**ПЛАН-КОНСПЕКТ ЛЕКЦІЇ**

**з дисципліни «Системне адміністрування ОС Linux»**

**Викладач:** студент групи 641м Бужак Андрій

**Дата проведення:** 24.09.2021

**Група: 541 м**

**Вид заняття:** лекція

**Тривалість пари:** 80 хвилин

**Тема:** Фізична організація і характеристики файлових систем.

**Мета:** Набути теоретичних знань щодо організації та характеристик файлових систем.

**ХІД РОБОТИ:**

**Базові відомості про дискові пристрої. Принцип дії жорсткого диска.** Накопичувачі на жорстких магнітних дисках(НЖМД)(рис. 5.1, далі – диски) складаються з набору дискових пластин (platters), які покриті магнітним матеріалом і обертаються двигуном із високою швидкістю. Кожній пластині відповідають дві головки (heads), одна зчитує інформацію зверху, інша – знизу. Головки прикріплені до спеціального дискового маніпулятора (disk arm). Маніпулятор може переміщатися по радіусу диска – від центра до зовнішнього краю і назад, таким чином відбувається позиціювання головок.

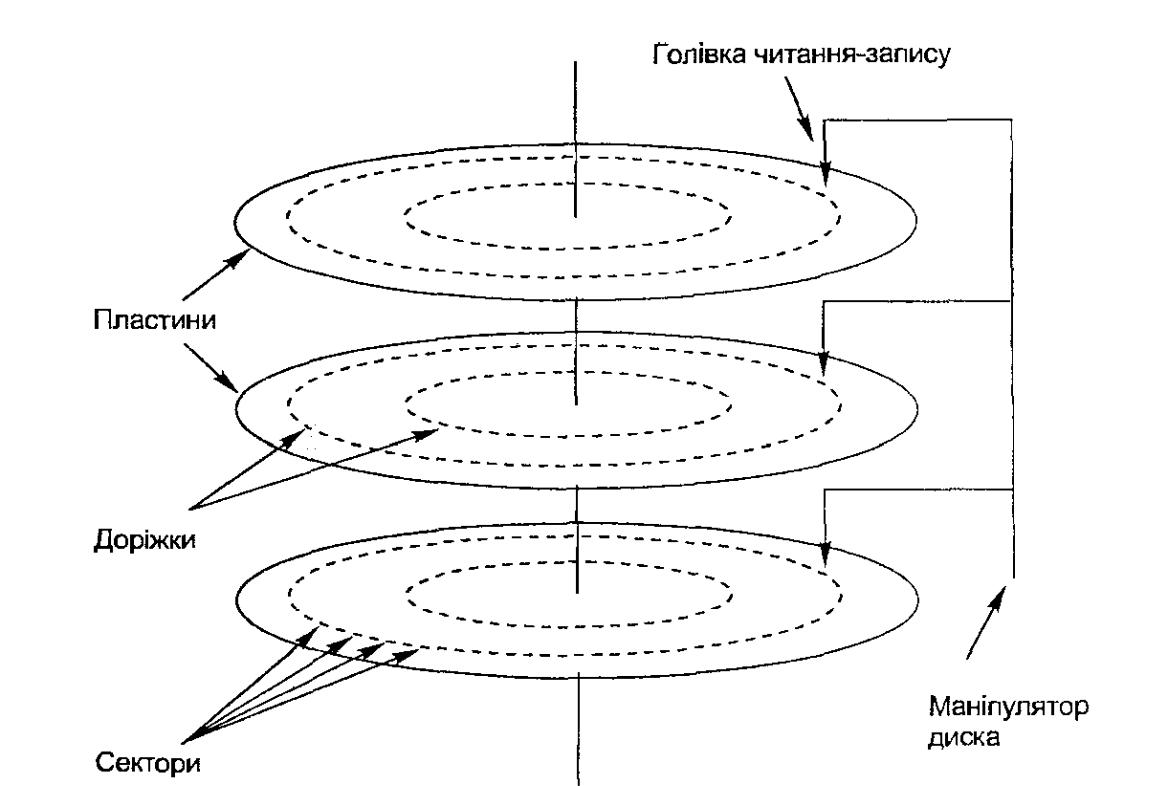


Рис. 5.1. Схематична будова накопичувача на жорстких магнітних дисках

Головки зчитують інформацію із доріжок (tracks), які мають вигляд концентричних кіл. Мінімальна кількість доріжок на поверхні пластини в сучасних дисках – 700, максимальна – більше 20 000. Сукупність усіх доріжок одного радіуса на всіх поверхнях пластин називають циліндром.

Кожну доріжку під час низькорівневого форматування розбивають на сектори (sectors), обсяг даних сектора для більшості архітектур становить 512 байт (він обов'язково має дорівнювати степеню числа 2). Кількість секторів для всіх доріжок однакова (у діапазоні від 16 до 1600).

**Ефективність операцій доступу до диска**.Основнимихарактеристиками доступу до диска є:

– час пошуку (seek time) – час переміщення маніпулятора для позиціювання головки на потрібній доріжці (у середньому становить від 10 до 20 мс);

– ротаційна затримка (rotational delay) – час очікування, поки пластина повернеться так, що потрібний сектор опиниться під доріжкою (у середньому становить 8 мс);

– пропускна здатність передавання даних (transfer bandwidth) – обсяг даних, що передаються від пристрою в пам'ять за одиницю часу; для сучасних дисків ця характеристика порівнянна із пропускною здатністю оперативної пам'яті (200 Мбайт/с), час передавання одного сектора вимірюють у наносекундах.

Час, необхідний для читання сектора, одержують додаванням часу пошуку, ротаційної затримки і часу передавання (при цьому час передавання можна вважати дуже малим). Очевидно, що час читання одного сектора практично не відрізняється від часу читання кількох розташованих поряд секторів, а час читання цілої доріжки за одну операцію буде менший, ніж час читання одного сектора через відсутність ротаційної затримки.

Швидкість доступу до диска, порівняно із доступом до пам'яті, надзвичайно мала, зараз диски є основним «вузьким місцем» з погляду продуктивності комп'ютерної системи. При цьому реальне поліпшення останніми роками наявне лише для пропускної здатності передавання даних. Час пошуку і ротаційна затримка майже не змінюються, тому що вони пов'язані з керуванням механічними пристроями (дисковим маніпулятором і двигуном, що обертає пластини) і обмежені їхніми

фізичними характеристиками.

* + результаті час читання великих обсягів неперервних даних усе менше відрізняється від часу читання малих. Як наслідок, першорядне значення для розробників файлових систем набуває розв'язання двох задач:

– організації даних таким чином, щоб ті з них, які будуть потрібні одночасно, перебували на диску поруч (і їх можна було зчитати за одну операцію);

– підвищення якості кешування даних (оскільки пам'яті стає все більше, зростає ймовірність того, що всі потрібні дані міститимуться в кеші, і доступ до диска стане взагалі не потрібний).

**Розміщення інформації у файлових системах.** Файлова системазвичайно будує базове відображення даних поверх того, яке їй надають драйвери дискових пристроїв. Насамперед, ОС розподіляє дисковий простір не секторами, а спеціальними одиницями розміщення – кластерами (clusters) або дисковими блоками (disk blocks, термін «дисковий блок» більш розповсюджений в UNIX-системах). Визначення розміру кластера і розміщення інформації, необхідної для функціонування файлової системи, відбувається під час високорівневого форматування розділу. Саме таке форматування створює файлову систему в розділі.

Розмір кластера визначає особливості розподілу дискового простору в системі. Використання кластерів великого розміру може спричинити значну внзгтрішню фрагментацію через файли, які за розміром менші, ніж кластер.

Деякі додатки (насамперед, сервери баз даних) можуть реалізовувати свою власну фізичну організацію даних на диску. Для них файлова система може виявитися зайвим рівнем доступу, що тільки сповільнюватиме роботу. Багато ОС надають таким застосуванням можливість працювати із розділами, поданими у вигляді простого набору дискових секторів, який не

містить структур даних файлової системи. Про такі розділи кажуть, що вони містять неорганізовану файлову систему (raw file system). Для них не виконують операцію високорівневого форматування.

**Фізична організація розділів на диску.** Перед тим як перейти дорозгляду особливостей фізичної організації файлової системи в рамках розділу, коротко ознайомимося з організацією розділів на диску.

Початковий (нульовий) сектор диска називають головним завантажувальним записом (Master Boot Record, MBR). Наприкінці цього запису міститься таблиця розділів цього диска, де для кожного розділу зберігається початкова і кінцева адреси.

Один із розділів диска може бути позначений як завантажувальний (bootable) або активний (active). Після завантаження комп'ютера апаратне забезпечення звертається до MBR одного з дисків, визначає з його таблиці розділів завантажувальний розділ і намагається знайти в першому кластері цього розділу спеціальну невелику програму – завантажувач ОС (OS boot loader). Саме завантажувач ОС відповідає за пошук на диску і початкове завантаження у пам'ять ядра операційної системи. Усередині розділу розташовані структури даних файлової системи.

**Основні вимоги до фізичної організації файлових систем**.Зпогляду користувача файл із заданим іменем (далі вважатимемо, що інформацію про шлях включено в ім'я) – це неструктурована послідовність байтів, а з погляду фізичної структури файлової системи, файл – це набір дискових блоків, що містять його дані. Завдання файлової системи полягає

* забезпеченні перетворення сукупності імені файла і логічного зсуву в ньому на фізичну адресу всередині відповідного дискового блоку.

Необхідність такого перетворення визначає основне завдання файлової системи – відстежувати розміщення вмісту файлів на диску. Інформація про розміщення даних файла на диску зберігається у структурі

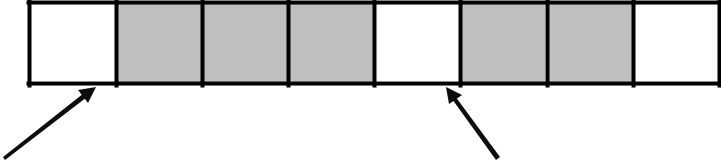
даних, що називають заголовком файла. Такі заголовки звичайно зберігають на диску разом із файлами.

Під час розробки структури даних для такого заголовка потрібно враховувати, що більшість файлів мають малий розмір, а основну частину дискового простору розподіляють, навпаки, під файли великого розміру, із якими переважно і виконують операції введення-виведення.

Оскільки продуктивність файлової системи залежить від кількості операцій доступу до диска, важливо максимально її обмежити. Кілька сотень таких операцій можуть додатково зайняти кілька секунд часу. На практиці слід враховувати, що всі імена файлів (і самі файли) каталогу і всі блоки у файлі зазвичай використовують разом, послідовно.

Принципи, що лежать в основі фізичної організації файлової системи, визначають різні способи розміщення файлів на диску. Крім обліку розміщення даних, фізичне розміщення потребує також обліку вільних кластерів.

**Неперервне розміщення файлів.** Найпростіший підхід дофізичної організації файлових систем – це неперервне розміщення файлів. При цьому кожному файлові відповідає набір неперервно розташованих кластерів на диску (рис. 5.2). Для кожного файла мають зберігатися адреса початкового кластера і розмір файла. +



a.txt (start=1, lеn=3) b.txt (start=5, lеn=2)

Рис. 5.2. Неперервне розміщення файлів

Зазначимо, що розподіл дискового простору в цьому разі подібний до динамічного розподілу пам'яті. Для пошуку вільного блоку на диску можна

використати алгоритми першого підходящого або найкращого підходящого блоку.

Неперервне розміщення файлів вирізняється простотою в реалізації та ефективністю (наприклад, весь файл може бути зчитаний за одну операцію), але має істотні недоліки.

– Під час створення файла користувач має заздалегідь задати його максимальну довжину і виділити весь простір на диску за один раз. Збільшувати розміри файлів під час роботи не можна. У багатьох ситуаціях це абсолютно неприйнятно (наприклад, неможливо вимагати від користувача текстового редактора щоб він вказував остаточну довжину файла перед його редагуванням).

– Вилучення файлів згодом може спричинити велику зовнішню фрагментацію дискового простору з тих самих причин, що й за динамічного розподілу пам'яті.

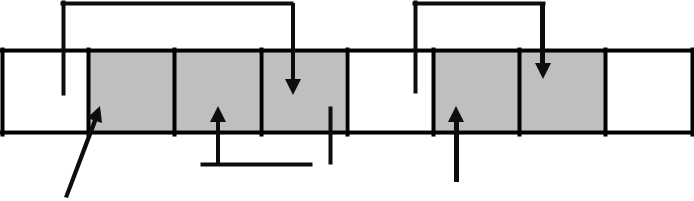
У сучасних ОС для організації даних на жорстких дисках неперервне розміщення майже не використовують, проте його застосовують у таких файлових системах, де можна заздалегідь передбачити, якого розміру буде файл. Прикладом є файлові системи для компакт-дисків. Вони мають кілька властивостей, що роблять неперервне розміщення файлів найкращим рішенням:

– записування такої файлової системи здійснюють повністю за один раз, під час записування для кожного файла заздалегідь відомий його розмір;

– доступ до файлових систем на компакт-диску здійснюють лише для читання, файли в них ніколи не розширюють і не вилучають, тому відсутні причини появи зовнішньої фрагментації.

**Розміщення файлів зв'язними списками. Прості зв'язні списки.** Іншим підходом є організація кластерів,що належать файлу,узв'язний список. Кожен кластер файла містить інформацію про те, де

перебуває наступний кластер цього файла (наприклад, його номер). Найпростіший приклад такого розміщення бачимо на рис. 5.3. Заголовок файла в цьому разі має містити посилання на його перший кластер, вільні кластери можуть бути організовані в аналогічний список.



a.txt (start=1, lеn=3) b.txt (start=5, lеn=2)

Рис. 5.3. Найпростіший приклад зв'язаного розміщення файлів

Розміщення файлів з використанням зв'язних списків надає такі переваги:

– відсутність зовнішньої фрагментації (є тільки невелика внутрішня фрагментація, пов'язана з тим, що розмір файла може не ділитися націло на розмір кластера);

– мінімум інформації, яка потрібна для зберігання у заголовку файла (тільки посилання на перший кластер);

– можливість динамічної зміни розміру файла;

– простота реалізації керування вільними блоками, яке принципово не відрізняється від керування розміщенням файлів.

Цей підхід, однак, не позбавлений і серйозних недоліків:

– відсутність ефективної реалізації випадкового доступу до файла: для того щоб одержати доступ до кластера з номером п, потрібно прочитати всі кластери файла з номерами від 1 до n-1;

– зниження продуктивності тих додатків, які зчитують дані блоками, за розміром рівними степеню числа 2 (а таких додатків досить багато): частина будь-якого кластера повинна містити номер наступного, тому корисна інформація в кластері займає обсяг, не кратний його розміру (цей обсяг навіть не є степенем числа 2);

– можливість втрати інформації у послідовності кластерів: якщо внаслідок збою буде втрачено кластер на початку файла, вся інформація в кластерах, що йдуть за ним, також буде втрачена.

Є модифікації цієї схеми, які зберегли своє значення дотепер, найважливішою 3 них є використання таблиці розміщення файлів.

**Зв'язні списки з таблицею розміщення файлів.** Цей підхід(рис.5.4) полягає в тому, що всі посилання, які формують списки кластерів файла, зберігаються в окремій ділянці файлової системи фіксованого розміру, формуючи таблицю розміщення файлів (File Allocation Table, FAT). Елемент такої таблиці відповідає кластеру на диску і може містити:

– номер наступного кластера, якщо цей кластер належить файлу і не є його останнім кластером;

– індикатор кінця файла, якщо цей кластер є останнім кластером файла;

– індикатор, який показує, що цей кластер вільний.

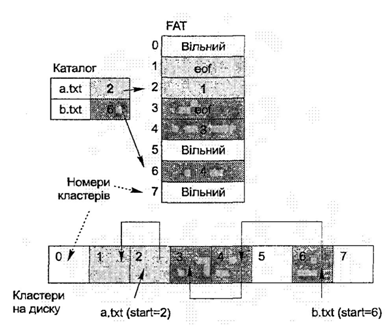


Рис. 5.4. Використання таблиці розміщення файлів

Для організації файла достатньо помістити у відповідний йому елемент каталога номер першого кластера файла. За необхідності прочитати файл система знаходить за цим номером кластера відповідний елемент FAT, зчитує із нього інформацію про наступний кластер і т. д. Цей процес триває доти, поки не трапиться індикатор кінця файла.

Використання цього підходу дає змогу підвищити ефективність і надійність розміщення файлів зв'язними списками. Це досягається завдяки тому, що розміри FAT дозволяють кешувати її в пам'яті. Через це доступ до диска під час відстеження посилань заміняють звертаннями до оперативної пам'яті. Зазначимо, що навіть якщо таке кешування не реалізоване, випадковий доступ до файла не призводитиме до читання всіх попередніх його кластерів – зчитані будуть тільки попередні елементи FAT.

Крім того, спрощується захист від збоїв. Для цього, наприклад, можна зберігати на диску додаткову копію FAT, що автоматично синхронізуватиметься з основною. У разі ушкодження однієї з копій інформація може бути відновлена з іншої.

* + нарешті, службову інформацію більше не зберігають безпосередньо
* кластерах файла, вивільняючи в них місце для даних. Тепер обсяг корисних даних всередині кластера майже завжди (за винятком, можливо, останнього кластера файла) дорівнюватиме степеню числа 2.

Однак, у разі такого способу розміщення файлів для розділів великого розміру обсяг FAT може стати доволі великим і її кешування може потребувати значних витрат пам'яті. Скоротити розмір таблиці можна, збільшивши розмір кластера, але це, в свою чергу, призводить до збільшення внутрішньої фрагментації для малих файлів (менших за розмір кластера).

Також руйнування обох копій FAT (внаслідок апаратного збою або дії програми-зловмисника, наприклад, комп'ютерного вірусу) робить

відновлення даних дуже складною задачею, яку не завжди можна розв'язати.

Про реалізацію цього підходу йтиметься в розділі 13 під час знайомства із файловими системами лінії FAT.

**Індексоване розміщення файлів.** Базовою ідеєю ще одногопідходу до розміщення файлів є перелік адрес всіх кластерів файла в його заголовку. Такий заголовок файла дістав назву індексного дескриптора, або і-вузла (іnode), а сам підхід - індексованого розміщення файлів.

За індексованого розміщення із кожним файлом пов'язують його індексний дескриптор. Він містить масив із адресами (або номерами) усіх кластерів цього файла, при цьому п-й елемент масиву відповідає *n*-му кластеру. Індексні дескриптори зберігають окремо від даних файла, для цього звичайно виділяють на початку розділу спеціальну ділянку індексних дескрипторів. В елементі каталогу розміщують номер індексного дескриптора відповідного файла (рис. 5.5).

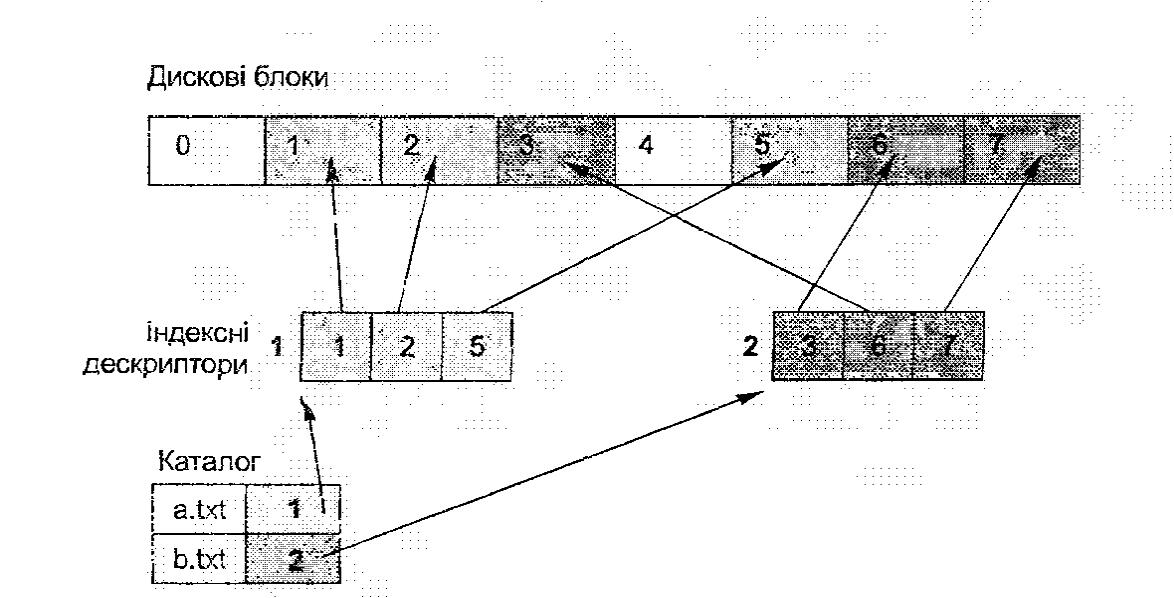


Рис. 5.5. Індексоване розміщення файлів

Під час створення файла на диску розміщують його індексний дескриптор, у якому всі покажчики на кластери спочатку є порожніми. Під час першого записування в п-й кластер файла менеджер вільного простору виділяє вільний кластер і його номер або адресу заносять у відповідний елемент масиву.

Цей підхід стійкий до зовнішньої фрагментації й ефективно підтримує як послідовний, так і випадковий доступ (інформація про всі кластери зберігається компактно і може бути зчитана за одну операцію). Для підвищення ефективності індексний дескриптор повністю завантажують у пам'ять, коли процес починає працювати з файлом, і залишають у пам'яті доти, поки ця робота триває.

**Структура індексних дескрипторів.** Основною проблемою єпідбір розміру і задання оптимальної структури індексного дескриптора, оскільки:

– з одного боку, зменшення розміру дескриптора може значно зекономити дисковий простір і пам'ять (дескрипторів потрібно створювати значну кількість – по одному на кожний файл, разом вони можуть займати досить багато місця на диску; крім того, для кожного відкритого файла дескриптор буде розташовано в оперативній пам'яті).

– з іншого боку, дескриптора надто малого розміру може не вистачити для розміщення інформації про всі кластери великого файла.

Одне з компромісних розв'язань цієї задачі, яке застосовується вже багато років у UNIX-системах, зображене на рис. 5.6. Під час його опису замість терміна «кластер» вживатимемо його синонім «дисковий блок».

* цьому випадку індексний дескриптор містить елементи різного призначення.

– Частина елементів (зазвичай перші 12) безпосередньо вказує на дискові блоки, які називають прямими (direct blocks). Отже, якщо файл може вміститися у 12 дискових блоках (за розміру блоку 4 Кбайт максимальний розмір такого файла становитиме 4096 х 12 = 49 152 байти), усі ці блоки будуть прямо адресовані його індексним дескриптором і жодних додаткових структур даних не буде потрібно.

– Якщо файлу необхідно для розміщення даних більше, ніж 12 дискових блоків, використовують непряму адресацію першого рівня. У цьому разі 13-й елемент індексного дескриптора вказує не на блок із даними, а на спеціальний непрямий блок першого рівня (single indirect block). Він містить масив адрес наступних блоків файла (за розміру блоку 4 Кбайт, а адреси – 4 байти в ньому міститимуться адреси 1024 блоків, при цьому максимальний розмір файла буде 4096 х (12+1024) = 4234456 байт).

– Якщо файлу потрібно для розміщення більше ніж 1024 + 12 = 1036 дискових блоків, використовують непряму адресацію другого рівня. 14 -й елемент індексного дескриптора в цьому разі вказуватиме на непрямий блок другого рівня (double indirect block). Такий блок містить масив з 1024 адрес непрямих блоків першого рівня, кожен із них, як зазначалося, містить масив адрес дискових блоків файла. Тому за допомогою такого блоку можна адресувати 10242 додаткових блоків.

– Нарешті, якщо файлу потрібно більше ніж 1036 + 10242 дискових блоків, використовують непряму адресацію третього рівня. Останній 15-й елемент індексного дескриптора вказуватиме на непрямий блок третього рівня (triple indirect block), що містить масив з 1024 адрес непрямих блоків другого рівня, даючи змогу адресувати додатково 10243 дискових блоків.

Розмір блоку може відрізнятися від 4 Кбайт. Чим більший блок, тим більшим є розмір, що може бути досягнутий файлом, поки не виникне необхідності у непрямій адресації вищого рівня. З іншого боку, більший розмір блоку спричиняє більшу внутрішню фрагментацію.

Можна розподіляти дисковий простір не блоками (кластерами), а їхніми групами (неперервними ділянками із кількох дискових блоків). Такі групи ще називають екстентами (extents). Кожен екстент характеризується довжиною (у блоках) і номером початкового дискового блоку. Коли виникає необхідність виділити кілька неперервно розташованих блоків одночасно, замість цього виділяють лише один екстент потрібної довжини. У результаті обсяг службової інформації, яку потрібно зберігати, може бути скорочений.

Характер перетворення адреси в номер кластера робить цей підхід аналогом сторінкової організації пам'яті, причому індексний дескриптор відповідає таблиці сторінок.

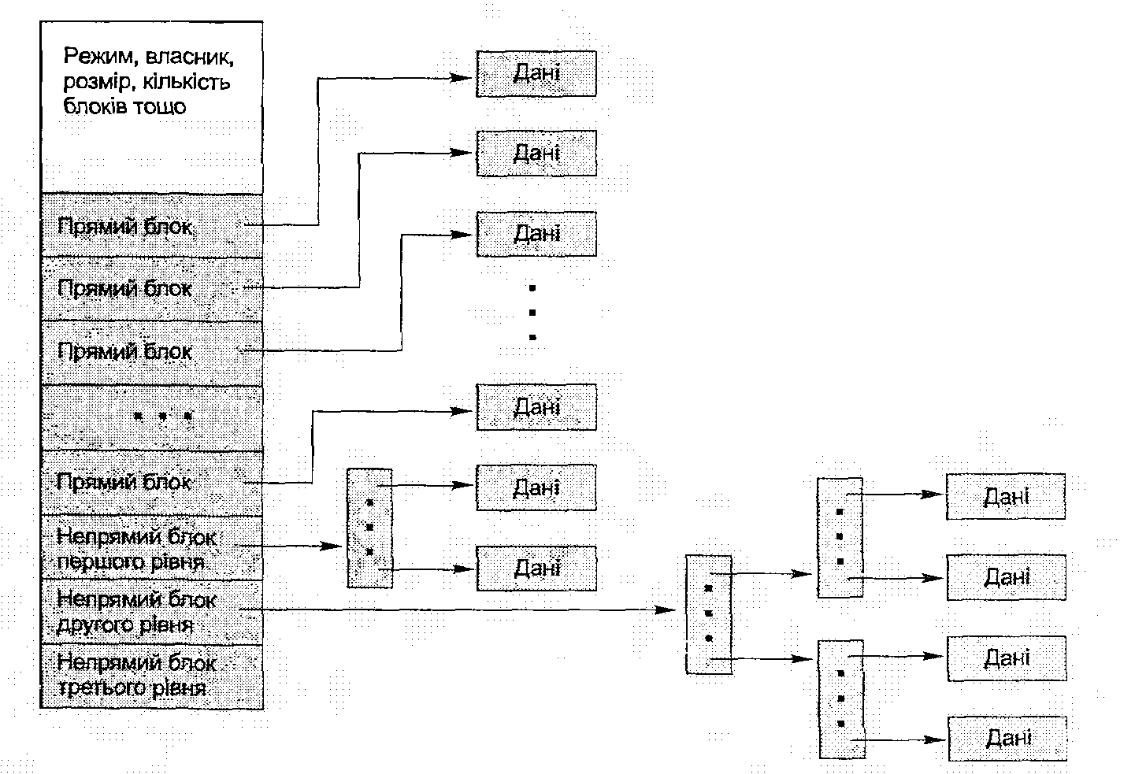


Рис. 5.6 . Структура індексного дескриптора в UNIX

**Розріджені файли.** Багато операційних систем не зберігаютьпокажчики на дискові блоки файлів у їхніх індексних дескрипторах, поки до них не було доступу для записування.

Фрагменти, до яких цього доступу не було з моменту створення файла, називають «дірами» (holes), дисковий простір під них не виділяють, але під час розрахунку довжини файла їх враховують. У разі читання вмісту «діри» повертають блоки, заповнені нулями, звертання до диска не відбувається.

На практиці «діри» найчастіше виникають, коли покажчик поточної позиції файла переміщують далеко за його кінець, після чого виконують операцію записування. У результаті розмір файла збільшується без додаткового виділення дискового простору. Подібні файли називають розрідженими файлами (sparse files).

Вони реально займають на диску місця набагато менше, ніж їхня довжина, фактично довжина розрідженого файла може перевищувати розмір розділу, на якому він перебуває.

**Організація каталогів.** Каталоги звичайно організують якспеціальні файли, що містять набір елементів каталогу (directory entries), кожен з яких відповідає одному файлові.

**Елементи каталогу.** Елемент каталогу обов'язково містить ім'яфайла та інформацію, що дає змогу за іменем файла знайти на диску адреси його кластерів. Структуру такої інформації визначають підходи до розміщення файлів: для неперервного розміщення в елементі каталогу зберігатиметься адреса початкового кластера і довжина файла, для розміщення зв'язними списками – тільки адреса або номер початкового кластера, для індексованого розміщення достатньо зберігати номер індексного дескриптора файла.

Крім обов'язкових даних, елемент каталогу може зберігати додаткову інформацію, характер якої залежить від реалізації. Це може бути, наприклад, набір атрибутів файла (так найчастіше роблять при неперервному розміщенні або розміщенні зв'язаними списками). З іншого боку, за індексованого розміщення всі атрибути файла та іншу службову інформацію зберігають в індексному дескрипторі, а в елемент каталогу додаткову інформацію не заносять (там є тільки ім'я файла і номер дескриптора).

**Організація списку елементів каталогу.** Реалізація каталогувключає також організацію списку його елементів. Найчастіше елементи об'єднують у лінійний список, але якщо очікують, що в каталогах буде багато елементів, для підвищення ефективності пошуку файла можна використати складніші структури даних, такі як бінарне дерево пошуку або хеш-таблиця. Для прискорення пошуку можна також кешувати елементи каталогу, при цьому під час кожного пошуку файла спочатку перевіряється його наявність у кеші, у разі влучення пошук буде зроблено дуже швидко.

**Організація підтримки довгих імен файлів**.Розглянемо,якимчином у каталозі зберігають довгі імена файлів. Є ряд підходів до вирішення цієї проблеми.

– Найпростіше зарезервувати простір у кожному елементі каталогу для максимально допустимої кількості символів у імені. Такий підхід можна використати, якщо максимальна кількість символів невелика, у протилежному випадку місце на диску витрачатиметься даремно, оскільки більша частина імен не займатиме весь зарезервований простір.

– Можна зберігати довгі імена в елементах каталогу повністю, у цьому випадку довжина такого елемента не буде фіксованою. Перед кожним елементом каталогу зберігають його довжину, а кінець імені файла позначають спеціальним (зазвичай нульовим) символом. Недоліки цього підходу пов'язані з тим, що через різну довжину елементів виникає зовнішня фрагментація і каталог надто великого розміру може зайняти кілька сторінок у пам'яті, тому під час перегляду такого каталогу є ризик виникнення сторінкових переривань.

– Нарешті, можна зробити всі елементи каталогу однієї довжини, при цьому кожен із них міститиме покажчик на довге ім'я. Усі довгі імена зберігатимуться окремо (наприклад, наприкінці каталогу). Це вирішує проблему зовнішньої фрагментації для елементів каталогу, але питання про керування ділянкою зберігання довгих імен залишається відкритим.

**Облік вільних кластерів.** Порад з урахуванням кластерів,виділених для розміщення даних файла, файлові системи мають вести облік вільних кластерів. Це насамперед необхідно для того щоб розв'язати задачу виділення нових кластерів для даних. Для організації керування вільним дисковим простором найчастіше використовують два підходи.

– Бітовий масив (бітова карта кластерів), у якій кожен біт відповідає одному кластеру на диску. Якщо відповідний кластер вільний, біт дорівнює одиниці, якщо зайнятий – нулю. Головна перевага такого підходу полягає в тому, що пошук першого ненульового біта можна легко реалізувати, спираючись на апаратну підтримку.

– Зв'язний список вільних кластерів. Такий підхід, як зазначалося, найзручніше використати, коли зв'язні списки використовують і для організації розміщення файлів. Звичайно в цьому разі організують список, елементами якого є кластери з адресами (номерами) вільних кластерів на диску.

Достатньо зберігати в пам'яті один елемент списку вільних кластерів або один кластер із бітовою картою. Коли вільні блоки в ньому закінчаться, зчитують наступний елемент. У разі вилучення файла номери його кластерів додають у поточний елемент списку або в поточну бітову карту. Коли місця там більше немає, поточний елемент (карту) записують на диск, в пам'яті створюють новий елемент або нову карту, куди заносять номери кластерів, для яких забракло місця.

**Продуктивність файлових систем.** Розглянемо різні підходи,яківикористовують для підвищення продуктивності файлових систем. Є дві категорії таких підходів.

– До першої належать різні підходи, які можуть бути впроваджені на етапі проектування і розробки файлової системи: керування розміром блоку, оптимізація розміщення даних, застосування продуктивніших алгоритмів і структур даних.

–До другої належать низькорівневі універсальні підходи, прозорі для файлової системи (їхнє впровадження може підвищити продуктивність файлової системи без її модифікації). До них належать реалізація дискового кеша і планування переміщення голівок диска.

**Оптимізація продуктивності під час розробки файлових систем**.Розглянемо,яким чином можна оптимізувати продуктивністьфайлової системи зміною структур даних і алгоритмів, які в ній застосовують. У викладі використовуватимемо класичний приклад оптимізації традиційної файлової системи вихідної версії UNIX під час розроблення системи Fast File System (FFS) для BSD UNIX (у наш час ця файлова система також відома як ufs).

Традиційна файлова система UNIX складається із суперблока (що містить номери блоків файлової системи, поточну кількість файлів, покажчик на список вільних блоків), ділянки індексних дескрипторів і блоків даних (рис. 5.7).

Розмір блока фіксований і становить 512 байт. Вільні блоки об'єднані у список.

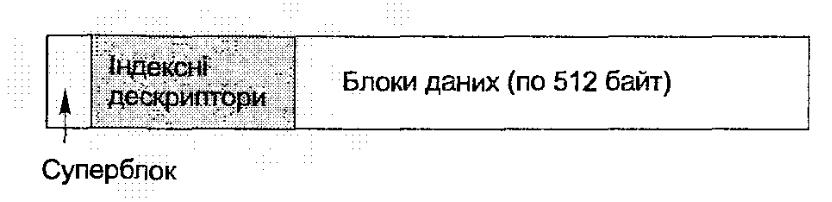


Рис. 5.7. Традиційна файлова система UNIX

Така система є прикладом простого і витонченого вирішення, яке виявилось неприйнятним із погляду продуктивності. На практиці ця файлова система могла досягти на пересиланні даних пропускної здатності, що становить усього 2 % можливостей диска. Назвемо деякі причини такої низької продуктивності.

– Розмір дискового блоку виявився недостатнім, внаслідок чого для розміщення даних файла була потрібна велика кількість блоків; індексні дескриптори навіть для невеликих файлів потребували кількох рівнів непрямої адресації, перехід між якими сповільнював доступ; пересилання даних одним блоком призводила до зниження пропускної здатності.

– Пов'язані об'єкти часто виявлялися віддаленими один від одного і не могли бути зчитані разом, зокрема, індексні дескриптори були розташовані далеко від блоків даних і для каталогу не перебували разом; послідовні блоки для файла також не містилися разом (це траплялося тому, що протягом експлуатації системи через вилучення файлів список вільних блоків ставав «розкиданим» по диску, внаслідок чого файли під час створення отримували блоки, віддалені один від одного).

До розв'язання цих проблем під час розробки файлової системи FFS були запропоновані декілька підходів.

Насамперед, у цій системі було збільшено розмір дискового блока (у FFS звичайно використовували два розміри блока: 4 і 8 Кбайт). Для того щоб уникнути внутрішньої фрагментації (яка завжди зростає зі збільшенням розміру блока), було запропоновано в разі необхідності розбивати невикористані блоки на частини меншого розміру – фрагменти, які можна використати для розміщення невеликих файлів. Мінімальний розмір фрагмента дорівнює розміру сектора диска, звичайно було використано фрагменти на 1 Кбайт.

Крім того, велику увагу було приділено групуванню взаємозалежних даних. З огляду на те, що найбільші втрати часу трапляються під час переміщення головки, було запропоновано розміщувати такі дані в рамках групи циліндрів, яка об'єднує один або кілька суміжних циліндрів. Під час доступу до даних однієї такої групи головку переміщувати було не потрібно, або її переміщення виявлялося мінімальним. Кожна така група за своєю структурою повторювала файлову систему: у ній був суперблок, ділянка індексних дескрипторів і ділянка дискових блоків, виділених для файлів. Тому індексний дескриптор кожного файла розміщувався в тому самому циліндрі, що і його дані, в одній групі циліндрів розміщувалися також всі індексні дескриптори одного каталогу. Послідовні блоки файла прагнули розміщувати в суміжних секторах.

Нарешті, ще одна важлива зміна була зроблена у форматі зберігання інформації про вільні блоки – список вільних блоків було замінено бітовою картою, яка могла бути повністю завантажена у пам'ять. Пошук суміжних блоків у такій карті міг бути реалізований ефективніше. Для ще більшої ефективності цього процесу в системі постійно підтримували деякий вільний простір на диску (коли є вільні дискові блоки, ймовірність знайти суміжні блоки зростає).

Внаслідок реалізації цих і деяких інших рішень пропускна здатність файлової системи зросла в 10-20 разів (до 40 % можливостей диска).

На підставі цього прикладу можна зробити такі висновки:

– розмір блоку впливає на продуктивність файлової системи, при цьому потрібно враховувати можливість внутрішньої фрагментації;

– програмні зусилля, витрачені на скорочення часу пошуку і ротаційної затримки, окупаються (насамперед вони мають спрямовуватися на забезпечення суміжного розміщення взаємозалежної інформації);

– використання бітової карти вільних блоків теж спричиняє підвищення продуктивності.

Ідеї, що лежать в основі FFS, вплинули на особливості проектування файлової системи ext2fs – основної файлової системи Linux.

**Кешування доступу до диска.** Найважливішим засобомпідвищення продуктивності файлових систем є організація дискового кеша (disk cache). Зупинимося докладніше на цьому найважливішому компоненті операційної системи.

Дисковим кешем називають спеціальну ділянку в основній пам'яті, яку використовують для кешування дискових блоків. У разі спроби доступу до дискового блока його поміщають в кеш, розраховуючи на те, що він незабаром буде використаний знову. Під час наступного повторного використання можна обійтися без звертання до диска.

Керування дисковим кешем відбувається на нижчому рівні порівняно

* реалізацією файлової системи. Кеш має бути прозорий для файлової системи: блоки, що перебувають у ньому з погляду файлової системи розташовані на диску, відмінності є тільки у швидкості доступу.

Загальний принцип організації дискового кеша показано на рис. 5.8. Для прискорення пошуку потрібного блоку звичайно використовують хеш-функцію, що переводить номер блока у хеш-код. У пам'яті зберігають хеш-таблицю, елементи якої відповідають окремим значенням хеш-коду. Усі блоки з однаковим значенням хеш-коду об'єднують у зв'язні списки. Для пошуку в кеші блока із конкретним номером досить обчислити значення хеш-функції та переглянути відповідний список.

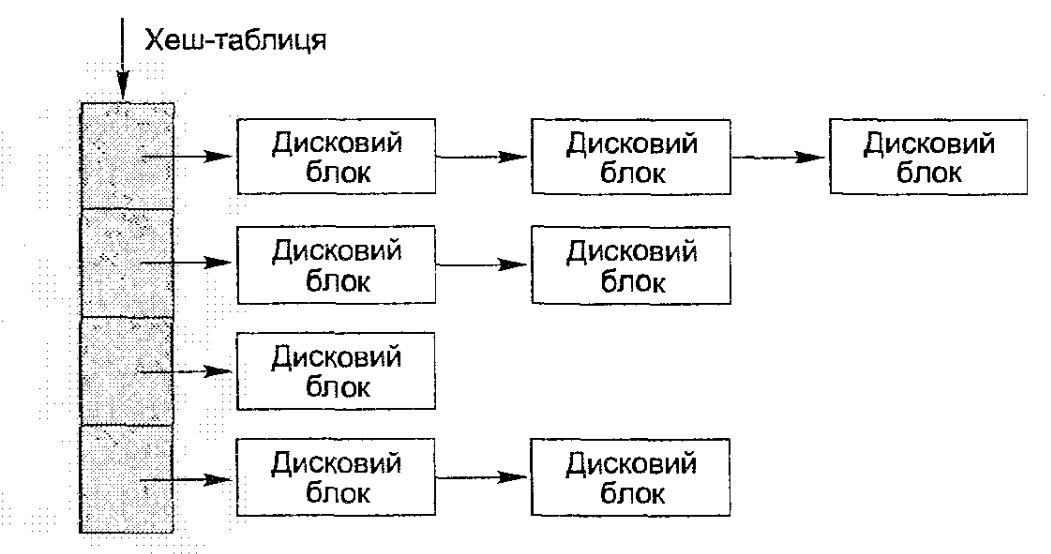


Рис. 5.8 . Дисковий кеш

Під час реалізації керування дисковим кешем необхідно отримати відповіді на такі запитання.

– Якого розміру має бути кеш?

– Як має бути організоване заміщення блоків у кеші?

– Як організувати збереження модифікованої інформації із кеша на диск?

– Яким чином оптимізувати завантаження блоків у кеш?

**Розмір дискового кеша.** Визначаючи розмір дискового кеша,важливо пам'ятати, що він разом із менеджером віртуальної пам'яті з погляду використання основної пам'яті є двома головними підсистемами, що конкурують між собою.

* два різновиди кеша з погляду керування його розміром: кеш фіксованого розміру і кеш змінного розміру.

Для кеша фіксованого розміру межі дискового і сторінкового кешів задають жорстко і змінюватися під час виконання вони не можуть. Недоліком такого кеша є те, що він не може підлаштовуватися під зміну навантаження в системі (наприклад, коли відкрито багато файлів, може знадобитися розширити дисковий кеш, коли завантажено багато процесів

– сторінковий).

Стандартна реалізація кеша змінного розміру виділяє спільну пам'ять

* під сторінковий, і під дисковий кеш. Під час виконання кожна із підсистем працює з цією пам'яттю, розміщуючи там і сторінки, і дискові блоки (зручно, якщо розмір сторінки дорівнює розміру блока). Недоліком такої реалізації є те, що один файл великого розміру, відкритий процесом, може зайняти блоки, призначені не тільки для відкритих файлів інших процесів, але й для сторінкового кеша віртуальної пам'яті. Для вирішення цієї проблеми можна перейти до змішаного керування кешем, коли межі задають, але їх можна змінити під час виконання.

**Заміщення інформації в кеші.** Організація заміщення блоків удисковому кеші багато в чому подібна до реалізації заміщення віртуальної пам'яті. Фактично для визначення заміщуваного блока можна використати більшість алгоритмів заміщення сторінок, такі як FIFO, LRU, годинниковий алгоритм. Зазначимо, що в цьому випадку LRU-алгоритм реалізувати простіше, оскільки звертання до дискового кеша відбувається значно рідше, ніж звертання до пам'яті, і за тривалістю інтервал між такими звертаннями значно перевищує той час, який потрібно затратити на фіксування моменту звертання. Для реалізації LRU-алгоритму можна підтримувати список усіх блоків кеша, упорядкований за часом використання (найстаріший блок - на початку, найновіший – наприкінці), переміщуючи блоки у кінець списку під час звертання до них.

Необхідно враховувати, що за такої ситуації LRU-алгоритм не завжди

* кращим або навіть просто прийнятним вирішенням. Наприклад, якщо деякий критично важливий блок не буде записаний на диск після його модифікації, подальший збій може бути фатальним для всієї файлової системи. У той же час у разі використання LRU-алгоритму цей блок буде поміщено в кінець списку, і його записування відкладеться на якийсь час.

Не рекомендують також використовувати алгоритм LRU у численних ситуаціях, коли розмір файла, який зчитують послідовно, перевищує розмір блока.

* цьому разі найоптимальнішою є саме зворотна стратегія – MRU (Most Recently Used), коли витісняють блок, що був використаний останнім.

Розглянемо приклад. Нехай файл містить 4 дискові блоки, а в кеші є місце для трьох. Файл зчитують послідовно, при цьому отримують рядок посилань 1234. Якщо використати LRU-алгоритм, то при зчитуванні блоку 4 він заміщує в пам'яті блок 1. Якщо тепер прочитати файл іще раз (отримавши рядок посилань 12341234), побачимо, що першим потрібним блоком буде саме блок 1, який щойно витиснули.

Доведеться його зчитувати ще раз, при цьому буде заміщено блок 2, який відразу ж виявиться потрібним знову і т. д. У MRU-алгоритмі блок 4 під час першого читання заміщує блок 3, блоки 1, 2 і 4 під час другого читання перебуватимуть у кеші. На практиці застосовуються модифіковані схеми LRU, які беруть до уваги категорії важливості блоків, а в разі послідовного доступу до файлів, більших за розміром, ніж блок, зводяться до схеми MRU.

**Наскрізне і відкладене записування**.З погляду реалізаціїзаписування модифікованих даних на диск розрізняють два основні типи дискових кешів: із наскрізним (write through cache) і з відкладеним записом (write back cache).

Для кеша із наскрізним записом у разі будь-якої модифікації блоку, що перебуває в кеші, його негайно зберігають на диску. Основною перевагою такого підходу (широко розповсюдженого в часи персональних ОСІ реалізованого, наприклад, в MS-DOS) є те, що файлова система завжди перебуватиме в несуперечливому стані. Головний недолік цього підходу полягає у значному зниженні продуктивності під час записування даних.

Для кеша із відкладеним записом (такі кеші застосовують у більшості сучасних ОС) під час модифікації блоку його позначають відповідним чином, але на диску не зберігають. Записування модифікованих блоків на диск здійснюється окремо за необхідності (зазвичай у фоновому режимі). Цей підхід значно виграє у продуктивності, але вимагає додаткових зусиль із забезпечення надійності та несуперечності файлової системи. Якщо система зазнає краху в той момент, коли модифіковані блоки ще не були записані на диск, без додаткових заходів всі зміни втратяться.

Основна проблема, пов'язана із реалізацією відкладеного записування, полягає у виборі ефективного компромісу між продуктивністю і надійністю. Що більший інтервал між збереженням модифікованих даних на диску, то вища продуктивність кеша, але більше роботи потрібно виконати для поновлення даних у разі збою.

Відокремлюють чотири випадки, коли зміни в кеші зберігаються на диску:

– модифікований блок витісняють з кеша відповідно до алгоритму заміщення (аналогічно до того, як це реалізовано для віртуальної пам'яті);

– закривають файл;

– явно видають команду зберегти всі зміни із кеша на диск; в UNIX-системах таку команду звичайно називають sync, вона зводиться до виконання системного виклику із тим самим ім'ям:

#іinclude <unistd.h>

Sync(): // скидання всіх даних з кеша на диск

– проходить заданий проміжок часу (в UNIX-системах за цим стежить спеціальний фоновий процес, який називають update; за замовчуванням такий інтервал для нього становить 30 хвилин).

Системний виклик sync() зберігає всі зміни із кеша на диск. Гнучкішим підходом є збереження змін для конкретних файлів. Для цього

у POSIX передбачено системний виклик fsync(). Він блокує виконання процесу до повного збереження всіх буферів відкритого файла із пам'яті на диск.

Аналогом fsync() у Win32 АРІ є функція FlushFileBuffers():

Найважливіші блоки (наприклад, блоки з довідковою інформацією, каталогами, атрибутами файлів) варто зберігати якнайчастіше, можливо, навіть після кожної модифікації; блоки із даними можна зберігати рідше.

Для застосувань, які працюють із важливими даними, рекомендовано час від часу робити явне збереження змін на диску (наприклад, виконувати системні виклики sync(), fsync() або їхні аналоги). Для таких додатків, як СУБД, найкращим вирішенням в основному є повна відмова від дискового кеша (використання прямого режиму доступу до диска) і реалізація свого власного кешування. Уже згадувалося про те, що часто такі застосування зовсім відмовляються від послуг файлової системи.

Для реалізації прямого доступу до диска в Linux необхідно під час відкриття файла увімкнути спеціальний прапорець 0\_DIRECT: