Міністерство освіти і науки України
Донбаська державна машинобудівна академія

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни

«СИСТЕМНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ»

(для студентів спеціальності 123“Комп’ютерна інженерія»)

Освітній рівень - бакалавр

Краматорськ 2020

ТЕМА 1.
КЛАСИФІКАЦІЯ СИСТЕМНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

1.1 Загальні відомості про розподіл програмного забезпечення на прикладне та системне.

Програмне забезпечення(ПЗ) являє собою сукупність програм, призначених для розв'язання завдань на комп'ютері. Програма – це впорядкований набір команд. Програмне та апаратне забезпечення працюють взаємопов'язано і в неперервній взаємодії. Будь-який апаратний пристрій управляється програмно.

Програмне забезпечення можна поділити на три класи: системне, прикладне та інструментальне (Рис.1.1). Наведена класифікація є досить умовною. Інтеграція програмного забезпечення призвела до того, що практично будь-яка програма має риси кожного класу.

Рис. 1.1 Класифікація програмного забезпечення

Системне ПЗ призначено для управління роботою комп'ютера, розподілу його ресурсів, підтримки діалогу з користувачами, надання їм допомоги в обслуговуванні комп'ютера, а також для часткової автоматизації розробки нових програм. Системне ПЗ — це комплекс програм, багато з яких постачаються разом з комп'ютером та документацією до неї. Системне програмне забезпечення здійснює управління роботою обчислювальної

системи. Як правило, системні програми забезпечують взаємодію інших програм з апаратними складовими, організацію інтерфейсу користувача.

Основні функції системного ПЗ:

Підтримка ефективної роботи будь-якої обчислювальної системи.

- Розгортання на комп'ютері або в мережевому оточенні середовища для роботи прикладного програмного забезпечення.
- Виконання фонових процесів роботи з файловою системою, захисту даних від витоку, перевірка на наявність шкідливих скриптів і вірусів.
- Здійснення діагностики та запобігання виходу з ладу апаратної частини персонального комп'ютера, ноутбука та іншого цифрового пристрою.
- Взаємозв'язок фізичних пристроїв і перетворення їх в логічні.

У першому випадку застосовуються спеціальні утиліти. Вони можуть входити до складу самої операційної системи або встановлюватися з інших джерел.

У другому випадку розгортання здійснюється за допомогою операційної системи, тобто програми-оболонки в якій може працювати будь-яке програмне забезпечення (ПЗ).

Третя функція здійснює роботу над елементами файлової системи, тобто каталогами і файлами. Їх можна переміщати в інші місця, копіювати, видаляти, змінювати і т. д. Крім того, існує певна група, звана архіваторами. Останні дозволяють значно зменшувати розміри файлів, готувати будь-яку одиницю даних (картинок, документів і т.д.) до розсилки в інтернеті.

Четверта функція забезпечує захист важливої інформації від зловмисників, які прагнуть отримати паролі від електронної пошти, платіжних систем та інших важливих даних для конкретного користувача.

Прикладне програмне забезпечення призначене для розв'язання прикладних завдань фахової діяльності людини (тобто, прикладене до практики). Спектр таких програм надзвичайно широкий: від виробничих та наукових до навчальних та розважальних. Сюди відносять розрахункові, навчаючі, моделюючі програми, комп'ютерні ігри, тощо. У структурі прикладного програмного забезпечення можна виділити прикладні програми:

- Загального призначення: це комплекс програм, який одержав широке використання серед різних категорій користувачів. Найбільш відомими серед них є:
  - текстові редактори дозволяють готувати текстові документи: технічні опи-си, службові листи, статті та ін. Найбільш відомі такі текстові редактори: Notepad++, WordPad, Word.
  - графічні системи багаточисельні, а їх функції — різноманітні. Серед них можна виділити системи:
  - ділової графіки (Microsoft PowerPoint, Lotus Freelance Graphics)
  - художньої графіки, які ще називають просто графічними редакто-рами (Paintbrush)

- інженерної графіки та автоматизованого проектування (Autodesk AutoCad)
- системи обробки фотографічних зображень (Adobe Photoshop)
- універсальні графічні системи (CorelDRAW).
- Електронні таблиці. Програми роботи з електронними таблицями (ЕТ) дозволяють розв'язувати широке коло задач, зв'язаних з числовими розрахунками. Найширше використ-товують серед програм такого класу Supercalk, Microsoft Excel та Lotus 1-2-3.
- Системи управління базами даних. Системи управління базами даних призначені для об'єднання наборів даних з метою створення єдиної інформаційної моделі об'єкта. Ці програми дозволяють накопичувати, обновляти, коригувати, вилучати, сорту-вати інформацію, організовану спеціальним засобом у вигляді банку даних. Найпоширеніші СУБД: dBase III Plus, FoxBase+, Oracle, MS Acces, FoxPro, Paradox, MySQL.
- Крім перерахованих систем до складу прикладного ПЗ загального призна-чення слід віднести й інтегровані системи. Ці системи об'єднують у собі можливості текстових редакторів, графічних систем, електронних таблиць та систем управління базами даних. Головна перевага інтегрованих систем перед окремими системами прикладного ПЗ загального призначення полягає у тому, що вони створюють єдині правила роботи для користувача, тобто вони мають єдиний інтерфейс як при роботі з текстом, так і при роботі з електронними таблицями та ін. Найвідоміші серед них: Microsoft Works, Microsoft Office, Lotus SmartSuite, Perfect Office.

▪ Спеціального призначення. Прикладні програми спеціального призначення використовують у специфічній діяльності користувачів, функції специфічних систем залежать від їх призначення. Наприклад:
- Систем навчального призначення. Це можуть бути інструментальні засоби для розробки комп'ютерних уроків (гіпермедійні та гіпертекстові системи, авторсь-кі та інші системи), імітаційне моделюючі програми навчального призначення, програми для розробки та підтримки шкільного розкладу, педагогічні про- грамні засоби різного призначення та ін.
- Пакети прикладних програм (ППП), які широко використовуються, наприклад, для статистичної обробки даних, бухгалтерського обліку, розра-хунку будівельних конструкцій та ін. Наявність у комп'ютері різноманітних ППП дозволяє розв'язувати значну частину простих прикладних задач, майже без програмування. В цьому випадку завдання на розв'язування тієї чи іншої задачі записується у вигляді директиви спеціальною проблемно-орієнтованою мовою та повідомляється комп'ютеру.

Інструментальне програмне забезпечення призначене для розробки всіх видів інформаційно-програмного забезпечення. При цьому під інформаційним забезпеченням розуміють сукупність попередньо підготовлених даних, необхідних для роботи програмного забезпечення. Наприклад, будь-яка сучасна програма має вбудовану довідку для роботи з цією програмою. Файл довідки являє собою інформаційне забезпечення. До інструментального програмного забезпечення відносять:

- редактори (текстові, графічні, музичні тощо)
- системи табличної обробки даних (табличні процесори)
- системи управління базами даних
- транслятори мов програмування
- інтегровані системи для виробництва

1.2 Склад системного програмного забезпечення.

До системного програмного забезпечення відносяться:

- операційні системи;
- інтерфейсні оболонки для взаємодії користувача з ОС;
- системи управління файлами;
- системи програмування;
- утиліти.

Розглянемо більш докладно складові системного програмного забезпечення.

Операційні системи є невід'ємною частиною системного програмного забезпечення(СПЗ). Взагалі, сама ОС може бути представлена як комплекс системних програм, які відповідають за отримання, обробку, використання інформації та управління всіма системними пристроями. З одного боку, ОС виступає в ролі «прокладки» між користувачем і його комп'ютером, з іншого боку, відповідає за якомога більш ефективне використання як програмної, так і апаратної начинки останнього. У літературі ОС називається також базове системне програмне забезпечення.

Користувачі здійснюють управління комп'ютером, використовуючи спеціальний компонент ОС - **командний процесор або оболонку системи** (shell). Основною функцією цього модуля є отримання команд для виконання, які вводяться за допомогою клавіатури або допоміжних пристроїв (миша, «трекбол»), та виведення результату виконання введеної команди.

Інтерфейс - засоби та сукупність команд ОС, які допомагають користувачеві опрацьовувати інформацію за допомогою комп'ютера.

У сучасних ОС найбільш поширеними є :

1) інтерфейс командного рядка, який дозволяє користувачеві набирати команди на клавіатурі;

2) графічний інтерфейс користувача забезпечує уведення команд ОС за допомогою виконання дій з візуальним представленням об'єктів ОС; тут може використовуватися і клавіатура, і миша.

Найпершим способом забезпечення управління обчислювальною системою був інтерфейс командного рядка. При цій технології, як єдиний спосіб введення інформації від людини до комп'ютера використовується клавіатура, а комп'ютер виводить інформацію за допомогою монітора.

Оболонка користувача реалізується як окремий модуль (програма) (сімейство операційних систем Unix) або вбудований в ядро (сімейство операційних систем Windows).

Найважливішою функцією СПЗ є **управління файлами**. Файлові менеджери призначені для більш зручного доступу до фізичних даними на диску, для відображення їх в графічному поданні. Таким чином, вони дозволяють використовувати тільки логічне ім'я файлу, а не його безпосередні координати на жорсткому диску. Файлові системи і файлові менеджери самі по собі не існують, так як їх розробляли під конкретні операційні системи і для вирішення певних завдань. Таким чином, деяка кількість фахівців сходиться на думці, що до системного програмного забезпечення відносяться і ці додатки. Але є кілька фактів, які зумовлюють ставлення до такого роду програм з точки зору самостійних засобів управління інформацією:

- Багато операційних систем (причому вже давно і практично всі) можуть одночасно працювати практично з усіма файловими системами.
- Деякі примітивні (але все ж ОС) системи можуть взагалі обходитися без подібних додатків.

Системи програмування відносяться до специфічних засобів, але загальне уявлення про них також необхідно, так як без цього неможливо зрозуміти принципи роботи ОС і комп'ютера в цілому. Вони потрібні не тільки для створення, але також для налагодження і запуску додатків.

Фахівці кажуть, що для даного випадку в системне програмне забезпечення входять:

- Засоби для набору і редагування тексту.
- Транслятор або інтерпретатор (для найпростіших мов).
- Редактор зв'язків, також званий компоновщиком.
- Відладчики.
- Повні прикладні бібліотеки для виконання програми.

Відзначимо, що створюється додаток (за рідкісними винятками) під якусь певну ОС. Але системні засоби, які призначені для їх розробки, можуть працювати в абсолютно іншій системі (для цього і потрібні окремі системні бібліотеки). Це дуже важливо, тому що не обмежує самих розробників у виборі бажаної системи для роботи.

Утиліта - невелика програма, призначена для настройки обладнання, операційної системи або виконання інших допоміжних робіт.

Утиліти підрозділяються по роду їх діяльності і зв'язку з операційною системою (ОС).

Класифікація за функціями:

- архіватори;

- переглядачі;
- Видалення програм і файлів;
- Для роботи зі списками файлів;
- Для виконання відновлення;
- Утиліти для управління процесами;
- Діагностика програм і обладнання;
- Оптимізація програм і обладнання.

Класифікація по зв'язку з ОС:

Незалежні - можуть працювати без операційної системи.

- Системні утиліти - входять в ОС і не можуть без неї працювати.
- Оптимізатори диска - дозволяють виконувати відновлення структури файлової системи або виконати дефрагментацію.

Приклади корисних утиліт:

1. CureIt - лікувальна утиліта. Виконує перевірку комп'ютера на наявність шкідливих файлів.
2. CCleaner - чистка комп'ютера.
3. Defraggler - дефрагментація жорсткого диска.
4. Autoruns - управління автозапуском програм в Windows.
5. SpeedFan - контроль швидкості вентиляторів.
6. CrystalDiskInfo - перегляд стану жорстких дисків.

1.3 Вимоги до системного програмного забезпечення

Системні програми повинні відповідати таким вимогам:

- Прозорість роботи;
- Гарантована надійність виконання у відповідності зі специфікаціями (специфікаціями називаються функціональні вимоги);
- Максимальна швидкість виконання;
- Мінімальні витрати на зберігання машинних кодів;
- Підтримка стандартних засобів зв'язку з прикладними програмами.

Ефективність системних програм залежить від часу їх створення і надійності виконуваного коду.

Вимога ефективності системних програм викликає необхідність використання спеціальних мов:

- Машинно-орієнтованих типу мови Assembler і
- Високого рівня типу С або С ++.

До типів даних цих мов віднесено покажчики на дані різних типів або адреси даних і програмних об'єктів.

Робота з більшістю пакетів для розробки системного програмного забезпечення передбачає знання і використання асемблера для створення модулів і асемблерних вставок.

ТЕМА 2. ВВЕДЕННЯ В ОПЕРАЦІЙНІ СИСТЕМИ.

2.1 Поняття та функції операційної системи

Операційна система, скорочено ОС (англ. operating system, OS) — це базовий комплекс програм, що виконує управління апаратною складовою комп'ютера або віртуальної машини; забезпечує керування обчислювальним процесом і організовує взаємодію з користувачем. Операційна система звичайно складається з ядра операційної системи та базового набору прикладних програм.

Функції оперативної системи:

- Головні функції:
  - Виконання на вимогу користувача тих елементарних (низькорівневих) дій, які є спільними для більшості програм і часто зустрічаються майже в усіх програмах (введення та виведення даних, запуск і зупинка інших програм, виділення та вивільнення додаткової пам'яті тощо.
  - Стандартизований доступ до периферійних пристроїв (пристрої введення-виведення).
  - Завантаження програм у оперативну пам'ять і їх виконання.
  - Керування оперативною пам'яттю (розподіл між процесами, організація віртуальної пам'яті).
  - Керування доступом до даних енергонезалежних носіїв (апаратний диск, оптичні диски тощо), організованим у тій чи іншій файловій системі.
  - Відтворення інтерфейсу користувача.
  - Мережеві операції, підтримка стеку мережевих протоколів.
- Додаткові функції:
  - Паралельне або псевдопаралельне виконання задач (багатозадачність).
  - Розподіл ресурсів обчислювальної системи між процесами.
  - Організація надійних обчислень (неможливості впливу процесу на перебіг інших), основана на розмежуванні доступу до ресурсів.
  - Взаємодія між процесами: обмін даними, синхронізація.
  - Захист самої системи, а також даних користувача і програм від дій користувача або інших програм.
  - Багатокористувацький режим роботи та розподілення прав доступу (автентифікація, авторизація).

2.2 Структура ОС

Сучасні операційні системи характеризуються методами керування чотирьох основних компонент: процесами, пристроями введення/виведення, пам’яттю, файлами.

Звичайно більшість сучасних ОС побудовано на базі мікроядра (kernel чи nucleus), яке забезпечує планування і диспетчеризацію задач, а також здійснює їх взаємодію.

Схематично операційну систему можна зобразити як взаємодію наступних компонент:

ядро операційної системи,
драйвери,
інтерфейс користувача,
конфігураційні файли,
допоміжні програми (утіліти).

2.2.1 Ядро операційної системи

Формально ядро – це центральна частина операційної системи, яка забезпечує програмам координований доступ до ресурсів комп'ютера, таким як процесорний час, оперативна пам'ять, зовнішнє обладнання та сервіси файлової системи.

Є наступні типи архітектур ядер: монолітна, модульна, мікроядро, екзоядро та гібридне ядро.

Монолітне ядро надає багатий набір абстракцій обладнання, однак усі частини монолітного ядра працюють в одному адресному просторі. Такий метод має той недолік, що при порушенні нормальної роботи одного з компонентів можна втратити працездатність усієї системи.

Старі монолітні ядра вимагали повторного транслювання всіх компонент за будь-якої зміни складу обладнання. Більшість сучасних ядер дозволяють під час роботи довантажувати модулі, що виконують деякі частини функції ядра, що значно спрощує розроблення додаткових модулів. Прикладом операційних систем з такими ядрами є UNIX (з традиційним ядром, таким як BSD), Linux, MS-DOS.

Модульне ядро це вдосконалена модифікація архітектури монолітних ядер операційних систем комп'ютерів. На відміну від "класичних" монолітних ядер, модульні ядра, як правило, не вимагають повної перекомпіляції ядра після зміни складу апаратного забезпечення комп'ютера. Замість цього модульні ядра надають механізми підвантаження модулів ядра, які підтримують певне апаратне забезпечення (драйвери).

Підвантаження модулів може бути як динамічною (виконується "на ходу", без перезавантаження ОС, в працюючій системі), так і статичною (виконується під час перезавантаження ОС внаслідок зміни конфігурації системи). Але всі модулі ядра працюють в адресному просторі ядра і можуть користуватися всіма функціями, що надаються ядром. Тому модульні ядра продовжують залишатися монолітними.

Модульні ядра надають особливий програмний інтерфейс (API – Application Programming Interface) для зв'язування модулів з ядром, для забезпечення динамічного підвантаження і вивантаження модулів. В свою чергу, не кожна програма може бути зроблена модулем ядра: на модулі ядра накладаються певні обмеження в частині використовуваних функцій (наприклад, вони не можуть користуватися функціями стандартної бібліотеки C/C++ і повинні використовувати спеціальні аналоги, що є функціями API ядра).

Мікроядро. Мікроядро надає тільки елементарні функції керування процесами і мінімальний набір абстракцій для роботи з обладнанням. Більша частина роботи здійснюється за допомогою спеціальних користувацьких процесів, що називаються сервісами.

Перевагою таких ядер є стійкість до збоїв обладнання, помилками в компоненах системи. Недоліками – передавання даних між процесами вимагає накладних витрат.

Прикладами систем побудованих на базі мікроядра є: QNX; Mach, Mac OS X, AIX, Minix, ChorusOS, AmigaOS тощо.

Екзоядро. Екзоядро (екзо – це приставка, що позначає дещо зовнішнє, таке що знаходиться ззовні) – це ядро операційної системи комп'ютерів, що надає лише функції для взаємодії між процесами та безпечного виділення і звільнення ресурсів.

Гібридне ядро. Гібридне ядро це модифіковане мікроядро, що дозволяє для пришвидшення роботи запускати "несуттєві" частини в просторі ядра. Прикладами операційних систем є Windows NT, DragonFly BSD.

З певними особливостями функції ядра операційної системи можна згрупувати наступним чином:

- організація паралельного процесу обчислень, програмне перемикання задач,
- керування процесами та потоками, захист оперативної пам'яті процесу тощо,
- розподіл ресурсів комп'ютера (процесорний час, оперативна пам'ять, системні об'єкти синхронізації обчислень, тощо),
- організація доступу до файлової системи (надання абстрактних методів збереження інформації на постійних носіях).

2.2.2 Драйвери

Дра́йвер(англ. driver) – це комп'ютерна програма яка реалізує абстрактний рівень доступу до реального устаткування комп'ютера. За допомогою такої програми інша програма (операційна система або

програма користувача) отримує доступ до апаратного забезпечення стандартним чином.

В загальному випадку ефективне використання пристрою потребує спеціального драйвера. Переважно інсталяція операційної системи має у своєму складі драйвери головних компонент системи без яких система просто не зможе працювати а також драйвери для компонент від зареєстрованих виробників.

Операційна система керує своєрідним "віртуальним пристроєм", який розуміє типовий набір команд. Драйвер перетворює ці команди у команди нижчого рівня які розуміє безпосередньо устаткування. Така структура називається "абстрагуванням від апаратного забезпечення".

Драйвер виконує кілька важливих функцій, а саме:

•завантаження, початкове реєстрування в системі та ініціалізація;

•вивантаження та звільнення захоплених ресурсів;

•забезпечення файлового методу доступу до своїх ресурсів згідно стандарту POSIX:

•відкрити Файл (Open/Create),

•прочитати або записати дані (Read/Write),

•закрити файл (Close),

•керування пристроєм (I/O Control).

З появою складних операційних системи функції БІОС є практично заміщені відповідними драйверами за умов нормальної роботи. Спеціалізований драйвер який володіє набагато більшою інформацією про особливості певного устаткування зможе перевести його в більш оптимальний режим і керувати з максимальною ефективністю.

2.2.3 Інтерфейс користувача

Слово **інтерфейс** (англ. interface) – місце чи спосіб з'єднання (зв'язку). Цей термін використовується в різних областях науки і техніки. Інтерфейси є основою взаємодії користувача та інформаційних систем. Якщо інтерфейс якого-небудь об'єкту (персонального комп'ютера, програми, функції) не змінюється (стабільний, стандартизований), це дає можливість модифікувати сам об'єкт без перебудови принципів його взаємодії з іншими об'єктами. Наприклад, користувач навчившись працювати з однією програмою в середовищі Windows легко засвоїть й інші – оскільки вони мають подібний інтерфейс.

Графічний інтерфейс користувача. Графічний інтерфейс користувача (GUI – graphical user interface) у комп'ютерній термінології означає систему засобів взаємодії користувача з комп'ютером, яка ґрунтується на представленні наявних ресурсів та операцій у вигляді графічних зображень

на робочому полі екрану. (меню вікон, значків, піктограм, меню тощо). На відміну від командного рядка користувач має довільний доступ до всіх видимих компонент (наприклад використовуючи мишку).

Початково ідея графічного інтерфейсу бала запропонована вченими з лабораторії Xerox PARC в 1970-х роках, але отримала комерційне застосування в серії комп'ютерів Apple Computer. В операційній системі AmigaOS графічний інтерфейс із використанням багатозадачності був запропонований у 1985 році. На сучасному етапі розвитку комп'ютерної техніки ГІ є стандартом і його включають до складу усіх комерційних операційних систем.

Прикладами найпопулярніших ОС з графічним інтерфейсом є Mac OS, Microsoft Windows, NEXTSTEP, X Window System тощо.

Інтерфейс командного рядка. Інтерфейс командної стрічки – різновид діалогового текстового інтерфейсу людини і комп'ютера, в якому інструкції комп'ютеру даються лише шляхом введення з клавіатури текстових стрічок (команд). Такий інтерфейс існує майже з часу появи перших комп'ютерів.

Загальний формат команд (в квадратні дужки поміщені необов'язкові частини):[символ-запрошення початку команди] команда [параметр_1 [параметр_2 […]]]

Символ початку команди є в більшості візуальним елементом і може бути будь-яким (/, >, #, $ та ін.).

Команда означає певну дію яку потрібно виконати комп'ютеру. Якщо дія стосується певного об'єкта або повинна бути виконана з певними особливостями то додатково вказуються параметри розділені символом прогалини.

3.4.3.4. Утиліти

Групу утиліт складає набір функцій та програм які виконують допоміжні операції в роботі операційної системи. Вони відіграють важливе значення в тих системах де ядро містить обмежений набір головних функцій і виникає потреба спрощувати методи роботи з ними для вирішення прикладних задач.

Початково операційна система створена розробниками має певний набір таких сервісних програм який називають системними утилітами. Згодом в процесу адаптування ОС до вирішення певного кола прикладних задач перелік утиліт поповнюється додатковими програмами виготовленими іншими виробниками.

2.2.4 Конфігурація (конфігураційні файли, настроювання)

Початковий старт системи забезпечено кількома відомими програмами. Проте наступні кроки які повинна зробити система – це налаштувати роботу комп’ютера для роботи із заданим устаткуванням та для виконання певних прикладних задач. Для цих потреб в складі кожної операційної системи є методи які дозволяють користувачу вказати ті особливості роботи які потрібні саме йому. Для прикладу - в операційній системі MS DOS для служать 2 текстові файли із зарезервованими назвами які "розуміє" система: config.sys та autoexec.bat.

До головних функцій конфігураційних файлів операційної системи належать:

- завантаження необхідних драйверів встановленого устаткування комп’ютера,
- виконання початкового сценарію завантаження допоміжних програм,
- перехід до діалогового режиму (командний рядок або графічний інтерфейс користувача),
- встановлення глобальних змінних (шляхи пошуку програм, папка тимчасових файлів, особливі змінні певних програм тощо).

2.3 Поняття процесу

Програма - статичний об'єкт, що представляє собою файл або сукупність файлів з кодами і даними. Для того щоб програма могла бути запущена на виконання, операційна система повинна створити оточення або середу виконання завдання, що включає можливості доступу до різних системних ресурсів (пам'ять, пристрої введення-виведення, файли і т.д.). Таке оточення отримало назву процесу. Процес являє собою виконуваний образ програми, що включає відображення в пам'яті виконуваного файлу, отриманого в результаті компіляції і зв'язування вихідного коду програми, кодів даних і бібліотек, стека, а також ряду структур даних ядра, необхідних для управління процесом. В цілому, процес можна уявити як сукупність даних ядра системи, необхідних для опису образу програми в пам'яті і управління її виконанням.

Процеси можна умовно розбити на три категорії:

- системні:
- фонові (демони);
- прикладні (призначені для користувача).

Системні процеси є частиною ядра ОС і завжди розташовані в оперативній (основний) пам'яті. Що Їх інструкції і дані цих процесів знаходяться в ядрі системи, і тому вони можуть викликати функції і звертатися

до даних, недоступним для інших процесів, наприклад диспетчер сторінкового заміщення, диспетчер пам'яті ядра, диспетчер буферного кешу і інші.

Фонові процеси або демони - це неінтерактивні процеси, які зазвичай запускаються при ініціалізації системи (після ініціалізації ядра) і забезпечують роботу різних підсистем. Наприклад, системи термінального доступу, системи друку, системи мережевого доступу та інші. Демони не пов'язані з одними сеансами роботи і не можуть керувати безпосередньо користувачами.

Прикладні процеси, як правило, породжуються в рамках призначеного для користувача сеансу. Вони можуть виконуватися як в інтерактивному, так і в фоновому режимах.

Більшість процесорів підтримують два режими роботи: привілейований, або режим ядра, і призначений для користувача, або режим завдання. Певні команди виконуються тільки в привілейованому режимі, наприклад команди управління пам'яттю і введенням-виведенням. Режим роботи встановлюється в регістрі слова стану процесора (PSW) бітом режиму виконання, який може бути змінений при настанні деяких подій. Наприклад, якщо в результаті переривання управління призначеним для користувача процесом переходить до процедури ОС, дана процедура змінює режим виконання на привілейований. Перед поверненням управління призначеному для користувача процесу режим виконання змінюється назад на призначений для користувача. Програми, що виконуються в режимі ядра, володіють повним контролем над процесором і мають доступ до всіх елементів пам'яті.

ТЕМА 3.
УПРАВЛІННЯ ПРОЦЕСАМИ

3.1 Стани процесу

Поняття обчислювальний процес є одним з основних при розгляданні ОС. Процес або задача – це програма під час виконання на процесорі із послідовним виконанням команд. Сам процесор розглядається в двох аспектах:

- Він є носієм даних.
- Він виконує операції пов’язані з обробкою даних.

Процесом може бути виконання утиліти, виконання прикладної програми, транслятор деякої вихідної програми. При чому коли транслюється одна програма – це один процес, а коли інша – інший процес, хоча працює один і той самий транслятор.

Розглянемо комп’ютер з одним центральним процесом, але всі наступні питання справедливі і для багатопроцесорних систем:

Процес приймає ряд дискретних станів (Рис. 3.1).

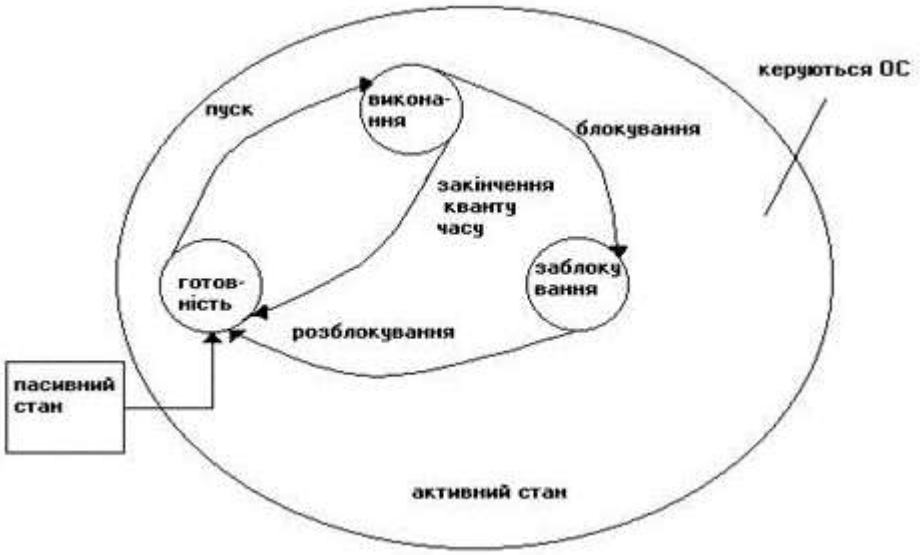


Рис. 3.1 Стани процесу.

Зміну станів процесу можуть викликати різні події. Процес знаходиться в стані виконання, якщо в біжучий момент йому надається центральний процесор. Процес знаходиться в стані готовності, якщо він міг би одразу використовувати центральний процесор, який знаходиться в його розпорядженні. Процес знаходиться в стані заблокованості, якщо він очікує на деяку подію; приклад: завершення операції вводу-виводу для того, щоб отримати можливість продовжити виконання. Це основні стани процесу.

В однопроцесорному комп'ютері в кожен конкретний момент часу може викликатися один процес, але декілька процесів можуть знаходитись в стані готовності, а деякі бути заблоковані. Тому створюють списки: список процесів готових до виконання: 2 список або черга заблокованих процесів. Перший список або черга впорядкований за пріоритетом таким чином наступний процес який отримує в своє розпорядження центральний процесор буде першим цього списку. Другий список не впорядкований, не передбачається ніякого пріоритетного порядку розблокування процесорів, тобто їх переводу в стан готовності. Розблокування відбувається в тому порядку, в якому відбуваються події, які очікуються заблокованими процесами. ОС як правило процес з'являє при запуску довільної програми. Коли процес створено, він займає місце в кінці списку готових до виконання процесів. Під час функціонування ОС такий процес поступово просувається до початку списку в відповідності з тим, як завершиться виконання попередніх процесів. Коли процес буде першим у списку готових до виконання і коли звільниться центральний процесор, то центральний процесор буде цьому процесу і в той момент відбувається зміна стану процесу. Вибір процесу до виконання називається пуском. Це виконання за допомогою програми ОС, яка називається диспетчер. В мультипрограмних системах для того, щоб запобігти випадковому або навмисному захвату ресурсів комп'ютера одним довільним процесом ОС встановлює в спеціальному апаратному таймері переривань

деяке значення, яке визначає часовий інтервал або квант часу на протязі якого біжучому процесу користувача дозволяється утримувати центральний процесор в своєму розпорядженні. Після закінчення часу таймер виробить сигнал переривань за яким керувати буде передане ОС. Після цього ОС переведе процес, який перед цим виконувався в стані готовності, а перший процес із списку готовий переведе в стан виконання.

Якщо процес, який виконувався ще до закінчення кванту часу згенерує операцією вводу-виводу він тим самим сам звільнить центральний процесор, тобто себе сам заблокує на час завершення операції вводу-виводу.

Системи з трьома активними станами процесів можлива ще одна зміна станів після завершення операції вводу-виводу. Процес перейде із заблокованого стану в стан готовності. Таким чином маємо чотири можливі зміни стану процесу. Єдина зміна станів, яка викликається самим процесом це блокування інші при зміні викликів об'єктами, які є зовнішніми по відношенню до біжучого процесу.

Пасивний стан – це програма, яка не виконується.

Процес із пасивного стану може перейти в стан готовності в таких випадках:

- За командою користувача. Це має місце в таких інтерактивних або діалогових ОС, де програма може мати статус задачі, а не бути файлом виконання і тільки на час виконання вона може отримувати статус задачі, тобто процесу.
- При виборі планувальника з черги процесу.
- За викликом із інших задач. Один процес може створювати, ініціювати, призупинити і зупинити інший процес.
- За перериванням від зовнішнього пристрою. Сигнал при виконання деякої події може записувати відповідну задачу

При надходженні запланованого часу програм із стану виконання процес може вийти з таких причин:

- процес завершується. При цьому він передає керування ОС і повідомляє про своє завершення. В результаті процес переходить в пасивний стан або знищується. Знищується не сама програма, а саме активний процес, який відповідає виконанню деякої програми. В пасивний стан також може бути переведений за командою користувача.
- процес переводиться ОС в стан готовності у зв'язку виникнення більш пріоритетної задачі або через завершення виділення кванту часу.
- процес блокується або через запит операції вводу-виводу або через те, що йому неможливо надати ресурс на який виник запит, або за командою користувача призупинку процесу. З початком відповідної дії процес буде деблоковано і переведено в стан готовності до виконання.

Таким чином силою, яка міняє стани процесу це події і один з видів подій це - переривання.

3.2 Блок керування процесом.

Для того, щоб ОС могла керувати процесуми, вона повинна володіти всією необхідною інформацією про процеси. Для цього для кожного процесу створюється спеціальна інформаційна структура, яка називається блок керування процесом (PCB) або дескриптор процесу або описувач задачі. В загальному випадку PCB вміщує таку інформацію:

- Ідентифікатор процесу (PID).
- Тип або клас процесу, який визначає для ОС деякі правила надання ресурсів.
- Пріоритет процесу в відповідності з яким ОС надає ресурси в рамках. В рамках одного класу у процесі обслуговуються більш пріоритетні процеси.
- Зміна стану, яка визначає в якому стані знаходиться процес (готовність, виконання, блокування).
- Адреса захищеної області пам’яті, в якій зберігаються біжучі значення регістрів процесора, якщо процес призупиняється або переривається не завершивши роботу. Ця інформація називається контекст процесу або контекст задачі.
- Інформація про ресурси, якими володіє процес або має право використовувати.
- Адреса місця в основній пам’яті для організації спілкування з іншими процесами.
- Параметри часу запису. Це момент часу, коли процес повинен активізовуватись, а також вказується періодичність цієї процедури.

Для дисрезидентних задач, тобто таких, які постійно знаходяться на зовнішній пам’яті і завантажуються в основну пам’ять тільки на час виконання, зберігається адреса задачі на диску в її вихідному стані. PCB як правило постійно розташований в ОП з метою прискорення роботи ОС, яка організовує їх в чергу і відображає зміну стану процесу шляхом переміщення відповідного PCB з одного стану в інший. Для кожного стану ОС веде відповідний список процесів, які знаходяться в цьому стані таким чином PCB це об’єкт, який визначає процес для ОС.

3.3 Операції над процесами.

При керуванні процесами ОС повинна мати можливість виконувати певні операції над процесом, а саме:

1. Створення процесу.
2. Знищення процесу.
3. Відновлення процесу.

4. Зміну пріоритету процесу.
5. Блокування процесу.
6. Розблокування процесу.
7. Вибір процесу або пуск.

При ***створенні процесу*** відбувається:

1. Присвоєння процесу ім’я.
2. Включення всього імені в список імен процесів, які відомі системі.
3. Визначення початкового пріоритету процесу.
4. Формування блоку РСВ.
5. Виділення процесу початкових ресурсів.

Процес може викликати новий процес. В такому випадку перший процес це батьківський процес, а знову створений – це згенерований активний процес. При такому підході створюється ієрархічна структура процесів(Рис.3.2), в якій у згенерованого процесу є тільки один батьківський процес, а у кожного батьківського може бути багато згенерованих.

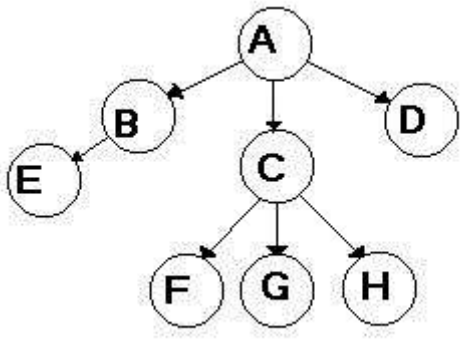


Рис.3.2 Ієрархічна структура процесів

Знищення процесу.

Знищення процесу – вилучення процесу із системи. Ресурси, які були у розпорядженні процесу повертаються в систему, ім’я процесу в системних списках та таблицях стираються. Блок РСВ знищується.

3.4 Призначення операцій призупинення та відновлення процесів. Розширення діаграми переходів процесу зі стану в стан.

Призупинення – важлива операція, яка реалізується в багатьох різних системах по різному. Призупинений процес не може продовжувати своє виконання до того моменту, доки його не активізує довільний інший процес. Дуже часто ОС використовує призупинення для короткочасного виключення певних процесів в періоди пікового навантаження на систему.

У випадку довгострокового призупинення процесу його ресурси повинні бути вивільнені.

Рішення про необхідність вивільнення конкретних ресурсів в значній мірі залежить від природи цього ресурсу. Основна пам’ять звільнюється негайно, в той час як зовнішній запам’ятовуючий пристрій у випадку

короткострокового призупинення може залишатись за проц. Призупинення та відновлення відіграють важливу роль з декількох причин:

Якщо система працює не надійно, є ознаки, що вона може відмовити, то біжучі процеси можна буде призупинити для того, щоб знову їх активізувати після виправлення помилки.

Користувач у якого окремі проміжні результати роботи викликають сумніви може призупинити процес, а не вилучити його зовсім доки не стане зрозуміло правильно чи неправильно працює цей процес.

Деякі процеси можна призупиняти в періоди пікового навантаження на систему з тим, щоб потім їх відновити.

З врахуванням призупинення та відновлення діаграма станів процесу буде мати такий вигляд(Рис. 3.3):

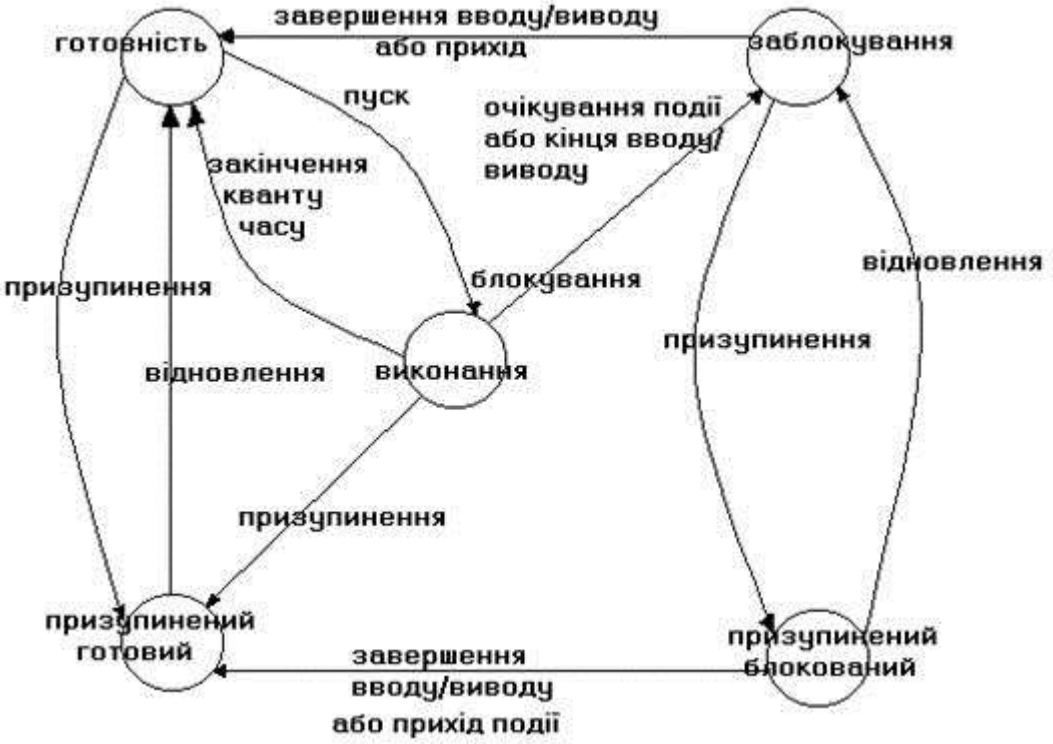


Рис. 3.3 Діаграма станів процесу.

Виникає питання: для чого переводити заблокований процес в стан призупинення?

Справа в тому, що завершення операції вводу-виводу, або подія, яка очікується може ніколи не відбутись, або можуть затриматись на невизначений час, тому перед розробниками ОС виникає вибір:

1.Виконувати призупинення заблокованого процесу.

2.Передбачити механізм, який дозволив би переводити процес із стану готовності в стан призупинення, коли завершиться операція вводу-виводу, або наступить інша очікувана подія.

Оскільки призупинення є операцією найвищого пріоритету, її треба виконувати негайно.

3.5 Обробка переривань.

В комп'ютері переривання – це подія, при якій міняється нормальна послідовність команд, яка виконується процесором.

Нормальна послідовність команд вважається така, яка визначається програмою, тобто процесом.

Переривання представляє собою механізм, який дозволяє координувати паралельне функціонування окремих пристроїв комп'ютерної системи та реагувати на особливі стани, які виникають при роботі процесора.

Переривання – примусова передача керування від програми, яка виконується до ОС, а через неї до відповідної програми обробки переривань.

Переривання відбуваються при виникненні відповідної події. Механізм переривань реалізований апаратно-програмними засобами.

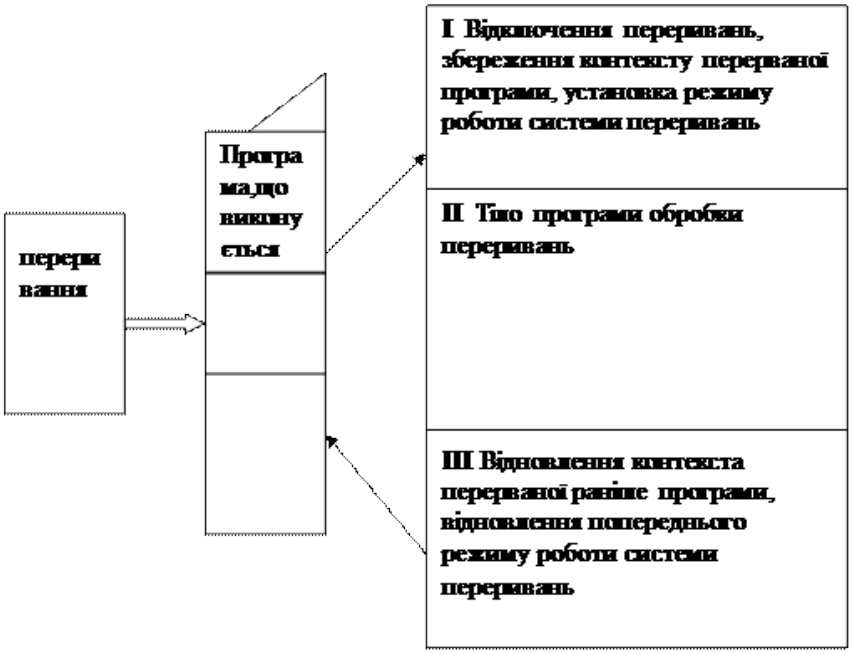


Рис. 3.4 Структура системи переривань

Структура системи переривань (Рис. 3.4) в залежності від архітектури компютера системи може бути різною, але всі вони реалізують одну ідею: переривання викликає зміну порядку виконання команд процесора.

Перехід від перерваної програми до обробленої програми і назад повинен виконуватись як можна швидше. Один з методів реалізації цього є використання таблиці, яка вміщує перелік всіх допустимих для комп'ютера переривань та адреси відповідних обробників переривань. Для коректного повернення до переривань програма перед передачею управління обробнику переривань вмістиме регістрів процесора запам'ятовується або в пам'яті з прямим доступом, або в системному стеку. Таким чином, якщо відбулось переривання, то:

Керування передається ОС.

ОС запам’ятовує стан перерваного процесу, як правило ця інформація запам’ятовується в блоці PCB для перерваного процесу.

ОС аналізує тип переривань і передає керування відповідній програмі обробки цього переривання.

Ініціатором переривання зокрема може бути процес, який виконується або воно може бути викликано деякою подією, пов’язаною або навіть непов’язаною з цим процесом.

Переривання, які виникають при роботі комп’ютерної системи можна розділити на такі основні групи:

1. Зовнішні або асинхронні переривання.

Вони викликаються асинхронними подіями, які відбуваються поза процесом, який переривається. Це такі:

1. Переривання від таймера, тобто від системного годинника.

2. Переривання від зовнішніх пристроїв.

3. Переривання від збоїв живлення.

4. Переривання з пульта оператора системи або від користувача.

5. Переривання від іншого процесора або від іншої комп’ютерної системи.

2. Внутрішні переривання.

Викликаються подіями, які пов’язані з роботою процесора і є синхронними з операціями процесора. Це такі:

1. при неправильній адресації, коли в адресній частині команди, яка виконується вказано заборонену або неіснуючу адресу, вказано звертання до неіснуючого сегмента пам’яті або сторінки, при організації механізму віртуальної пам’яті;

2. при наявності в полі коду операції коду неіснуючої команди;

3. при діленні на нуль;

3. при переповненні або попаданні порядку в числах з рухомою комою;

4. при виявленні помилок парності, а також помилок в роботі різних пристроїв в апаратурі комп’ютерними засобами контролю.

3. Переривання за звертанням до супервізора.

В деяких комп’ютерах існують привілейовані команди, які може використовувати тільки ОС, а не програмний користувач. Відповідно в апаратурі передбачені різні режими роботи. При спробі виконати команду заборонену в даному режимі відбувається внутрішнє переривання і керування передається супервізору.

4. Програмні переривання.

Ці переривання відбуваються за відповідних командних переривань. За такими командами процесор виконує практично ті самі дії, що і при звичайних внутрішніх перериваннях. Оскільки сигнали переривань виникають в довільні моменти часу, то на момент переривання може існувати декілька таких сигналів, які можуть бути оброблені тільки послідовно. Щоб обробляти їх в деякому логічному порядку їм присвоюють пріоритети. За рівнями пріоритетів переривання розподіляють таким чином(Рис.3.5):

Рис.3.5 Розподіл переривань за рівнями пріоритетів

Сигнали з більш високим пріоритетом обробляють в першу чергу, обробка інших сигналів відкладається.

3.6.Переключення контексту як засіб реалізації переривань.

Для обробки кожного з видів переривань в складі ОС передбачені програми, які називаються оброблювачі переривань (IH, Interrupt Handler).

Коли відбувається переривання, ОС запам’ятовує стан перерваного процесу і передає керування відповідному оброблювачу. Це робиться в спосіб, який називається переключення контексту. При реалізації цього способу використовується слово стану програми (PSW, Program Status Word). Ці слова керують порядком виконання команд і вміщують інформацію відносно стану процесу. Існує три типи PSW:

Нові PSW	Біжачі PSW	Старі PSW
Виклик супервізора		
Ввід вивід		
Внутрішні переривання	B A	
Контроль програми		
Таймер		
Рестарт		

Адреса наступної команди, яка має виконуватись вміщується в біжучому PSW, в якому вказані також типи переривань дозволені і заборонені в біжучий момент.

Центральний процесор реагує тільки на дозволені переривання, обробка заборонених переривань або затримок в деяких випадках ігнорується. Процесору не можна заборонити реагувати на переривання за викликом супервізора, за рестартом, а також на деякі види програмних переривань. В одно процесорній машині є тільки одне біжуче PSW, але n нових PSW, і n

старих PSW. Нове PSW для біжучого типу переривань містить постійну адресу, за якою резидентно розміщується оброблювач переривань цього типу.

Коли відбувається переривання, якщо воно не заборонено для процесора відбувається автоматично, яке виконується апаратно переключення PSW таким чином:

1. біжуче PSW робиться старим PSW для переривань цього типу;
2. нове PSW для переривань цього типу робиться біжучим PSW.

Після такого заміщення біжуче PSW буде вміщувати адресу відповідного оброблювача переривань, який починає відробляти біжуче переривання. Коли обробка переривань завершиться, центральний процесор починає обслуговувати або той процес, який виконується в момент переривань, або готовий процес з найвищим пріоритетом.

3.7 Стратегії планування і диспетчеризація процесів

Планування – це системний процес, який здійснює установку користувацьких процесів в чергу та визначення атрибутів їх виконання в рамках використовуваної обчислювальної системи.

У загальному випадку стратегія планування визначає які процеси повинні бути заплановані на виконання з метою досягнення результатів вирішуваної задачі.

Стратегії можуть бути наступними:

- закінчувати обчислення в тому самому порядку, в якому вони були розпочаті;
- перевагу надавати більш коротким процесам;
- надавати всім процесам однакові послуги, в тому числі і однаковий час очікування.

Розрізняють три типи планування:

- *короткострокове планування* – диспетчеризація – рішення про додавання процесу в пул процесів, що виконуються;
- *середньострокове планування* – рішення про додавання процесу до числа процесів, що повністю або частково розміщені в оперативній пам’яті;
- *довгострокове планування* – рішення про те, який з доступних процесів буде виконуватися процесором (полягає у виборі таких обчислювальних процесів, які менше всього б конкурували між собою в процесі досягнення мети обчислень).

Системний процес який здійснює планування може бути реалізований двома способами:

- *цілісний планувальник* – це програмний модуль, який є частиною операційної системи, його робота ініціюється перериваннями і він виконується як окремий системний процес;

- *розподілений планувальник* – програмний модуль планувальника або його частина записується в кожний процес при його завантаженні в оперативну пам’ять.

У загальному випадку планувальник здійснює наступні функції:

1) запис копії процесу в пам’ять обчислювальної системи;

2) аналіз атрибутів при виконанні обчислювального процесу та формування набору робочих атрибутів в залежності від характеристик обчислювальної системи;

3) запис процесу в чергу процесів у випадку, якщо є вільне місце в черзі, або перевід процесу в пасивний стан у випадку, якщо всі елементи черги зайняті;

4) періодичний перегляд черги процесів, запис або знищення процесів в черзі у відповідності з часом функціонування процесів.

Відома велика кількість правил (дисциплін диспетчеризації), у відповідності до яких формується список (черга) готових до виконання задач.

Розрізняють два великих класи дисциплін диспетчеризації: безпріоритетні і пріоритетні.

При безпріоритетній організації задачі або процеси вибираються згідно з певним, раніше установленим, порядком без врахування важливості цих процесів та необхідного часу обслуговування.

При реалізації пріоритетних дисциплін певним окремим задачам чи процесам надаються виключне право або набір прав для установки такого процесу в чергу.

На рисунку 3.6 представлена структура дисциплін диспетчеризації.

Далі опишемо принципи роботи деяких дисциплін диспетчеризації:

Дисципліна FIFO (first input first output) – є реалізацією безпріоритетної черги. Якщо в черзі звільняється елемент і він не останній, то всі після нього посуваються вперед і в останній вільний елемент черги записується ім’я нового процесу. Першим з черги вибирається елемент, який стоїть на початку черги.

Дисципліна LIFO (last input first output) – першим з черги вибирається елемент, який прийшов останнім.

Ця дисципліна враховує перебування процесів в станах блокування, наприклад при операціях вводу/виводу. Ті, які були заблоковані в процесі роботи після переходу в стан готовності поступають в чергу готовності перед новими задачами, які ще не обслуговувались. При такій дисципліні організовуються дві черги: черга готових до обслуговування процесів і черга нових процесів (рисунок 3.7).

Дисципліна FCFS (first come first server) – є подібна до FIFO.

Ця дисципліна не потребує втручання ззовні в хід обчислювального процесу. Не відбувається розподіл процесорного часу.

Переваги: простота реалізації та невеликий розхід системних ресурсів для організації черги задачі. Однак при збільшенні завантаженості ОС, збільшується середній час обслуговування задачі, коли «короткі» завдання (які потребують невеликих затрат машинного часу) повинні очікувати такий самий час, що й довгі завдання.

Рисунок 3.6 – Структура диспетчеризації

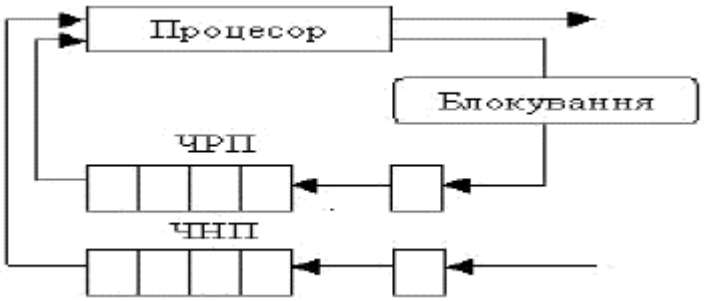


Рисунок 3.7 – Дисципліна FCFS (ЧНП – черга нових процесів; ЧРП – черга робочих процесів)

Дисципліна SJN (short jump next) – згідно з цією дисципліною ОС вимагає, щоб для кожного процесу була відома оцінка необхідних обчислювальних ресурсів. Для постановки задачі в чергу диспетчер оцінює необхідний час виконання задачі і ставить задачу перед довшою задачею. При цій дисципліні існує одна черга процесів і завдання, що були заблоковані знову поступають в кінець черги як і нові завдання. Це приводить до того, що завдання які потребують мало часу очікують процесу як і довгі процеси.

Дисципліна SRT (short remain time) – розроблена з метою забезпечення більш якісного обслуговування коротких завдань. Виконує спочатку ті процеси, час завершення виконання яких найменший.

Дисципліна RR (round robin) – кожна задача отримує порцію часу (квант часу). Після закінчення цього кванту часу задача знімається з виконання

на процесорі і він передається наступній задачі. Задача, яка була знята з черги записується в кінець черги задач, які готові до виконання. Величина кванту часу, як правило, вибирається, як середнє значення між достатнім часом реакції системи на запити користувачів та процесорним часом, який необхідний для перемикання між задачами.

3.8 Алгоритми в диспетчеризації з витісненням та без

Диспетчеризація без розподілу процесорного часу (*багатозадачність без витіснення*) – це такий спосіб диспетчеризації, при якому активний процес виконується до тих пір, поки він за власною ініціативою не передасть управління диспетчеру задач для вибору іншого, готового до виконання процесу. До них відносяться дисципліни: FCFS, SJN, SRT.

Диспетчеризація з розподілом процесорного часу (*багатозадачність з витісненням*) – це такий спосіб, де рішення про перемикання з однієї задачі на іншу приймається диспетчером задач, а не власною активною задачею. Процес, що виконується може бути перерваним і переведений операційною системою в стан готовності до виконання. Рішення про витіснення може прийматися при запуску нового процесу про перериванню або періодично, згідно переривань від таймеру. До таких дисциплін відносяться RR та інші, реалізовані на її основі.

Будь-які процеси мають бути гарантовано певним чином обслужені операційною системою. Гарантоване обслуговування може бути досягнуто трьома способами:

ОС виділяє мінімальну частину процесорного часу деякому класу процесів, у випадку, коли хоча б один з них готовий до виконання;

ОС виділяє мінімальну долю процесорного часу певному конкретному процесу якщо він готовий до виконання;

ОС виділяє стільки часу певному процесу, щоб він міг виконати своє обчислення в певний термін.

Для порівняння алгоритмів диспетчеризації можуть бути використані наступні критерії:

використання (завантаження) процесору – відсоток часу протягом якого процесор зайнятий;

пропускна здатність процесора – це кількість процесів, яка виконується процесором за одиницю часу;

час обороту – інтервал часу від моменту появи процесу у вхідній черзі до моменту завершення процесу. Цей час обороту включає в себе час очікування у вхідній черзі, час очікування у черзі готовності, час готовності у чергах до периферійних пристроїв, час виконання на процесорі та час вводу/виводу;

час очікування – це сумарний час знаходження процесу в черзі очікування готових процесів;

час відповіді – це час від моменту поступлення процесу на вхідну чергу до моменту першого звернення процесу на ввід/вивід даних.

Як відомо, при виконанні операційною системою багатьох процесів може відбуватися зниження продуктивності обчислювальної системи.

Існують певні методи підвищення продуктивності системи:

сумісне планування, при якому всі потоки однієї задачі одночасно вибираються для виконання процесором (у випадку наявності мультипроцесорної системи) і одночасно знімається з виконання. В цьому випадку зменшується час перемикання між задачами.

планування, при якому задачі, що знаходяться в критичній області не перериваються, а закінчують виконання критичної секції. Задачі, які чекають на виконання критичної секції коду не виконують її поки не завершиться переривання.

При виконанні програм, які реалізують функції контролю чи управління в системах реального часу може виникнути ситуація, коли одна або декілька задач не можуть бути вирішені протягом довгого проміжку часу через значну завантаженість обчислювальної системи. В цьому випадку втрати, які пов’язані з невиконанням цих задач можуть бути більшими ніж втрати від невиконання задач з більш високим пріоритетом. Тому іноді виникає необхідність динамічної зміни пріоритету задачі в процесі її виконання. Це дозволяє реалізувати більш швидку реакцію на запити користувачів і гарантувати виконання будь-яких запитів.

Операційна система може змінити пріоритет задачі за допомогою:

Підвищення пріоритету активної задачі: при завантаженні задачі на виконання її пріоритет автоматично збільшується і при цьому знижується час реакції цієї активної задачі на дії користувачів в порівнянні з іншими фоновими задачами;

Підвищення пріоритету операції вводу/виводу: після завершення операції вводу/виводу задача отримує найвищий пріоритет в певному класі задач, що дозволяє більш швидко закінчити всі незавершені операції вводу/виводу;

Підвищення пріоритету «забутих» задач: якщо задача не отримує часу процесора протягом певного відрізку часу, то диспетчер задач тимчасово присвоює їй більш високий пріоритет, який не перевищує певної заданої межі, і таким чином можливо перемикатися на такі «забуті» задачі більш швидко. Після виконання такої задачі за певний квант часу її пріоритет знижується до попереднього значення.

3.9 Взаємодія процесів

3.9.1 Проблема взаємного виключення і способи її вирішення

Два процеси називаються *паралельними (concurrent)*, якщо їх виконання може перекриватися в часі, тобто, наприклад, другий процес починається раніше, ніж завершується перший. Два процеси називаються *послідовними (serial*), якщо вони не є паралельними. мультипроцесорній системі досягається

фізична паралельність. Якщо виконання декількох процесів чергується в одному-єдиному процесорі, то досягається *логічна паралельність.*

Можна виділити три види взаємодії процесів:

• *обмін інформацією* – якщо процеси безпосередньо проінформовані про наявність один одного;

• *конкуренція за доступ до ресурсів* – якщо процеси не проінформовані про наявність один одного; ОС повинна регулювати такі звернення;

• *узгодження дій процесів* – якщо процеси не прямо обізнані один про одний (наприклад, процес А генерує дані, а процес В виводить їх на друк).

При цьому паралельні процеси взаємодіють за допомогою або *механізму поділюваних змінних*, або *механізму передачі повідомлень*.

При роботі паралельних процесів, як в однопроцесорних, так і в багатопроцесорних системах неможливо передбачити відносну швидкість їх роботи, і тому виникають проблеми при взаємодії таких процесів:

1. *Стан гонок* (змагань), коли два процеси використовують одну і ту ж глобальну змінну і обидва виконують читання і запис цієї змінної. Критичним при цьому виявляється порядок читання і запис цієї змінної різними процесами.
2. *Блокування та взаємоблокування при доступі до ресурсів.* Процес А може зажадати і одержати контроль над деяким пристроєм введення/виведення (ПВВ), після чого тимчасово призупинити роботу. При цьому небажано, щоб ОС блокувала даний ПВВ і не дозволяла іншим процесам використовувати його, так як це може призвести до взаємоблокування.
3. *Програмні помилки*, які важко виявити, так як результат роботи програми перестає бути детермінованим і відтворюваним.

Приклад стану гонок. Нехай у нас є процедура, що виконує читання символу з клавіатури і його відображення на екрані, яка знаходиться в пам'яті, що розділяється і використовується всіма процесами:

```
void echo ()
{
chin = getchar (); chout = chin; putchar (chout);
}
```

Розглянемо таку послідовність подій:

1. процес А виконує оператор `chin = getchar ()` – введено символ «*x*»;
2. процес В виконує процедуру echo до кінця – введено і надруковано символ «*y*»;
3. процес А продовжує своє виконання, але при цьому буде виведений вже символ «*y*», а не «*x*», тобто «*y*» буде виведений двічі, а «*x*» – втрачений.

Така проблема не буде виникати, якщо в кожен момент часу тільки один процес буде входити в процедуру `echo`, і ця процедура обов'язково повинна бути повністю виконана процесом, що увійшов до неї до того, як стане можливим її виконання іншим процесом.

Та ж проблема залишається і в багатопроцесорній системі.

Для уникнення проблем типу гонок, блокувань та взаємоблокувань в концепції процесів існує *механізм взаємного виключення* (mutual exclusion).

Взаємне виключення – механізм, який гарантує, що в будь-який момент часу тільки один процес виконує деяку визначену послідовність дій, і тим самим виключає можливість роботи іншого процесу.

Критичний ресурс (КР) – ресурс, до якого в кожний момент часу можливий доступ тільки одного процесу. Доступ до такого ресурсу здійснюється в *критичній секції (КС)* – частини коду, яка в будь-який момент часу може виконуватися тільки одним процесом.

Але взаємне виключення може привести до двох проблем:

- *взаємного блокування* (*deadlock*).
- *голодування* (*starvation*).

Розглянемо як виникає проблема голодування. Нехай є процеси *A, B* і *C*, кожному з яких періодично потрібен доступ до ресурсу *P*. Нехай процес *A* використовує ресурс *P*, а процеси *B* і *C* – очікують. Потім процес *B* використовує ресурс *P*, потім знову процес *A* використовує ресурс *P* і т.д. Тобто, процес *C* може ніколи не отримати доступ до ресурсу *P*, незважаючи на те, що ніякого взаємного блокування немає. Така ситуація і називається *голодуванням*.

Отже, щоб уникнути гонок, ми повинні уникати знаходження двох процесів одночасно в критичних секціях. Але цієї вимоги ще недостатньо! Для правильної спільної роботи паралельних процесів необхідно виконання шести умов для досягнення взаємного виключення:

- два процеси не повинні одночасно перебувати в критичних секціях;
- у програмі не повинно бути закладено ніяких початкових припущень про швидкість або кількість процесорів;
- процес, що перебуває поза критичною секцією, не повинен блокувати інший процес;
- повинна бути неможлива ситуація, коли процес вічно чекає попадання
- критичну секцію (тобто не повинні з'являтися взаємоблокування і голодування);
- коли в КС немає жодного процесу, будь-який процес, що запитав можливість входу в цю КС, повинен негайно отримати доступ до неї;
- процес залишається в КС лише протягом обмеженого часу.

Існують такі підходи до досягнення взаємного виключення:

програмний – в цьому випадку сам процес є «відповідальним» за досягнення взаємного виключення, тобто процес (як системний, так і користувацький) повинен координувати свої дії з іншими процесами для

досягнення взаємного виключення без підтримки з боку ОС або середовища програмування;

апаратний – з використанням спеціальних машинних команд;

примітиви синхронізації – семафори, монітори та інші подібні засоби.

3.9.2 Апаратні способи досягнення взаємного виключення

Існують такі апаратні способи досягнення взаємного виключення:

- *Заборона переривань.*
- *Спеціальні команди, що виконуються атомарно.*

При забороні переривань процесор перемикається на інший процес тільки по перериванню. Відповідно, процес забороняє переривання перед входом у критичну секцію і дозволяє переривання відразу після виходу з критичної секції. Взаємне виключення при цьому досягається, тому що процес не переривається всередині критичної секції, і тому інші процеси не можуть увійти в свої критичні секції.

У такого підходу є певні недоліки: по-перше, він може бути застосований тільки в однопроцесорних системах, по-друге, важливі події введення/виведення можуть бути не оброблені і, по-третє, вразі краху процесу всередині критичної секції може відбутися порушення роботи ОС.

Спеціальні команди, що виконуються атомарно. Під атомарними операціями маються на увазі такі команди зміни даних в оперативній пам'яті, які виконуються процесором з повним блокуванням доступу до пам'яті, тобто жоден інший процесор або зовнішній пристрій не може отримати доступ до комірок пам'яті, поки процесор-монополіст не завершить операцію з нею.

Розглянемо реалізацію взаємного виключення за допомогою атомарної команди «перевірити і встановити» (TestAndSet). Ця команда виконує атомарно дві операції: читання з комірки пам'яті і запис в неї значення «зайнято».

Нехай нульове значення змінної lock відповідає умові, що критична секція вільна, а значення lock, відмінне від нуля, відповідає зайнятості критичної секції. Тоді перед входом у критичну секцію процес повинен в циклі перевіряти, чи звільнилася критична секція:

```
while (TestAndSet (& lock));
/ * Критична секція * /
```

Відповідно, після виходу з критичної секції процес повинен встановити змінну в значення «вільно»:

```
lock = 0;
```

3.9.3 Програмні способи досягнення взаємного виключення

Розрізняють наступні програмні способи досягнення взаємного виключення:

- Змінні блокування
- Алгоритм Деккера.
- Алгоритм Петерсона.

Змінні блокування. Процес для здійснення входу в критичну секцію безперервно перевіряє значення деякої змінної, щоб визначити, чи є вільним ресурс, що розділяється. Ця змінна має назву «*змінна блокування*» і зберігає стан критичної секції. Початкове значення змінної блокування дорівнює *0*. Якщо процес хоче увійти в критичну секцію, то він попередньо зчитує значення змінної блокування, і якщо воно дорівнює 0, то процес змінює його на 1 і входить в критичну секцію. Якщо ж значення цієї змінної вже дорівнює 1, то процес чекає, поки це значення не зміниться на 0. Таким чином, 0 означає, що жодного процесу в критичній секції немає, а 1 – що певний процес вже перебуває в критичній секції.

```
while (busy);
busy = 1;
/ * Критична секція * /
…….
busy = 0;
/ * Решта коду * /
```

При такому підході виникають певні проблеми, а саме:

- якщо після виходу з критичної секції процес з якоїсь причини не встановить значення змінної блокування рівним 0, то інші процеси не зможуть увійти в критичну секцію, порушується умова про те, що процес, який знаходиться поза критичною секцією не повинен блокувати інший процес;
- проблема непродуктивної витрати часу процесору. *Активне очікування* (*busy waiting*) – постійна перевірка змінної в очікуванні деякого значення. *Спін-блокування* – блокування, що використовує активне очікування.

Алгоритм Деккера . Голландський математик Т. Деккер був першим, хто розробив програмне рішення проблеми взаємного виключення (1970). Коли процес *P0* намагається увійти в критичну секцію, він встановлює свій прапор *flag [0]* таким, що рівний *true*, а потім перевіряє стан прапора іншого процесу *P1*. Якщо його прапор дорівнює *false*, то процес *P0* може негайно увійти в критичну секцію, інакше *P0* звертається до змінної *turn*.

Якщо *turn = 0*, то це означає, що зараз – черга процесу *P0* на вхід в критичну секцію, і тоді *P0* періодично перевіряє стан прапора процесу *P1* Цей процес, в свою чергу, в певний момент часу виявляє, що зараз не його черга

для входу в критичну секцію, і встановлює свій прапор рівним *false*, даючи можливість процесу *P0* увійти в критичну секцію.

Після того як *P0* вийде з критичної секції, він встановить свій прапор рівним *false* для звільнення критичної секції і присвоїть змінній *turn* значення *1* для передачі прав на вхід в критичну секцію процесу *P1*:

P0	*P1*
flag [0] = true; *while (flag[0])* *{* *if(turn == 1)* *{* *flag[0] = false ;* *while(turn == 1)* */* очікувати */;* *flag[0] = true;* *}* *}* */* критична секція* */* *turn = 1;* *flag[0] = false;*	*flaq[1] = true;* *while (flag[1])* *{* *if (turn == 2)* *{* *Flag[0] = false;* *while(turn == 1)* */* очікувати */;* *flag [1] = true;* *}* *}* */* критична секція */ *turn =o;* *ffagf/J = false;*

Алгоритм Петерсена. Г. Петерсон розробив набагато більш простий алгоритм взаємного виключення (1981 р.). З цього моменту алгоритм Деккера вважається застарілим.

```
    int turn; / * 0 або 1 - чия зараз черга? * / int
interested [2];
    / * Всі змінні спочатку рівні 0 * /
    void enter_cs (int proc)
    {
    int other = 1 - proc;
    interested [proc] = true;
    turn = proc;
    while (turn == proc && interested [other]);
    }
    void leave_cs (int proc)
    {
    interested [proc] = false;
    }
```

Перед входом у критичну секцію процес викликає функцію enter_cs зі своїм номером (0 або 1) в якості параметра. Після виходу з КС процес викликає

функцію leave_cs , щоб позначити свій вихід і тим самим дозволити іншому процесу вхід в КС.

Початково обидва процеси знаходяться поза КС. Коли процес 0 викликає функцію enter_cs, ця функція повертає керування тому, що процес 1 не зацікавлений в попаданні в КС. Тепер, якщо процес 1 теж викличе функцію enter_cs , то йому доведеться чекати, коли interested [0] стане рівним false, це відбудеться тільки в той момент, коли процес 0 викличе функцію leave_cs. Нехай тепер обидва процеси намагаються викликати функцію enter_cs практично одночасно. Тоді обидва вони збережуть свої номери в змінну turn. При цьому в ній збережеться номер того процесу, який був другим, а попередній номер буде загублений. Нехай другим був процес 1, і відповідно, turn = l. У цьому випадку, коли обидва процеси дійдуть до оператора while, процес 0 увійде в КС, а процес 1 залишиться в циклі і буде чекати, поки процес 0 вийде з КС.

3.10 Примітиви синхронізації

3.10.1 Семафори

Семафор – це спеціальна змінна, яка має ціле значення і пов'язану з ним чергу. Над семафором визначено три операції:

1. семафор може бути ініціалізований ненегативним значенням;
2. операція wait зменшує значення семафора; якщо це значення стає негативним, то процес, що виконує операцію wait, блокується;
3. операція signal збільшує значення семафора; якщо це значення не позитивне, то заблокований операцією wait процес розблокується.

Існує також більш обмежена версія семафора – бінарний семафор, що приймає тільки значення 0 або 1.

Для зберігання процесів, що очікують семафори, використовується черга. Якщо вона обслуговується за принципом FIFO, то такий семафор називається сильним семафором, інакше – слабким семафором.

Можна сказати, що семафори забезпечують рішення для всіх видів проблем синхронізації, але їм також властиві недоліки:

процес, що використовує семафор, повинен бути обізнаний про інші процеси, що використовують цей же семафор, так як операції над семафорами у всіх взаємодіючих процесах повинні бути чітко скоординовані;

операції над семафорами у всіх процесах повинні бути ретельно налагоджені, так як пропуск однієї з таких операцій може призвести до неспроможності (порушення цілісності ресурсу, що розділяється) або до взаємного блокування;

програми, що використовують семафори, дуже важко перевіряти на коректність.

Наведемо приклад застосування семафорів при взаємному виключенні. Розглянемо рішення задачі взаємовиключення з використанням семафора s. Нехай є n процесів. У кожному з цих процесів перед входом у критичну секцію

виконується виклик wait (s). Якщо значення s при цьому стає негативним, то відповідний процес призупиняється. Якщо ж це значення дорівнює 1, то воно зменшується до нуля, а процес негайно входить у критичну секцію. І оскільки s вже більше не є позитивним, жоден інший процес не може увійти в критичну секцію.

```
semaphore s = 1:
……….
wait (s);
/ * Критична секція * /
signal (s);
/ * Решта коду * /
```

Семафор ініціалізується значенням 1. Отже, перший процес, що виконує виклик wait (s), може негайно потрапити в критичну секцію, встановлюючи при цьому значення семафора рівним 0. Будь-який інший процес при спробі увійти в критичну секцію виявить, що вона зайнята. Відповідно, відбудеться блокування такого процесу, а значення семафора буде зменшено до -1. Намагатися увійти в критичну секцію може будь-яка кількість процесів, і кожна така неуспішна спроба зменшує значення семафору. Після того як процес, який увійшов в критичну секцію першим, покине його, значення s збільшується, а один із заблокованих процесів (якщо такі є) віддаляється з черги семафора і активізується. Таким чином, як тільки планувальник ОС надасть цьому процесу можливість виконання, процес тут же зможе увійти в критичну секцію.

Сигналізуючий семафор – це семафор з нульовим початковим значенням. Процес сигналізує про подію, виконуючи операцію signal (s), а всі інші процеси очікують події, виконуючи операцію wait (s).

3.10.2 М'ютекси

М'ютекс - це спрощена версія семафора: змінна, яка може знаходитися в одному з двох станів – блокованому і неблокованому.

Значення м'ютекса встановлюється за допомогою двох процедур. Якщо процес (або потік) збирається увійти в критичну секцію, то він викликає процедуру mutex_lock. Якщо м'ютекс не заблокований, тобто вхід в критичну секцію дозволений, то запит виконується, а процес, що викликає входить в критичну секцію. Якщо ж м'ютекс заблокований, то процес блокується до тих пір, поки інший процес, що перебуває в критичній секції, не вийде з неї, викликавши процедуру mutex_unlock. Якщо при цьому м'ютекс блокував кілька процесів; то з них вибирається один.

3.10.3 Монітори

Щоб спростити написання програм, був запропонований механізм, синхронізації більш високого рівня.

Монітор – це набір процедур, змінних і інших структур даних, об'єднаних в особливий модуль. Він являє собою високорівневу конструкцію мови програмування, яка забезпечує функціональність, еквівалентну до функціональності семафорів, але при цьому ним набагато легше управляти. Монітори реалізовані в мовах програмування Concurrent Pascal, Pascal-Plus, Modula-2, -3, Java.

Монітор являє собою програмний модуль, який складається з послідовності, що ініціалізується однією або декількома процедурами і локальних даних. Основні характеристики монітора:

- локальні змінні монітора доступні тільки його процедурам; зовнішні процедури доступу до локальних даних монітора не мають;
- процес входить в монітор шляхом виклику однієї з його процедур;
- в моніторі в певний момент часу може виконуватися тільки один процес; будь-який інший процес, що викликав монітор, буде припинений в очікуванні доступності монітора.

Перші дві характеристики нагадують нам об'єкти в об'єктно-орієнтованому програмуванні; дотримання ж третьої умови дозволяє монітору забезпечити взаємовиключення. Дані монітора доступні в деякий конкретний момент тільки одному процесу, - отже, захистити спільно використовувані дані можна, помістивши їх в монітор. Якщо дані в моніторі представляють якийсь ресурс, то монітор забезпечує взаємовиключення при зверненні до цього ресурсу.

Для широкого застосування в паралельних обчисленнях монітори повинні включати в себе інструмент синхронізації. Припустимо, наприклад, що якийсь процес використовує монітор і, перебуваючи в цьому моніторі, він повинен бути припинений до виконання певної умови. При цьому потрібно механізм, який не тільки призупиняє процес, але і звільняє монітор, дозволяючи увійти в нього іншому процесу. Пізніше, коли умова виявиться виконаною, а монітор доступним, припинений процес зможе продовжити свою роботу з того місця, де він був призупинений.

Монітор підтримує синхронізацію за допомогою змінних умови, що розташовуються в моніторі і доступних тільки в моніторі. Працювати з цими змінними можуть дві функції:

swait (c) – призупиняє виконання процесу, що викликає за умовою c; монітор при цьому доступний для використання іншим процесом;

csignal (c) – відновлює виконання процесу, припиненого викликом cwait (c); якщо є декілька таких процесів, то вибирається один з них, а якщо таких процесів немає, то ця функція нічого не робить.

ТЕМА 4.
УПРАВЛІННЯ ОПЕРАТИВНОЮ ПАМ'ЯТТЮ.

Основна пам'ять (ОП), в якій розміщуються процеси, є досить обмеженим і дорогим ресурсом. Тому організація і управління ОП ЕОМ є одним із найважливіших факторів, що визначають побудову та розвиток ОС.

Саме організація і управління ОП багато в чому визначають фактичний рівень мультипрограмування ОС, тобто можливості виконання декількох паралельних процесів. Функціями ОС з управління пам'яттю є:

- відстеження вільної і зайнятої пам'яті;
- виділення пам'яті процесам і звільнення пам'яті при завершенні процесів;
- витіснення процесів з оперативної пам'яті на диск, коли розміри основної пам'яті не достатні для розміщення в ній усіх процесів, і повернення їх в оперативну пам'ять, коли в ній звільняється місце;
- налаштування адрес програми на конкретну область фізичної пам'яті.

Основними завданнями підсистеми управління пам'яттю є:

1. Ефективність розміщення процесів в ОП.
2. Захист пам'яті процесів.

Основна мета управління пам'яттю - забезпечити максимальний рівень мультипрограмування і тим самим максимальне завантаження центрального процесора. На рис. 4.1 показана ієрархічна структура пам'яті ЕОМ.

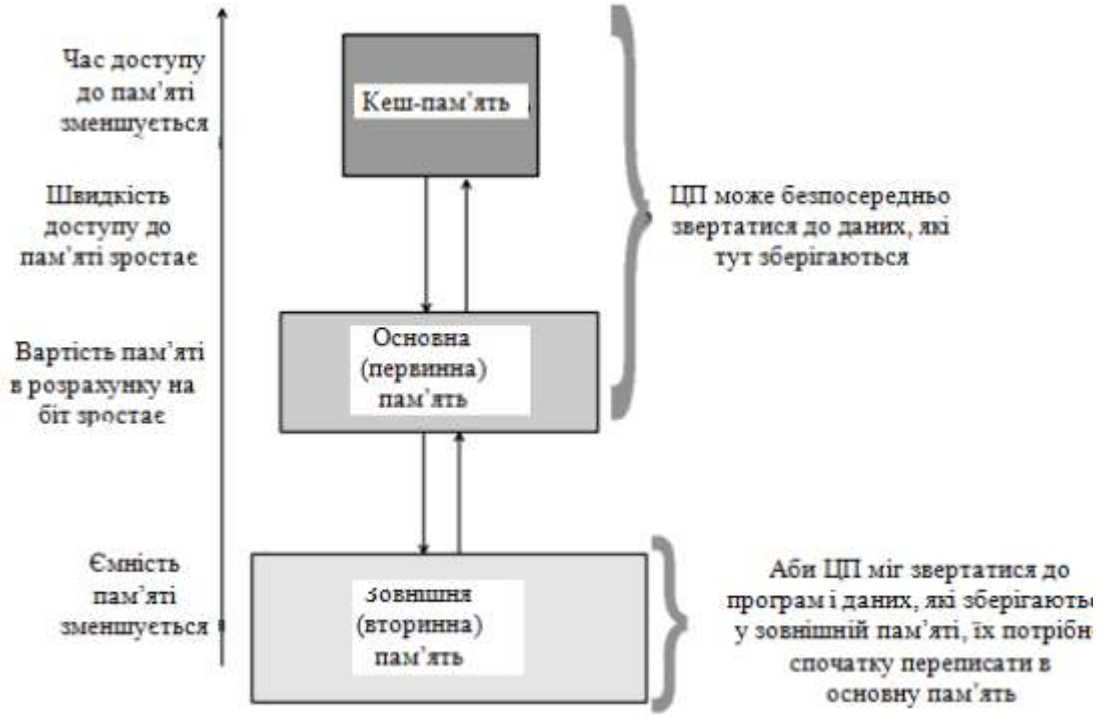


Рис. 4.1. Ієрархічна організація пам'яті ЕОМ

Сучасні ЕОМ мають 3-х рівневу, ієрархічну організацію запам'ятовуючих пристроїв (ЗУ) (внутрішні регістри процесора, різні типи надоперативної і оперативної пам'яті, диски, стрічки), що відрізняються

середнім часом доступу і вартістю зберігання даних у розрахунку на один біт (рис. 4.1) . Користувачеві хотілося б мати і недорогу і швидку пам'ять. Кеш-пам'ять є деяким компромісним рішення цієї проблеми.

Кеш-пам'ять - це спосіб організації спільного функціонування двох типів запам'ятовуючих пристроїв (ЗП), що відрізняються часом доступу і вартістю зберігання даних. Цей пристрій дозволяє зменшити середній час доступу до даних за рахунок динамічного копіювання в "швидкий" ЗП найбільш часто використовуваної інформації з "повільного" ЗП.

Кеш-пам'яттю часто називають не тільки спосіб організації роботи двох типів запам'ятовуючих пристроїв, а й один із пристроїв - "швидкий" ЗП. Він коштує дорожче і, як правило, має порівняно невеликий обсяг. Важливо, що механізм кеш-пам'яті є прозорим для користувача. Цей механізм не повинен повідомляти ніякої інформації про інтенсивність використання даних і не повинен ніяк брати участь у переміщенні даних із ЗП одного типу в ЗП іншого типу, все це робиться автоматично системними засобами.

Розглянемо окремий випадок використання кеш-пам'яті для зменшення середнього часу доступу до даних, що зберігаються в оперативній пам'яті. Для цього між процесором і оперативною пам'яттю розміщується швидкий ЗП, який називається кеш-пам'яттю. За такий може бути використана, наприклад, асоціативна пам'ять. Вміст кеш-пам'яті є сукупністю записів про всі завантажені в неї елементи даних. Кожний запис про елемент даних включає адресу, яку цей елемент даних має в оперативній пам'яті, і управляючу інформацію: ознака модифікації і ознака звернення до даних за деякий останній період часу.

У системах, оснащених кеш-пам'яттю, кожний запит до оперативної пам'яті виконується відповідно такому алгоритму: переглядається вміст кеш-пам'яті з метою визначення, чи не знаходяться потрібні дані в кеш-пам'яті; кеш-пам'ять не є адресованою, тому пошук потрібних даних здійснюється за вмістом - значенням поля "адреса в оперативній пам'яті", взятому із запиту. Якщо дані виявляються в кеш-пам'яті, то вони зчитуються з неї, і результат передається в процесор. Якщо потрібних даних немає, то вони разом зі своєю адресою копіюються з оперативної пам'яті в кеш-пам'ять, і результат виконання запиту передається в процесор. При копіюванні даних може виявитися, що в кеш-пам'яті немає вільного місця, тоді вибираються дані, до яких в останній період було найменше звернень, для витіснення з кеш-пам'яті. Якщо витісняються дані, модифіковані за час знаходження в кеш-пам'яті, то вони переписуються в оперативну пам'ять. Якщо ж ці дані не були модифіковані, то їх місце в кеш-пам'яті оголошується вільним.

На практиці в кеш-пам'ять зчитується не один елемент даних, до якого відбулося звертання, а блок даних, це збільшує ймовірність "попадання в кеш", тобто знаходження потрібних даних у кеш-пам'яті.

У реальних системах імовірність влучення в кеш складає приблизно 0,9. Високе значення ймовірності перебування даних у кеш-пам'яті пов'язано з наявністю у даних об'єктивних властивостей: просторової і часової локальності.

Просторова локальність означає, що коли відбулося звертання за деякою адресою, то з високою ймовірністю найближчим часом відбудеться звертання до сусідніх адрес.

Часова локальність означає, що коли відбулося звертання за деякою адресою, то наступне звернення за цією ж адресою з великою ймовірністю відбудеться найближчим часом.

Всі попередні міркування справедливі і для інших пар запам'ятовуючих пристроїв, наприклад, для оперативної пам'яті і зовнішньої пам'яті. У цьому випадку зменшується середній час доступу до даних, розташованих на диску, а роль кеш-пам'яті виконує буфер в оперативній пам'яті.

Фізично ОП має лінійну організацію і є послідовністю комірок, що адресуються від 0 до N. Ця послідовність ділиться на слова, блоки, сегменти. Розмір ОП визначається в кілобайтах, мегабайтах, гігабайтах, терабайтах і т.д.

Слово - це одиниця обміну ОП з ЦП, яке визначається розрядністю ЦП.

Блок - це безперервна область пам'яті із загальним ключем захисту.

Сегмент - це деяка ділянка пам'яті (для IBM PC від 16 байт до 64Кбайт). Сегмент може містити кілька блоків.

Для ефективного використання ОП необхідно визначити стратегію управління пам'яттю. ОС постійно доводиться вирішувати завдання: коли, куди і за рахунок чого ввести в ОП процес і дані. Існує три стратегії управління ОП:

1. Стратегія вибірки - визначає, коли розмістити в ОП черговий блок програми або даних:

а) вибірка за запитом (на вимогу), коли черговий блок завантажується на вимогу процесу. При такій реалізації неможливо в загальному випадку визначити передачу управління в програмі (налаштування адрес повинно виконуватися після завантаження);

б) упереджувальна вибірка ґрунтується на властивостях послідовного виконання програми і локальності циклів. В даний час найбільш вживана стратегія.

2. Стратегія розміщення, що визначає, куди розміщувати програму, яка надходить:

а) "першу підходящу" ділянку (ефективність за часом розміщення);

б) "найбільш підходящу" ділянку (ефективність за об'ємом);

в) "найменш підходящу" - дивна стратегія з такою аргументацією: після розміщення процесу в більшу вільну ділянку, місце, що залишилося, також велике і може бути достатнім для розміщення ще однієї програми.

3. Стратегія заміщення, що визначає блок/сегмент програми або даних, який можна виштовхнути з ОП для звільнення місця для більш пріоритетних програм (у системах зі свопінгом).

Організація пам'яті тісно пов'язана з потужністю ЦП, об'ємом ОП і режимом мультипрограмування.

Існують 4 види організації реальної пам'яті:

1. **Однопрограмна організація** пам'яті з виділенням безперервній області одному користувачеві. Це найпростіша організація, яка

використовувалася на перших ЕОМ і на мікро-ЕОМ, а також і на перших персональних комп'ютерах (РС) (рис. 4.2).

Рис. 4.2. Організація пам'яті з виділенням безперервній області одному користувачеві

Переваги: простота захисту оперативної пам'яті. Для захисту потрібна пара регістрів, що визначають межі доступу до ОП.

Недоліки: Простоює ЦП, а, отже, і ОП, і пристрої введення/виведення. Неефективне використання ЕОМ навіть за наявності потоку завдань, коли завдання формуються в пакети.

2. **Мультипрограмна організація** з фіксованими розділами. ОП при завантаженні ОС статично розбивається на ряд розділів фіксованого розміру, в кожному з яких може виконуватися одне завдання (рис. 4.3).

Завдання (програма) може розміщуватися в будь-який вільний розділ, розмір якого це допускає. Захист пам'яті здійснюється парою регістрів для кожного розділу ОП.

До переваг мультипрограмної організації з фіксованими розділами відноситься велике завантаження ЦП і підвищення пропускної спроможності.

Серед її недоліків такі:

1) зовнішня фрагментація, яка виражається в недостатності розділів для великих програм, що вимагає перезавантаження ОС для призначення великих розділів;

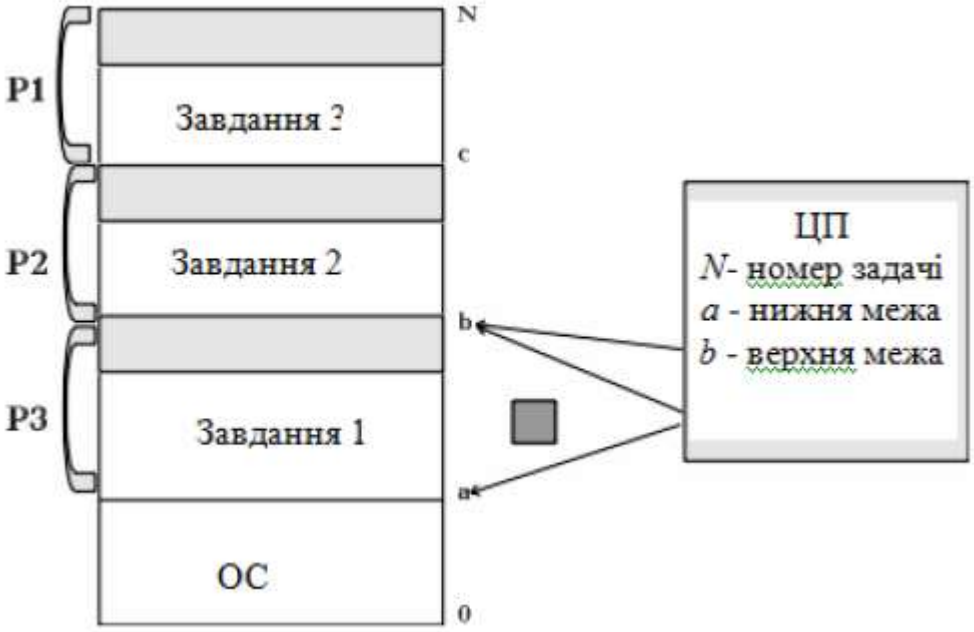


Рис.4.3. Мультипрограмна організація з фіксованими розділами

2) внутрішня фрагментація, при якій сукупна невикористана пам'ять може бути достатньою для виконання програми, але розділена на незв'язні ділянки і не може бути задіяна для розміщення процесів і даних;

3) мультипрограмна організація зі змінними розділами, при якій оперативна пам’ять ділиться динамічно між процесами за запитами завдань (програм) користувачів.

3. Області пам'яті виділяються безперервно з ділянок вільної пам'яті згідно з реалізованими стратегіями розміщення (рис. 4.4). При завершенні

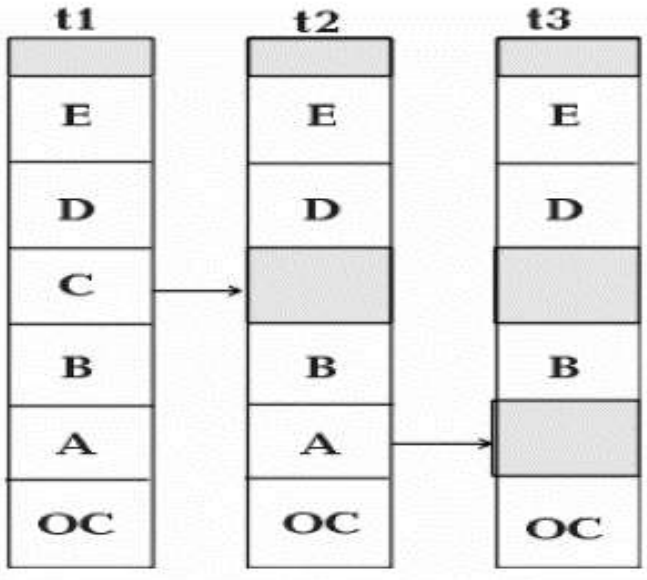


Рис. 4.4. Мультипрограмна організація зі змінними розділами

завдань сусідні вільні ділянки ОП об'єднуються. Захист пам'яті аналогічний режиму з фіксованими розділами.

До переваг такої організації відноситься підвищений рівень мультипрограмування, зникає внутрішня фрагментація (виділяється пам'яті стільки, скільки потрібно).

Серед недоліків варто відмітити зовнішню фрагментацію пам'яті - утворення невикористаних ділянок в цілому може давати великі втрати обсягу і мультипрограмування. Проблема – це сукупність вільних ділянок, достатня для виконання програми, а оскільки вони не суміжні, то не можуть бути використані для розміщення процесів або даних. Ця проблема, зокрема, вирішується методом ущільнення пам'яті (збірки сміття). Така процедура може виконуватися або при кожному завершенні задачі, або тільки тоді, коли для задачі, яка надійшла, немає вільного розділу достатнього розміру. У першому випадку потрібно менше обчислювальної роботи при корегуванні таблиць, а в другому - рідше виконується процедура стиснення. Оскільки програми переміщуються оперативною пам'яттю в ході свого виконання, то перетворення адрес із віртуальної форми подання у фізичну повинно виконуватися динамічно.

Сторінкова організація пам'яті реалізує концепцію незв'язного розподілу пам'яті ЕОМ і означає:

а) поділ ОП ЕОМ на блоки фіксованої довжини розміром від 1 до 4 Kb;

б) поділ адресного простору програми завантажувачем на сторінки фіксованого розміру, рівного довжині блоку;

в) динамічно поблочно видається пам'ять програмам у будь-якому місці ЕОМ;

г) апаратна підтримка відповідності номерів блоків ЗП ЕОМ і сторінок процесу за відповідною таблицею відображення сторінок. Переваги:

1) відсутність зовнішньої фрагментації (достатньо розділів для великих програм);

2) внутрішня фрагментація обмежена розміром блоку ОП ЕОМ <1-4 Кб.

Загальний підсумок розвитку форм організації реальної ОП - практично повна ліквідація фрагментації.

ТЕМА 5. УПРАВЛІННЯ ПРИСТРОЯМИ.

Зовнішні, або периферійні, пристрої ЕОМ діляться на пристрої для введення і виведення інформації і зовнішні запам'ятовуючі пристрої (рис. 5.1).

Рис. 4.3. Класифікація зовнішніх пристроїв ЕОМ

5.1 Зовнішні запам'ятовуючі пристрої

Зовнішні запам'ятовуючі пристрої (ВЗУ) бувають магнітного, оптичного, магнитооптического та електронного принципу дії. ***Накопичувач*** - це пристрій для запису і (або) зчитування інформації з ***носія*** певного типу.

У **накопичувачах на магнітних носіях** використовується магнітний принцип запису, при якому па поверхні, покритої шаром магнітного матеріалу, може фіксуватися два різних стану намагніченості, відповідні записи нуля і одиниці. З'явившись в 1960-х рр., Накопичувачі на магнітних дисках досі залишаються найнеобхіднішими для роботи комп'ютерної системи. Це пояснюється наявністю таких властивостей, як висока швидкість запису та зчитування інформації, висока щільність запису інформації (число біт на одиницю площі носія), можливість багаторазового використання. До недоліків відноситься невисока надійність: записана інформація схильна до впливу електромагнітних полів, а також механічних впливів.

Розрізняються магнітні ВЗУ з послідовним і прямим доступом до інформації. До ВЗУ з ***послідовним доступом*** відносяться накопичувачі на магнітній стрічці, звані ***стримерами.*** Основним недоліком їх є низька швидкість запису / зчитування інформації, характерна для пристроїв з послідовним доступом (для звернення до потрібного блоку інформації необхідно послідовно переглянути всі попередні ділянки носія). До достоїнств відносяться велика ємність (сотні і тисячі гігабайт), низька вартість (це найдешевший в перерахунку на мегабайт спосіб зберігання даних), стабільність роботи. Використовуються стримери для храпения архівних даних і резервного копіювання.

У магнітних ВЗУ з ***прямим доступом*** до інформації звернення до даних здійснюється безпосередньо за адресою інформації, тому може бути

забезпечена висока швидкість запису / зчитування даних. Носієм інформації є жорсткі або гнучкі магнітні диски (накопичувачі на гнучких магнітних дисках є застарілими і мало використовуються).

Накопичувач на жорсткому магнітному диску (НЖМД) - основний пристрій зберігання великих обсягів інформації, яке являє собою один або декілька дисків, захищених жорстким корпусом. У настільних ПК НЖМД форм-фактора 3,5 "розміщуються всередині системного блоку (в ноутбуках форм-фактор 2,5").

Нерідко доводиться чути, як замість терміна "жорсткий диск" використовують поняття "вінчестер", хоча пристрій комп'ютерної пам'яті зовсім не схоже на рушницю. Пояснюється походження увійшов в ужиток терміна аналогією короткого позначення першого жорсткого диска "30/30", створеного в 1973 р, що містив 30 доріжок по 30 секторів у кожній, і гвинтівки калібру 30/30. Ємність першого диска становила 16 Кбайт (уявляєте, як мало?).

Зараз місткість жорстких дисків досягає сотень і тисяч гігабайт. Швидкість НЖМД характеризується швидкістю читання / запису і середнім часом доступу. В цілому ж швидкодію диска найбільшою мірою визначається швидкістю обертання пластин, яка досягає 10 000-15 000 об / хв і більше.

Внутрішні НЖМД підключаються через інтерфейси SATA (замінив IDE і EIDE) і SAS (замінив SCSI).

Крім внутрішніх НЖМД можуть використовуватися зовнішні накопичувачі інформації на жорстких магнітних дисках, які підключаються через інтерфейси USB або FireWire.

RAID-масив - це кілька сполучених в єдиній стійці змінних магнітних дисків. Запис різних блоків одного файлу може йти паралельно на кілька дисків. Крім того, одні й ті ж дані можуть одночасно записуватися на декілька дисків (зеркалирование) для підвищення надійності збереження даних. Існують різні варіанти поєднання паралельного запису і зеркалирования. RAID-масиви використовуються в серверах у випадках, коли необхідно виконувати паралельні запис / зчитування великих потоків даних для безлічі користувачів і забезпечити високу надійність їх зберігання.

У **накопичувачах на оптичних носіях** при запису інформації під впливом оптичного випромінювання змінюється стан окремих ділянок носія, в результаті при зчитуванні на цих ділянках лазерний промінь буде відображатися або поглинатися, що інтерпретується як двійкові нулі чи одиниці. З'явившись наприкінці XX ст., Накопичувачі на оптичних дисках міцно зайняли своє становище серед зовнішніх ЗУ, і тепер жоден комп'ютер не випускається без них. Це можна пояснити такими перевагами оптичних носіїв, як велика ємність, низька питома вартість зберігання інформації, незалежність від електромагнітних полів.

Накопичувачі на компакт-дисках (CD - Compact Disc) використовують змінні носії, які поділяються на компакт-диски тільки для зчитування інформації CD-ROM (Read Only Memory), диски з можливістю однократного запису інформації CD-R (Recordable) і диски з багаторазової записом

інформації CD-RW (Rewritable). Звичайний компакт-диск вміщує інформацію об'ємом 700 Мбайт.

У 1996 р з'явилися DVD-диски (Digital Versatile Disc), зовні схожі на CD, але за рахунок зміни фізичних характеристик і застосування нових технологій дозволяють зберігати значно більший обсяг інформації (4,7 Гбайт в перших моделях). Зараз виробляється кілька видів DVD-дисків, що відрізняються кількістю шарів на носії, причому диски бувають односторонніми і двосторонніми. Наприклад, DVD-9 - односторонній двошаровий диск ємністю 8,5 Гбайт; DVD-18 - двошаровий двосторонній диск ємністю до 17 Гбайт.

Як і компакт-диски, DVD можуть бути тільки для зчитування інформації, з одноразовою і багаторазовою записом інформації. Але на відміну від CD було розроблено декілька різних стандартів: DVD-R, DVD-RW, DVD + RW, DVD-RW, DL-RW, DVD-RAM. Зараз, коли ціни на оптичні дисководи значно знизилися, багато комп'ютерів укомплектовані накопичувачем на оптичному диску, який здатний підтримувати роботу з декількома форматами. Наприклад, запис: CD: R24x / RW16x, DVD: R16x, RW8x, RAM 12x означає, що накопичувач може записувати і зчитувати інформацію з дисків CD-R, CD-RW, DVD-R, DVD-RW DVD-RAM. Швидкість зчитування / запису оптичних дисків вимірюється в умовних одиницях, множник "1x" дорівнює 150 Кбайт / с. Тому виразу "24x", "16x" і т.д. означають, що швидкість накопичувача дорівнює відповідно 24 • 150 Кбайт / с, 16 • 150 Кбайт / с і т.д. При цьому треба мати на увазі, що швидкості запису і зчитування з одного і того ж типу носія зазвичай відрізняються, але швидкість відтворення аудіо-та відеоінформації залишається одноразовою, а характеристики накопичувача відображаються тільки на якості відтворення інформації.

У 2006 р з'явився ще один стандарт оптичних носіїв: BD (Blu-ray Disc - від англ. ***Blue ray*** - синій промінь). За рахунок використання лазера з коротшою довжиною хвилі (синьо-фіолетового кольору, в той час як в DVD використовується червоний, в CD - інфрачервоний лазери) досягається більш висока щільність запису, в результаті чого носії BD мають велику ємність (до 50 Гбайт). Накопичувачі на BD-дисках випускаються тільки для зчитування (BD-ROM), для однократного запису (BD-R) і для багаторазового запису (BD-RE), а також дозволяють виконувати запис і зчитування всіх форматів CD і DVD.

Накопичувачі па оптичних носіях можуть бути внутрішніми і зовнішніми. Внутрішні підключаються через інтерфейси SATA (або IDE) і SCSI, зовнішні - через USB, FireWire або eSATA (external SATA). Зовнішні накопичувачі є більш повільними, але дозволяють проводити підключення і відключення без виключення комп'ютера і перезавантаження операційної системи.

У **накопичувачах на магнітооптичних носіях** поєднуються технології магнітного та оптичного принципів запису: поверхню носія може бути перемагнічена тільки при нагріванні її до високої температури, яке досягається за допомогою лазерного променя. Магнітооптичні носії за рахунок більш високої щільності запису інформації дозволяють зберігати па тій же площі

великі обсяги інформації, ніж магнітні, стійки до зовнішніх впливів, допускають практично необмежене число циклів перезапису.

У ПК накопичувачі на магнітооптичних носіях не знайшли широкого застосування через високу вартість. Застосовуються в основному для резервного копіювання даних і зберігання рідко використовуваної інформації.

Накопичувачі на флеш-пам'яті - SSD (Solid State Drive) являють собою твердотільний накопичувач інформації на основі флеш-мікросхем для довготривалого зберігання даних. На відміну від магнітних накопичувачів флеш-накопичувачі нс мають механічних елементів. Виконані на основі електронних мікросхем флеш-накопичувачі мають ряд переваг: швидке час доступу, висока надійність, компактність (за формою і розмірами нагадують звичайний брелок довжиною 5-7 см), низьке електроспоживання, легкість використання.

Сучасні флеш-накопичувачі здатні вміщати до 500 Гбайт інформації та стають конкурентами НЖМД, проте поки великі по ємності моделі мають дуже високу ціну, а також обмежену кількість циклів читання-запису (порядку 100000), що може бути істотною, якщо використовувати SSD для зберігання операційної системи.

5.2 Пристрої введення-виведення

5.2.1 Пристрої введення інформації

Пристрої введення розділяються на ручні і машинні. До ***ручним пристроям введення інформації*** ставляться клавіатура, миша, диджитайзер.

Клавіатура - основний пристрій ручного введення інформації. Натискання клавіші передає процесору код (умовний номер) натиснутоюклавіші, який в залежності від використовуваної програми може по-різному інтерпретуватися. Тому одна й та ж клавіатура може використовуватися для введення латинських символів, кирилиці, японських, китайських та інших ієрогліфів. Але для цього потрібні відповідні програми. Деякі клавіші самостійний код не формують і зазвичай використовуються спільно з іншими клавішами. Клавіатура підключається через інтерфейси PS / 2 або USB. Бувають і бездротові клавіатури.

В останні роки до звично розташовуваним на клавіатурі буквено-цифровим і функціональним клавішах стали додавати мультимедійні: клавіші для управління програванням музичних компакт-дисків і гучністю; клавіші швидкого виклику додатків; клавіші управління харчування (вхід в "сплячий" режим, повне вимикання); клавіші прямого виклику Інтернету та електронної пошти.

При тривалій роботі з клавіатурою через неправильне положення рук можуть з'явитися болі в зап'ясті, долоні і пальцях, які викликані защемленням нерва в зап'ястному каналі. Щоб цього уникнути, треба стежити за тим, щоб не було ніяких перегинів на ділянці руки, де передпліччя переходить в зап'ясті,

а кут згину в ліктьовому суглобі повинен становити 90 градусів. Для оберігання від хвороби деякі виробники клавіатур пропонують так звані ергономічні клавіатури. До них відносяться розщеплені (split keyboards), що настроюються клавіатури наметового типу (tented keyboards), хвилеподібні (curved) і рельєфні або увігнуті (concave keyboards) клавіатури. Всі вони мають форму, відмінну від прямокутної, і повинні забезпечувати зручне розташування рук при роботі.

Бувають ***повнорозмірні*** (для настільних ПК) і ***зменшені*** (для портативних ПК) клавіатури.

Маніпулятор миша - основне пристрій позиціонування настільних ПК при роботі з графічним режимом відображення даних на моніторі.

При переміщенні миші по поверхні формуються два числа, які передаються процесору і інтерпретуються програмою управління мишею як координати точки двовимірного простору екрану. У результаті програма переміщує зображення покажчика миші по екрану. Натискання клавіш миші викликає передачу процесору коду натиснутої клавіші, який відповідним чином інтерпретується програмою.

Автором миші вважається американський винахідник Дуглас Енгельбарт, який придумав в 1963 р пристрій, дерев'яний корпус якого переміщався на двох перпендикулярно розташованих колесах, що дозволяло маніпулювати об'єктами на площині екрану. Вже пізніше, у 1980-х рр., Були випущені пристрої, схожі на сучасну миша, які поступово стали невід'ємною частиною комп'ютера і на які з клавіатури перейшла частина функцій з управління комп'ютером.

Найчастіше миші підключаються до комп'ютера через шіпу USB, іноді - через порт PS / 2. ПК фірми Apple використовують для підключення шину USB, але частіше - інтерфейс Bluetooth.

За принципом роботи миші діляться на механічні та оптичні. Зараз найбільш поширені оптичні миші, які є більш надійними і точними, ніж механічні. Крім того, існують індукційні миші (використовують спеціальний килимок-планшет) і гіроскопічні (розпізнають рух не тільки на поверхні, але й у просторі).

Як правило, миша має дві кнопки (буває три і навіть п'ять кнопок). Також у багатьох моделях є колесо прокрутки (в сучасних моделях колесо прокрутки за рахунок відхилення вправо і вліво може прокручувати документи не тільки по вертикалі, але і по горизонталі).

У портфельних портативних ПК (ноутбуки, субноутбуки) як миші використовуються ***тачпад*** - сенсорна панель, що дозволяє управляти положенням покажчика за допомогою переміщення пальця по поверхні панелі; ***трекбол*** - вбудований в клавіатуру куля, обертання якого викликає той же ефект, що і переміщення миші. Зараз трекбол пропонується як окремий пристрій переміщення покажчика і являє собою перевернуту мишку механічного або оптичного принципу дії. Для трекбола потрібна менша місце на столі, він більш точно управляє курсором на будь-якій поверхні, але при

кращій в порівнянні з мишею ергономічністю (рука не втомлюється) розташування кнопок на ньому менш зручно і відсутня колесо прокрутки.

Існують бездротові миші, але натомість зручності позбутися обмежує свободу дроти вони (за винятком бездротових Bluetootli-мишей) вимагають підтримувати в робочому стані батарейки або акумулятори харчування, вбудовані в пристрій.

Діджітайзер - пристрій ручного введення графічної інформації в ПК. Являє собою планшет, на якому спеціальним пером малюється зображення. Координати окремих точок малюнка передаються процесору, де обробляються спеціальною програмою.

До ***пристроям автоматичного введення інформації*** відносяться сканер, мікрофон, відеокамера, ТВ-тюнер.

Сканер - пристрій автоматичного введення графічних зображень.

Принцип дії полягає в тому, що окремі елементи носія інформації (паперу, картону, тканини і т.д.) підсвічуються яскравою лампою, фотодіоди фіксують відображення кольору конкретної точки зображення, колір точки певним чином кодується і передається в процесор або основну пам'ять, де обробляється спеціальною програмою. Таким чином, сканер перетворює зображення в послідовність точок різного кольору, яка передається обробної зображення програмою.

Для того щоб вибрати з ліченого сканером зображення символи тексту, використовуються так звані програми розпізнавання символів (OCR - Optical Character Recognition), які на основі складних алгоритмів визначають, які саме букви, цифри та спеціальні знаки закодовані конкретним набором точок. Пізнані символи записуються у форматі, в якому вони можуть оброблятися програмами обробки текстів.

За способом зчитування інформації сканери діляться на ручні і настільні.

У ***ручних*** сканерах користувач повинен акуратно переміщати скануючу головку по сканується документом, тому якість одержуваного комп'ютерного документа залежить від рівномірності переміщення. Ручні сканери дешевше, більш компактні, дозволяють відсканувати книгу, не порушуючи її цілісність, але використовуються рідше, ніж настільні. Різновид ручних сканерів часто використовується для сканування штрих-кодів.

Настільні сканери діляться на планшетні, рулонні і проекційні. Найбільш поширені ***планшетні*** сканери, які зараз стали звичайним елементом і домашньою, та офісної комп'ютерної системи. Вони являють собою плоске настільний пристрій. На скло, під яким знаходиться блок сканера, поміщається сканований документ і притискається кришкою. Скануючий блок автоматично переміщається уздовж скануєте.

У рулонних сканерах аркуші документів протягаються через ***зчитувальний*** пристрій сканера. Вони використовуються для картографії, художнього друку, при широкоформатного друку.

Проекційні сканери являють собою своєрідний проекційний апарат і можуть випускатися для роботи з непрозорими і прозорими зображеннями (слайд- сканери).

Основними характеристиками сканерів є якість прочитування інформації, яке визначається роздільною здатністю (кількість точок на одиницю довжини) і числом градацій кольору; швидкість сканування; максимальний розмір відсканованого зображення.

Сканери підключаються до ПК через шину USB, паралельний порт або інтерфейс SCSI.

Мікрофон потрібен для передачі в комп'ютер аудіоінформації і за наявності спеціального програмного забезпечення може використовуватися для телеконференцій, спілкування через програму Skype. Під'єднуються до ПК за допомогою гнізда в звуковій платі.

Розрізняються мікрофони настільні, підвісні, у вигляді петлички, гарнітури (сукупність мікрофона з навушниками) і т.д.

Мікрофони можуть бути безпровідними, але вони мають значно більш високу вартість.

Цифрова фотокамера - фотоапарат, що записує зображення в цифровому форматі у власну пам'ять. Для подальшої обробки зображення передається в ПК.

ТВ-тюнер - пристрій, що дозволяє переглядати телевізійне зображення па екрані монітора. Багато ТВ-тюнери дозволяють захоплювати і записувати на диск фрагменти трансляції або відеозаписів, які потім можна редагувати па ПК.

Пристрої виведення інформації включають відеосистему, аудіосистему, друкуючі пристрої.

Відеосистема ПК включає монітор і відеоадаптер.

Монітор (дисплей) - пристрій, що забезпечує висновок динамічно оновлюваного зображення.

Відеоадаптер - дочірня плата, що забезпечує формування та виведення зображення на екран монітора.

Відеосистема ПК може працювати в ***текстовому*** або ***графічному*** режимах. У текстовому режимі на екрані монітора може відображатися тільки обмежений набір символів і спеціальних знаків. Стандартно, в текстовому режимі відображаються 25 рядків по 80 символів у кожному рядку.

У графічному режимі зображення формується з окремих точок. Формовані відеосистемою комп'ютера в ***графічному*** режимі зображення характеризуються наступними параметрами:

- дозволом;
- числом кольорів;
- частотою оновлення зображення.

Дозвіл - число точок, що відображаються на екрані по горизонталі і вертикалі при формуванні зображення.

Стандартні режими дозволу: 800 × 600,1024 × 768, 1280 × 800, 1280 × 1024, 1400 × 1050 та ін. Професійні монітори мають і більш високі дозволу.

Число кольорів, відтворюваних монітором, визначається кількістю двійкових розрядів, використовуваних для запису однієї точки (пікселя). Застосовуються такі стандарти відображення кольору:

- 16 кольорів - 4 біти;
- 256 кольорів - 8 біт;
- 65536 кольорів - 16 біт (High Color);
- 16777216 кольорів - 24 біта (True Color).

Частота оновлення зображення характеризує, скільки разів в секунду оновлюється зображення на екрані монітора. Вимірюється в герцах.

В даний час монітори на електронно-променевих трубках (CRT-монітори) практично витіснені жілкокрісталліческімі (LCD) моніторами. Застосування CRT-моніторів залишається при вирішенні спеціалізованих завдань (в системах відеоспостереження, касових апаратах і ін.).

У LCD-моніторах кожна точка формується світінням одного елемента екрана. Для LCD-моніторів частота оновлення екрану нс є суттєвою характеристикою, і зображення зазвичай виглядає стабільним навіть при низькій частоті оновлення (60 Гц).

Достоїнствами LCD-моніторів є компактні розміри, відсутність спотворень, стабільність зображення, хороша яскравість і контрастність зображення, низьке енергоспоживання. Недоліками є найгірша, ніж у CRT-моніторів, передача кольору, гірше відтворення швидко рухомих об'єктів.

Важливим параметром LCD-моніторів є тип рідкокристалічної матриці. Основними при виготовленні LCD-моніторів є матриці TN, IPS, MVA. Великого поширення набули монітори з матрицею типу TN, вони є найбільш дешевими, але мають неточну передачу кольору і низьку контрастність. Дорожчі монітори з матрицею типу IPS забезпечують значно кращу передачу кольору і рівномірність підсвічування, але гірший час відгуку (час, що витрачається на перехід від перемикання точки екрану з чистого чорного кольору в чистий білий і назад). Матриці MVA є проміжними за параметрами і вартості між TN і IPS. Існують й інші варіанти матриць.

Стандартно, монітори мають довжину діагоналі 14, 15, 17, 19, 20, 21, 22, 24 дюйма. Для роботи з сучасними програмами рекомендується використовувати монітори з довжиною діагоналі 17-21 дюйм.

Існують моделі LCD-моніторів, в які вбудовані звукові колонки, в інші вбудовані ТВ-тюнери, що дозволяють використовувати монітор як телевізор.

Перспективними моделями моніторів є плазмові панелі. Вони мають багато переваг у порівнянні з LCD-моніторами, але поки дуже дорогі і використовуються тільки в якості великих якісних проекційних екранів і в складі домашніх кінотеатрів високого класу.

Друкуючі пристрої - це пристрої, що забезпечують виведення зображень на тверді носії (папір, стрічку, тканину і т.д.). До них відносяться принтери та графопостроители (плоттери).

Принтер формує зображення порядково. Папір чи інший носій послідовно протягується під друкуючими голівками які відображають умовну рядок зображення.

Практично всі сучасні принтери формують зображення з окремих точок. Кожен друкований символ відображається як певна сукупність окремих точок. Принцип формування точок зображення у різних типів принтерів розрізняється. Різниться і щільність точок на одиницю поверхні. Чим менше розмір точки, тим вище щільність точок і чіткіше зображення. Щільність точок вимірюється в dpi (dots per inch, число точок на квадратний дюйм зображення). Чим вище значення показника dpi у принтера, тим він кращий.

Сучасні принтери підключаються до системного блоку через паралельний порт або шину USB.

За способом друку принтери поділяються на струменеві, лазерні, ударні, термічні та спеціальні.

Струменеві принтери формують точки зображення, виплескуючи мікроскопічні краплі спеціального чорнила на папір. Кожна крапля - одна точка зображення.

Практично всі сучасні струменеві принтери підтримують функцію кольорового друку. Кольорові струменеві принтери формують точку зображення, виплескуючи кілька крапель базових кольорів в одну точку. За рахунок змішування базових кольорів одержують потрібний похідний колір. В принципі, достатньо трьох базових кольорів (червоний, зелений, синій) для формування будь-якого похідного кольору.

Найбільш прості і дешеві принтери формують колір саме з трьох базових кольорів. У більш досконалих моделях використовується більше число базових кольорів. Наприклад, часто крім базових кольорів використовується чорний колір.

У простих моделях для чорно-білого друку використовують картридж з чорним чорнилом, а для кольорового друку його треба замінити на картридж з трьома чорнильницями, що містять базові кольори. Чорний колір при кольорового друку виходить брудно-сірим.

У більш досконалих моделях одночасно використовуються два картриджа: один з чорною фарбою, а інший - з чорнилом базових кольорів. У ще більш досконалих принтерах використовується більше двох картриджів (друкуючих головок).

Для якісного друку кольорових зображень на кольорових принтерах треба використовувати спеціальну (досить дорогу) папір. Вже зараз технологія така, що якість друку на спеціальному папері близько до якості фотографії.

Достоїнствами струменевих принтерів є те, що вони недорогі і забезпечують кольоровий друк. Недоліком є дорогі витратні матеріали (картриджі, спеціальний папір).

Лазерні принтери формують точки зображення, нагріваючи лазером або лінійкою світлодіодів найдрібніші порошинки спеціального порошку - ***тонера.*** Там, де повинна бути виведена точка зображення, лазер включається, нагріває тонер, і частка нагрітого тонера друкується на папері.

Технічно вони є більш складними пристроями, ніж струменеві принтери. Мають свій потужний процесор і пам'ять.

Достоїнствами лазерних принтерів є:

• швидка друк текстів і зображень, що складаються з елементів, що відображаються відтінками сірого кольору;

• чітка якісний друк навіть на звичайному папері;

• менш дорогі витратні матеріали, ніж для струменевих принтерів.

Основний недолік - вища ціна, ніж у струменевих. Існуючі кольорові лазерні принтери забезпечують високу якість друку, але дуже дороги.

Ударні принтери формують точки, виштовхуючи тонкий стрижень з друкуючої головки. Він, б'ючи по красящей стрічці, залишає відбиток точки на папері.

Це найстаріша різновид принтерів. В даний час використовуються тільки в спеціальних цілях (друк чеків в касових апаратах, банкоматах і т.д.). У матричних принтерів розмір точок зображення істотно більше, ніж у струменевих і лазерних, тому істотно гірше якість друку. Основною перевагою є низька вартість витратних матеріалів.

Термічні принтери використовують нагрівання барвника і перенесення його на папір в рідкій або газоподібній формі. При охолодженні барвник застигає на папері, формуючи зображення. Термічні принтери дозволяють друкувати високоякісні кольорові зображення фотографічної якості.

Спеціалізовані принтери є частиною різних технічних пристроїв і призначені для друку не тільки на папері, а й на інших носіях - картоні, тканині, металі та ін.

Графобудівник - пристрій для викреслювання складних зображень. Бувають ***планшетні*** і ***рулонні*** графопостроители.

У планшетних папір закріплюється на великому планшеті і малює перо переміщається вгору-вниз і вправо-вліво. У рулонних перо переміщається тільки вправо-вліво, а лист простягається уздовж лінії пера.

Графобудівники - досить дорогі пристрої і застосовуються тільки в спеціальних цілях - для викреслювання складних креслень, широкоформатних плакатів і т.д.

Аудіосистема ПК призначена для виведення звукової інформації; для цього використовуються аудиоадаптер і акустична система (динаміки з підсилювачем низької частоти, навушники). Також в акустичну систему входить пристрій введення звукової інформації - мікрофон.

Аудиоадаптер - дочірня плата, що забезпечує перетворення цифрових даних в аналогові і назад для вводу / виводу звуку за допомогою ПК; має вихід для передачі звукового сигналу на підсилювач і вхід для введення звукового сигналу з зовнішнього джерела в ПК для подальшої обробки. Дорогі аудіоадаптер мають кілька входів і виходів.

Аудіоадаптер розрізняються способами синтезу звуку і наявністю / відсутністю мікросхем створення додаткових звукових ефектів (перетворення звуку, об'ємний 3D-звук і т.д.). За допомогою аудіосистеми ПК можна відтворювати звичайні аудіо-CD, формати MP3 і WMA

5.2.2 Пристрої введення-виведення в зовнішнє середовище

Їх представником є модем.

Модем - пристрій для перетворення даних з цифрового формату в аналоговий і назад для передачі їх по телефонних лініях.

Розрізняють ***внутрішні*** і ***зовнішні*** модеми. Внутрішні модеми виконані у вигляді дочірньої плати і встановлюються в PCI-роз'єм на материнській платі, а зовнішні підключаються до ПК через послідовний порт або шину USB.

Сучасні модеми можуть передавати дані зі швидкістю 56000 біт / с, а деякі - 76800 біт / с.

Більшість модемів дозволяють передавати факси, багато можуть використовуватися як телефон, виступати в якості автовідповідача.

5.2.3 Блокові і символьні пристрої

Типовий приклад блокового пристрою - пристрій керування дисками. Він виконує команди виду: read, write, seek (прочитати, записати або знайти блок із заданим номером). Пристрій може виконувати чисте введення-виведення або доступ до файлової системи. Є можливість доступу до файлу, псевдонімом в пам'ять.

Типові приклади символьних пристроїв - клавіатура, миша, послідовні порти. Такі пристрої виконують команди виду: get, put (вважати або записати символ). Бібліотеки верхнього рівня в операційній системі для символьних пристроїв допускають порядкове редагування по символьний введеної інформації.

5.3 Фізична організація пристроїв вводу-виводу

Однією з головних функцій ОС є управління усіма пристроями вводу-виводу комп’ютера. ОС повинна передавати пристроям команди, перехоплювати переривання і обробляти помилки; вона також повинна забезпечувати інтерфейс між пристроями і іншою частиною системи.

Пристрої вводу-виводу діляться на два типи: блок-орієнтовані пристрої і байт-орієнтовані пристрої. Блок-орієнтовані пристрої зберігають інформацію в блоках фіксованого розміру, кожен з яких має свою власну адресу. Найпоширеніший блок-орієнтований пристрій – диск. Байт-орієнтовані пристрої не адресовані і не дозволяють проводити операцію пошуку, вони генерують або зчитують послідовність байтів. Прикладами є принтери, мережеві адаптери.

Зовнішній пристрій зазвичай складається з механічного і електронного компонента. Електронний компонент називається контролером пристрою або адаптером. Механічний компонент являє власне пристрій. Деякі контролери можуть управляти декількома пристроями. Якщо інтерфейс між контролером і пристроєм стандартизований, то незалежні виробники можуть випускати сумісні як контролери, так і пристрої.

Операційна система зазвичай має справу не з пристроєм, а з контролером. Контролер, як правило, виконує прості функції, наприклад,

перетворює потік біт в блоки, що складаються з байт і здійснює контроль і виправлення помилок. Кожен контролер має декілька регістрів, які використовуються для взаємодії з центральним процесором. У деяких комп'ютерах ці регістри є частиною фізичного адресного простору. У таких комп'ютерах немає спеціальних операцій вводу-виводу. У інших комп'ютерах адреси регістрів вводу-виводу, званих часто портами, утворюють власний адресний простір за рахунок вступу спеціальних операцій вводу-виводу.

ОС виконує ввід-вивід, записуючи команди в регістри контролера. Наприклад, контролер зовнішнього диску приймає 15 команд, таких як READ, WRITE, FORMAT і так далі. Коли команда прийнята, процесор залишає контролер і займається іншою роботою. При завершенні команди контролер організовує переривання для того, щоб передати управління процесором операційній системі, яка повинна перевірити результати операції. Процесор отримує результати і статус пристрою, читаючи інформацію з регістрів контролера.

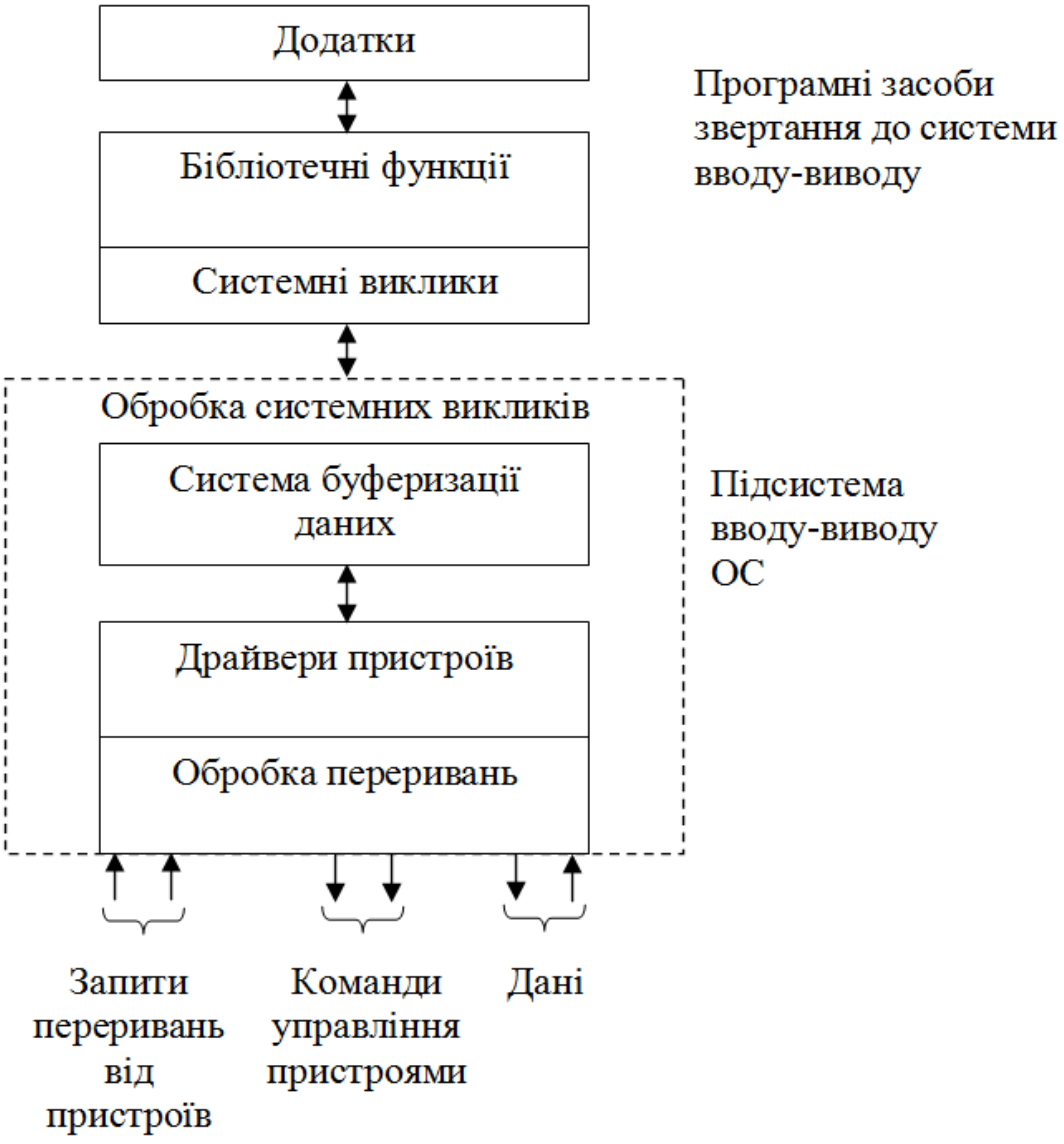


Рис. 5.2 Багаторівнева організація підсистеми вводу-виводу

5.4 Організація програмного забезпечення вводу-виводу

Основна ідея організації програмного забезпечення вводу-виводу полягає в розбитті його на декілька рівнів, причому нижні рівні забезпечують екранування особливостей апаратури від верхніх, а ті, у свою чергу, забезпечують зручний інтерфейс для користувачів.

Ключовим принципом є незалежність від пристроїв. Вид програми не повинен залежати від того, чи читає вона дані з оптичного диска або з жорсткого диска.

Важливим питанням для програмного забезпечення вводу-виводу є обробка помилок. В загальному, помилки слід обробляти як можна ближче до апаратури. Якщо контролер виявляє помилку читання, то він повинен спробувати її скоректувати. Якщо ж це йому не вдається, то виправленням помилок повинен зайнятися драйвер пристрою. І тільки якщо нижній рівень не може впоратися з помилкою, він повідомляє про помилку верхній рівень.

Ще одне ключове питання – це використання блокуючих (синхронних) і неблокуючих (асинхронних) передач. Більшість операцій фізичного вводу-виводу виконуються асинхронно – процесор починає передачу і переходить на іншу роботу, поки не настає переривання. Програми користувача набагато легше писати, якщо операції вводу-виводу блокуючі – після команди READ програма автоматично припиняється до тих пір, поки дані не потраплять в буфер програми. ОС виконує операції вводу-виводу асинхронно, але представляє їх для програм користувача в синхронній формі.

Остання проблема полягає в тому, що одні пристрої є поділюваними, а інші – виділеними. Диски – це поділювані пристрої, оскільки одночасний доступ декількох користувачів до диску не є проблемою. Принтери – це виділені пристрої, тому що не можна змішувати дані, що друкуються різними користувачами. Наявність виділених пристроїв створює для операційної системи деякі проблеми.

Для вирішення поставлених проблем доцільно розділити програмне забезпечення вводу-виводу на чотири шари (рис. 5.2):

- Обробка переривань,
- Драйвери пристроїв,
- Незалежний від пристроїв шар операційної системи,
- Користувацький шар програмного забезпечення.

5.5 Обробка переривань

Переривання мають бути приховані як можна глибше в надрах операційної системи, щоб як можна менша частина ОС мала з ними справу. Найкращий спосіб полягає в дозволі процесу, що ініціював операцію вводу-виводу, блокувати себе до завершення операції і настання переривання. Процес може блокувати себе, використовуючи, наприклад, виклик DOWN для семафора, або виклик WAIT для змінної умови, або виклик RECEIVE для очікування повідомлення. При настанні переривання процедура обробки

переривання виконує розблокування процесу, що ініціював операцію вводу-виводу, використовуючи виклики UP, SIGNAL або посилаючи процесу повідомлення. У будь-якому випадку ефект від переривання полягатиме в тому, що раніше заблокований процес тепер продовжить своє виконання.

5.6 Драйвери пристроїв

Увесь залежний від пристрою код поміщається в драйвер пристрою. Кожен драйвер управляє пристроями одного типу або, можливо, одного класу.

У ОС тільки драйвер пристрою знає про конкретні особливості певного пристрою. Наприклад, тільки драйвер диска має справу з доріжками, секторами, циліндрами і іншими чинниками, що забезпечують правильну роботу диска.

Драйвер пристрою приймає запит від пристроїв програмного шару і вирішує, як його виконати. Типовим запитом є читання n блоків даних. Якщо драйвер був вільний під час надходження запиту, то він починає виконувати запит негайно. Якщо ж він був зайнятий обслуговуванням іншого запиту, то запит, що поступив, приєднується до черги вже наявних запитів, і він буде виконаний, коли наступить його черга.

Перший крок в реалізації запиту вводу-виводу полягає в перетворенні його з абстрактної форми в конкретну. Наприклад, для дискового драйвера це означає перетворення номерів блоків в номери циліндрів, головок, секторів, перевірку, чи працює мотор, чи знаходиться головка над потрібним циліндром. Отже, він повинен вирішити, які операції контролера треба виконати і в якій послідовності.

Після передачі команди контролеру драйвер повинен вирішити, чи блокувати себе до закінчення заданої операції або ні. Якщо операція займає значний час, як при друці деякого блоку даних, то драйвер блокується до тих пір, поки операція не завершиться, і обробник переривання не розблоковує його. Якщо команда вводу-виводу виконується швидко (наприклад, прокрутка екрану), то драйвер чекає її завершення без блокування.

5.7 Незалежний від пристроїв шар операційної системи

Велика частина програмного забезпечення вводу-виводу є незалежною від пристроїв. Точна межа між драйверами і незалежними від пристроїв програмами визначається системою, оскільки деякі функції, які могли б бути реалізовані незалежним способом, насправді виконані у вигляді драйверів для підвищення ефективності або з інших причин.

Типовими функціями для незалежного від пристроїв шару є:

- забезпечення загального інтерфейсу до драйверів пристроїв,
- іменування пристроїв,
- захист пристроїв,
- забезпечення незалежного розміру блоку,

- буферизація,
- розподіл пам’яті на блок-орієнтованих пристроях,
- розподіл і звільнення виділених пристроїв,
- повідомлення про помилки.

Верхнім шарам програмного забезпечення не зручно працювати з блоками різної величини, тому цей шар забезпечує єдиний розмір блоку, наприклад, за рахунок об’єднання декількох різних блоків в єдиний логічний блок. У зв’язку з цим верхні рівні мають справу з абстрактними пристроями, які використовують єдиний розмір логічного блоку незалежно від розміру фізичного сектора.

При створенні файлу або заповненні його новими даними необхідно виділити йому нові блоки. Для цього ОС повинна вести список або бітову карту вільних блоків диска. На підставі інформації про наявність вільного місця на диску може бути розроблений алгоритм пошуку вільного блоку, реалізований програмним шаром, що знаходиться вище шару драйверів.

5.8 Користувацький шар програмного забезпечення

Хоча велика частина програмного забезпечення вводу-виводу знаходиться усередині ОС, деяка його частина міститься в бібліотеках, що зв’язуються з програмами користувача. Системні виклики, що включають виклики вводу-виводу, зазвичай робляться бібліотечними процедурами. Стандартна бібліотека вводу-виводу містить велике число процедур, які виконують ввід-вивід і працюють як частина програми користувача.

Іншою категорією програмного забезпечення вводу-виводу є підсистема спулінгу (spooling). **Спулінг** – це спосіб роботи з виділеними пристроями в мультипрограмній системі. Розглянемо типовий пристрій, що вимагає спулінгу – принтер. Хоча технічно легко дозволити кожному процесу користувача відкрити спеціальний файл, пов’язаний з прінтером, такий спосіб небезпечний через те, що процес користувача може монополізувати прінтер на довільний час. Замість цього створюється спеціальний процес – монітор, який отримує виняткові права на використання цього пристрою. Також створюється спеціальний каталог (каталогом спулінгу). Для того, щоб надрукувати файл, процес користувача поміщає інформацію, що виводиться, в цей файл і поміщає його в каталог спулінгу. Процес-монітор по черзі роздруковує усі файли, що містяться в каталозі спулінгу.

ТЕМА 6. УПРАВЛІННЯ ІНФОРМАЦІЄЮ.

6.1 Мета і завдання файлової системи

Файл – це іменована область зовнішньої пам'яті, в яку можна записувати й з якої можна зчитувати дані. Звичайно файли зберігаються в

енергонезалежній пам'яті – диску. Однак є й виключення. Основні цілі використання файлу:

- довгострокове й надійне зберігання інформації;
- спільне використання інформації.

Файл може бути створений одним користувачем, а використовуватися іншим. При цьому можуть бути визначені права доступу до інформації.

Файлова система – частина ОС, що включає:

- сукупність всіх файлів на диску;
- набори структур даних для керування файлами: каталоги файлів, дескриптори файлів, таблиці розподілу вільного й зайнятого простору на диску;
- комплекс системних програмних засобів, що реалізують операції над файлами: створення, знищення, запис, читання, іменування й пошук файлів.

Файлова система дозволяє обходитися набором простих операцій над деяким абстрактним об'єктом, який названо файлом.

Файлова система екранує всі складності фізичної організації довгострокового зберігання даних і надає набір зручних у використанні команд для маніпулювання файлами.

Завдання, які вирішує ФС, залежать від способу організації обчислювального процесу в цілому. Найпростіший тип ФС – однокористувацька, однопрограмна. До їхнього числа належить MS-DOS.

Її функції наступні:

- іменування файлів;
- програмний інтерфейс для додатків;
- відображення логічної моделі файлової системи на фізичну організацію сховища даних;
- забезпечення стійкості файлової системи до збоїв живлення, помилок апаратних і програмних засобів.

Завдання ФС ускладнюються в однокористувальницьких мультипрограмних ОС. Прикладом такої ОС є OS/2. Тут додається нове завдання – спільний доступ до файлу з декількох процесів.

У цьому випадку, файл – поділюваний ресурс із усіма проблемами, що звідси з’являються.

У багатокористувацьких системах додається ще одне завдання – захист файлів від несанкціонованого доступу.

6.2 Логічна модель файлової системи

Логічна модель файлової системи матеріалізується у вигляді: дерева каталогів, що виводиться на екран, наприклад, за допомогою Norton Commander або Windows Explorer, у символьних складених іменах файлів і командах роботи з файлами.

Типи файлів

Звичайні файли містять інформацію довільного характеру. Більшість сучасних ОС (UNIX, Windows, OS/2) ніяк не обмежують і не контролюють вміст і структуру файлу.

Всі ОС повинні розпізнавати хоча б один тип файлів – їх власні виконувані файли.

Каталоги – особливий тип файлів. Вони містять системну довідкову інформацію про набір файлів, згрупованих користувачами за якоюсь неформальною ознакою (наприклад, один документ і т.п.).

У багатьох ОС у каталог можуть входити спеціальні файли – це фіктивні файли, що асоціюються із пристроями вводу–виводу. Вони використовуються з метою уніфікації механізму доступу до файлів і зовнішніх пристроїв. Спеціальні файли дозволяють користувачеві виконувати операції введення-виведення за допомогою звичайних команд запису у файл або читання з файлу. Ці команди обробляються спочатку програмами файлової системи, а потім на деякому етапі виконання запиту перетворюються ОС у команди керування відповідним пристроєм.

6.3 Ієрархічна структура файлової системи.

Більшість файлових систем має ієрархічну структуру, у якій рівні створюються за рахунок того, що каталог більш низького рівня може входити в каталог більш високого рівня (рис. 6.1).

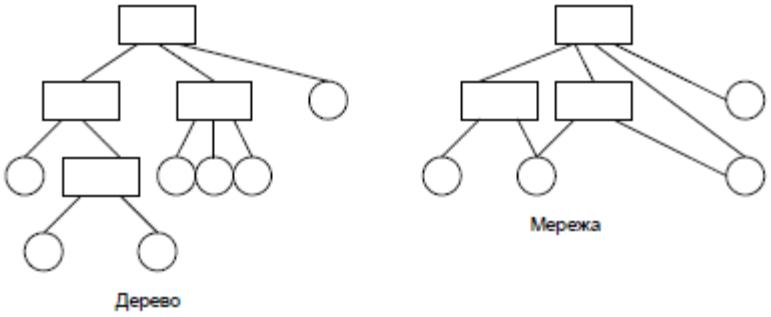


Рис. 6.1. Графи ієрархії каталогів

Граф, описуючий ієрархію каталогів, може бути деревом або мережею. Якщо файлу дозволено входити тільки в один каталог, файли утворять дерево. Якщо мережа – файл може входити відразу у декілька каталогів. Наприклад, в MS DOS й Windows каталоги утворюють деревоподібну структуру, а в UNIX – мережну. У деревоподібній структурі кожен файл є листом. каталог самого верхнього рівня називається кореневим каталогом або коренем.

6.4 Імена файлів

У файлових системах використовується три типи імен файлів: прості, складені й відносні.

Просте (коротке, символьне) ім'я ідентифікує файл у межах одного каталогу. Ці імена привласнюють користувачі з урахуванням обмежень ОС. Максимальна (у файловій системі FAT) довжина імені обмежується схемою 8.3 (ім'я – 8 символів, розширення – 3), а у файлових системах NTFS й FAT32, що входять до складу ОС Windows NT, ім'я файлу може містити до 255 символів.

В ієрархічних файлових системах різним файлам дозволено мати однакові прості символьні імена за умови, що вони належать різним каталогам.

Для однозначної ідентифікації в таких системах використовується так зване повне ім'я.

Повне ім'я являє собою ланцюжок, який є шляхом від кореня до даного файлу.

У деревоподібній файловій системі між файлом і його повним ім'ям є взаємно однозначна відповідність один файл – одне повне ім'я.

У випадку мережної структури має місце відповідність: один файл – багато повних імен.

Файл може бути також ідентифікований відносним ім'ям. Воно утвориться через поняття поточного каталогу. ОС фіксує ім'я поточного каталогу й використовує його як «добавку» до повного імені, використовуючи відносне ім'я. Наприклад, поточний каталог – USER, а відносне ім'я – main.exe. Повне ім'я – USER/main.exe.

6.5 Монтування

Обчислювальна система може мати кілька дискових пристроїв. Більш того, один фізичний пристрій може мати кілька логічних дисків. Виникає проблема зберігання файлів у системі, що має кілька пристроїв зовнішньої пам'яті.

Перше рішення. На кожному із пристроїв розміщається автономна файлова система. Тобто є два незалежних дерева каталогів. Тут у повне ім'я файлу входить ідентифікатор відповідного логічного диску.

Друге рішення. Файлові системи поєднуються в єдину файлову систему, що описується єдиним деревом каталогів.

Така операція називають **монтуванням.**

При цьому ОС виділяє один дисковий пристрій, який називають системним. Нехай є дві файлові системи, розташовані на різних логічних дисках, причому один з них є системним. Файлова система, що розташована на системному диску, призначається кореневою. Для зв'язку ієрархій файлів у кореневій файловій системі вибирається деякий існуючий каталог. Після виконання монтування обраний каталог стає кореневим каталогом другої

файлової системи. Через цей каталог файлова система, що монтують, приєднується як піддерево до загального дерева (рис. 6.2).

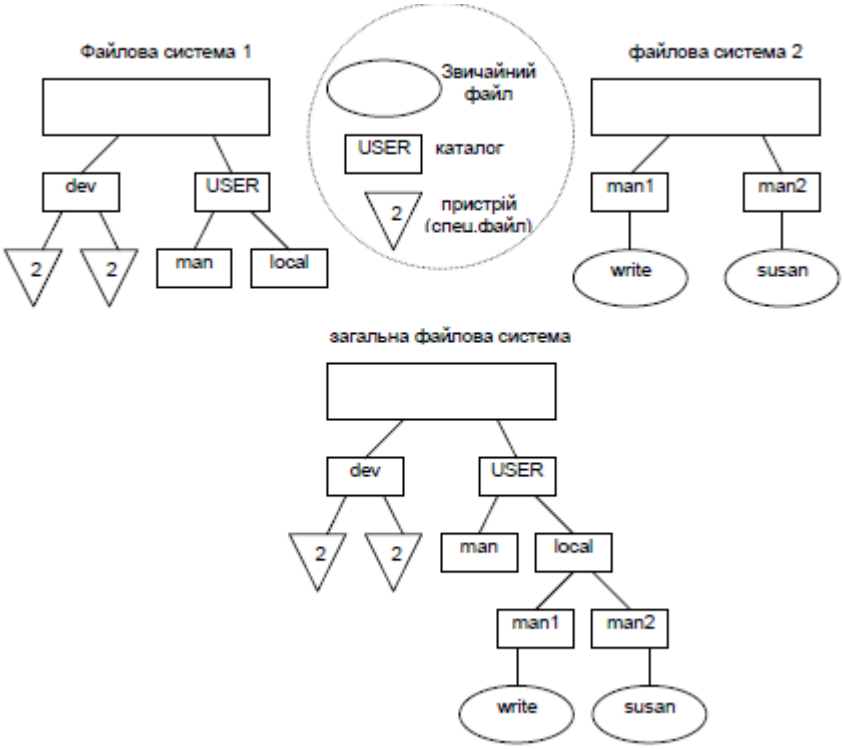


Рис. 6.2. Приклад монтування

6.6 Атрибути файлів

Атрибути – інформація, що описує властивості файлу.
Можливі атрибути:
- тип файлу (звичайний файл, каталог, спеціальний файл);
- власник файлу;
- творець файлу;
- пароль для доступу до файлу;
- інформація про дозволені операції доступу до файлу;
- часи створення, останнього доступу й останньої зміни;
- поточний розмір файлу;
- максимальний розмір файлу;
- ознака «тільки для читання»;
- ознака «прихований файл»;
- ознака «системний файл»;
- ознака «архівний файл»;
- ознака «двійковий/символьний»;
- ознака «тимчасовий» (видалити по завершенні процесу);
- ознака блокування; - ознака запису у файл;
- покажчик на ключове поле в записі;
- довжина ключа.

Конкретний перелік атрибутів визначається специфікою файлової системи. Наприлад, в MS-DOS зроблено як на рис.6.3.

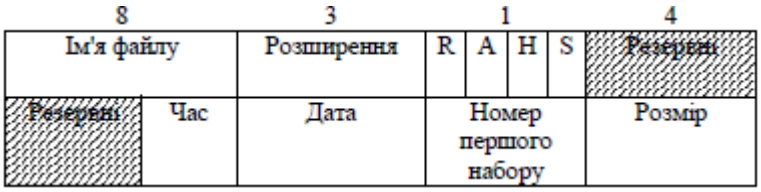


Рис.6.3. Перелік атрибутів файлів

Іншим варіантом є розміщення атрибутів у спеціальних таблицях, коли в каталогах містяться тільки посилання на ці таблиці. У такій файловій системі (ОС UNIX) структура каталогу дуже проста (рис. 6.4).

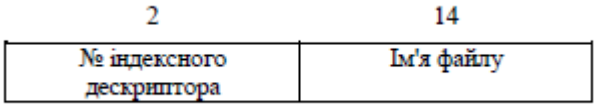


Рис. 10.4. Індексний дескриптор файлу

Індексний дескриптор файлу – таблиця, у якій зосереджені значення атрибутів файлу. Така система більш гнучка. Файл може бути включений відразу в кілька каталогів.

6.7 **Логічна організація файлу**

Ознаками, що відокремлюють один структурний елемент від іншого, можуть служити певні кодові послідовності або просто відомі програмі значення зсувів цих структурних елементів відносно початку файлу. Підтримка структури даних може бути або цілком покладена на програму, або, в тій або іншій мірі, цю роботу може брати на себе файлова система.

У першому випадку, файл представляється ФС як неструктурована послідовність даних. Програма формулює запити до файлової системи на введення-виведення, використовуючи загальні для всіх програм системні засоби, наприклад, указуючи зсув від початку файлу й кількість байт, які необхідно прочитати або записати. Потік байт, що надійшов до програми, інтерпретується відповідно до закладеної в програмі логіки.

Модель файлу, відповідно до якої його вміст представляється неструктурованою послідовністю (потоком) байт, широко використовується у більшості сучасних ОС (MS-DOS, Windows NT/2000). Неструктурована модель файлу дозволяє легко організувати поділ файлу між декількома програмами: різні додатки можуть по-своєму структурувати й інтерпретуватидані, що містяться у файлі.

Інша модель файлу – структурований файл. Ця модель застосовувалася раніше (в OS/360). Підтримка структури файлу забезпечується файловою системою. Вона бачить файл як упорядковану послідовність логічних записів.

Програма може звертатися до ФС із запитами на введення-виведення на рівні записів. Наприклад, зчитавши запис 25 з файлу FILE.DOC, ФС має інформацію про структуру файлу, достатню для виділення будь-якого запису. ФС надає додатку доступ до запису, а вся подальша обробка даних, що міститься в цьому записі, виконується програмою.

Розвитком цього є системи, що підтримують складні структури даних і взаємозв'язок між ними.

Логічний запис –- найменший елемент даних, яким оперує програміст при обміні із зовнішнім пристроєм. ФС може забезпечувати два способи доступу до логічних записів:

- послідовний доступ;
- прямий доступ (позиціонування на конкретний запис файлу).

6.8 Фізична організація файлової системи

Подання про файлову систему, як про ієрархічно організовану множину інформаційних об'єктів, має мало спільного з фактичним порядком зберігання файлів на диску.

Файл може бути розкиданий «шматочками» по всьому диску. Логічно об'єднані файли з одного каталогу зовсім не обов'язково є сусідніми на диску.

Принципи розміщення файлів, каталогів і системної інформації на реальному пристрої описуються фізичною організацією файлової системи. Основним носієм є жорсткий диск, що складається з пакету пластин.

На кожній стороні пластини розміщені тонкі концентричні кільця – доріжки (tracks), на яких зберігаються дані, Кількість доріжок залежить від типу диска. Нумерація доріжок починається з 0 від зовнішнього краю до центру. Коли диск обертається, головка зчитує дані з магнітної доріжки або записує їх на доріжку дискретними кроками. Кожен крок зсувається на 1 доріжку. У деяких дисках замість однієї головки є по головці на кожну доріжку.

Сукупність доріжок одного радіусу на всіх поверхнях всіх пластин називається циліндром (cylinder). Кожна доріжка розбивається на фрагменти, називані секторами (sectors) або блоками (blocks).

Всі доріжки мають однакове число секторів, у яких можна максимально записати таке ж саме число байт.

Сектор має фіксований для конкретної системи розмір, що виражається степенем двійки. Найчастіше – 512 байт. Оскільки доріжки різного радіусу мають однакове число секторів, щільність запису на доріжках різна – більша у центрі.

Сектор – найменша одиниця обміну даними з оперативною пам'яттю, яку можна адресувати.

Для того, щоб контролер міг знайти на диску потрібний сектор, необхідно задати йому всі складові адреси сектора: номер циліндра, номер доріжки й номер сектора.

Оскільки прикладній програмі потрібний не сектор, а деяка кількість байт, не обов'язково кратна розміру сектора, типовий запит включає читання декількох секторів, які містять необхідну інформацію з одного або двох секторів з надлишковими даними.

ОС при роботі з диском використовує власну одиницю дискового простору, що називають кластером. При створенні файлу на диску місце йому виділяють кластерами. Наприклад, розмір кластера може дорівнює 1024 байт.

Доріжки й сектори створюються в результаті виконання процедури фізичного (або низькорівневого) форматування. Форматування передує використанню диска.

Для визначення границь блоків на диск записується ідентифікаційна інформація.

Низькорівневий формат диску не залежить від типу ОС, що цей диск використовує.

Розмітку диску під конкретний тип файлової системи виконують процедури високорівневого або логічного форматування. При цьому визначається розмір кластера й записується інформація, необхідна для роботи файлової системи, у тому числі інформація про доступний і невикористовуваний простір, про границі областей, відведених під файли й каталоги, про ушкоджені області. Крім того, на диск записується завантажник ОС – програма (звичайно невелика), що починає процес ініціалізації ОС після включення живлення.

Перш, ніж форматувати диск під певну файлову систему, він може бути розбитий на розділи. Розділ – неперервна частина фізичного диску. ОС представляє його як логічний пристрій (логічний диск).

Оскільки файлова система, з якою працює одна ОС, не може, у загальному випадку, інтерпретуватися ОС іншого типу, логічні диски не можуть використовуватись ОС різного типу.

На кожному логічному диску може створюватися тільки одна файлова система.

На різних логічних дисках того самого фізичного диску можуть розташовуватися файлові системи різного типу.

Всі розділи одного диску мають однаковий розмір низькорівневого форматування. Але при високорівневому форматуванні в різних логічних дисках можуть бути встановлені файлові системи, розміри кластерів яких мають різні розміри.

ОС може підтримувати різні статуси логічних дисків, особливим чином їх розрізняючи. Одні розділи (диски) можна використати для завантаження модулів ОС, в інші можна встановлювати тільки програми й зберігати файли даних.

Один з розділів диска позначається як завантажувальний (або активний). Саме з нього зчитується завантажник ОС.

6.9 Фізична організація й адресація файлу

Фізична організація файлу – це спосіб розміщення файлу на диску. Основними критеріями ефективності фізичної організації файлів є:

- швидкість доступу до даних;
- обсяг адресної інформації файлу;
- ступінь фрагментованості дискового простору;
- максимально можливий розмір файлу.

Неперервне розміщення – найпростіший варіант фізичної організації. При цьому файлу надається послідовність кластерів диску, що утворюють неперервну ділянку дискової пам'яті. Основне достоїнство – висока швидкість доступу, тому що витрати на пошук і зчитування кластерів файлу мінімальні. Мінімальний й обсяг адресної інформації – досить зберігати тільки номер першого кластера й обсяг файлу. Максимально можливий розмір файлу не обмежується.

Цей варіант має істотний недолік:

- неможливо передбачити, якого розміру повинна бути неперервна область, якщо файл при кожній модифікації може збільшувати свій розмір;
- велика проблема фрагментації (виникає багато вільних областей малого розміру);

6.10 Розміщення файлу у вигляді зв'язаного списку кластерів

При цьому способі на початку кожного кластера міститься покажчик на наступний кластер.

Адресна інформація мінімальна: розташування файлу задане одним числом - номером першого кластера. Фрагментація на рівні кластерів відсутня.

Недолік – складність реалізації доступу до довільно заданого місця файлу; потрібно простежити весь ланцюжок кластерів від початку.

6.11 Використання зв'язаного списку адрес

Файлу виділяється пам'ять у вигляді зв'язаного списку кластерів (рис. 10.5). Номер першого кластера запам'ятовується у записі каталогу, де зберігаються характеристики цього файлу. Інша адресна інформація відділена від кластерів файлу. З кожним кластером диска зв'язується деякий елемент – індекс. Таблиця індексів розташовується в окремій області диску і займає один кластер. Коли пам'ять вільна, всі індекси мають нульове значення.

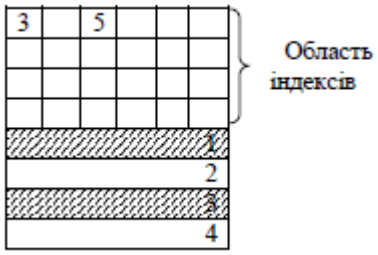


Рис. 6.5. Зв’язаний список адрес

Якщо деякий кластер N призначений деякому файлу, то індекс цього кластера стає рівним або номеру M наступного кластера даного файлу, або приймає спеціальне значення, що є ознакою того, що цей кластер є для файлу останнім. Індекс же попереднього кластера файлу приймає значення N, вказуючи на новопризначений кластер. Цей спосіб використовується у ФС FAT.

Ще один спосіб полягає у простому перерахуванні номерів кластерів, займаних файлом. Цей перелік і служить адресою файлу.

Недолік очевидний – довжина адреси залежить від розміру файлу й для великого файлу може скласти значну величину.

Перевага – висока швидкість доступу до довільного кластера, тому що тут використовується пряма адресація, а не перегляд ланцюжків.

Фрагментація також відсутня.

Як приклад розглянемо фізичну організацію ФС FAT.

6.12 Логічна організація FAT

Логічний розділ, відформатований під ФС FAT, складається з наступних областей (рис. 6.6):

- завантажувальний сектор. Містить програму початкового завантаження ОС. Вид цієї програми залежить від типу ОС, що буде завантажуватися із цього розділу;
- основна копія FAT. Містить інформацію про розміщення файлів і каталогів на диску;
- резервна копія FAT;
- область розміром у 32 рядки (16 Кбайт), що дозволяє зберігати 512 записів про файли й каталоги, тому що кожен запис каталогу складається з 32 байт;
- область даних. Призначена для розміщення всіх файлів і всіх каталогів, крім кореневого каталогу.

FAT підтримує всього два типи файлів: звичайний файл і каталог. ФС розподіляє пам'ять тільки з області даних. Таблиця FAT (основна й копія) складається з масиву індексних покажчиків, кількість яких дорівнює числу кластерів області даних. Між кластерами й індексними покажчиками є взаємно однозначна відповідність.

Індексний покажчик може приймати наступні значення, що характеризують стани пов'язаного з ним кластера:

- кластер вільний;
- кластер використовується файлом і не є останнім кластером файлу.

У цьому випадку, індексний покажчик містить номер наступного кластера файлу;

- останній кластер файлу;
- дефектний кластер;
- резервний кластер.

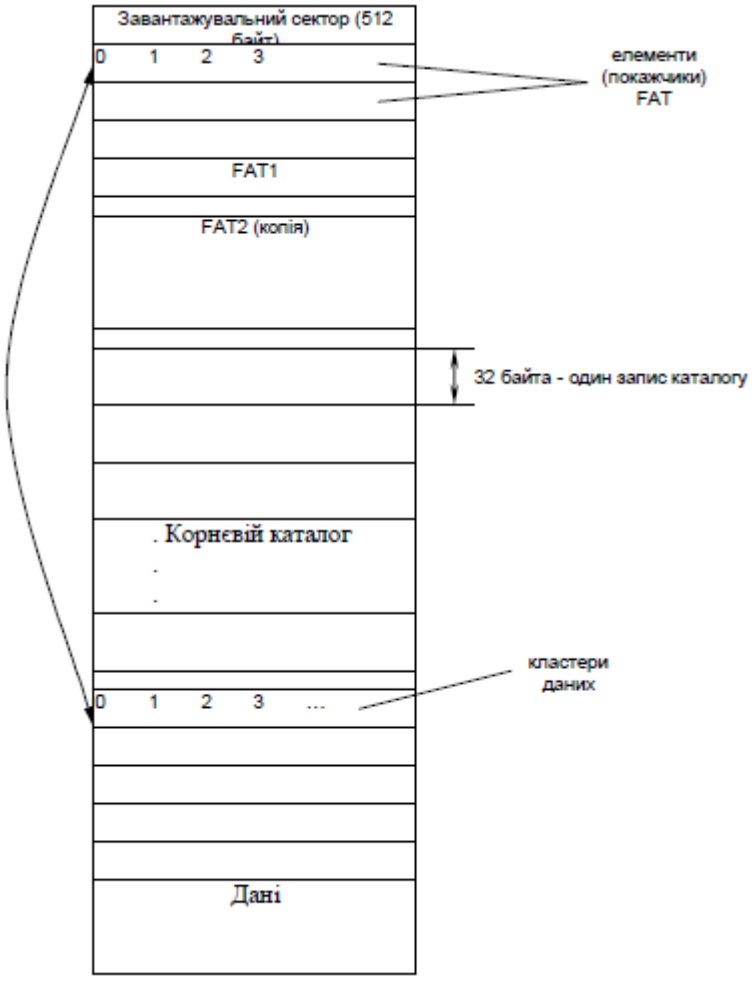


Рис. 6.6. Фізична організація FAT

Таблиця FAT є загальною для всіх файлів розділу.

У вихідному стані, після форматування, всі кластери вільні й всі індексні покажчики (крім тих, які відповідають резервним і дефектним блокам) приймають значення «кластер вільний».

При розміщенні файлу ОС переглядає FAT від початку, шукаючи перший вільний індексний покажчик. Після його виявлення в поле запису каталогу «номер першого кластера» фіксується номер цього покажчика. У кластер із цим номером записуються дані файлу. Він стає першим кластером файлу. Якщо файл уміщується в один кластер, то в покажчик відповідно

записується значення «останній кластер файлу». Якщо розмір файлу більший, то ОС продовжує перегляд FAT і шукає наступний покажчик на вільний кластер. Після виявлення в попередній покажчик заноситься номер цього кластера, і він стає наступним кластером файлу.

Процес повторюється, поки не розміститься весь файл і не буде створений зв'язний список всіх кластерів файлу.

Розмір таблиці FAT і розрядність використовуваних у ній індексних покажчиків визначається кількістю кластерів в області даних. Для зменшення втрат через фрагментацію бажано робити кластери невеликими, а для скорочення обсягу адресної інформації й підвищення швидкості обміну – навпаки.

При формуванні диску під FAT звичайно приймають компромісне рішення, і розміри кластерів вибираються з діапазону від 1 до 128 секторів або від 512 байт до 64 Кбайт.

Існують кілька різновидів FAT індексних покажчиків, що відрізняються розрядністю – умовн позначають FAT12, FAT16 й FAT32. 12 розрядні покажчики підтримують 4096 кластерів, 16 розрядні – 65535, 32 розрядні – більш 4 млн.

При видаленні файлу із ФС у перший байт відповідного запису каталогу заноситься ознака того, що запис вільний, а в усі індексні покажчики файлу – нульові значення.

Інші дані запису каталогу, у тому числі номер першого кластера файлу, залишаються незмінними, що надає шанси для відновлення помилково вилученого файлу.

Використовуваний в FAT метод зберігання адресної інформації про файли не відрізняється великою надійністю – при розриві списку індексних покажчиків в одному місці (за різними причинами), втрачається інформація про всі наступні кластери файлу.

FAT12 й FAT16 застосовувалися в ОС MS-DOS, Windows 3.1, Windows NT/2000 й Windows 95/98. У зв'язку з ростом обсягів дисків вони витісняються FAT32.

Для поглиблення вивчення питань фізичної організації файлових систем, ознайомимось детальніше з ФС FAT32 та NTFS.

6.13 Файлова система FAT

Абревіатура FAT (File Allocation Table) або таблиця розміщення файлів відноситься до лінійної таблиці з відомостями про файли: їх найменуванням, атрибутами та іншими даними, що визначають місцезнаходження файлів у середовищі FAT.

Елемент FAT визначає фактичну область диску, у якій зберігається початок фізичного файлу.

У файловій системі FAT логічний дисковий простір поділяється на дві області: системну та область даних. Системна область логічного диску створюється та ініціалізується при форматуванні. Область даних логічного

диску вміщує файли та каталоги, що підпорядковуються кореневому каталогу. На відміну від системної область даних, вона доступна через користувацький інтерфейс DOS. Системна область складається з наступних компонентів, які розміщені у логічному адресному просторі послідовно:

- Завантажувальний запис (або сектор);
- Зарезервовані сектори;
- Таблиця розміщення файлів;
- Кореневий каталог.

6.13.1 Таблиця розміщення файлів.

Ця таблиця являє собою карту або образ області даних, у якій прописано стан кожної частки області даних.

Область даних розбивається на кластери. Кластер – це один або декілька суміжних секторів у логічному дисковому адресному просторі. У таблиці FAT кластери, що належать одному файлу (тобто некореневому каталогу), пов'язані у ланцюги. Для визначення номера кластера у системі FAT16 використовується 16 бітове слово, звідси можна мати 65536 кластерів. Файли або каталог займають ціле число кластерів. Останній кластер може бути задіяний не повністю, що призводить до значної втрати дискового простору при великому розмірі кластера. На дискетах кластер займає один або два сектори, а на жорстких дисках – у залежності від їх місткості. Таблиця розміщення файлів наведена на рис.6.7.

Ємність розділу	Кількість секторів у кластері	Розмір кластерІв(Кбайт)
16-127	4	i
128-255	S	4
256-511	16	8
....	....	....
1024-204?	64	32

Рис.6.7. Таблиця розміщення файлів

Номер кластера завжди відноситься до області даних диску. Перший допустимий номер кластера завжди починається з 2. Номери кластерів відповідають області елементів таблиці розміщення файлів.

Логічний поділ області даних на кластери як сукупність секторів замість використання одиночних секторів має такий сенс: по перше, зменшується розмір самої таблиці FAT; зменшується можлива фрагментація файлів; прискорюється доступ до файлу, оскільки у декілька разів зменшується довжина ланцюжків фрагментів дискового проекту.

Однак великий розмір кластерів веде до неефективного використання області даних, особливо у випадку великої кількості маленьких файлів.

Наприклад, при розмірі кластера у 32 сектори, тобто розміром 16 Кбайт, середній розмір втрат становить 8 Кбайт. Якщо кількість файлів становить

декілька тисяч, то витрати можуть становити більше ніж 100 Мбайт. Тому у більшості сучасних ОС розміри кластерів зменшуються до 512-4 Кбайт.

Ідея розміщення файлової системи з використанням таблиць може бути проілюстрована рис.6.8.

Рис. 6.8. Розміщення файлової системи з використанням таблиць

Файл починається з восьмого кластеру. Всього файл займає 11 кластерів, 18 кластер позначено спеціальним кодом F7 як дефектний, а 1С позначено кодом FF, що визначає кінець файлу. Вільні кластери позначено кодом 00.

Оскільки файли на диску змінюються, дані одного файлу можуть розміщуватись не у сусідніх кластерах, а утворювати досить складні ланцюги. Це призводить до суттєвого зменшення швидкодії у роботі з файлами.

Розмір поля(байт)	Зміст поля даних
12	Ім'я файлу
1	Атрибути файлv
1	Резервне поле
3	Час створення
2	Дата створенні
2	Дата останнього доступу
2	Зарезервовано
2	Час останньої молнфікації
2	Дата останньої модифікації
2	Номер початкового кластера FAT
4	Розмір файла

Рис. 6.9. Структура корневого каталогу

У зв’язку з тим, що цілісність FAT дуже важлива, вона зберігається у двох ідентичних примірниках. Обновлюються копії одночасно. Для роботи використовується тільки перший екземпляр. Якщо з якихось причин він

зруйнується, то буде використано другий екземпляр. Так у системі існує утиліта перевірки та відновлення файлової структури ScanDisc у ОС Windows, яка у разі виникнення невідповідності між першою та резервною копіями FAT пропонує відновити головну таблицю, використовуючи копію.

Кореневий каталог відрізняється від звичайного тим, що він має фіксоване число елементів. Для кожного файлу та каталогу зберігається інформація у відповідності з наступною структурою (рис. 6.9):

Структура системи файлів – ієрархічна. У ній елементом каталогу може бути файл, який сам може бути каталогом (рис.6.10).

6.14 Файлові системи VFAT та FAT32

Раніше визначали, що дуже важливою характеристикою FAT було використання імен файлів формату “8.3”, де 8 символів відводять для вказівки імені файлу, а 3 – для його розширення. Пізніше до стандартної FAT16 додали ще 2 різновиди: VFAT(віртуальний FAT) та FAT32.

VFAT була розроблена для використання файлового введення-виведення у захищеному режимі. Також VFAT підтримує довгі найменування файлів. При цьому, вона зберігає сумісність із початковим варіантом FAT16, тобто підтримує формат “8.3”

Основні недоліки FAT та VFAT полягають у тому, що мають місце великі втрати на кластерізацію при великих розмірах логічного диску та обмеження розміру самого логічного диску. Це стало причиною розробки нової файлової системи з використанням старої ідеї використання таблиць FAT32.

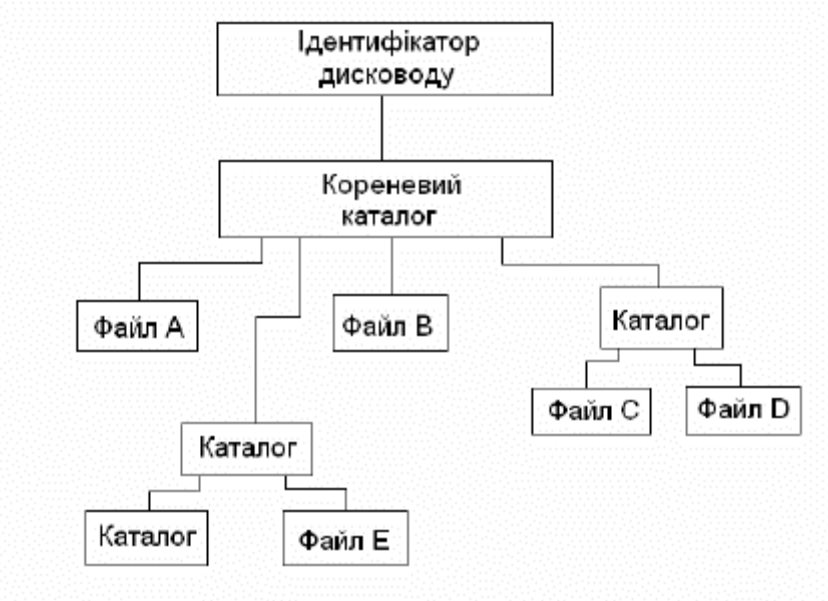


Рис. 6.10. Структура системи каталогів

FAT32 – 32 розрядна файлова система, яка доповнена деякими удосконаленнями у порівнянні з FAT16 та VFAT. Принциповим є те, що FAT32 більш ефективно використовує дисковий простір.

Вона використовує кластери локального розміру (оскільки у старій версії були обмеження на 65535 кластерів, і при збільшенні дискового простору треба було збільшувати розміри кластера). Наприклад, для дисків у 8 Гбайт FAT32 може використовувати 4 Кбайтові кластери. Як результат, економія у порівнянні з FAT16 досягла 10-15%.

FAT32 може переміщувати кореневий каталог та використовувати резервну копію замість стандартної. Кореневий каталог FAT32 задається у вигляді звичайного ланцюжка кластерів. Тобто кореневий каталог може знаходитись у довільному місці диску, що знімає старе обмеження на розмір кореневого каталогу у 512 елементів.

Окрім збільшення ємності FAT до 4 Тбайт (тера байт – 1024 Гбайт), FAT32 має необхідні вдосконалення у структурі кореневого каталогу. FAT16 вимагала щоб вся інформація кореневого каталогу була розміщена у одному дисковому кластері. При цьому вся його інформація мала не перевищувати 512 файлів. Необхідність представляти довгі імена та забезпечення сумісності з попередніми версіями FAT привели розробників до того, що для представлення довгого імені вони стали використовувати елементи каталогу. Структура елементів каталогів для FAT16 та FAT32 представлені на рис. 6.11, 6.12 та 6.113.

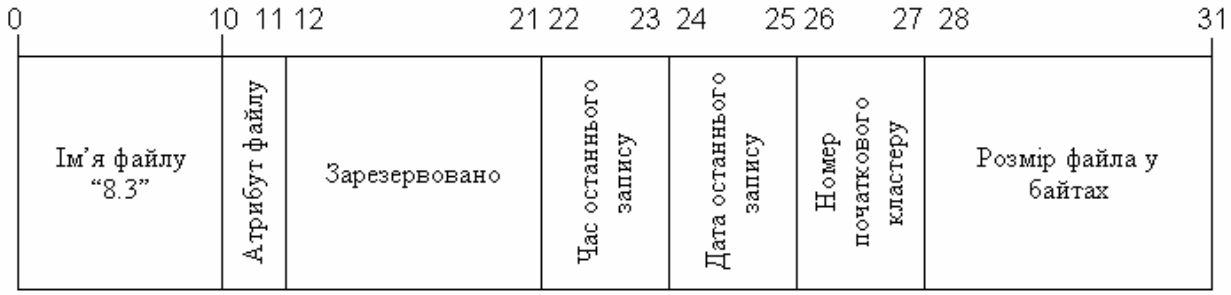


Рис 6.11. Елемент каталогу для короткого імені FAT16

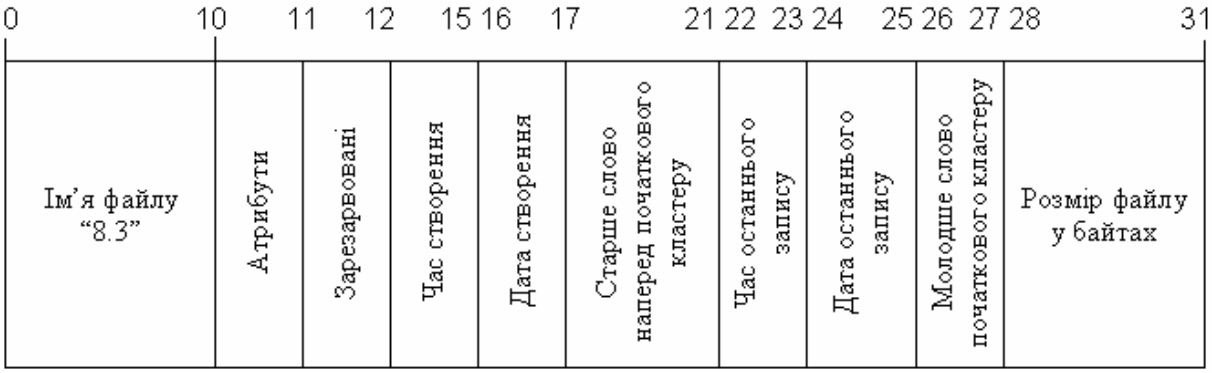


Рис. 6.12. Елемент каталогу для короткого імені FAT32

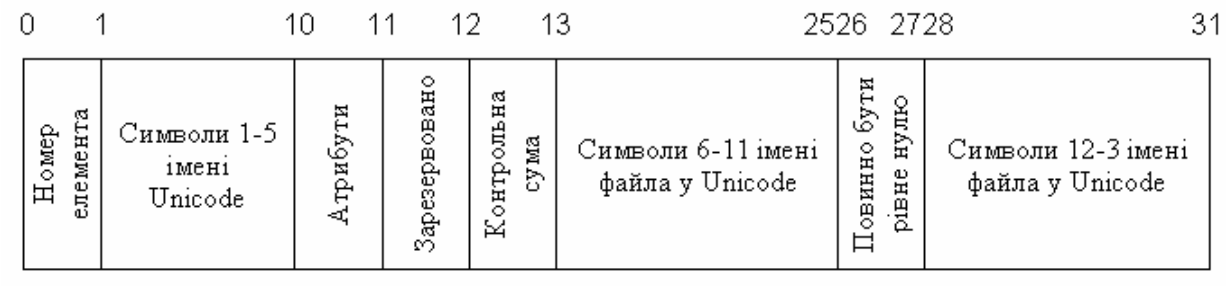


Рис. 6.13. Елемент каталогу для довгого імені FAT32

FAT32 успішно справляється з проблемою довгих імен у кореневому каталозі, але проблеми з обмеженням довжини повної файлової специфікації залишається. Тому рекомендовано обмежити довгі імена 75-80 символами, щоб залишити достатньо місця для задання шляху (180-185 символів).

6.15 Файлова система NTFS

Файли зберігаються у каталогах (здебільшого їх називають папками). На відміну від FAT, робота NTFS з дисками великого розміру виконується більш ефективно. У її розпорядженні також існують засоби для обмеження доступу до файлів та каталогів, введені механізми для підвищення надійності файлової системи, знято багато обмежень на максимальну кількість дискових секторів або кластерів.

NTFS добре працює при роботі з томами до 300-400Мбайт. Максимальний розмір тому та файлу становить 16 Ебайт (Екзабайт≈2 64 ≈16000 млрд. Гігабайт). Кількість файлів у кореневому та кореневих каталогах не обмежена.

Основою структури каталогів NTFS є структура даних, яка дістала назву “бінарне дерево” (B-Tree). У файловій системі FAT каталог має лінійну структуру, яка спеціально не упорядкована, тому при пошуку файлу послідовно проглядаємо його із самого початку.

У випадку B-Tree (двійкове дерево, див. Н. Вірт “Алгоритми та структури даних”) структура каталогу являє собою збалансоване дерево з записами, упорядкованими за алфавітом. Кожен запис, що входить до цього дерева, має атрибути файлу, показчик на відповідний файловий вузол, інформацію про час та дату створення, час і дату останнього оновлення та звертання, дані про розширені атрибути, лічильник звертань до файлу, довжину імені файлу, саме ім’я та іншу інформацію.

При пошуку файлу у каталозі файлова система проглядає тільки необхідні гілки двійкового дерева. Такий метод більш ефективний ніж послідовне читання усіх записів у каталозі, що мало місце у FAT.

Для того щоб знайти необхідний файл у каталозі (точніше вказати на його інформаційну структуру F-Node), більшість записів взагалі читати не

обов’язково. У результаті пошуку інформації про файл необхідно виконати значно меншу кількість операцій читання з диску.

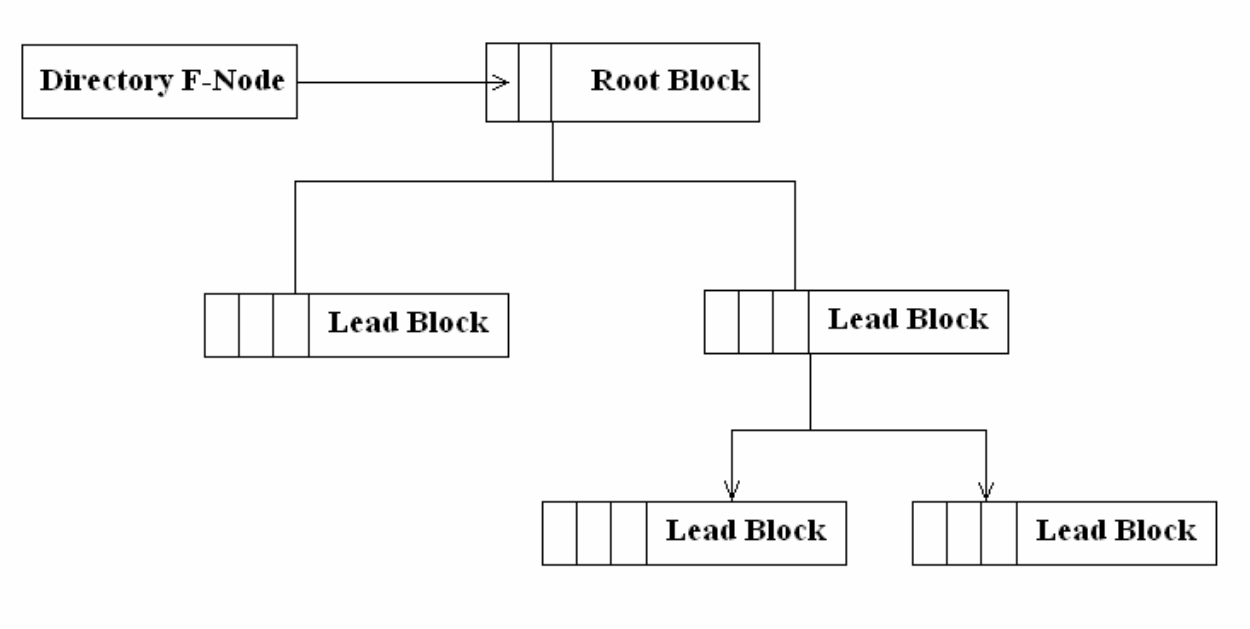


Рис. 6.14. Збалансоване двійкове дерево

Дійсно, якщо каталог містить 4096 файлів, при пошуку FAT потребує читання у середньому 64 секторів. NTFS буде читати лише 2-4 сектори для знаходження файлу.

Дійсно, при використанні 40 входів на блок (сектор диску) каталоги дерева з двома рівнями можуть мати 1640 входів, а каталоги дерева з трьома рівнями – 65640 входів. Тобто, деякий файл може бути знайдено у типовому каталозі із 65640 файлів максимум за 3 звертання. У порівнянні з FAT потрібно буде, у найгіршому випадку, більш ніж 4000 секторів.

Одним з основних понять, що використовують при роботі з NTFS є поняття тому. NTFS поділяє дисковий простір тому на кластери – блоки даних, що адресуються як одиниці даних. Вона підтримує розміри кластерів від 512 байт до 64 Кбайт. Стандартним вважається кластер розміром 2 або 4 Кбайт.

Весь дисковий простір поділено на 2 нерівні частки (рис. 10.14).

MFT Зона MFT Зона для розміщення файлів та каталогів Копія перших 16 записів MFT Зона для розміщення файлів та каталогів

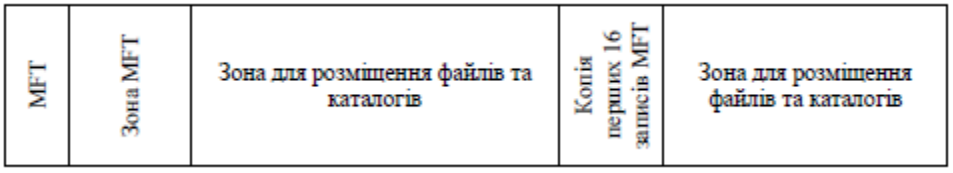


Рис. 6.16. Поділ дискового простору

Перші 12% диску відводяться під так звану MFT зону – простір, який може займати, збільшуючись у розмірі, головний службовий метафайл. MFT (Maste File Table) – спеціальний файл, головна системна структура даних, яка дозволяє визначати місцезнаходження всіх інших файлів.

MFT являє собою централізований каталог усіх інших файлів диску, у тому числі, і самого себе. MFT поділено на записи фіксованого розміру 1Кбайт, кожний запис відповідає конкретному файлу. Перші 16 файлів носять службовий характер і недоступні ОС, вони звуться метафайлами, причому самий першим з них є сам MFT.

Ці перші 16 елементів в MFT – єдина частина диску, що має строго фіксоване положення. Копія цих самих 16 записів зберігається у середині тому для надійності, оскільки вони дуже важливі. Інші частки MFT файлу можуть бути розміщені, як і інші файли, у довільних місцях диску – поновити його положення можна за допомогою його самого, якщо "закріпитись" за основу – за перший елемент MFT.

Згадані 16 файлів NTFS (метафайли) носять службовий характер. Кожен з них відповідає за деякий аспект роботи системи. Метафайли знаходяться у кореневому каталозі NTFS тому. Усі вони починаються з символу імені "$". Нижче наведено основні відомі метафайли та їх призначення:

$MFT – Can Master File Table.

$MFTmirr – Копія 16 записів MFT, розміщених у середині тому.

$LogFile – Файл підтримки операцій журналювання.

$Volume – Службова інформація (мітка тому, версія файлової системи)

$AttrDel – Список стандартних атрибутів файлів у томі.

$ – Кореневий каталог.

$Bitmap – Карта вільного місця тому.

$Boot – Завантажувальний сектор.

$Quota – Файл, де записані права користувачів на використання дискового простору.

$Upcase Файл – Таблиця співвідношення заголовних та прописних букв у найменуваннях файлів. У NTFS найменування файлів записуються у Unicode (65 тис. різних символів), тому пошук великих та малих еквівалентів – складна задача.

Усі файли тому згадуються у MFT. У цій структурі зберігається вся інформація про файли, за винятком власне даних. Ім'я файлу, розмір, положення на диску окремих фрагментів – все це зберігається у відповідному записі. Якщо одного запису MFT замало, то використовується декілька записів, причому не обов'язково послідовних.

Якщо файли мають дуже маленький розмір, то дані файлу зберігаються безпосередньо в MFT у межах одного запису.

Файл у томі з NTFS ідентифікується з так званою файловим посиланням, (File Reference), яке визначається як 64-розрядне число. Воно складається з номеру файлу, що відповідає позиції його файлового запису у MTF, та номеру послідовності. Цей номер збільшується кожний раз, коли ця позиція у MTF

використовуються повторно, що дозволяє файловій системі NTFS виконувати внутрішні перевірки цілостності.

Кожен файл задається за допомогою потоків (streams), тобто у нього немає “просто даних”, а є “потоки”.

Щоб визначитись з потоками, достатньо вказати, що один з потоків несе звичайне поняття – дані файлу. Але більшість атрибутів файлу – це теж потоки.

Таким чином визначається, що базова сутність у файлу тільки одна – номер у MTF, а все інше, включаючи потоки – опції.

Даний підхід може бути ефективно використано, наприклад, файлу можна додавати потоки та записувати у них будь-які дані.

Структурні атрибути файлів такі:

- стандартна інформація про файл. Традиційні атрибути Read Only, Hidden, Archive, System, час створення, останньої модифікації, число каталогів з посиланням на файл;

- список атрибутів. Список атрибутів, з яких складається файл, файлова вказівка на файловий запис в MTF, де розміщено кожний з атрибутів. Останній використовується, якщо файлу необхідно більш ніж один запис у MTF;

- ім’я файлу. У символах Unicode файл може мати декілька атрибутів – імен файлу;

- дескриптор захисту. Хто має доступ до файлу, його власник та захист від несанкціонованого доступу;

- дані. Власні дані файлу, його склад. Файл може мати додаткові атрибути; - корінь індексу, розміщення. Атрибути, що використовуються для індексу, бітова картка індексів імен файлів у великих каталогах (тільки для каталогів);

- розширені атрибути NTFS.

Ім’я файлу на відміну від FAT може використовувати будь-які символи, вказуючи повний перелік національних атрибутів, оскільки символи представлені 16 бітами, тобто є 65535 різних символів. Максимальний розмір імені файлу 255 символів.

Каталог – це спеціальний файл, у якому є вказівки на інші файли та каталоги, створюючи ієрархічну будову даних на диску. Файл каталогу поділено на блоки, кожний з яких має у своєму складі ім’я файлу, базові атрибути та вказівку на елемент MTF, який представляє повну інформацію про елемент каталогу.

Головний каталог диску – кореневий – нічим не відрізняється від звичайних каталогів, окрім спеціальної вказівки на нього з початку метафайла MTF. Внутрішня структура каталогу являє собою бінарне дерево. Це дерево розміщує імена файлів таким чином, щоб пошук файлу виконувався за допомогою отримання двозначних відповідей на запити про положення файлу. Бінарне дерево може дати відповідь на запитання: у якій групі, відносно даного елементу, знаходиться дане ім’я – вище чи нижче. Починаємо з такого запиту до середнього елементу. Кожна відповідь звужує зону пошуку

у 2 рази. Якщо файли відсортовані за алфавітом, то відповідь отримаємо порівнянням початкових букв.

Область пошуку, зменшена попереднього у 2 рази, знову починається з середини і т.д.

6.16 Основні відмінності FAT та NTFS

Накладні витрати на зберігання службової інформації FAT різняться від NTFS більшою компактністю та меншою складністю. У більшості томів FAT для зберігання таблиці розміщення, у якій знаходиться інформація про всі файли тому, використовується не більше 1 Мбайт. Це дозволяє формувати FAT не тільки на жорсткі диски, але й на флоппі-диски.

У NTFS службові дані займають більше місця. Кожний елемент каталогу займає 2 Кбайти. Однак це має і свої переваги. Так вже згадувалось, що вміст файлів об'ємом до 1500 байт може зберігатися безпосередньо у каталозі.

Системою NTFS не рекомендовано користуватися для форматування розділів об'ємом менше 50 Мбайт. Відповідно великі накладні витрати призводять до того, що для малих розділів службові дані можуть займати до 25% загального об'єму даних.

Корпорація Microsoft рекомендує використовувати FAT для розділів об'ємом до 256 Мбайт, а NTFS – для розділів від 400 Мбайт та більше.

Наступний критерій для порівняння – розмір файлів. Розділи FAT мають об'єм до 2 Гбайт, FAT32 – до 4 Тбайт, однак завдяки своїй внутрішній побудові, розділи FAT краще всього працюють при розмірах 200 Мбайт та менше.

Розділи NTFS можуть досягати 16 Тбайт, хоча існують апаратні проблеми, завдяки яким розміри файлів зменшуються до 2 Тбайт.

Щодо безпеки. Розділи FAT не забезпечують максимальної безпеки. З іншої сторони, розділи NTFS забезпечують максимальну безпеку як файлів, так і каталогів. Для розділів FAT можна встановити загальні правила доступу до каталогів мережі. Однак такий захист не завадить користувачу з локальним входом отримати доступ до файлів. NTFS більш захищена. Розділи NTFS можуть заборонити чи обмежити доступ як до віддалених, так і до локальних користувачів. Тобто до захищених файлів зможуть звернутися лише користувачі, які мають відповідні права.

У останній час, завдяки прогресу у конструкції дискових механізмів, значно перевищені можливості FAT у 8.4 Гбайта. Тому при роботі у середовищі Windows часто використовують FAT32 або NTFS.

У файловій системі NTFS використаний метод збалансованих бінарних дерев для зберігання та пошуку інформації про місцезнаходження файлу.

Структура каталогу являє собою збалансоване дерево із записами, що впорядковані за алфавітом. Кожний запис, що входить у склад B-Tree дерева містить атрибути файлу, вказівник та відповідний файловий вузол, інформацію про час та дату створення файлу, час та дату останнього

оновлення та звертання, розмір даних, лічильник звертань до файлу, розмір імені файлу, саме ім’я та деяку іншу інформацію.

NTFS при пошуку файлу у каталозі проглядає тільки необхідні гілки двійкового дерева. Цей метод набагато ефективніший ніж послідовне читання усіх записів у каталозі, що має місце у FAT.

Для того, щоб знайти деякий файл у каталозі (точніше вказівник на його інформаційну структуру), більшість записів вказувати не треба. Тобто використана впорядкованість для суттєвого зменшення кількості операцій читання диску.

Збалансованість дерева зменшує глибину дерева і, як наслідок, теж зменшує тривалість пошуку.

6.17 Файлові операції

Файлова система надає користувачам певний набір операцій для роботи з файлами. Але які б операції не здійснювалися, ОС необхідно виконати ряд операцій, універсальних для всіх виконуваних дій:

- за символьним іменем файлу знайти його характеристики, що зберігаються у файловій системі;
- скопіювати характеристики файлу в оперативну пам'ять;
- на підставі характеристик файлу перевірити права користувача на виконання запитаної операції (читання, запис, видалення й т.п.);
- очистити область пам'яті, відведену під тимчасове зберігання характеристик файлу.

ОС може виконувати послідовність дій над файлом двома способами:

- для кожної операції виконуються як універсальні, так й унікальні дії. Така схема іноді називається схемою без запам'ятовування стану операції;
- всі універсальні дії виконуються на початку й наприкінці послідовності операцій, а для кожної проміжної операції виконуються тільки унікальні дії.

Більшість ФС підтримують другий спосіб організації файлових операцій, як більш економічний і швидкий.

Проте, перший спосіб є більш стійким до збоїв у роботі системи, тому що кожна операція є самодостатньою й не залежить від результату попередньої.

6.18 Відкриття файлу

Системний виклик open працює із двома аргументами: символьним ім'ям файлу, що відкривається, і режимом відкриття файлу. Режим відкриття говорить системі, які операції будуть виконуватися над файлом у процесі роботи до закриття файлу за системним викликом close. Наприклад, тільки читання, тільки запис або читання й запис.

6.19 Обмін даними з файлом

Для обміну даними з файлом (попередньо відкритим) використовуються процедури read і write. У тому випадку, коли необхідно явно вказати, з якого байта файлу необхідно читати або записувати дані, використовуються додаткові системні виклики.

На основі системних викликів введення-виведення будуються могутніші бібліотечні функції введення-виведення, що складають прикладний інтерфейс ОС.

6.20 Блокування файлів

Блокування файлів й окремих записів у файлах є засобом синхронізації між працюючими в кооперації процесами, які намагаються використати один і той же файл одночасно.

Багатокористувацькі ОС звичайно підтримують спеціальний системний виклик, що дозволяє програмістові встановити й перевірити блокування на файл і його окремі області.

Процеси, що кооперуються, обов'язково повинні перевіряти наявність блокування на файл, щоб синхронізувати свою роботу.

6.21 Контроль доступу до файлів

Файли – окремий вид поділюваних ресурсів, доступ до яких ОС повинна контролювати.

У кожного об'єкта доступу існує власник. Власником може бути як окремий користувач, так і група користувачів. Власник об'єкта має право виконувати з ним будь-які припустимі для даного об'єкта операції. У багатьох ОС існує особливий користувач (superuser, root, administrator), що має усі права стосовно будь-яких об'єктів системи, не обов'язково будучи їхнім власником. Під таким ім'ям працює адміністратор системи, якому необхідний повний доступ до всіх файлів і пристроїв для керування політикою доступу.

Розрізняють два основних підходи до визначення прав доступу: Вибірний доступ. Має місце, коли для кожного об'єкта сам власник може визначити припустимі операції з об'єктами. Цей підхід називається так само довільним доступом, тому що дозволяє адміністраторові й власникам об'єктів визначити права доступу довільним образом, за їхнім бажанням. Між користувачами й групами користувачів у системах з вибірним доступом немає жорстких ієрархічних взаємин, які визначені за замовчуванням, і тільки адміністратор, за замовчуванням, наділяється всіма правами.

Мандатний доступ. Система наділяє користувача певними правами стосовно кожного поділюваного ресурсу, залежно від того, до якої групи користувача віднесено.

Від імені системи виступає адміністратор, а власники об'єктів не мають змоги управляти доступом до них за своїм розсудом.

Всі групи користувачів у такій системі утворять чітку ієрархію, причому кожна група користується всіма правами доступу групи більш низького рівня ієрархії.

Членам якої-небудь групи не дозволяється надавати свої права членам груп більш низьких рівнів ієрархії.

Мандатні системи доступу вважаються більш надійними, але менш гнучкими, звичайно вони приймаються в спеціалізованих обчислювальних системах з підвищеними вимогами до захисту інформації.

6.22 Механізм контролю доступу

Кожен користувач і кожна група користувачів звичайно має символьне ім'я, а також унікальний числовий ідентифікатор.

При виконанні процедури логічного входу в систему користувач повідомляє своє символьне ім'я й пароль, а ОС визначає відповідні числові ідентифікатори користувача й груп, у які він входить.

Всі ідентифікаційні дані, у тому числі імена й ідентифікатори користувачів і груп, паролі користувачів, а також відомості про входження користувача в групи зберігаються

Вхід користувача в систему породжує процес - оболонку, що підтримує діалог з користувачем і запускає для нього інші процеси.

Процес - оболонка одержує від користувача символьне ім'я й пароль і знаходить по них числові ідентифікатори користувача і його груп. Ці ідентифікатори зв'язуються з кожним процесом, запущеним оболонкою для даного користувача. Говорять, що процес виступає від імені даних користувача й даних груп користувачів.

Визначити права доступу до ресурсу – означає визначити для кожного користувача набір операцій, які йому дозволено застосовувати до даного ресурсу. У різних ОС для тих самих типів ресурсів може бути визначений свій список диференційованих операцій доступу.

Для файлових об'єктів цей список містить наступні операції:

- створення файлу;
- знищення файлу;
- відкриття файлу;
- закриття файлу;
- запис у файл;
- доповнення файлу;
- пошук у файлі;
- одержання атрибутів файлу;
- установка нових значень атрибутів;
- переписування;
- виконання файлу;
- читання каталогу;

- зміна власника; - зміна прав доступу.

У найбільш загальному випадку, права доступу можуть бути описані матрицею прав доступу, де стовпці відповідають всім файлам системи, рядки – всім користувачам, а на перетині рядків і стовпців вказуються різні операції (рис. 6.17). Практично у всіх ОП матриця прав доступу зберігається "вроздріб", тобто для кожного файлу й каталогу створюється так званий список керування доступом, у якому описуються права користувачів і групи користувачів на виконання операцій щодо цього файлу й каталогу. Список керування доступом є частиною характеристик файлу або каталогу й зберігається на диску у відповідній області.

		modern.txt	win.exe	...	...
Імена користувачів	Kira	читати	виконувати		
	Victor	читати	виконувати		
	Nataly	-	-	створити	
	. . .				

Рис. 6.17. Механізм прав доступу

Однак, не всі файлові системи підтримують список керування доступом, наприклад, його не підтримує ФС FAT, оскільки вона розроблялася для однокористувацької MS DOS.

Список керування доступом з доданим до нього ідентифікатором власника називається характеристиками обов'язку.

ТЕМА 7. ОРГАНІЗАЦІЯ ФУНКЦІОНУВАННЯ СИСТЕМ ПРОГРАМУВАННЯ.

7.1Трансляція, компіляція і інтерпретація

Програма –це послідовність інструкцій, призначених для виконання комп'ютером. В даний час програми оформляються у вигляді тексту, який записується у файли. Цей текст є результатом діяльності програміста і, незважаючи на специфіку формальної мови, залишається програмою для програміста.

Процес створення програми передбачає декілька етапів. За етапом розробки проекту програми слідує етап програмування. На цьому етапі

пишеться програма. Програмістами цей текст сприймається легше двійкового коду, оскільки різні мнемонічні скорочення і імена укладають додаткову інформацію.

Файл з вихідним текстом програми (його також називають вихідним модулем) обробляється транслятором, який здійснює переклад програми з мови програмування в зрозумілу машині послідовність кодів.

Транслятор - программа або технічний засіб, що виконує трансляцію програми. Це машинна програма, яка транслює з однієї мови на іншу і, зокрема, з однієї мови програмування на іншу. Це переробна програма, призначена для перетворення вихідної програми в об'єктний модуль.

Транслятор зазвичай виконує також діагностику помилок, формує словники ідентифікаторів, видає для друку тексти програми і т.д.

Трансляція програми – перетворення програми, представленої на однієї з мов програмування, в програму на іншій мові і, в певному сенсі, рівносильну першій.

Мова, на якому представлена вхідна програма, називається вхідною мовою, а сама програма – вхідним кодом. Вихідна мова називається цільовою мовою або об'єктним кодом.

Види трансляторів

Транслятори поділяють на типи:

адресний. Функціональне пристрій, що перетворюєвіртуальні адреси (англ. *Virtual address*) в реальні иадреси (англ. *Memory address*).

діалоговий. Забезпечує використання мови програмування в режимі розподілу часу.

багатопрохідний. Формує об'єктний модуль за кілька переглядів вихідної програми.

зворотний. Те ж, що детранслятор. Див. також:декомпілятор, дизассемблер.

однопрохідний. Формує об'єктний модуль за один послідовний перегляд вихідної програми.

оптимізуючий. виконуєоптимізацію коду в створюваному об'єктному модулі.

Синтаксично-орієнтований (синтаксично-керований). Отримує на вхід опис синтаксису і семантики мови і текст на описаній мові, який і транслюється відповідно до заданим описом.

тестовий. Набір макрокоманд мови асемблера, що дозволяють задавати різні налагоджувальні процедури в програмах, складених на мові асемблера.

Транслятори реалізуються у вигляді компіляторів або інтерпретаторів. З точки зору виконання роботи компілятор і інтерпретатор істотно розрізняються.

Компілятор (англ. compiler– укладач, збирач) –транслятор, що виконує перетворення програми, складеної на вхідній мові, в об'єктний модуль. Програма, яка переводить текст програми на мові високого рівня, в еквівалентну програму на машинній мові.

Програма, призначена для трансляції високорівневого мови в абсолютний код або, іноді, в мову асемблера. Вхідною інформацією для компілятора (вихідний код) є опис алгоритму або програма на проблемно-орієнтованій мові, а на виході компілятора - еквівалентний опис алгоритму на машинно-орієнтованій мові (об'єктний код).

Компіляція –трансляція програми, складеної мовою оригіналу, в об'єктний модуль. Здійснюється компілятором.

Компілювати – проводити трансляцію машинної програми з проблемно-орієнтованої мови на машинно-орієнтована мова.

Компілятор читає всю програму цілком, робить її переклад і створює закінчений варіант програми на машинній мові, який потім і виконується.

Інтерпретатор(англ. Interpreter - тлумач, усний перекладач) переводить і виконує програму рядок за рядком. Інтерпретатор бере черговий оператор мови з тексту програми, аналізує його структуру і потім відразу виконує (зазвичай після аналізу оператор транслюється в деякий проміжне представлення або навіть машинний код для більш ефективного подальшого виконання). Тільки після того як поточний оператор успішно виконаний, інтерпретатор перейде до наступного. При цьому якщо один і той же оператор буде виконуватися в програмі багато разів, інтерпретатор буде виконувати його так як, начебто зустрів вперше. Внаслідок цього програми, в яких потрібно здійснити великий обсяг обчислень, будуть виконуватися повільно. Крім того,

По-іншому можна сказати, що інтерпретатор моделює деяку обчислювальну віртуальну машину, для якої базовими інструкціями служать не елементарні команди процесора, а оператори мови програмування.

Відмінності між компіляцією і інтерпретацією.

1.Після того, як програма відкомпільована, ні сама вихідна програма, ні компілятор більш не потрібні. У той же час програма, що обробляється інтерпретатором, повинна заново переводитися на машинну мову при кожному черговому запуску програми.

2. Відкомпілювалися програми працюють швидше, але інтерпретуються простіше виправляти і змінювати.

3. Кожен конкретний мову орієнтований або на компіляцію, або на інтерпретацію –в залежності від того, для яких цілей він створювався. наприклад, Паскаль зазвичай використовується для вирішення досить складних задач, в яких важлива швидкість роботи програм. Тому дана мова зазвичай реалізується за допомогою компілятора.

З іншого боку, Бейсік створювався як мова для початківців програмістів, для яких порядкове виконання програми має незаперечні переваги.

Практично все мови програмування низького рівня і третього покоління, начебто асемблер, С компілюються, а більш високорівневі мови, типу Python або SQL, - інтерпретуються.

Іноді для однієї мови є і компілятор, і інтерпретатор. В цьому випадку для розробки і тестування програми можна скористатися інтерпретатором, а потім скомпілювати налагоджену програму, щоб підвищити швидкість її

виконання. Існує взаємопроникнення процесів трансляції та інтерпретації: інтерпретатори можуть бути компілюючими (в тому числі з динамічним компіляцією), а в трансляторах може вимагатися інтерпретація для конструкцій метапрограмування (Наприклад, для макросів в мові асемблера, умовної компіляції в С або для шаблонів в С ++).

4. Трансляція і інтерпретація –різні процеси: трансляція займається перекладом програм з однієї мови на іншу, а інтерпретація відповідає за виконання програм. Однак, оскільки метою трансляції як правило є підготовка програми до інтерпретації, то ці процеси зазвичай розглядаються разом.

Висновок: Недолік компілятора - трудомісткість трансляції мов програмування, орієнтованих на обробку даних складних структур, часто заздалегідь невідомої або яка динамічно змінюється під час роботи програми. Тоді в машинний код доводиться вставляти безліч додаткових перевірок, аналізувати наявність ресурсів операційної системи, динамічно їх захоплювати і звільняти, формувати і обробляти в пам'яті комп'ютера складні об'єкти, що на рівні жорстко заданих машинних інструкцій здійснити досить важко, а для завдання майже неможливо.

За допомогою інтерпретатора, навпаки, допустимо в будь-який момент зупинити програму, досліджувати вміст пам'яті, організувати діалог з користувачем, виконати як завгодно складні перетворення і при цьому постійно контролювати стан навколишнього програмно - апаратного середовища, завдяки чому досягається висока надійність роботи. Інтерпретатор при виконанні кожного оператора перевіряє безліч характеристик операційної системи і при необхідності максимально докладно інформує розробника про виникаючі проблеми. Крім того, інтерпретатор дуже зручний для використання в якості інструменту вивчення програмування, так як дозволяє зрозуміти принципи роботи будь-якого окремого оператора мови.

7.2 Етапи компіляції. Лексичний аналіз.

Процес компіляції розділяється на кілька етапів:

Препроцесор. Вихідна програма обробляється шляхом підстановки наявних макросів і заголовків файлів.

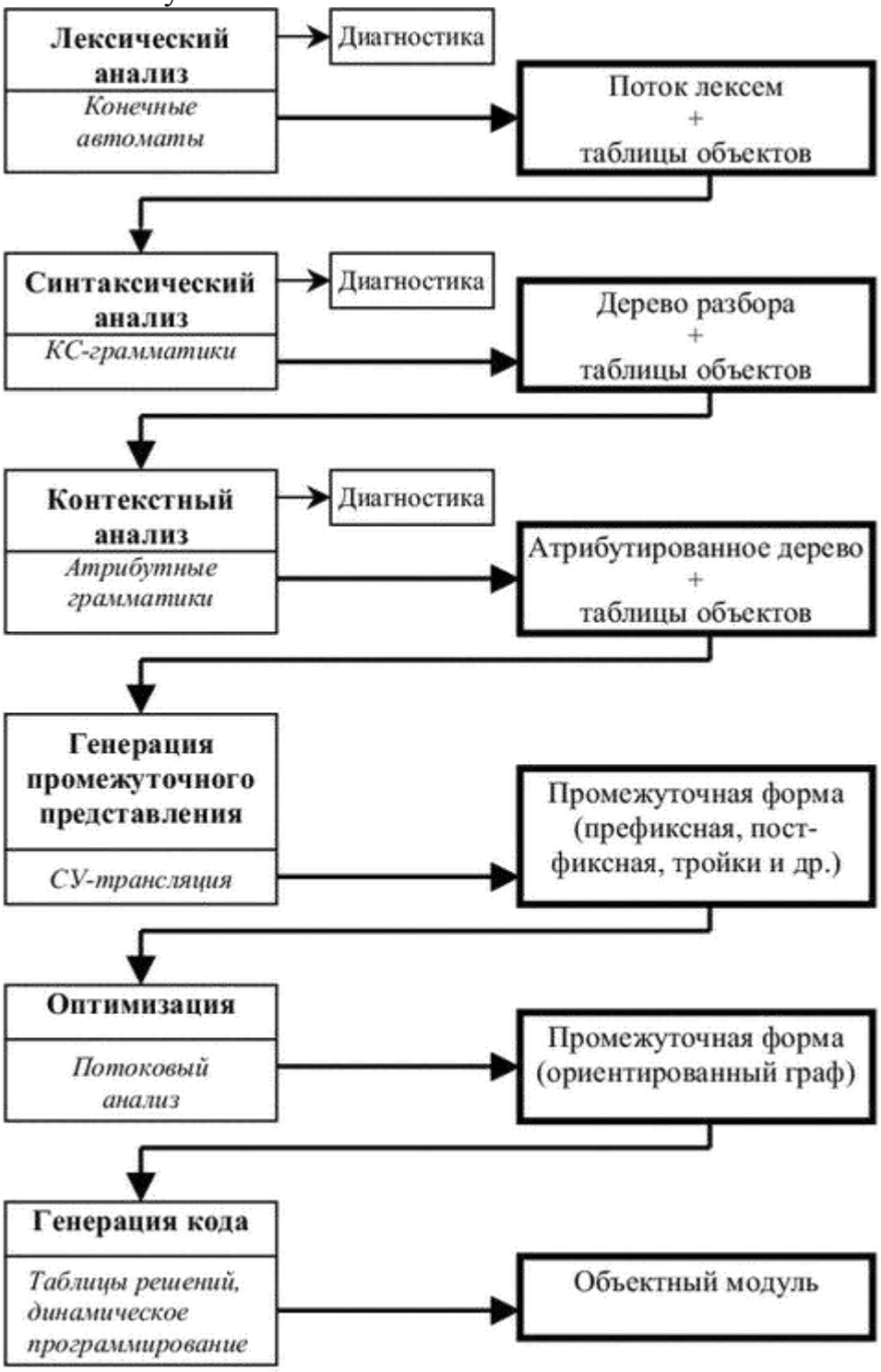
Лексичний і синтаксичний аналіз. Програма перетворюється в ланцюжок лексем, а потім у внутрішнє представлення у вигляді дерева.

Глобальна оптимізація. Внутрішнє представлення програми неодноразово перетворюється з метою скорочення розміру і часу виконання програми.

Генерація коду. Внутрішнє представлення перетворюється в блоки команд процесора, які перетворюються в ассемблеровскій текст або в об'єктний код.

Асемблювання. Якщо генерується асемблерний текст, проводиться його асемблювання з метою отримання об'єктного коду.

Збірка. Складальник з'єднує кілька об'єктних файлів в виконуваний файл або бібліотеку.



На фазі лексичного аналізу (ЛА) вхідні програма, що представляє собою потік символів, розбивається на лексеми ─ слова відповідно до визначень мови. Основним формалізмом, що лежить в основі реалізації лексичних аналізаторів, є кінцеві автомати та регулярні вирази. Лексичний аналізатор може працювати в двох основних режимах: або як підпрограма, що викликається синтаксичним аналізатором за черговий лексемою, або як повний прохід, результатом якого є файл лексем. У процесі виділення лексем ЛА може як самостійно будувати таблиці імен і констант, так і видавати

значення для кожної лексеми при черговому зверненні до нього. В цьому випадку таблиця імен будується в наступних фазах (наприклад, в процесі синтаксичного аналізу).

На етапі ЛА виявляються деякі (найпростіші) помилки (неприпустимі символи, неправильний запис чисел, ідентифікаторів і ін.).

Розглянемо більш докладно стадію лексичного аналізу.

Основне завдання лексичного аналізу –розбити вхідний текст, що складається з послідовності одиночних символів, на послідовність слів, або лексем, тобто виділити ці слова з безперервної послідовності символів. Всі символи вхідної послідовності з цієї точки зору поділяються на символи, що належать будь-яким лексем, і символи, що розділяють лексеми (роздільники). У деяких випадках між лексемами може і не бути роздільників. З іншого боку, в деяких мовах лексеми можуть містити незначні символи (наприклад, символ пробілу в Фортране). У С розділове значення символів-роздільників може блокуватися («\» в кінці рядка всередині «...»).

Зазвичай все лексеми поділяються на класи. Прикладами таких класів є числа (цілі, восьмеричні, шістнадцяткові, дійсні і т.д.), ідентифікатори, рядки. Окремо виділяються ключові слова і розділові знаки розташовуються (іноді їх називають символи-обмежувачі). Як правило, ключові слова ─ це деяка кінцева підмножина ідентифікаторів. У деяких мовах (наприклад, PL / 1) сенс лексеми може залежати від її контексту і неможливо провести лексичний аналіз у відриві від синтаксичного.

З точки зору подальших фаз аналізу лексичний аналізатор видає інформацію двох сортів: для синтаксичного аналізатора, що працює слідом за лексичним, істотна інформація про послідовність класів лексем, обмежувачів та ключових слів, а для контекстного аналізу, який працює слідом за синтаксичним, важлива інформація про конкретних значеннях окремих лексем (ідентифікаторів, чисел і т.д.).

Таким чином, загальна схема роботи лексичного аналізатора така. Спочатку виділяється окрема лексема (можливо, використовуючи символи-роздільники). Ключові слова розпізнаються або явним виділенням безпосередньо з тексту, або спочатку виділяється ідентифікатор, а потім робиться перевірка на приналежність його безлічі ключових слів.

Якщо виділена лексема є обмежувачем, то він (точніше, певний його ознака) видається як результат лексичного аналізу. Якщо виділена лексема є ключовим словом, то видається ознака відповідного ключового слова. Якщо виділена лексема є ідентифікатором – видається ознака ідентифікатора, а сам ідентифікатор зберігається окремо. Нарешті, якщо виділена лексема належить якому-небудь з інших класів лексем (наприклад, лексема є число, рядок і т.д.), то видається ознака відповідного класу, а значення лексеми зберігається окремо.

Лексичний аналізатор може бути як самостійної фазою трансляції, так і підпрограмою, що працює за принципом «дай лексему». У першому випадку (рис. 3.1, а) виходом аналізатора є файл лексем, у другому (рис. 3.1, б) лексема видається при кожному зверненні до аналізатору (при цьому, як правило,

ознака класу лексеми повертається як результат функції «лексичний аналізатор» , а значення лексеми передається через глобальну змінну). З точки зору обробки значень лексем, аналізатор може або просто видавати значення кожної лексеми, і в цьому випадку побудова таблиць об'єктів (ідентифікаторів, рядків, чисел і т.д.) переноситься на більш пізні фази, або він може самостійно будувати таблиці об'єктів. У цьому випадку в якості значення лексеми видається покажчик на вхід в відповідну таблицю.

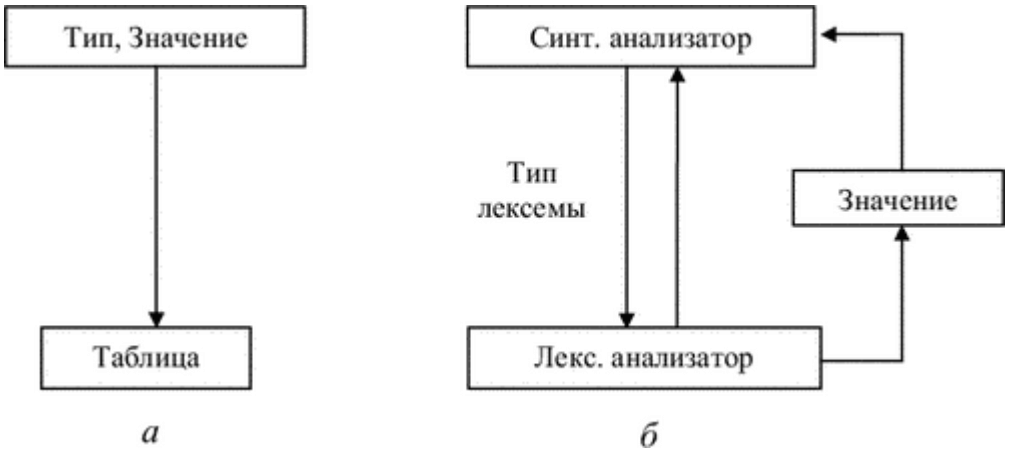


Рис. 7.1 Робота лексичного і синтаксичного аналызаторів

Робота лексичного аналізатора задається деяким кінцевим автоматом. Однак, безпосереднє опис кінцевого автомата незручно з практичної точки зору. Тому для завдання лексичного аналізатора, як правило, використовується або регулярний вираз, або праволінейная граматика. Всі три формалізму (кінцевих автоматів, регулярних виразів і праволінейних граматик) мають однакову виразну потужність. Зокрема, по регулярному виразу або праволінейной граматиці можна сконструювати кінцевий автомат, що розпізнає ту саму мову.

Основне завдання синтаксичного аналізу — розбір структури програми. Як правило, під структурою розуміється дерево, відповідне розбору в контекстно-вільної граматики мови. В даний час найчастіше використовується або LL (1) − аналіз (і його варіант − рекурсивний спуск), або LR (1) − аналіз і його варіанти (LR (0), SLR (1), LALR (1) та інші) . Рекурсивний спуск частіше використовується при ручному програмуванні синтаксичного аналізатора, LR (1) - при використанні систем автоматизації побудови синтаксичних аналізаторів.

Результатом синтаксичного аналізу є синтаксичне дерево з посиланнями на таблицю імен. У процесі синтаксичного аналізу також виявляються помилки, пов'язані зі структурою програми.

На етапі контекстного аналізу виявляються залежності між частинами програми, які не можуть бути описані контекстно вільною синтаксисом. Це в основному зв'язку «опісаніе- використання», зокрема аналіз типів об'єктів, аналіз областей видимості, відповідність параметрів, мітки та інші. В процесі

контекстного аналізу будується таблиця символів, яку можна розглядати як таблицю імен, доповнену інформацією про описи (властивості) об'єктів.

Основним формалізмом, що використовується при контекстному аналізі, є атрибутних граматики. Результатом роботи фази контекстного аналізу є атрибутувати дерево програми. Інформація про об'єкти може бути як розосереджена в самому дереві, так і зосереджена в окремих таблицях символів. В процесі контекстного аналізу також можуть бути виявлені помилки, пов'язані з неправильним використанням об'єктів.

Потім програма може бути переведена у внутрішнє представлення. Це робиться для цілей оптимізації і / або зручності генерації коду. Ще однією метою перетворення програми у внутрішнє представлення є бажання мати переносний компілятор. Тоді тільки остання фаза (генерація коду) є машинно-залежною. В якості внутрішнього уявлення може використовуватися префиксная або Постфіксний запис, орієнтований граф, трійки, четвірки і інші.

Фаз оптимізації може бути кілька. Оптимізації зазвичай ділять на машинно-залежні і машинно-незалежні, локальні і глобальні. Частина машинно-залежною оптимізації виконується на фазі генерації коду. Глобальна оптимізація намагається взяти до уваги структуру всієї програми, локальна–тільки невеликих її фрагментів. Глобальна оптимізація грунтується на глобальному потоковому аналізі, який виконується на графі програми і представляє по суті перетворення цього графа. При цьому можуть враховуватися такі властивості програми, як межпроцедурний аналіз, межмодульного аналіз, аналіз областей життя змінних і т.д.

Нарешті, генерація коду –остання фаза трансляції. Результатом її є або асемблерний модуль, або об'єктний (або завантажувальний) модуль. В процесі генерації коду можуть виконуватися деякі локальні оптимізації, такі як розподіл регістрів, вибір довгих або коротких переходів, облік вартості команд при виборі конкретної послідовності команд. Для генерації коду розроблені різні методи, такі як таблиці рішень, зіставлення зразків, що включає динамічне програмування, різні синтаксичні методи.

Звичайно, ті чи інші фази транслятора можуть або відсутні зовсім, або об'єднуватися. У найпростішому випадку однопрохідного транслятора немає явної фази генерації проміжного представлення і оптимізації, інші фази об'єднані в одну, причому немає і явно побудованого синтаксичного дерева.

7.3 Формальні мови та граматики. Найпростіший компілятор.

Алфавіт –це будь-яка множина символів. Поняття символу не визначається. Ланцюжок символів 0,1,2 записується як «012» (або 012). Інші позначення:

x^R – ланцюжок x з символами в зворотному порядку

x^n – ланцюжок x, повторена n раз

x * – ланцюжок x, повторена 0 або більше разів

x + – ланцюжок x, повторена 1 або більше разів

xy – зчеплення (конкатенація) ланцюжків x і y

| X | – довжина (число символів) ланцюжка x

e – порожній ланцюжок

Ланцюжок з одного символу будемо позначати самим символом. Букви x, y, z, u, v, w, t будемо застосовувати для позначення ланцюжків. Безліч всіх ланцюжків з елементів множини E природно позначити через E *. Мова - це підмножина E *. Приклади мов: C, російська–{0 1 | n> = 0}.

Мову програмування можна задати:

– перерахувавши всі ланцюжки;

– написавши програму-розпізнавач, яка отримує на вхід ланцюжок символів і видає відповідь «так», якщо ланцюжок належить мові і «ні» в іншому випадку;

– за допомогою механізму породження – граматики.

Щоб задати граматику, потрібно вказати:

– безліч символів алфавіту (або термінальних символів) E. Будемо позначати їх малими символами алфавіту і цифрами;

– безліч нетермінальних символів (або метасимволов), що не перехресні з E зі спеціально виділеним початковим символом S. Будемо позначати їх великими літерами;

–безліч правил виведення, що визначають правила підстановки для ланцюжків. Кожне правило складається з двох ланцюжків (наприклад, x і y), причому x повинна містити принаймні один нетермінал; і означає, що ланцюжок x в процесі виведення можна замінити на y. Висновок ланцюжків мови починається з нетермінала S. Правило граматики будемо записувати у вигляді x: y. (Також вживається запис x :: = y або x -> y)

Більш строго, визначимо поняття виведеної ланцюжка:

– S – виведений ланцюжок;

– якщо xyz – виведений ланцюжок і в граматиці є правило y: t, то xtz – виведений ланцюжок;

– визначаєма граматикою мова складається з виведених ланцюжків, що містять тільки термінальні символи.

приклади:

а)	S: e	б)	S: e
	S: 0S1		S: (S)
			S: SS

Для скорочення запису прийнято використовувати символ «або» - «|».

Коротка форма записи попередніх прикладів:

а) S: e | 0S1б) S: e | (S) | SS

Більш складний приклад:

в) S: aSBC | abC
CB: BC
bB: bb
cC: cc
bC: bc
nnn

Ця граматика породжує мову abc.

Граматики в свою чергу утворюють т.зв. метамову.Вище була описана «академічна» форма запису метамови. Існують різні способи запису синтаксичних правил, що в основному визначається умовними позначеннями і обмеженнями на структуру правил, прийнятими в використовуваних метамовах. Метамови використовуються для завдання граматики мов програмування з часів Алгола 60. Коротко розглянемо основні віхи становлення і розвитку **метамов.** У всіх випадках будемо визначати ідентифікатор.

7.3.1 Метамова Хомського

Метамова Хомського вийшла з надр математичної логіки. Він має наступну систему позначень:

символ "→"Відділяє ліву частину правила від правої (читається як «породжує » і « це є »);

нетермінали позначаються літерою А з індексом, що вказує на його номер;

термінали - це символи використовуються в описуваному мовою;

кожне правило визначає породження однієї нової ланцюжка, причому один і той же нетермінал може зустрічатися в декількох правилах зліва.

Опис ідентифікатора на метамові Хомського буде виглядати наступним чином:

1. A1 → A	**23. A1 → W**	**45. A1 → s**
2. A1 → B	**24. A1 → X**	**46. A1 → t**
3. A1 → C	**25. A1 → Y**	**47. A1 → u**
4. A1 → D	**26. A1 → Z**	**48. A1 → v**
5. A1 → E	**27. A1 → a**	**49. A1 → w**
6. A1 → F	**28. A1 → b**	**50. A1 → x**
7. A1 → G	**29. A1 → c**	**51. A1 → y**
8. A1 → H	**30. A1 → d**	**52. A1 → z**

9. A1 → I	**31. A1 → e**	**53. A2 → 0**
10. A1 → J	**32. A1 → f**	**54. A2 → 1**
11. A1 → K	**33. A1 → g**	**55. A2 → 2**
12. A1 → L	**34. A1 → h**	**56. A2 → 3**
13. A1 → M	**35. A1 → i**	**57. A2 → 4**
14. A1 → N	**36. A1 → j**	**58. A2 → 5**
15. A1 → O	**37. A1 → k**	**59. A2 → 6**
16. A1 → P	**38. A1 → l**	**60. A2 → 7**
17. A1 → Q	**39. A1 → m**	**61. A2 → 8**
18. A1 → R	**40. A1 → n**	**62. A2 → 9**
19. A1 → S	**41. A1 → o**	**63. A3 → A1**
20. A1 → T	**42. A1 → p**	**64. A3 → A3A1**
21. A1 → U	**43. A1 → q**	**65. A3 → A3A2**
22. A1 → V	**44. A1 → r**	

7.3.2 Метамова Хомського-Щутценберже

Наведений в попередньому розділі приклад опису ідентифікатора показує громіздкість метамови Хомського, що дозволяє ефективно використовувати його тільки для опису невеликих абстрактних мов. Більш компактне опис можливо із застосуванням метамови Хомського-Щутценберже, що використовує такі позначення метасимволів:

символ "=" відділяє ліву частину правила від правої (замість символу "→ ");

нетермінали позначаються літерою A з індексом, що вказує на його номер;

термінали - це символи використовуються в описуваному мовою;

кожне правило визначає породження кількох альтернативних ланцюжків, відокремлюваних один від одного символом "+", що дозволяє, при бажанні, використовувати в лівій частині тільки різні нетермінали.

Введення можливості альтернативного перерахування дозволило скоротити опис мов. Опис ідентифікатора буде виглядати наступним чином:

A1 = A + B + C + D + E + F + G + H + I + J + K + L + M + N + O + P + Q + R + S + T + U + V + W + X + Y + Z + a + b + c + d + e + f + g + h + i + j + k + l + m + n + o + p + q + r + s + t + u + v + w + x + y + z

A2 = 0 + 1 + 2 + 4 + 5 + 6 + 7 + 8 + 9
A3 = A1 + A3A1 + A3A2

7.3.3 Бекуса-Наура форми (БНФ)

Метамови Хомського і Хомського-Щутценберже використовувалися в математичній літературі при описі простих абстрактних мов. Метамови, запропоновані Бекусом і Наурумом, вперше було використано для опису синтаксису реального мови програмування Алгол 60. Поряд з новими позначеннями метасимволов, в ньому використовувалися змістовні позначення нетерміналів. Це зробило опис мови наочніше і дозволило надалі широко використовувати дану нотацію для опису реальних мов програмування. Були використані наступні позначення:

символ «:: =" відділяє ліву частину правила від правої;

нетермінали позначаються довільній символьним рядком, укладеної в кутові дужки «<» і «>«;

термінали - це символи, використовувані в описуваному мовою;

кожне правило визначає породження кількох альтернативних ланцюжків, відокремлюваних один від одного символом вертикальної риски «|».

Приклад опису ідентифікатора з використанням БНФ:

<Буква> :: = A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

<Цифра> :: = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<Ідентифікатор> :: = <буква> | <Ідентифікатор> <буква> | <ідентифікатор> <цифра>

Правила можна задавати і окремо:

<Ідентифікатор> :: = <буква>
<Ідентифікатор> :: = <ідентифікатор> <буква>
<Ідентифікатор> :: = <ідентифікатор> <цифра>

наприклад, Граматику цілих чисел без знака можна записати у вигляді:

<Число>: <цифра> | <Цифра> <число>
<Цифра>: 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

7.3.4 Розширені Бекуса-Наура форми (РБНФ)

Метамови, представлені вище, дозволяють описувати будь-який синтаксис. Однак, для підвищення зручності та компактності опису, доцільно вести в мову додаткові конструкції. Зокрема, спеціальні метасимволи були розроблені для опису необов'язкових ланцюжків, що повторюються

ланцюжків, обов'язкових альтернативних ланцюжків. Існують різні розширені форми метамовою, незначно відрізняються один від одного. Їх різноманітність найчастіше пояснюється бажанням розробників мов програмування по-своєму описати створюваний мову. До прикладів таких широко відомих метамовою можна віднести: метамова PL / I, метамова Вірта, який використовується при описі Модули-2, метамова Керніган-Рітчі, що описує Сі. Найчастіше такі мови називаються розширеними формами Бекуса-Наура (РБНФ).

Зокрема, РБНФ, використовувані Віртом, мають такі особливості:

Квадратні дужки «[« і «]» означають, що укладена в них синтаксична конструкція може бути відсутнім;

фігурні дужки «{« і «}» означають її повторення (можливо, 0 раз);

круглі дужки «(« і «)» використовуються для обмеження альтернативних конструкцій;

поєднання фігурних дужок і косою риси «{/» і «/}» використовується для позначення повторення один і більше разів. Нетермінальні символи зображуються словами, що виражають їх інтуїтивний сенс і написаними російською мовою.

Якщо нетермінал складається з декількох смислових слів, то вони повинні бути написані разом. В цьому випадку для підвищення зручності в сприйнятті фрази доцільно кожне її слово починати з великої літери або розділяти слова у фразах символом підкреслення. Термінальні символи зображуються словами, написаними літерами латинського алфавіту (зарезервовані слова) або ланцюжками знаків, укладеними в лапки. Синтаксичним правилам передує знак «$» на початку рядка. Кожне правило закінчується знаком «.» (крапка). Ліва частина правила відділяється від правої знаком «=« (так само), а альтернативи - вертикальною лінією «|». Цей варіант РБНФ і буде використовуватися для опису синтаксису мов в лабораторній роботі. Відповідно до цих правил синтаксис ідентифікатора буде виглядати наступним чином:

$ Буква = «A» | »B» | »C» | »D» | »E» | »F» | »G» | »H» | »I» | »J» | »K» | »L» | »M» | »N» | »O» | »P» | »Q» | »R» | «S» | »T» | »U» | »V» | »W» | »X» | » Y »|» Z »|» a »|» b »|» c »|» d »|» e »|» f »|» g »|» h »|» i »|» j »|» k » | »l» | »m» | »n» | «o» | »p» | »q» | »r» | »s» | »t» | »u» | »v» | »w» | » x »|» y »|» z ».

$ Цифра = "0" | »1» | »2» | »3» | »4» | »5» | »6» | »7» | »8» | »9».

$ Ідентифікатор = буква {буква | цифра}.

7.3.5 Діаграми Вірта

Поряд з текстовими способами опису синтаксису мов широко використовуються і графічні метамови, серед яких найбільш широку популярність здобув мову діаграм Вірта, вперше застосований для опису мови Паскаль. Метасимволи замінені наступними графічними позначеннями (рис. 2.1):

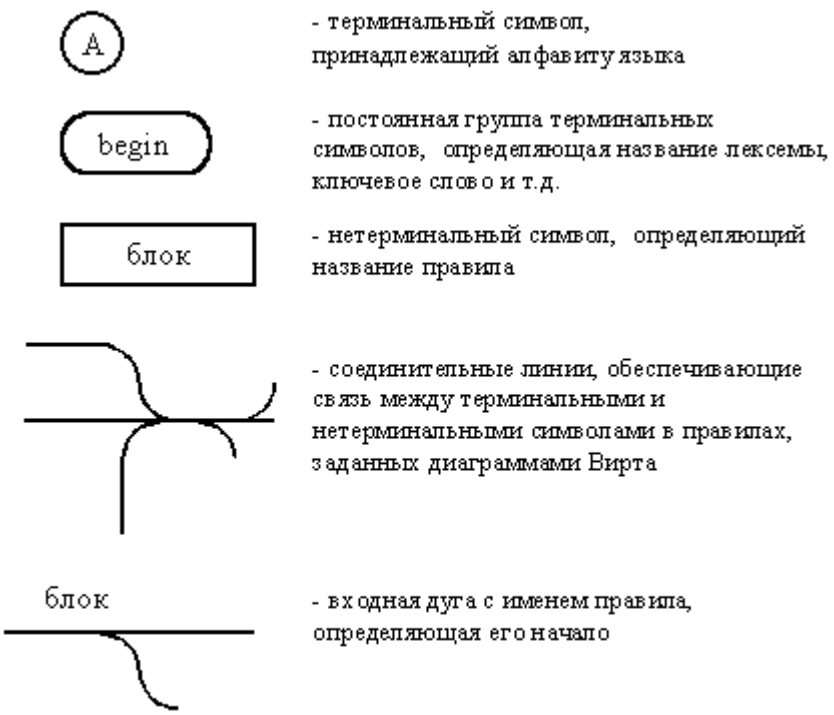


Рис 7.2 Графічні примітиви, використовувані при побудові діаграм Вірта

термінальні символи і їх постійні групи розташовуються в колах або прямокутниках з округленим вертикальними сторонами;

нетермінальні символи заносяться всередину прямокутників;

кожен графічний елемент, відповідний терміналу або нетерміналу, має по одному входу і виходу, які зазвичай малюються на протилежних сторонах;

кожного правила відповідає своя графічна діаграма, на якій термінали і нетермінали з'єднуються за допомогою дуг;

альтернативи в правилах задаються розгалуженням дуг, а ітерації - їх злиттям;

повинна бути одна вхідна дуга (розташовується зазвичай зліва і зверху), що задає початок правила і позначена ім'ям визначається нетермінала, і одна вихідна, що задає його кінець (зазвичай розташовується праворуч і знизу).

Приклад опису ідентифікатора з використанням діаграм Вірта представлений на рис 7.3.

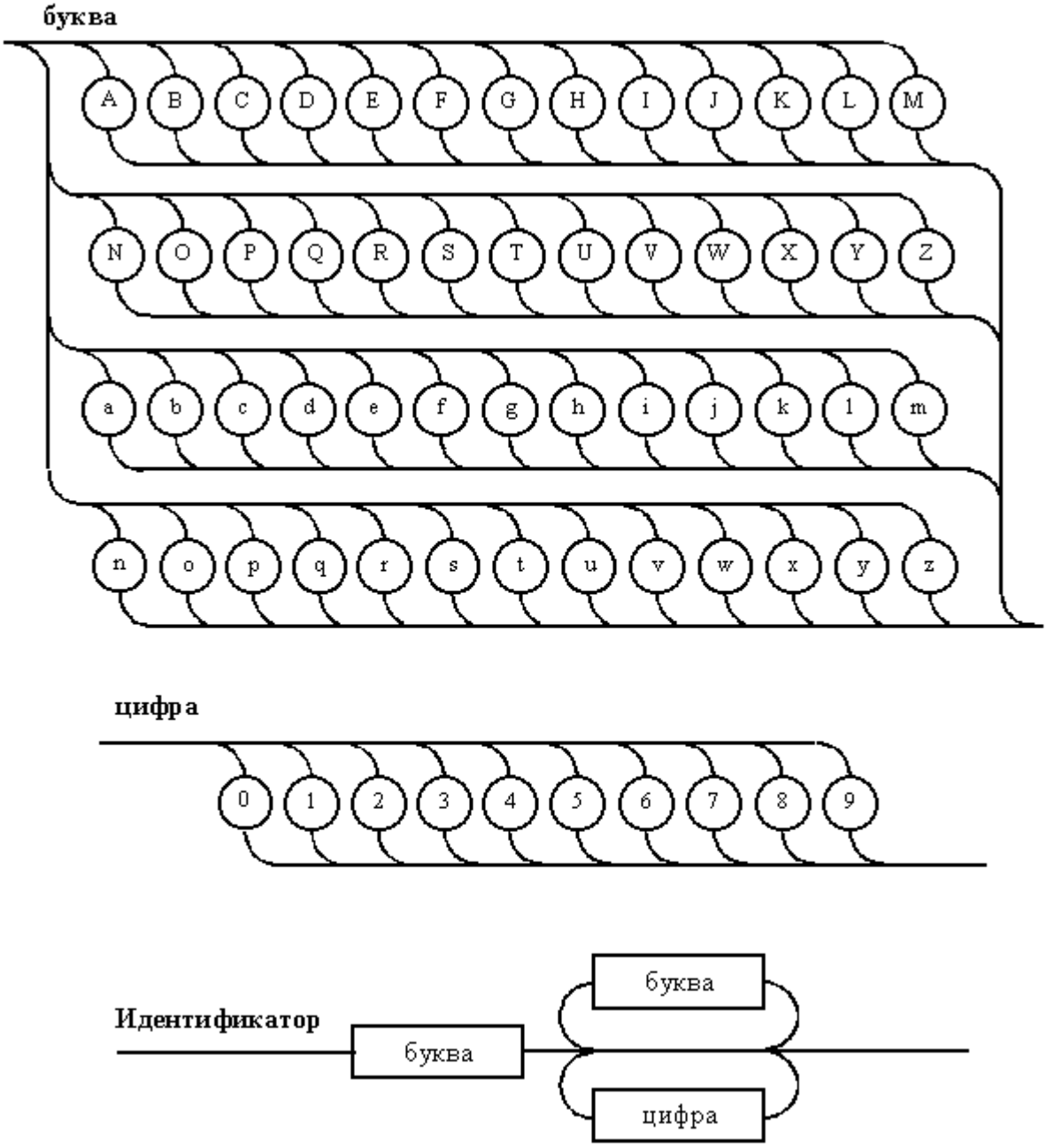


Рис. 7.3 Опис ідентифікатора з використанням діаграм Вірта

Зазвичай стрілки на дугах діаграм не ставляться, а напрямки зв'язків відслідковуються рухом від початкової дуги відповідно до плавними вигинами проміжних дуг і розгалужень. Таким же чином визначаються входи і виходи терміналів і нетерміналів. Спеціальних стандартів на діаграми Вірта немає, тому графічні позначення можуть змінюватися в залежності від засобів малювання. Можна, наприклад, використовувати псевдографіку або просто текстові символи, зв'язку зі стрілками. Однак такий вид правил менш зручний для сприйняття і тому застосовується вкрай рідко.

Діаграми Вірта дозволяють задавати альтернативи, рекурсії, ітерації і по образотворчої потужності еквівалентні РБНФ. Але графічне відображення правил більш наочно. Крім цього допускається довільне проведення дуг, що

зменшує кількість елементів в правилі за рахунок його неструктурованість. Діаграми Вірта є зручним вихідним документом для побудови лексичного і синтаксичного аналізаторів.

7.3.6 Розглянемо мову найпростіших арифметичних формул:

Наведемо граматику математичної формули з використанням метамови Бекуса-Наура:

<Формула>: (<формула>) | <Число> | <Формула> <знак> <формула>

<Знак>: + | *

Чому «3 + 5 * 2» є формулою? Наведемо послідовність перетворень ланцюжків (так званий «розбір» або «висновок»):

<формула> :	<формула>		<знак><формула>
	<формула>	<знак>	<формула><знак><формула>
	<число>	<знак>	<формула><знак><формула>
	3	<знак>	<формула><знак><формула>
	3	+	<формула><знак><формула>
	3	+	<число> <знак><формула>
	3	+	5 <знак><формула>
	3	+	5 * <формула>
	3	+	5 * <число>
	3	+	5 * 2

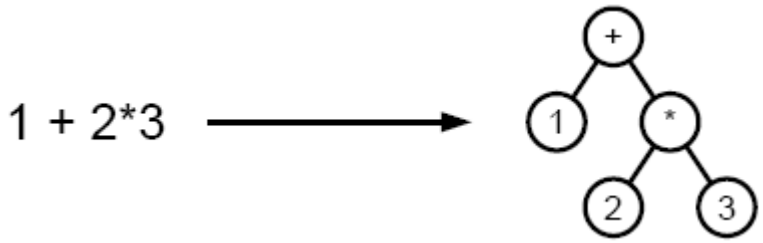
Скорочено наявність висновку (ланцюжки перетворень) будемо за приписувати у вигляді <формула> :: 3 + 5 * 2. Більшість граматик допускають кілька різних висновків для однієї і тієї ж ланцюжка з мови. Побудуйте інший висновок для ланцюжка «3 + 5 * 2»– вправу.

Якщо в процесі виведення ланцюжка правила граматики застосовуються тільки до самого лівому нетерміналу, кажуть, що отриманий лівий висновок ланцюжка. Аналогічно визначається правий висновок.

Зобразимо виконувані заміни ланцюжків у вигляді т.зв. «Дерева розбору» (або дерева виведення). За традицією дерево зображується «догори ногами»:

Намальоване дерево має гілки (лінії) і вузли (позначені терміналами і нетерміналом), з яких ростуть гілки. Кінцеві вузли (термінали) називаються листям. Поняття «поддерево», «корінь дерева», мабуть, не потребують визначення.

Одне і те ж дерево розбору може описувати різні висновки (в дереві не фіксований порядок застосування правил). Однак, між лівими (або правими) висновками і деревами розбору для ланцюжків існує однозначна відповідність.



Якщо для однієї і тієї ж ланцюжка можна зобразити два різних дерева розбору (або, що те ж саме, побудувати, два різних правих виведення), граматика називається неоднозначною. Описана граматика неоднозначна. Той же самий мову можна описати однозначною граматикою:

<Формула>: <терм> | <Терм> <знак> <формула>
<Терм>: (<формула>) | <Число>
<Знак>: + | *

Як зміниться дерево розбору? Дерево розбору визначає деяку структуру ланцюжка мови. Так, ми бачимо, що подцепочка «3 + 5» є «формулою». Це суперечить нашим (інтуїтивним) поняттям про сенс вихідної формули: 3 + 5 на відміну від 5 * 2 не є подвираженіем. Ми можемо врахувати пріоритет операцій, змінивши граматику:

<Формула>: <терм> | <Формула> + <терм>
<Терм>: <елемент> | <Терм> * <елемент>
<Елемент>: (<формула>) | <Число>

Крім звичної форми запису арифметичних формул (т.зв. «інфіксной», тобто зі знаком операції між операндами), широко поширена «Постфіксний» (або зворотна польська) форма запису, в якій операція розташована після операндів.

приклади:

2 + 32 3 +
2 * 3 + 42 3 * 4 +
2 * (3 + 4) 2 3 4 + *

У зворотної польської записи дужки не потрібні, а значення формули дуже легко вирахувати при використанні стека чисел.

Як ви вже знаєте переклад з однієї мови на іншу називається трансляцією або компіляцією. Так як будь-яку інфіксне формулу можна

переписати в постфіксний запис зі збереженням порядку проходження чисел, то для компіляції формули з інфіксной в Постфіксний запис слід виділити зовнішню операцію, записати в постфіксной формі обидва операнда і записати операцію слідом за ними. Для перекладу операнда в Постфіксний запис можна рекурсивно звернутися до програми перекладу формули. Однак дотримання правил граматики при компіляції формули за один її перегляд зліва направо призводить до наступного фрагменту:

```
CompForm () {
CompForm ()
...
```

виконання якого, звичайно ж, ніколи не завершиться. Проблема виникла через те, що ланцюжки в лівій і правій частинах правила починаються з одного нетермінала (кажуть, що граматика леворекурсівна).

Якщо усунути ліву рекурсію:

```
<Формула>: <терм> | <Терм> <плюс мінус> <формула>
<Терм>: <елемент> | <Елемент> <помножити розділити> <терм>
<Плюс мінус>: + | -
<Помножити розділити>: * | /
```

то описана проблема зникне, рекурсивний компілятор можна буде написати, проте з'являться нові труднощі (яке дерево розбору буде відповідати ланцюжку «5-3-2»?).

Фактично, перетворивши граматику, ми змінили порядок згортки операцій. Традиційно операції одного пріоритету виконуються зліва направо (кажуть, що операції левоассоціативни), а тільки що написана граматика визначає операції як правоассоціативние.

Найбільш просто вирішити цю проблему можна, додавши в метамова НФБН символи ітерації {} «повторити 0 або більше разів». Із застосуванням нових позначень граматика легко запишеться без лівої рекурсії:

```
<Формула>: <терм> {<плюс мінус> <формула>}
<Терм>: <елемент> {<помножити розділити> <елемент>}
```

Написаний по цій граматиці рекурсивний компілятор також буде виглядати просто:

```
char * infix, * postfix; / * Покажчики на вхідні і вихід-
 ву ланцюжка * /
 CompForm () {/ * компілювати формулу * /
 register char sign;
 CompTerm ();
 while ((sign = * infix) == '+' || sign == '-') {
 ++ infix;
```

```
CompTerm ();
* Postfix ++ = sign;
* Postfix ++ = '';
}
}
CompTerm () {/ * компілювати терм * /
register char sign;
CompEl ();
while ((sign = * infix) == '*' || sign == '/') {
++ infix;
CompEl ();
* Postfix ++ = sign;
* Postfix ++ = '';
}
}
CompEl () {/ * компілювати елемент * /
if (* infix == '(') {
++ infix;
CompForm ();
if (* infix ++! = ')') error ();
} Else {
if (! isdigit (* infix)) error ();
while (isdigit (* infix)) * postfix ++ = * infix ++;
* Postfix ++ = '';
}
}
```

Використана нами при написанні компілятора техніка зветься рекурсивного спуску. Вхідні ланцюжок ми переглядаємо зліва направо, дерево виведення проходимо зверху вниз (тобто від початкового нетермінала <формула>).

Функція error в компіляторі служить для виведення повідомлення про те, що пред'явлений ланцюжок не входить в мову арифметичних формул. Якщо компілятор при отриманні на вхід ланцюжка не видає повідомлення про помилку, кажуть, що цей ланцюжок допущений.

Якщо розбір ланцюжка-програми супроводжується не перекладом її в інше уявлення для подальшого виконання, а негайним виконанням (в нашому випадку – обчисленням значення), говорять, що саме така програма інтерпретується.