Пояснювальна записка
до курсової роботи

на тему: файлова система з консольним інтерфейсом

Київ — 2023

ЗМІСТ

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 швидкодією обробки інформації.

У даній курсовій роботі буде розглянуто концепцію віртуальних файлових систем, проведений аналіз існуючих рішень, та обгрунтована 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. Роглянемо схему роботи на прикладі малюнку [1]:

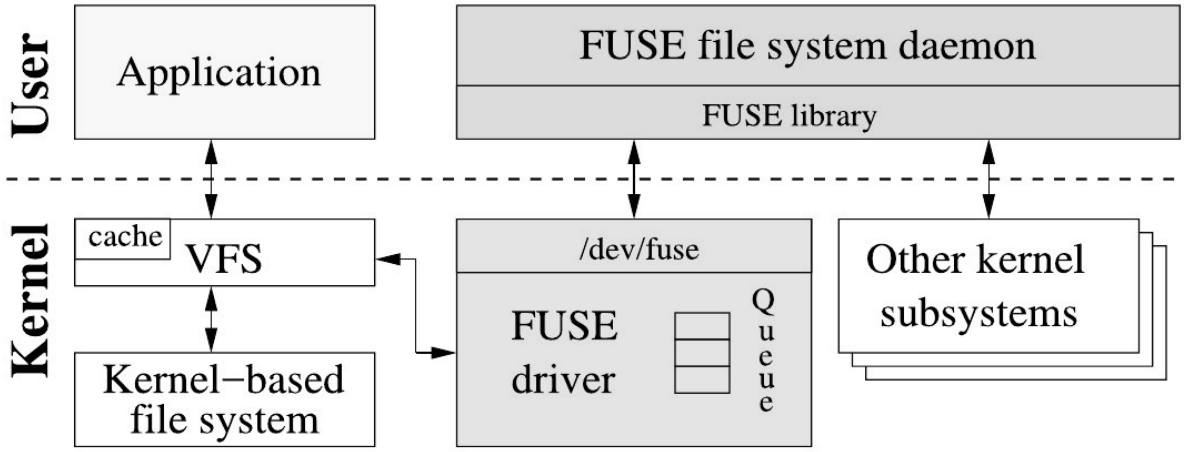

Рисунок 1.1 Схема роботи файлової системи, побудованої на основі FUSE бібліотеки

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.

1.2 Аналіз успішних IT-проектів

У цьому розділі ми ретельно аналізуємо успішні IT-проекти, зосереджуючись на віртуальних файлових системах. Розглядаючи їх переваги та недоліки, ми визначимо ключові аспекти, які можуть визначати ефективні рішення в сучасному IT-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 швидкодією 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 IT-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 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 (раніше Sun Microsystems).

Основні функціональні можливості:

- ZFS —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 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:

Рисунок 2.1 портабельність: оскільки NTFS створена для платформи Windows, вона може бути менш сумісною з іншими операційними системами;

Рисунок 2.2 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 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 файлих 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 є файловою системою, розробленою для операційних систем Apple, таких як macOS, iOS, watchOS та tvOS;
- 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

Розгялнувши наявні успішні IT-проекти, складемо таблицю порівння.

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 розповсюджується як відкрите програмне	Oracle Corporation (раніше Sun Microsystems)	Microsoft Corporation	Розробницьке співтовариство Linux	Apple Inc.

	забезпечення				
Основні функціональні можливості	Створення віртуальних файлових систем у просторі користувача	Розподілена файлова система, журналювання, автоматичне виявлення та виправлення помилок	Розширений контроль доступу, журналювання, квоти	Розширений розмір файлів та директорій, журналювання, відновлення	Оптимізація для SSD, розширене шифрування
Переваги	—Гнучкість в створенні різноманітних файлових систем	—Висока ефективність, автоматичне виявлення та виправлення помилок	—Безпека, журналювання	—Висока швидкодія, журналювання	— Швидкодія на SSD, вбудоване шифрування
Недоліки	—Затримки через простір користувача	—Високі вимоги до обладнання, складність налаштувань	— Специфічність для платформи Windows	— Фрагментація, обмежені функції	— Сумісність з іншими ОС, неідентифіковані недоліки

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.

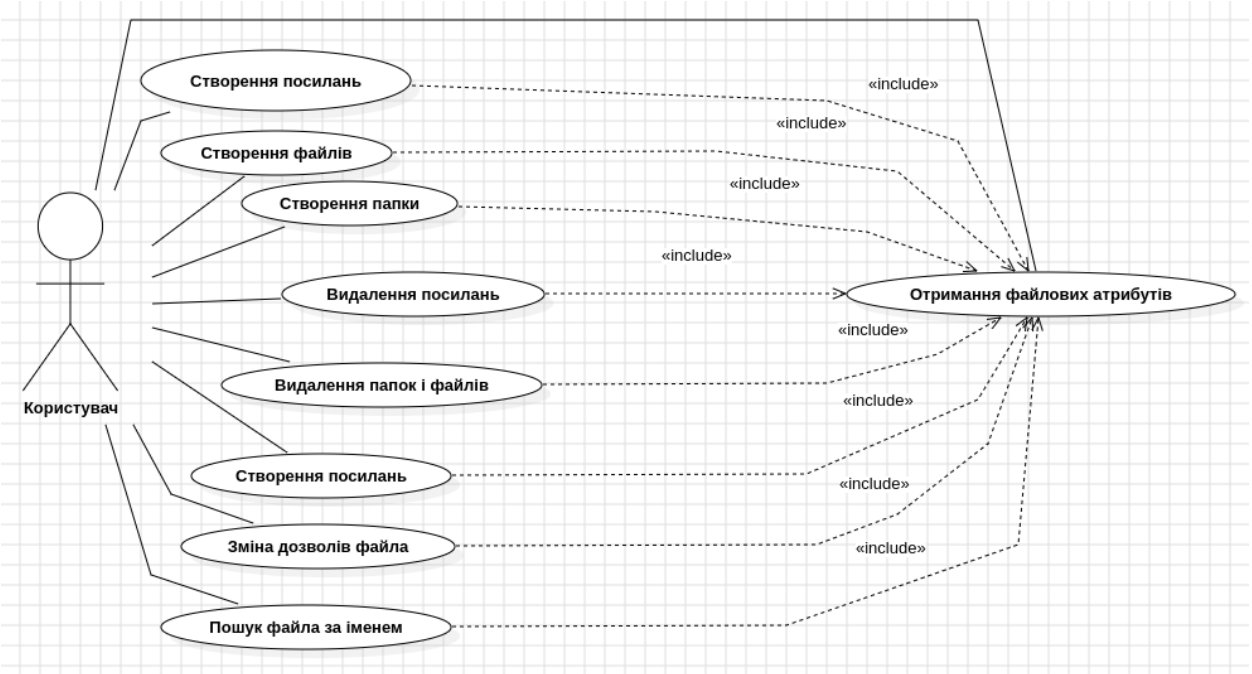

Рисунок 1.2 Діаграма використання

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)	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)	Система повинна забезпечити можливість користувачеві зчитувати вміст символьного посилання та виводити його у консоль.
FR-3	Створення файла (mknod)	Система повинна дозволяти користувачеві створювати новий файл у віртуальній файловій системі та виводити підтвердження про успішне створення у консоль.
FR-4	Створення папки	Система повинна дозволяти користувачеві

	(mkdir)	створювати нову директорію у віртуальній файловій системі та виводити підтвердження про успішне створення у консоль.
FR-5	Видалення посилань (unlink)	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 (rmdir)	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)	Система повинна дозволяти користувачеві створювати символьні посилання на об'єкти у віртуальній файловій системі та виводити підтвердження про успішне створення у консоль.
FR-8	Зміна дозволів файла (chmod)	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)	Система повинна дозволяти користувачеві зчитувати вміст файлів у віртуальній файловій системі та виводити його у консоль.
FR-10	Редагування файла (write)	Система повинна дозволяти користувачеві змінювати вміст файлів у віртуальній файловій системі та виводити підтвердження про успішне редагування у консоль.
FR-11	Зчитування папки (readdir)	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.

1.5.2 Трасування вимог

Таблиця трасування вимог 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.

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

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

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, масштабованість, ефективність, доступність 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 пошуковими 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 пошуку файлів за іменем. Запити на пошук повинні бути валідовані та авторизовані перед обробкою.
NFR-6	Масштабованість	Система повинна бути здатна масштабуватися та обробляти одночасно багато запитів на пошук файлів за іменем без значних втрат продуктивності.
NFR-7	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.

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 IT-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++, 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 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.

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, ядрова файлова система передає управління цій файловій системі. У випадку, якщо шлях не знайдений або вказана файлова система не існує, ядрова 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 надсила відповідь файловій системі ядра, яка в свою чергу передає цю відповідь зовнішній програмі. Зовнішня програма отримує результат операції, і обробка запиту завершується.

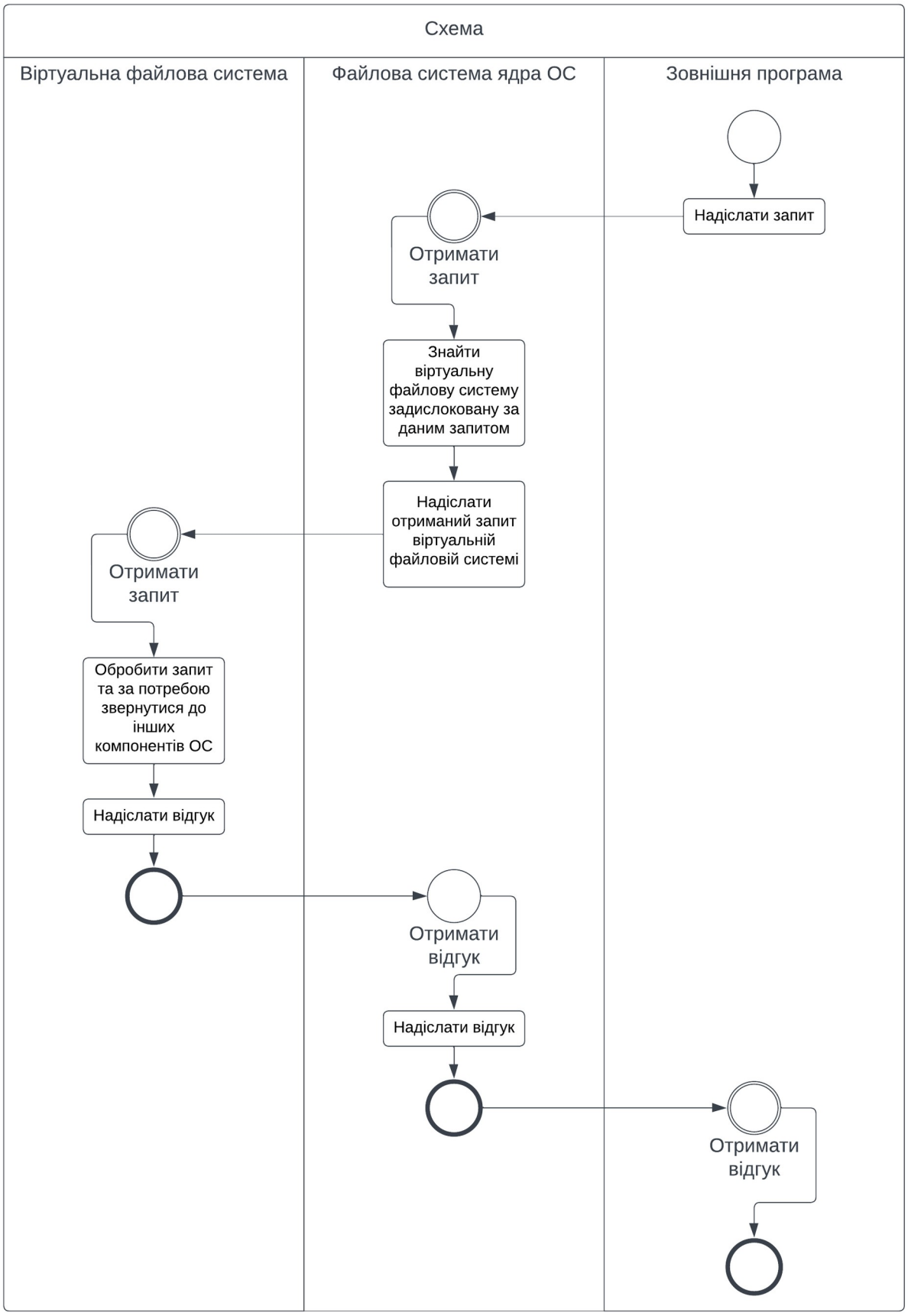

Рисунок 2.1 Схема бізнес-процесу отримання відгуку ВФС

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.

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) - 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, 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) - 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:

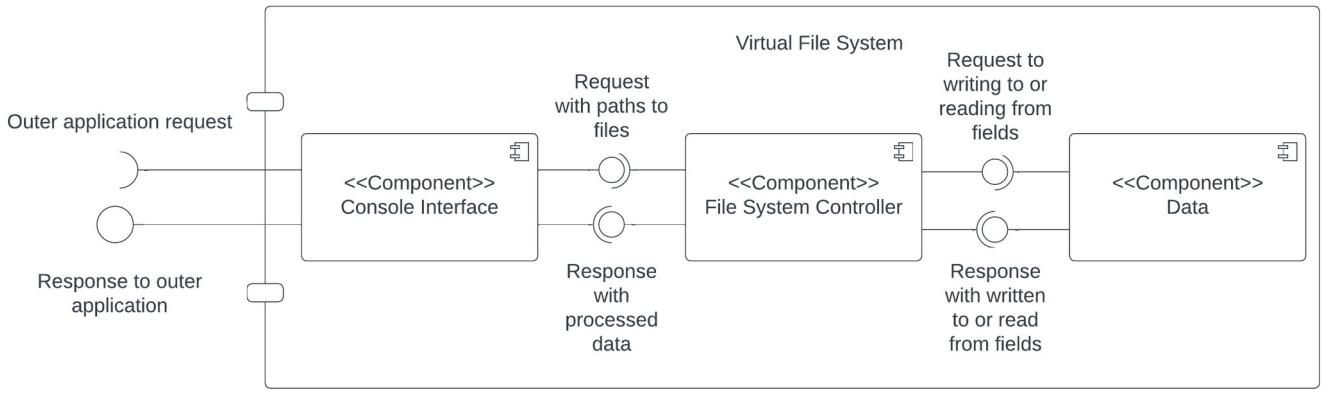

Рисунок 2.2 Діаграма компонентів ВФС

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 [2]. 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 напряму.

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). 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	TReadGuard<T> Read() const	Забезпечує багатопотокове зчитування даних, надаючи TRwLockReadGuard, який автоматично відімкнеться при завершенні області дії. Дозволяє багатьом потокам одночасно читати дані без блокування один одного.
Write	WriteGuard<T> Write()	Забезпечує однопотоковий запис даних, надаючи TRwLockWriteGuard, який автоматично відімкнеться при завершенні області дії. Гарантує, що операції запису виконуються безпечно та без конфліктів в однопотоковому режимі.
TryRead	std::optional<TRwLockTryRead	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<TRwLockTryWrite Guard<T>> TryWrite()	Спроба однопотокового запису даних без блокування. Повертає TRwLockTryWriteGuard, якщо операція успішна, або std::nullopt, якщо ресурс вже захоплений для читання чи запису.

TrwLockGuardBase<T> є шаблонним базовим класом для усіх guard`ів. Кожен дочірній клас відповідає за спеціалізцію 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 - проектовані 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 [3]. Цей шаблон проектування відноситься до структурних патернів і дозволяє клієнтам обробляти окремі об'єкти та їхні композиції (групи об'єктів) однаковим 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-дігараму моделей файлової системи на прикладі віртуальної файлової системи:

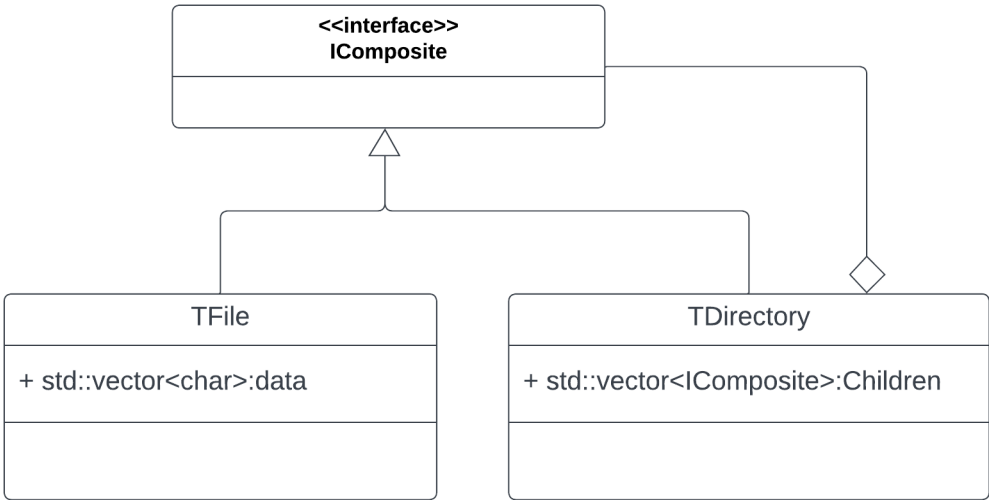

Рисунок 2.4 Composite шаблон на прикладі ВФС

Обгорнемо типи в TRwLock<T>:

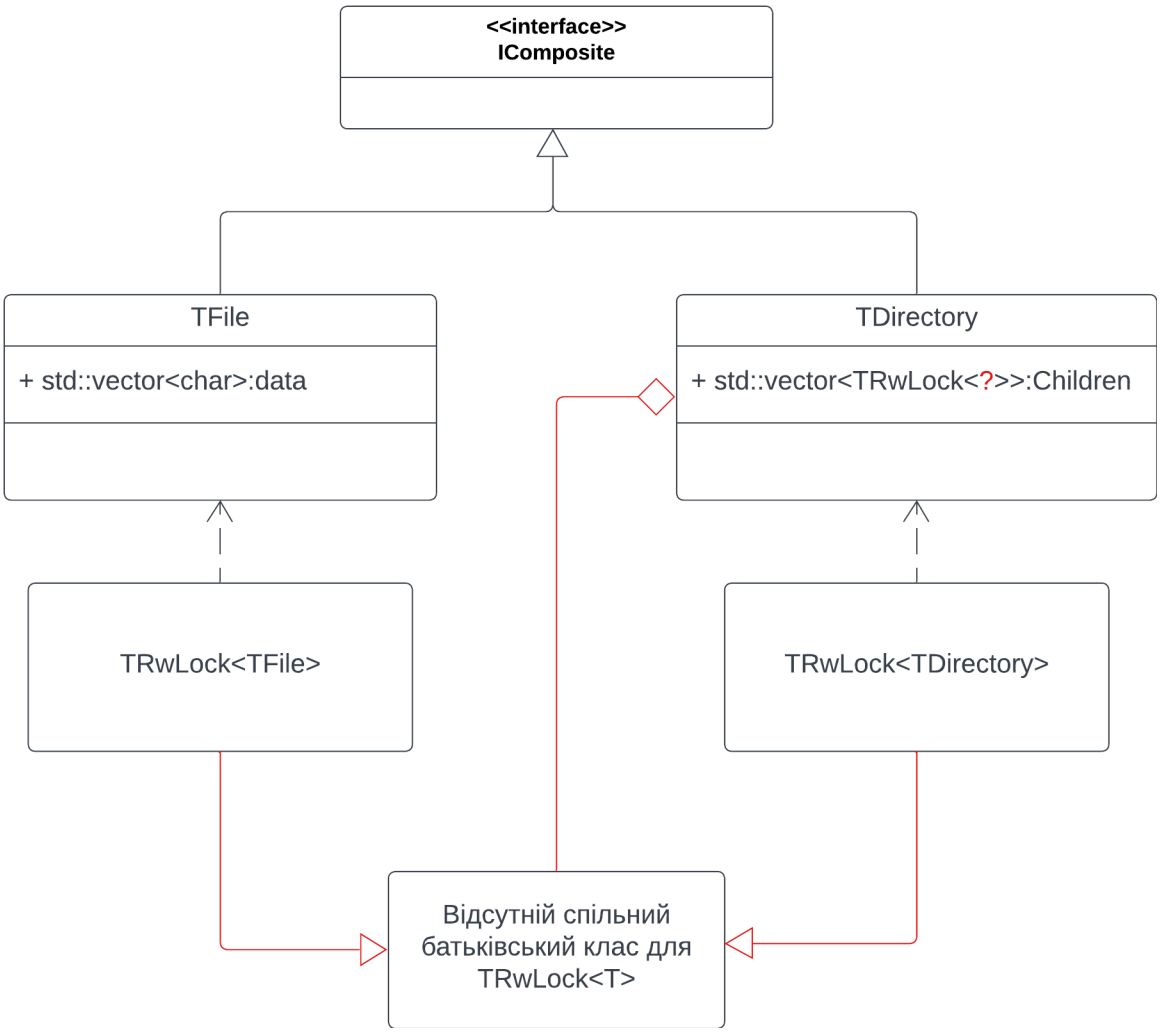

Рисунок 2.5 Composite шаблон з TrwLock на прикладі ВФС

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 [4].

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

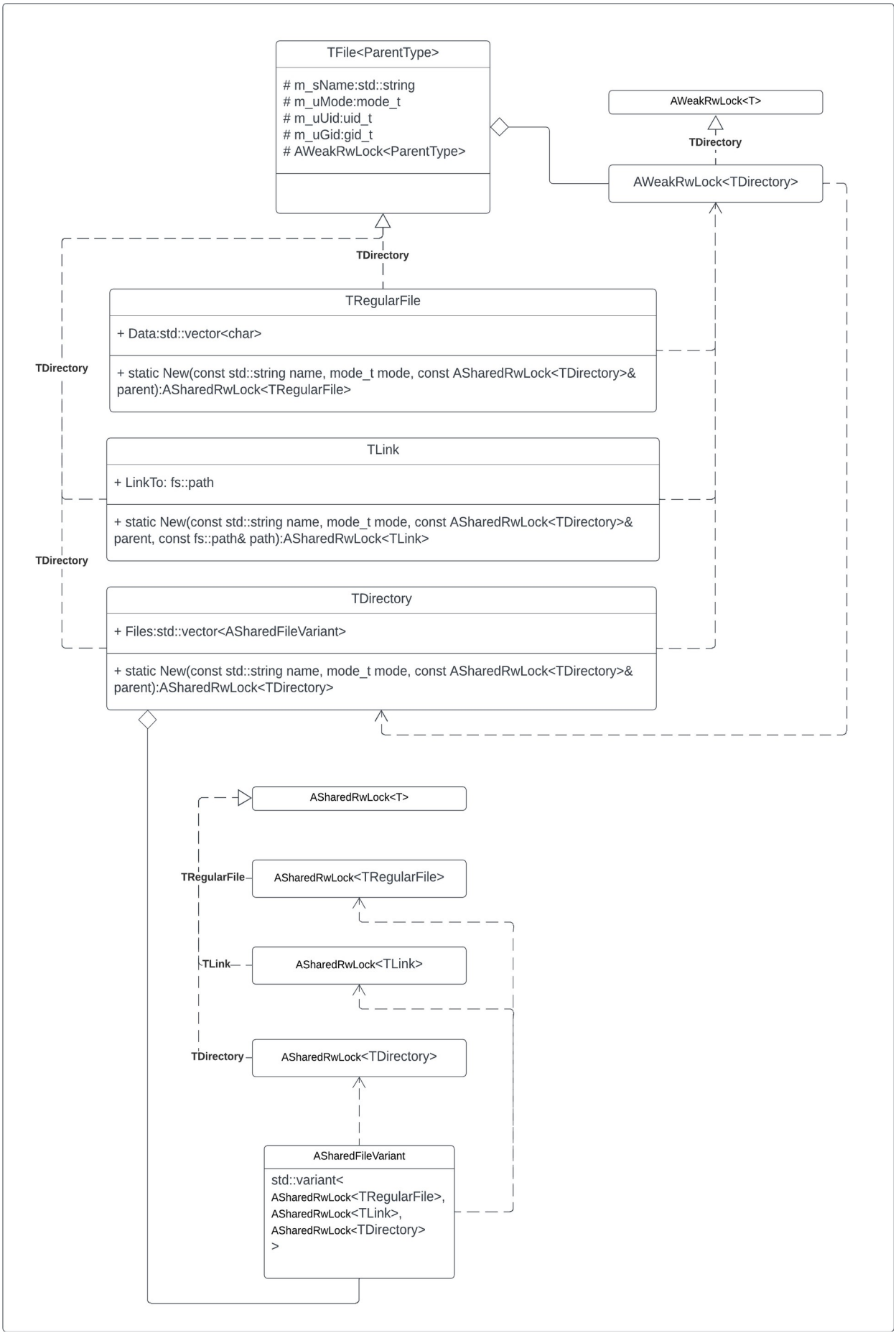

Рисунок 2.6 Діаграма класів моделей ВФС з використанням std::variant

2.3.2.2 Класи-контролери, “Visitor” паттерн

Visitor (відвідувач) [5] 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 імплеменувати 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 [6], де він розповідає про використання std::visit [7] як основного інтрумента 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. Потрібно також звернути увагу, що через специфіку мови C++ простори імен(namespaces), які складаються лише з функцій, будуть вважатися класами, що мають лише публічні статичні методи.

Таблиця 2.4 Класи для зміни атрибутів файлів

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

	TsetInfoUid.
TSetInfoMode	Дочірній клас, який відповідає за запис режимів доступу файла. Параметризується типами mode_t, TSetInfoMode.
TSetInfoParent	Дочірній клас, який відповідає за запис батька файла. Параметризується типом AsharedRwLock<Tdirectory>.

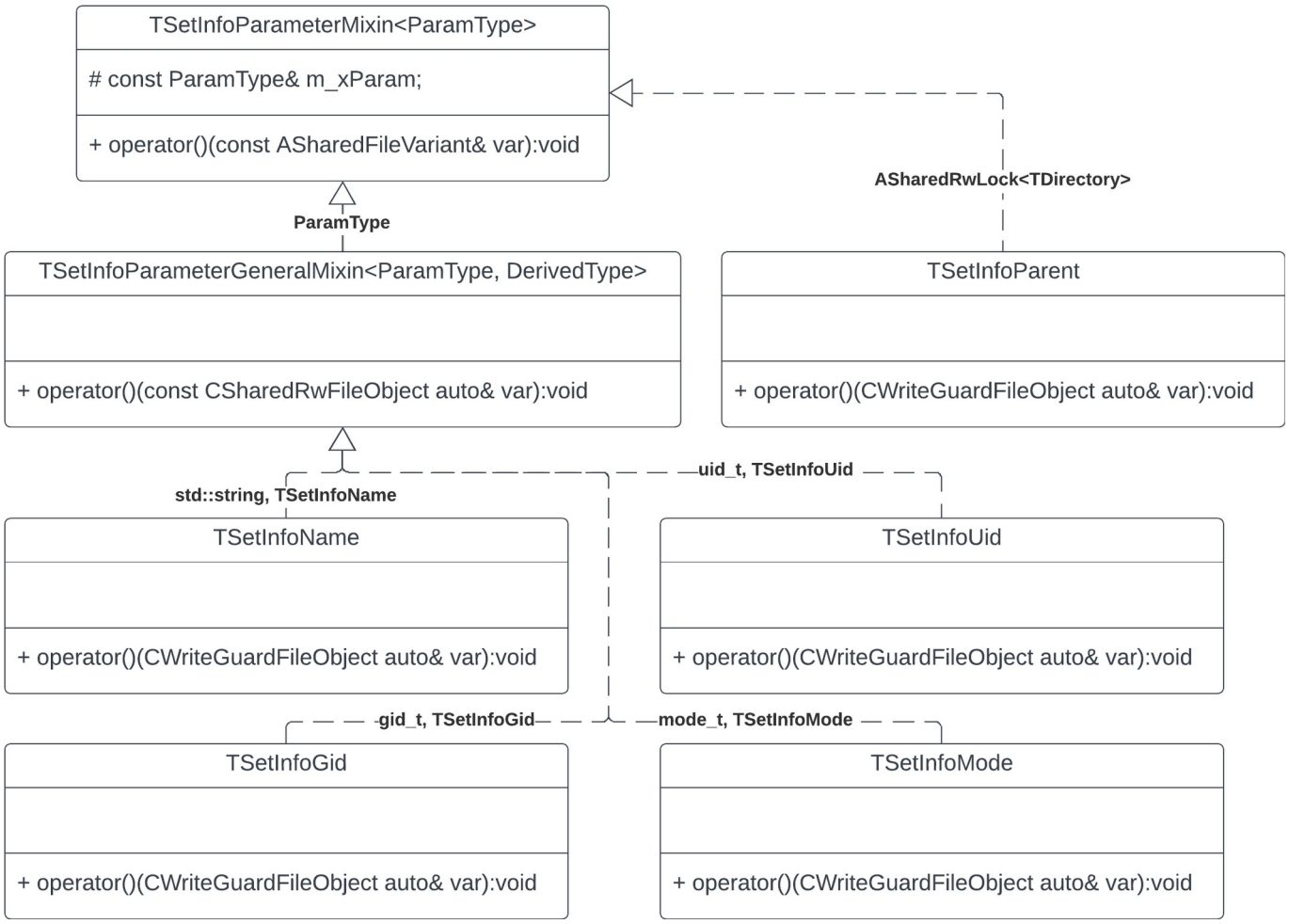

Рисунок 2.7 Діаграма класів для зміни атрибутів файлів

Таблиця 2.5 Класи для зчитування атрибутів файлів

Назва	Опис
TGetFileParameter<ParamType, Derived>	Змішуючий клас, який додає функціональність перевантаження оператору () 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.

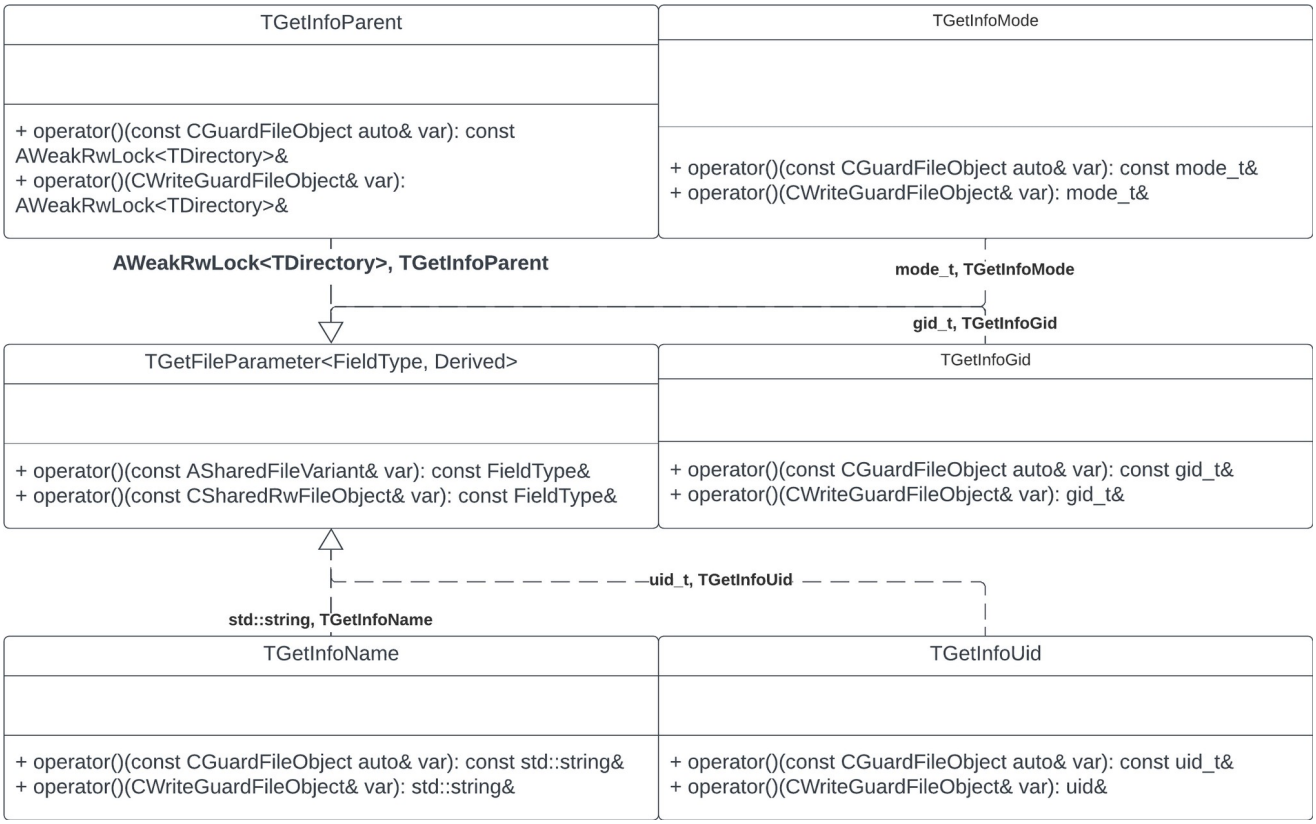

Рисунок 2.8 Діаграма класів для зчитування атрибутів файлів

ПЕРЕЛІК ПОСИЛАНЬ

1. Схема роботи файлової системи, побудованої на основі FUSE бібліотеки [Електронний ресурс] — https://georgesims21.github.io/posts/fuse/.
2. “Adapter” паттерн [Електронний ресурс] — https://refactoring.guru/design-patterns/adapter.
3. “Composite” паттерн [Електронний ресурс] — https://refactoring.guru/design-patterns/composite.
4. Офіційна документація до std::variant [Електронний ресурс] — https://en.cppreference.com/w/cpp/utility/variant.
5. “Visitor” паттерн [Електронний ресурс] — https://refactoring.guru/design-patterns/visitor.
6. Breaking Dependencies - The Visitor Design Pattern in Cpp - Klaus Iglberger - CppCon 2022 [Електронний ресурс] — https://www.youtube.com/watch?v=PEcy1vYHb8A&t=1658s.
7. Офіційна документація до функції std::visit [Електронний ресурс] — https://en.cppreference.com/w/cpp/utility/variant/visit.

ДОДАТОК А

ПРИКЛАДИ ВИКОРИСТАННЯ ФАЙЛОВОЇ СИСТЕМИ

Таблиця 1.5 Варіант використання UC-01

Use Case ID	UC-01
Use Case Name	Отримання файлових атрибутів (getattr)
Goals	Отримати інформацію про атрибути файлу
Actors	Користувач в системі
Trigger	Користувач бажає отримати інформацію про файл
Pre-conditions	Користувач має доступ до системи та файлів
Flow of Events	1. Користувач викликає команду для отримання атрибутів файлу. 2. Система передає запит на обробку віртуальній файловій системі на основі FUSE. 3. Віртуальна файлова система виконує операцію getattr та повертає інформацію про атрибути файлу. 4. Система повертає отримані атрибути користувачеві.
Extension	У випадку, якщо файл не існує, система повідомляє користувача про помилку.
Post-Condition	Користувач отримує інформацію про атрибути файлу.

Таблиця 1.6 Варіант використання UC-02

Use Case ID	UC-02
Use Case Name	Зчитування посилань (readlink)
Goals	Отримати цільовий об'єкт (шлях або файлину)
Actors	Користувач в системі
Trigger	Користувач бажає отримати цільовий об'єкт, на який вказує символьне посилання
Pre-conditions	Користувач має доступ до системи та віртуальної файлової

	системи
Flow of Events	1. Користувач викликає команду для зчитування вмісту символьного посилання. 2. Система передає запит на обробку віртуальній файловій системі на основі FUSE. 3. Віртуальна файлова система виконує операцію readlink та повертає шлях або ім'я цільового об'єкта. 4. Система повертає отриманий вміст користувачеві.
Extension	У випадку, якщо символьне посилання не існує, система повідомляє користувача про помилку.
Post-Condition	Користувач отримує інформацію про цільовий об'єкт.

Таблиця 1.7 Варіант використання UC-03

Use Case ID	UC-03
Use Case Name	Створення файла (mknod)
Goals	Створити новий файл у віртуальній файловій системі
Actors	Користувач в системі
Trigger	Користувач бажає створити новий файл
Pre-conditions	Користувач має доступ до системи та віртуальної файлової системи
Flow of Events	1. Користувач викликає команду для створення нового файлу. 2. Система передає запит на обробку віртуальній файловій системі на основі FUSE. 3. Віртуальна файлова система виконує операцію mknod та створює новий файл. 4. Система повідомляє користувача про успішне створення файлу.
Extension	У випадку, якщо створення файла неможливе (наприклад, через відсутність прав), система повідомляє користувача про помилку.

Post-Condition	Користувач отримує підтвердження про створення нового файла.

Таблиця 1.8 Варіант використання UC-04

Use Case ID	UC-04
Use Case Name	Створення папки (mkdir)
Goals	Створити нову директорію у віртуальній файловій системі
Actors	Користувач в системі
Trigger	Користувач бажає створити нову директорію
Pre-conditions	Користувач має доступ до системи та віртуальної файлової системи
Flow of Events	1. Користувач викликає команду для створення нової директорії. 2. Система передає запит на обробку віртуальній файловій системі на основі FUSE. 3. Віртуальна файлова система виконує операцію mkdir та створює нову директорію. 4. Система повідомляє користувача про успішне створення директорії.
Extension	У випадку, якщо створення директорії неможливе (наприклад, через відсутність прав), система повідомляє користувача про помилку.
Post-Condition	Користувач отримує підтвердження про створення нової директорії.

Таблиця 1.9 Варіант використання UC-05

Use Case ID	UC-05
Use Case Name	Видалення посилань (unlink)
Goals	Видалити посилання на об'єкт у віртуальній файловій системі
Actors	Користувач в системі

Trigger	Користувач бажає видалити посилання
Pre-conditions	Користувач має доступ до системи та віртуальної файлової системи
Flow of Events	1. Користувач викликає команду для видалення посилання. 2. Система передає запит на обробку віртуальній файловій системі на основі FUSE. 3. Віртуальна файлова система виконує операцію unlink та видаляє посилання. 4. Система повідомляє користувача про успішне видалення посилання.
Extension	У випадку, якщо видалення посилання неможливе (наприклад, через відсутність прав), система повідомляє користувача про помилку.
Post-Condition	Користувач отримує підтвердження про видалення посилання.

Таблиця 1.10 Варіант використання UC-06

Use Case ID	UC-06
Use Case Name	Видалення папок, файлів (rmdir)
Goals	Видалити об'єкт (папку, файл) у віртуальній файловій системі
Actors	Користувач в системі
Trigger	Користувач бажає видалити об'єкт
Pre-conditions	Користувач має доступ до системи та віртуальної файлової системи
Flow of Events	1. Користувач викликає команду для видалення об'єкта. 2. Система передає запит на обробку віртуальній файловій системі на основі FUSE. 3. Віртуальна файлова система виконує операцію rmdir та видаляє об'єкт. 4. Система повідомляє користувача про успішне видалення

	об'єкта.
Extension	У випадку, якщо видалення об'єкта неможливе (наприклад, через відсутність прав), система повідомляє користувача про помилку.
Post-Condition	Користувач отримує підтвердження про видалення об'єкта.

Таблиця 1.11 Варіант використання UC-07

Use Case ID	UC-07
Use Case Name	Створення soft-посилань (symlink)
Goals	Створити символьне посилання на об'єкт у віртуальній файловій системі
Actors	Користувач в системі
Trigger	Користувач бажає створити символьне посилання
Pre-conditions	Користувач має доступ до системи та віртуальної файлової системи
Flow of Events	1. Користувач викликає команду для створення символьного посилання. 2. Система передає запит на обробку віртуальній файловій системі на основі FUSE. 3. Віртуальна файлова система виконує операцію symlink та створює символьне посилання. 4. Система повідомляє користувача про успішне створення символьного посилання.
Extension	У випадку, якщо створення символьного посилання неможливе (наприклад, через відсутність прав), система повідомляє користувача про помилку.
Post-Condition	Користувач отримує підтвердження про створення символьного посилання.

Таблиця 1.12 Варіант використання UC-08

Use Case ID	UC-08
Use Case Name	Зміна дозволів файла (chmod)
Goals	Змінити права доступу до файла віртуальної файлової системи
Actors	Користувач в системі
Trigger	Користувач бажає змінити права доступу до файла
Pre-conditions	Користувач має доступ до системи та віртуальної файлової системи
Flow of Events	1. Користувач викликає команду для зміни прав доступу до файла. 2. Система передає запит на обробку віртуальній файловій системі на основі FUSE. 3. Віртуальна файлова система виконує операцію chmod та змінює права доступу до файла. 4. Система повідомляє користувача про успішну зміну прав доступу до файла.
Extension	У випадку, якщо зміна прав доступу неможлива (наприклад, через відсутність прав), система повідомляє користувача про помилку.
Post-Condition	Користувач отримує підтвердження про зміну прав доступу до файла.

Таблиця 1.13 Варіант використання UC-09

Use Case ID	UC-9
Use Case Name	Зчитування файла (read)
Goals	Отримати вміст файла віртуальної файлової системи
Actors	Користувач в системі
Trigger	Користувач бажає прочитати вміст файла
Pre-conditions	Користувач має доступ до системи та віртуальної файлової системи

Flow of Events	1. Користувач викликає команду для зчитування вмісту файла. 2. Система передає запит на обробку віртуальній файловій системі на основі FUSE. 3. Віртуальна файлова система виконує операцію read та повертає вміст файла. 4. Система повідомляє користувача про отримання вмісту файла.
Extension	У випадку, якщо зчитування файла неможливе (наприклад, через відсутність прав), система повідомляє користувача про помилку.
Post-Condition	Користувач отримує вміст файла.

Таблиця 1.14 Варіант використання UC-10

Use Case ID	UC-10
Use Case Name	Редагування файла (write)
Goals	Змінити вміст файла віртуальної файлової системи
Actors	Користувач в системі
Trigger	Користувач бажає змінити вміст файла
Pre-conditions	Користувач має доступ до системи та віртуальної файлової системи
Flow of Events	1. Користувач викликає команду для редагування вмісту файла. 2. Система передає запит на обробку віртуальній файловій системі на основі FUSE. 3. Віртуальна файлова система виконує операцію write та змінює вміст файла. 4. Система повідомляє користувача про успішне редагування вмісту файла.
Extension	У випадку, якщо редагування вмісту файла неможливе (наприклад, через відсутність прав), система повідомляє користувача про помилку.
Post-Condition	Користувач отримує підтвердження про зміну вмісту файла.

Таблиця 1.15 Варіант використання UC-11

Use Case ID	UC-11
Use Case Name	Зчитування папки (readdir)
Goals	Отримати перелік об'єктів у вказаній директорії віртуальної файлової системи
Actors	Користувач в системі
Trigger	Користувач бажає переглянути вміст директорії
Pre-conditions	Користувач має доступ до системи та віртуальної файлової системи
Flow of Events	1. Користувач викликає команду для зчитування вмісту директорії. 2. Система передає запит на обробку віртуальній файловій системі на основі FUSE. 3. Віртуальна файлова система виконує операцію readdir та повертає перелік об'єктів у директорії. 4. Система повідомляє користувача про отримання переліку об'єктів.
Extension	У випадку, якщо зчитування директорії неможливе (наприклад, через відсутність прав), система повідомляє користувача про помилку.
Post-Condition	Користувач отримує перелік об'єктів у директорії.