ПЛАН-КОНСПЕКТ ЛЕКЦІЇ
з дисципліни «Системне адміністрування ОС Linux»

Викладач: студент групи 641м Бужак Андрій
Дата проведення: 17.09.2021
Група: 541 м
Вид заняття: лекція
Тривалість пари: 80 хвилин
Тема: ***Поняття файла і файлової системи. Організація інформації у файловій системі.***
Мета: Ознайомити студентів з особливостями організації інформації у файлових системах.

ХІД РОБОТИ:

Файлова система. Файлова система – це частина операційної системи, призначення якої полягає в тому, щоб забезпечити користувачеві зручний інтерфейс при роботі з даними, що зберігаються на диску, і забезпечити спільне використання файлів кількома користувачами і процесами.

У широкому розумінні поняття "файлова система" містить:

• сукупність всіх файлів на диску;

• набори структур даних, використовуваних для управління файлами, такі, наприклад, як каталоги файлів, дескриптори файлів, таблиці розподілу вільного і зайнятого простору на диску;

• комплекс системних програмних засобів, що реалізують управління файлами, зокрема: створення, знищення, читання, запис, іменування, пошук і інші операції над файлами.

Імена файлів. Файли ідентифікуються іменами. Користувачі можуть давати файлам символьні імена, при цьому враховуються обмеження ОС як на використовувані символи, так і на довжину імені. Донедавна ці кордони були вельми вузькими. Так у популярній файловій системі FAT довжина імен обмежується певною схемою 8.3 (8 символів – власне ім'я, 3 символи – розширення імені), а в ОС UNIX SystemV ім'я не може містити більше 14 символів. Однак користувачеві набагато зручніше працювати з довгими іменами, оскільки вони дозволяють дати файлу дійсно мнемонічну назву, за якою навіть через значний проміжок часу можна буде згадати, що містить цей файл. Тому сучасні файлові системи, як правило, підтримують довгі символьні імена файлів. Наприклад, Windows NT у своїй файловій системі NTFS припускає, що ім'я файлу може містити до 255 символів, не рахуючи завершального нульового символу.

При переході до довгих імен виникає проблема сумісності з раніше створеними додатками, що використовують короткі імена. Щоб додатки могли звертатися до файлів відповідно до прийнятих раніше угод, файлова система повинна вміти надавати еквівалентні короткі імена (псевдоніми) файлам, які мають довгі імена. Таким чином, одним із важливих завдань стає проблема генерації відповідних коротких імен.

Довгі імена підтримуються не тільки новими файловими системами, а новими версіями добре відомих файлових систем. Наприклад, в ОС Windows 95 використовувалась файлова система VFAT, що являла собою істотно змінений варіант FAT. Серед багатьох інших удосконалень однією з головних переваг VFAT є підтримка довгих імен. Крім проблеми генерації еквівалентних коротких імен, при реалізації нового варіанту FAT важливим завданням було завдання зберігання довгих імен за умови, що принципово метод зберігання і структура даних на диску не повинні були змінитися.

Зазвичай різні файли можуть мати однакові символьні імена. У цьому випадку файл однозначно ідентифікується так званим складовим ім'ям, що являє собою послідовність символьних імен каталогів. У деяких системах для одного й того ж файлу не може бути дано кілька різних імен, а в інших таке обмеження відсутнє. В останньому випадку операційна система привласнює файлу додатково унікальне ім'я, так, щоб можна було встановити взаємно-однозначну відповідність між файлом і його унікальним ім'ям. Унікальне ім'я являє собою числовий ідентифікатор і використовується програмами операційної системи. Прикладом такого унікального імені файлу є номер індексного дескриптора в системі UNIX.

Типи файлів. Файли бувають різних типів: звичайні файли, спеціальні файли, файли-каталоги.

Звичайні файли в свою чергу поділяються на текстові та двійкові. Текстові файли складаються з рядків символів, представлених в ASCII-коді. Це можуть бути документи, вихідні тексти програм тощо. Текстові файли можна прочитати на екрані і роздрукувати на принтері. Виконавчі файли не використовують ASCII-коди, вони часто мають складну внутрішню структуру, наприклад, об'єктний код програми або архівний файл. Всі операційні системи повинні вміти розпізнавати хоча б один тип файлів – їх власні виконувані файли.

Спеціальні файли – це файли, асоційовані з пристроями введення-виведення, які дозволяють користувачеві виконувати операції введення-виведення, використовуючи звичайні команди запису у файл або читання з файлу. Ці команди обробляються спочатку програмами файлової системи, а потім на деякому етапі виконання запиту перетворюються ОС в команди управління відповідним пристроєм. Спеціальні файли, так само як і пристрої введення-виведення, діляться на блок-орієнтовані і байт-орієнтовані.

Каталог – це, з одного боку, група файлів, об'єднаних користувачем виходячи з деяких міркувань (наприклад, файли, що містять програми ігор, або файли, що складають один програмний пакет), а з іншого боку – це файл, що містить системну інформацію про групу файлів, які входять до його складу. У каталозі міститься список файлів, що входять до його складу, і встановлюється відповідність між файлами і їх характеристиками (атрибутами).

У різних файлових системах можуть використовуватися в якості атрибутів файлів різні характеристики, наприклад:

- інформація про дозволений доступ;
- пароль для доступу до файлу;
- власник файлу;
- автор файлу;
- ознака "тільки для читання";

• ознака "прихований файл";
• ознака "системний файл";
• ознака "архівний файл";
• ознака "двійковий / символьний";
• ознака "тимчасовий" (видалити після завершення процесу);
• ознака блокування;
• довжина запису;
• покажчик на ключове поле в записі;
• довжина ключа;
• часів створення, останнього доступу і останньої зміни;
• поточний розмір файлу;
• максимальний розмір файлу.

Каталоги можуть безпосередньо містити значення характеристик файлів, як це зроблено в файлової системі MS-DOS, або посилатися на таблиці, що містять ці характеристики, як це реалізовано в ОС UNIX (рис. 4.1). Каталоги можуть утворювати ієрархічну структуру за рахунок того, що каталог нижчого рівня може входити в каталог більш високого рівня (рис. 4.2).

Ієрархія каталогів може бути деревом або мережею. Каталоги утворюють дерево, якщо файлу дозволено входити тільки в один каталог, і мережу – якщо файл може входити відразу в кілька каталогів. В MS-DOS каталоги утворюють деревоподібну структуру, а в Unix'і – мережеву. Як і будь-який інший файл, каталог має символьне ім'я і однозначно ідентифікується складовим ім'ям, що містить ланцюжок символьних імен всіх каталогів, через які проходить шлях від кореня до даного каталогу.

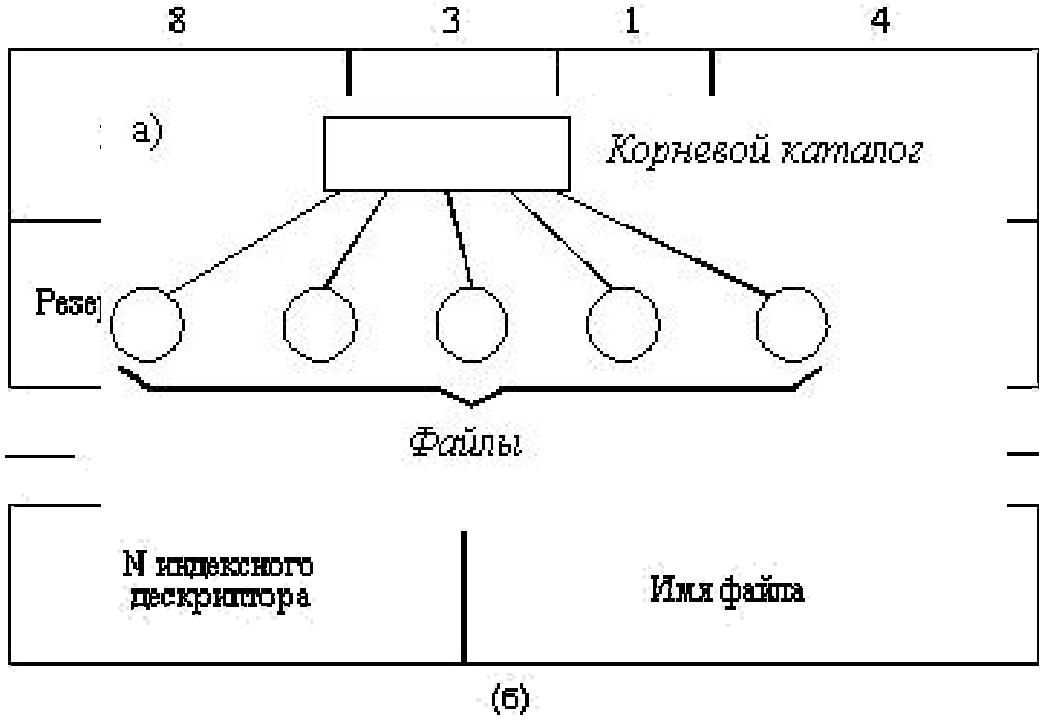

Рис. 4.1. Структура каталогів: а – структура запису каталога MS-DOS (32 байта); б – структура запису каталога ОС UNIX

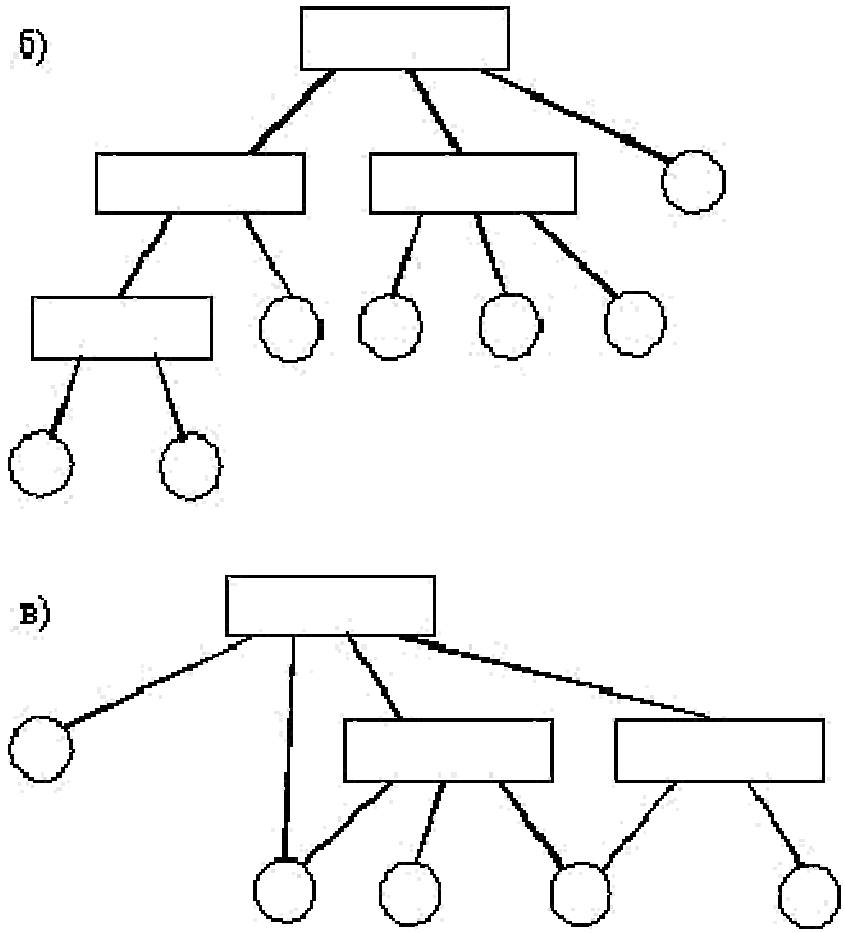

Рис. 4.2. Логічна організація файлової системи
а – однорівнева; б – ієрархічна (дерево); в – ієрархічна (мережа)

Логічна організація файлу. Програміст має справу з логічною організацією файлу, представляючи файл у вигляді певним чином організованих логічних записів. Логічний запис – це найменший елемент даних, яким може оперувати програміст під час обміну даними з зовнішнім пристроєм. Навіть якщо фізичний обмін з пристроєм здійснюється великими одиницями, операційна система забезпечує програмісту доступ до окремого логічного запису. На рис 4.3 показані кілька схем логічної організації файлу. Записи можуть бути фіксованої або змінної довжини. Записи можуть бути розташовані у файлі послідовно (послідовна організація) або в більш складному порядку, з використанням так званих індексних таблиць, що дозволяють забезпечити швидкий доступ до окремого логічного запису (індексно-послідовна організація). Для ідентифікації запису може бути використане спеціальне поле запису, що називається ключем. У файлових системах ОС UNIX і MS-DOS файл має найпростішу логічну структуру – послідовність однобайтових записів.

Индекс	1	2	3	4	5	6
Адрес	21	201	315	661	670	715

Индекс ≡ ключ

Рис. 4.3. Способи логічної організації файлів

Фізична організація і адреса файлу. Фізична організація файлу описує правила розташування файлу на пристрої зовнішньої пам'яті, зокрема на диску. Файл складається з фізичних записів – блоків. Блок – найменша одиниця даних, якою зовнішній пристрій обмінюється з оперативною пам'яттю. Безперервне розміщення – найпростіший варіант фізичної організації (рис 4.4, а), при якому файлу надається послідовність блоків диска, що утворять єдину суцільну ділянку дискової пам'яті. Для завдання адреси файлу в цьому випадку досить вказати тільки номер початкового блоку. Інша перевага цього методу – простота. Але є й два суттєвих недоліки. По-перше, під час створення файлу заздалегідь не відома його довжина, а значить не відомо, скільки пам'яті треба зарезервувати для цього файлу, по-друге, при такому порядку розміщення неминуче виникає фрагментація, і простір на диску використовується не ефективно, так як окремі ділянки маленького розміру (мінімально 1 блок) можуть залишитися не використовуваними.

Наступний спосіб фізичної організації – розміщення у вигляді пов'язаного списку блоків дискової пам'яті (рис 4.4, б). При такому способі на початку кожного блоку міститься покажчик на наступний блок. У цьому випадку адреса файлу також може бути задана одним числом – номером першого блоку. На відміну від попереднього способу, кожен блок може бути приєднаний в ланцюжок якого-небудь файлу, отже фрагментація відсутня. Файл може змінюватися під час свого існування, нарощуючи число блоків. Недоліком є складність реалізації доступу до довільно заданого місця файлу: для того, щоб прочитати п'ятий по порядку блок файлу, необхідно послідовно прочитати чотири перших блоки, простежуючи ланцюжок номерів блоків. Крім того, при цьому способі кількість даних файлу, що містяться в одному блоці, не дорівнює ступеню двійки (одне слово витрачено на номер наступного блоку), а багато програм читають дані блоками, розмір яких дорівнює степеню двійки.

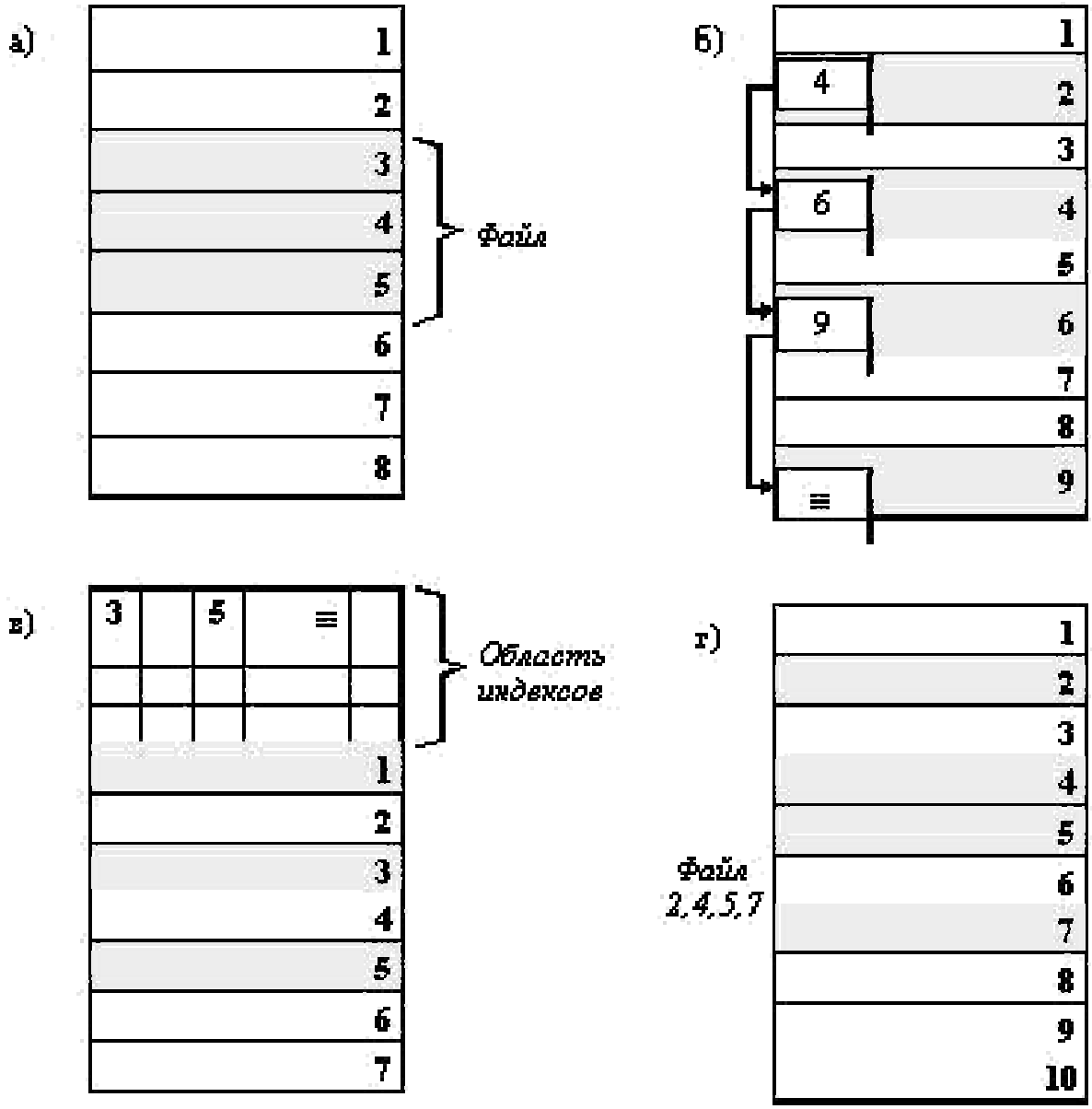

Рис. 4.4. Фізична організація файла
а – неперервне розміщення; б – зв’язаний список блоків;
в – зв’язаний список індексів; г – перелік номерів блоків

Популярним способом, що використовується, наприклад, в файловій системі FAT операційної системи MS-DOS, є використання зв'язаного списку індексів. З кожним блоком зв'язується певний елемент – індекс. Індекси розташовуються в окремій області диска (в MS-DOS це таблиця FAT). Якщо деякий блок розподілений деякому файлу, то індекс цього блоку містить номер наступного блоку даного файлу. При такій фізичній організації зберігаються всі переваги попереднього способу, але знімаються обидва зазначених недоліки: по-перше, для доступу до довільного місця файлу достатньо прочитати тільки блок індексів, відрахувати потрібну кількість блоків файлу по ланцюжку і визначити номер потрібного блоку, і, по-друге, дані файлу займають блок цілком, а значить мають об'єм, рівний степені двійки.

На закінчення розглянемо задання фізичного розташування файлу шляхом простого перерахування номерів блоків, які займає цей файл. ОС UNIX використовує варіант даного способу, що дозволяє забезпечити фіксовану довжину адреси, незалежно від розміру файла. Для зберігання адреси файлу виділено 13 полів. Якщо розмір файлу менший або рівний 10 блокам, то номери цих блоків безпосередньо перераховані в перших десяти полях адреси. Якщо розмір файлу більший 10 блоків, то наступне 11-е поле містить адресу блоку, в якому можуть бути розташовані ще 128 номерів наступних блоків файлу. Якщо файл більший, ніж 10 + 128 блоків, то використовується 12-е поле, в якому знаходиться номер блоку, що містить 128 номерів блоків, які містять по 128 номерів блоків даного файлу. І, нарешті, якщо файл більший 10 + 128 + 128, то використовується останнє 13-е поле для потрійний непрямої адресації, що дозволяє задати адресу файлу, що має розмір максимум 10+ 128 + 128.

Права доступу до файлу. Визначити права доступу до файлу – значить визначити для кожного користувача набір операцій, які він може застосувати до даного файлу. У різних файлових системах може бути визначений свій список диференційовних операцій доступу. Цей список може включати наступні операції:

- створення файлу;
- знищення файлу;
- відкриття файлу;
- закриття файлу;
- читання файлу;
- запис у файл;
- доповнення файлу;
- пошук у файлі;
- отримання атрибутів файлу;
- встановлення нових значень атрибутів;
- перейменування;
- виконання файлу;
- читання каталогу;

та інші операції з файлами та каталогами.

У загальному випадку права доступу можуть бути описані матрицею прав доступу, в якій стовпці відповідають всім файлам системи, рядки - всім користувачам, а на перетині рядків і стовпців вказуються дозволені операції (рис. 4.5). У деяких системах користувачі можуть бути розділені на окремі категорії. Для всіх користувачів однієї категорії визначаються єдині права доступу. Наприклад, у системі UNIX всі користувачі поділяються на три категорії: власника файлу, членів його групи і всіх інших.

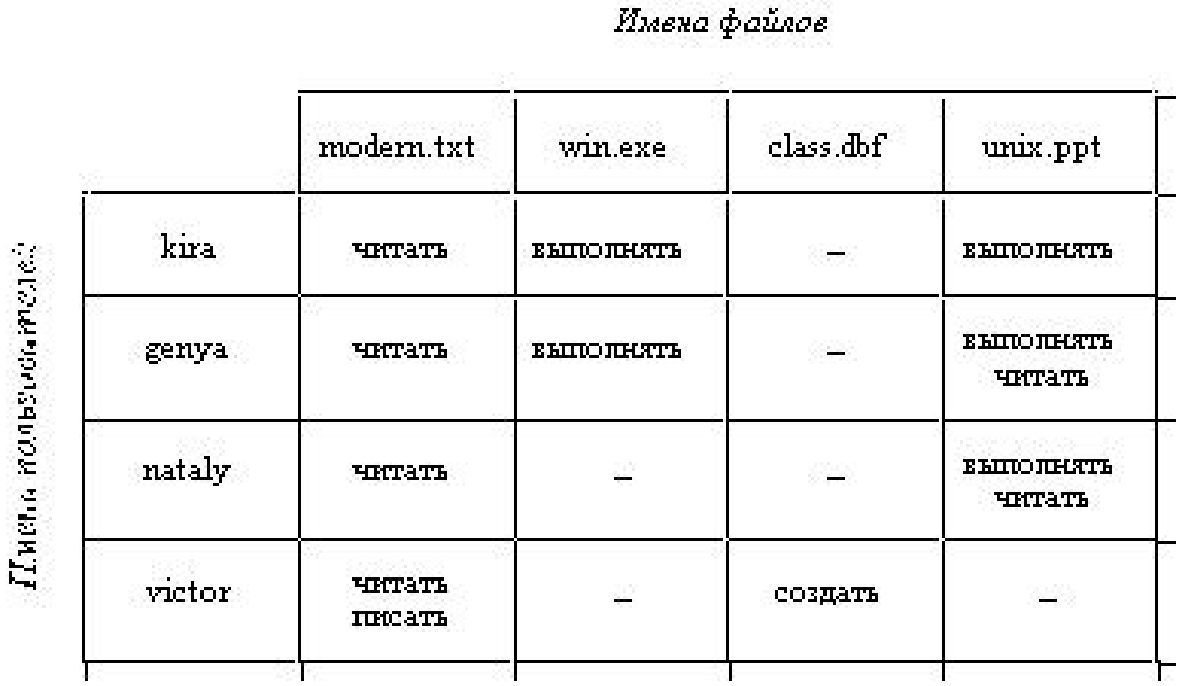

Рис. 4.5. Матрица прав доступу

Розрізняють два основних підходи до визначення прав доступу:

• виборчий доступ, коли для кожного файлу і кожного користувача сам власник може визначити припустимі операції;

• мандатний підхід, коли система наділяє користувача певними правами по відношенню до кожного ресурсу, що розділяється (в даному випадку файлу) залежно від того, до якої групи користувач він віднесений.

Кешування диска. У деяких файлових системах запити до зовнішніх пристроїв, в яких адресація здійснюється блоками (диски, стрічки), перехоплюються проміжним програмним шаром-підсистемою буферизації. Підсистема буферизації є буферним пулом, розташованим в оперативній пам'яті, і комплексом програм, що керують цим пулом. Кожен буфер пулу має розмір, рівний одному блоку. При надходженні запиту на читання деякого блоку підсистема буферизації переглядає свій буферний пул і, якщо знаходить необхідний блок, то копіює його в буфер запитуючого процесу. Операція введення-виведення вважається виконаною, хоча фізичного обміну з пристроєм не відбувалося. Очевидний виграш у часі доступу до файлу. Якщо ж потрібний блок в буферному пулі відсутній, то він зчитується з пристрою і одночасно з передачею запитувачу процесу копіюється в один з буферів підсистеми буферизації. При відсутності вільного буфера на диск витісняється найменш використовувана інформація. Таким чином, підсистема буферизації працює за принципом кеш-пам'яті.

Загальна модель файлової системи. Функціонування будь-якої файлової системи можна представити багаторівневою моделлю (рис. 4.6), в якій кожен рівень надає деякий інтерфейс (набір функцій) вищому рівневі, сам, у свою чергу, для

виконання своєї роботи використовує інтерфейс (звертається з набором запитів) нижчого рівня.

Завданням символьного рівня є визначення за символьним іменем файлу його унікального імені. У файлових системах, в яких кожен файл може мати тільки одне символьне ім'я (наприклад, MS-DOS), цей рівень відсутній, оскільки символьне ім'я, присвоєне файлу користувачем, є одночасно унікальним і може бути використано операційною системою. В інших файлових системах, в яких один і той же файл може мати кілька символьних імен, на даному рівні проглядається ланцюжок каталогів для визначення унікального імені файлу. У файловій системі UNIX, наприклад, унікальним ім'ям є номер індексного дескриптора файлу (i-node).

Рис. 4.6. Загальна модель файлової системи

На наступному, базовому рівні за унікальним іменем файлу визначаються його характеристики: права доступу, адреса, розмір та інші. Як вже було сказано, характеристики файлу можуть входити до складу каталогу або зберігатися в окремих таблицях. Під час відкриття файлу його характеристики переміщаються з диска в оперативну пам'ять, щоб зменшити середній час доступу до файлу. У деяких файлових системах (наприклад, HPFS) під час відкриття файлу разом з його характеристиками оперативну пам'ять переміщаються кілька перших блоків файлу, що містять дані.

Наступним етапом реалізації запиту до файлу є перевірка прав доступу до нього. Для цього порівнюються повноваження користувача або процесу, які видали запит, зі списком дозволених видів доступу до даного файлу. Якщо запитуваний вид доступу дозволений, то виконання запиту продовжується, якщо ні, то видається повідомлення про порушення прав доступу.

На логічному рівні визначаються координати запитуваного логічного запису у файлі, тобто потрібно визначити, на якій відстані (в байтах) від початку файлу знаходиться необхідний логічний запис. При цьому абстрагуються від фізичного розташування файлу, він представляється у вигляді безперервної послідовності байтів. Алгоритм роботи даного рівня залежить від логічної організації файлу. Наприклад, якщо файл організований як послідовність логічних записів фіксованої довжини l, то n-ий логічний запис має зсув l ((n-1) байт. Для визначення координат логічного запису у файлі з індексного-послідовною організацією виконується читання таблиці індексів (ключів), в якій безпосередньо вказується адреса логічного запису.

На фізичному рівні файлова система визначає номер фізичного блоку, який містить необхідний логічний запис, і зсув логічного запису у фізичному блоці. Для вирішення цього завдання використовуються результати роботи логічного рівня – зміщення логічного запису у файлі, адреса файлу на зовнішньому пристрої, а також відомості про фізичну організації файлу, включаючи розмір блоку. На рис. 4.7 проілюстровано роботу фізичного рівня для найпростішої фізичної організації файлу у вигляді безперервної послідовності блоків. Підкреслимо, що завдання фізичного рівня вирішується незалежно від того, як був логічно організований файл.

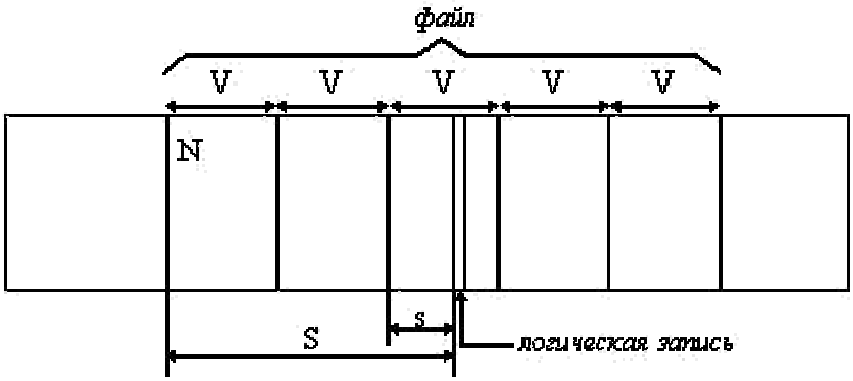

Рис. 4.7. Функції фізичного рівня файлової системи

Вихідні дані:
V - розмір блоку;
N - номер першого блоку файлу;
S - зміщення логічного запису у файлі.
Потрібно визначити на фізичному рівні: n - номер блоку, що містить необхідний логічний запис; s - зсув логічного запису в межах блоку;

n = N + [S / V], де [S / V] - ціла частина числа S / V; s = R
[S / V] - дробова частина числа S / V.

Після визначення номера фізичного блоку, файлова система звертається до системи введення-виведення для виконання операції обміну з зовнішнім пристроєм. У відповідь на цей запит в буфер файлової системи буде переданий потрібний блок, в якому на підставі отриманого при роботі фізичного рівня зміщення вибирається необхідний логічний запис.

Відображені в пам'ять файли. У порівнянні з доступом до пам'яті, традиційний доступ до файлів виглядає заплутаним і незручним. З цієї причини деякі ОС, починаючи з MULTICS, забезпечують відображення файлів в адресний простір виконуваного процесу. Це виражається в появі двох нових системних викликів: MAP (відобразити) і UNMAP (скасувати відображення). Перший виклик передає операційній системі в якості параметрів ім'я файлу та віртуальну адресу, і операційна система відображає вказаний файл у віртуальний адресний простір за вказаною адресою.

Припустимо, наприклад, що файл f має довжину 64 К і відображається на область віртуального адресного простору з початковою адресою 512 К. Після цього будь-яка машинна команда, яка читає вміст байта за адресою 512 К, отримує 0-ий байт цього файлу і т.д . Очевидно, що запис за адресою 512 К + 1100 змінює 1100 байт файлу. При завершенні процесу на диску залишається модифікована версія файлу, так ніби він був змінений комбінацією викликів SEEK і WRITE.

Насправді при відображенні файлу внутрішні системні таблиці змінюються так, щоб даний файл служив сховищем сторінок віртуальної пам'яті на диску. Таким чином, читання за адресою 512 К викликає сторінкову відмову, в результаті чого сторінка 0 переноситься у фізичну пам'ять. Аналогічно, запис за адресою 512 К + 1100 викликає сторінковий відмову, в результаті якої сторінка, що містить цю адресу, переміщується в пам'ять, після чого здійснюється запис в пам'ять за потрібною адресою. Якщо ця сторінка витісняється з пам'яті алгоритмом заміни сторінок, то вона записується назад у файл у відповідне його місце. При завершенні процесу всі відображені і модифіковані сторінки переписуються з пам'яті в файл.

Відображення файлів найкраще працює в системі, яка підтримує сегментацію. У такій системі кожен файл може бути відображений в свій власний сегмент, так що k-ий байт у файлі є k-им байтом сегмента. На рис. 4.8, а зображено процес, який має два сегменти - коду і даних. Припустимо, що цей процес копіює файли. Для цього він спочатку відображає файл-джерело, наприклад, abc. Потім він створює порожній сегмент і відображає на нього файл призначення, наприклад, файл ddd.

З цього моменту процес може копіювати сегмент-джерело в сегмент-приймач за допомогою звичайного програмного циклу, що використовує
команди пересилання в пам'яті типу mov. Ніякі виклики READ або WRITE не потрібні. Після виконання копіювання процес може виконати виклик UNMAP для видалення файлу з адресного простору, а потім завершитися. Вихідний файл ddd буде існувати на диску, як якби він був створений звичайним способом.

Хоча відображення файлів виключає потребу у виконанні введення-виведення і тим самим полегшує програмування, цей спосіб породжує і деякі нові проблеми. По-перше, для системи складно дізнатися точну довжину вихідного файлу, в даному прикладі ddd. Простіше вказати найбільший номер записаної сторінки, але немає способу дізнатися, скільки байт в цій сторінці було записано. Припустимо, що програма використовує тільки сторінку номер 0, і після виконання всі байти все ще встановлені в значення 0 (їх початкове значення). Може бути, що файл складається з 10 нулів. А може

бути, він складається з 100 нулів. Як це визначити? Операційна система не може це повідомити. Все, що вона може зробити, так це створити файл, довжина якого дорівнює розміру сторінки.

Друга проблема проявляється (потенційно), якщо один процес відображає файл, а інший процес відкриває його для звичайного файлового доступу. Якщо перший процес змінює сторінку, то ця зміна не буде відображена у файлі на диску доти, поки сторінка не буде витіснена на диск. Підтримання узгодженості даних файлу для цих двох процесів вимагає від системи великих турбот.

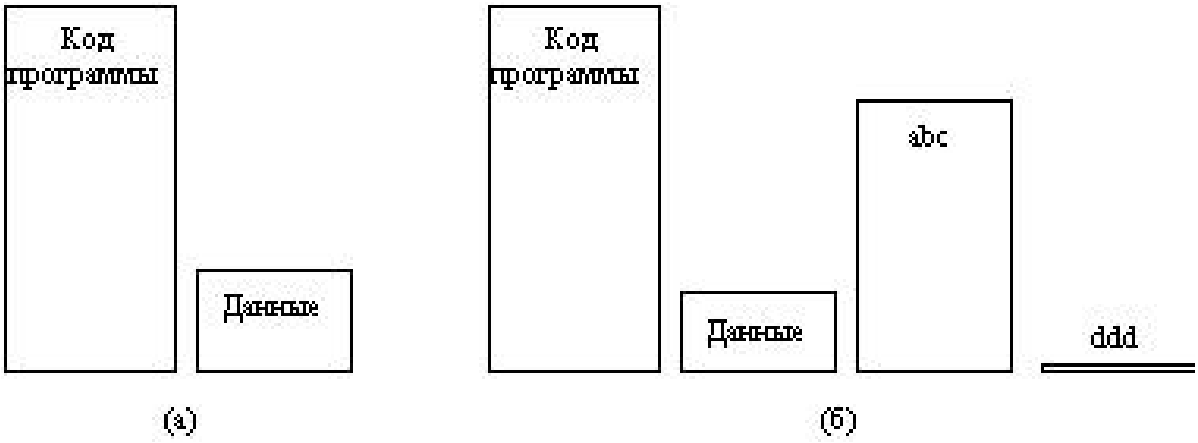

Рис. 4.8. (а) Сегменти процесу перед відображенням файлів в адресний простір; (б) Процес після відображення існуючого файла abc в один сегмент і створення нового сегмента для файла ddd

Третя проблема полягає в тому, що файл може бути більшим, ніж сегмент, і навіть більшим, ніж весь віртуальний адресний простір. Єдиний спосіб її вирішення полягає в реалізації виклику MAP таким чином, щоб він міг відображати не весь файл, а його частину. Хоча така робота, очевидно, менш зручна, ніж відображення цілого файлу.

Сучасні архітектури файлових систем. Розробники нових операційних систем прагнуть забезпечити користувача можливістю працювати відразу з кількома файловими системами. У новому розумінні файлова система складається з багатьох складових, в число яких входять і файлові системи в традиційному розумінні.

Нова файлова система має багаторівневу структуру (рис. 4.9), на верхньому рівні якої розташовується так званий перемикач файлових систем (в Windows, наприклад, такий перемикач називається встановлюваним диспетчером файлової системи - installable filesystem manager, IFS). Він забезпечує інтерфейс між запитами програми та конкретною файловою системою, до якої звертається цей додаток. Перемикач файлових систем перетворює запити у формат, що сприймається наступним рівнем - рівнем файлових систем.

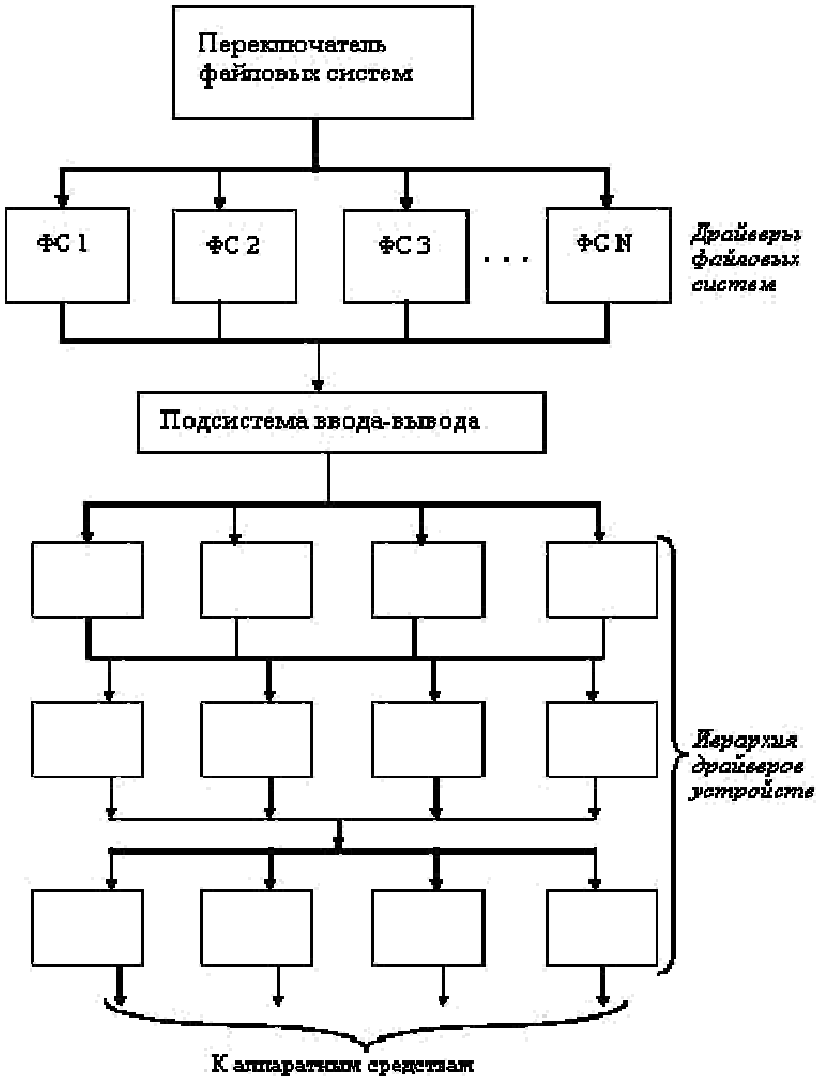

Рис. 4.9. Архітектура сучасної файлової системи

Кожен компонент рівня файлових систем виконаний у вигляді драйвера відповідної файлової системи і підтримує певну організацію файлової системи. Перемикач є єдиним модулем, який може звертатися до драйвера файлової системи. Додаток не може звертатися до нього безпосередньо. Драйвер файлової системи може бути написаний у вигляді реєнтерабельного коду, що дозволяє відразу декільком додаткам виконувати операції з файлами. Кожен драйвер файлової системи в процесі власної ініціалізації реєструється у перемикача, передаючи йому таблицю точок входу, які будуть використовуватися при наступних зверненнях до файлової системи.

Для виконання своїх функцій драйвери файлових систем звертаються до підсистеми введення-виведення, що утворює наступний шар файлової системи нової архітектури. Підсистема введення виведення - це складова частина файлової системи, яка відповідає за завантаження, ініціалізацію і управління всіма модулями нижчих рівнів файлової системи. Зазвичай ці модулі є драйвери портів, які безпосередньо займаються роботою з апаратними засобами. Крім цього підсистема введення-виведення забезпечує деякий сервіс драйверам файлової системи, що дозволяє їм здійснювати запити до конкретних пристроїв. Підсистема введення-виведення повинна постійно бути присутньою в пам'яті і організовувати спільну

роботу ієрархії драйверів пристроїв. У цю ієрархію можуть входити драйвери пристроїв певного типу (драйвери жорстких дисків або накопичувачів на стрічках), драйвери, підтримувані постачальниками (такі драйвери перехоплюють запити до блокових пристроїв і можуть частково змінити поведінку існуючого драйвера цього пристрою, наприклад, зашифрувати дані), драйвери портів, які управляють конкретними адаптерами.

Велике число рівнів архітектури файлової системи забезпечує авторам драйверів пристроїв велику гнучкість - драйвер може отримати управління на будь-якому етапі виконання запиту - від виклику додатком функції, яка займається роботою з файлами, до того моменту, коли працюючий на найнижчому рівні драйвер пристрою починає переглядати регістри контролера . Багаторівневий механізм роботи файлової системи реалізований за допомогою ланцюжків виклику.

У ході ініціалізації драйвер пристрою може додати себе до ланцюжка виклику деякого пристрою, визначивши при цьому рівень подальшого звернення. Підсистема введення-виведення поміщає адресу цільової функції в ланцюжок виклику пристрою, використовуючи заданий рівень для того, щоб належним чином впорядкувати ланцюжок. У міру виконання запиту, підсистема введення-виведення послідовно викликає всі функції, раніше поміщені в ланцюжок виклику.

Внесена до ланцюжка виклику процедура драйвера може вирішити передати запит далі - в зміненому або в незміненому вигляді - на наступний рівень, або, якщо це можливо, процедура може задовольнити запит, не передаючи його далі по ланцюжку.

Організація файлової системи. Файл - це поняття, звичне будь-якому користувачеві комп'ютера. Для користувача кожен файл - це окремий предмет, у якого є початок і кінець і який відрізняється від всіх інших файлів ім'ям і розташуванням («як називається» і «де лежить»). Як і будь-який предмет, файл можна створити, перемістити і знищити, однак без зовнішнього втручання він буде зберігатися незмінним невизначено довгий час. Файл призначений для зберігання даних будь-якого типу - текстових, графічних, звукових, виконуваних програм і багато чого іншого. Аналогія файлу з предметом дозволяє користувачеві швидко освоїтися під час роботи з даними в операційній системі.

Для операційної системи Linux файл - не менш важливе поняття, ніж для її користувача: всі дані, що зберігаються на будь-яких носіях, обов'язково знаходяться усередині якого-небудь файлу, в іншому випадку вони просто недоступні ні для операційної системи, ні для її користувачів. Більш того, всі пристрої, підключені до комп'ютера (починаючи з клавіатури і закінчуючи будь-якими зовнішніми пристроями, наприклад, принтерами та сканерами) Linux представляє як файли (так звані файли-дірки). Звичайно, файл, що містить звичайні дані, сильно відрізняється від файла, призначеного для поводження з пристроєм, тому в Linux визначені декілька різних типів файлів. В основному користувач має справу з файлами трьох типів: звичайними файлами, призначеними для зберігання даних, каталогами та файлами-посиланнями, саме про них і піде мова.

Файл – окрема область даних на одному з носіїв інформації, у якої є власне ім'я.

Система файлів: каталоги. Файлова система з точки зору користувача - це «простір», в якому розміщуються файли, наявність файлової системи дозволяє визначити не тільки «як називається файл», а й «де він знаходиться». Розрізняти файли тільки за іменем було б занадто неефективним: про кожен файл доводилося б пам'ятати, як він називається і при цьому піклуватися про те, щоб імена ніколи не повторювалися. Більш того, необхідний механізм, що дозволяє працювати з групами тематично пов'язаних між собою файлів (наприклад, компонентів однієї і тієї ж програми або різних розділів однієї роботи). Інакше кажучи, файли потрібно систематизувати.

Файлова система – спосіб зберігання та організації доступу до даних на інформаційному носії або його розділі. Класична файлова система має ієрархічну структуру, в якій файл однозначно визначається повним шляхом до нього.

Linux може працювати з різними типами файлових систем, які розрізняються списком підтримуваних можливостей, продуктивністю в різних ситуаціях, надійністю та іншими ознаками.

Більшість сучасних файлових систем (але не всі!) Використовують в якості основного організаційного принципу каталоги. Каталог - це список посилань на файли або інші каталоги. Прийнято говорити, що каталог містить в собі файли або інші каталоги, хоча насправді він тільки посилається на них, фізичне розміщення даних на диску зазвичай ніяк не пов'язане з розміщенням каталогу. Каталог, на який є посилання в даному каталозі, називається підкаталогом або вкладеним каталогом. Каталог в файловій системі більш за все нагадує бібліотечний каталог, що містить посилання на об'єднані за якимись ознаками книги та інші розділи каталогу (файли і підкаталоги). Посилання на один і той же файл може міститися в кількох каталогах одночасно, це може зробити доступ до файлу більш зручним. В файлової системи Ext2 кожен каталог - це окремий файл особливого типу («d», від англ. «Directory»), що відрізняється від звичайного файла з даними: у ньому можуть міститися тільки посилання на інші файли і каталоги.

В файловій системі Linux немає папок і документів. Є каталоги і файли, можливості яких значно ширші.

Досить часто замість терміну каталог можна зустріти папка (англ. Folder). Цей термін добре вписується в уявлення про файли як про предмети, які можна розкладати по папках, однак частина можливостей файлової системи, яка суперечить цим поданням, таким чином затемнюється. Зокрема, з терміном «папка» погано узгоджується те, що посилання на файл може бути присутнім одночасно в декількох каталогах, файл може бути посиланням на інший файл тощо. В Linux ці можливості файлової системи вельми важливі для ефективної роботи, тому будемо усюди використовувати більш відповідний термін «каталог».

В файловій системі, організованій за допомогою каталогів, на будь-який файл має бути посилання як мінімум з одного каталогу, в іншому випадку файл просто не буде доступний всередині цієї файлової системи, інакше кажучи, не існуватиме.

Імена файлів і каталогів. Головні відмінні ознаки файлів і каталогів - їх імена. В Linux імена файлів і каталогів можуть бути довжиною не більше 256 символів, і можуть містити будь-які символи, крім «/». Причина цього обмеження очевидна: цей символ використовується як роздільник імен у складі шляху, тому не повинен зустрічатися в самих іменах. Причому Linux завжди розрізняє великі та

малі літери в іменах файлів і каталогів, тому «methody», «Methody» і «METHODY» будуть трьома різними іменами.

Є кілька символів, допустимих в іменах файлів і каталогів, які, при цьому, потрібно використовувати з обережністю. Це - так звані спецсимволи

«*», «\», «&», «<», «>», «;», «(», «)», «|», а також пропуски і табуляції.

Справа в тому, що ці символи мають особливе значення для будь командної оболонки, тому потрібно буде спеціально подбати про те, щоб командна оболонка сприймала ці символи як частину імені файлу або каталогу.

Кодування і кириличні імена. Як можна було помітити, поки у всіх прикладах імен файлів і каталогів вживалися тільки символи латинського алфавіту і деякі розділові знаки. Це не випадково і викликано бажанням забезпечити однаковий вигляд на будь-яких системах. В Linux в іменах файлів і каталогів можна використовувати будь-які символи будь-якої мови, проте така свобода вимагає жертв.

Справа в тому, що з давніх пір кожен символ (буква) кожної мови традиційно представлявся у вигляді одного байта. Таке уявлення накладає дуже жорсткі обмеження на кількість букв в алфавіті: їх може бути не більше 256, а за винятком керуючих символів, цифр, розділових знаків та іншого - і того менше. Обширні алфавіти (наприклад, ієрогліфічні японський і китайський) довелося замінювати спрощеним їх поданням. До того ж, перші 128 символів з цих 256 краще завжди залишати незмінними, відповідними стандарту ASCII, що включає латиницю, цифри, розділові знаки і найбільш популярні символи з тих, що зустрічаються на клавіатурі друкарської машинки. Інтерпретація інших 128 символів залежить від того, яке кодування встановлено в системі. Наприклад, в російському кодуванні KOI8-R 228-й символ такої таблиці відповідає букві «Д», а в західноєвропейському кодуванні ISO-8859-1 цей же символ відповідає букві «а».

Імена файлів, записані на диск в одному кодуванні, виглядають безглуздо, якщо при перегляді каталогу було встановлено інше. Гірше того. Більшість кодувань заповнюють діапазон символів з номерами від 128 до 255 не повністю, тому відповідного символу може взагалі не бути! Це означає, що ввести таке спотворене ім'я файлу з клавіатури (наприклад, для того, щоб його перейменувати) напряму не вдасться, доведеться застосовувати різні хитрощі. Нарешті, багато мов, у тому числі і російська, історично мають кілька кодувань. На жаль, в даний час немає стандартного способу вказувати кодування прямо в імені файлу, тому в рамках однієї файлової системи варто дотримуватися єдиного кодування при іменуванні файлів.

Існує універсальне кодування, що включає символи всіх письменностей світу - UNICODE. Стандарт UNICODE в даний час набуває все більшого поширення і претендує на статус спільного для всіх текстів, що зберігаються в електронній формі. Однак поки він не досяг бажаної універсальності, особливо в області імен файлів. Один символ в UNICODE може займати більше одного байта - і в цьому головний його недолік, оскільки безліч корисних прикладних програм, відмінно працюють з однобайтними кодуваннями, необхідно грунтовно або навіть повністю переробляти для того, щоб навчити їх поводитися з UNICODE. Можливо, причина недостатньої поширеності цього кодування також і в тому, що UNICODE - дуже

громіздкий стандарт, і він може виявитися неефективним при роботі з файловою системою, де швидкість і надійність обробки - дуже істотні якості.

Це не означає, що називаючи файли, не слід використовувати мови, відмінні від англійської. Поки точно відомо, в якому кодуванні задано ім'я файлу - проблем не виникне.

Розширення. Багатьом користувачам знайоме поняття розширення - частина імені файлу після точки, зазвичай обмежується декількома смволами і вказує на тип даних, що містяться у файлі. В файлової системи Linux немає ніяких розпоряджень з приводу розширення: в імені файлу може бути будь-яка кількість крапок (в тому числі і жодної), а після останньої крапки може бути будь-яка кількість символів. Хоча розширення не обов'язкові і не нав'язуються технологією в Linux, вони широко використовуються: розширення дозволяє людині або програмі, не відкриваючи файл, тільки по його імені визначити, якого типу дані в ньому містяться. Однак потрібно враховувати, що розширення - це тільки набір угод по найменуванню файлів різних типів. Строго кажучи, дані у файлі можуть не відповідати заявленому розширенню з тієї або іншоїх причини, тому цілком покладатися на розширення просто не можна.

Визначити тип вмісту файлу можна і на підставі самих даних. Багато форматів передбачають вказівку на початку файлу, як слід інтерпретувати подальшу інформацію: як програму, вихідні дані для текстового редактора, сторінку HTML, звуковий файл, зображення або щось інше. У розпорядженні користувача Linux завжди є утиліта file, яка призначена саме для визначення типу даних, що містяться у файлі.

Дерево каталогів. Поняття каталогу дозволяє систематизувати всі об'єкти, розміщені на носії даних (наприклад, на диску). У більшості сучасних файлових систем використовується ієрархічна модель організації даних: існує один каталог, який об'єднує всі дані в файлової системі - це «корінь» всієї файлової системи, кореневої каталог. Кореневий каталог може містити будь-які об'єкти файлової системи, і зокрема, підкаталоги (каталоги першого рівня вкладеності). Ті, в свою чергу, також можуть містити будь-які об'єкти файлової системи і підкаталоги (другого рівня вкладеності) і т. Д. Таким чином все, що записано на диску - файли, каталоги і спеціальні файли - обов'язково «належить» кореневого каталогу: або безпосередньо (міститься в ньому), або на деякому рівні вкладеності.

Ієрархію вкладених один в одного каталогів можна співвіднести з ієрархією даних у системі: об'єднати тематично пов'язані файли в каталог, тематично пов'язані каталоги - в один загальний каталог і т. д. Якщо строго слідувати ієрархічним принципам, то чим глибшим буде рівень вкладеності каталогу, тим більше індивідуальною ознакою повинні бути об'єднані дані , що містяться в ньому. Якщо цим принципом не слідувати, то незабаром виявиться набагато простіше складати всі файли в один каталог і шукати потрібний серед них, ніж проробляти такий пошук по всіх підкаталогам системи. Однак у цьому випадку про яку б то не було систематизацію файлів говорити не доводиться.

Структуру файлової системи можна представити наочно у вигляді дерева, «коренем» якого є кореневий каталог, а у вершинах розташовані всі інші каталоги.

На рис. 4.10 зображено дерево каталогів, курсивом позначені імена файлів, прямим накресленням - імена каталогів.

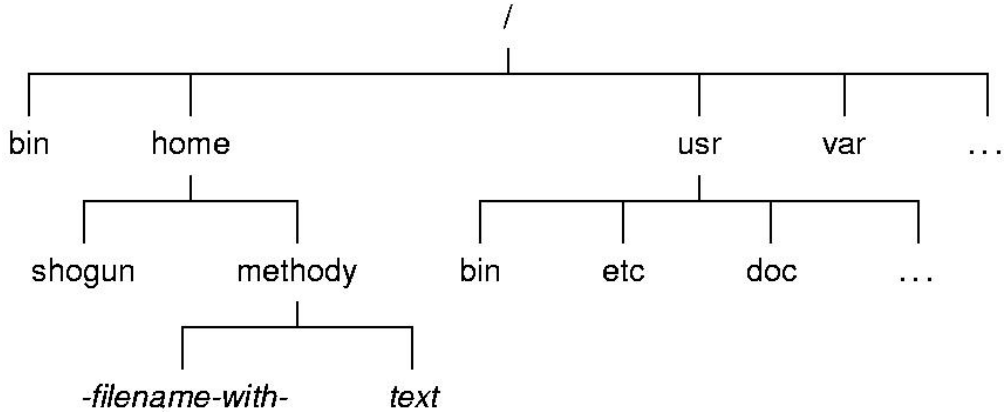

Рис. 4.10. Дерево каталогів в Linux

У будь-якої файлової системи Linux завжди є тільки один кореневий каталог, який називається «/». Користувач Linux завжди працює з єдиним деревом каталогів, навіть якщо різні дані розташовані на різних носіях: кількох жорстких або мережевих дисках, знімних дисках, CD-ROM і т. п. Для того, щоб підключати та відключати файлові системи на різних пристроях в одне загальне дерево, використовуються процедури монтування та размонтування. Після того, як файлові системи на різних носіях підключені до загального дерева, що містяться на них дані доступні так, як якби всі вони складали єдину файлову систему: користувач може навіть не знати, на якому пристрої які файли зберігаються. Положення будь-якого каталогу в дереві каталогів точно і однозначно описується за допомогою повного шляху. Повний шлях завжди починається від кореневого каталогу і складається з перерахування всіх вершин, які зустрілися під час руху по ребрах дерева до шуканого каталогу включно. Назви сусідніх вершин розділяються символом «/» («слеш»). В Linux повний шлях, наприклад, до каталогу «methody» в файловій системі, наведеної на рис. 4.10 записується таким чином: спочатку символ «/», що позначає кореневий каталог, потім до нього додається «home», потім роздільник «/», за яким слідує назва шуканого каталогу «methody», в результаті виходить повний шлях «/ home / methody ».

Розташування файлу у файловій системі аналогічним чином визначається за допомогою повного шляху, тільки останнім елементом в даному випадку буде не назву каталогу, а назва файлу. Наприклад, повний шлях до створеного файлу «-filename-with-» виглядатиме так: «/ home / methody / -filename-with-».

Організація каталогів файлової системи у вигляді дерева не допускає появи циклів: тобто каталог не може містити в собі каталог, в якому міститься сам. Завдяки цьому обмеженню повний шлях до будь-якого каталогу чи файлу у файловій системі завжди буде кінцевим.

Рекомендації стандарту з розміщення файлів і каталогів грунтуються на принципі розносити в різні підкаталоги файли, які по-різному використовуються в системі. За типом використання файлів їх можна розділити на наступні групи:

– *файли користувача / системні файли;*

Файли користувача - це всі файли, створені користувачем, які не належать жодному з компонентів системи.

– *змінювані / незмінні файли;*

До незмінених файлів належать усі статичні компоненти програмного забезпечення: бібліотеки, виконувані файли та ін. - все, що не змінюється саме без втручання системного адміністратора. Змінювані - це ті, які змінюються без втручання людини в процесі роботи системи: системні журнали, черги друку та ін. Виділення незмінних файлів в окрему структуру (наприклад, / usr) дозволяє використовувати відповідну частину файлової системи в режимі "тільки читання", що зменшує ймовірність випадкового пошкодження даних і дозволяє використовувати для зберігання цієї частини файлової системи CD-ROM та інші носії, доступні тільки для читання.

– *колективні / неподілювані файли;*

Це розмежування стає корисним, якщо мова йде про мережі, в якій працює кілька комп'ютерів. Значна частина інформації при цьому може зберігатися на одному з комп'ютерів і використовуватися усіма іншими по мережі (до такої інформації відносяться, наприклад, багато програм і домашні каталоги користувачів). Однак частину файлів не можна розділяти між системами (наприклад, файли для початкового завантаження системи).

Контрольні запитання

1. Що таке файлова система?
2. З чого складається файлова система?
3. Яким чином ідентифікуються файли?
4. В чому проблема переходу до довгих імен файлів?
5. За якими типами поділяють файли?
6. Які особливості спеціальних файлів?
7. Що таке каталог? Чим він відрізняється від папки?
8. Що таке атрибути файлів?
9. Якою може бути ієрархія каталогів?
10. Що таке логічний запис?
11. Що таке логічна організація файлу?
12. Що таке фізична організація файлу?
13. Які типи фізичної організації файлів використовуються у файлових системах?
14. Які права доступу до файлів підтримують сучасні операційні системи?
15. Що таке кешування диска і для чого воно використовується?
16. Яким чином можна представити загальну модель файлової системи?
17. Назвіть основні рівні моделі файлової системи і їх призначення.
18. Для чого виконується відображення файлів у пам’ять?
19. Опишіть сучасні архітектури файлових систем.
20. Охарактеризуйте принципи організації файлових систем.