НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
«Системи і методи штучного інтелекту»
спеціальності 122 «Комп'ютерні науки»
на тему: «Особливості будови сучасних файлових систем»

Виконав:
студент IV курсу, групи КІ-11
Драган Богдан Володимирович _____

Керівник:
доцент кафедри ШІ, к. т. н., доцент,
Коваленко Анатолій Єпіфанович _____

Консультант з економічного розділу:
доцент кафедри ЕК ФММ, к.е.н., доцент,
Рощина Надія Василівна _____

Консультант з нормоконтролю:
фахівець першої категорії кафедри ШІ, к.т.н., доцент,
Комариста Богдана Миколаївна _____

Рецензент:
доцент кафедри ММСА, к.т.н., доцент,
Зінченко Артем Юрійович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В. о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«15» січня 2025 р.

ЗАВДАННЯ
на дипломну роботу студенту
Драгану Богдану Володимировичу

1. Тема роботи «Особливості будови сучасних файлових систем», керівник роботи Коваленко Анатолій Єпіфанович, к.т.н., доцент кафедри ШІ, затверджені наказом по НН ІПСА від «26» травня 2025 р. № 1759-с.

2. Термін подання студентом роботи «9» червня 2025 року.

3. Вихідні дані до роботи

4. Зміст роботи: дослідження теоретичних основ файлових систем, аналіз архітектури, структурних елементів та принципів організації зберігання даних; порівняння функціональних можливостей, технічних характеристик, обмежень і сумісності сучасних файлових систем; аналіз методів застосування штучного інтелекту у файлових системах та програмна реалізація інструменту для аналізу структури директорій і вмісту файлів; аналіз отриманих результатів; функціонально-вартісний аналіз програмного продукту.

5. Перелік ілюстративного матеріалу: презентація до захисту роботи.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Надія Василівна, доцент, к. е. н.		

7. Дата видачі завдання «03» лютого 2025 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Підготовка даних до роботи	21.04.2025	Виконано
2	Вивчення літератури за темою роботи	28.04.2025	Виконано
3	Підготовка першого розділу	05.05.2025	Виконано
4	Підготовка другого розділу	12.05.2025	Виконано
5	Підготовка третього розділу	19.05.2025	Виконано
6	Підготовка четвертого розділу	26.05.2025	Виконано
7	Оформлення розділів дипломної роботи	02.06.2025	Виконано
8	Оформлення презентаційних матеріалів до захисту	09.06.2025	Виконано

Студент Богдан ДРАГАН

Керівник Анатолій КОВАЛЕНКО

РЕФЕРАТ

Дипломна робота: 118 с., 27 рис., 22 табл., 48 посилань, 1 додаток.

ФАЙЛОВА СИСТЕМА, ОПЕРАЦІЙНА СИСТЕМА, ФАЙЛ, СТРУКТУРА, КАТАЛОГ, МЕТАДАНІ, БЛОК, ШТУЧНИЙ ІНТЕЛЕКТ.

Об’єкт дослідження - файлові системи як компонент зберігання та організації даних в операційних системах.

Предмет дослідження - архітектурні, структурні й функціональні особливості сучасних файлових систем. Методи застосування штучного інтелекту у файлових системах.

Метою роботи є аналіз та дослідження будови сучасних файлових систем, виявлення ключових архітектурних рішень і особливостей їхньої реалізації, а також дослідження можливостей використання штучного інтелекту у файлових системах.

Актуальність теми зумовлена зростанням обсягів даних і поширенням нових типів пристроїв зберігання. Дослідження будови файлових систем дозволить глибше зрозуміти принципи роботи сучасних операційних систем, підвищити продуктивність обчислювальних процесів і забезпечити надійність зберігання інформації.

У роботі досліджено особливості будови сучасних файлових систем, зокрема архітектурні принципи, структури зберігання даних, методи адресації та керування простором. Проведено порівняння їхніх функціональних і технічних характеристик. Розглянуто підходи до впровадження штучного інтелекту у файлових системах. Розроблено програму, яка здійснює інтелектуальний аналіз вмісту директорій, визначає ключові характеристики файлів і формує аналітичний звіт.

ABSTRACT

Bachelor's thesis: 118 p., 27 figures, 22 tables, 48 references, 1 appendix.

FILE SYSTEM, OPERATING SYSTEM, FILE, STRUCTURE, DIRECTORY, METADATA, BLOCK, ARTIFICIAL INTELLIGENCE.

The object of research is file systems as a component of data storage and organization in operating systems.

The subject of research is the architectural, structural and functional features of modern file systems. Methods of applying artificial intelligence in file systems.

The purpose of the work is to analyze and study the structure of modern file systems, identify key architectural solutions and features of their implementation, as well as study the possibilities of using artificial intelligence in file systems.

The relevance of the topic is due to the growth of data volumes and the spread of new types of storage devices. Research into the structure of file systems will allow a deeper understanding of the principles of operation of modern operating systems, increase the productivity of computing processes and ensure the reliability of information storage.

The work investigates the structural features of modern file systems, in particular, architectural principles, data storage structures, addressing methods and space management. Their functional and technical characteristics are compared. Approaches to the implementation of artificial intelligence in file systems are considered. A program has been developed that performs intelligent analysis of directory contents, determines key file characteristics, and generates an analytical report.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ФС – файлова система

ОС – операційна система

MBR – Master Boot Record

GPT – GUID Partition Table

MFT – Master File Table

CoW – Copy-on-Write

i-node – index node

ШІ – штучний інтелект

LLM – Large Language Model

ВСТУП

Зі стрімким розвитком інформаційних технологій, зокрема систем зберігання даних, обсяг цифрової інформації, постійно зростає. Це зумовлює необхідність у надійних та ефективних механізмах організації даних, які здатні забезпечити швидкий доступ, структуроване зберігання та захист інформації. Одним із фундаментальних компонентів у цьому процесі є файлова система, яка виступає основою для роботи з даними в усіх типах обчислювальних систем.

Останніми роками розвиток комп'ютерної інфраструктури супроводжується зростанням вимог до зберігання даних. Збільшення обчислювальної потужності, поширення цифрових пристроїв і кількості користувачів створюють додаткове навантаження на системи, що актуалізує потребу в удосконаленні методів керування даними. Сучасні файлові системи орієнтуються на оптимальні рішення, а однією з ключових тенденцій стає інтеграція інструментів штучного інтелекту для підвищення ефективності та надійності інформаційних систем.

Ця робота присвячена аналізу будові сучасних файлових систем в користувацьких версіях операційних системах, для того, щоб покращити розуміння їх внутрішньої структури, а також перспективам використання штучного інтелекту у сфері організації та управління даними[1].

[1]Тут і нижче використано такий інструмент штучного інтелекту як чат-бот з генеративним штучним інтелектом ChatGPT виключно для корегування та редагування тексту, створеного автором цієї дипломної роботи, на основі автоматизованої перевірки граматики, структури та стилю, що відповідає Політиці використання штучного інтелекту для академічної діяльності в КПІ ім. Ігоря Сікорського (протокол №11 Вченої ради КПІ ім. Ігоря Сікорського від 11 грудня 2023 р.).

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ФАЙЛОВИХ СИСТЕМ

1.1 Поняття файлової системи

Для більшості користувачів файлова система є найбільш помітним аспектом операційної системи загального призначення. Вона забезпечує структурований спосіб зберігання, упорядкування та керування даними на пристроях зберігання даних, таких як жорсткі диски, SSD-накопичувачі та USB-накопичувачі. По суті, файлова система діє як міст між операційною системою та фізичним обладнанням для зберігання даних, дозволяючи користувачам і програмам створювати, читати, оновлювати та видаляти файли організованим та ефективним чином.

Файлова система складається з двох окремих частин: набору файлів, кожен з яких зберігає пов'язані дані, та структури каталогів, яка організовує та надає інформацію про всі файли в системі [1, 2].

1.2 Організація файлів

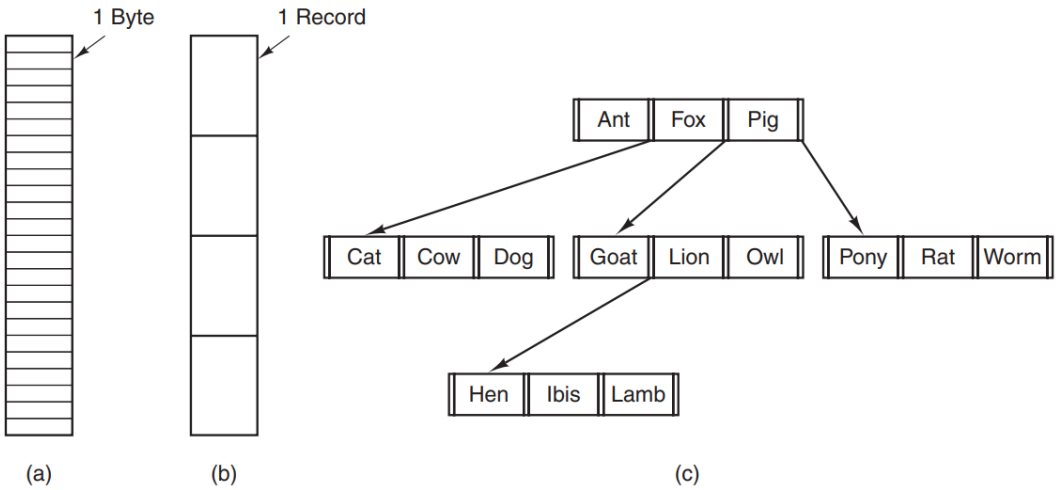
Комп'ютери можуть зберігати інформацію на різних носіях, таких як жорсткі диски, магнітні стрічки, CD-ROM, CD-DVD, флеш-пам’ять тощо. Щоб комп'ютерною системою було зручно користуватися, операційна система забезпечує єдине логічне представлення інформації, що зберігається. Операційна система абстрагується від фізичних властивостей пристроїв зберігання даних, щоб визначити логічну одиницю зберігання - файл. Пристрої зазвичай містять тисячі або навіть мільйони файлів, кожен з яких не залежить від інших.

Зазвичай файли представляють програми та дані. Файли даних можуть бути числовими, алфавітними, алфавітно-цифровими або двійковими.

Загалом файл - це послідовність бітів, байтів, рядків або записів, значення яких визначається творцем файлу та користувачем. Таким чином, поняття файлу є надзвичайно загальним [1].

1.2.1 Структура файлів

Основними способами структурування файлів є подання за допомогою трьох типів структур (рис. 1.1): у вигляді послідовності байтів, послідовності записів і дерева [3].



(a) Послідовність байт (b) Послідовність записів (c) Дерево

Рисунок 1.1 – Три типи файлів [3]

Структура файлової системи у вигляді послідовності байтів надає ОС максимальної гнучкості, але ускладнює систему керування файлами. Файли, подані послідовністю записів являють собою лінійні набори записів фіксованої довжини і однієї структури. Дерево являє собою дерево записів не обов’язково однакової довжини. Кожний запис у фіксованій позиції має ключ, який дозволяє проводити швидкий пошук у дереві.

Кожна ОС підтримує певні типи файлів. UNIX і Windows мають регулярні файли і каталоги. UNIX також має символьні та блокові спеціальні файли. Регулярні файли містять інформацію користувача, наприклад у вигляді дерева записів.

Каталоги являють собою системні файли, які підтримують структуру файлової системи. Спеціальні символьні файли використовуються для моделювання послідовних пристроїв введення/виведення, таких як термінали, принтери та мережі. Блочні спеціальні файли використовують для моделювання дисків.

Регулярні файли являють собою переважно ASCII-файли, які складаються із текстових рядків (не обов’язково однакової довжини), і двійкові файли. Завершення рядка є символом, зокрема символом переведення рядка для UNIX, символом повернення каретки для багатьох інших систем або обидва символи (наприклад, Windows). Файли ASCII зручніші для організації конвеєрів. Двійкові файли мають структуру, відому програмі, яка їх обробляє [3].

1.2.2 Атрибути та операції з файлами

Кожен файл має ім'я та свої дані. Крім того, всі операційні системи пов'язують з кожним файлом іншу інформацію, наприклад, дату і час останньої зміни файлу та його розмір. Ці додаткові елементи називаються атрибутами файлу. Іноді їх називають метаданими.

Атрибути файлу залежать від конкретної ОС, але зазвичай складаються з таких елементів, як ім’я файлу, унікальний ідентифікатор, тип, розмір, фізичне розташування, рівень доступу, а також дати створення і останньої модифікації [1].

Файли існують для того, щоб зберігати інформацію і давати змогу її пізніше знайти. Різні системи забезпечують різні операції для зберігання та пошуку інформації. Типовими операціями є створення, видалення, відкриття, закриття, зчитування, запис, отримання атрибутів та перейменування файлів.

Це базові операції складають мінімальний набір необхідних дій над файлами. Їх можна комбінувати для виконання інших, більш складних, операцій з файлами [3].

1.3 Структура каталогів у файлових системах

Каталоги (директорії, directories, папки) містить інформацію про файли, включаючи атрибути, розташування та власника. Більша частина цієї інформації, особливо та, що стосується зберігання, управляється операційною системою. Каталог сам по собі є файлом, доступним для різних процедур управління файлами. З точки зору користувача, каталог забезпечує відповідність між іменами файлів, відомими користувачам і програмам, та самими файлами [4].

Серед можливих структур каталогів виділяють однорівневу, дворівневу та ієрархічну (деревоподібну).

Однорівнева структура каталогів є найпростішою, оскільки всі файли розташовуються у єдиному каталозі. У такій структурі всі імена файлів повинні бути унікальними, що створює певні незручності при збільшенні кількості файлів або при використанні системи багатьма користувачами.

Дворівнева структура каталогів дозволяє розмежовувати файли між різними користувачами, де кожен користувач має власний окремий каталог. Головний каталог містить посилання на всі каталоги користувачів, іменовані, як правило, іменами відповідних користувачів. Відповідно, файли користувачів розташовуються у їхніх власних каталогах. Це вирішує

проблему конфлікту імен, дозволяючи використовувати однакові імена файлів у різних каталогах користувачів, але обмежує спільний доступ до файлів.

Ієрархічна (деревоподібна) структура каталогів є більш поширеною у сучасних файлових системах. Вона дозволяє організовувати файли і каталоги у вигляді ієрархічної структури з кореневим каталогом та підкаталогами різних рівнів, як зображено на рис. 1.2.

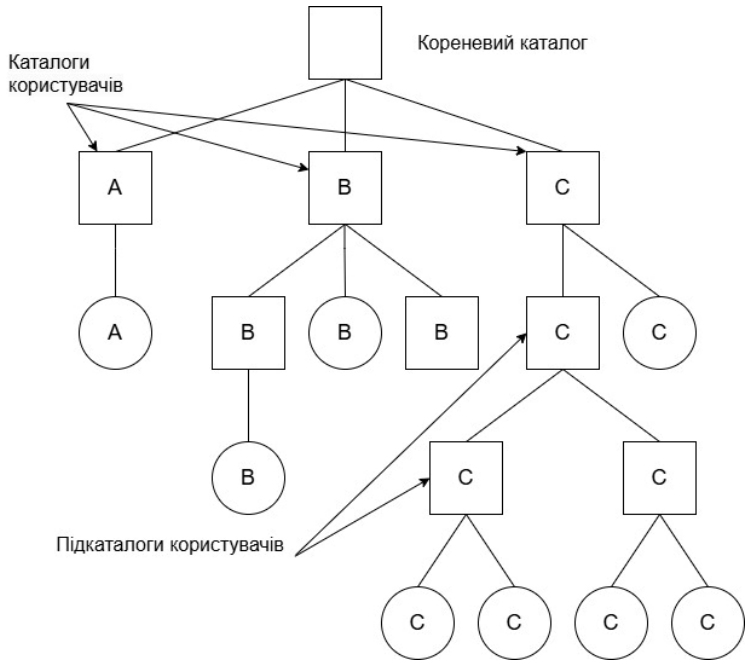


Рисунок 1.2 – Ієрархічний каталог [3]

Ієрархічна структура забезпечує гнучке і зручне управління файлами, їх пошук та організацію, наприклад, за такими ознаками як: електронна пошта, тексти програм, документи тощо.

В ієрархічній структурі використовуються абсолютні та відносні шляхи до файлів. Абсолютний шлях починається з кореневого каталогу та включає всю послідовність каталогів до файлу. Відносний шлях визначається від поточного каталогу користувача, що дозволяє швидко переміщуватись і працювати з файлами без необхідності щоразу вказувати повний шлях.

Системні виклики до каталогів можуть суттєво розрізнятись в залежності від ОС. Наприклад, для UNIX важливими командами є створення (Create), вилучення (Delete), відкриття (Opendir), закритття (Closedir) каталога, зчитування наступного елемента відкритого каталогу (Readdir), перейменування (Rename), зв'язування (Link). Зокрема, Link дозволяє з’являтися одному файлу в кількох каталогах за допомогою лічильників і-вузлів (жорсткий зв’язок). Вилучення посилання на файл з каталогу виконує команда UnLink. Є також інші системні виклики, зокрема призначені для захисту даних у файлах [3].

1.4 Структурні одиниці простору зберігання у файлових системах

У процесі організації зберігання даних на фізичних носіях комп’ютерних систем використовуються різні логічні одиниці, серед яких основними є розділ, кластер та сектор. Розуміння цих компонентів є необхідним для подальшого аналізу будови файлових систем, що буде здійснено в наступному розділі.

Сектор (sector) - це основна одиниця зберігання даних на диску. На більшості пристроїв зазвичай визначено розмір сектора 512 байт, у сучасних пристроях усе частіше використовується розмір сектора 4096 байт. Усі операції читання та запису на рівні апаратного забезпечення відбуваються саме по секторах.

Кластер (або блок; cluster/block) – це набір секторів. Розмір кластера завжди є кратним розміру сектору. Формати файлових систем використовують кластери для ефективнішого керування простором зберігання; розмір кластера, більший за розмір сектора. Кожен файл займає принаймні один кластер, незалежно від свого фактичного розміру. У

випадку, коли обсяг файлу менший за розмір виділеного кластера, невикористана частина простору не може бути задіяна іншими файлами.

Розділ (або том; partition/volume) - це набір суміжних секторів на пристрої. Розділи створюються до встановлення файлової системи. Файлова система створюється всередині розділу. Таблиця розділів або інша база даних керування розділами зберігає початковий сектор розділу, його розмір та інші характеристики і знаходиться на тому ж фізичному пристрої, що і сам розділ [5].

1.5 Реалізація файлових систем

Реалізація файлової системи визначає спосіб організації зберігання інформації на фізичних носіях та механізми управління цими даними операційною системою.

1.5.1 Структура файлової системи

Файлова система розміщується на диску, який може бути поділений на розділи з окремими файловими системами. Історично склалося, що для керування розділами використовувалася структура Master Boot Record (MBR). Перший сектор диска містить головний завантажувальний запис (Master Boot Record), що включає таблицю розділів і визначає активний розділ для запуску системи. Коли комп'ютер завантажується, BIOS зчитує і виконує MBR. Перше, що робить програма MBR, це знаходить активний розділ, зчитує його перший блок, який називається завантажувальним

блоком, і виконує його. Програма у завантажувальному блоці завантажує операційну систему, що міститься у цьому розділі.

Більш сучасним механізмом розмітки дисків є GUID Partition Table (GPT), який прийшов на заміну MBR. Він дозволяє створювати значно більшу кількість розділів та підтримує диски обсягом понад 2 ТБ. На відміну від MBR, GPT зберігає резервні копії таблиці розділів на початку та в кінці диска, що підвищує надійність структури. Під час створення кожного розділу для нього генерується унікальний глобальний ідентифікатор (GUID), що гарантує його однозначну ідентифікацію.

Структура файлової системи розділу зазвичай включає такі компоненти: суперблок, який містить загальні параметри файлової системи, систему керування вільним місцем, таблицю i-nodes для зберігання метаданих файлів, кореневу директорію та простір для інших файлів і каталогів, як показано на рис. 1.3. Незалежно від типу файлової системи, структура зберігає логічну послідовність [3, 6].

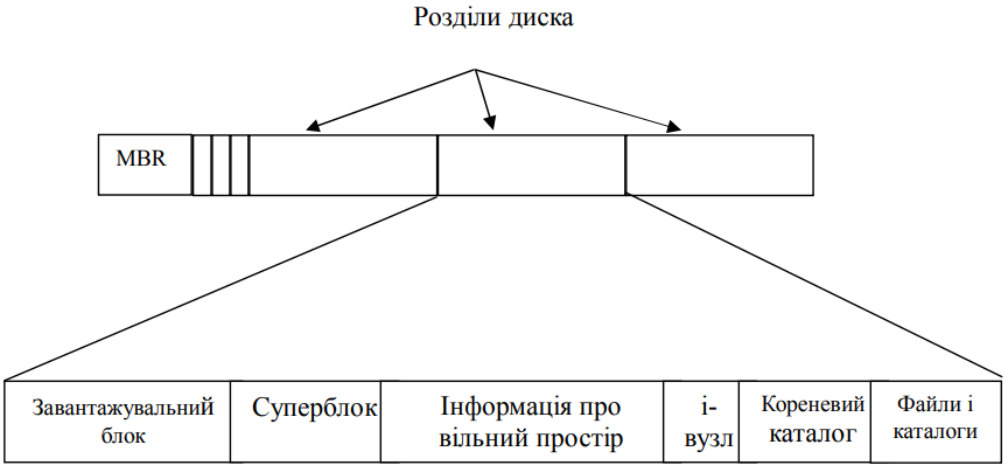


Рисунок 1.3 – Архітектура файлової системи [7]

1.5.2 Методи фізичного розміщення файлів

Організація фізичного розміщення файлів на диску визначає, як логічна структура файлу відображається на послідовність фізичних блоків пристрою зберігання. В операційних системах найчастіше застосовуються три базові методи: неперервний розподіл**,** списковий або зв'язаний розподіл та розподіл на основі індексних вузлів**.**

Неперервний розподіл (contiguous allocation) полягає у зберіганні кожного файлу як послідовності блоків на диску. Цей підхід має дві суттєві переваги. По-перше, його просто реалізувати, оскільки відстеження розташування блоків файлу зводиться до запам'ятовування двох чисел: дискової адреси першого блоку та кількості блоків у файлі. По-друге, продуктивність читання відмінна, тому що весь файл може бути прочитаний з диска за одну операцію. Потрібен лише один пошук (до першого блоку). Однак цей метод створює проблему фрагментації диску. При видаленні файлів у дисковому просторі залишаються «дірки», які важко повторно використати, якщо новий файл не поміщається у наявну послідовність блоків. На рис. 1.4 показано загалом сім файлів, кожен з яких починається з наступного блоку після закінчення попереднього.

Зв’язаний розподіл (linked-list allocation) базується на зберіганні файлу у вигляді ланцюжка блоків, де кожен блок містить вказівник на наступний блок (рис. 1.5). На відміну від неперервного розподілу, в цьому методі може бути використаний кожен блок диска. При фрагментації диска не втрачається місце (за винятком внутрішньої фрагментації в останньому блоці). Крім того, для запису каталогу достатньо лише зберігати адресу диска першого блоку. Решту можна знайти, починаючи з нього.

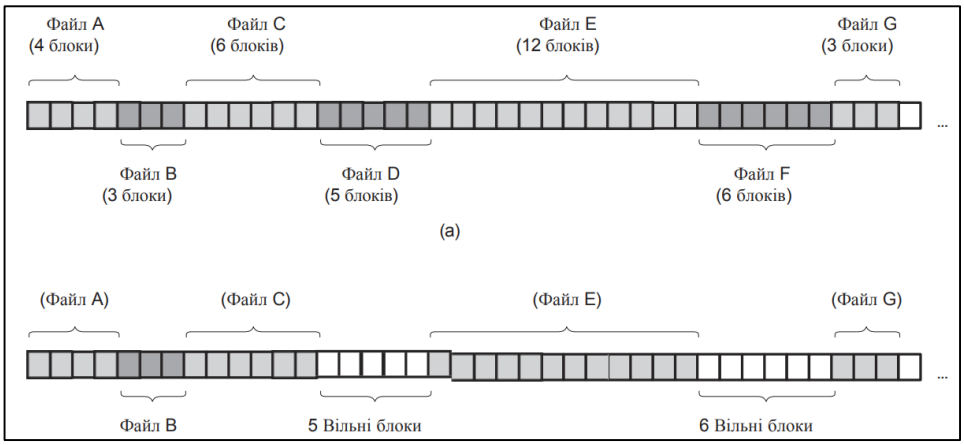


Рисунок 1.4 – Неперервний розподіл блоків файлу [3]

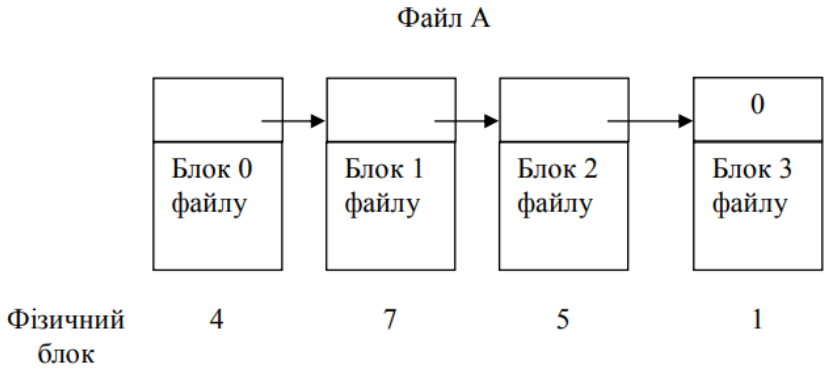


Рисунок 1.5 – Ланцюговий розподіл блоків файлу [7]

Основним недоліком є повільний доступ до довільного блоку, оскільки для досягнення конкретної позиції необхідно пройти ланцюг від початку. Крім того, кожен блок втрачає частину об’єму на зберігання вказівника, а в разі пошкодження одного блоку може бути втрачено доступ до решти файлу.

Індексний розподіл (i-node, index node, і-вузол) використовує спеціальний індексний блок, який містить вказівники на фізичні блоки файлу (рис. 1.6).

Це дозволяє здійснювати швидкий доступ до будь-якої частини файлу, однак для малих файлів може бути неефективним. Якщо кожен і-вузол займає n байт і одночасно може бути відкрито максимум k файлів, то

загальна пам'ять, яку займає масив, що містить і-вузли для відкритих файлів, становить лише k*n байт [3].

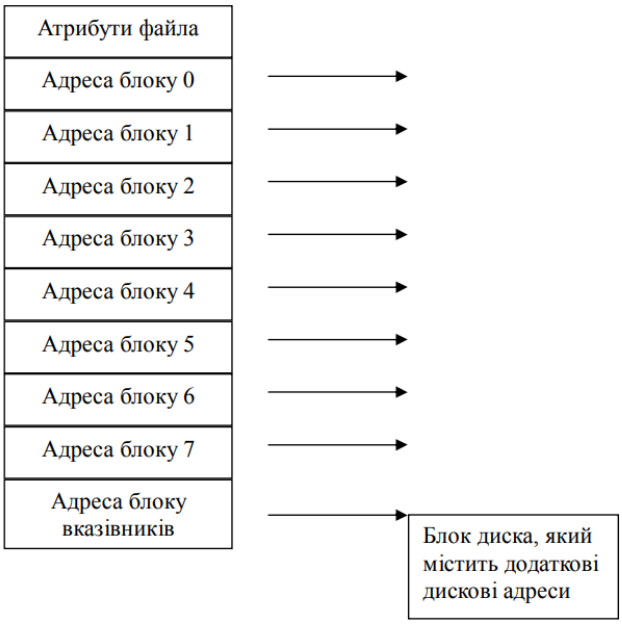


Рисунок 1.6 – Індексний розподіл блоків файлу [3]

1.5.3 Механізми керування вільним простором

Кожна файлова система має механізм відстеження вільного простору на диску. Найпоширенішими способами є:

- бітові карти (bitmaps);
- зв'язані списки (linked lists);
- групування (grouping);
- підрахунок (counting).

Бітові карти використовують для кожного блоку один біт для позначення стану блоку (зайнятий чи вільний). Наприклад, розглянемо диск, на якому блоки 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26 і 27 є вільними, а

решта блоків розподілені. Бітове зображення вільного простору матиме такий вигляд:

00111100111111000110000011100000 ...

Основною перевагою цього підходу є його відносна простота та ефективність у пошуку першого вільного блоку або n послідовних вільних блоків на диску.

Зв’язний список полягає у зв'язуванні всіх вільних блоків, зберігаючи вказівник на перший вільний блок у спеціальному місці у файловій системі та кешуючи його в пам'яті. Цей перший блок містить вказівник на наступний вільний блок і так далі. У прикладі, у якому блоки 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26 і 27 були вільними, а решта блоків були розподілені, ми збережемо вказівник на блок 2 як перший вільний блок. Блок 2 містив би вказівник на блок 3, який вказував би на блок 4, який вказував би на блок 5, який вказував би на блок 8, і так далі. Даний підхід не є ефективним, оскільки, щоб пройти по списку, ми повинні прочитати кожен блок, що вимагає значного часу вводу/виводу на жорстких дисках.

Групуванням є модифікацією підходу зв’язного списку зберігає адреси n вільних блоків у першому вільному блоці. Перші n-1 з цих блоків дійсно вільні. Останній блок містить адреси ще n вільних блоків і так далі. Адреси з великої кількості вільних блоків тепер можна швидко знайти, на відміну від ситуації, коли використовується стандартний підхід linked-list.

Метод підрахунку використовує той факт, що, як правило, декілька суміжних блоків можуть бути виділені або звільнені одночасно, особливо коли простір розподіляється за алгоритмом суміжного розподілу або за допомогою кластеризації. Таким чином, замість того, щоб зберігати список з n адрес вільних блоків, ми можемо зберігати адресу першого вільного блоку і кількість (n) вільних суміжних блоків, які слідують за першим блоком. Кожен запис у списку вільних блоків складається з адреси пристрою та кількості блоків. Хоча кожен запис потребує більше місця, ніж проста адреса

диска, загальний список буде коротшим, якщо кількість елементів буде більшою за 1 [1].

1.6 Функціональні можливості сучасних файлових систем

Сучасні файлові системи забезпечують не лише базове зберігання та організацію даних, а й широкий спектр функціональних можливостей, які покращують продуктивність, безпеку та надійність роботи з інформацією. До них можна віднести:

- шифрування;
- стиснення;
- журналювання;
- копіювання при записі (Copy-on-Write);
- знімки.

Шифрування - процес, який використовує криптографічний алгоритм і ключ для перетворення даних відкритого тексту в дані зашифрованого тексту. Вміст файлу може зберігатися в зашифрованому вигляді для захисту від несанкціонованого доступу. Шифрування може застосовуватися програмою, яка створює файл, зовнішньою програмою, яка зчитує незашифрований файл і створює зашифрований файл, або операційною системою під час створення файлу. Перед записом файлу на диск операційна система шифрує файл і зберігає зашифрований текст в одиницях даних. Дані, що не є вмістом, такі як ім'я файлу та час останнього доступу, зазвичай не шифруються.

Деякі файлові системи підтримують зберігання даних у стисненому вигляді з метою зменшення обсягу зайнятого простору на диску. Стиснення може відбуватись на трьох рівнях. Найвищий рівень — стиснення в межах самого формату файлу, як-от у JPEG, де стискаються лише дані зображення,

тоді як заголовок залишається незмінним. Наступний рівень — стиснення всього файлу за допомогою зовнішньої програми, яка створює окремий файл, що потребує розпакування перед використанням. Найнижчий рівень — стиснення на рівні файлової системи, яке є прозорим для прикладного програмного забезпечення. Існують дві основні техніки такого стиснення. Перша полягає у застосуванні стандартних методів стиснення безпосередньо до блоків даних. Друга — у відмові від виділення фізичного блоку, якщо він повністю заповнений нулями. Такі файли називають розрідженими.

Журналювання - це механізм підвищення надійності файлової системи, який дозволяє операційній системі швидко відновлювати узгоджений стан файлової системи після збою. Його суть полягає у записі змін метаданих у спеціальний журнал до їх фактичного застосування. Якщо система зазнає збою до завершення операції, журнал дозволяє визначити, які зміни не були завершені, і повернути файлову систему до стабільного стану. Журналування зменшує ризик пошкодження структури файлової системи при неочікуваному завершенні роботи системи [8].

Копіювання при записі (CoW) - це політика, яка використовується в багатьох сучасних файлових системах, де копія ресурсу створюється при зміні оригіналу. Це означає, що оригінал часто залишається у файловій системі.

Знімок (Snapshot) - це зображення файлової системи в певний момент часу. Багато сучасних файлових систем дозволяють створювати знімки. Зазвичай це досягається за допомогою копіювання при записі. Ці знімки використовуються як механізм резервного копіювання і тому можуть показати старі версії файлової системи [9].

1.7 Постановка задачі

Задачею цієї дипломної роботи є проведення комплексного дослідження особливостей будови сучасних файлових систем, які використовуються в актуальних версіях операційних систем. У ході дослідження передбачається виявити ключові архітектурні принципи, що лежать в основі організації ФС, та визначити, яким чином реалізуються базові функції управління файлами. Крім того, суттєве значення приділиться дослідженню використання штучного інтелекту у файлових системах.

Висновки до розділу 1

У результаті проведеного дослідження встановлено, що файлова система є важливим компонентом операційних систем, який відповідає за організацію, зберігання та управління даними на фізичних носіях. Вона забезпечує користувачам і додаткам єдине логічне представлення інформації незалежно від фізичних особливостей пристроїв зберігання.

Розглянуто структуру та типи файлів, серед яких виділено три основні способи організації: у вигляді послідовності байтів, послідовності записів і деревоподібних структур.

Досліджено також поняття каталогів і їх роль у структуризації даних. Встановлено, що найпоширенішою моделлю організації каталогів у сучасних операційних системах є ієрархічна (деревоподібна) структура.

Проаналізовано компоненти організації простору зберігання у файлових системах, серед яких основними є сектор, кластер, розділ.

Проведено аналіз методів фізичного розміщення файлів та механізмів керування вільним простором. Визначено особливості, переваги та недоліки кожного підходу.

Досліджено функціональні можливості сучасних файлових систем, зокрема, шифрування, стиснення, журналювання, копіювання при записі та знімки.

Таким чином, у першому розділі закладено теоретичну основу для подальшого комплексного дослідження будови сучасних файлових систем.

РОЗДІЛ 2 КОМПЛЕКСНЕ ДОСЛІДЖЕННЯ БУДОВИ СУЧАСНИХ ФАЙЛОВИХ СИСТЕМ

У цьому розділі здійснено комплексне дослідження будови конкретних сучасних файлових систем, з метою виділення сучасних рішень, особливостей та порівняння.

На сьогодні існує велика кількість файлових систем – від традиційних дискових, до сучасних відмовостійких. Проте, найбільший інтерес представляють ті файлові системи, які зустрічаються в основних операційних системах: Windows, MacOS і Linux (включаючи Android). З цієї причини комплексне дослідження зроблено на основі файлових систем FAT і NTFS для Windows, ext, Btrfs і F2FS для Linux (Android), а також APFS для macOS та iOS. У таблиці 2.1 представлено загальну інформацію про ці файлові системи. Вона дає уявлення про їхній вік, початкове програмне середовище та компанію-розробника.

Таблиця 2.1 – Загальна інформація про кожну файлову систему [10]

ФС	Рік випуску	Оригінальна ОС	Розробник
FAT32	1996	Windows 95 OSR2	Microsoft
exFAT	2006	Windows Embedded CE 6.0	Microsoft
NTFS	1993	Windows NT 3.1	Microsoft
NTFS v3.1	2001	Windows XP	Microsoft
Ext3	1999	Linux	Stephen Tweedle
Ext4	2006	Linux	Різні
APFS	2017	macOS High Sierra, iOS 10.3	Apple Inc.
Btrfs	2009	Linux	Chris Mason
F2FS	2012	Linux (Android)	Samsung Electronics

2.1 Файлові системи FAT

FAT (File Allocation Table, Таблиця розподілу файлів) - це архітектура файлової системи, яка використовується на багатьох комп'ютерних системах і носіях пам’яті, таких як USB-накопичувачі. Завдяки своїй відносній простоті та, незважаючи на свій вік, ця архітектура досі широко використовується на картах флеш-пам'яті, цифрових фотоапаратах і багатьох інших портативних пристроях. FAT також була дуже поширеною на жорстких дисках в епоху DOS і Windows 9x, але її використання на пристроях цього типу зменшилося з появою Windows XP, яка почала використовувати новішу файлову систему NTFS [11].

Існує різні версії FAT: FAT12, FAT16, FAT32 та exFAT [8]. Ці версії відрізняються в основному розміром доступного адресного простору. У FAT12 є 2^{12} кластерів, 2^{16} у FAT16 і 2^{28} у FAT-32 [9].

Найсучаснішою версією у сімействі FAT є файлова система exFAT. Її характеристики та структуру розглянемо в цьому підрозділі.

2.1.1 exFAT

exFAT (Extensible FAT, Розширений FAT) - це остання версія сімейства FAT, представлена Microsoft у 2006 році, призначена як для заміни старої версії FAT32 і пов'язаних з нею обмежень, так і для задоволення сучасних вимог до файлової системи [12]. Вона має схожу конструкцію, але містить багато суттєвих покращень:

- більші обмеження щодо обсягу та розміру файлів;
- імена файлів у форматі Unicode;
- краща продуктивність;

- підтримка зміщення часового поясу [13].

Основні характеристики ФС exFAT, в порівнянні з минулою версією FAT32, представлені в таблиці 2.2.

Станом на 2025 рік, exFAT це єдина, сучасна, ФС що «з коробки» підтримується всіма популярними ОС — Windows, Linux, MacOS, Android, iOS [14]. Зазвичай використовується на картах пам'яті для цифрових камер або для розширення пам'яті телефонів Android [12].

Таблиця 2.2 – Порівняння характеристик FAT32 та exFAT [15]

Характеристика	**FAT32**	**exFAT**
Максимальний розмір тома	8 ТБ (теоретично)	128 ПБ
Максимальний розмір файлу	4 ГБ	128 ПБ
Максимальний розмір кластера	32 КБ	32 МБ
Максимальна кількість кластерів	До 2^{28}	До 2^{32}
Максимальна довжина імені файлу	255	255
Роздільна здатність дати/часу	2 с	10 мс

2.1.2 Розподіл обсягу

Структура розділу exFAT, детально показана на рис. 2.1. Її можна розділити на три різні області. Перша область — це *Область Завантаження (Boot Region)*, також звана *суперблоком*, яка займає перші 512 байт кожного розділу exFAT.

Тут зберігається базова конфігурація файлової системи, яка містить інформацію про те, як інтерпретувати розділ. *Суперблок* дублюється і записується безпосередньо після оригіналу, щоб мати резервну копію на випадок пошкодження оригіналу. Наступна область — *Область FAT (FAT Region)*, яка зберігає таблицю розподілу файлів, представлену в таблиці 2.3. Остання область — *Область Даних (Data Region)*, яка зберігає, крім вмісту

файлів, також метадані файлів і папок. На рис. 2.1 зображено точні межі окремих областей.

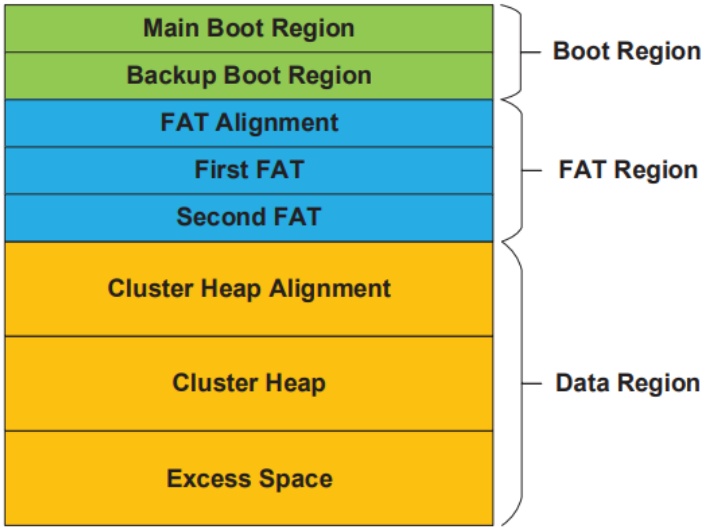


Рисунок 2.1 – Структура exFAT [12]

Таблиця 2.3 – Приклад таблиці розподілу файлів [12]

Кластер	Значення	Зміщення (Байт)
0	0xFFFFFFF8	+0
1	0xFFFFFFFF	+4
2	…	+8
…		…
40	41	+160
41	80	+164
42	…	+168
43	…	+172
80	101	+320
81	…	+324
…		…
100	…	+400
101	0xFFFFFF	+404
102	…	+408

2.1.3 Таблиця розподілу файлів

У таблиці 2.3 представлено спосіб зберігання файлів у системі за допомогою таблиці розподілу файлів. Кожен рядок таблиці відповідає одному кластеру на розділі. Перші два рядки є попередньо визначеними, оскільки перші два кластери файлової системи exFAT не можуть бути виділені користувачем. Зсув показує позицію в байтах для відповідного кластерного запису. Кожен запис має розмір 4 байти і містить значення від 2 до кількості кластерів плюс один. Два додаткові значення — 0xFFFFFFF7, яке позначає несправний кластер, і 0xFFFFFF, яке позначає кінець файлу, так званий маркер кінця файлу.

У таблиці 2.3 показано, що файл у кластері 40 продовжується в кластері 41. Таким чином, значення в кластерах можна простежити далі, доки не буде досягнуто кінця файлу в кластері 101.

Для зберігання файлів у файловій системі exFAT, крім вмісту файлу, на розділ необхідно записати метадані про файл. Суперблок містить поле RootDirectoryCluster, яке вказує кластер, в якому знаходиться коренева папка файлової системи. Цей кластер містить, крім записів даних і папок, спеціальні записи, які є важливими для exFAT, такі як ім'я розділу, яке зберігається в мітці тома.

Інформація про вільні кластери зберігається окремо — у Allocation Bitmap (Бітова мапа розміщення), що складається з байтів, кожен біт яких відповідає кластеру: 1 — зайнятий, 0 — вільний. Вільний кластер у файловій системі exFAT не обов'язково означає, що пам'ять обнулена. Щоб запобігти непотрібним операціям запису, кластер просто позначається як вільний. Це збільшує термін служби флеш-пам'яті, яка використовується, наприклад, у картах пам'яті або USB-накопичувачах.

У файловій системі exFAT використовується спеціальна структура, відома як таблиця перетворення у верхній регістр (Upcase Table), яка містить

список відповідностей між символами нижнього регістру та їхніми аналогами у верхньому регістрі згідно з таблицею Unicode. Це дозволяє файловій системі не враховувати регістр символів під час роботи з іменами файлів, що значно прискорює пошук та порівняння імен. Завдяки такому механізму, наприклад, імена "file.txt" і "FILE.TXT" розглядаються як однакові [12].

2.1.4 Файли та каталоги

Як описувалося раніше, файли зберігаються в Data Region розділу exFAT. У цій частині файлової системи існують два різних типи кластерів: кластери даних, які містять вміст файлів, і кластери метаданих, які містять інформацію про файли. Тому кластер, вказаний RootDirectoryCluster, є кластером метаданих. Кожен файл і папка, незалежно від того, порожні вони чи ні, займають принаймні один кластер.

Метадані файлу або папки складаються з трьох різних записів:

- File Directory Entry (Запис файлу);
- Stream Extension Directory Entry (Запис розширення потоку);
- File Name Directory Entry (Запис імені файлу).

Ці записи базуються на загальному шаблоні, представленому в таблиці 2.4. Перший байт кожної структури починається з поля EntryType, яке групує інформацію про запис у різні прапорці, і яке має свою внутрішню структуру представлену в таблиці 2.5. Тільки перші 5 бітів відповідають за фактичну ідентифікацію структури. Importance описує, чи є цей запис критичним для exFAT, тобто чи може файлова система функціонувати без нього. Наступне поле, а саме Category, розрізняє додаткову інформацію (довжина імені файлу, розмір файлу) та основну інформацію (дата створення, дата модифікації). Останнє поле IsUse вказує, чи використовується

структура, а отже, і файл. Якщо біт не встановлено, файл вважається видаленим [12, 16].

Таблиця 2.4 – Шаблон структури метаданів [16]

Зсув	Розмір (Байт)	Поле
0	1	EntryType
1	19	CustomDefined
20	4	FirstCluster
24	8	DataLength

Таблиця 2.5 – Внутрішня структура поля EntryType [16]

Біти	Розмір (Біт)	Поле
0-4	5	Code
5	1	Importance
6	1	Category
7	1	IsUse

Для зменшення навантаження на апаратне забезпечення і продовження термін його експлуатації, при видалені папки, змінюються тільки метадані цієї папки. Таблиця FAT і бітова мапа розміщення файлів відповідно коригуються, звільняючи пам'ять. Метадані файлів у видаленій папці не змінюються; оскільки для exFAT папка більше не існує, зв'язок із даними в цій папці також відсутній. Таким чином, можуть існувати файли, які були видалені, але все ще мають встановлений біт InUse.

Усі записи метаданих, що належать до файлу, групуються в кластер без будь-яких перерв (рис. 2.2). Перший запис для кожного файлу — це запис файлів (File Directory Entry), який містить такі основні поля:

- SetChecksum — контрольна сума всіх записів метаданих (контроль цілісності);
- FileAttributes — атрибути файлу, зокрема прапорець Directory, який вказує, чи є об’єкт файлом чи каталогом (єдина відмінність між ними в структурі записів exFAT);

- CreateTimestamp, LastModifiedTimestamp, LastAccessedTimestamp — мітки часу створення, останньої модифікації та доступу;
- Create10msIncrement, LastModified10msIncrement — уточнення часу створення та модифікації з точністю до 10 мс (діапазон значень: від 0 до 199).

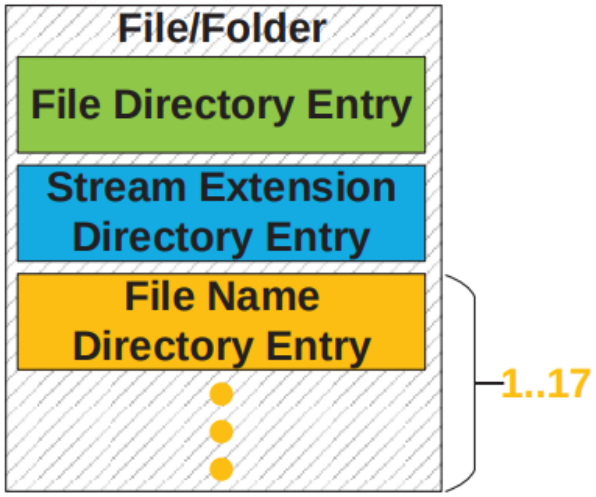


Рисунок 2.2 – Записи метаданих для файлу або папки [12].

Другий обов’язковий запис метаданих - це запис розширення потоку (Stream Extension Directory Entry), який містить технічні характеристики файлу або папки. Основні поля цього запису включають:

- ValidDataLength — фактично записаний обсяг даних;
- DataLength — загальний розмір файлу в байтах;
- FirstCluster — номер першого кластера, де починається вміст файлу;
- NameLength — довжина імені файлу (кількість символів);
- NameHash — хеш значення імені для прискореного пошуку.

Поля ValidDataLength та DataLength визначають розмір файлу або каталогу: перше вказує, скільки даних було фактично записано, друге — скільки байтів передбачено. Якщо файл було успішно записано повністю,

обидва поля будуть однаковими. Тому, коли ми говоримо про розмір файлу, ми маємо на увазі обидва поля. Для папок, як розмір, використовується обсяг метаданих у кластері. Поле FirstCluster вказує на початковий кластер із вмістом: для файлів — це самі дані, для папок — набір записів метаданих, які описують вкладені файли й підкаталоги. Саме це поле дозволяє побудувати ієрархію директорій, починаючи з головного кластера RootDirectoryCluster.

Хоча раніше визначені записи існують тільки один раз для кожного файлу, запис імен файлів (File Name Directory Entry) таблиця 2.6 - може бути присутнім до сімнадцяти разів [12]. Кожен запис може містити до 15 символів Unicode, тому максимальна довжина імені файлу — 255. Невикористану частину поля «FileName» потрібно встановити на 0x0000 [17].

Таблиця 2.6 - Запис імен файлів [17]

Зсув	Розмір (Байт)	Поле
0	1	EntryType
1	1	GeneralSecondaryFlags
2	30	FileName

2.2 Файлова система NTFS

Наступна файлова система, яку розглянемо у цьому розділі буде NTFS**.** NTFS (New Technology File System, Файлова система нової технології) - це власна файлова система, розроблена корпорацією Microsoft. Є одною з найбільш цікавих файлових систем, оскільки використовується в якості файлової системи основного диска (тобто C:) у всіх версіях Windows, починаючи з Windows NT. Вперше цю файлову систему було представлено у Windows NT 3.1 у 1993 році, і до цього часу вона залишається файловою системою Windows за замовчуванням.

Розрізняють декілька версій NTFS: v1.2 використовується в Windows NT 3.51 і Windows NT 4.0, v3.0 поставляється з Windows 2000, v3.1 — з Windows XP і Windows Server 2003. Іноді останні версії позначають як v4.0, v5.0 і v5.1 відповідно до версій Windows NT, з якими вони поставляються [9, 18].

Основні характеристики файлових систем NTFS v3.1 та NTFS порівняно з exFAT представлено в таблиці 2.7.

Таблиця 2.7 – Порівняння характеристик NTFS v3.1, NTFS та exFAT [19]

Характеристика	**NTFS v3.1**	**NTFS**	**exFAT**
Максимальний розмір тому	2^{64}-1 кластери	2^{32}-1 кластери	128 ПБ
Максимальна кількість файлів на томі	2^{32}-1	2^{32}-1	Майже не обмежено
Максимальний розмір файлу	2^{64} Байт – 1 КБ	2^{44} Байт – 64 КБ	128 ПБ
Максимальна кількість кластерів	2^{64}-1	2^{32}-1	2^{32}-1
Максимальна довжина імені файлу	До 255	До 255	До 255

NTFS включає в себе ряд розширених функцій, таких як захист файлів і каталогів, альтернативні потоки даних, розріджені файли, стиснення файлів, каталоги на основі B-дерева, шифрування.

2.2.1 Структура файлової системи

Основною особливістю структури файлової системи NTFS є наявність головної файлової таблиці (Master File Table, MFT) та її копії (MFTMirr),

спеціальної групи метафайлів (зокрема, $LogFile, $Volume та інші), а також файли і каталоги користувачів (user/file directory).

У файловій системі NTFS підтримуються 16-розрядні символи Unicode для зберігання назв файлів, каталогів і томів. Це дозволяє унікально представити кожен символ у кожній з основних мов світу, що полегшує переміщення даних з однієї країни в іншу. Кожна директорія та ім'я файлу у шляху може мати довжину до 255 символів і може містити символи Unicode, вбудовані пробіли та декілька крапок.

NTFS використовує надзвичайно простий, але ефективний підхід до організації інформації на диску. Кожен елемент на диску є файлом, а кожен файл складається з набору атрибутів. Навіть вміст файлу розглядається як атрибут. Завдяки такій простій структурі для організації та управління файловою системою достатньо декількох універсальних функцій.

На рис. 2.3 зображено структуру тома NTFS, який складається з чотирьох областей: завантажувальний сектор (Boot sector), MFT, системні файли (System files) і область файлів (File area) [4, 5].

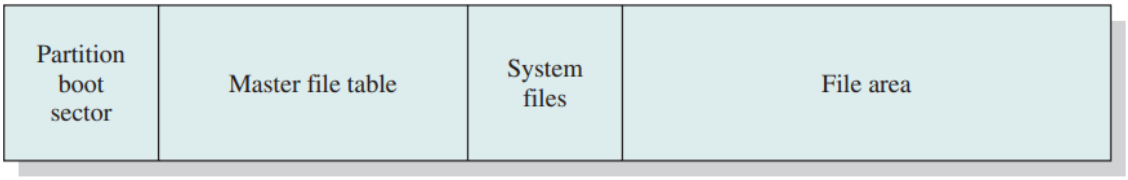


Рисунок 2.3 – Структура тому NTFS [4]

2.2.2 Системні файли NTFS

NTFS містить кілька системних файлів, усі вони приховані від перегляду на томі NTFS. Системний файл – це файл, який використовується

файловою системою для зберігання своїх метаданих та реалізації файлової системи [20].

Системні файли NTFS охоплюють:

- $MFT – це головна файлова таблиця (MFT), яка містить записи для кожного файлу у файловій системі NTFS. Це найважливіша структура для NTFS;
- $MFTMirr – дублікат перших чотирьох записів MFT. Цей файл гарантує доступ до MFT у разі відмови одного сектора;
- $Logfile містить журнал, у якому реєструються зміни метаданих. З $Logfile можна відновити інформацію щодо попередніх станів файлової системи;
- $Volume містить інформацію про том, таку як мітка, ідентифікатор та версія;
- $AttrDef містить інформацію про атрибути, що використовуються в MFT, такі як імена, ідентифікатори та розміри;
- $ - містить кореневий каталог файлової системи;
- $Bitmap – ця структура містить статус розподілу (використаний чи невикористаний) кожного кластера у файловій системі;
- $Boot містить завантажувальний сектор і завантажувальний код, який часто називають Volume Boot Record (VBR). Використовується для розміщення першого кластера $MFT;
- $BadClus містить список кластерів, які мають погані сектори;
- $Secure містить інформацію про безпеку та контроль доступу до файлів;
- $Upcase містить версію верхнього регістру всіх символів юнікоду;
- $Extend каталог, який містить розширення файлової системи, що не мають зарезервованого номера запису MFT.

Файл метаданих $Boot є єдиним файлом у файловій системі NTFS, для якого відома позиція. Цей файл завжди міститься в секторі 0 тому. Для підвищення надійності файлової системи у кінці тому зберігається резервна

копія. Файл $Boot виконує ту саму функцію, що і завантажувальний запис тому у FAT/ExFAT, тому його часто називають завантажувальним записом тому для NTFS [9, 21].

Розмір $Boot — 512 байт; він містить початковий завантажувальний код і дані про структуру тому. Важливо, що $Boot вказує на розташування $MFT, з якого можна відновити всі файли файлової системи. Також він містить інформацію про розмір кластера і запису $MFT.

Основні елементи структури $Boot:

- OEM Name – назва файлової системи. Завжди "NTFS";
- Sector Size – розмір сектора в байтах. Зазвичай 512 байт;
- Sectors/Cluster – кількість секторів у кластері. Завжди ступінь двійки;
- # Sectors – загальна кількість секторів на пристрої;
- MFT Location – логічний номер кластера для першого кластера файлу $MFT;
- MFTMirr Location – логічний номер кластера для першого кластера файлу $MFTMirr;
- MFT Record Size – розмір одного запису у $MFT у байтах. У випадку від’ємного числа x, розмір запису визначається як $2^{|x|}$ байт;
- Clusters/Index – кількість кластерів у буфері індексації;
- Serial Number – серійний номер тома [9].

2.2.3 Головна файлова таблиця MFT

Структура тому задається в головній файловій таблиці MFT. MFT організовано у вигляді таблиці з 1024-байтових рядків, які називаються записами. Структуру кожного запису зображено на рис. 2.4. Кожен запис

починається із заголовка, за яким слідує набір атрибутів. Атрибути файлів та каталогів включають в себе наступні:

- Standard Information включає таку інформацію, як позначка часу та кількість посилань;
- Attribute List використовується у випадку, якщо файлу потрібно декілька MFT-записів. Якщо присутній, вказує на інші місця, де можна знайти решту атрибутів;
- File Name – ім'я файлу для відповідного файлу/каталогу;
- Data містить дані файлу. NTFS дозволяє використовувати кілька атрибутів Data для кожного файлу;
- Index Root використовується для реалізації папок та інших індексів;
- Index Allocation -використовується для великих каталогів, дані яких не вміщуються у Index Root;
- Bitmap – структура бітової карти, яка інформує про стан розподілу кластерів.

Атрибути складаються із заголовка та компонентів (блоків) даних. Якщо дані атрибута можуть уміститися в записі MFT, вони називаються резидентними. Наприклад, така інформація, як ім’я файлу та позначка часу, завжди зберігається безпосередньо в записі MFT. Якщо ж дані надто великі, щоб уміститися в одному записі, деякі атрибути стають нерезидентними. У такому випадку їм виділяється один або кілька кластерів дискового простору в іншій частині тому, а сам атрибут містить вказівник на місце розташування цих даних. Такий покажчик складається із трьох елементів: порядкового номера кластера на томі (LCN), номера кластера всередині файлу (VCN) і довжини екстента (рис. 2.5) Якщо всі атрибути не вміщаються в одному записі MFT, NTFS створює додаткові записи та поміщає атрибут Attribute List до MFT-запису, щоб описати розташування всіх записів атрибутів [4, 9, 22].

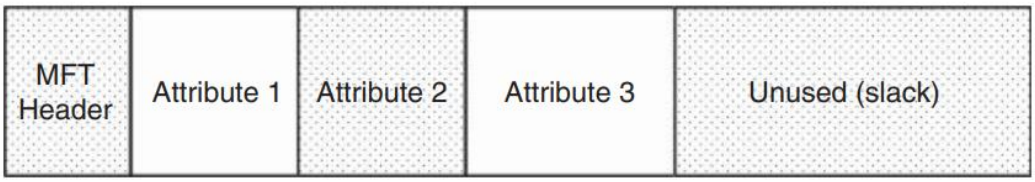


Рисунок 2.4 – Структура одного запису MFT [9]

Кожен запис MFT відповідає окремому звичайному файлу, що містить атрибути та блоки даних. Номер першого блоку MFT зберігається у завантажувальному блоці та фіксується під час інсталяції операційної системи. Другий запис у таблиці використовується для фіксації змін у файловій структурі, зокрема при зміні або видаленні каталогів.

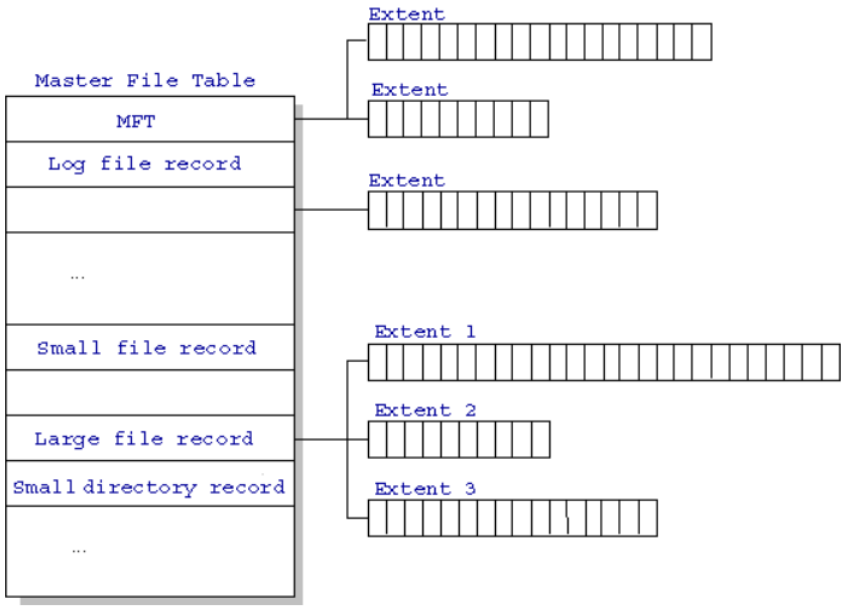


Рисунок 2.5 – Головна файлова таблиця NTFS [23]

Залежно від розміру, файли можуть зберігатися по-різному:

- малі файли розміщуються повністю в одному записі MFT і не потребують додаткових екстентів;

- файли середнього розміру використовують нерезидентні атрибути, у яких зберігаються покажчики на кожен екстент даних у межах атрибута Data головного запису;
- великі або сильно фрагментовані файли, дані яких не вміщаються в одному записі (атрибут Data не вміщує всіх покажчиків на екстенти), зберігаються у кількох записах MFT.

NTFS використовує індекси для зберігання груп атрибутів у впорядкованому порядку. Однією з найпоширеніших структур індексів у NTFS є каталог. У цьому випадку в індексі зберігається ряд атрибутів $FILENAME. NTFS використовує структуру B-дерева для зберігання індексів. B-дерево - це самобалансуюча деревоподібна структура даних, яка часто зустрічається в сучасних файлових системах. B-дерева складаються з вузлів, які пов'язані між собою ієрархічно. Кожне B-дерево має верхній рівень, головний вузол, який має два або більше дочірніх вузлів. Внутрішні вузли мають батьківський вузол і два або більше дочірніх вузлів, а листяні вузли мають батьківський вузол і нуль дочірніх вузлів. Індексні структури (тобто B-дерева) мають велику перевагу в продуктивності над лінійними структурами зберігання, оскільки вони набагато швидші в пошуку. B-дерева дозволяють здійснювати пошук, вставку та видалення за логарифмічний час.

Записи каталогів зберігаються в головній файловій таблиці, як і записи файлів. Замість даних каталоги містять індексну інформацію. Існує дві форми індексних атрибутів: $INDEX_ROOT та $INDEX_ALLOCATION. У файловій системі NTFS каталоги зберігаються двома способами. Якщо каталог містить невелику кількість файлів, уся індексна інформація зберігається у резидентному атрибуті $INDEX_ROOT як впорядкований список записів. У випадку, коли обсяг даних перевищує допустимий для резидентного зберігання, використовується нерезидентний атрибут $INDEX_ALLOCATION, що дозволяє організувати дані у вигляді B-дерева. Кожен вузол дерева містить посилання на файл, а також копії деяких атрибутів із відповідного запису MFT, зокрема ім’я файлу, дату доступу та

розмір. Це дозволяє ефективно виконувати операції, які не потребують безпосереднього доступу до вмісту файлів, наприклад, швидко перелічувати вміст каталогу [9, 23].

2.2.4 Можливості NTFS

Як вже було сказано раніше, файлова система NTFS підтримує низку функцій. Деякі з цих функцій доступні у вигляді інтерфейсів API для використання програмами, а інші є внутрішніми функціями:

Альтернативні потоки даних (Alternate data streams, ADS) - це назва для розгалужень у NTFS. Розгалуження дозволяють асоціювати з іменем файлу більше одного потоку даних. Зазвичай ADS не відображаються у провіднику Windows при перегляді вмісту каталогу. Видно лише основний потік даних. Інструменти аналізу покажуть альтернативний потік даних на додаток до основного потоку.

NTFS – це файлова система з журналюванням. Це зробило NTFS більш відмовостійкою, ніж попередні файлові системи. NTFS реалізує журналювання змін метаданих, фіксуючи їх у спеціальному системному файлі $LogFile. Кожна транзакція над структурою файлової системи виконується атомарно: або повністю, або не виконується зовсім. У журнал записують інформацію про початок і підтвердження транзакції. Якщо в журналі бракує місця, ОС Windows NT ставить транзакції в чергу, і всі нові операції введення-виведення відміняються. Коли всі поточні операції виконані, NTFS звертається до менеджера кеша, щоб той записав весь кеш на диск, після чого журнал очищають і виконують усі відкладені транзакції.

Розріджений файл – це файл, значна частина якого складається з порожніх (обнулених) кластерів. Більшість файлових систем зберігають ці обнулені кластери як частину вмісту файлу, що призводить до неефективного

використання дискового простору. Натомість у файловій системі NTFS обнулені ділянки не зберігаються на диску. Замість цього інформація про них зберігається в метаданих, а сам простір може бути використаний іншими файлами.

Стиснення: Файлова система NTFS підтримує стиснення даних на рівні файлової системи, що дозволяє зменшити обсяг фізично зайнятого дискового простору. Стиснення і розпакування здійснюються прозоро для прикладних програм, тому не потребують змін у їхньому коді. Стискання може застосовуватись як до окремих файлів, так і до каталогів. Якщо каталог позначено як стиснутий, усі файли, які будуть створені або переміщені до нього, автоматично стискаються. У процесі стиснення використовується алгоритм, заснований на методі LZ77.

Шифрування даних у NTFS реалізується через механізм EFS (Encrypting File System). EFS забезпечує прозоре шифрування та дешифрування даних на рівні файлової системи. Для шифрування використовується симетричний ключ, який застосовується до вмісту файлу. Цей ключ, у свою чергу, шифрується за допомогою відкритого ключа користувача на основі асиметричного алгоритму і зберігається у захищеному потоці файлу. Доступ до зашифрованих даних можливий лише при наявності відповідного приватного ключа, захищеного паролем користувача. У разі втрати пароля або фізичного доступу до пристрою зловмисником, дані залишаються захищеними. Шифрування недоступне для файлів у кореневому каталозі системного тому та каталозі \Windows, оскільки ці файли можуть використовуватись до активації EFS під час завантаження системи.

Списки контролю доступу (Access Control Lists): NTFS використовує дескриптори безпеки для визначення власника. Кожен дескриптор безпеки містить два списки контролю доступу (ACLs). Дискреційний список керування доступом (DACL) визначає дії (читання, запис тощо), дозволені/заборонені для кожного користувача/групи. Системний список контролю доступу (SACL) визначає дії, які слід реєструвати [5, 9].

2.3 Файлова система APFS

Apple File System (APFS) — це власна файлова система, розроблена компанією Apple Inc. у 2017 році для операційних систем macOS Sierra (10.12.4) та iOS 10.3. APFS працює на всіх платформах Apple: для iPhone, iPad, Mac, Apple TV, Apple Watch та Apple Vision Pro, також є оптимізованою для флеш-/SSD-сховищ. Вона спрямована на виправлення основних проблем HFS+ (також званої Mac OS Extended), попередника APFS, який використовувався з 1998 року [24, 25].

Основні характеристики файлової системи APFS порівняно з HFS+ представлено в таблиці 2.8.

Таблиця 2.8 – Характеристики APFS порівняно з HFS+ [26]

Характеристика	*APFS*	*HFS+*
Кількість блоків розподілу	2^{63} (9 квінтильйонів)	2^{32} (4 мільярди)
ID файлів	64-бітний	32-бітний
Максимальний розмір файлу	2^{63} байти	2^{63} байти
Точність позначення часу	1 наносекунда	1 секунда
Копіювання під час запису	Так	-
Захист від збою	Так	зафіксовано в журналі

2.3.1 Можливості APFS

APFS має ряд розширених функцій, які дозволяють їй перевершити HFS+ в плані як функціональність, так і ефективність, зокрема:

- 64-бітні іноди забезпечують масштабованість до трильйонів файлів завдяки адресації з 60-бітною глибиною (решта чотири біти використовуються для опису типу об'єкта);
- Copy-on-Write (CoW) – модифікація метаданих ніколи не відбувається безпосередньо у місці зберігання. Замість цього створюється копія, яка змінюється, після чого оновлюється покажчик;
- шифрування – підтримка повного або вибіркового шифрування файлів, зберігання ключів в об'єктах файлової системи;
- знімки (snapshots) – фіксація стану файлової системи у конкретний момент часу без дублювання даних;
- розріджені файли – забезпечує підтримку розріджених файлів, у яких послідовності нульових байтів не записуються на диск;
- спільне використання простору – усі томи в контейнері спільно використовує його простір;
- захист від збою – APFS не оновлює метадані; натомість використовує копіювання при записі (copy-on-write, CoW) для створення нового екземпляра структури метаданих, оновлює його та змінює покажчик.
- checkpoints – автоматичне збереження проміжних станів файлової системи. Це дозволяє отримати історичний огляд файлової системи;
- повна підтримка Unicode зберігає імена файлів у початковому вигляді без нормалізації;
- підтримка флеш-накопичувачів на рівні файлової системи. ApFS записує дані так, як вони будуть розташовані на флеш-накопичувачі, тоді як HFS+ завжди припускає, що вони записують на жорсткий диск [9, 27].

2.3.2 Типи об’єктів в APFS

Майже всі елементи в файловій системі APFS зберігаються як об'єкти. Кожен об'єкт ідентифікується за допомогою ідентифікатора об'єкта (Object Identifier, OID). Існує три типи OID, що відображають три різні механізми зберігання/адресації об'єктів. Це фізичні, віртуальні та ефемерні.

У випадку фізичних OID об’єкти зберігаються за відомою адресою блоку на диску, яка й виступає в ролі ідентифікатора. Такі об’єкти найпростіше знайти — достатньо звернутися до відповідного номера блоку. При внесенні змін у APFS через механізм копіювання при записі (CoW) створюється нова копія об’єкта за іншою фізичною адресою, отже, й з іншим OID. Таким чином, кожна модифікація фізичного об’єкта генерує новий OID.

Об’єкти з віртуальним OID, також зберігаються на диску, але їхні OID не відповідають фізичним адресам. Для пошуку таких об’єктів потрібно перевести віртуальний OID у фізичну адресу за допомогою карти об’єктів. Після модифікації віртуального об’єкта (знову ж CoW) його OID залишається незмінним, але оновлюється ідентифікатор транзакції, що дозволяє знаходити конкретні версії об’єкта.

Ефемерні об’єкти існують лише в оперативній пам’яті змонтованого контейнера. Після демонтажу контейнера ці об’єкти записуються в область контрольної точки. Вони змінюються без використання CoW, тобто модифікуються безпосередньо, що зумовлено їхнім зберіганням у пам’яті.

Всі об'єкти, незалежно від призначення або типу, складаються з 32-байтового заголовка, за яким слідує сам об'єкт. Заголовок складається з таких полів:

1. Checksum (8 байт) – контрольна сума, обчислена за модифікованим алгоритмом Флетчера, яка охоплює вміст блоку за винятком самих 8 байт контрольної суми.

2. OID (8 байт) – ідентифікатор об’єкта, що використовується для його адресації.
3. XID (8 байт) – ідентифікатор транзакції, що збільшується кожного разу при оновленні об’єкта.
4. Type (2 байти) – код, що визначає тип об’єкта. Існує 33 типи, серед яких найбільш важливі: Container superblock, B-Tree, B-Tree node, Object map, Checkpoint map, Volume superblock, File system B-Tree.
5. Flags (2 байти) – Прапори використовуються у заголовку об'єкта для надання додаткової інформація про поточний об'єкт. Значення для прапорців такі: віртуальний об’єкт, фізичний об’єкт, ефемерний об’єкт, відсутній заголовок, зашифрований об’єкт, непостійний об’єкт.
6. Subtype (2 байти) – підтип об’єкта, значення якого повторює код типу.
7. Padding (2 байти) – заповнювальні байти, які використовуються для вирівнювання структури [9, 28].

2.3.3 Структура файлової системи

Особливістю APFS є спосіб структурування томів. Для поділу сховища на розділи з окремими томами не використовується типова таблиця розділів. Натомість, APFS вводить у файлові системи нове поняття - контейнер. Цей контейнер поділений на декілька частин (рис. 2.6) і може містити декілька томів, які можна використовувати для зберігання файлів. Томи в одному контейнері спільно використовують простір цього контейнера. У контейнері зберігається вся інформація про розташування томів, а також інформація про всі структури, які є спільними для всіх томів.

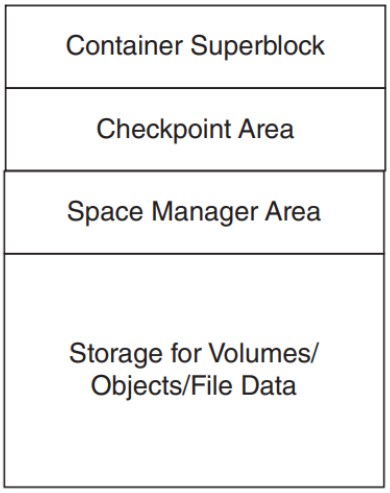


Рисунок 2.6 – Структура контейнера APFS [9]

Контейнер починається з Container Superblock (Суперблок контейнера, CSB), який знаходиться в блоці 0 контейнера та зберігає ключову інформацію про структуру: розміри блоків, кількість блоків, ідентифікатори об’єктів, розташування контрольних точок та інші параметри.

CSB містить багато інформації, необхідної для аналізу файлової системи, основною є:

- Object Header підтверджує, що поточна структура є суперблоком контейнера;
- Magic Value використовується разом із типом об’єкта для підтвердження CSB, має значення NXSB;
- Block Size – розмір блоку в байтах;
- UUID можна використовувати, щоб визначити, чи був пристрій підключений до певного комп'ютера;
- Checkpoint Descriptor and Data Area locations – поля, що вказують розташування областей контрольних точок та пов’язаних з ними даних, включаючи дескрипторні блоки та базу даних;

- OMAP OID – фізичний ідентифікатор об’єкта, який містить карту об’єктів, що використовується для трансляції віртуальних адрес у фізичні;
- Max File Systems – максимальна кількість файлових систем у контейнері, визначає розмір масиву FS_OID;
- FS_OID[] – масив віртуальних ідентифікаторів коренів файлових систем, кожен з яких відповідає окремій томовій структурі.

Контрольна точка (checkpoint area) - це історичний стан контейнера. Кожна контрольна точка ініціалізується за допомогою CSB, а поточний стан зазвичай є останнім CSB у колекції. CSB у поточному стані є тим, з якого походить головний суперблок. Контрольна точка включає метадані контейнера та тому. Точки відновлення та знімки схожі між собою. Основна відмінність між контрольною точкою та знімком полягає в можливості користувача відновити файлову систему із збережених знімків за допомогою API файлової системи.

Після області контрольної точки контейнер містить диспетчер простору (space manager). Диспетчер простору призначений для управління розподілом блоків у всьому контейнері. У контейнері є одна область диспетчера простору, яка відповідає за весь простір. Через можливість наявності декількох томів в одному контейнері APFS, менеджер простору може відповідати за розподіл блоків у більш ніж одному томі. Вся інша інформація в APFS (томи, вміст файлів, метадані тощо) може з'являтися в будь-якому місці після області менеджера простору [9, 28].

Починаючи з macOS 10.15, завантажувальний APFS-контейнер містить щонайменше п’ять томів, зокрема приховані Preboot, VM і Recovery, які спільні для всіх системних томів. Системний том доступний лише для читання й містить компоненти macOS, тоді як користувацькі дані зберігаються окремо на Data-томі. На пристроях з iOS, iPadOS та visionOS використовуються два основні томи: System і Data [25].

Кожен том APFS окремо містить структуру суперблока тому (volume superblock, VSB), яка містить інформацію, що стосується відповідного тому. До основних полів цієї структури належать:

- заголовок об’єкта (Object Header) містить службову інформацію, що дозволяє ідентифікувати структуру як об’єкт типу VSB і забезпечує перевірку її цілісності;
- Magic використовується разом з типом заголовка об'єкта для підтвердження, що це суперблок тому. Це значення повинно бути APSB;
- OMap OID – фізичний ідентифікатор об’єктної мапи тому. Цей елемент необхідний для зіставлення віртуальних ідентифікаторів об’єктів (OID) у фізичні адреси блоків за допомогою дерева мапи обєктів;
- Root Tree OID – віртуальний ідентифікатор кореневого вузла B-дерева файлової системи, з якого починається відновлення файлів.

Окрім наведених полів, суперблок тому містить додаткову інформацію, зокрема: час останнього від’єднання пристрою, статистику файлової системи (кількість файлів і каталогів), UUID тому (подібно до ідентифікатора контейнера), час останнього редагування, дані про програмне забезпечення, яким том було створено або змінено, включно з номерами версій. Також зберігається назва тому та його роль, що може свідчити про системне призначення пристрою.

Як і багато інших сучасних файлових систем, APFS використовує структури B-дерева (B-Tree) для зберігання більшої частини інформації.

У B-деревах APFS існує три типи вузлів.

1. Кореневі вузли (Root Node). У кожному B-дереві завжди існує один кореневий вузол. Цей вузол знаходиться на найвищому рівні дерева. Залежно від розміру B-дерева, кореневий вузол також може бути індексним або листовим вузлом.

2. Індексні вузли (Index Node) містять вказівники на інші вузли дерева. Як і у всіх B-деревах, ключі в індексних вузлах APFS (як і у всіх вузлах) відсортовані, що дозволяє швидко знайти потрібний дочірній вузол.
3. Листові вузли (Leaf Node) містять фактичні дані в дереві. Тип даних залежить від B-дерева. Наприклад, дерево файлової системи міститиме структури inode та extent, тоді як дерево мапи об'єктів міститиме відображення з віртуальних OID на фізичні місця на диску.

Майже всі вузли, незалежно від типу, мають спільну структуру, зображену на рис. 2.7. Тут показано кореневий вузол B-дерева. Некореневі вузли ідентичні цій структурі, за винятком того, що вони не включають інформаційну структуру B-Tree у хвості вузла.

Рисунок 2.7 – Структура кореневого вузла B-дерева APFS [9]

Основні компоненти вузла:

- заголовок вузла (Node Header) містить дані про структуру вузла;
- зміст (Table of Contents, ToC) визначає розташування ключів і значень у вузлі. Дозволяє пов'язувати ключі і значення, а також отримувати доступ до фактичних даних;
- ключі (Keys) використовуються для впорядкування і пошуку в дереві;
- значення (Values) зберігаються наприкінці вузла; у корені - перед інформацією про дерево;

- вільний простір (Free Space) розташовується між ключами (йдуть від початку) і значеннями (йдуть від кінця);
- інформація про B-дерево (B-Tree Info) містить дані про все дерево.

Карта об’єктів (object map) — це структура, яка використовується для перетворення віртуального ідентифікатора об’єкта (OID) у фізичну адресу блоку, де зберігається відповідний вміст. Вона існує як на рівні контейнера (для пошуку кореневих структур томів), так і на рівні окремих томів (для обробки віртуальних OID).

Карта об'єктів реалізована у вигляді B-дерева. Перший блок цієї структури містить службову інформацію, включаючи фізичний OID кореневого вузла дерева. Ключі у B-дереві складаються з пари значень: 8-байтового OID та 8-байтового XID (ідентифікатора транзакції), а відповідні значення - це 16-байтові записи, що містять фізичну адресу блока разом з додатковими параметрами.

Кожен том APFS має дерево файлової системи. Віртуальний OID цього дерева знаходиться у суперблоці тому. Потім його можна зіставити з фізичною адресою блоку за допомогою карти об'єктів тома. APFS використовує структуру B-дерева для зберігання інформації, пов'язаної з файлами. До неї входять іноди (містять метадані файлів), записи каталогів (показують файли, наявні у каталозі) і розширення (дозволяють визначити місцезнаходження вмісту даних), а також ряд інших записів. Розглянемо ці структури.

Структура інодів містить всі метадані про файл. Сюди входить інформація про час, розмір файлу, власників тощо. Також можуть містити розширені поля.

Розширені поля (Extended fields) використовуються для збереження додаткової інформації, найчастіше, імені файлу або потоків даних. Розширені поля можуть містити як іноди, так і записи каталогів. Щоб визначити, чи містить запис розширене поле, звертаються до змісту вузла дерева файлової системи. Якщо довжина значення для іноду більша за 92 байти, запис містить

розширені поля. Аналогічно, якщо довжина запису каталогу у змісті більша за 18 байт, запис містить розширені поля. Ці поля мають свою структуру, яка складається з трьох частин. Перша з них - це інформаційна структура розширеного поля. Вона містить інформацію про всі розширені поля, присутні в цій області. Далі йде масив елементів, що містять інформацію про кожне окреме розширене поле. Кількість елементів у цьому масиві дорівнює загальній кількості розширених полів, присутніх у записі. Далі область розширених полів містить масив даних. Ця область також містить один запис на кожне розширене поле в записі.

Структура імені файлу тривіальна, це просто завершений нулем (0x00) рядок у кодуванні UTF-8. Потік даних дозволяє отримати фактичний розмір файлу разом з іншою інформацією.

Записи каталогів зберігають інформацію про файли, присутні у каталозі (а також час, коли їх було додано до каталогу). Ключ запису каталогу формується з OID, за яким слідує чотирибайтова довжина імені та хеш-значення. Ця структура використовує найменш значущі 10 бітів як довжину, а решту 22 бітів як хеш-значення. Далі йде ім'я файлу, що закінчується нульовим символом. Структура значення запису каталогу включає: ідентифікатор файлу, дату додавання, прапори (невідомий, каталог або звичайний файл) та розширені поля [9].

У файловій системі APFS для обліку використаних та вільних блоків використовується бітова карта (bitmap), яка охоплює весь контейнер і є спільною для всіх томів, що в ньому розміщені. Структура бітової карти має ієрархічний характер. На верхньому рівні розміщується елемент, який визначає загальні межі обліку блоків. Нижчі рівні містять блоки, які безпосередньо фіксують стан кожного блоку пам’яті в контейнері. У таких блоках один байт відповідає за вісім блоків файлової системи, де кожен окремий біт вказує на стан конкретного блоку — чи він вільний, чи вже використовується [28].

2.4 Файлова система Ext

Файлові системи сімейства ext є найпоширенішими файловими системами у системах Linux. У більшості сучасних дистрибутивів за замовчуванням використовується ext4, тоді як ext2 і ext3 зустрічаються рідше, переважно на знімних носіях або у системах із застарілою архітектурою. Файлова система ext також присутня на багатьох телефонах Android.

У цьому підрозділі спочатку буде наведено деякі відомості про сімейство файлових систем ext, і детально описано їхню структуру. Це дозволить зрозуміти, як файлова система зберігає інформацію. Дальше розглядатимуться розширені можливості, представлені в ext3 і ext4 [9].

Основні характеристики файлових систем ext представлено в таблиці 2.9.

Таблиця 2.9 – Характеристики файлових систем ext [9]

Файлова система	*Максимальний розмір файлу*	*Максимальний розмір тому*
Ext2	2 ТБ	32 ТБ
Ext3	2 ТБ	32 ТБ
Ext4	16 ТБ	1 ЕБ

2.4.1 Структура даних ext

Файлові системи ext організовані у вигляді набору блокових груп. Кожна блокова група здатна керувати власним простором для зберігання та метаданими файлів. Також містить інформацію про всі інші блокові групи у

файловій системі. На рис. 2.8 зображено загальну структуру файлової системи.

Увесь том складається з груп блоків однакового розміру, за винятком початку тому. Усі томи ext починаються з 1024 байтів нерозподіленого (Unallocated) простору. Блок-група 0 (Block Group 0, BG0) починається відразу після цієї області. Блок-групи, як міні-файлові системи, мають власну внутрішню структуру. На рисунку 2.19 зображено структуру блок-групи 0 (BG0). Не всі блокові групи містять структуру суперблоку (хоча ця структура дублюється в декількох місцях файлової системи). Однак BG0 завжди містить структуру суперблоку.

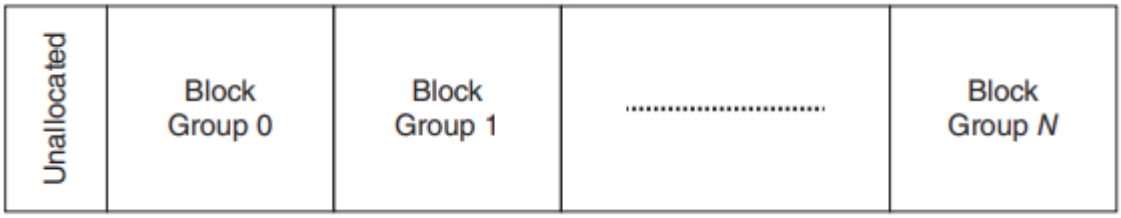


Рисунок 2.8 – Структура ext [9]

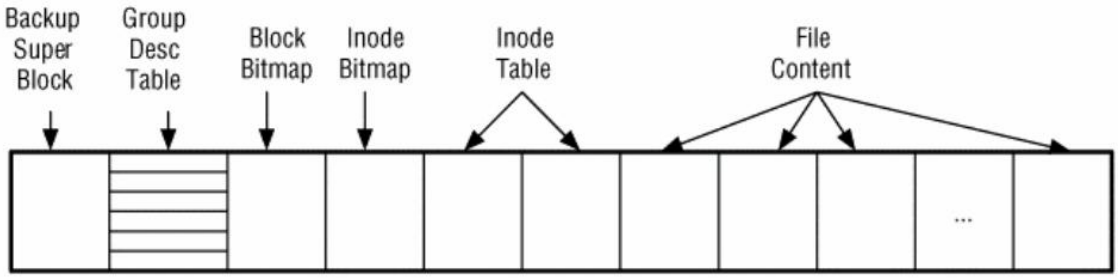


Рисунок 2.9 – Внутрішня структура блок-групи [8]

Суперблок є однією з найважливіших структур в ext і початком будь-якого аналізу файлової системи. Він розташований на відстані 1,024 байт від початку файлової системи і має розмір 1,024 байт, хоча більша частина цього обсягу не використовується. Суперблок схожий на завантажувальний запис

тома FAT і Boot NTFS. Містить основну інформацію, таку як розмір блоку, загальна кількість блоків, кількість блоків на групу блоків і кількість зарезервованих блоків перед першою групою блоків. Суперблок також містить загальну кількість інодів і кількість інодів на групу блоків. До необов'язкових даних суперблоку належать ім'я тому, час останнього запису, час останнього монтування та шлях, за яким файлова система була монтована востаннє.

У ранніх реалізаціях ext копії суперблоку розміщувались у кожній блоковій групі. Надалі була впроваджена опція sparse_superblock, яка дублює суперблок лише у вибраних групах (0, 1, та кратних степеням 3, 5 або 7).

У блоці, що слідує за суперблоком, знаходиться таблиця дескрипторів груп (Group Descriptor Table), яка містить структуру даних дескриптора групи для кожної групи блоків у файловій системі. Резервні копії таблиці існують у кожній групі блоків, якщо не ввімкнено функцію sparse_superblock. Окрім вмісту файлів, групи блоків містять адміністративні дані, такі як суперблоки, таблиці дескрипторів груп, таблиці inode, бітові мапи inode та бітові мапи блоків. Дескриптор групи описує, де можна знайти ці дані.

Кожен дескриптор групи блоків має розмір 32 байти. Таблиця дескрипторів груп блоків містить по одному дескриптору для кожної з груп блоків у файловій системі.

У файлових системах ext для керування станом розподілу блоків і інодів у межах груп блоків використовуються бітові мапи. Зокрема, бітова мапа блоків (Block Bitmap) відображає, які блоки зайняті, а які - вільні. Її початкова адреса зберігається в дескрипторі групи. Під час створення файлової системи Linux визначає кількість блоків у групі таким чином, щоб вона дорівнювала кількості бітів в одному блоці, що дозволяє розмістити бітову мапу в одному блоці. Аналогічно функціонує бітова мапа інодів (Inode Bitmap), яка зберігає інформацію про стан розподілу інодів у групі. Її початкова адреса також фіксується в дескрипторі групи, а розмір у байтах

дорівнює кількості інодів у групі, поділеній на вісім. Зазвичай кількість інодів менша за кількість блоків [8, 9].

Таблиця інодів (Inode Table) розміщується у межах кожної групи блоків, а її початкова адреса зазначається в дескрипторі групи. Розмір одного інода, за замовчуванням становить 128 байтів. Кожен інод містить метадані файлу або каталогу: розмір, права доступу, власника, часові мітки та покажчики на блоки з даними. Структура інодів містить 4 способи вказівки на дані файлу:

- перші 12 вказівників вказують на блоки, які містять дані файлу;
- далі вказівник вказує на непрямий блок, який сам по собі містить вказівники на дані файлу;
- передостанній вказівник вказує на подвійно непрямий блок. Цей блок містить вказівники на інші непрямі блоки;
- потрійний непрямий блок містить вказівники на подвійно непрямі блоки [11].

Дескриптор групи містить також інформацію про кількість вільних блоків і інодів у межах конкретної групи, тоді як суперблок відображає загальну кількість вільних блоків та інодів у всій файловій системі.

У бітових мапах для представлення стану елементів (блоків або інодів) використовується один біт на кожен елемент. Значення 1 означає, що елемент використовується, а значення 0 означає, що елемент не використовується [8, 9].

2.4.2 Ext3

Файлова система Ext3 була запропонована у 1998 році та повністю інтегрована в ядро Linux у листопаді 2001 року. Вона є прямим розширенням Ext2 і зберігає з нею повну зворотну сумісність, що дозволяє оновлення

існуючих томів Ext2 до Ext3 без необхідності форматування або відновлення з резервної копії. Ext3 додала до ext2 низку нових можливостей:

- підтримка журналювання;
- можливість динамічного (онлайн) розширення файлової системи;
- впровадження індексованої структури каталогів HTree.

Журнал файлової системи записує зміни, які були зроблені у файловій системі до того як вони будуть записані на диск. Таким чином, журнал слугує захистом від збоїв, які можуть зруйнувати цілісність файлової системи.

Ext3 підтримує три режими журналювання:

- повне журналювання (journal): як метадані, так і вміст файлів записуються до журналу перед тим, як потрапити в основну файлову систему;
- упорядкований режим (ordered): використовується за замовчуванням. У журналі фіксуються лише метадані, однак гарантується, що вміст файлів буде записаний на диск до оновлення відповідних метаданих;
- запис після зміни (writeback): журнал ведеться лише для метаданих, а вміст файлів може бути записаний до або після оновлення журналу.

Для ефективної роботи з великими каталогами у Ext3 впроваджена індексована структура каталогів HTree, яка є подібною до B-дерева, що використовуються у файлових системах NTFS та APFS. Глибина HTree обмежена двома рівнями, що дозволяє мати широку розгалуженість з великою кількістю дочірніх вузлів на кожному рівні. HTrees індексуються на основі хешу імені файлу, а не самого імені файлу.

Ext HTree містить три типи вузлів. Один кореневий вузол знаходиться на початку файлу кореневого каталогу. Він потім пов'язується з внутрішніми або листовими вузлами. Листові вузли в структурі HTree є просто лінійним масивом записів каталогу, які можуть оброблятися за допомогою традиційних алгоритмів обробки ext2. Листові вузли іноді називають

блоками записів каталогу, оскільки вони містять лише традиційні записи каталогу. Кореневий вузол містить набір хеш-значень і номер блоку, в якому можна знайти файли з хешем, що відповідає цьому значенню. У разі виникнення колізій, що вимагають більше одного блоку, головне хеш-значення вказує на проміжний вузол (також званий блоком індексу каталогу), який використовує другорядні хеші для відображення на наступні листові вузли.

Завдяки використанню HTree, практична межа кількості файлів у каталозі в Linux-системах зросла з кількох тисяч (у випадку лінійної структури) до десятків мільйонів записів [9, 11].

2.4.3 Ext4

Починаючи з 2008 року, четверта розширена файлова система (Ext4) стала типовим вибором для більшості дистрибутивів Linux. Вона є наступницею Ext3 і усуває ряд її обмежень, водночас пропонуючи ширший набір функцій. Ext4 має низку нових можливостей:

- збільшений розмір інодів (до 256 байт);
- покращена точність міток часу;
- зняття обмеження на кількість підкаталогів;
- використання екстентів як основної структури зберігання даних;
- підтримка вбудованого зберігання невеликих файлів.

Ext4 за замовчуванням використовує розмір іноду 256 байт, що вдвічі більше, ніж у попередніх версіях файлової системи ext. Перші 128 байт іноду ext4 ідентичні тим, що були в попередніх версіях ext. Решта байтів inode можуть бути використані для розширених атрибутів.

Ще одним удосконаленням є збільшена точність міток часу. Якщо у Ext3 час запису, зміни або доступу зберігається з точністю до секунд, то в

Ext4 - з точністю до наносекунд, що дозволяє точніше фіксувати зміни у системі.

Ext3 має обмеження на кількість підкаталогів у межах одного каталогу — 31998. Ext4 збільшує обмеження до 64000, і, встановивши конфігураційну змінну, можна перевищити це обмеження.

Основна зміна це перехід від блочних покажчиків до екстентів. У попередніх версіях для кожного блоку даних використовувався окремий покажчик. У Ext4 замість цього застосовуються екстенти - структури, що описують послідовності суміжних блоків. Це дозволяє значно зменшити обсяг метаданих при роботі з великими нефрагментованими файлами. Екстенти організовані у деревоподібну структуру: кореневий вузол містить заголовок і, залежно від глибини дерева, або безпосередні посилання на дані (листові вузли), або індексні вузли, що ведуть до нижчих рівнів дерева. Розмір кожного заголовка або екстенту становить 12 байт. Для невеликих або нефрагментованих файлів уся структура екстентів може зберігатися безпосередньо у полі покажчиків інода.

Функція inline data дозволяє зберігати малі файли безпосередньо у самому іноді. Цей метод зберігання схожий на резидентні атрибути DATA в NTFS [9, 11].

2.5 Файлова система Btrfs

Btrfs (B-tree File System) - це сучасна файлова система, яка використовує B-дерева як основний механізм зберігання даних, спрямована на реалізацію розширених функцій, а також на стійкість до відмов, відновлення та легке адміністрування. Створена Крісом Мейсоном у 2007 році для використання в Linux, а з листопада 2013 року формат файлової системи на диску оголошено стабільним у ядрі Linux [29, 30]. У 2020 році

Btrfs обрано файловою системою за замовчуванням замість ext4 для Fedora 33 для варіантів робочого столу [31].

Основні характеристики файлової системи Btrfs представлені в таблиці 2.10.

Таблиця 2.10 – Основні характеристики файлової системи Btrfs [9]

Характеристика	**Значення**
Розмір тома	2^{64} Байт (16 ЕБ)
Розмір файлу	2^{64} Байт (16 ЕБ)
Кількість файлів	2^{64}
Довжина імені файлу	255 Байт = 255 символів ASCII

Btrfs забезпечує підтримку багатьох принципів сучасних файлових систем, таких як:

- B-Tree-Based File System: усі метадані (за винятком суперблока) зберігаються за допомогою B-дерев;
- Copy-on-Write (CoW): зміни до даних здійснюються шляхом створення нових копій замість перезапису існуючих блоків, що дозволяє зберігати попередні версії даних і метаданих;
- Extent-Based File Storage: дані зберігаються у вигляді екстентів;
- integrated Checksums: кожен вузол має власну контрольну суму, що дозволяє перевіряти цілісність метаданих;
- subvolumes: підрозділи всередині файлової системи, де кожен пристрій Btrfs монтується як підтом; за замовчуванням монтується верхньорівневий підтом, але можливе призначення іншого;
- snapshots: механізм створення знімків стану файлової системи без зміни вихідних даних, реалізується через CoW на рівні підтомів;
- стиснення: під час створення файлової системи можна активувати стиснення даних [9].

2.5.1 Структура Btrfs

Як і багато інших файлових систем, Btrfs містить суперблок, який надає інформацію про файлову систему в цілому. У Btrfs суперблок виконує ту саму функцію, що і в ext, а також функцію завантажувального запису тому у FAT і NTFS. З інформації суперблоку, що стосується одиниць розміщення даних на пристроях (секторів, вузлів і листків у випадку Btrfs), визначаються логічні адреси певних дерев, які необхідні для відновлення всіх даних і метаданих у файловій системі тощо [9].

У файловій системі існує декілька копій структури суперблоків, які зберігаються у фіксованих місцях: 64 КБ на кожен блоковий пристрій, з резервними копіями (для надійності) на 64 МБ, 256 ГБ і 1 ПБ.

Суперблок містить багато полів. Серед цих полів деякі вказують на тип, версію, розмір та інші характеристики файлової системи:

- sb_MagicNum є ідентифікатором файлової системи і необхідне для підтвердження того, чи є файлова система Btrfs чи ні;
- sb_Generation містить номер генерації суперблоку, який є монотонно зростаючим числом, записаним під час контрольної точки. Ці контрольні точки за замовчуванням відбуваються кожні 30 секунд;
- sb_Generation, оскільки суперблок завжди редагується на місці, це поле можна використовувати для передбачення версії суперблоку або того, скільки разів суперблок було оновлено;
- sb_TotalBytes вказує на загальну кількість байт, яку містить файлова система;
- sb_BytesUsed вказує на використані байти;
- sb_SectorSize визначає розмір сектора;
- sb_NodeSize та sb_LeafSize – визначають розмір вузлів дерева в яких зберігаються дані та метадані;

- sb_NodeSize містить розмір блоку дерева, який за замовчуванням дорівнює 4 КБ і може підтримувати максимальне значення 64 КБ;
- sb_LeafSize містить розмір листових вузлів і завжди дорівнює розміру вузла;
- sb_RootTreeAddr містить початкову логічну адресу кореня кореневого дерева. Це поле у поєднанні з полем sb_NodeSize використовується для пошуку кореневого вузла кореневого дерева;
- sb_RootDirObjId містить ідентифікатор об'єкта кореневого каталогу.

Всі ці поля є важливими, оскільки вони не лише ідентифікують тип і версію файлової системи, але й надають всі необхідні дані (такі як адреса і розмір) для знаходження всіх інших структур даних файлової системи [29].

2.5.2 Дерева в Btrfs

B-дерево у Btrfs, як і в багатьох сучасних файлових системах, є базовою структурою для зберігання метаданих. У файлових системах Btrfs існує декілька системних B-дерев:

- ROOT_TREE (іноді його називають деревом дерев) є первинною структурою, необхідною для перебудови файлової системи та всіх інших структур метаданих. ROOT_TREE містить інформацію про розташування всіх інших дерев у системі;
- EXTENT_TREE містить інформацію про розміщення даних та метаданих у файловій системі. Він надає інформацію про місця, в яких різні типи даних можуть зберігатися і зберігаються у файловій системі;
- CHUNK_TREE містить інформацію про всі пристрої, які присутні у файловій системі. CHUNK_TREE - це структура, яка використовується для відображення логічних адрес у фізичні.

Логічна файлова система розділена на декілька частин, записи для яких з'являються у CHUNK_TREE. Ці записи надають фізичні смуги, пов'язані з логічною адресою, таким чином дозволяючи перетворити логічні адреси у фізичні;

- DEV_TREE використовується в ситуаціях, коли необхідно зіставити логічну адресу з фізичною. Ця структура зазвичай використовується при зміні конфігурації пристрою у файловій системі;
- FS_TREE (або дерево файлової системи) дозволяє відновити вміст усієї файлової системи. Ця структура містить інформацію про кожен файл і каталог. Обробка цієї структури дозволяє перерахувати і відновити всі файли у файловій системі а також відновити пов'язані з ними метадані;
- Дерево кореневих каталогів забезпечує представлення кореневого каталогу (фактично вказує на FS_TREE);
- CSUM_TREE використовується для перевірки даних. Він містить контрольні суми для кожного розділу даних у файловій системі [9].

Кожне B-дерево складається з одного або декількох вузлів. З суперблоку можна дізнатися розмір вузлів у конкретній файловій системі. Btrfs надає два різних типи: внутрішні та листові.

Внутрішні вузли дерев Btrfs містять списки пар [ключ, вказівник] і вказують на інші вузли того ж дерева (рис. 2.10). Ці вузли не містять жодної інформації про файл або каталог. Замість цього вони містять вказівники блоків на інші внутрішні або листові вузли разом з відповідними найменшими значеннями ключів. Вказівник містить ключ елемента, адрес наступного вузла та покоління.

Листові вузли використовують для зберігання фактичної інформації про файли або каталоги, наприклад, імен файлів, даних про вкладені файли, даних про розмір. На початку листового вузла зберігається масив елементів, які містять ключі до місцезнаходження даних, а в кінці - відсортований у зворотному порядку масив даних елементів таким чином, що ці два масиви

зростають один до одного (рис. 2.11). Елемент масиву має фіксований розмір, тоді як елемент даних може бути змінного розміру [29].

Рисунок 2.10 – Структура внутрішнього вузла [9]

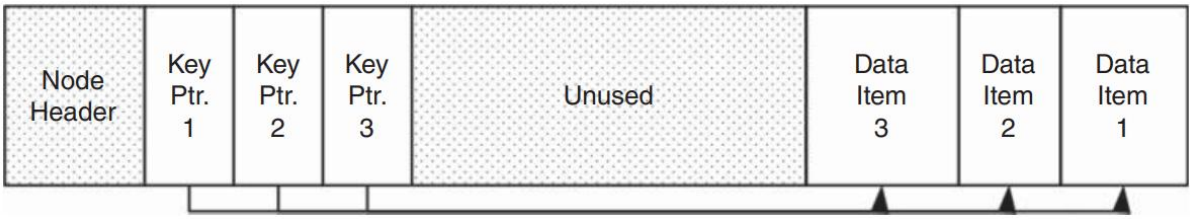


Рисунок 2.11 – Структура листового вузла [9]

Як можна побачити з рис. 2.10 та рис. 2.11 кожен вузол, незалежно від того, чи є він внутрішнім або листовим, починається із заголовка (Node Header). Цей заголовок містить інформацію про вузол та дерево, в якому знаходиться. У структурі заголовка зберігаються такі дані: контрольна сума, яка охоплює всі дані блоку, UUID файлової системи, логічна адреса вузла, прапори файлової системи і ревізія зворотного посилання, яка дорівнює 1 у нових файлових системах. Далі слідує значення покоління, знайдене в суперблоці, ідентифікатор дерева, до якого належить вузол, кількість елементів у вузлі, а також рівень вузла. Рівень 0 відповідає листовим вузлам, тоді як більші значення вказують на внутрішні вузли, де число визначає кількість рівнів, які потрібно пройти до листа.

При досліджені структури вказівників у внутрішньому, так і у листовому вузлах, є таке поняття як ключ елемента. Ці ключі використовуються для ідентифікації типів елементів. Кожен ключ має

фіксований розмір 17 байт і виконує функцію сортування елементів у вузлі дерева. Структура ключа включає три поля: ідентифікатор об’єкта (OID), який визначає об’єкт, до якого відноситься ключ; тип елемента (Item Type); та зміщення (Offset), значення якого залежить від конкретного типу елемента [9].

Поле Item Type в ключі описує тип наявного елемента. У Btrfs є багато типів об'єктів, найбільш важливими є:

- INODE_ITEM містить усі метадані щодо файлу або каталогу;
- INODE_REF зберігає інформацію про назву файлу. Зокрема, він зіставляє ідентифікатор об'єкта з іменем у каталозі, подібно до запису каталогу в ext. Зустрічається у файлових деревах, але також може бути знайдений у ROOT_TREE;
- DIR_ITEM знаходиться у каталогах і містить записи каталогів. Ключове зміщення для DIR_ITEM містить хешоване значення імені файлу в DIR_ITEM;
- EXTENT_DATA надає інформацію про те, де зберігається вміст файлу. У Btrfs вміст файлу може зберігатися в рядку або з використанням розширень (екстентів);
- ROOT_ITEM розміщуються лише у кореневому дереві ROOT_TREE, дозволяє знайти корінь B-дерева;
- ROOT_REF містить інформацію про підтоми, таку як назва тому. Цей елемент і елемент ROOT_BACKREF містяться лише у ROOT_TREE;
- DEV_ITEM містить інформацію про пристрій. DEV_ITEM знаходяться у CHUNK_TREE, яке містить DEV_ITEM для кожного окремого пристрою у файловій системі [9, 30].

2.6 Файлова система F2FS

F2FS (Flash-Friendly File System) - це файлова система флеш-пам'яті, спочатку розроблена компанією Samsung Electronics для ядра Linux наприкінці 2012 року. Мотивом для F2FS було створення файлової системи, яка з самого початку враховує характеристики пристроїв зберігання даних на основі флеш-пам'яті NAND (SSD, eMMC та SD-карти), яка базується на логарифмічно структурованій файловій системі (Log-structured File System, LFS).

Крім цього, F2FS підтримується операційною системою Android, і використовується деякими виробниками мобільних пристроїв, як основна файлова система.

Основні характеристики F2FS:

- максимальний розмір файлу: 16 ТБ;
- максимальний розмір тому: 16 ТБ;
- максимальна довжина імені файлу: 255 байт [32-33].

2.6.1 Структура F2FS

F2FS розділяє весь том на сегменти, кожен з яких має фіксований розмір 2 МБ (рис. 2.12). Сегмент (segment) є основною одиницею управління в F2FS і використовується для визначення початкового розміщення метаданих файлової системи. Послідовність сегментів утворює секції (section), а кілька секцій формують зони (zone).

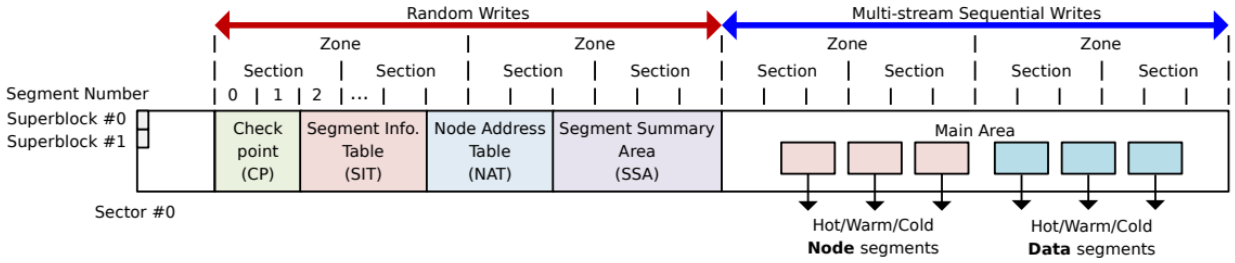


Рисунок 2.12 – Розташування F2FS на диску [34]

F2FS розділяє весь том на шість областей:

1. Суперблок (Superblock, SB) містить основну інформацію про розділи та параметри за замовчуванням F2FS, які задаються під час форматування і не можуть бути змінені.
2. Контрольна точка (Checkpoint, CP) зберігає стан файлової системи, бітові мапи для дійсних наборів NAT/SIT, списки покинутих інодів та дані про активні сегменти. Область CP зберігає два пакети контрольних точок у двох сегментах (0 і 1): один для останньої стабільної версії, а інший для проміжної (застарілої) версії.
3. Таблиця інформації про сегменти (Segment Information Table, SIT) містить інформацію про сегменти, таку як кількість дійсних блоків і бітова мапа дійсності всіх блоків в області «Main».
4. Таблиця адрес вузлів (Node Address Table, NAT) - використовується для відображення node ID у фізичні адреси блоків вузлів.
5. Область підсумків сегментів (Segment Summary Area, SSA) зберігає підсумкові записи, що містять інформацію про власника всіх блоків в основній області, таку як номер батьківського іноду та його зміщення вузла/даних.
6. Основна область (Main Area) складається з 4 КБ (або 16 КБ [32]) блоків, що зберігають дані або вузли. Вузли містять inode або індекси до блоків з даними; блоки з даними містять вміст файлів або каталогів. Секція не може одночасно містити як вузли, так і дані.

З огляду на наведені вище структури даних на диску, проілюструємо, як виконується операція пошуку файлу. Припустимо, що є файл «/dir/file». F2FS виконує такі кроки:

1. Отримує кореневий inode, читаючи блок, розташування якого отримано з NAT.
2. У його блоках даних знаходиться запис каталогу з ім’ям dir, з якого витягується відповідний inode.
3. Перетворює отриманий номер inode у фізичне розташування за допомогою NAT.
4. Отримує inode з іменем dir, прочитавши відповідний блок.
5. У inode dir ідентифікує запис каталогу з іменем file і, нарешті, отримує inode файлу, повторивши кроки (3) і (4) для file.

Фактичні дані можна отримати з основної області за допомогою індексів, отриманих через відповідну структуру файлу.

2.6.2 Структура файлів та каталогів

У файловій системі F2FS структура вузлів (node) використовується для зберігання та індексації даних. Кожен блок вузла має унікальний node ID, за яким таблиця NAT визначає фізичне розташування відповідного блоку на диску. Вузли бувають трьох типів: inode, direct (прямі) та indirect (непрямі). Inode-блок містить метадані файлу, зокрема ім’я, розмір, номери inode, часи доступу та видалення. Прямі вузли зберігають адреси блоків даних, тоді як непрямі містять node ID інших вузлів, що забезпечує багаторівневу індексацію. Прямі вузли містять посилання на блоки даних, а непрямі — на інші вузли, забезпечуючи ієрархічну структуру доступу. Завдяки цьому, при зміні файлу оновлюється лише відповідний прямий вузол і запис у NAT, що знижує навантаження на файлову систему та покращує продуктивність.

Блок inode (рис. 2.13) містить прямі покажчики на блоки даних файлу, два одинарні непрямі (single-indirect), два подвійні непрямі (double indirect) та один потрійний непрямий (triple-indirect) покажчик. F2FS підтримує вбудовані дані (inline data) та розширені атрибути (inline extended attributes), які вбудовують невеликі дані або розширені атрибути в сам блок inode.

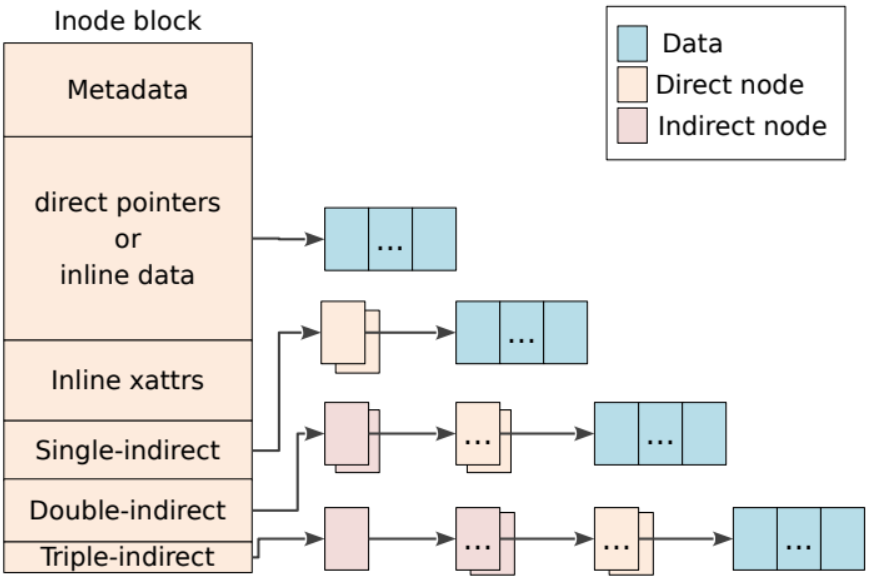


Рисунок 2.13 – Структура inode-блоку [34]

Вбудовування зменшує вимоги до простору та покращує продуктивність вводу-виводу. Багато систем мають невеликі файли та невелику кількість розширених атрибутів. За замовчуванням F2FS активує вбудовування даних, якщо розмір файлу менше 3692 байтів. Для розширених атрибутів зарезервовано 200 байтів у блоці inode.

У F2FS блок запису каталогу розміром 4 КБ (dentry, directory entry) і містить бітову мапу, масиви слотів та відповідні імена. Бітова мапа вказує, чи є кожен слот дійсним. Слот займає 11 байт та містить хеш-значення, номер inode, довжину імені та тип файлу (наприклад, звичайний файл, каталог або символічне посилання). Файл каталогу створює багаторівневі хеш-таблиці для ефективного управління великою кількістю записів dentries. Коли F2FS шукає певне ім'я файлу в каталозі, спочатку обчислює хеш-значення імені

файлу. Потім послідовно проходить побудовані хеш-таблиці від рівня 0 до максимального рівня, записаного в inode. На кожному рівні сканує один сегмент з двох або чотирьох блоків dentry, що дає складність O(log(# dentries)). Щоб швидше знайти dentry, він послідовно порівнює бітову мапу, хеш-значення та ім'я файлу. Якщо перевага надається великим каталогам (наприклад, у серверному середовищі), користувачі можуть налаштувати F2FS так, щоб спочатку виділяти простір для багатьох dentries. Завдяки більшій хеш-таблиці на низьких рівнях F2FS швидше досягає цільового dentry [33, 34].

2.7 Порівняння характеристик, функціональності та сумісності файлових систем

Для порівняння файлових систем побудовано чотири таблиці. Таблиця 2.10 містить порівняльний аналіз обмежень щодо довжини імен файлів, використаного кодування символів, а також максимальної довжини шляху. Таблиця 2.11 – містить огляд обмежень пов’язаних з максимальним розміром файлів і томів, які може підтримувати певна файлова система. Таблиця 2.12 – містить функціональні можливості кожної з розглянутих файлових систем. Таблиця 2.13 – демонструє сумісність з операційними системами.

Таблиця 2.12 – Порівняння обмежень іменування в файлових системах [9 - 11]

ФС	*Макс. довжина імені файлу*	*Кодування символів*	*Макс. довжина імені шляху*
exFAT	255 символів	UTF-16	32,760
NTFS v3.1	255 символів	UTF-16	32,760
Ext4	255 байт	ASCII	4096 байт (обмеження ядра Linux)

Кінець таблиці 2.12

ФС	*Макс. довжина імені файлу*	*Кодування символів*	*Макс. довжина імені шляху*
APFS	255 символів	UTF-8	Не обмежено формально (реально ≈ 1024 символів)
Btrfs	255 байт	ASCII	4096 байт (обмеження ядра Linux)
F2FS	255 байт	UTF-8	4096 байт (обмеження ядра Linux)

Існує певна схожість, коли йдеться про імена файлів. Більшість файлових систем, такі як exFAT, NTFS, APFS підтримують імена файлів довжиною до 255 символів. Однак, системи ext4, Btrfs і F2FS встановлюють обмеження не за кількістю символів, а за кількістю байтів — 255 байт. Це дорівнює 255 символам у випадку кодування ASCII, однак для багатобайтових кодувань, таких як Unicode, фактична кількість допустимих символів може бути меншою [10, 35]. Також максимальна довжина шляхів обмежується не самими файловими системами, а ядром Linux — на рівні 4096 байт [36]. Максимальна довжина повного шляху для файлових систем FAT та NTFS у середовищі Windows обмежується значенням 32 760 символів у форматі Unicode [37]. У файловій системі APFS не передбачено формального обмеження на довжину шляху, однак на практиці воно рідко перевищує 1024 символи [38].

Таблиця 2.13 – Максимальні розміри файлів і томів [9, 10]

ФС	*Максимальний розмір файлу*	*Максимальний розмір тому*
exFAT	128 ПБ	128 ПБ
NTFS v3.1	До 16 ЕБ	До 16 ЕБ
Ext4	16 ТБ	1 ЕБ
APFS	8 ЕБ	8 ЕБ
Btrfs	16 ЕБ	16 ЕБ
F2FS	До 16 ТБ	До 16 ТБ

Сучасні файлові системи демонструють значне розширення меж підтримуваних обсягів даних. Зокрема Btrfs, NTFS v3.1 та APFS, підтримують файли та томи розміром до 8 або, навіть, 16 ексабайтів. Однак в реальних умовах ці межі зазвичай обмежуються можливостями операційної системи [15, 18]. Ext4 суттєво перевершує свої минулі версії завдяки впровадженню extent-структур: вона підтримує файли до 16 ТБ і томи до 1 ексабайта [10], але поступається іншим системам. F2FS має менші межі - до 16 ТБ для файлів і томів. Це обмеження пов’язане з особливостями внутрішньої архітектури системи, зокрема розміром блоків, та розраховане на використання у мобільних та вбудованих пристроях [32].

Таблиця 2.14 – Порівняння функціональних можливостей файлових систем [9, 10]

	exFAT	*NTFS v3.1*	*Ext4*	*APFS*	*Btrfs*	*F2FS*
Журналю-вання	Ні	Так	Так	Так	Так	Так
Шифрува-ння	Ні	Так	Ні	Так	Ні	Ні
Стиснення	Ні	Так	Ні	Так	Так	Так
Copy-on-Write	Ні	Ні	Ні	Так	Так	Так
Знімки	Ні	Так	Ні	Так	Так	Ні

Дані таблиці 2.14 свідчать про те, що exFAT є однією з найпростіших файлових систем, оскільки вона не підтримує жодної з розглянутих функцій. Така спрощеність не обов’язково є недоліком: завдяки своїй простоті і, незважаючи на свій вік, вони все ще дуже популярні для використання на портативних пристроях. Ext4 демонструє трохи кращу функціональність.

Найбільш багатофункціональними файловими системами є APFS і Btrfs, які виділяються серед інших систем завдяки широкому спектру

сучасних технологій. NTFS та F2FS показують достатньо високу функціональність.

Таблиця 2.15 – Підтримка операційними системами

ФС	*Windows*	*Linux*	*macOS*	*Android*	*iOS*
exFAT	Так	Так	Так	Так	Так
NTFS v3.1	Так	читання або СПЗ	читання або СПЗ	Ні	Ні
Ext4	СПЗ	Так	СПЗ	Так	Ні
APFS	СПЗ	СПЗ	Так	Ні	Так
Btrfs	СПЗ	Так	СПЗ	Ні	Ні
F2FS	Ні	Так	Ні	Так	Ні

Пояснення до таблиці 2.15:

- Так – операційна система підтримує файлову систему;
- СПЗ – підтримка можлива лише за умови використання стороннього програмного забезпечення або драйверів;
- Ні – відсутність офіційної підтримки та знань про стороннє програмне забезпечення.

Аналіз таблиці 2.15 показує, що exFAT підтримується всіма основними операційними системами. NTFS (v3.1) є типова для Windows, а підтримка в інших системах реалізується або обмежено - лише читання в macOS і Linux або через стороннє програмне забезпечення. Повноцінна робота з цими файловими системами за межами Windows здебільшого недоступна без додаткових драйверів [10, 18]. Файлова система APFS демонструє замкненість у межах екосистеми Apple. Вона підтримується виключно в macOS та iOS [10, 24]. Ext4 є типовою ФС для Linux (включаючи Android). У Windows та macOS доступ можливий за допомогою сторонніх інструментів [39]. Btrfs має повноцінну підтримку лише в Linux, що зумовлено її походженням. У Windows та macOS вона може використовуватися тільки через неофіційні інструменти, які не забезпечують повного функціоналу, в Android підтримка зовсім відсутня. F2FS найбільш активно використовується

в Android і Linux. Інші операційні системи, не підтримують її на офіційному рівні і немає відомості про стороннє програмне забезпечення для забезпечення сумісності [10].

Для файлових систем, що не підтримуються нативно певною операційною системою, сумісність може бути забезпечена за допомогою стороннього програмного забезпечення (наприклад, продуктів компанії Paragon Software) або спеціалізованих драйверів — як розроблених третіми сторонами, так і тих, що інтегровані в операційну систему. Зокрема, драйвер NTFS3, включений до ядра Linux, забезпечує підтримку файлової системи NTFS до версії 3.1 [40, 41]. Використання таких засобів дозволяє здійснювати повноцінний або обмежений доступ до даних у середовищах різних операційних систем.

2.8 Узагальнення та порівняння архітектурних і структурних особливостей досліджених файлових систем

Файлова система exFAT побудована на основі традиційної FAT-архітектури з поділом розділу на три основні області: Boot Region, FAT Region та Data Region. Її особливістю є проста кластерна адресація, відсутність журналювання та сумісність з великою кількістю платформ. Основна перевага — збереження простоти при підтримці великих томів і файлів.

NTFS побудована навколо центральної таблиці MFT, у якій кожен файл і директорія мають власний запис. Метадані організовано у вигляді атрибутів, частина з яких зберігається безпосередньо в MFT (резидентно), а частина — у зовнішніх екстентах (нерезидентно). Каталоги організовані через B-дерева.

APFS реалізує контейнерну структуру, в межах якої існує кілька логічних томів. Усі об’єкти системи (файли, каталоги, метадані) зберігаються у вигляді об'єктів із унікальними ідентифікаторами. Для побудови всієї структури використовуються B-дерева та карти об’єктів (Object Maps).

Ext4 має блочну ієрархічну структуру з групами блоків, кожна з яких містить таблицю інодів, суперблок і бітові мапи. Для зберігання даних застосовуються екстенти, а структура метаданих централізовано зберігається в інодах.

Btrfs побудована виключно на системі B-дерев. Для кожного типу інформації (метадані, дані, іноди, контрольні суми) використовуються окремі спеціалізовані дерева — зокрема ROOT_TREE, EXTENT_TREE, FS_TREE. Дані й метадані зберігаються у листових вузлах B-дерев із чіткою логікою сортування.

F2FS має сегментну архітектуру (LFS-модель), оптимізовану для флеш-пам’яті. Основними структурами є NAT (Node Address Table) і SIT (Segment Information Table), що забезпечують індексацію та управління сегментами. Дані організовано через вузли (inode, direct, indirect).

Таблиця 2.16 – Узагальнення та порівняння будови сучасних файлових систем

ФС	*Основний принцип організації даних*	*Методи адресації*	*Управління простором*	*Особливості*
exFAT	Таблична модель (FAT), поділ на Boot, FAT і Data-регіони	Кластерна адресація, FAT-ланцюжки.	Allocation Bitmap (бітова мапа).	Таблиця FAT.
NTFS	Централізована організація даних навколо MFT. Використовує B-дерева для каталогів та індексів.	Екстенти, резидентні/ нерезидентні атрибути.	$Bitmap – бітова мапа.	Використання MFT. Системні файли. Резидентні та нерезидентні атрибути.

Кінець таблиці 2.16

ФС	*Основний принцип організації даних*	*Методи адресації*	*Управління простором*	*Особливості*
APFS	Контейнерна модель з томами, використання B-дерев для всіх метаданих. Принцип Copy-on-Write. Має карти об'єктів (Object Maps).	Карти об'єктів (Object Maps), віртуальна та фізична адресація блоків.	Бітові мапи (Space Manager).	Карти об'єктів. Контейнери з внутрішніми томами.
Ext4	Блокова ієрархічна організація з групами блоків, таблицями інодів, суперблоками.	Екстенти, таблиці інодів для файлів і каталогів.	Бітові мапи блоків та інодів.	Групова структура блоків. екстенти як основний спосіб адресації даних.
Btrfs	Повністю структурована навколо B-дерев. Copy-on-Write модель запису змін. Всі дані і метадані організовані у спеціалізовані B-дерева.	Екстенти та логічна адресація через B-дерева (EXTENT_TREE та CHUNK_TREE).	EXTENT_TREE та CHUNK_TREE.	B-дерева для кожного типу даних. Кореневе дерево (ROOT_TREE) для зберігання адрес інших дерев.
F2FS	LFS-модель з використанням сегментів. Вузли (inode, direct, indirect) організовані для індексації та зберігання метаданих.	NAT-таблиця для адресації вузлів.	Segment Info Table, бітові мапи блоків у сегментах.	Сегментна структура. NAT-таблиця. Ієрархія вузлів.

З таблиці 2.16 можна зробити низку узагальнень щодо еволюції архітектурних рішень у сучасних файлових системах.

По-перше, чітко простежується тенденція до ускладнення та гнучкості організації даних. Якщо exFAT ще зберігає класичний табличний підхід із використанням FAT-таблиці, то всі більш сучасніші файлові системи (NTFS, APFS, Btrfs, F2FS, Ext4) реалізують багаторівневу, модульну чи ієрархічну структуру.

Серед методів розподілу простору та адресації даних домінують два основні підходи: використання екстентів та структур B-дерева. Такі системи,

як NTFS, Ext4, Btrfs, APFS, використовують екстенти, для зниження фрагментації та спрощення керування великими файлами. Архітектура B-дерев стає стандартом для організації метаданих і каталогів, оскільки дозволяє швидко здійснювати пошук і змінювати файлову ієрархію. Особливо це виражено у Btrfs та APFS, де B-дерева лежать в основі всієї файлової структури.

Для управління вільним простором, більшість ФС використовують стандартні бітові мапи. Водночас також застосовуються більш складніші механізми - екстентні дерева в Btrfs та багаторівневі бітові мапи в APFS.

Архітектурною тенденцією є поширення підходу Copy-on-Write (CoW), який є ключовим для Btrfs, APFS і частково F2FS. Завдяки цьому метадані та дані ніколи не перезаписуються безпосередньо, а завжди створюється нова версія.

Простежується прагнення до модульності. Наприклад, APFS організовує дисковий простір у вигляді контейнерів із багатьма незалежними томами, а Btrfs дозволяє створювати підтоми й миттєві знімки завдяки єдиній ієрархії дерев.

Найбільш поширеними й ефективними рішеннями сучасних файлових систем стали: ієрархічна організація на основі B-дерев, адресація за допомогою екстентів, застосування механізму Copy-on-Write, а також багаторівневі методи управління простором. Традиційні моделі (таблична адресація й проста бітова мапа) залишаються актуальними, проте поступово замінюються складнішими і надійнішими рішеннями.

Висновки до розділу 2

У другому розділі проведено комплексне дослідження архітектурних та структурних особливостей сучасних файлових систем, які широко

застосовуються в операційних системах Windows, macOS, Linux та Android. Розглянуто детальні особливості будови файлових систем exFAT, NTFS, APFS, ext4, Btrfs та F2FS. Проведено аналіз внутрішньої структури, механізмів адресації, організації метаданих, а також розглянуто особливості підтримки додаткових функцій, таких як журналювання, шифрування, стиснення та використання Copy-on-Write.

Виконано порівняльний аналіз досліджуваних файлових систем за їх функціональними можливостями, обмеженнями на розмір файлів і томів, сумісністю з різними операційними системами. Узагальнено сучасні тенденції розвитку файлових систем, серед яких виділено домінування B-дерев для управління даними, використання екстентів та механізмів Copy-on-Write.

Загалом, у цьому розділі визначено основні архітектурні підходи та технологічні рішення, які забезпечують високу продуктивність, масштабованість та адаптивність сучасних файлових систем відповідно до вимог сучасних пристроїв та операційних середовищ.

РОЗДІЛ 3 ЗАСТОСУВАННЯ ШТУЧНОГО ІНТЕЛЕКТУ В ФАЙЛОВИХ СИСТЕМАХ

У цьому розділі проаналізовано основні напрями впровадження штучного інтелекту у файлові системи. Також розроблено програму, яка реалізує застосування ШІ у файлових системах.

3.1 Можливі напрями застосування ШІ у файлових системах

Одним з напрямків застосування ШІ є впровадження голосового асистента для управління файловою системою, який дає змогу користувачам здійснювати типові операції - створення, перейменування, видалення файлів і тек — за допомогою голосових команд.

Архітектура такої системи складається з кількох ключових модулів:

- модуль розпізнавання мовлення відповідає за перетворення голосових команд у текст;
- модуль обробки природної мови (NLP) аналізує текст і визначає наміри користувача;
- модуль доступу до файлової системи реалізує безпосереднє виконання команд [42].

Прикладом голосового асистента на основі штучного інтелекту, який можна адаптувати для управління файловою системою є проєкт Braina, який дозволяє задавати голосові команди для виконання дій над файлами в середовищі Windows. Асистент підтримує створення власних голосових сценаріїв, розпізнає інструкції користувача і може виконувати такі дії, як відкриття, перейменування, переміщення чи видалення файлів і папок [43].

Штучний інтелект також активно застосовується для забезпечення безпеки файлових систем, зокрема у виявленні аномальної активності та потенційно шкідливих дій, які можуть свідчити про наявність зловмисного програмного забезпечення або несанкціонованого доступу.

Розробки у цьому напрямі демонструє такий інструмент, як Microsoft Defender for Endpoint, який використовує ШІ та машинне навчання для аналізу поведінки системи, прогнозування загроз і блокування підозрілих дій у файловій системі [44].

3.2 Розробка програмного забезпечення

Для демонстрації прикладу застосування штучного інтелекту у файлових системах розроблено програму, яка автоматично аналізує структуру директорій, вміст файлів та формує звіт на основі великої мовної моделі.

3.2.1 Велика мовна модель

Велика мовна модель (Large language model, LLM) - це тип програми штучного інтелекту, навчена за допомогою самокерованого машинного навчання на величезній кількості тексту, призначена для завдань обробки природної мови, зокрема генерації мови. Найбільшими та найпотужнішими LLM є генеративні попередньо навчені трансформери (GPT), які широко використовуються в генеративних чат-ботах. LLM можна точно налаштувати для конкретних завдань або керувати швидким проектуванням [45].

Прикладами реальних LLM є ChatGPT (від OpenAI), Gemini (Google), Llama (Meta) та Bing Chat (Microsoft).

3.2.2 Вибір мови програмування, бібліотек та моделі

Для розробки програмного забезпечення вибрано мову програмування Python, мовну модель Llama 3.1 та середовище розробки Google Colab.

Python – мова високого рівня з простою та зрозумілою синтаксичною структурою [46]. Використовується для розробки в сфері штучного інтелекту. Вона підтримує велику кількість бібліотек для обробки тексту, роботи з файлами, а також глибокого навчання.

LLAMA (Meta) – серія відкритих великих мовних моделей, розроблена компанією Meta AI. У роботі використано модель Meta-Llama-3.1-70B-Instruct, яка призначена для генерації інструктивних відповідей та підтримує сценарії аналізу структурованої текстової інформації. Модель має 70 мільярдів параметрів та працює в режимі генерації за заданим запитом (prompt) [47].

TF-IDF – класичний статистичний підхід, реалізований у вигляді Python-бібліотеки scikit-learn, використовується для вилучення ключових слів та словосполучень з текстових документів.

BLIP – модель для перетворення візуальної інформації на текстовий опис [48].

Google Colaboratory (Colab) – середовище програмування, яке підтримує апаратне прискорення за допомогою графічних процесорів (GPU) та забезпечує доступ до значного обсягу оперативної і відеопам’яті.

Hugging Face — це відкрита платформа для розробки, поширення та розгортання моделей штучного інтелекту в реальних застосунках.

Використовується для інтеграції з великою мовною моделлю Llama через API.

Використані бібліотеки:

- transformers – для завантаження та взаємодії з моделлю Blip;
- python-docx, PyMuPDF – для вилучення тексту з DOCX та PDF файлів;
- PIL (Pillow) – для роботи з зображеннями;
- os, datetime – для обходу файлової системи та збору інформації про файли;
- requests - для виклику мовної моделі за допомогою API;
- nltk – для видалення стоп-слів;
- re – для фільтрації ключових слів за регулярними виразами;
- scikit-learn – для реалізації TF-IDF.

3.2.3 Опис методів та алгоритму роботи програми

extract_text - здійснює зчитування тексту з файлів типу TXT, MD, CSV, JSON, ігноруючи помилки кодування.

extract_pdf_text - обробляє PDF-документи за допомогою бібліотеки PyMuPDF та повертає зчитаний текст.

extract_docx_text - зчитує текстовий вміст з DOCX-документів за допомогою бібліотеки python-docx.

extract_from_file - визначає тип файлу та викликає відповідну функцію вилучення тексту, об’єднуючи обробку різних текстових форматів.

save_to_file - зберігає результати аналізу, сформовані великою мовною моделлю, у текстовий файл для подальшого опрацювання.

get_keywords - виконує обчислення найрелевантніших ключових слів у текстових документах за допомогою методу TF-IDF.

get_image_description - генерує короткий текстовий опис зображення із використанням моделі BLIP та фільтрує результат від стоп-слів.

get_file_paths - рекурсивно обходить цільову директорію та повертає перелік усіх файлів і підкаталогів із підрахунком кількості.

get_full_fs_structure - створює повну структуру файлової системи, визначає тип кожного об'єкта (текст, зображення, інше) та додає інформацію (розмір, ключові слова, опис).

data - форматує зібрану інформацію про структуру директорії у вигляді списку з відносним шляхом, типом, розміром і коротким описом.

generated_prompt - створюється запит (prompt) до великої мовної моделі, що містить метадані директорії та опис вмісту кожного об’єкта.

query - здійснює HTTP-запит до мовної моделі Llama через API платформи Hugging Face для отримання аналітичного звіту.

Отже, алгоритм автоматичного аналізу вмісту директорії складається з послідовних етапів.

1. Вказується шлях до цільової директорії.
2. Виконується автоматичний перебір усіх файлів та підкаталогів обраної директорії.
3. Визначається тип кожного файлу: текстовий, зображення або інший.
4. Зчитуються основні метадані кожного файлу.
5. Якщо файл є текстовим - виконується вилучення тексту та формування ключових слів або фраз за допомогою TF-IDF. Якщо зображення – модель BLIP генерує короткий опис зображення.
6. Генерується текстовий запит (prompt) для мовної моделі, що містить перелік файлів, їх типи, ключові слова чи описи.
7. Prompt передається у LLM, після чого формується аналітичний звіт.
8. Аналітичний звіт виводиться на екран для перегляду користувачем та зберігається у текстовому файлі analysis_result.

У результаті роботи програма формує стислий звіт, який містить:

- наявні дублікати або незвичні файли;
- загальний опис вмісту директорії;
- коротку статистику про типи файлів та їх розмір.

3.3 Проведення експерименту та аналіз результатів

Для проведення експерименту сформовано створено експериментальну директорію test_folder, що містить 9 вкладених підкаталогів із 32 файлами різного типу. Основу наповнення становлять текстові документи у форматах pdf, docx та txt, графічні файли типів jpeg і png, також добавлено декілька файлів типу wbep та zip. Тематика файлів здебільшого пов’язана з файловими системами, проте наявні також матеріали на іншу тему, для коректної перевірки роботи програми. Вхідний текстовий запит (prompt) зображено на рис. 3.1 та рис. 3.2. Отриманий результат аналізу, сформований мовною моделлю, представлено на рис. 3.3.

```
Directory name: Test_files
Directory path: /content/drive/MyDrive/Test_files

You are a research assistant specializing in scientific analysis of file system directories. Your task is to
produce an analytical summary of the contents of the provided directory, based on the following requirements:

1. Summarize the main topic and subtopics, approaches, and types of content in the files, using extracted
keywords and image descriptions, but avoid repeating themes or phrases.
2. Identify and briefly describe any anomalies, duplicates, empty folders, or unusual files.
3. Include a short statistical overview of the distribution of file types and the total volume of files
(number and size).
4. Draw brief, clear scientific conclusions about the contents and organization of the directory.
5. Do not use bullet points, tables, or bold formatting. Structure your response into coherent paragraphs.
Avoid any introductions or conclusions unrelated to the provided data.

Directory information:
Main directory path: /content/drive/MyDrive/Test_files
Number of files: 32
Number of subfolders: 9

Directory structure (each line: relative_path, type (file/folder), size in KB, content summary (keywords or
image description)):
```

Рисунок 3.1 – Основна частина вхідного запиту

```
reference/Andrew S. Tanenbaum - Modern Operating Systems.pdf, file, 383.9, keywords: file, disk, files, block,
blocks, directory, systems, node, file systems, read
reference/An_Introduction_to_the_exFAT_File_System_and_How_to_Hide_Data_Within.pdf, file, 734.6, keywords:
file, metadata, data, cluster, embedding, exfat, files, used, file, size
reference/File System Forensic Analysis ( PDFDrive )-129-338.pdf, file, 2513.8, keywords: file, data, entry,
directory, attribute, block, bytes, entries, mft, size
reference/2.1.png, file, 73.4, dsc: diagram block different levels fat fat
reference/Windows Internals Part 2_6th Edition-413-520.pdf, file, 3842.8, keywords: file, ntfs, data, volume,
log, files, directory, windows, disk, record
reference/APFS.pdf, file, 12257.5, keywords: object, id, value, tree, apfs, fig, key, node, file, bytes
reference/Forensic analysis of B-tree file system (Btrfs).pdf, file, 7376.0, keywords: file, data, node, nodes,
btrfs, tree, forensic, leaf, fig, items
reference/F2FS_A_New_File_System_for_Flash_Storage.pdf, file, 1111.3, keywords: data, file, blocks,
performance, writes, storage, flash, logging, block, node
reference/Ext4.pdf, file, 232.1, keywords: file, time, files, number, data, block, allocation, systems,
performances, results
Projects/MKR.ipynb, file, 2.9
Projects/Comments.zip, file, 2096.4
images/Nature, folder, 4.0
images/2.1.png, file, 73.4, dsc: diagram block different levels fat fat
images/2.5.png, file, 34.9, dsc: diagram table number columns number columns
```

Рисунок 3.2 – Частина вхідного запиту з переліком вмісту директорії

```
LLM analysis report:

The main topic of the directory appears to be a collection of files and folders related to various subjects,
including finance, cryptocurrency, music, projects, and reference materials. The finance subtopic is further
divided into cryptocurrency and bitcoin, with files containing information on cryptocurrency exchanges,
wallets, and protocols. The images folder contains a mix of diagrams and nature photography, while the music
folder consists of audio files. The projects folder contains a Jupyter notebook and a zip file, and the
reference folder is filled with PDF files on file system forensic analysis, operating systems, and various
file systems.

Anomalies in the directory include the presence of duplicate files, such as the "2.1.png" file found in both
the images and reference folders. There are also empty folders, such as the "music" folder, which contains no
subfolders or files. Unusual files include the "Comments.zip" file in the projects folder, which is not
accompanied by any other files or folders.

The distribution of file types in the directory is as follows: PDF files make up the majority, with 14 files
totaling 23,419.1 KB in size. Image files come in second, with 9 files totaling 2,121.9 KB in size. Audio
files make up the smallest portion, with 3 files totaling 20,675.9 KB in size. The total volume of files in
the directory is 32 files, totaling 46,217 KB in size.

Based on the analysis, it appears that the directory is a collection of miscellaneous files and folders, with
a focus on finance and cryptocurrency. The presence of duplicate files and empty folders suggests a lack of
organization and maintenance. The large number of PDF files in the reference folder suggests a focus on
research and education. Overall, the directory appears to be a personal collection of files, rather than a
professionally organized repository.
```

Рисунок 3.3 – Результати аналізу

Програма змогла розпізнати наявність текстових документів, зображень та інших файлів. Коректно виділила, основну тематику зібраних даних - це інформація, присвячена файловим системам. Також визначила наявність файлів на другорядні теми.

Успішно виявила дублікат файлу 2.1.png, який навмисно був розміщений у двох різних підкаталогах, та ідентифікувала цю аномалію у підсумковому звіті. Відзначила наявність єдиного zip-архіву серед усіх файлів.

Щодо статистичної інформації, програма змогла визначити домінування PDF-файлів у загальному обсязі директорії. Не пройшла повз інші типи даних, таких як графічні та аудіофайли, представлені в меншій кількості. Також виділила, що аудіофайли характеризуються значним розміром.

Висновки до розділу 3

У цьому розділі досліджено застосування штучного інтелекту в сучасних файлових системах. Розроблено програмне забезпечення на основі великої мовної моделі (LLM), яке автоматизовано вилучає ключові слова з текстових файлів, формує описи зображень та на основі отриманих даних генерує узагальнений звіт про структуру директорії та вміст файлів.

Проведений експеримент продемонстрував, що програма здатна коректно визначати типи файлів, виявляти дублікати й незвичні файли та формувати аналітичні звіти із зазначенням статистичної інформації про склад і розміри файлів у директоріях.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У цьому розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на аналізі файлових систем.

Також в даному дослідженні показано різні варіанти використання програмного забезпечення для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність із поставленими завданнями. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу реалізації програмного продукту, призначеного для

аналізу вмісту каталогів, включно з аналізом вмісту файлів та опису зображень. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп'ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка можливого програмного продукту, яка дозволяє аналізувати різні характеристики, що безпосередньо впливають на стійкість підприємства. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування;

F_2 – вибір способу реалізацій алгоритмів;

F_3 – вибір середовища програмування.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

a) Python

b) C++

Функція F_2.

a) Застосування готових бібліотек

b) Створення власних алгоритмів

Функція F_3:

a) Google Colab

b) Visual Studio Code

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

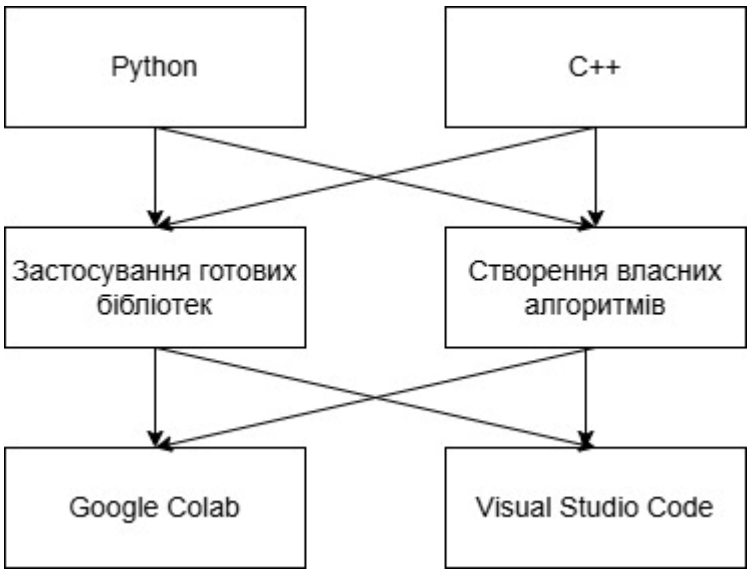


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Таблиця 4.1 – Позитивно-негативна матриця

Функції	*Варіанти реалізації*	*Переваги*	*Недоліки*
F_1	*А*	Простий синтаксис, велика кількість готових бібліотек, зручність для швидкої розробки	Складність мови, довший час розробки
	Б	Висока продуктивність, контроль пам’яті	Складність мови, довший час розробки
F_2	*А*	Прискорення розробки, стабільність, мінімізація ризику помилок	Обмежена гнучкість, складність адаптації під нестандартні задачі
	Б	Можливість точного налаштування під задачі, вища оптимізація	Підвищена трудомісткість, потреба в додатковому тестуванні
F_3	*А*	Простота початку роботи, інтеграція з хмарними обчисленнями, зручність для Python	Потреба в стабільному інтернет-з’єднанні, обмежена підтримка інших мов програмування
	Б	Потужна IDE, підтримка багатьох мов, розширення для налагодження та аналізу	Необхідність налаштування середовища, вища складність для початківців

Функція F_1: перевагу даємо простоті та зручності мови програмування Python; варіант Б має бути відкинутий. Функція F_2: перевагу даємо меншим часовим витратам на розробку програмного продукту; для спрощення роботи

варіант Б має бути відкинутий. Функція F_3: програма допускає обрання обох варіантів; можливо використати варіанти А чи Б.

Таким чином, будемо розглядати такий варіанти використання програмного продукту

$$F_1\text{а} - F_2\text{а} - F_3\text{а},$$
$$F_1\text{а} - F_2\text{а} - F_3\text{б}.$$

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- *X*1 – швидкодія мови програмування;
- *X*2 – об’єм пам’яті для обчислень та збереження даних;
- *X*3 – час виконання аналізу;
- *X*4 – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 – Основні параметри програмного продукту

Назва параметра	*Умовні позначення*	*Одиниці виміру*	*Значення параметра*		
			гірші	*середні*	*кращі*
Швидкодія мови програмування	*X1*	оп/мс	5	15	40
Об’єм пам’яті	*X2*	МБ	500	300	150
Час виконання аналізу	*X3*	мс	800	500	200
Об’єм програмного коду	*X4*	Рядки коду	400	200	100

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

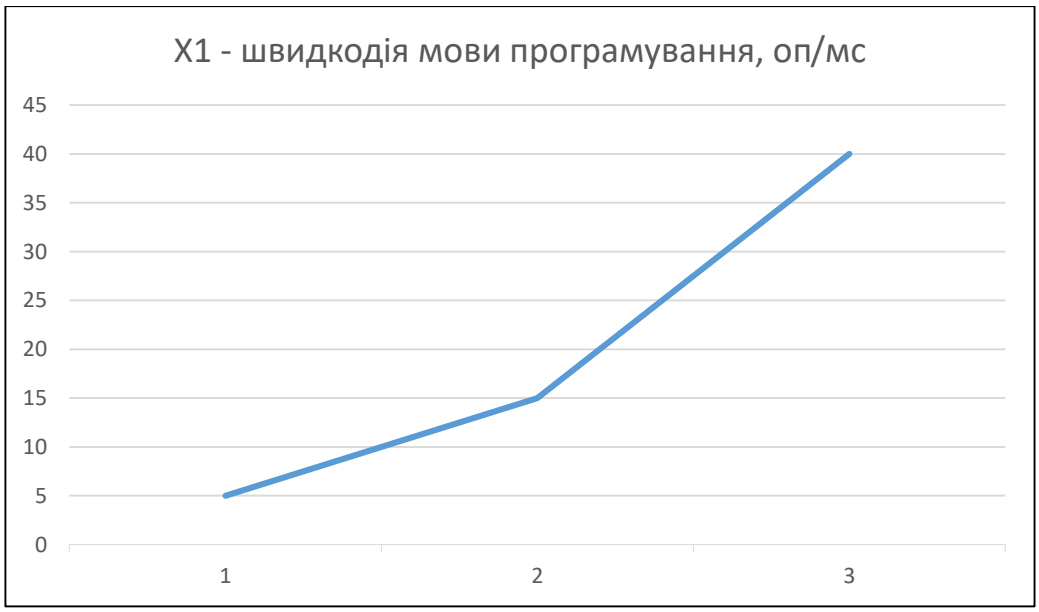


Рисунок 4.2 – *X*1, швидкодія мови програмування

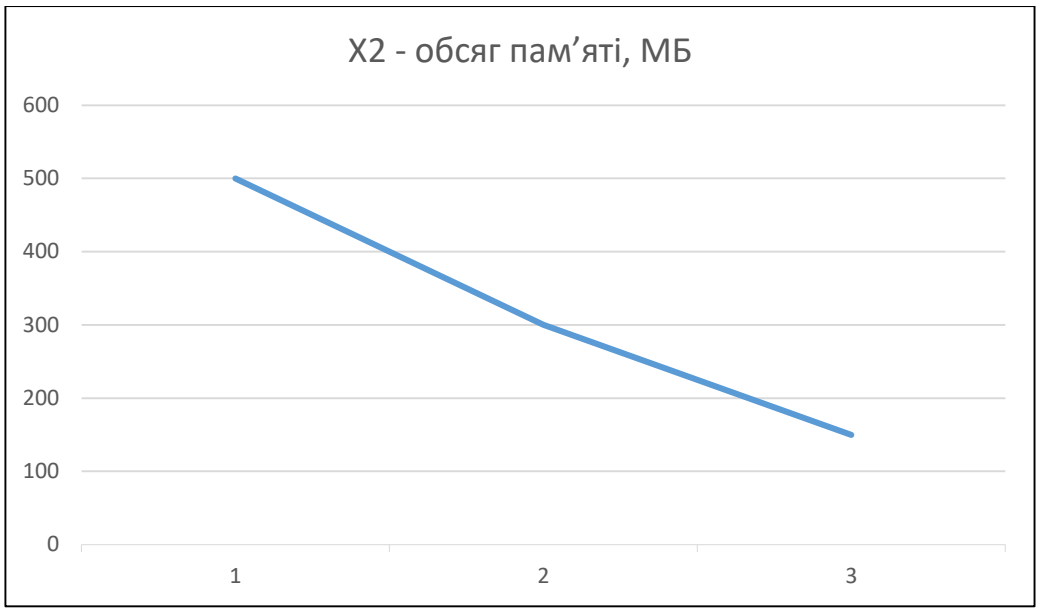


Рисунок 4.3 – *X*2, об’єм пам’яті

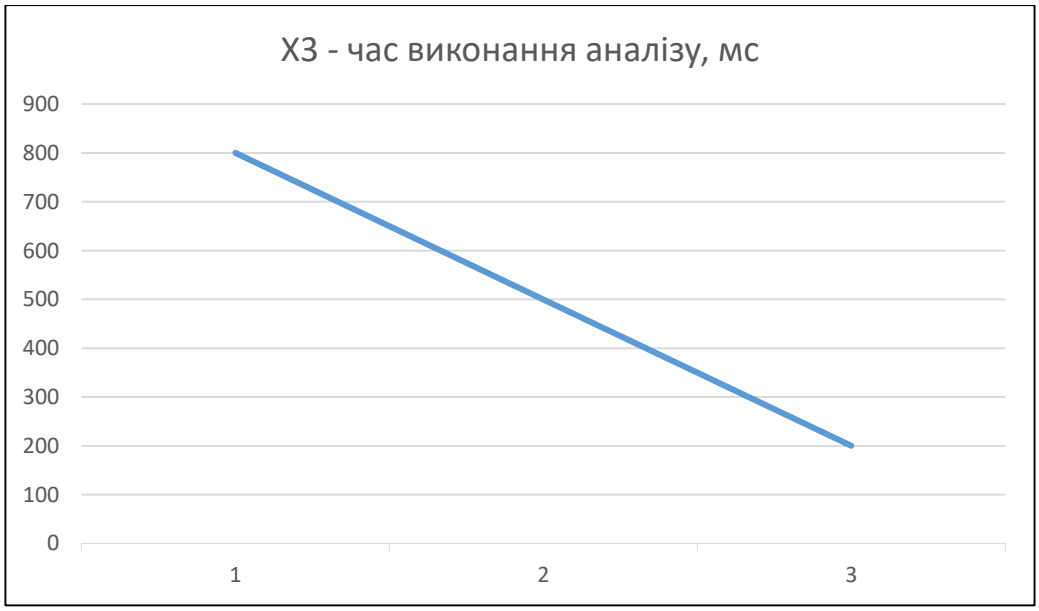


Рисунок 4.4 – *X*3, час виконання аналізу

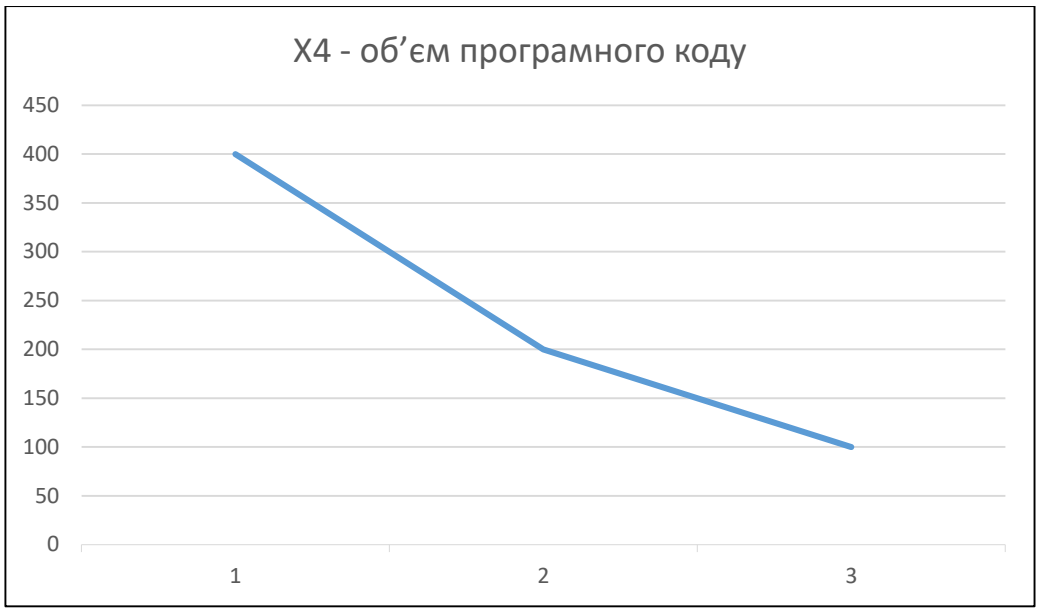


Рисунок 4.5 – *X*4, об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який використовує обрані моделі для аналізу вмісту каталогів.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значущості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Основні параметри програмного продукту

Позначення параметра	*Назва параметра*	*Одиниці виміру*	*Ранг параметра за оцінкою експерта*							*Сума рангів* R_i	*Відхилення* Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	1	1	1	1	1	1	2	10	-7,5	56,25
X2	Об’єм пам’яті для обчислень та збереження даних	МБ	3	4	3	3	4	3	4	24	6,5	42,25
X3	Час виконання аналізу	мс	4	3	2	4	2	2	1	11	-6,5	42,25
X4	Потенційний об’єм програмного коду	Кількість	2	1	4	2	3	4	3	25	7,5	56,25
	Разом		10	10	10	10	10	10	10	70	0	197

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70,$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5.$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T,$$

сума відхилень по всім параметрам повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 197.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 197}{7^2(4^3 - 4)} = 0{,}754 > W_k = 0{,}67.$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 – Попарне порівняння параметрів.

Параметри	*Експерти*							*Кінцева оцінка*	*Числове значення*
	1	2	3	4	5	6	7		
X1 і X2	<	<	<	<	<	<	<	<	0,5
X1 і X3	<	>	>	<	<	<	>	<	0,5
X1 і X4	<	<	<	<	<	<	<	<	0,5
X2 і X3	>	>	>	>	>	>	>	>	1,5
X2 і X4	<	>	<	<	>	<	<	<	0,5
X3 і X4	<	<	<	<	<	<	<	<	0,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j, \\ 1.0 \text{ при } X_i = X_j, \\ 0.5 \text{ при } X_i < X_j. \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю $A=\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{\text{вi}} = \frac{b_i}{\sum_{i=1}^{n} b_i},$$

$$b_i = \sum_{i=1}^{N} a_{ij}\,.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{вi}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'},$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j .$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітерація		Друга ітерація		Третя ітерація	
	X1	*X2*	*X3*	*X4*	b_i	$K_{\text{вi}}$	b_i^1	$K_{\text{вi}}^1$	b_i^2	$K_{\text{вi}}^2$
X1	1	0,5	0,5	0,5	2,5	0,16	9,25	0,16	34,125	0,16
X2	1,5	1	1,5	0,5	4,5	0,28	16,25	0,28	59,125	0,28
X3	1,5	0,5	1	0,5	3,5	0,22	12,25	0,21	41,875	0,2
X4	1,5	1,5	1,5	1	5,5	0,34	21,25	0,35	77,875	0,36
Всього:					16	1	59	1	213	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (Об’єм пам’яті для обчислень та збереження даних), *X3* (Час виконання аналізу) та *X4* (Потенційний об’єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (Швидкодія мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}i,j} B_{i,j},$$

де n – кількість параметрів;

$K_{\text{в}i}$– коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Функція	*Варіант реалізації функції*	*Параметр*	*Абсолютне значення параметра*	*Бальна оцінка параметра*	*Коефіцієнт вагомості параметра*	*Коефіцієнт рівня якості*
F_1	А	*X1*	15	9	0.18	1.62
F_2	А	*X2*	150	10	0.27	2.7
F_3	А	*X3*	200	6	0.34	2.04
	Б	*X4*	100	3	0.11	0.33

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \cdots + K_{\text{ТУ}}[F_{zk}].$$

визначаємо рівень якості кожного з варіантів:

$$\text{К}_{\text{К1}} = 1.62 + 2.7 + 2.04 = 6.36,$$
$$\text{К}_{\text{К2}} = 1.62 + 2.7 + 0.33 = 4.65.$$

Як видно з розрахунків, кращим є 1 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості використання ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М},$$

де T_P – трудомісткість використання ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

$K_М$ – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P = 37 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.8. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.9. Тоді загальна трудомісткість програмування першого завдання розраховується як $T_1 = 37 \cdot 1.8 \cdot 0.9 = 59{,}94$ людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто T_P =29 людино-днів, $К_П$ =0.9, $К_{СК}$ = 1, $К_{СТ}$ =0.8. Таким чином, загальна трудомісткість $T_2 = 29 \cdot 0.9 \cdot 0.8 = 20.88$ людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість, розрахунки наведено в формулах (4.18) та (4.19).

$$T_{\text{I}} = (59{,}94 + 20.88 + 4.8 + 20.88) \cdot 8 = 852 \text{ людино} - \text{годин.}$$

$$T_{\text{II}} = (59{,}94 + 20.88 + 6.91 + 20.88) \cdot 8 = 868{,}88 \text{ людино} - \text{годин.}$$

Найбільш високу трудомісткість має варіант II.

У розробці беруть участь один програміст з окладом 25000 грн. та один аналітик в області даних з окладом 19000 грн. Визначаємо середню зарплату за годину за формулою:

$$С_ч = \frac{М}{T_m \cdot t} \text{ грн,}$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

Тоді, розраховуємо заробітну плату за формулою:

$$С_ч = \frac{25000 + 19000}{3 \cdot 21 \cdot 8} = 87{,}3 \text{ грн.}$$

$$С_{зп} = С_ч \cdot Т_i \cdot К_д,$$

де $С_ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ВІД} = 87{,}3 \cdot 852 \cdot 1{,}2 = 89255{,}52$ грн.

II. $С_{ВІД} = 87{,}3 \cdot 868.88 \cdot 1{,}2 = 91023{,}87$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 89255{,}52 \cdot 0.22 = 19636{,}214$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 91023{,}87 \cdot 0.22 = 20025{,}25$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 25000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 25000 \cdot 0{,}2 = 60000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 60000 \cdot (1 + 0.2) = 72000 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 72000 \cdot 0,22 = 15840 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 19000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.4 \cdot 0.25 \cdot 19000 = 6650 \text{ грн,}$$

де $К_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$ – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.4 \cdot 19000 \cdot 0.08 = 2128 \text{ грн,}$$

де $К_Р$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.40 = 745,6 \text{ години,}$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 745,6 \cdot 0,4 \cdot 0,2 \cdot 9,43 = 562,48 \text{ грн},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 19000 \cdot 0,67 = 12730 \text{ грн}.$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н,$$

$$С_{ЕКС} = 72000 + 15840 + 6650 + 2128 + 563,48 + 12730 = 109911,48 \text{ грн}.$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 109911,48 / 745,6 = 147,414 \text{ грн/год}.$$

Оскільки в даному випадку всі роботи, які пов’язані з використання програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot T,$$

I. $C_М = 147,414 \cdot 852 = 125596,728$ грн.

II. $C_М = 147,414 \cdot 868,88 = 128085,076$ грн.

Накладні витрати складають 67% від заробітної плати:

$$C_Н = C_{ЗП} \cdot 0,67,$$

I. $C_Н = 89255,52 \cdot 0,67 = 59801,198$ грн.

II. $C_Н = 91023,87 \cdot 0,67 = 60985,991$ грн.

Отже, вартість використання ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_М + C_Н,$$

I. $C_{ПП} = 89255,52 + 19636,214 + 125596,728 + 59801,198$
$= 294289,66$ грн.

II. $C_{ПП} = 91023,8 + 20025,25 + 128085,076 + 60985,991 = 300120,117$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР_j} = {К_{К_j}}/{C_{Ф_j}},$$

$К_{ТЕР1} = 6,36 / 294289,66 = 2,1611 \cdot 10^{-5}$,

$К_{ТЕР2} = 4,65 / 300120,117 = 1,5494 \cdot 10^{-5}$.

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 2,1611 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 2,1611 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- вибір мови програмування – Python;
- вибір способу реалізацій алгоритмів – застосування готових бібліотек;
- вибір середовища програмування – Google Colab.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку реалізацію програми та доступний функціонал для роботи.

Висновки до розділу 4

В цьому розділі було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

У результаті виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

У цій роботі в першому розділі сформовано теоретичну основу дослідження. Розглянуто основні поняття, структури та функції файлових систем, визначено принципи організації файлів, каталогів і простору зберігання. Проаналізовано методи фізичного розміщення даних, механізми управління вільним простором і сучасні функціональні можливості файлових систем.

У другому розділі здійснено детальний комплексний аналіз будови сучасних файлових систем: exFAT, NTFS, APFS, ext4, Btrfs і F2FS. Для кожної з них описано архітектуру, структуру даних, методи адресації та методи розміщення файлів. Виконано порівняння файлових систем за основними характеристиками, функціональністю та сумісністю, а також виділено ключові тенденції розвитку та особливості архітектурних рішень сучасних файлових систем.

У третьому розділі проаналізовано основні напрямки застосування штучного інтелекту у файлових системах. Розроблено програмне забезпечення, яке демонструє інтеграцію великих мовних моделей для автоматичного аналізу структури директорій, генерації описів і формування аналітичних звітів щодо вмісту файлової системи.

Можливі подальші удосконалення роботи передбачають розширення дослідження на інші файлові системи, оцінку їхньої поведінки під специфічними навантаженнями, аналіз перспектив розвитку файлових систем з урахуванням нових типів носіїв інформації та вимог до безпеки, а також поглиблення використання штучного інтелекту для управління та оптимізації процесів у файлових системах.

ПЕРЕЛІК ПОСИЛАНЬ

1. Silberschatz A., Galvin P. B., Gagne G. Operating system concepts. – 10th ed. – Hoboken: Wiley, 2018. – 1180 p.
2. File Systems in Operating System. GeeksforGeeks. – Режим доступу: https://www.geeksforgeeks.org/file-systems-in-operating-system/
3. Tanenbaum A. S., Bos H. Modern Operating Systems. – 4th ed. – Boston : Pearson Education, 2015. – 1136 p.
4. Stallings W. Operating Systems: Internals and Design Principles. – 8th ed. – Boston : Pearson Education, 2015. – 1136 p.
5. Russinovich M., Solomon D., Ionescu A. Windows Internals: Part 2. 6th ed. Redmond: Microsoft Press, 2012. 752 p.
6. GUID Partition Table (GPT) Disk Layout. UEFI Specification Version 2.10. – UEFI Forum, 2022. – Режим доступу: https://uefi.org/specs/UEFI/2.10/05_GUID_Partition_Table_Format.html
7. Коваленко А. Є. Операційні системи: навч. посіб. Київ: НТУУ «КПІ», 2010. 248 с.
8. Carrier B. File System Forensic Analysis. – Upper Saddle River: Addison-Wesley, 2005. – 512 p.
9. Toolan F. File System Forensics. – Hoboken: John Wiley & Sons, 2025. – 472 p.
10. Comparison of file systems. Wikipedia – the free encyclopedia. – Режим доступу: https://en.wikipedia.org/wiki/Comparison_of_file_systems
11. Maes B. Comparison of contemporary file systems, 2012. 60 p.
12. Heeger J., Yannikos Y., Steinebach M. An introduction to the exFAT file system and how to hide data within. Journal of Cyber Security and Mobility. – 2022. – Vol. 11, №2. – P. 239–264. – DOI: 10.13052/jcsm2245-1439.1125.
13. ExFAT overview. LSoft Technologies Inc. – Режим доступу: https://www.ntfs.com/exfat-overview.htm

14. exFAT. Wikipedia – the free encyclopedia. – Режим доступу: https://uk.wikipedia.org/wiki/ExFAT

15. exFAT vs. FAT32 Comparison. LSoft Technologies Inc. – Режим доступу: https://www.ntfs.com/exfat-comparison.htm

16. exFAT Directory Structure. LSoft Technologies Inc. – Режим доступу: https://www.ntfs.com/exfat-directory-structure.htm

17. exFAT: File Name Directory Entry. LSoft Technologies Inc. – Режим доступу: https://www.ntfs.com/exfat-filename-dentry.htm

18. NTFS. Wikipedia – the free encyclopedia. – Режим доступу: https://en.wikipedia.org/wiki/NTFS

19. NTFS vs FAT vs exFAT. LSoft Technologies Inc. – Режим доступу: https://www.ntfs.com/ntfs_vs_fat.htm

20. NTFS System Files. LSoft Technologies Inc. – Режим доступу: https://www.ntfs.com/ntfs-system-files.htm

21. NTFS Partition Boot Sector. LSoft Technologies Inc. – Режим доступу: https://www.ntfs.com/ntfs-partition-boot-sector.htm

22. NTFS File Types. LSoft Technologies Inc. – Режим доступу: https://www.ntfs.com/ntfs-files-types.htm

23. NTFS Master File Table (MFT). LSoft Technologies Inc. – Режим доступу: https://www.ntfs.com/ntfs-mft.htm

24. Apple File System. Wikipedia – the free encyclopedia. – Режим доступу: https://en.wikipedia.org/wiki/Apple_File_System

25. Apple Platform Security. Apple Inc. – Режим доступу: https://support.apple.com/uk-ua/guide/security/seca6147599e/web

26. Comparing ApFS and HFS. LSoft Technologies Inc. – Режим доступу: https://ntfs.com/apfs-compare.htm

27. What's New in APFS. LSoft Technologies Inc. – Режим доступу: https://www.ntfs.com/apfs-new.htm

28. ApFS Structure. LSoft Technologies Inc. – Режим доступу: https://www.ntfs.com/apfs-structure.htm

29. Bhat W. A., Wani M. A. Forensic analysis of B-tree file system (Btrfs). Digital Investigation. – 2018. – Vol. 26. – P. S1–S14. – DOI: https://doi.org/10.1016/j.diin.2018.09.001

30. Introduction. Btrfs Documentation – Режим доступу: https://btrfs.readthedocs.io/en/latest/Introduction.html

31. Btrfs Coming to Fedora 33. Chris Murphy, Langdon White. – Режим доступу: https://fedoramagazine.org/btrfs-coming-to-fedora-33/.

32. F2FS. Wikipedia – the free encyclopedia. – Режим доступу: https://en.wikipedia.org/wiki/F2FS

33. Flash-Friendly File System (F2FS). The Linux Kernel Documentation. – Режим доступу: https://docs.kernel.org/filesystems/f2fs.html.

34. Lee C., Sim D., Hwang J.-Y., Cho S. F2FS: A New File System for Flash Storage February 16–19, 2015, Santa Clara, CA, USA. – USENIX Association, 2015. – P. 273–286. – URL: https://www.usenix.org/conference/fast15/technical-sessions/presentation/lee

35. Btrfs. Wikipedia – the free encyclopedia. – Режим доступу: https://en.wikipedia.org/wiki/Btrfs

36. How to check maximum path length in Linux. Medium. – Режим доступу: https://medium.com/@linuxadminhacks/how-to-check-maximun-path-length-in-linux-b4296c99313e

37. Filesystem functionality comparison. Microsoft Learn. – Режим доступу: https://learn.microsoft.com/en-us/windows/win32/fileio/filesystem-functionality-comparison

38. APFS Directories and Names. Eclectic Light. – Режим доступу: https://eclecticlight.co/2024/03/25/apfs-directories-and-names/

39. Ext4. Wikipedia – the free encyclopedia. – Режим доступу: https://en.wikipedia.org/wiki/Ext4

40. Products A to Z. Paragon Software. – Режим доступу: https://www.paragon-software.com/about/products-a-to-z/

41. NTFS3 Driver FAQ. Paragon Software. – Режим доступу: https://www.paragon-software.com/home/ntfs3-driver-faq/

42. Needhi, J., Prasath, R., Vikram, D., & Vishnu, G. (2024). Performance Optimization of Voice-Assisted File Management Systems – Archives ouvertes. – Режим доступу: https://hal.science/hal-04611992v1

43. Braina – Intelligent Personal Assistant for Windows. – Режим доступу: https://www.brainasoft.com/braina/

44. Advanced technologies used by Microsoft Defender Antivirus. – Microsoft Learn. – Режим доступу: https://learn.microsoft.com/en-us/defender-endpoint/adv-tech-of-mdav

45. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A. et al. (2020). Language Models are Few-Shot Learners. – Режим доступу: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

46. Van Rossum, G. Computer Programming for Everybody. – Режим доступу: https://www.python.org/doc/essays/blurb/

47. Meta AI. Llama 3.1 70B Instruct. Hugging Face. – Режим доступу: https://huggingface.co/meta-llama/Llama-3.1-70B-Instruct

48. Salesforce Research. BLIP Image Captioning Large. Hugging Face. – Режим доступу: https://huggingface.co/Salesforce/blip-image-captioning-large

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
!pip install python-docx pymupdf

import os
import requests
import datetime
from docx import Document
from PIL import Image
import fitz
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from transformers import pipeline

pipeline_blip = pipeline("image-to-text", model="Salesforce/blip-image-captioning-large")

import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

def extract_text(path):
    with open(path, encoding="utf-8", errors="ignore") as f:
        return f.read()
def extract_pdf_text(path):
    text = ""
    with fitz.open(path) as doc:
        for page in doc:
            text += page.get_text()
    return text
def extract_docx_text(path):
    doc = Document(path)
    return "\n".join([para.text for para in doc.paragraphs])

def extract_from_file(path):
    ext = os.path.splitext(path)[-1].lower()
    text = ""
    if ext in (".txt", ".md", ".csv", ".json"):
        text = extract_text(path)
    elif ext == ".pdf":
        text = extract_pdf_text(path)
```

```
    elif ext == ".docx":
        text = extract_docx_text(path)
    if not text.strip():
        return ""
    return text

def save_to_file(llm_report, prompt_text, folder_name, folder_path):
    with open("analysis_result.txt", "w", encoding="utf-8") as f:
        f.write(f"Directory name: {folder_name}\n")
        f.write(f"Directory path: {folder_path}\n\n")
        f.write(prompt_text)
        f.write("\n\nLLM analysis report:\n\n")
        f.write(llm_report)

def get_keywords(path, top_n=10,  ngram_range=(1, 2)):
    text = extract_from_file(path)
    vectorizer = TfidfVectorizer(stop_words='english', ngram_range=ngram_range)
    tfidf_matrix = vectorizer.fit_transform([text])
    feature_names = vectorizer.get_feature_names_out()
    tfidf_scores = tfidf_matrix.toarray()[0]
    word2score = {feature_names[i]: tfidf_scores[i] for i in
range(len(feature_names))}
    sorted_keywords = sorted(word2score.items(), key=lambda x: x[1],
reverse=True)
    keywords_no_digits = [word for word, score in sorted_keywords if not
re.search(r'\d', word)]
    top_keywords = keywords_no_digits[:top_n]
    return ", ".join([kw for kw in top_keywords])

def get_image_description(path, remove_stop_word = True):
      image = Image.open(path)
      caption = pipeline_blip(image)
      generated_text = caption[0]['generated_text']
      if not remove_stop_word:
          return generated_text
      words = generated_text.split()
      filtered_words = [word for word in words if word.lower() not in stop_words]
      text = ' '.join(filtered_words)
      return text

def get_file_paths(root_path):
    all_file_paths = []
    num_files = 0
    num_dirs = 0
```

```
    for root, dirs, files in os.walk(root_path):
        dirs[:] = [d for d in dirs if d != '.ipynb_checkpoints']

        for name in dirs:
            full_path = os.path.join(root, name)
            num_dirs += 1
            all_file_paths.append(full_path)

        for name in files:
            if name == '.ipynb_checkpoints':
                continue
            full_path = os.path.join(root, name)
            num_files += 1
            all_file_paths.append(full_path)

    return all_file_paths, num_files, num_dirs

def get_full_fs_structure(file_paths, folder_path):

    IMAGE_EXTS = {'.jpg', '.jpeg', '.png', '.webp'}
    TEXT_EXTS = {'.txt', '.md', '.csv', '.json', '.docx', '.pdf'}
    fs = []
    for full_path in file_paths:
        ext = os.path.splitext(full_path)[-1].lower()
        try:
            stat = os.stat(full_path)

            if os.path.isdir(full_path):
                tag = None
            elif ext in IMAGE_EXTS:
                tag = ['img_file', get_image_description(full_path)]
            elif ext in TEXT_EXTS:
                tag = ['text_file', get_keywords(full_path)]
            else:
                tag = ['other_file', '']

            info = [
                os.path.relpath(full_path, folder_path),
                os.path.isdir(full_path),
                stat.st_size,
                tag
            ]
        except Exception as e:
            info = [
                os.path.relpath(full_path, folder_path),
```

```
                os.path.isdir(full_path),
                str(e)
            ]
        fs.append(info)
    return fs

def data(structure):
    lines = []
    for entry in structure:
        rel_path, is_dir, size_bytes, meta = entry
        size_kb = round(size_bytes / 1024, 1)
        entry_type = 'folder' if is_dir else 'file'
        key = ""
        if not is_dir:
            filetype, value = meta
            if filetype == 'text_file' and value:
                key = f'keywords: {value}'
            elif filetype == 'img_file' and value:
                key = f'dsc: {value}'
        elems = [rel_path, entry_type, str(size_kb)]
        if key:
            elems.append(key)
        line = ", ".join(elems)
        lines.append(line)
    return "\n".join(lines)

def generated_prompt(info):
    promt_text = (
        "You are a research assistant specializing in scientific analysis of file system directories. "
        "Your task is to produce an analytical summary of the contents of the provided directory, based on the following requirements:\n\n"
        "1. Summarize the main topic and subtopics, approaches, and types of content in the files, using extracted keywords and image descriptions, but avoid repeating themes or phrases.\n"
        "2. Identify and briefly describe any anomalies, duplicates, empty folders, or unusual files.\n"
        "3. Include a short statistical overview of the distribution of file types and the total volume of files (number and size).\n"
        "4. Draw brief, clear scientific conclusions about the contents and organization of the directory.\n"
        "5. Do not use bullet points, tables, or bold formatting. Structure your response into coherent paragraphs. Avoid any introductions or conclusions unrelated to the provided data.\n\n"
        f"Directory information:\n"
```

```
        f"Main directory path: {info['folder_path']}\n"
        f"Number of files: {info['total_num_files']}\n"
        f"Number of subfolders: {info['total_num_folders']}\n\n"
        "Directory structure (each line: relative_path, type (file/folder), size in KB, content summary (keywords or image description)):\n\n"
        f"{info['data']}\n\n"
        "Now generate your summary analysis as described above:"
    )
    return promt_text

folder_path = "/content/drive/MyDrive/Test_files"
folder_name = os.path.basename(os.path.normpath(folder_path))

file_paths, num_files, num_folders = get_file_paths(folder_path)
structure = get_full_fs_structure(file_paths, folder_path)
data_fs = data(structure)

info = {'data' : data_fs, 'folder_path' : folder_path, 'total_num_files' : num_files, 'total_num_folders' : num_folders}
prompt_text = generated_promt(info)

API_URL = "https://router.huggingface.co/fireworks-ai/inference/v1/chat/completions"
headers = {
    "Authorization": f"Bearer hf_fzaCkmJcCHJOzixVyWWwDBRcyxnKl*****",
}
def query(payload):
    response = requests.post(API_URL, headers=headers, json=payload)
    return response.json()

messages = {
    "messages": [
        {
            "role": "user",
            "content": prompt_text
        }
    ],
    "model": "accounts/fireworks/models/llama-v3p1-70b-instruct",
    "temperature": 0.3,
    "max_tokens": 400
}
response = query(messages)
llm_result = response["choices"][0]["message"]['content']
print(llm_result)
save_to_file(llm_result, prompt_text, folder_name, folder_path)
```