УДК 004.451.5
DOI: 10.18523/2617-3808.2025.8.205-212

Левченко А. С., Франків О. О., Петелєв Є. Р.

ДОСЛІДЖЕННЯ ТА ОПТИМІЗАЦІЯ МЕТОДІВ ОЦІНЮВАННЯ РОЗМІРУ ФАЙЛОВОЇ ІЄРАРХІЇ В APFS (APPLE FILE SYSTEM)

Цю статтю присвячено дослідженню та оптимізації процесів сканування файлової системи APFS (Apple File System). Розглянуто ключові інструменти доступу до APFS та алгоритмічні стратегії, зокрема верхньорівневий обхід, повний обхід, фільтрацію за стоп-словами та інтерактивний підхід. Реалізовано методи оброблення файлових ієрархій, які передбачають послідовне та паралельне оброблення з використанням Grand Central Dispatch (GCD) і Swift Concurrency. Розроблено застосунок для сканування APFS, який демонструє практичне застосування запропонованих підходів. Проведено тестування й порівняльний аналіз методів сканування APFS.

Ключові слова: Apple File System, APFS, сканування файлової системи, macOS.

Вступ

Файлові системи є невід'ємною складовою будь-якої сучасної операційної системи. Вони відіграють важливу роль у швидкому доступі до даних, надійності їх збереження та ефективному використанні простору. Apple File System (APFS) було представлено 14 червня 2016 р. як файлову систему наступного покоління для пристроїв Apple [12]. Вона прийшла на зміну Hierarchical File System Plus (HFS+), яку використовували з 1998 р. [3]. Станом на 2024 рік macOS є другою за популярністю десктопною операційною системою, а загальна частка операційних систем Apple на ринку становить близько 25 % [8].

Зростання популярності пристроїв Apple зумовлює потребу в глибшому розумінні внутрішніх механізмів роботи файлової системи цих пристроїв, а також оптимізації процесу її сканування, який лежить в основі таких задач, як резервне копіювання, індексація, антивірусне сканування, відновлення даних і цифрова криміналістика.

Apple File System принесла значні зміни, спрямовані на покращення доступу до даних, забезпечення їхньої цілісності, ефективності управління сховищем та підвищення безпеки. На відміну від HFS+, де кожен том має фіксований розмір і не ділиться вільним простором з іншими розділами, APFS запровадила спільне використання простору [11]. Це дозволяє кільком томам у контейнері динамічно розподіляти сховище, гнучко змінюючи розмір без необхідності переформатування (рис. 1).

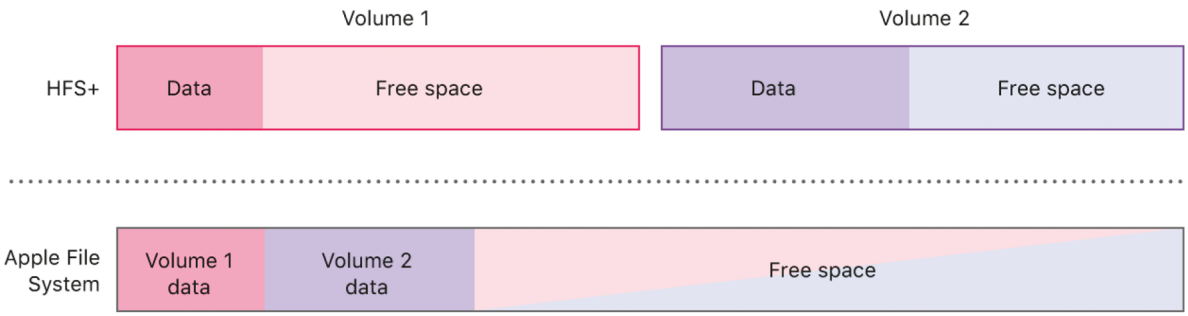

Рис. 1. Розподіл простору між томами в HFS+ та APFS

APFS використовує механізм copy-on-write, який гарантує, що дані не перезаписуються, поки зміни не буде повністю завершено. Це допомагає мінімізувати ризик пошкодження даних і підвищує

загальну стабільність файлової системи [1]. Для керування файлами та каталогами APFS використовує високооптимізовану структуру B-дерева, яка прискорює процеси копіювання, видалення та перейменування файлів [2].

Однією з ключових особливостей файлової системи APFS є можливість створення знімків (snapshots), які фіксують стан файлової системи у певний момент часу. Вони допомагають швидко відновити дані без використання додаткового місця для дублікатів даних [6].

Усі ці особливості Apple File System роблять її високопродуктивною та надійною файловою системою, але водночас ускладнюють процес її сканування. Саме тому оптимізація сканування APFS є важливою темою сучасних досліджень, оскільки швидкість і точність цього процесу безпосередньо впливають на загальну ефективність роботи системи. У цій статті ми зосередимося на аналізі методів і алгоритмів, які дають змогу максимально ефективно сканувати APFS, та визначимо оптимальне рішення.

Інструменти для роботи з APFS

У межах цього дослідження ми зосереджуємо увагу на визначенні розміру файлової ієрархії. У файловій системі APFS кожен елемент характеризується двома типами розмірів: логічним і фізичним. Логічний розмір (actual size) відображає обсяг даних, фактично записаних у файл або директорію, тоді як фізичний розмір (allocated size або size on disk) визначає кількість дискового простору, зарезервованого файловою системою для зберігання цього елемента. У нашому дослідженні для сканування APFS ми використовуватимемо саме фізичний розмір, оскільки він дає змогу точніше оцінити структуру та використання простору.

Для роботи з APFS існує декілька інструментів: NSFileManager, URLResourceKey і термінальна команда du. NSFileManager — це базовий API для роботи з Apple File System. Він забезпечує зручний доступ до файлів і каталогів. Метод attributesOfItem(atPath:) дозволяє отримати розмір файлу через атрибут '.size'. Однак для обчислення розміру директорії потрібно застосовувати рекурсивний обхід. Це робить процес сканування повільним і ресурсоємним, особливо для великих файлових структур. Крім того, за допомогою NSFileManager неможливо отримати дані про фізичний розмір файлів на диску.

URLResourceKey є більш сучасним підходом, який використовує об'єкти URL для безпечної роботи з файловою системою. Ключі fileSizeKey та fileAllocatedSizeKey дозволяють отримати логічний і фізичний розмір файлів відповідно. Для обчислення розміру директорій також потрібно застосовувати рекурсивний підхід, проте URLResourceKey оптимізує цей процес, опрацьовуючи лише необхідні атрибути. Це підвищує продуктивність сканування порівняно з NSFileManager. URLResourceKey зменшує кількість помилок, пов'язаних із неправильним форматом шляхів до елементів файлової системи, і є більш універсальним інструментом.

Термінальна команда du ("disk usage") ефективно оцінює використання дискового простору файлами та директоріями. Вона відображає інформацію у блоках, кожен з яких — це 512 байтів [9]. Команда du не потребує використання рекурсивного підходу для сканування файлової ієрархії. Також цей інструмент підтримує такі параметри, як -a (all — відображає розмір для кожного файла у ієрархії), -s (summarize — відображає загальний розмір файлу або директорії без деталізації по піддиректоріях), -d (depth — відображає розмір для всіх елементів лише на вказану глибину файлової структури) [10]. Для інтеграції цього інструменту в Swift використовується клас Process. Проте оброблення даних можливе лише після завершення виконання команди, і це може сповільнити процес сканування. Також під час використання цього інструменту виникають проблеми з доступом до певних директорій, що вимагає додаткового оброблення помилок.

Порівняльний аналіз інструментів для роботи з Apple File System демонструє, що NSFileManager зручний API для роботи з окремими файлами, але досить повільний для директорій. Команда du швидка у скануванні файлової ієрархії, але складна в інтеграції та обробленні. URLResourceKey — найефективніший інструмент для роботи з APFS, який поєднує швидкість, безпеку та гнучкість.

Алгоритми обходу файлової системи APFS

Для ефективного сканування Apple File System варто застосувати різні алгоритмічні стратегії для обходу файлової ієрархії. Вони допоможуть врахувати складність структури і мінімізувати зайві опе-

рації. Основні алгоритми, які застосовувались, передбачають верхньорівневий підхід, повний обхід, інтерактивний обхід і фільтрацію за стоп-словами.

Верхньорівневий підхід сканує лише елементи верхнього рівня. Це забезпечує високу швидкість завдяки мінімальним зверненням до файлової системи. Реалізація за допомогою команди du з параметром -d 1 забезпечує мінімальні витрати ресурсів. Однак NSFileManager та URLResourceKey потребують використання рекурсивного обходу, який знижує ефективність, особливо для NSFileManager. URLResourceKey залишається продуктивним завдяки оптимізованим запитам до APFS.

Повний обхід файлової системи рекурсивно сканує файлову ієрархію і надає детальну інформацію про розміри файлів та директорій. Команда du з параметром -a надає повний список елементів файлового дерева, а NSFileManager та URLResourceKey використовують ключі .size та fileAllocatedSizeKey відповідно. Цей метод більш точний, але час його виконання значний. Тому це може призвести до певних незручностей для користувача.

Інтерактивний обхід файлової системи дозволяє отримувати результати поступово та покращує взаємодію з користувачем. Цей метод реалізовано за допомогою AsyncStream [4]. Файли та директорії додаються в асинхронний потік, використовуючи метод yield(_:), а після завершення викликається метод finish(). Інтерактивний підхід дозволяє розпочати оброблення даних ще до завершення процесу сканування, а також робить сканування більш гнучким і зручним для користувача.

Фільтрація за стоп-словами оптимізує сканування APFS шляхом вилучення непотрібних елементів. Для її реалізації використовуються алгоритми DFS і BFS. Алгоритм DFS [5] виконує рекурсивний обхід директорій та пропускає елементи, що містять стоп-слова, та має вищу швидкість для глибоких структур. Алгоритм BFS [7] дозволяє швидше проаналізувати елементи верхніх рівнів файлової ієрархії, оскільки елементи файлової системи обробляються по рівнях за допомогою черги. На практиці DFS переважає BFS за швидкістю сканування APFS, але BFS усе ж є досить ефективним алгоритмом.

Отже, вибір алгоритму для сканування Apple File System залежить від задачі: верхньорівневий підхід — для швидкого аналізу, повний обхід — для детального аналізу, інтерактивний обхід — для зручності користувача, а фільтрація за стоп-словами з використанням DFS або BFS — для оптимізації ресурсів.

Багатопотокове оброблення файлової ієрархії APFS

Послідовне оброблення не завжди є ефективним, особливо якщо структура даних є досить складною та багаторівневою. Тому для підвищення швидкості сканування файлової системи APFS застосовується багатопотокове оброблення, реалізоване за допомогою Swift Concurrency та Grand Central Dispatch (GCD).

Послідовне оброблення виконує операції в одному потоці. Воно просте і зручне для невеликих директорій або окремих файлів. Також, оскільки немає потреби в управлінні потоками та синхронізації, послідовне оброблення легко відлагоджувати. Однак для складних файлових ієрархій цей метод є недостатньо швидким, що стало основною причиною для застосування методів багатопотокового оброблення.

Swift Concurrency впроваджений починаючи з версії Swift 5.5. Він використовує async/await, Task Groups та Actors для структурованого паралелізму. Task у Swift Concurrency — це асинхронні блоки коду, які виконуються без блокування основного потоку. Task Groups дозволяють створювати й керувати кількома завданнями, які виконуються паралельно. Функція withThrowingTaskGroup дозволяє створювати та керувати групами асинхронних завдань, які виконуються паралельно, одночасно обробляючи потенційні помилки. Actors запобігають проблемам типу data race, автоматично керуючи доступом до ресурсів, що усуває потребу в ручних блокуваннях, таких як NSLock, і знижує ризик взаємного блокування (deadlock).

Grand Central Dispatch (GCD) — це низькорівневий API для управління паралельними операціями. GCD керує виконанням завдань за допомогою DispatchQueue, які можуть бути послідовними та паралельними, що дозволяє ефективно розподіляти системні ресурси відповідно до поточного навантаження. Для координації виконання декількох задач використовується DispatchGroup, що дає можливість запускати асинхронні операції паралельно, але синхронізувати момент завершення групи. Функція withCheckedThrowingContinuation забезпечує інтеграцію GCD із сучасним підходом async/await. Хоча GCD є старішим механізмом, він і досі залишається ефективним.

Отже, для багатопотокового оброблення APFS Swift Concurrency є більш ефективним, ніж GCD, завдяки швидкості, безпеці та масштабованості. Для простих задач допускається використання послідовного оброблення.

Архітектура розробленого рішення

Розроблене рішення для сканування файлової системи APFS базується на модульній архітектурі, яка забезпечує гнучкість і масштабованість. Архітектура, зображена на рис. 2, включає ключові компоненти: протокол FS API для взаємодії з файловою системою через різні API (NSFileManager, URLResourceKey або термінальна команда du), модуль Concurrency для послідовного або багатопотокового оброблення елементів за допомогою Swift Concurrency або GCD. Також є модуль Techniques для стратегій обходу файлової ієрархії: верхньорівневої, повної, інтерактивної та фільтрації за стоп-словами з використанням алгоритмів DFS та BFS. Модульна структура дозволяє легко адаптувати рішення та змінювати алгоритми чи API залежно від вимог для забезпечення ефективного сканування Apple File System.

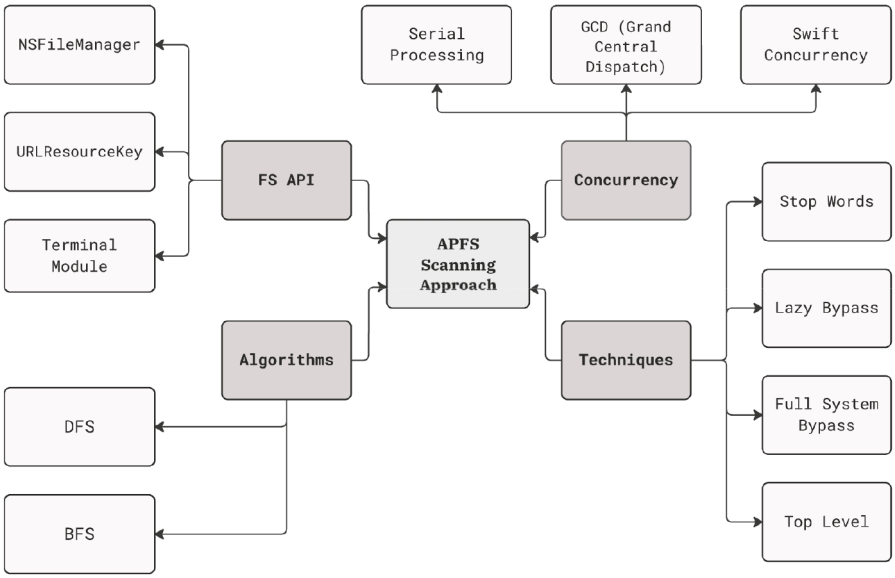

Рис. 2. Структура розробленого рішення

Оцінка ефективності

Для оцінки ефективності методів сканування APFS було проведено тестування з використанням спеціальних наборів даних, які моделюють різні файлові ієрархії. Це дало можливість проаналізувати продуктивність у типових і крайніх випадках. Зокрема було застосовано чотири шаблони:

- Breadth-First Structure (BFS) — шаблон для широких структур, який горизонтально розширюється перед поглибленням;
- Depth-First Structure (DFS) — шаблон, який моделює глибокі файлові структури;
- Balanced Tree Structure — шаблон для рівномірних і впорядкованих структур;
- Unbalanced Tree Structure — шаблон, який моделює непередбачувані та нерегулярні сценарії, які наближені до реальних користувацьких систем.

Тестування проводили на комп'ютері MacBook Air із чипом Apple M2 (8-ядерний процесор: 4 ядра продуктивності і 4 ядра ефективності), 16 GB оперативної пам'яті та SSD-накопичувачем, що працює з файловою системою APFS. Операційна система — macOS Sequoia 15.4.1. Кожен алгоритм випробовували на чотирьох шаблонах даних із різними розмірами файлів. Алгоритми оцінювали за часом виконання та піковим використанням оперативної пам'яті. Результати тестування проілюстровано графіками, які порівнюють ефективність різних методів сканування APFS.

Спочатку було проведено порівняння інструментів для сканування APFS за часом виконання на чотирьох типах файлових ієрархій: BFS, DFS, Balanced і Unbalanced (рис. 3). Графік відображає значні відмінності в продуктивності залежно від структури ієрархії. Найскладнішою виявилась BFS

структура через її широку організацію з великою кількістю дочірніх елементів, що ускладнює та подовжує сканування. Натомість найпростішою була Balanced Tree Structure завдяки своїй рівномірній і передбачуваній структурі. Найефективнішим інструментом став URLResourceKey, який значно перевершив NSFileManager і du для всіх типів ієрархій. Другою за продуктивністю є команда du, що показала стабільні результати, поступившись NSFileManager лише під час сканування Balanced структури.

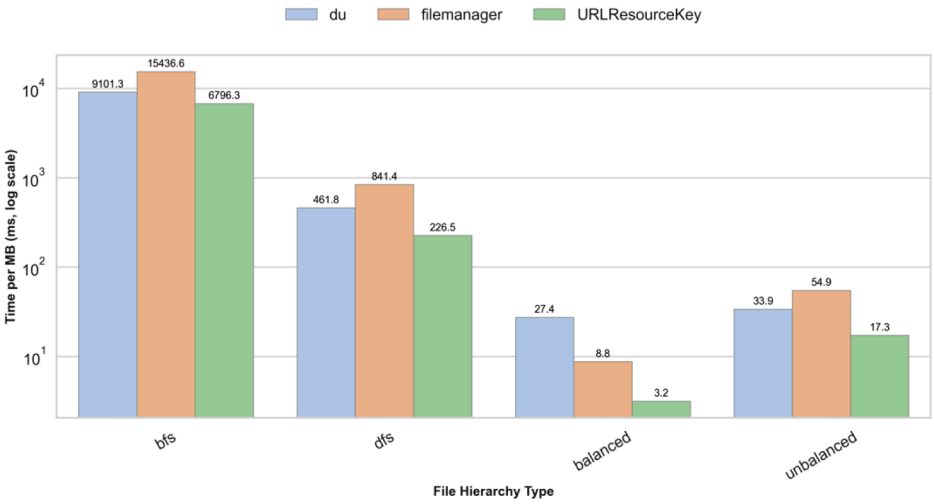

Рис. 3. Порівняння часу виконання сканування за допомогою різних інструментів для чотирьох типів файлових ієрархій

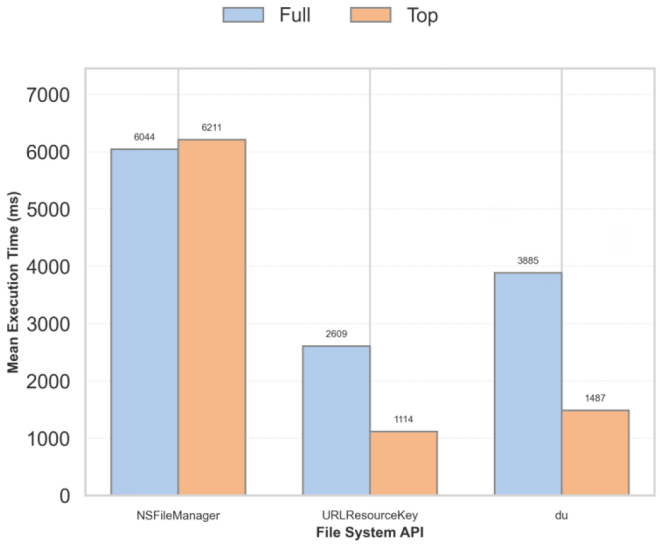

Рис. 4. Порівняння часу виконання сканування за допомогою повного та верхньорівневого підходу та різних інструментів

На рис. 4 подано діаграму, що порівнює середній час сканування за допомогою NSFileManager, URLResourceKey і команди du для верхньорівневого та повного обходу файлової ієрархії. Верхньорівневий підхід скорочує час сканування в 2–3 рази для URLResourceKey і du. Проте для NSFileManager продуктивність не змінюється через необхідність застосування рекурсивного обходу в обох методах.

Також було досліджено інтерактивний обхід і фільтрацію за стоп-словами, які показали помітне покращення ефективності сканування файлової системи. Інтерактивний підхід забезпечує більш швидкий доступ до перших результатів без втрати точності, що значно покращує користувацький досвід при роботі з великими ієрархіями. Фільтрація за стоп-словами, своєю чергою, зменшує час сканування та навантаження на системні ресурси, обробляючи лише необхідні елементи файлової ієрархії.

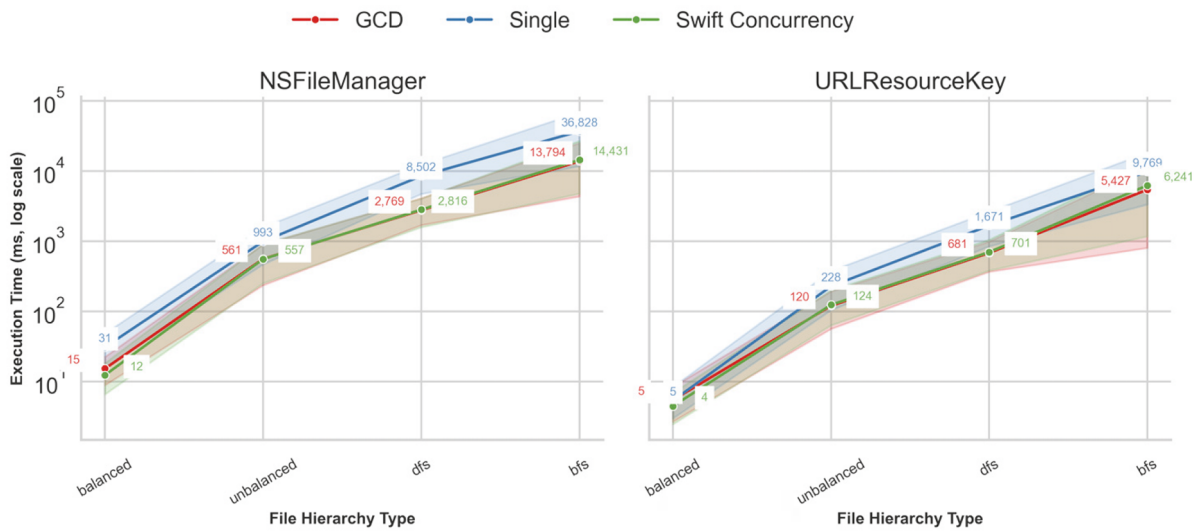

Рис. 5. Порівняння середнього часу виконання для послідовного оброблення, GCD і Swift Concurrency

Рисунок 5 демонструє графік порівняння багатопотокових підходів для чотирьох типів файлових структур, розділених на дві частини: ліва відображає результати для NSFileManager, а права — для URLResourceKey. Багатопотокові методи GCD та Swift Concurrency значно перевершують послідовне оброблення і прискорюють сканування APFS у 2–3 рази. Натомість GCD і Swift Concurrency демонструють майже однаковий рівень продуктивності.

Таблиця 1. **Середнє використання оперативної пам'яті різними багатопотоковими підходами та API**

File System API	GCD (MB)	Single (MB)	Swift concurrency (MB)
NSFileManager	222.0	226.5	241.1
URLResourceKey	96.4	70.2	113.9
du	59.9	43.5	65.7

Ще одним важливим параметром для оцінювання ефективності розробленого рішення було використання оперативної пам'яті під час сканування APFS. Таблиця 1 демонструє споживання пам'яті інструментами NSFileManager, URLResourceKey та командою du, а також різними підходами до багатопотокового оброблення. NSFileManager виявився найбільш ресурсоємним, тоді як команда du показала найменше споживання пам'яті. Серед багатопотокових методів оброблення Swift Concurrency споживає найбільше пам'яті, Grand Central Dispatch — дещо менше, а послідовне оброблення створює мінімальне навантаження. Хоча послідовне оброблення є найменш вимогливим до ресурсів, багатопотокові методи значно прискорюють сканування і компенсують додаткове використання пам'яті суттєвим підвищенням продуктивності.

Висновки

Для дослідження ефективності методів оптимізації сканування Apple File System було створено застосунок із модульною архітектурою, яка забезпечує гнучкість, масштабованість і можливість комбінувати різні підходи. Було проведено тестування на чотирьох типах штучних файлових ієрархій (BFS, DFS, Balanced, Unbalanced), що охоплюють як типові, так і крайні випадки файлових структур. Результати показали високу продуктивність і точність запропонованих методів.

Застосування паралельного оброблення за допомогою GCD і Swift Concurrency скоротило час сканування у 2–3 рази порівняно з послідовним обробленням файлових ієрархій. Особливо ефективною виявилась комбінація Swift Concurrency та URLResourceKey, яка показала найкращі результати для типових користувацьких структур. Реалізовані стратегії обходу файлової системи, зокрема верхньорівневий, повний, інтерактивний обхід і фільтрація за стоп-словами, оптимізують сканування APFS, забезпечуючи гнучке налаштування процесу під конкретні завдання.

Дослідження показало потенціал для подальшого вдосконалення, а запропоновані методи можуть бути застосовані не лише для оптимізації сканування APFS, а й для поглиблення розуміння управління даними в сучасних файлових системах macOS.

Список літератури

1. Apple Inc. Apple File System Guide [Electronic resource] / Apple Inc. — 2018. — Mode of access: https://developer.apple.com/library/archive/documentation/FileManagement/Conceptual/APFS_Guide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40016999-CH1-DontLinkElementID_15.
2. Apple Inc. Apple File System Reference [Electronic resource] / Apple Inc. — 2020. — Mode of access: https://developer.apple.com/support/downloads/Apple-File-System-Reference.pdf.
3. Apple Inc. HFS Plus Volume Format / Apple Inc. — Technical Note TN1150, 2010.
4. Apple Inc. Streams, Sockets, and Ports [Electronic resource] / Apple Inc. — Mode of access: https://developer.apple.com/documentation/foundation/streams_sockets_and_ports.
5. Garg D. Analysis of the Depth First Search Algorithms / D. Garg, N. Kaur. — Thapar University, 2012.
6. Hansen K. H. Decoding the APFS file system [Electronic resource] / K. H. Hansen, F. Toolan // Digital Investigation. — 2017. — Vol. 22. — Pp. 107–132. — Mode of access: https://doi.org/10.1016/j.diin.2017.07.003.
7. Holdsworth H. The Nature of Breadth-First Search / H. Holdsworth. — School of Computer Science Mathematics and Physics, James Cook University, Australia, 1999.
8. Kosisochukwu H. U. Exploring operating system diversity: A comparative analysis of Windows, Mac OS, Android and IOS / H. U. Kosisochukwu, M. I. Abdullahi // Systematic and Modern Science Research. — 2024. — Vol. 5, no. 9. — Pp. 23–40.
9. Nordvik R. APFS [Electronic resource] / R. Nordvik // Mobile Forensics — The File Format Handbook. — 2022. — Pp. 3–39. — Mode of access: https://doi.org/10.1007/978-3-030-98467-0.
10. Platt D. Tweak Your Mac Terminal [Electronic resource] / D. Platt. — Berkeley (CA) : Apress, 2021. — Mode of access: https://doi.org/10.1007/978-1-4842-6171-2_1.
11. Rane R. Demystifying File Systems: A Comprehensive Exploration of Data Organization [Electronic resource] / R. Rane, A. Singh – 2024. — Mode of access: https://doi.org/10.13140/RG.2.2.31160.35845.
12. Tamura E. Introducing Apple File System / E. Tamura, D. Giampaolo. — 2016.

References

Apple Inc. (2018). *Apple File System Guide.* Apple Developer. https://developer.apple.com/library/archive/documentation/FileManagement/Conceptual/APFS_Guide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40016999-CH1-DontLinkElementID_15.

Apple Inc. (2020). *Apple File System Reference.* Apple Developer. https://developer.apple.com/support/downloads/Apple-File-System-Reference.pdf.

Apple Inc. (2010). *HFS Plus Volume Format — Technical Note TN1150*.

Apple Inc. (n. d.). *Streams, Sockets, and Ports.* Apple Developer Documentation. https://developer.apple.com/documentation/foundation/streams_sockets_and_ports.

Garg, D., & Kaur, N. (2012). *Analysis of the Depth First Search Algorithms.* Thapar University.

Hansen, K. H., & Toolan, F. (2017). Decoding the APFS file system. *Digital Investigation*, *22*, 107–132. https://doi.org/10.1016/j.diin.2017.07.003.

Holdsworth, H. (1999). *The Nature of Breadth-First Search.* School of Computer Science Mathematics and Physics, James Cook University.

Kosisochukwu, H. U., & Abdullahi, M. I. (2024). Exploring operating system diversity: A comparative analysis of Windows, Mac OS, Android and IOS. *Systematic and Modern Science Research (JSMSR)*, *5* (9), 23–40.

Nordvik, R. (2022). APFS. In *Mobile Forensics — The File Format Handbook* (pp. 3–39). Springer. https://doi.org/10.1007/978-3-030-98467-0.

Platt, D. (2021). *Tweak Your Mac Terminal*. Apress. https://doi.org/10.1007/978-1-4842-6171-2_1.

Rane, R., & Singh, A. (2024). *Demystifying File Systems: A Comprehensive Exploration of Data Organization. https://doi.org/10.13140/RG.2.2.31160.35845*.

Tamura, E., & Giampaolo, D. (2016). *Introducing Apple File System*.

A. Levchenko, O. Frankiv, Y. Peteliev

INVESTIGATION AND OPTIMIZATION OF FILE HIERARCHY SIZE ESTIMATION METHODS IN APFS (APPLE FILE SYSTEM)

File systems are an integral part of modern operating systems, providing the foundation for organizing, storing, and accessing data. The efficiency of a file system plays a critical role in determining software performance, particularly when handling large volumes of data. This study focuses on the research and analysis of optimization methods for scanning the Apple File System (APFS), a modern file system developed by Apple to enhance data access, integrity, and storage management. APFS introduces advanced features such as shared space allocation, B-tree structures, and support for snapshots, which, while improving performance, also pose challenges for efficient scanning.

The article explores a range of scanning strategies, including top-level traversal, full system bypass, interactive bypass, and stop-word filtering, as well as serial and parallel processing approaches using Swift Concurrency and Grand Central Dispatch (GCD). Various tools for accessing APFS, such as NSFileManager, URLResourceKey, and du, were utilized to facilitate this analysis. To enable a systematic evaluation of these methods across various file hierarchies, a specialized tool for scanning APFS was developed. The research aims to assess key performance aspects such as speed, scalability, and resource utilization, offering insights into optimizing APFS scanning for improved efficiency.

Testing was conducted on four types of file hierarchies: Breadth-First Structure (BFS), Depth-First Structure (DFS), Balanced Tree Structure, and Unbalanced Tree Structure. The results demonstrated the effectiveness of the proposed methods, highlighting their ability to adapt to different structural complexities while maintaining high performance. This validation underscores the practical utility of the developed tool and the potential for these optimization techniques to enhance APFS scanning in real-world applications.

Keywords: Apple File System, APFS, file system scanning, macOS.

Матеріал надійшов 19.06.2025