuse$ nano main.py
sideshowbobgot@localhost:/mnt/fuse$ python3 main.py
sideshowbobgot@localhost:/mnt/fuse$ ls -l
total 0
drwxrwxr-x 2 sideshowbobgot sideshowbobgot  0 Jan  1  1970 dir
-rw-rw-r-- 2 sideshowbobgot sideshowbobgot 206 Jan  1  1970 main.py
sideshowbobgot@localhost:/mnt/fuse$ cat dir/text.txt
information
sideshowbobgot@localhost:/mnt/fuse$
```

Рисунок 3.12 Виконання Python скрипта

```
sideshowbobgot@localhost: /mnt...
sideshowbobgot@localhost:/mnt/fuse$ mkdir a b c d e
sideshowbobgot@localhost:/mnt/fuse$ touch a/text
touch: setting times of 'a/text': Function not implemented
sideshowbobgot@localhost:/mnt/fuse$ touch b/text
touch: setting times of 'b/text': Function not implemented
sideshowbobgot@localhost:/mnt/fuse$ touch c/text
touch: setting times of 'c/text': Function not implemented
sideshowbobgot@localhost:/mnt/fuse$ touch d/text
touch: setting times of 'd/text': Function not implemented
sideshowbobgot@localhost:/mnt/fuse$ touch e/text
touch: setting times of 'e/text': Function not implemented
sideshowbobgot@localhost:/mnt/fuse$ CppFuseFind text
--file-name is required
Run with --help for more information.
sideshowbobgot@localhost:/mnt/fuse$ CppFuseFind -f text
--file-name: 1 required TEXT missing
Run with --help for more information.
sideshowbobgot@localhost:/mnt/fuse$ CppFuseFind text
/a/text
/b/text
/c/text
/d/text
/dir/text
/e/text
sideshowbobgot@localhost:/mnt/fuse$
```

Рисунок 3.13 Пошук повного шляха до файла за іменем

```
sideshowbobgot@localhost: /mnt/fuse
sideshowbobgot@localhost:/mnt/fuse$ ln -s dir/text linkToText
sideshowbobgot@localhost:/mnt/fuse$ ls -l
total 0
drwxrwxr-x 2 sideshowbobgot sideshowbobgot 0 Jan  1  1970 a
drwxrwxr-x 2 sideshowbobgot sideshowbobgot 0 Jan  1  1970 b
drwxrwxr-x 2 sideshowbobgot sideshowbobgot 0 Jan  1  1970 c
drwxrwxr-x 2 sideshowbobgot sideshowbobgot 0 Jan  1  1970 d
drwxrwxr-x 2 sideshowbobgot sideshowbobgot 0 Jan  1  1970 dir
drwxrwxr-x 2 sideshowbobgot sideshowbobgot 0 Jan  1  1970 e
lrwxrwxrwx 2 sideshowbobgot sideshowbobgot 8 Jan  1  1970 linkToText -> dir/text
-rw-rw-r-- 2 sideshowbobgot sideshowbobgot 0 Jan  1  1970 main.py
sideshowbobgot@localhost:/mnt/fuse$ rm -r *
sideshowbobgot@localhost:/mnt/fuse$ ls -l
total 0
sideshowbobgot@localhost:/mnt/fuse$
```

Рисунок 3.14 Створення посилання та видалення усіх файлів з ВФС

3.4 Висновки до розділу

У висновку було виявлено, що абсолютно відсутня валідація на права володіння файлами всереді ВФС. Також неправильно відпрацьовував метод Write, тому неможливе було редагування файлів всередині консольних текстових редакторів. Було виявлено, що іноді ВФС виходила за межі буфера, тому вона падала при зчитуванні посилань.

Під час мануального тестування було проведено тести на реагування системи

на коректне та некоректне введення даних; побачили, що система виводить зрозумілі для користувача повідомлення про помилку.

Система задовольняє умову сумісності з іншими програмними застосунками, що було показано на прикладі виконання скриптів Python всередині ВФС.

Також протестували, що ВФС задовольняє швидкість пошуку файлів всередині неї.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Для розгортання ВФС потрібно виконати такі кроки:

- встановити бібліотеку FUSE:
`sudo apt-get update -y && sudo apt-get install -y libfuse-dev;`
- скопувати CppFuse репозиторій:
`git clone https://github.com/SideShowBoBGOT/CppFuse`
- Збудувати проект з допомогою Cmake:
`cd CppFuse && cmake -S . -B build && cmake --build build`
- Створити FIFO-файл для комунікації з ВФС:
`mkfifo fifo && chmod fifo 0775`
- Створити директорію для ВФС:
`mkdir /mnt/fuse && chmod /mnt/fuse/ 0775`
- Приєднайте ВФС до основної файлової системи:
`mount -t /mnt/fuse && fusermount -u /mnt/fuse`
- Запустіть ВФС:
`./build/MainExecutable -f -d -m /mnt/fuse -p fifo`

Файлова система може приймати різні прапорці при запуску:

Коротке ім'я	Довге ім'я	Чи є обов'язкове?	Опис
-f	--foreground-process	Ні	Тримати як фронтальний процес
-n	--no-threads	Ні	Вимкнути підтримку багатопотоковості
-d	--debug Needs	Потребує -f	Відображати символи налагодження
-m	--mount-point	Так	Точка приєднання ВФС до основної ФС
-p	--pipe-point	Так	Точка FIFO-файла для комунікації

4.2 Висновки до розділу

У даному розділі було визначено необхідні бібліотеки для встановлення, такі як `libfuse-dev`, що надає можливість операційній системі надавати функції для програмування власної ВФС.

Було показано, що для того, щоб розгорнути додаток, то треба встановити збілдити його з допомогою `Cmake` та створити FIFO файл для міжпроцесної комунікації.

Були наведені основні командні аргументи ВФС, що включають в себе можливість виконання в багатопотоковому режимі, встановлення процесу ВФС на передній план, налаштування точки приєднання ВФС та точки комунікації з допомогою FIFO-файла.

ВИСНОВКИ

У процесі роботи над курсовою роботою та розробкою ВФС було отримав підвищив ряд важливих навичок та знань:

- вивчив принципи розробки ефективних та гнучких архітектур програмного забезпечення;
- навчився використовувати паттерни проектування для досягнення гнучкості та розширюваності коду;
- здобув досвід використання шаблонів C++ для створення загальних та гнучких компонентів;
- освоїв статичний поліморфізм за допомогою CRTP (Curiously Recurring Template Pattern);
- набув вмінь використовувати фреймворк googletest для написання та виконання тестів;
- освоїв техніки тестування різних аспектів програмного забезпечення, забезпечивши високу стабільність та надійність;
- вивчив принципи роботи та взаємодії з бібліотекою FUSE для реалізації ВФС;
- здобув досвід роботи з системними викликами для операцій над файлами та директоріями;
- вивчив процес оптимізації програмного забезпечення для підвищення його ефективності;
- отримав практичні навички розгортання ВФС, встановлення бібліотек та конфігураційних параметрів;
- отримав досвід встановлення пріоритетів та планування робочих завдань.

У процесі роботи над проектом та курсовою роботою я вдосконалив та поглибив свої знання в області розробки програмного забезпечення, освоїв нові інструменти та техніки, що покращило мої навички у галузі системного програмування та архітектурного проектування.

У процесі аналізу вимог до Віртуальної Файлової Системи (ВФС) визначено ключові функціональні та нефункціональні вимоги. Серед них — висока ефективність, забезпечення безпеки та гнучкість використання. Спроековано основні взаємодії з файловою системою, визначено формати даних та розглянуто

основні сценарії використання.

У розділі моделювання та конструювання була розроблена архітектура ВФС. Вона включає класи та компоненти, представлені для файлів та директорій. Використано різні паттерни проектування, такі як Visitor для забезпечення різноманітності операцій над різними типами файлів. Статичний поліморфізм реалізовано за допомогою шаблонів та CRTP паттерну.

Тестування ВФС було проведено за допомогою фреймворку googletest. Обрано його через високу ефективність, гнучкість та можливість паралельного виконання тестів. Тестування охопило різні аспекти функціональності, забезпечивши стабільність та надійність програмного забезпечення.

У розділі впровадження були детально описані кроки для розгортання ВФС. Це включало встановлення необхідних бібліотек, створення FIFO-файлу для міжпроцесної комунікації та встановлення зв'язку з основною файловою системою. Документовано командні аргументи для легкості розгортання та налаштування.

Майбутній розвиток ВФС може включати розширення функціональності для підтримки нових операцій, покращення ефективності та роботи з великим обсягом даних. Інтеграція з іншими системами та підтримка нових типів файлів можуть стати напрямками подальшого розвитку. Постійна оптимізація та удосконалення можуть допомогти ВФС вдосконалюватися та відповідати зростаючим вимогам.

У майбутньому можливе створення NetFS[48] - розподіленої файлової системи, яка дозволить взаємодіяти з файлами та директоріями через мережу.

Основною ідеєю NetFS буде розширення можливостей ВФС на рівень роботи з файлами через мережу. Це може включати в себе можливість зчитування та запису файлів на віддалених серверах, спільний доступ до файлових ресурсів для різних користувачів та роботу з файлами через інтернет.

Створення NetFS відкриває широкі перспективи для розробки клієнтських та серверних компонентів, взаємодії між різними пристроями та впровадження додаткових функціональних можливостей для забезпечення ефективного та зручного користування файловою системою.

ПЕРЕЛІК ПОСИЛАНЬ

1. Схема роботи файлової системи, побудованої на основі FUSE бібліотеки [Електронний ресурс] — <https://georgesims21.github.io/posts/fuse/>.
2. FUSE (File System in User Space) [Електронний ресурс] — <https://www.kernel.org/doc/html/next/filesystems/fuse.html>.
3. Oracle Corporation (company) [Електронний ресурс] — https://en.wikipedia.org/wiki/Oracle_Corporation.
4. ZFS (filesystem) [Електронний ресурс] — <https://en.wikipedia.org/wiki/ZFS>.
5. NTFS (filesystem) [Електронний ресурс] — <https://en.wikipedia.org/wiki/NTFS>.
6. Ext4 (filesystem) [Електронний ресурс] — <https://en.wikipedia.org/wiki/Ext4>.
7. APFS (filesystem) [Електронний ресурс] — https://en.wikipedia.org/wiki/Apple_File_System.
8. Getattr [Електронний ресурс] — <https://man7.org/linux/man-pages/man2/getxattr.2.html>.
9. Readlink [Електронний ресурс] — <https://man7.org/linux/man-pages/man2/readlink.2.html>.
10. Mknod [Електронний ресурс] — <https://man7.org/linux/man-pages/man2/mknod.2.html>.
11. Mkdir [Електронний ресурс] — <https://en.wikipedia.org/wiki/Mkdir>.
12. Unlink [Електронний ресурс] — <https://man7.org/linux/man-pages/man2/unlink.2.html>.
13. Rm [Електронний ресурс] — [https://en.wikipedia.org/wiki/Rm_\(Unix\)](https://en.wikipedia.org/wiki/Rm_(Unix)).
14. Rmdir [Електронний ресурс] — <https://man7.org/linux/man-pages/man2/rmdir.2.html>.
15. Symlink [Електронний ресурс] — <https://man7.org/linux/man-pages/man2/symlink.2.html>.
16. Chmod [Електронний ресурс] — <https://man7.org/linux/man-pages/man1/chmod.1.html>.
17. Read [Електронний ресурс] — <https://man7.org/linux/man-pages/man2/read.2.html>.
18. Write [Електронний ресурс] — <https://man7.org/linux/man-pages/man2/write.2.html>.
19. Readdir [Електронний ресурс]

<https://man7.org/linux/man-pages/man3/readdir.3.html>.

20. Traceability matrix [Електронний ресурс] — https://en.wikipedia.org/wiki/Traceability_matrix.

21. C++ language [Електронний ресурс] — <https://en.cppreference.com/w/cpp/language>.

22. Business Process Model and Notation [Електронний ресурс] — https://en.wikipedia.org/wiki/Business_Process_Model_and_Notation.

23. MVC (Model-View-Controller) [Електронний ресурс] — <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>.

24. CLI11 [Електронний ресурс] — <https://github.com/CLIUtils/CLI11>.

25. Unified Modeling Language [Електронний ресурс] — https://en.wikipedia.org/wiki/Unified_Modeling_Language.

26. “Adapter” паттерн [Електронний ресурс] — <https://refactoring.guru/design-patterns/adapter>.

27. RwLock — впапер типів для багатопотокового середовища [Електронний ресурс] — <https://github.com/SideShowBoBGOT/RwLock>.

28. “Composite” паттерн [Електронний ресурс] — <https://refactoring.guru/design-patterns/composite>.

29. Офіційна документація до std::variant [Електронний ресурс] — <https://en.cppreference.com/w/cpp/utility/variant>.

30. “Visitor” паттерн [Електронний ресурс] — <https://refactoring.guru/design-patterns/visitor>.

31. Breaking Dependencies - The Visitor Design Pattern in Cpp - Klaus Iglberger - CppCon 2022 [Електронний ресурс] — <https://www.youtube.com/watch?v=PEcy1vYHb8A&t=1658s>.

32. Офіційна документація до функції std::visit [Електронний ресурс] — <https://en.cppreference.com/w/cpp/utility/variant/visit>.

33. Офіційна документація до операцій FUSE [Електронний ресурс] — http://libfuse.github.io/doxygen/structfuse_operations.html.

34. CCCC (C and C++ Code Counter) [Електронний ресурс] — <https://github.com/sarnold/cccc>.

35. LOC (Lines of Code) [Электронный ресурс] — <https://buildpulse.io/blog/different-ways-of-measuring-code-complexity>.
36. MVG (McCabe's Cyclomatic Complexity) [Электронный ресурс] — https://en.wikipedia.org/wiki/Cyclomatic_complexity.
37. GoogleTest [Электронный ресурс] — <https://github.com/google/googletest>.
38. Open [Электронный ресурс] — <https://man7.org/linux/man-pages/man2/open.2.html>.
39. Echo command [Электронный ресурс] — [https://en.wikipedia.org/wiki/Echo_\(command\)](https://en.wikipedia.org/wiki/Echo_(command)).
40. Cat (Unix) [Электронный ресурс] — [https://en.wikipedia.org/wiki/Cat_\(Unix\)](https://en.wikipedia.org/wiki/Cat_(Unix)).
41. Ls [Электронный ресурс] — <https://en.wikipedia.org/wiki/Ls>.
42. Opendir [Электронный ресурс] — <https://man7.org/linux/man-pages/man3/openssl.3.html>.
43. Touch [Электронный ресурс] — [https://en.wikipedia.org/wiki/Touch_\(command\)](https://en.wikipedia.org/wiki/Touch_(command)).
44. Access [Электронный ресурс] — <https://man7.org/linux/man-pages/man2/access.2.html>.
45. /bin directory [Электронный ресурс] — <https://uk.wikipedia.org/wiki//bin>.
46. Ln (command) [Электронный ресурс] — <https://man7.org/linux/man-pages/man1/ln.1.html>.
47. Python [Электронный ресурс] [https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)).
48. NetFS — Networking Through The File System [Электронный ресурс] — <https://www.strayalpha.com/netfs/>.