# Конспект лекцій з дисципліни

**“Операційні системи”**

**Київ 2017**

# **ЗМІСТ**

[**Основні функції ОС** 7](#_Toc483268836)

[**Класифікація ОС** 7](#_Toc483268837)

[ДОС(Дискові операційні системи) 7](#_Toc483268838)

[ОС загального призначення 8](#_Toc483268839)

[Системи віртуальних машин 8](#_Toc483268840)

[Системи реального часу 9](#_Toc483268841)

[Засоби крос-розробки 9](#_Toc483268842)

[Системи проміжних типів 10](#_Toc483268843)

[Сімейства операційних систем 10](#_Toc483268844)

[**Архітектура ОС 10**](#_Toc483268845)

[Утиліти 12](file:///C:\Users\LenIN\Desktop\Конспект_ОС%2520.doc#_Toc483268846)

[Библіотеки 12](file:///C:\Users\LenIN\Desktop\Конспект_ОС%2520.doc#_Toc483268847)

[Багатошарова структура ОС 14](#_Toc483268848)

[Супервізор 19](#_Toc483268849)

[Мікроядерна архітектура 22](#_Toc483268850)

[Процеси 25](#_Toc483268851)

[Реалізація процесів 28](#_Toc483268852)

[Спілкування між процесами 31](#_Toc483268853)

[**Синхронізація процесів** 32](#_Toc483268854)

[Синхронізація за допомогою елементарних прийомів нижнього рівня 35](#_Toc483268855)

[Елементарні прийоми синхронізації на верхньому рівні 48](#_Toc483268856)

[Монітор Хоара. 50](#_Toc483268857)

[Тупики 53](#_Toc483268858)

[Алгоритм запобігання тупикових ситуацій 53](#_Toc483268859)

[Проблема розподілу ресурсів та запобігання тупіків 57](#_Toc483268862)

[**Мультипрограмування.** **Розподіл часу процесора** 66](#_Toc483268863)

[Системи пакетної обробки 67](#_Toc483268864)

[Системи поділу часу 68](#_Toc483268865)

[Системи реального часу 68](#_Toc483268866)

[Мультипроцесорна обробка 69](#_Toc483268867)

[**Процеси і потоки** 71](#_Toc483268868)

[**Алгоритми планування** 75](#_Toc483268869)

[Витісняючі і не витісняючі алгоритми планування 75](#_Toc483268870)

[Алгоритми планування, засновані на квантуванні 76](#_Toc483268871)

[Алгоритми планування, засновані на пріоритетах 77](#_Toc483268872)

[Змішані алгоритми планування 78](#_Toc483268873)

[Планування в системах реального часу 78](#_Toc483268874)

[Використання динамічних пріоритетів 84](#_Toc483268875)

[**Мультипрограмування на основі переривань** 86](#_Toc483268876)

[Переривання 86](#_Toc483268877)

[Програмні переривання 89](#_Toc483268878)

[Диспетчеризація й пріоритезація переривань в ОС 89](#_Toc483268879)

[Процедури обробки переривань і поточний процес 90](#_Toc483268880)

[Системні виклики 91](#_Toc483268881)

[**Керування пам'яттю** 94](#_Toc483268882)

[Функції ОС по керуванню пам'яттю 94](#_Toc483268883)

[Типи адрес 95](#_Toc483268884)

[Алгоритми розподілу пам'яті 101](#_Toc483268885)

[Розподіл пам'яті динамічними розділами 103](#_Toc483268886)

[Переміщувані розділи 104](#_Toc483268887)

[Свопінг і віртуальна пам'ять 104](#_Toc483268888)

[Сторінковий розподіл 105](#_Toc483268889)

[Сегментний розподіл 113](#_Toc483268890)

[Сегментно-сторінковий розподіл 117](#_Toc483268892)

[Поділювані сегменти пам'яті 122](#_Toc483268893)

[Кешування даних 122](#_Toc483268894)

[**Введення - виведення і файлова система** 130](#_Toc483268895)

[Завдання ОС по керуванню файлами й пристроями 131](#_Toc483268896)

[Основні поняття та концепція організації введення - виведення 131](#_Toc483268897)

[Режим керування введенням-виведенням 133](#_Toc483268898)

[Закріплення пристроїв. Загальні пристрої введення - виведення 136](#_Toc483268899)

[Основні системні таблиці введення - виведення 136](#_Toc483268900)

[Синхронне та асинхронне введення - виведення 140](#_Toc483268901)

[Керування та буферизація при роботі з магнітними дисками 141](#_Toc483268902)

[Організація доступу до зовнішних пристроїв. 143](#_Toc483268903)

[Порти введення-виведення. 145](#_Toc483268904)

[Шини 152](#_Toc483268905)

[Організація паралельної роботи процесора та пристроїв введення - виведення 156](#_Toc483268906)

[Узгодження швидкостей обміну й кешування даних 156](#_Toc483268907)

[Розподіл пристроїв і даних між процесами 157](#_Toc483268908)

[Забезпечення зручного логічного інтерфейсу між пристроями й іншою частиною системи 157](#_Toc483268909)

[Підтримка широкого спектру драйверів і простота включення нового драйвера в систему 158](#_Toc483268910)

[Динамічне завантаження й вивантаження драйверів 159](#_Toc483268911)

[Підтримка синхронних й асинхронних операцій введення - виведення 160](#_Toc483268912)

[Багатошарова модель підсистеми введення - виведення 160](#_Toc483268913)

[Менеджер введення - виведення 161](#_Toc483268914)

[Багаторівневі драйвери. 161](#_Toc483268915)

[Спеціальні файли 164](#_Toc483268916)

[**Файлові системи** 165](#_Toc483268917)

[Логічна організація файлової системи 165](#_Toc483268918)

[Мети й завдання файлової системи 165](#_Toc483268919)

[Типи файлів 166](#_Toc483268920)

[Ієрархічна структура файлової системи 167](#_Toc483268921)

[Імена файлів 167](#_Toc483268922)

[Монтування 168](#_Toc483268923)

[Атрибути файлів 170](#_Toc483268924)

[Логічна організація файлу 171](#_Toc483268925)

[Фізична організація файлової системи 172](#_Toc483268926)

[Файлова система FAT 180](#_Toc483268927)

[Файлові системи VFAT та FAT-32 184](#_Toc483268928)

[Файлова система NTFS 186](#_Toc483268929)

[Основні відмінності FAT та NTFS 192](#_Toc483268930)

[Файлові операції 193](#_Toc483268931)

[**Архітектурні** **особливості побудови ОС** 198](#_Toc483268932)

[Віртуальна машина 199](#_Toc483268933)

[Користувач 200](#_Toc483268934)

[Інтерфейс користувача 200](#_Toc483268935)

[Привілейований користувач 201](#_Toc483268936)

[Команди та командний інтерпретатор 201](#_Toc483268937)

[Процеси 202](#_Toc483268938)

[Підсистема введення - виведення 203](#_Toc483268939)

[Файлова система 205](#_Toc483268940)

[Захист файлів 207](#_Toc483268942)

[Міжпроцесорні комунікації 209](#_Toc483268943)

**ВСТУП**

Дисципліна«Операційні системи» є спеціальною дисципліною циклу професійної та практичної підготовки бакалаврів за напрямом 6.050102 «Комп’ютерна інженерія», яка ознайомлює студентів з фундаментальними концепціями та принципами побудови операційних систем (ОС)

# **Основні функції ОС**

* Забезпечувати завантаження користувальницьких програм в оперативну пам'ять та їх виконання.
* Забезпечувати керування пам'яттю. У найпростішому випадку це вказівка єдиній завантажуваній програмі адреси, на якій кінчається пам'ять, доступна для використання, і починається пам'ять, зайнята системою.
* Забезпечувати роботу із пристроями довгострокової пам'яті.
* Надати стандартизований доступ до різних периферійних пристроїв.
* Надати користувальницький інтерфейс.

У ряді ОС цим функції вичерпуються.(MS-DOS)

Більш розвинені ОС надають додаткові можливості:

* Паралельне (або псевдопаралельне) виконання декількох завдань.
* Організація взаємодії завдань один з одним.
* Організація максимальної взаємодії й поділу ресурсів.
* Захист системних ресурсів, даних і програм користувача.
* Аутентифікація користувача (перевірка користувача на предмет того, що він є тим, за кого себе видає й має право виконати ту або іншу операцію).

# **Класифікація ОС**

По тому, які з перерахованих вище функцій виконують ОС, їх можна розділити на наступні:

## ДОС(Дискові операційні системи)

Це системи, які беруть на себе виконання тільки перших чотирьох функцій. Як правило, це резидентний набір програм, і не більш того. ДОС завантажує користувальницьку програму на згадку й передає їй керування, після чого програма робить із системою, що їй заманеться. При завершенні програми вважається “гарним тоном” залишити машину в такому стані, щоб ДОС змогла продовжити роботу. Якщо програма приводить машину в якийсь інший стан, то ДОС нічим їй у цьому перешкодити не може.

Характерний приклад - різні завантажувальні монітори. Такі системи працюють одночасно тільки з однією програмою.

MS DOS для IBM PC - прямий спадкоємець такого монітора.

Існування таких систем обумовлено їхньою простотою й тим, що для їхньої реалізації необхідно мало ресурсів.

Ще одна причина по якій вони можуть використатися навіть на досить потужних машинах - вимога програмної сумісності з ранніми моделями того ж сімейства комп'ютерів.

## ОС загального призначення

Сюди ставляться системи, що виконують усі перераховані функції.

Прикладами таких систем є IBMDOS й OC/360, а також наші аналоги ОС ЄС.

Ці ОС розраховані на інтерактивну роботу одного або декількох користувачів у режимі поділу часу, при не дуже жорстких вимогах вчасної реакції системи на зовнішні події. У таких системах велика увага приділяється захисту самої системи, ПО й користувальницьких даних від помилкових і зловмисних дій програм і користувачів.

До цього класу відноситься відома ОС Windows 2000, а також системи сімейства UNIX.

## Системи віртуальних машин

Це ОС, що допускає одночасну роботу декількох програм, але створює при цьому для кожної програми ілюзію того, що машина перебуває в повному її розпорядженні, як при роботі під керуванням ДОС.

Віртуальні машини - цілий засіб при розробці й тестуванні крос-платформних програм. Вони також використовуються для налагодження модулів ядра або самої операційної системи.

Для таких систем характерні високі накладні витрати й порівняно низька надійність. Тому вони рідко застосовуються для промислового викотистання.

У системах віртуальних машин приділяється велика увага емуляції роботи апаратури.

Часто ці системи є підсистемами ОС загального призначення: наприклад підсистема W0W в Windows NT, Windows 2000, емулятор RT-II в VAX і т.д.

## Системи реального часу

Це системи, призначені для полегшення розробки так званих програм **реального часу** – програм, які керують некомп'ютерним устаткуванням, часто із жорсткими тимчасовими обмеженнями.

Здатність гарантувати час реакції є відмінною ознакою системи реального часу. Важливо враховувати різницю між гарантованістю й просто високою продуктивністю й низькими накладними витратами. Далеко не всі алгоритми й технічні рішення, навіть ті, які забезпечують відмінний середній час реакції, підходять для програм й ОС реального часу.

Типовими представниками цього класу систем є відомі системи OS-9 й OS-9000.

По суті, мультимедійні алгоритми (тобто ті, що потребують синхронізації звуку та зображення на екрані) також є системами реального часу. Розбіжність звуку й зображення фіксується людиною вже в межах 30 мсек.

## Засоби крос-розробки

Це алгоритми, призначені для розробки програм у двохмашинній конфігурації.

При цьому редагування, компіляція й налагодження виконуються на інструментальній машині, а потім скомпільований код завантажується в цільову машину.

Прикладами таких ОС є системи програмування мікроконтролерів Intel. Такі системи, як правило, містять у собі:

* набір компіляторів й асемблерів, що працюють на інструментальній машині з “нормальною” ОС;
* бібліотеки, які виконують велику частину функцій ОС при роботі програми (крім функції завантаження);
* засоби налагодження.

## Системи проміжних типів

Є системи, віднести які до одного з перерахованих типів важко. Найбільш відомими з таких систем є MS Windows 3.x й Windows 95. Вони, як ОС використовують апаратні засоби процесора для захисту й віртуалізації пам'яті, та навіть забезпечують дещо накшталт багатозадачності, але не захищають себе й програми від помилок інших програм (тобто поводяться в цьому сенсі як ДОС).

## Сімейства операційних систем

Часто існує спадковість між різними ОС.

Така спадковість обумовлена вимогами сумісності й переносимості прикладного ПО, а також запозиченням окремих вдалих рішень.

На основі такої спадковості можна побудувати “генеалогічні дерева” ОС.

Так, наприклад, можна виділити кілька сімейств нині експлуатованих ОС, а також сімейств, які вже вимерли або вимирають.

Процвітають:

* системи для великих комп'ютерів IBM OS-360, IBM-Vm, OS/2;
* сімейство UNIX, у тому числі ОС Linux.

# Архітектура ОС

Найбільш загальним підходом до структуризації операційної системи є поділ всіх її модулів на дві групи:

1. Ядро - модулі, що виконують основні функції ОС.
2. Модулі, що виконують допоміжні функції.

Модулі ядра виконують:

* + керування процесами
  + керування пам'яттю
  + керування пристроями введення - виведення

Ядро виконує такі функції, як перемикання контекстів, завантаження/вивантаження сторінок пам'яті, обробку переривань. Ці функції недоступні для програм. Вони створюють для них так назване **прикладне програмне середовище***.* Додатки звертаються до ядра із запитами - системними викликами - для виконання тих або інших дій (наприклад, відкриття й читання файлу, виведення графіки, одержання системного часу й т.п.).

Функції ядра, які можуть викликатися додатками, утворюють **інтерфейс прикладного програмування***.*

Функції, виконувані модулями ядра - найбільш часто використовувані, тому швидкість їхнього виконання визначає продуктивність всієї системи в цілому. Тому більша частина модулів ядра є резидентними, тобто постійно перебувають в оперативній пам'яті.

**Допоміжні модулі ОС** оформлюються або у вигляді програм, або у вигляді бібліотек процедур.

Оскільки частина модулів ОС виконуються як звичайні додатки (тобто в стандартному для даної ОС форматі), то часто буває складно провести чітку грань між ОС і додатками.

Рішення про те, чи є яка-небудь програма частиною ОС чи ні, приймає виробник ОС.

Допоміжні модулі ОС поділяються на наступні групи:

* утиліти - програми, що вирішують окремі завдання керування й супроводу комп'ютерної системи (наприклад, програми стискання дисків, архівування даних);
* системні обробні програми - текстові й графічні редактори, компілятори, компоновщики, наладчики;
* програми надання користувачеві додаткових послуг - спеціальний користувальницький інтерфейс, калькулятор і т.п.
* бібліотеки процедур - спрощують розробку програм (наприклад, бібліотека математичних функцій, функцій введення - виведення й т.п.).

Допоміжні модулі ОС звертаються до функцій ядра за допомогою системних викликів.

Поділ операційної системи на ядро й модулі-додатки забезпечує легку розширюваність ОС.

# Утиліти

# Библіотеки

програм

Програми користувача

Системні

обробні програми

Рис. 3.1. Загальна структура ОС

Допоміжні модулі ОС завантажуються в оперативну пам'ять тільки на час виконання своїх функцій, тобто є **транзитними**.

Важливою властивістю архітектури, заснованої на ядрі, є можливість захисту кодів і даних операційної системи за рахунок виконання функцій ядра в привілейованому режимі.

Забезпечити привілей ОС неможливо без спеціальних засобів апаратної підтримки. Апаратура комп'ютера повинна підтримувати як мінімум два режими роботи: користувальницький режим і привілейований режим, його також називають режимом ядра або режимом супервізора*.*

Додатки ставляться в підлегле положення за рахунок заборони виконання в користувальницькому режимі деяких критичних команд, пов'язаних з перемиканням процесора із завдання на завдання, керуванням пристроями введення - виведення, доступом до механізмів розподілу й захисту пам'яті.

Умова дозволу виконання критичних інструкцій перебуває під контролем ядра й забезпечується за рахунок набору інструкцій, безумовно, заборонених для користувальницького режиму.

Аналогічно забезпечуються привілеї ядра при доступі до пам'яті. Наприклад, виконання інструкції доступу до пам'яті для додатка дозволяється, якщо інструкція звертається до області пам'яті, відведеної ОС даному додатку, і забороняється при звертанні до інших областей.

Між кількістю рівнів привілеїв, реалізованих апаратно, і кількістю рівнів привілеїв, підтримуваних ОС, немає прямої відповідності.

Так, наприклад, на базі чотирьох рівнів, забезпечуваних процесорами Intel, ОС OS/2 будує трьохрівневу систему привілеїв, а Windows NT - дворівневу.

З іншого боку, якщо апаратура підтримує хоча б два рівні, програмним способом можна побудувати ОС із як завгодно розвиненою системою захисту.

Розглянута архітектура ОС, заснована на привілейованому ядрі й додатках користувальницького режиму, стала класичної. Її використовують відомі ОС: UNIX, VMS, OS/390, OS/2, Windows NT.

У деяких випадках розроблювачі відступають від цієї класики, і привілейований режим використовується й для програм ОС.

У цьому випадку обіг програм до ядра здійснється швидше, але при цьому відсутній надійний апаратний захист пам'яті.

## Багатошарова структура ОС

Обчислювальну систему, що працює під керуванням ОС на основі ядра, можна розглядати як систему, що складається із трьох, ієрархічно розташованих, шарів:

* нижній шар - утвориться апаратурами;
* проміжний - ядро;
* верхній шар - модулі, що реалізують допоміжні функції.

– ядро ОС

– апаратура

– допоміжні функції

Рис.3.2. Трьохшарова схема обчислювальної системи

Шарувату структуру, зображену на малюнку, зручно представляти для ілюстрації того факту, що кожен шар може взаємодіяти тільки із сусіднім шаром. При такій організації додатки не можуть безпосередньо взаємодіяти з апаратурами, а тільки через шар ядра.

**Багатошаровий підхід** є універсальним й ефективним способом декомпозиції складних систем. Відповідно до нього система складається з ієрархії шарів. Кожен шар обслуговує выще лежачий шар, виконуючи для нього деякий набір функцій, які утворять міжшарний інтерфейс.

до шару k+2

шар k+1

шар k

міжшаровий

интерфейс

До шару k-1

Рис.3.3. Концепція багатошарової взаємодії

На основі функцій нижче лежачого шару, наступний шар будує свої функції - більш складні й потужніші, які, у свою чергу, стають основою для створення ще більш потужних функцій вище лежачого шару.

Суворі правила взаємодії обумовлюються тільки між шарам. Усередині шару зв'язки між модулями можуть бути довільними.

Окремий модуль може виконувати свою роботу як самостійно, так і звернутися до іншого модуля свого шару, або до нижче лежачого шару через міжшаровий інтерфейс.

Переваги такої системи організації:

* спрощується розробка системи;
* при модернізації системи легко робити зміни усередині кожного шару, не зважаючи на інші шари.

Оскільки ядро являє собою складний багатофункціональний комплекс, цей підхід поширюють і на структуру ядра.

Наприклад:

* **Засоби апаратної підтримки ОС***.* Сюди відносять не всі апаратні засоби, а тільки ті, які прямо беруть участь в організації обчислювальних процесів: засоби підтримки привілейованого режиму, системи переривань, засоби перемикання контекстів процесів, засоби захисту областей пам'яті.
* **Машинно-залежні компоненти ОС***. Шар* утворюють програмні модулі, у яких відбивається специфіка апаратної платформи комп'ютера. В ідеалі цей шар повністю відділяє вище лежачі шари ядра від особливостей апаратур. Це дозволяє розробляти вище лежачі шари на основі машинно-незалежних модулів для всіх типів апаратних платформ, підтримуваних даною ОС.
* **Базові механізми ядра.**Виконує найбільш примітивні операції ядра, такі як перемикання контекстів процесів, диспетчеризацію переривань, переміщення сторінок пам'яті на диск і назад і т.п. Модулі даного шару не ухвалюють рішення щодо розподілу ресурсів - виконавчі механізмами для модулів верхніх шарів.
* **Менеджери ресурсів.** *Шар* складається з потужних функціональних модулів, що реалізують стратегічні завдання по керуванню основними ресурсами обчислювальної системи. Тут працюють менеджери (диспетчери) процесів, введення - виведення, файлової системи й оперативної пам'яті. Розбивка на менеджери може бути різною. Наприклад менеджер файлової системи іноді поєднують із менеджером введення - виведення, а функції керування доступом користувачів до системи в цілому і її окремим об'єктам доручають окремому менеджеру безпеки.

Кожний з менеджерів веде облік вільних і використовуваних ресурсів певного типу й планує їхній розподіл відповідно до запитів програм.

Для виконання ухвалених рішень менеджер звертається до нижче лежачого шару. Усередині шару менеджерів існують тісні взаємозв'язки, що відбивають той факт, що для виконання процесу потрібен доступ одночасно до декількох ресурсів - наприклад, процесору, пам'яті, введення - виведення й т.п.

* **Інтерфейс системних викликів.**Це самий верхній шар ядра. Він взаємодіє безпосередньо з додатками й системними утилітами, утворюючи прикладний програмний інтерфейс ОС. Функції, які обслуговують системні виклики, надають доступ до ресурсів системи в зручній і компактній формі, без вказівки деталей їхнього фізичного розташування. Для здійснення таких дій системні виклики звичайно звертаються по допомогу до функцій шару менеджерів ресурсів і т.д.

Наведена розбивка ядра ОС на шари - досить умовна. Така розбивка й розподіл функцій може бути іншим.

Також може бути іншим спосіб взаємодії шарів. Для прискорення роботи ядра в ряді випадках відбувається обіг з верхнього шару до функцій шарів нижнього рівня, минаючи проміжні.

Вибір кількості шарів ядра є відповідальною й складною справою: збільшення числа шарів веде до вповільнення роботи ядра, а зменшення - погіршує розширюваність і логічність системи.

Незалежно від того, які функції виконує ОС, вона повинна задовольняти експлуатаційним вимогам. Вона, зокрема, повинна мати наступні якості:

1. **Надійність.** У випадку помилки в програмному або апаратному встаткуванні система повинна виявити помилку й або спробувати виправити положення. Або сповістити про це користувачеві й намагатися звести до мінімуму збиток.
2. **Захист.** Користувач не хоче, щоб інші користувачі (якщо він, наприклад, працює в мережі) йому заважали. Тому ОС повинна захищати користувача від впливу чужих помилок і від спроб злочинного втручання.
3. **Ефективність.** ОС - досить складна програма, що використовує значну частину ресурсів для своїх власних потреб. Ресурси, які споживають ОС, не надходять у розпорядження користувача. Отже ОС повинна бути якомога економнішою. Крім того, вона повинна керувати ресурсами користувачів так, щоб звести до мінімуму час затримки й простоїв.
4. **Передбачуваність.** Вимоги, які користувач може пред'являти до системи, у більшості випадків, непередбачувані. Але користувач бажає, щоб обслуговування не дуже сильно змінювалося протягом тривалого часу. Зокрема, при введенні програми в машину користувач повинен мати, засноване на попередньому досвіді, приблизне уявлення про те, коли йому варто очікувати видачі результатів.
5. **Зручність**. Все ясно, як і те, що універсальних зручностей не існує. Тут може йти мова про певний клас завдань.

Декілька слів про виконувані функції:

**Розподіл процесора.** У випадку немудрої системи вся вона розподіляється як єдиний ресурс. Користувач або розпоряджається машиною, або чекає, коли вона буде надана в його розпорядження. Таку стратегію дуже легко організувати, але вона не буде ефективно використовувати устаткування. Для того щоб працювати паралельно із процесором, можна зробити так, щоб одна програма виконувала операції введення - виведення, поки інша займає головний процесор. Хоча реалізація такого підходу - більш складна, у наявності перевага - кожен пристрій використовується більш інтенсивно.

**Керування пам'яттю.** Керування пам'яттю тісно пов'язано з розподілом процесора. Програми можуть працювати тільки тоді, коли вони перебувають в оперативній пам'яті, але не обов'язково їх тримати там, якщо надія одержати процесор - невелика. У цьому випадку виявиться, що пам'ять, яку вони займають - пропадає даремно.

Оперативна пам'ять - це ресурс, що розподіляє теж. Тому система витрачає час для того, щоб розташувати інформацію раціонально, намагаючись тримати корисні програми в оперативній пам'яті й знищувати “вільні проміжки” між програмами. Для цього система може використовувати переміщення програм. Це робиться для того, щоб зменшити обсяг даремно використовуваної пам'яті. Переміщення легше здійснити, якщо використовувати спеціальні стратегії організації пам'яті. Ці стратегії дозволяють ОС досить гнучко регулювати обмін інформацією між оперативною й допоміжною пам'яттю.

**Зовнішні пристрої.** Методи розподілу пристроїв введення - виведення й каналів зв'язку різні. Завдання використовує периферійний пристрій стільки часу, скільки їй потрібно, а потім відмовляється від нього. Пристрої зі швидким довільним доступом, такі як, накопичувачі на дисках, можна спільно використовувати в декількох завданнях за принципом “операція за операцією”, тобто декільком завданням дозволяється використовувати пристрій поперемінно. Якщо, скажімо, два завдання спробують виконати операцію введення - виведення на одному пристрої одночасно, то виникають черга й затримка.

Підхід до розподілу пристроїв впливає як на правильність, так і на ефективність роботи.

Стратегія розподілу впливає й на ефективність використання пристроїв, наприклад, того ж диска.

Ефективний розподіл периферійних пристроїв важко реалізувати з двох причин:

*По-перше,* за допомогою існуючих математичних методів не можна провести необхідні дослідження й відшукати оптимальні способи розподілу декількох різних типів пристроїв для загального випадку рішення завдань.

*По-друге,* ефективність стратегії розподілу дуже важко виміряти.

Можна оцінити тільки загальний зовнішній прояв неправильного розподілу. Якщо ж при оцінці враховувати й вплив взаємодії з оперативною пам'яттю й центральним процесором, то аналітичні й емпіричні методи виявляються ще менш перспективними.

Методи розподілу пристроїв введення - виведення, контролерів і каналів сильно залежить від пристроїв.

**Програмні ресурси.** Часто в операційних системах є системи прикладних програм і бібліотеки програм користувачів. Будучи ресурсами, що підлягають розподілу, ці бібліотеки мають багато загального з апаратними ресурсами.

Спільне використання інтерпретаторів, редакторів текстів і т.п. можна організувати, якщо ці програми допускають паралельне використання.

Якщо в них робоча частина повністю відділена від даних й операції запису на згадку застосовуються тільки до розділу даних, то такі програми допускають паралельне використання. Якщо ж програма не допускає паралельного використання, то кожен її екземпляр може бути (доступний)? в цей момент тільки одному користувачеві, так само, як і будь-який апаратний пристрій.

При паралельному використанні кожному користувачеві виділяється особистий екземпляр розділу даних, а з єдиним екземпляром робочої частини всі користувачі можуть працювати в режимі поділу.

## Супервізор

Традиційний підхід при проектуванні ОС полягає в тому, що безліч процесів, що виконують основні функції системи, підкоряються головній програмі, що називається супервізором. Здійснюючи централізоване керування, супервізор зв'язує воєдино інші частини системи. Він організовує спільну роботу програм, встановлюючи привілей або призначаючи покарання. Він забезпечує засоби зв'язку й синхронізації між процесами й фізичними пристроями. Звичайно й повідомлення, передані від процесу до процесу, і запуск, і закінчення роботи пристроїв, і сигнали від устаткування обробляються супервізором.

Супервізор, звичайно, керує поділом усіх ресурсів і послуг системи між користувачем(-ами)?.

**Функції супервізора.** Функції супервізора можна загалом розділити на чотири області:

* контроль і керування;
* організація зв'язків;
* захист й обмеження;
* обслуговуючі програми;

**Контроль і керування.** В обов'язки супервізора входить встановлення послідовності й контроль керування завданнями у системі. Для обробки завдань служать: планування порядку виконання завдань, облік споживання або ресурсів й інтерпретація мови керування завданнями. Ці функції можна реалізувати незалежно від супервізора або підкорити йому.

Користувач описує свої вимоги на деякій мові керування завданнями. Супервізор інтерпретує ці вимоги й повідомляє різним програмам, що розподіляють ресурси, які саме ресурси й завдання запитано завданнями. Розподільники ресурсів обслуговують запити у порядку їхніх пріоритетів. Програма обліку запам'ятовує кількість спожитих ресурсів.

**Організація зв'язку.** У супервізорі передбачені засоби зв'язку між різними програмами, коли дві незалежні програми такі, як файлова система й програми керування пам'яттю хочуть зв'язатися один з одним, вони повинні просити супервізор установити контакт. Після встановлення первісного контакту програмам дозволяється передавати один одному повідомлення, прийнятим у даній системі способом, скажімо, за допомогою “листів” і загальної “поштової скриньки”. Первісний зв'язок встановлюється через супервізор. Доступна тільки супервізору інформація дозволяє йому також встановлювати контакти між системними програмами й програмами користувачів.

**Захист й обмеження.** Для забезпечення гарантій виконання роботи супервізор повинен накласти деякі обмеження, як на систему, так і на користувачів. Є ряд програм, що обробляють сигнали апаратури, особливо відхилення від нормальних умов. Наприклад, під час роботи програми копіювання може виникнути кілька десятків різних особливих ситуацій, наприклад, збій при читанні або записі, неготовність дисководів до читання або запису, відсутність місця на дискеті для копійованого файлу й т.д.

Коли надходить сигнал про помилку (звичайно це переривання), супервізор повинен прийняти рішення, повторити або навіть викинути завдання. Для всіх цих ситуацій необхідно передбачити відповідні повідомлення й коригувальні дії.

Обмеження повинні накладати на час роботи програми й кількість видаваних нею результатів.

Супервізор може також організувати систему захисту за допомогою спеціальних “паролів”.

Доступ до захищених ресурсів дозволяється тільки по відповідному паролі.

З метою захисту супервізора в деяких системах існують два типи режиму роботи:

* режим супервізора;
* робочий режим.

Завдяки такій обережності супервізор має особливу владу над привілейованими частинами системи й програмами користувачів. Супервізор може захищати себе, зберігаючи життєво важливу інформацію в захищеній області пам'яті, доступної тільки в режимі супервізора.

**Обслуговування програми.** Крім розподілу ресурсів супервізор виконує деякі з функцій системи. У ньому є набір обслуговуючих програм для аварійних зборів, для доступу до бібліотек програм, для обробки повідомлень від програм, що працюють в оперативному режимі. Усередині супервізора часто перебуває область пам'яті, призначена для організації зв'язків і таблиці системи захисту. У ряді випадків, коли потрібно виконати складні дії, супервізор викликає спеціальні обробні програми.

Використання супервізора - це приклад побудови ОС на принципі централізованого керування.

Ідеологія централізації ключових функцій системи під контролем супервізора має свої переваги й недоліки. Одна з переваг - проста реалізація захисту. Друга перевага - простота реалізації.

З погляду розробки легше зосередити найважливіші системні функції в підпорядкуванні супервізора, чим розподіляти їх по всій системі. Найкраще такий підхід виправдує себе, коли супервізор роблять порівняно невеликим. На жаль, велика спокуса передати супервізору дуже багато функцій, перетворюючи його тим самим, власне, в ОС.

У централізованого супервізора є й істотні недоліки. Так програмам не дозволяється встановлювати зв'язок один з одним самостійно.

## Мікроядерна архітектура

Мікроядерна архітектура - альтернатива класичному способу організації у вигляді багатошарового ядра, що працює в привілейованому режимі.

Тут у привілейованому режимі залишається працювати тільки дуже невелика частина ОС, названа **мікроядром.** Мікроядро захищене від інших частин ОС і додатку.

Сервери ОС

Програми користувачів

Утиліти ОС

Ядро ОС

Привілейований

режим

Користувальницький режим

Рис.3.4. Мікроядерна архітектура.

До складу мікроядра входять :

1. Машинно-залежні модулі;
2. Модулі, що виконують основні базові функції ядра з керування процесами;
3. Обробка переривань;
4. Керуванню віртуальною пам'яттю;
5. Пересилання повідомлень і керування введенням – виведенням.

Інші функції ядра оформлюються у вигляді програм, що працюють у користувальницькому режимі. У загальному випадку багато менеджерів ресурсів, виконуються у вигляді модулів, що працюють у користувальницькому режимі.

Ці менеджери, однак, мають принципову відмінність від традиційних утиліт й обробних програм ОС, хоча при мікроядерній архітектурі всі ці програмні компоненти також оформлені у вигляді програм. Утиліти й обробні програми викликаються в основному користувачами. За визначенням, основним призначенням такого додатка є обслуговування запитів інших програм, тому менеджери ресурсів, винесені в користувальницький режим, називаються **серверами ОС**.

Для реалізації мікроядерної архітектури необхідною умовою є наявність в ОС зручного й ефективного способу виклику одного процесу з іншого. Підтримка такого механізму - одне з головних завдань мікроядра.

##### Переваги мікроядерної архітектури

* Висока переносимість;
* Розширюваність;
* Висока надійність;
* Гарні передумови для підтримки розподілених програм;

Так, як використовуються механізми, аналогічні мережним: взаємодія клієнтів і серверів шляхом обміну повідомленнями, сервери мікроядерної ОС можуть працювати як на одному, так і на декількох процесорах.

**Недоліки мікроядерної архітектури**

Більш низька продуктивність. Це позначається на швидкості роботи прикладних середовищ, а значить і на швидкості виконання програм.

**Висновок:**

Мікроядерні апаратури є альтернативою класичному способу побудови ОС, що передбачає виконання своїх основних функцій ОС у привілейованому режимі. У мікроядерних ОС у привілейованому режимі залишається працювати дуже невелика частина ядра, що названа мікроядром. Всі інші функції ядра оформлюються у вигляді програм, що працюють у користувальницькому режимі.Мікроядерні ОС задовольняють більшості вимог, пропонованих до зроблених ОС, володіючи переносимістю, розширюваністю, надійністю й створюють гарні можливості для підтримки розподілених програм.

За ці переваги доводиться розплачуватися зниженням продуктивності, це і є основний недолік мікропроцесорної архітектури. Мікроядерну концепцію використають такі ОС, як Windows NT і деякі версії ОС UNIX.

**Додаткова література:**Андрій Рабаневский “Операційна система UNIX” Спб.: БХВ - Санкт-Петербург, 1999.

**Керування процесами і потоками**

## Процеси

Процес - основна одиниця роботи в ОС. Розглянемо спочатку найбільш знайомий тип роботи - послідовну програму.

Велику послідовну програму , написану алгоритмічною мовою, можна підрозділити на менші послідовні програми, названі процедурами. Якщо рекурсія виключена, то процедура ніколи не може приводитися в дію декількома обігами одночасно. Отже, будь-якій роботі, що виконується в цей момент, однозначно відповідає діюча процедура.

Однак, в ОС цей простий зв'язок між виконуваними роботами й процедурами - втрачено. Одна процедура може працювати одночасно в декількох частинах системи. Наприклад, кілька каналів, пов'язаних з дисками, можуть розділяти одну процедуру введення й навпаки, функція, що представляє за змістом єдину операцію, наприклад “введення”, може щораз по ходу роботи використовувати одну або кілька різних процедур.

Взаємно однозначної відповідності між процедурами й виконуваними роботами, що існує в мовах послідовного програмування, в ОС немає. Тому поняття процедури й роботи не можна вважати еквівалентними.

Ця паралельність відбивається й на програмному устаткуванні. ОС виконує безліч робіт, які рухаються вперед майже незалежно: введення, висновок, обчислення на центральному процесорі.

Поняття процесу – формалізація ідеї “незалежної роботи”.

Периферійні пристрої й пов'язані з ними канали працюють паралельно із блоком центрального процесора. У самому центральному процесорі теж можна широко використовувати паралелізм апаратного устаткування. У системах апаратного устаткування з декількома арифметичними процесорами й більшим числом каналів, що працюють паралельно, організація робіт стає складним завданням.

На додаток до безлічі апаратних пристроїв, що працюють паралельно, може бути ще кілька завдань, які також виконуються паралельно.

Ці паралельно виконувані завдання взагалі не повинні враховувати існування один одного. Коли якесь завдання віддає наказ “друкувати”, його не стосується те, що інше в цей час здійснює запис на диск і т.д.

Щоб одержати адекватну модель такого функціонування, потрібно ввести одиницю роботи, яка б відбивала цю паралельність, а це і є **процес**.

Оскільки, роботи виконуються незалежно і їхні процеси, що представляють, працюють із різними швидкостями, то повинні допускатися довільні співвідношення швидкостей виконання процесів.

Отже, **процеси** – незалежні роботи, які виконуються паралельно й з різною швидкістю.

Роботам, іноді, все-таки необхідно обмінюватися інформацією. Тому вони не цілком незалежні. Однак спілкування між процесами повинне проходити тільки за певними чіткими схемами, щоб уникнути плутанини й невизначеності.

Крім того, робиться допущення, що операції усередині процесу виконуються суворо в певній послідовності.

Варто спробувати дати формальне визначення поняттю процесу, оскільки існує дуже багато визначень цього поняття.

Нехай – набір змінних, що характеризують стан, названий набором змінних станів. **Стан** описується завданнями значень всіх елементів, що входять у набір змінних станів./ **Стан** описуєт завдання значень всіх елементів, що входять у набір змінних станів.



**Простір станів** для даного набору змінних станів – безліч станів, які може приймати цей набір змінних.

**Дія** – присвоювання значень деяким зі змінних даного набору. Послідовність станів, що належать простору станів, називається  **роботою**.

Один зі способів зробити роботу полягає в послідовному застосуванні різних дій. Кожна дія породжує новий стан, що за визначенням і є роботою.

**Функція дії** – це функція, що відображає стан у дії. Функція дії може також породжувати роботу із заданого початкового стану. Вони просто описують дію, яку треба застосувати до кожного чергового стану, а ця дія породжує новий стан і т.д. нескінченно.

На змістовному рівні: набір змінних - пам'ять, стан - вміст пам'яті, функція дії - програма.

Отже, можна визначити **процес** як трійку: простір станів, функцію дії в цьому просторі станів, і особливі елементи цього простору станів – **початковий стан**.

**Приклад:**

Нехай заданий процес **P**, який має дві змінні **x** й **y**. Роботу процесу Р можна описати послідовністю станів: .



Роботу процесу можна також описати, указавши простір станів

{(i,j), де i,j – натуральні числа}, один з початкових станів і функцію дії, що над всіма станами виконує дії .



Кожен стан процесу - це “моментальний знімок” ходу роботи, що виконує процес. Нехай програма разом зі своїми змінними перебуває в оперативній пам'яті, можна простежити за виконанням програми, спостерігаючи за пов'язаними з нею осередками й регістрами. Послідовність станів описує хід роботи програми в даному середовищі.

Відомо, якщо у двох процесів окремі набори змінних станів, то вони не можуть взаємодіяти. Спілкування між процесами забезпечується змінними, що поділяються

У ряді ОС крім поняття процес виділяється ще більш дрібна одиниця, що називається **потік**.

У чому ж різниця?

В ОС, де існують і процеси, і потоки, процес розглядається, як заявка на споживання всіх видів ресурсів, крім одного процесорного часу.

Цей найважливіший ресурс розподіляється ОС між потоками, які одержали свою назву завдяки тому, що вони являють собою послідовності (потоки виконання) команд. **Потоки** виникли як засіб розпаралелювання обчислень.

У найпростішому випадку процес складається з одного потоку. Надалі ми будемо, з метою спрощення, ототожнювати ці поняття й користуватися терміном **процес.**

## Реалізація процесів

Кожен програмний процес однозначно визначається інформаційною структурою, названою **дескриптором процесу.** У типовій системі дескриптор процесу складається з:

1. Змінної стану, що визначає положення процесу (готовий до роботи, працюючий, заблокований).
2. Захищеної області пам'яті, у якій перебувають поточні значення регістрів, коли процес переривається, не закінчивши роботи.
3. Інформації про ресурси, якими процес володіє або має право користуватися.

Крім цього, у дескрипторі процесу може бути відведене місце для організації спілкування з іншими процесам.

Дуже важливо розрізняти абстрактний і програмний процеси.

Програмний процес - це й абстрактний процес, а зворотне не завжди вірно.

Дескриптор й область пам'яті, з якої складається програмний процес, повинні бути виділені з наявних у машин ресурсів.

Є два підходи до утворення програмних процесів.

Можна побудувати систему з фіксованим числом програмних процесів, які існують завжди. У такому випадку програмні процеси утворяться одночасно із системою.

Щоб виконати роботу, необхідно тільки одержати у своє розпорядження один з існуючих програмних процесів.

Абстрактних процесів може бути більше, ніж є програмних процесів. Тому абстрактному процесу, можливо, доведеться очікувати, коли йому буде наданий програмний процес для виконання роботи.

*Інший шлях* полягає в тому, що в системі передбачають механізм, що утворює і знищує програмні процеси, коли надходить відповідний запит.

Цей механізм, який сам програмним процесом не є, дає системі можливість маніпулювати програмними процесам. Він називається **стрижнем**.

У процесі можна викликати утворення або знищення інших процесів. Так один процес може забажати, щоб стрижень утворив інший процес. Стрижень заводить дескриптор нового процесу, виділяє для процесів пам'ять і повідомляє про завершення цієї діяльності утворюючий процес. Тоді утворюючий процес поміщає на згадку нового процесу програму (тобто подання деякої функції дії). У стрижні передбачена команда “запустити”, яка процесу виділяє процесор.

Аналогічно команда “зупинити” у процесу відбирає процесор, а команда “знищити” відбирає дескриптор і ресурси.

У системі з єдиним процесором може існувати кілька процесів. При цьому стрижень дає кожному процесу можливість користуватися процесором у певні моменти часу.

Процеси можуть існувати як не зв'язані один з одним одиниці, або можуть бути зв'язані особливими відносинами, створюючи структури. Якщо система не передбачає такої структурної організації, то супервізор зобов'язаний у міру потреби утворювати процеси. З появою завдання супервізор утворить процес для виконання завдання. Коли завдання виконане, процес знищується. При такій організації всі процеси рівні.

У більш складних системах процеси можуть бути нерівними. Наприклад, вони можуть утворювати деревоподібну структуру.

У системі з деревоподібною структурою процес називають **батьком** всіх процесів, які він утворить, і процес називають **сином** того процесу, що його утворив. Тут має місце визначення предків і нащадків. **Генеалогічне дерево** процесів у системі – це орієнтований граф, де кожен процес представляється вершиною, а дуга виходить із вершини **A** і заходить у вершину **B** тоді й тільки тоді, коли **A** – батько **B**. **Генеалогічне дерево** описує впорядкованість процесів усередині системи у будь-який момент часу.

У системі з деревоподібною структурою можуть бути дуже суворі правила щодо передачі ресурсів й організації керування.

Наприклад, при утворенні кожен процес може одержати лише ті ресурси, які “належать” його батькові. Батько може мати право контролювати дії своїх синів і приймати дії для виправлення ненормальних ситуацій.

Рис. 4.1. Дерево утворення процесів

У такої структурованої системи є переваги:

* Розподіл ресурсів перебуває під суворим контролем. Ресурси всякого процесу були колись власністю кожного з його предків. Якщо процес хоче звільнити деякі зі своїх ресурсів, він може віддати їх тільки своєму батькові. Ніколи не буває неясностей, який ресурс належить якому процесу.
* Вся структура в цілому така, що в одних процесів більше влади, чим в інших, тобто в ній передбачений простий механізм поділу праці.
* Завдяки наявності структури завжди ясно, якому процесу належить керування: батько управляє синами.

У безструктурній системі все знає й робить супервізор. Хоча централізоване керування допускає велику волю при розбивці завдань, вона ускладнює ведення обліку, тому що супервізору доводиться стежити за кожним процесом у системі.

## Спілкування між процесами

Якщо ми хочемо використовувати ефективно апаратні й програмні ресурси. Вони підлягають поділу.

Розглянемо завдання користувача, що виконується в системі, як єдиний програмний процес. Рідко буває, коли цьому процесу протягом усього часу існування був потрібно який-небудь апаратний пристрій для одноособового користування.

Допустимо, він використовує канал для роботи з периферійним пристроям. Якщо канал обслуговує вимоги процесу скоріше, ніж останній їх породжує, є сенс організувати поділ єдиного каналу між декількома процесами.

Для зниження втрат через неефективне використання програмного устаткування можна також і тут скористатися поділом.

Візьмемо, наприклад, таку програму як компілятор. Якщо процесу потрібно відтранслювати програму, він може скопіювати програму компілятора у свою пам'ять і виконати компіляцію. Якщо цим компілятором користуються кілька процесів, то таке розмноження, з погляду ресурсів пам'яті, неоптимальне. Доцільно розділити за часом один екземпляр компілятора.

Помітимо, що в апаратних і програмних ресурсів є загальні властивості.

Подивимося на систему керування файлами, яка у кожен момент може обслуговувати запити тільки одного процесу.

Якщо ця система - програмний ресурс, її поділ організується точно так, як і поділ апаратних ресурсів.

Фізичні пристрої називаються **фізичними** або природними ресурсами.

Частина програмного забезпечення, що поводиться подібно фізичному ресурсу, називається **логічним ресурсом**.

Кожному із процесів, що функціонують у системі неважливо, які ресурси він використовує - фізичні або логічні.

Процесу байдуже, яким він користується ресурсом - фізичним або логічним, важливо, щоб ресурс робив те, що повинен.

# **Синхронізація процесів**

Незважаючи на те, що ресурси можна розділяти, вони звичайно доступні в кожен момент тільки одному процесу. Ресурс, що допускає обслуговування тільки одного користувача за один раз, називається **критичним ресурсом**.

Якщо кілька процесів хочуть використовувати критичний ресурс у режимі поділу часу, їм варто синхронізувати свої дії таким чином, щоб цей ресурс завжди перебував у розпорядженні не більш, ніж одного з них.

Якщо ресурс зайнятий, то інші процеси, яким він потрібний, тимчасово одержують відмову й повинні чекати, поки він не звільнитися.

Усередині кожного процесу можна виділити місця, у яких відбувається звертання до критичних ресурсів. Ці місця називаються **критичними ділянками**.

Розглянемо два незалежних процеси:

**parbegin**

Процес\_1: **do while** (true)

**begin**

Критична ділянка 1;

**end**;

частина, Що Залишилася, процесу 1;

**end**;

Процес\_2: **do while** (true)

**begin**

Критична ділянка 2;

**end**;

частина, Що Залишилася, процесу 2;

**end**;

Конструкція виду:

**parbegin**

Оператор 1;

Оператор 1;

...

Оператор 1;

**parend**.

Означає, що оператори 1(N виконуються паралельно.

До речі, записи, наведені вище, називаються записами на  **псевдокоді**.

Критичні ділянки процесів відповідають тим групам операцій, які звертаються до поділюваного критичного ресурсу.

Щоб не допустити одночасного виконання обох критичних ділянок, у системі повинен бути передбачений механізм, який би синхронізував ці два процеси. Цей механізм повинен володіти двома властивостями:

1. Якщо один або кілька процесів хочуть звернутися до своїх критичних ділянок, то один з них повинен одержати дозвіл увійти у свою критичну ділянку.
2. У кожен момент часу не більш, ніж одному процесу дозволяється перебувати у своїй критичній ділянці.

У процесів виникає не тільки проблема синхронізації, але й необхідність в обміні інформацією.

Назвемо словом “постачальник” процес, що відправляє порції інформації іншому процесу, що назвемо “споживач”.

Ці порції інформації можна розглядати як повідомлення.

Один з методів, застосовуваних при реалізації передачі повідомлень, полягає в тому, що заводить пул вільних буферів, кожний з яких може містити одне повідомлення.

Між процесами Постачальник і Споживач є черга заповнених буферів, що містять повідомлення. Коли Постачальник хоче послати повідомлення, він додає у хвіст цієї черги ще один буфер. Споживач, щоб одержати повідомлення, забирає із черги буфер, який знаходиться на її початку.

Така організація вимагає, щоб Постачальник і Споживач кооперувалися в багатьох областях. Вони повинні стежити за кількістю заповнених і вільних буферів. Постачальник може передавати повідомлення тільки доти, доки є вільні буфери. Споживач може одержувати повідомлення тільки, якщо черга не порожня.

Для обліку заповнених і вільних буферів потрібні поділювані змінні. Але раз змінні використовуються в режимі поділу, то будуть потрібні й критичні ділянки. Отже, змінювати значення лічильника заповнених буферів треба в режимі взаємного виключення.

Та ж проблема виникає при реалізації **черги повідомлень**. Нехай черга реалізована у вигляді зв'язного списку. Якщо Постачальник додає повідомлення в чергу саме в той момент, коли Споживач забирає з неї найостанніше повідомлення, може вийти неправильне посилання в списку.

Такі процеси як Постачальник і Споживач, що спілкуються на однакових правах, називаються **сопрограмами**. На жаль, у літературі цьому терміну дається дуже багато різних визначень. Щоб уникнути плутанини ми замість нього будемо користуватися терміном “ процеси, що кооперуються”.

Якщо процесу Постачальник важливо знати, що його повідомлення отримане, то Споживач повинен підтверджувати одержання кожного повідомлення, тобто Постачальник і Споживач поміняються місцями.

Якщо в системі є трохи пара “Постачальник - Споживач”, то можна організувати поділ вільних буферів, об'єднавши їхній загальний пул.

Зупинимося на рішенні проблем синхронізації й відносин типу “Постачальник - Споживач”.

## Синхронізація за допомогою елементарних прийомів нижнього рівня

1. **Блокування пам'яті.**

Взаємне виключення реалізують апаратно, зробивши операції над пам'яттю неподільними, тобто якщо кожен із двох процесів намагається помістити якісь значення в той самий осередок, то суперечка дозволяється апаратно: одному процесу дозволяється виконати операцію засилання негайно, а іншому доводиться чекати, поки перший не закінчить. Таке рішення називається **блокуванням пам'яті**.

Повернемося до розгляду прикладу із двома паралельними процесами.

Оскільки швидкості обох процесів - довільні, зазначені вище умови повинні виконуватися які б ні минулого швидкості роботи кожного процесу щодо іншого.

Здавалося б, немає нічого простіше, як запропонувати наступне рішення:

Перемикач\_1, Перемикач\_2:boolean;

**Begin**

Перемикач\_1:=False;

Перемикач\_2:=False;

**parbegin**

Процес\_1: **do** **while** (True);

Цикл\_1: **do** **while** (Перемикач\_2);

**end**;

Перемикач\_1:=True;

/\* Критична ділянка 1 \*/

Перемикач\_1:=False;

/\* частина, Що Залишилася, процесу 1\*/

**end**;

Процес\_2: **do** **while** (True);

Цикл\_2: **do** **while** (Перемикач\_1);

**end**;

Перемикач\_2:=True;

/\* Критична ділянка 2 \*/

Перемикач\_2:=False;

/\* частина, Що Залишилася, процесу 2\*/

**end**;

**parend**;

**End**.

Однак, оскільки швидкості довільні, допустимо, що Процес 2 працює набагато швидше, ніж Процес 1. Настільки швидше, що фактично після того, як Процес 1 виявляє, що в Перемикачі 2 стоїть ”неправда”, але перш ніж він встигне встановити значення “істина”, у Перемикачі 1, Процес 2 пробігає свою частину, що залишилася, і перескакує через Цикл 2 (оскільки в Перемикачі 1 усе ще стоїть значення “неправда”). Обидва процеси в такому випадку перейдуть до виконання (одночасно) своїх критичних ділянок, тобто в такому перемикальному випадку система буде працювати неправильно.

Пропоную вам самостійно спробувати змінити наведену програму так, щоб вона задовольняла обмеження, які накладають на рішення проблеми критичної ділянки.

Ми ж приведемо варіант рішення, запропонованого Деккером.

**Алгоритм Деккера**

integer: C1,C2,Черга;

**begin**

C1:=0;

C2:=0;

Черга:=1;

**parbegin**

Процес\_1: **begin** C1:=1;

**do** **while** (C2=1);

**if** Черга=2 **then**

**begin**

C1:=0;

**do** **while** (Черга=2);

**end**;

C1:=1;

**end**;

**end**;

Критична ділянка Процесу 1;

C1:=0;

Черга:=2;

частина, Що Залишилася, Процесу 1;

**end**;

Процес\_2: **begin** C2:=1;

**do** **while** (C1=1);

**if** Черга=1 **then**

**begin**

C2:=0;

**do** **while** (Черга=1);

**end**;

C2:=1;

**end**;

**end**;

Критична ділянка Процесу 2;

C2:=0;

Черга:=1;

частина, Що Залишилася, Процесу 2;

**end**;

**parend**;

**end**.

Тут введена додаткова змінна - Черга, що вказує, чия черга спробувати ввійти, за умови, що обидва процеси хочуть виконати свої критичні ділянки. Це рішення має узагальнення для випадку довільного числа процесів, що конкурують через критичний ресурс.

**Перевірка та установка**

Ця машинна операція значно спрощує рішення проблеми критичної ділянки.

До операції “перевірка й установка” звертаються із двома параметрами: “локальний” й “загальний”. Операція бере значення параметра “загальний” і присвоює його змінній “локальний”, а потім установлює змінній “загальний” значення рівне одиниці. Ця операція одна із апаратних засобів вирішення задачі критичного інтервалу. Вона уперше була реалізована у IBM-360. Там вона дістала назву T ( test and set ). Команда TS двоадресна. Її дія полягає у тому, що процесор присвоює значення другого операнда першому, після чого другому операнду присвоюється значення одиниці. Ця змінна common TS буде загальною для усіх процесів. Вона буде дорівнювати 1, коли один із процесів знаходиться у критичній області. Якщо процес хоче увійти у критичну область, його локальна змінна приймає значення 1.

Головна властивість цієї операції - її неподільність. Коли процес виконує операцію “перевірка й установка” ніяких дій не може відбутися між її початком і кінцем.

Змінна “загальний” розділяється між процесами, які підлягають синхронізації стосовно деякого критичного ресурсу. У кожного процесу є своя власна змінна “локальний”.

Якщо “загальний” = 1, це значить, що якийсь процес перебуває у своїй критичній ділянці. Початкове значення змінної “загальний” = 0.

Наведемо рішення проблеми методом “перевірка й установка”. У цьому решенні припускається, що в машині передбачене блокування пам'яті, тобто операція “загальний” := 0 - неподільна.

**Алгоритм перевірки й установки:**

Var ccommon, Locall, Local2: integer;

begin

common:= 1;

parbegin

ПР1: while true do

begin

Local1:= 1;

while Local1=1 do TS(Local1, common);

< критична ділянка ПР1>;

common:=0;

< залишок ПР1>;

end;

end;

ПР2: while true do

begin

Local2:=1;

while Local2=1 do TS(Local2, common);

< критична ділянка ПР2 >;

common:=0;

< залишок ПР2 >;

end

end;

parend.

Нехай першим хоче увійти у свою критичну ділянку ПР1. У цьому випадку Local1 спочатку прийме значення 1, а після циклу перевірки за допомогою команди TS(Local1, common)стане дорівнювати 0. При цьому значення common буде дорівнювати 1. ПР1 увійде у критичну ділянку. Після виходу з критичної ділянки common, що дасть можливість ПР2 увійти у свою критичну секцію.

Прицьому ми вважаємо, що операція common:=0 – неподільна завдяки операції *блокування памяті.*

Недолік цього методу у тому, що у циклі перевірки значення змінної common непродуктивно використовується процесорний час. Сітуація ще більше погіршується, якщо під час виконання ПР1, у його критичній секції виникне переривання, а ПР1 почне виконання. Він увійде у цикл перевірки і не вийде з нього доти, доки ПР1 не вийде з переривання.

Доречі, у мікропроцесорів Intel, починаючи з 80/386 існують спеціальні команди під назвами BTC ( bit test and reset), BTS, BTR, які є прототипами команди “превірка та установка”.

Обидва розглянутих методи синхронізації можуть виявитися досить неефективними, оскільки щоразу, коли один процес виконує свою критичну ділянку, будь-який інший процес, що теж хоче ввійти в критичну ділянку, попадає в цикл і повинен там очікувати дозволу.

При такому очікуванні в циклі, що називається **активним очікуванням**, дарма витрачається час центрального процесора.

Одним із методів, що дозволяють уникнути активного очікування, є метод використання **“семафорів”.**

**Семафори**

Семафор – ціла змінна, значення якої можуть міняти тільки операції **P** й **V**. Нехай **S** – семафор. Коли процес виконує операцію **P(S)**, **S** зменшується на одиницю й:

1. Якщо **S≥0**, то процес продовжує роботу.
2. Якщо **S<0**, то процес зупиняється й стає в чергу очікування, пов'язану з **S**. Він залишається заблокованим доти, доки операція **V(S)**, виконана іншим процесом, не звільнить його.

Коли процес виконує **V(S), S** збільшується на одиницю й:

1. Якщо **S>0**, процес продовжує роботу.
2. Якщо **S≤0**, то один процес вилучається із черги очікування й одержує дозвіл продовжити роботу. Процес, що звернувся до операції **V(S)**, теж може продовжувати роботу.

Крім того, операції **P** й **V** – неподільні. У кожен момент часу тільки один процес може виконувати операцію **P** або **V** над даним семафором. Тому якщо, **S=1** і два процеси одночасно спробують виконати операцію **P(S)**, то тільки одному з них буде дозволено продовжити роботу. Інший процес буде заблокований і поставлений у чергу до семафора **S**.

Раніше ми говорили про те, що стрижень системи – це механізм, що реалізує процеси. Стрижень повинен виділяти процесам і відбирати в них процесор. Операції **P** й **V** можна реалізувати усередині стрижня. Таким чином, забезпечується неподільність цієї операції.

Семафор, максимальне значення якого дорівнює одиниці, називається **двійковим семафором**.

За допомогою двійкового семафора процеси можуть організувати взаємне виключення за допомогою операцій **P(S)** і **V(S).**

Приклад:

Var ВІЛЬНИЙ:integer;

**begin**

ВІЛЬНИЙ:=1;

**parbegin**

Процес\_1: **begin** **do** **while** (True);

<Початок Процесу 1>;

P(ВІЛЬНИЙ);

<Критична ділянка 1>;

V(ВІЛЬНИЙ);

<частина, що залишилася Процесу 1>;

**end**;

**end**;

Процес\_2: **begin** **do** **while** (True);

<Початок Процесу 2>;

P(ВІЛЬНИЙ);

<Критична ділянка 2>;

V(ВІЛЬНИЙ);

<частина, що залишилася Процесу 2>;

**end**;

**end**;

**parend**;

**end**.

Тут **S** приймає значення 1, 0, -1. Якщо **S=1**, це значить, що жоден процес не перебуває у своїй критичній ділянці. Якщо **S=0** – один процес перебуває у своїй критичній ділянці. Якщо **S=-1** – один процес перебуває у своїй критичній ділянці, а другий – у черзі очікування.

Рішення, що використовують двійкові семафори, можуть бути застосовані так само й у випадку великої кількості процесів. Якщо алгоритм Деккера стає дуже складним більш, ніж для двох процесів, рішення із семафором залишається тривіальним. У цьому головна перевага семафора.

Таким чином можна реалізувати виконання семафорних операцій. Наприклад.

P(s): if s>=1 then s:=s-1;

else wait(s) { зупинити процес та помістити його у чергу очікування до семафору s}

V(s): if s<0 then Reliase(s) ;{помістити один з очікуючих процесів, що знаходиться у черзі до семафору s, у чергу готових до виконання процесів}

S:=s+1.

Wait означає, що супервізор OS повинен перевести задачу у стан очікування, тобто віправити її у чергу семафора s.

Виклик Reliase означає звернення до диспетчера задач з проханням перевести перший з процесів, що стоять у черзі до семафору s, у стан готовності.

Розглянемо приклад. Однопроцесорна система. У ній неподільність операцій P(s) та V(s) можна забезпечити блокуванням переривань. Семафор s можна реалізувати у вигляді запису з двома полями. У одному – значення змінної s , у другому – вказівник на список процесів, що заблоковані на семафорі s.

Type semafore= record

рахівник: integer;

вказівник: pointer

end;

var s:semaphore;

procedure P(var s: semaphore);

begin

< заблокувати переривання>;

s.рахівник:=s.рахівник-1;

if s.рахівник <0 then

wait(s);{вставити процес, що звернувся у список s.вказівник та надати на процесор процес з черги на виконання}

<дозволити переривання >

end;

procedureV( var s: semaphore);

begin

<заблокувати переривання>;

s.рахівник:=s.рахівник+1:

if s.рахівник<=0 then

Release (s); {Разблокувати перший процес із черги s.вказівник}

<дозволити переривання>;

end;

procedure InitSem(var:semaphore);

begin

s.рахівник :=1;

s.вказшвник := nil;

end;

Реалізація семафорів у багатопроцесорних системах більш складна. Одночасний доступ до семафора s з двох процесів ми блокували обмеженням переривань. Цей механізм у багатопроцесорній системі не працює. Ми не зможемо заблокувати доступ декількох процесів одночасно до семафору s за допомогою раніше використаного механізму. Тут потрібен механізм, що дозволяв би виключення доступу для декількох процесорів одночасно. Одним з рішень є використання раніше розглянутого методу “перевірки та установки”. Подробиці можна знайти у згадуваній книзі Гордєєва та Молчанова, Системне програмування.

За допомогою загальних семафорів легко вирішується проблема “Постачальник – Споживач” з використанням поштової скриньки. Подільні змінні, у цьому випадку лічильники вільних та зайнятих буферів, які повинні бути захищені від обох процесів.

Var s.вільно, s.заповнено, s.взаємовиключено: semaphore;

begin

Init Sem (s.вільно,N); { N-кількість комірок буферу}

Init Sem (s.заповнено,0);

Init Sem (s.взаємовиключено,1);

parbegin

Постачальник: while true do

begin

<підготувати повідомлення>;

P(s.вільно);

P(s.взаємовиключено);

<передача повідомлення>;

V(s.заповнено);

V(s.взаємовиключено);

end

end;

Споживач: while true do

begin

P(s.заповнено);

P(s.взаємовиключено);

<отримати повідомлення>;

V(s.вільно);

V(s.взаємовиключено>;

<обробка повідомлення>;

end

end

parend

end.

s.вільно, s.заповнено – числові семафори. Вони використовуються як лічильники вільних та зайнятих буферів.

s.взаємовиллючення – двійковий семафор.

За допомогою семафорів можна організувати синхронізацію процесів, коли завершення одного процесу пов’язано зочікуванням завершення іншого.

Нехай існують два процеси Пр1 та Пр2. Необхідно, щоб Пр1 запускав процес Пр2 та чекав його виконаня, тобто Пр1 не продовжував би свого виконання до тих пір, поки процес Пр2 до кінця не виконав би свою роботу.

Var s:semaphore;

begin

Init Sem(s,0);

Пр1: begin

<початок Пр1>;

ON(Пр2); {Поставити на виконання Пр2}

P(s);

<завершення Пр1>;

STOP

end;

Пр2: begin

<виконаня Пр2>;

V(s);

STOP

end

end.

Початкове значення семафору дорівнює ‘0’ , див. процедуру Init Sem.

ЯкщоПр1 почав виконуватись першим, то через деякий час він встановить на виконання процес Пр2. Після цього буде виконана процедура P(s) та він перейде у стан очікування на семафорі s. Процес Пр2 виконає усі необхідні дії та виконає процедуру V(s)ш відкриє семафор s. Після цього Пр1 знову готовий до виконання.

За допомогою загальних семафорів можна організувати керування ресурсами, наприклад, дисків і для синхронізації процесів.

Однак, при використанні семафорів для синхронізації й керування ресурсами виникають труднощі, пов'язані з організацією черги до семафора.

Якщо значення семафора **S** стає негативним, це значить, що своєї черги, пов'язаної з **S**, очікує один або кілька процесів.

Коли виконується чергова операція V(S), стрижень повинен вибрати, який процес треба взяти із черги. Стрижень системи може обслуговувати чергу в порядку надходження або використати пріоритетну схему обслуговування. Порядок обслуговування черги повинен обумовлюватися цілями, для яких застосовується даний семафор. Питання визначення дисципліни обслуговування при проектуванні ОС тісно пов'язаний з моделюванням обчислювальних процесів, для чого можуть бути ефективно використані моделі масового обслуговування.

Пріоритет процесу можна визначати динамічно по числу дорогих ресурсів, якими цей процес розпоряджується.

Вважається, що семафори - досить ефективний засіб для задоволення потреб ОС при реалізації синхронізації й спілкування між процесами.

Проте, часто **семафори** виявляються **незручним засобом**. Наприклад, складно реалізувати схему **передачі повідомлень між декількома процесами**.

## Елементарні прийоми синхронізації на верхньому рівні

1. **Поштові скриньки**

Якщо процес **Р1** хоче спілкуватися із процесом **Р2, Р1** просить систему утворити поштова скриньку, що зв'яже ці два процеси так, щоб вони могли передавати один одному повідомлення. Для того, щоб надіслати процесу **Р2** якесь повідомлення процес **Р1** просто поміщає це повідомлення в поштову скриньку, звідки процес **Р2** може його в будь-який час взяти. При застосуванні поштової скриньки процес **Р2**, зрештою, обов'язково одержить повідомлення, коли звернеться за ним, якщо взагалі звернеться.

**Поштова скринька** – це інформаційна структура, для якої задаються правила, що описують її роботу. Вона складається з **головного** **елемента**, у якому перебуває опис даної поштової скриньки, і з декількох **гнізд**, у які поміщають повідомлення. Розмір кожного гнізда й кількість гнізд, звичайно задаються при утворенні поштової скриньки. Правила роботи можуть бути різними, залежно від складності поштової скриньки.

У найпростішому випадку, повідомлення передаються тільки в одному напрямку. Процес **Р1** може посилати повідомлення доти, доки є вільні гнізда. Якщо всі гнізда заповнені, то процес **Р1** може або чекати, або зайнятися іншими справами й спробувати надіслати повідомлення пізніше.

Аналогічно **Р2** може одержувати повідомлення доти, доки є заповнені гнізда.

**Двосторонній зв'язок.** Використовується, якщо необхідно передавати підтвердження про одержання повідомлення. При цьому дозволяється передача повідомлень через поштову скриньку в обох напрямках. Якщо передавальний процес **Р1** працює швидше, ніж приймаючий процес **Р2, Р1** може заповнити всі гнізда, не залишивши **Р2** гнізд для відповідних повідомлень. Щоб цього не відбулося, вимагають, щоб відповіді пересилалися в тих же гніздах, у яких перебували їхні повідомлення, що викликали.

**Багатовходові поштові скриньки.** Використовують, коли декільком процесам необхідно спілкуватися з одним процесом. Прикладом може служити система керування файлами. Такі скриньки більш ефективні в порівнянні з тим, якби для кожного окремого процесу заводилася б своя поштова скринька, але реалізація більш складна.

**Порти.** Для відсилання повідомлення в поштову скриньку процес повинен знати його ім'я. Іноді це не зручно. **Порт** – це сполучна ланка між процесом і поштовою скринькою. Коли поштова скринька з'єднана з певним портом, процесам, щоб відіслати повідомлення, потрібно вказати тільки ім'я порту.

В одному з варіантів реалізації, у кожного із процесів може бути **вступний** й **вивідний** порти. Поштові скриньки створюються і знищуються процесами. Коли процес створює поштову скриньку, його зв'язують із якимось портом за допомогою команди “зв'язати”. Щоб скасувати зв'язок її заміняють зв'язком між фіктивним портом і поштовою скринькою відповідно. Власник поштової скриньки може його знищити. Пам'ять, що він займав, повертається процесу, що його надав.

Повернемось до семафору. Аналіз задач синхронізації показує, що очевидні переваги реалізації (простота, відсутність активного очікування) семафорні механізми мають цілий ряд недоліків. Так семафорні механізми не вказують безпосередньо на синхронізуючу умову, з якою він пов’язаний. Тому при побудові складних схем синхронізації алгоритми вирішення задач виявляються досить складними. Вирішенням проблеми може стати застосування монітора.

## Монітор Хоара

Монітор - набір процедур й інформаційних структур, яким процеси користуються в режимі поділу, причому в кожен момент ним може користуватися тільки один процес.

Монітор можна уявити собі як кімнату, від якої є тільки один ключ.

Якщо якийсь процес має намір скористатися цією кімнатою й ключ перебуває зовні, то цей процес може відімкнути кімнату, увійти й скористатися однією із процедур монітора.

Якщо ключа зовні немає, то процесу прийдеться чекати, поки той, хто користується кімнатою в цей момент, не вийде з неї й не віддасть ключ. Крім того, у кімнаті не можна залишатися назавжди.

Розглянемо, наприклад, ресурс, котрий розподіляє деяка програма - планувальник.

Щоразу, коли процес хоче одержати у своє розпорядження якісь частини ресурсу, він повинен звернутися до планувальника. Процедура-планувальник розділяє всі процеси, і кожен процес може в будь-який момент звернутися до планувальника. Але планувальник не в змозі обслуговувати одночасно більше одного процесу. Виходить, планувальник являє собою приклад монітора.

Іноді монітору необхідно затримати процес, що звернувся, якщо процес просить ресурс, що уже кимсь використовується, до надання ресурсу.

Варто підкреслити, що монітор – це пасивний об'єкт, як кімната. Це не процес. Монітор оживає тільки тоді, коли який-небудь процес вирішує скористатися його послугами. *Особливість* програми-монітора полягає в тому, що в будь-який момент її може виконати *тільки один процес.*

Таким чином, монітор це механізм забезпечення паралелізму, який має як дані, так і процедури, що необхідні для реалізації динамічного розподілення загального ресурсу або групи ресурсів.

Процес, якому потрібен ресурс, що поділяється, має звернутися до монітору, який або забезпечить доступ , або дасть відмову.

Вхід у монітор знаходиться під жорстким контролем - виконується взаємовиключення процесів,так як у кожний момент часу у монітор може увійти тільки один процес, інші процеси чекають, причому режимом очікування керує сам монітор. Він вказує умову, за якою процес має очікувати.

Внутрішні дані монітору можуть бути глобальними або локальними стосовнодо однієї процедури. До всіх даних або ресурсів можна звертатись тільки із середени монітора. Процеси можуть тільки звертатись до процедур та не можуть безпосередньо звертатись до даних.

Якщо процес звертається до деякої процедури і виявляє, що відповідний ресурс зайнятий, процедура видає команду очікування Wait з певною умовою очікування. Процес, що займав ресурс, може його звільнити, однак, може статися, що вже є процеси, які чекали звільнення цього ресурсу. Монітор виконує програму SIGNAL, за якою один з процесів, які стоять у черзі за цим ресурсом, можуть його отримати. Якщо деякий процес повертає ресурс і він нікому не потрібен, то нічого не виконується, а ресурс заноситься у список вільних ресурсів.

Іноді роблять так, щоб процес, який стоїть у черзі до ресурсу мав вищий пріоритет перед новим процесом, що хоче оволодіти цим ресурсом.

Розглянемо приклад виділення одного ресурсу.

monitor Resours;

condition free; {умова вільний}

var busy: boolean ; { busy - зайнятий}

procedure Reqest; {запит}

begin

if busy then Wait(free) ;

busy:=true;

Take off; {видати ресурс}

end;

Procedure Rlese; {звільнити}

begin

Take ON { отримати ресурс}

busy:= false;

SIGNAL(free);

end;

begin

buse:=false;

end.

Ресурс запитується та звільняється процедурами Request {запит} та Release {звільнити} . Якщо процес звернувся до процедури Request коли ресурс використовується busy:= true і ця процедура виконує команду Wait (free). Ця операція блокує не команду Request, а процес, що до неї звернувся. Коли процес, що використовує ресурс, звертається до монітора з процедурою Release , операція SIGNAL блокує процес, що стоїть першим у черзі за даним ресурсом. Він виконує процедуру Request.

Якщо черги нема, ніякі дії не виконуються.

При синхронізації процесів семантика монітора гарантує, що якщо хоча б один процес очікує виконання умови, то ніякий інший, що звернувся, не може вмішатися між сигналом про виконання цієї умови й продовженням рівно одного із процесів, що очікують.

У порівнянні із семафорами монітори мають переваги в тому, що:

* Монітори - дуже гнучкий інструмент. Наприклад, можна реалізувати з його допомогою поштову скриньку.
* Здійснюється локалізація всіх поділюваних змінних усередині тіла монітора, що дозволяє позбутися малоприємних конструкцій в процесах, які синхронізуються.

## Тупики

Ситуація, коли процеси чекають один одного невиразно довго, називається **тупиком**  або **дедлоком (**deadlock**)**.

Існує три основних напрямки політики запобігання тупиків:

* запобігання тупиків;
* автоматичне виявлення;
* виявлення при участі оператора.

Метод автоматичного виявлення тупиків допускає, щоб система потрапила в тупикову ситуацію, але є можливість це виявити програмним шляхом. Потім система відбирає ресурси в інших процесів і віддає їх на те, щоб зрушити з місця процеси, що потрапили в тупик.

Третій підхід заснований на тім, що тупикові ситуації виникають занадто рідко, щоб про них слід було турбуватися. Коли така ситуація все-таки виникає, оператор її виявляє й перезапускає систему. Іноді це обходитися занадто дорого, зокрема, дискредитує систему в очах користувача.

## Алгоритм запобігання тупикових ситуацій

Розглянемо систему, у якій процеси конкурують через стрічкопротягувальні пристрої, якщо процесу надати всі необхідні пристрої, які йому потрібні, він буде працювати протягом кінцевого відрізка часу, після чого він повертає всі викликані пристрої.

**Розглянемо приклад:**

Нехай двом процесам, що виконуються в режимі мультипрограмування для виконання роботи потрібно два ресурси, наприклад, принтер і послідовний порт. Така ситуація може виникнути, наприклад, під час роздруковування інформації, що надходить по модемному зв'язку.

Представимо фрагмент двох програм:

……..

Зайняти ПОРТ

Зайняти ДИСК

……..

……..

Звільнити ПОРТ

Звільнити ДИСК

……..

A1

A2

A3

A4

Процес А

……..

Зайняти ДИСК

Зайняти ПОРТ

……..

……..

Звільнити ДИСК

Звільнити ПОРТ

……..

B1

B2

B3

B4

Процес В

А1

Процес А

Процес В

А2

В1

В2

Блокування В

Блокування А

а)

А2

А3

В1

Блокування В

Блокування А

b)

А1

Переривання А

А4

Переривання А

Переривання А

В1

В2

В3

В4

Процес А

Процес В

с)

В1

В2

В3

В4

Процес А

Процес В

А1

А2

А3

А4

Рис.5.1. Виникнення взаємного блокування при виконанні програм

Залежно від співвідношення швидкостей процесів, вони можуть або взаємно блокувати один одного (дедлок a), або утворювати черги до поділюваних ресурсів, або зовсім незалежно використовувати поділювані ресурси (З).

Програма повинна задовольняти запити таким чином, щоб процеси могли закінчитися, і не виникало тупиків.

Нехай система складається із трьох процесів і десяти пристроїв. Кожному процесу відповідає його максимальна потреба в пристроях, кількість їх, виділена процесу в даний момент і кількість, що він ще має право забажати.

На малюнку а) зображений деякий стан системи:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Ім'я процесу** | **Максимальна потреба** | Виділено | **Залишок** |
| a) | A | 4 | 2 | 2 |
|  | B | 6 | 3 | 3 |
|  | C | 8 | 2 | 6 |

Рис.5.2.а) Стан системи “а”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Ім'я процесу** | **Максимальна потреба** | Виділено | **Залишок** |
| б) | A | 4 | 2 | 2 |
|  | B | 6 | 3 | 3 |
|  | C | 8 | 4 | 4 |

Рис.5.2.б) Стан системи “б”

Допустимо, процес С запросив ще два пристрої і його задовольнили, то якщо один із процесів запросить ту кількість пристроїв, що йому покладено максимально або більше 1 для завершення, буде дедлок. Тому задовольняти запит З - небезпечно.

Для визначення, чи веде задоволення запиту до небезпечного стану, існують спеціальні алгоритми.

Один з них зветься “Алгоритм Банкіра”.

Кожному процесу поставлено у відповідність ціле число i (1) i (N). Процесу i відповідають його максимальна потреба в пристроях МАКС(i), кількість пристроїв, виділених йому в цей момент ВИДІЛПРИСТР(i), що покладається йому залишок - ЗАЛИШОК(i) і ознака МОЖЕ НЕ ЗАКІНЧИТИ(i).

Системи заводять глобальну змінну ЗАГАЛПРИСТР, що позначає загальне число наявних у системі пристроїв.

На початку роботи невідомо, чи може якийсь процес звкінчитися МОЖЕ НЕ ЗАКІНЧИТИ(i) true для всіх i.

Щораз, коли якийсь ЗАЛИШОК може бути виділений із числа незайнятими пристроїв, що залишаються, передбачається, що відповідний процес працює, поки не закінчиться, а потім його пристрої звільняються.

Якщо стан системи стає небезпечним, то вона не задовольняє відповідний запит.

..........

ВІЛПРИСТР:= ЗАГАЛПРИСТР;

**for** i:=1 **step** 1 **until** N **do**

**begin**

ВІЛПРИСТР:= ВІЛПРИСТР - ВИДІЛПРИСТР(i);

МОЖЕ НЕ ЗАКІНЧИТИ(i):=True;

ЗАЛИШОК(i):=МАКС(i)- ВИДІЛПРИСТР(i);

**end**;

ОЗНАКА:=True;

**do** **while** (ОЗНАКА);

ОЗНАКА:=False;

**for** i:=1 **step** 1 **until** N **do**

**begin**

**if** МОЖЕ НЕ ЗАКІНЧИТИ(i) **and** ЗАЛИШОК(i) <= ВІЛПРИСТР;

ВІЛПРИСТР:= ВІЛПРИСТР + ВИДІЛПРИСТР (i);

ОЗНАКА:=True;

**end**; **end**; **end**; **end**;

**if** ВІЛПРИСТР = ЗАГАЛПРИСТР **then** Стан системи безпечний

**else** Стан системи небезпечний

.........

## Проблема розподілу ресурсів та запобігання тупіків

Попердньо при вивченні процесів ми вирішували проблеми їх синхронізації та боротьби зі спеціальним станом, який визначили як тупик або дедлок. Це було пов’язано з проблемою сумісного використання деяких ресурсів, що забезпечують виконання обчислювального процесу.

Взагалі-то поняття ресурсів системи узагальнюють та поділяють на два класи:

* повторного використання (або системні);
* витратні (або одноразового використання).

В літературі їх умовно поділяють як SR (System Resource) та CR (Consumable Resource).

Ці два види ресурсів характерезуються такими особливостями:

1. SR.

* кількість одиниць ресурсу – константа;
* кожна одиниця ресурсу може бути доступною або виділена одному і тільки одному процесу на деякий час;
* процес може звільнити одиницю ресурсу або зробити її доступною у тому випадку, якщо він раніше отримав, тобто ніякий процес не може впливати ні на який ресурс, якщо він йому не належить.

Стосовно проблеми тупиків, то SR ресурси можуть розглядатися як: основна пам’ять, зовнішня пам’ять, периферійні пристрої, процесори, файли даних і тощо.

1. Витратні ресурси CR відрізняються від ресурсів SR у декількох важливих відношеннях:

* кількість доступних одиниць ресурсу типу CR змінюється по мірі використання і звільнюються їх окремі елементи. Кількість одиниць ресурсу є потенційно невичерпною: процес “виробник” збільшує число одиниць ресурсу, вивільняючи одну або більше одиниць ресурсу, які він створив;
* процес “користувач” зменшує число одиниць ресурсу, спочатку вимагаючи, а потім використовуючи одну або більше одиниць. Одиниці ресурсу, що використані у загальному випадку не повертаються ресурсу, а використовуються.

Такі властивості притаманні багатьом сигналам синхронізації, повідомленням та даним, що утворюються апаратурою або програмним забезпеченням.

Для вивчення проблеми тупиків, пов’язаних х вивченням паралельних процесів було розроблено декілька підходів. Однією з них є модель повторно використовуваних ресурсів Холта. Відповідно до цієї моделі система розглядається як множина процесів та набір ресурсів, причому кожен з ресурсів складається з деякої фіксованої кількості одиниць.

Кожен процес може змінити етап системи за допомою запиту, отримання або звілнення запиту з ресурсу.

У граничній формі процеси та ресурси позначаються квадратами та кружками відповідно.

Кожний кружок має деяку кільксть маркерів, що відповідає числу одиниць цього ресурсу. Дуга, що вказує з процесу на ресурс означає запит однієї одиниці цього ресурсу. Дуга, що вказує з ресурсу на процес означає віділення ресурсу процесу.

Оскільки кожна одиниця SR ресурсу може бути віділена одночасно не більше ніж одному процесу, то кількість дуг, що виходять з процесу до різних процесів не може перевищувати загальної кількості одиниць цього ресурсу.

Така модель зветься графом повторно використовуваних процесів.

Наприклад:

R1 П1

П2 R2

Рис.5.3. Граф повторно використовуваних процесів.

Нехай процес П1 записує дві одиниці ресурсу R1 і одну одиницю ресурсу R2. Процесу П2 належать дві одиниці ресурсу R1 та йому потрібна одна одиниця ресурсу R2.

Якщо процес П1 отримає тепер одиницю ресурсу R2 і прийнято правило, згідно з яким для того щоб звільнити хоча б одну одиницю якогось ресурсу, він має отримати всі ресурси, які йому потрібні, це призведе до тупикової ситуації: П1 не зможе продовжити роботу доки П2 не звільнить одиницю ресурсу R2, а П1 не звільнить одиницю R2.

Зрозуміло, що причиною такого дедлоку є неупорядковане використання ресурсів.

Приклад тупіку з ресурсами типу SR ми вже розглянули. Тепер роглянемо приклад тупіку з ресурсами CR.

Нехай існують три процеси П1, П2 та П3, які генерують повідомлення М1, М2 та М3. Ці повідомлення ніщо інше як CR ресурси. Нехай П1 “користувач” (отримує) повідомлення М3, процес П2 отримує повідомлення М1, а П3 - повідомлення М2 від процесу П2, тобто кожен процес є і ”виробником” і ”користувачем” одночасно, утворюючи кільце. Припустимо, що вони спілкуються через поштові скриньки (ПС).

П1

П3

П2

Рис.5.4. Приклад тупіку з ресурсами CR

Якщо цей зв’язок за допомогою повідомлень відтворюється відповідно до порядку, що встановлюється за лістингом:

П1: ……………………………………………..

Відправити повідомлення (П2, М1, ПС2)

Чекати повідомлення (П3, М3, ПС1)

……………………………………………..

П2: ……………………………………………..

Відправити повідомлення (П3, М2, ПС3)

Чекати повідомлення (П1, М1, ПС2)

……………………………………………..

П3: ……………………………………………..

Відправити повідомлення (П1, М3, ПС1)

Чекати повідомлення (П2, М2, ПС3)

……………………………………………..

то все гаразд. Але, якщо переставити місцями ці дві процедури, то ми потрапимо в тупик:

П1: ……………………………………………..

Відправити повідомлення (П3, М3, ПС1)

Чекати повідомлення (П2, М1, ПС2)

……………………………………………..

П2: ……………………………………………..

Відправити повідомлення (П1, М1, ПС2)

Чекати повідомлення (П3, М2, ПС3)

……………………………………………..

П3: ……………………………………………..

Відправити повідомлення (П2, М2, ПС3)

Чекати повідомлення (П1, М3, ПС1)

……………………………………………..

Дійсно, жоден процес не може відправити повідомлення до тих пір, поки сам його не отримає, а такої ситуації ніколи не може бути.

Приклад тупику на ресурсах CR та SR.

Нехай П1 має обмінятися повідомленням з П2 і кожен з них запитує деякий ресурс R, причому П1 потребує три одиниці ресурсу для роботи, а П2 – дві одиниці та тільки на час обробки повідомлення. Усього ж є у розпорядженні тільки 4 одиниці ресурсу R.

Запит ресурсу можна реалізувати через відповідній монітор з процедурами Request (R, N) – запит N одиниць ресурсу R та Release (R, N) звільненя та повернення N одиниць ресурсу R. Обмін повідомленнями будемо виконувати через поштову скриньку (ПС).

Наведемо фрагменти програми:

П1: Request (R, 9);

…………………………………………………

…………………………………………………

Send message (П2, повідомлення, ПС);

Wait Answer (відповідь, ПС);

…………………………………………………

Release (R,3);

П2: Wait message (П1, повідомлення, ПС);

…………………………………………………

…………………………………………………

Request (R,2);

Обробка повідомлення;

…………………………………………………

Release (R,2);

Send answer(відповідь, ПС);

Ці два процеси завжди будуть потрапляти в тупик. П2, якщо він буде включений першим, спочатку чекає на повідомлення від П1, після чого буде заблокований при запиті ресурсу R, частина якого вже була віддана П1. П1, отримавши частину ресурсу R буде також заблоковано, чекаючи відповіді, яку ніколи не отримає, оскільки для цього необхідно отримати ресурс R, який віддано у розпорядження П1.

Тупику можна уникнути лише у тому випадку, коли на час очікування відповіді від П2, П1 буде віддавати хоча б одну одиницю ресурсу R, яким він володіє. У даному випадку причиною тупику є похибка програмування.

Які ж існують формальні моделі для вивчення тупикових ситуацій. Взагалі моделей дуже багато. І якщо вивчати спеціально цей предмет, то нам би знадобилося дуже багато часу. Тому ми зупинимось на найбільш розповсюдженій моделі, так званій сітці Петрі, та вже згаданої моделі Холта.

Сітка Петрі була запропонована у 1962 р. Карлом Петрі для моделювання асинхронних потоків інформації у системах перетворення даних.

Взаємодія подій у паралельних асинхронних дискретних описуюя як ситуації, при яких деяка подія може статися. При цьому глобальні ситуації у системі формуються за допомогою локальних операцій, що названі умовами реалізації подій.

Визначений набір умов дозволяє реалізуватися деякій події (передумові), а реалізація події змінює деякі умови (постумовні події). Тобто події взаємодіють з умовами, а умови з подіями.

Формальний механізм сіток Петрі був використаний Холтом.

Існує декілька формальних представлень сіток Петрі:

* теоретично-множинна;
* графово – біхроматичний (орієнтований) граф;
* матричний (граф).

Ці сітки можуть бути використані з точки зору аналізу системи на можливість виникнення тупикових ситуацій. Цей аналіз використовується за допомою дослідження простору можливих станів сітки. І найчастіше для цього використовують графову модель. Такий підхід базується на побудові рядуцированого до дерева графа можливих маркувань. У такому дереві вершини графа – це стани, а гілки, що помічені відповідними переходами – можливі зміни станів сітки, тобто виконання її переходів.

Якщо взяти довільну вершину у такому дереві (за винятком початкової), то шлях до цієї вершини від кореня дерева (від початкової марки да заданної) буде послідовністю виконання переходів.

Кажуть, що перехід tj для розмітки М “живий”, якщо для усіх розміток Мʹ є σ (М) існує послідовність виконання переходів, яка призводить до маркування Мʹ , при якому перехід tj може спрацювати.

Сітка Петрі називається живою, якщо всі її переходи живі. Живуча розмітка – це така, при якій кожний з її переходів може запускатися безкінечну кількість разів. Коли досягнута така розмітка, при якій жодний перехід не може бути запущений – кажуть, що сітка Петрі завершилась (досягнуте бажане кінцеве маркування) або ж зависла (має місце тупікова ситуація).

У графічному представлені сітки, переходи зображаються вертикальними або горизонтальними “лініями”, а позиції – “кружечками”. Умови – позиції та події-переходи, пов’язані відношеннями безпосередньої залежності (безпосередньої причини спадкового зв’язку), як зображено за допомогою направлених дуг, що ведуть з позицій до переходів і з переходів у позиції. Позиції, з яких ведуть дуги на даний перехід звуться вхідними позиціями, а позиції, на які ведуть дуги з даного переходу – вихідними позиціями.

Виконання умови розміткою відповідної позиції, власне розміщенням числа N або зображенням N маркерів (фішок) у те місце, де N > 0 – ємкість умови.

Переміщення маркерів по сітці виконується за допомогою виконання її переходів. Виконяння збурненого переходу є локальним числом та в цілому призводить до зміни маркування сітки, тобто до зміни її стану.

Якщо для сітки M0 задано початкове маркування, при якому хоча б один перехід збуджено, то у ній починається рух маркерів.

Число маркерів, які перехід tj виймає зі своїх вихідних позицій може не дорівнюватись числу маркерів, які цей перехід переміщує у свої вихідні позиції, так як зовсім не обов’язково, щоб кількість вхідних дуг переходу дорівнювали кількості його вихідних дуг.

P1 P9

t1 t5

P8

P2 P10

t2 t2

…. P6

P3 P11

t3 t7

P4 P12

t4 P7 t8

P5 P13

Рис.5.5. Сітка відповідної тупікової ситуації

Сітка відповідної тупікової ситуації, розглянутої нами при кінцевій взаємодії процесів.

Початкове маркування (1, 0, 0, 0, 0, 4, 001, 0, 0, 0, 0). Тут позиція Р2 означає, що Р1 (Р - велике) отримав три одиниці ресурсу R. Дуга, що поєднує позицію Р6 (число в ній відповідає кількості доступних одиниць ресурсу R) має вагу 3, при спрацюванні пиреходу t1 процес Р1 отримає три одиниці ресурсу.Перехід t2 відповідає відправленню повідомлення Р2. Перехід t5 – прийом. Поява маркеру у позиції Р7 означає, що Р2 обробив та послав відповідь Р2. Виконання переходу t4 – це повернення 3 одиниць ресурсу, якими володів Р1.

Ця сітка (не) буде живою, бо в ній будуть мертвими переходи t2, t3, t6, t7, t8.

a b

2 2

c d

Рис.5.6. Сітка Петрі

Нехай це сітка Петрі. Розмітка М сітки – це функція, що ставить у відповідність міткам позицій невід’ємні цілі числа. Розмітка приписує кожній позиції деяку кількість міток у відповідності до функції розмітки.

Нехай Р –множина позицій в N, а n (P) – число позицій в Р. Кожна позиція в N однозначно пов’язана з набором номерів {1, 2, 3, … ,n(P)} .

Розмітку М можна представити як вектор з n(P) елементів, у якому i-ий елемент визначає кількість поміток в і-ій позицій.

Розмітка змінюється при умові запуску переходу. Перехід можна запустити, якщо для кожної позиції х, з якої стрілка вказує на t існує хоча б одна помітка (якщо стрілка одинична).

На малюнку переходи a та b – запускаємо М = [2,0,0]. Якщо зробити послідовність запусків: а М’ = [1,1,0]. Тепер a, b, c, які можна запустити. Якщо запустити С, отримаємо Мʹʹ = [2,0,0]. Можна, наприклад, запустити a та b – одночасно, тоді буде тупик. Мʹ = [1,0,1]. Далі “живих” переходів нема.

# Мультипрограмування. Розподіл часу процесора

Час процесора завжди був найважливішим з ресурсів системи, що підлягають розподілу.

**Мультипрограмування** або багатозадачність – спосіб організації обчислювального процесу, при якому на одному процесорі поперемінно виконуються відразу кілька програм.

Найбільш характерними критеріями ефективності обчислювальних систем є:

* Пропускна здатність - кількість завдань, виконуваних обчислювальною системою в одиницю часу.
* Зручності роботи користувачів, полягає в тому, що вони мають можливість інтерактивно працювати одночасно з декількома додатками на одній машині.
* Реактивність системи - здатність системи витримувати заздалегідь задані інтервали часу між запуском програми й одержанням результату.

Залежно від цього ОС діляться на:

* Системи пакетної обробки;
* Системи поділу часу;
* Системи реального часу.

## Системи пакетної обробки

Призначалися для рішення завдань обчислювального характеру, не потребуючих швидкого одержання результатів. Головною метою й критерієм ефективності систем пакетної обробки є максимальна пропускна здатність.

Для цієї мети формується **пакет** завдань, кожне завдання містить вимоги до системних ресурсів, із цих завдань формується мультипрограмна суміш, тобто безліч завдань, що виконуються одночасно.

Для пакетних ОС характерне сполучення операцій введення - виведення й обчислень. Таке сполучення може досягатися різними способами :

1. **Спеціалізований процесор введення - виведення**.

Іноді такі процесори називають каналами. Канал має систему команд, що відрізняється від системи команд центрального процесора. Ці команди спеціально орієнтовані для керування зовнішніми пристроями :

* + встановити магнітну головку;
  + надрукувати рядок тощо.

Канальні програми можуть зберігатися в тій же оперативній пам'яті, що й програми центрального процесора. У системі команд центрального процесора передбачається спеціальна інструкція, за допомогою якої каналу передаються параметри й вказівки на те, яку програму введення - виведення він повинен виконати.

Обчислення

Центральний

процесор

Команда запуску

Канальної

програми

Введення/виведення

Сигнал завершення

операції введення/виведення

Канальна програма

Канал

Рис.6.1. Канальна програма

**Зовнішні пристрої керуються контролерами**.

Кожний (або група) зовнішній пристрій має свій власний контролер, який опрацьовує команди, що надходять від центрального процесора. Контролер і ЦП працюють асинхронно. Контролер повідомляє ЦП про те, що він готовий прийняти наступну команду сигналом переривання або ЦП довідається про це, періодично опитуючи стан контролера.

Сигнали готовності пристрою

Обчислення

Центральний

процесор

Операція введення/виведення

Контролер

введення/виведення

Рис.6.2. Операція введення/виведення

Максимальний ефект при пакетній обробці досягається при найбільш повному перекритті обчислень і введення - виведення.

У випадку одного завдання прискорення залежить від його характеру. При перевазі обчислень або введення - виведення прискорення практично відсутнє.

## Системи поділу часу

Системи поділу часу усувають основний недолік пакетної обробки - ізоляцію користувачів-програмістів від процесу його завдань. Кожному користувачеві виділяється свій термінал, з якого можна вести свій діалог із програмою. У системах цього типу кожному завданню виділяється квант процесорного часу, жодне завдання не займає процесор надовго. Створюється ілюзія, що процесор належить тільки завданню (або програмісту).

## Системи реального часу

Основний критерій – здатність системи витримати заздалегідь задані інтервали часу між запуском програми й одержанням результату. Цей час називається **часом реакції системи,** а відповідна властивість системи – реактивністю. Вимоги вчасної реакції визначаються зовнішніми факторами (наприклад, специфікою системи керування).

У системах реального часу мультипрограмна суміш являє собою фіксований набір заздалегідь розроблених програм, а вибір програми на виконання здійснюється по перериваннях (наприклад, виходячи з поточного стану об'єкта керування) або в стані з розкладом робіт.

## Мультипроцесорна обробка

Мультипроцесорна обробка - спосіб організації обчислювального процесу в системах з декількома процесорами, при якому кілька завдань можуть одночасно виконуватися на різних процесорах системи.

У цей час стало звичайним явищем включення декількох процесорів в архітектуру персонального комп'ютера.

Функції підтримки мультипроцесорної обробки даних є в багатьох ОС, у тому числі й такий як Windows NT.

Мультипроцесорні системи характеризують як **симетричні** або як **несиметричні***,* залежно від того , до якого аспекту обчислювальної системи це відноситься:

* до архітектури;
* до способу організації обчислювального процесу.

**Симетрична архітектура** допускає однорідність всіх процесів й однаковість включення процесорів у схему мультипроцесорної системи. Традиційні симетричні мультипроцесорні конфігурації розділяють одну велику пам'ять між всіма процесорами.

**Масштабованість** або можливість нарощування числа процесорів у симетричних системах обмежена внаслідок того, що всі вони користуються однієї й тією же оперативною пам'яттю й розташовуються в одному корпусі. Це **масштабування по вертикалі**.

У симетричних архітектурах всі процеси користуються однієї й тією ж схемою відображення пам'яті. Вони можуть швидко обмінюватися даними. Забезпечується висока продуктивність для завдань, які активно між собою взаємодіють (наприклад, при роботі з базами даних).

В **асиметричній архітектурі** процесори можуть відрізнятися як своїми характеристиками, так і функціональною роллю, яка причається їм у системі.

Масштабування в асиметричній архітектурі реалізується інакше, ніж у симетричній. Система може складатися з декількох пристроїв, кожен з яких містить один або кілька процесорів. Це **масштабування по горизонталі**. Кожен такий пристрій називається **кластером**, а вся система звичайно називається **кластерною***.* Спосіб організації обчислювального процесу в мультипроцесорній системі визначається ОС.

**Асиметричне мультипроцесування** є найбільш простим способом організації. Цей спосіб іноді називають «ведучий - відомий».

На «провідному» процесорі працює ОС, що керує всіма іншими «відомими» процесорами. Він бере на себе функції розподілу завдань і ресурсів, а «відомі» працюють тільки як обробні пристрої та ніякі дії по організації роботи обчислювальної системи не виконують. Така ОС не набагато складніша ОС однопроцесорної системи.

Асиметрична організація обчислювального процесу може бути реалізована як для симетричної мультипроцесорної архітектури, так і для несиметричної.

**Симетричне мультипроцесування** як спосіб організації обчислювального процесу може бути реалізоване тільки в системах із симетричною мультипроцесорною архітектурою.

Симетричне **мультипроцесування** реалізується загально для всіх процесорів ОС. Всі процесори рівноправно беруть участь у керуванні обчислювальним процесом й у виконанні прикладних завдань. Наприклад, сигнал переривання від принтера, що роздруковує дані процесу, виконуваного на деякому процесорі, може бути оброблений зовсім іншим процесором. Різні процесори можуть у якийсь момент одночасно обслуговувати як різні, так й однакові модулі ОС. Для цього модулі ОС повинні мати властивість реентерабельності (повторної входимості). ОС повністю децентралізована. Як тільки процесор завершує виконання чергового завдання, він передає керування **планувальникові**, який обирає із загальної системної черги для всіх процесорів завдання, що буде виконуватися на даному процесорі наступним.

У випадку відмови одного із процесорів симетричні системи порівняно легко реконструюються, що є перевагою перед асиметричними системами.

# Процеси і потоки

Дотепер ми розглядали процес як деяку неподільну роботу, виконувану обчислювальною системою.

У ряді ОС визначені два типи роботи. Більша одиниця – процес, що вимагає для своєї реалізації дещо дрібніших робіт, і ця дрібніша одиниця називається **потоком**.

При реалізації потоків з'являється можливість організації паралельних обчислень у рамках процесу.

Справа в тому, що програми, виконувані в рамках одного процесу, можуть мати внутрішній паралелізм, що у принципі може прискорити час виконання процесу.

Із цього слідує, що в ОС поряд з механізмами керування процесами потрібний інший механізм розпаралелювання обчислень, що враховував би тісні зв'язки між окремими галузями обчислень того самого додатка.

Для цих цілей у ряді сучасних ОС використовується механізм **многопоточної обробки**. При цьому вводиться нова одиниця роботи – **потік виконання**, а поняття «процес» до деякої міри змінює зміст.

Поняттю «потік» відповідає послідовний перехід процесора від однієї команди програми до іншої. ОС розподіляє процесорний час між потоками, а процесу ОС призначає адресний простір і набір ресурсів, які спільно використовуються всіма його потоками.

При керуванні процесами ОС використовує два основних типи інформаційних структур:

* дескриптор процесу;
* контекст процесу.

**Дескриптор процесу** містить таку інформацію про процес, яка необхідна ядру ОС протягом усього життєвого циклу процесу незалежно від того, перебуває він в активному або пасивному стані, перебуває образ процесу в оперативній пам'яті або вивантажений на диск. **Образ** – сукупність кодів команд і даних.

Дескриптори процесів об'єднані в список, що утворює таблицю процесів. Пам'ять виділяється динамічно в області ядра. На підставі інформації, що міститься в таблиці процесів, ОС здійснює планування й синхронізацію процесів.

У дескрипторі прямо або побічно (через покажчики) утримується інформація про стан процесу, про розташування образу процесу, про ідентифікатор користувача, що створив процес, про родинні процеси, про події, появи яких очікує процес та ін.

**Контекст процесу** містить інформацію, необхідну для поновлення виконання процесу з перерваного місця: вміст регістрів процесу, коди помилок виконуваних процесором системних викликів, інформація про всі відкриті, даним процесом, файли і незавершені операції введення - виведення й інші дани, що характеризують стан обчислювальної системи в момент переривання.

**Контекст**, так само як і **дескриптор**, доступний тільки програмам ядра, тобто **перебуває у віртуальному адресному просторі ОС**.

Протягом існування процесу виконання його потоків може бути багаторазово перерване й продовжено (далі будемо вважати, що все сказане про потоки, буде стосуватися процесів у цілому, якщо ОС не підтримує потоки).

**Перехід** від виконання **одного потоку** до іншого здійснюється в результаті **планування й диспетчеризації**.

Робота з визначення того, у який момент часу необхідно перервати потік й якому потоку надати можливість виконуватися, називається **плануванням**.

**При плануванні** можуть прийматися до уваги **пріоритет потоків**, **час їхнього очікування** в черзі, накопичений час виконання, інтенсивність звертання до введення - виведення та інші фактори.

**ОС планує виконання потоків незалежно від того, чи належать вони одному або різним процесам**. Так, наприклад, після виконання потоку деякого процесу ОС може вибрати для виконання інший потік того ж процесу або ж призначити до виконання потік іншого процесу.

Планування потоків містить у собі рішення двох завдань:

* визначення моменту часу для зміни поточного активного потоку;
* вибір для виконання потоку із черги готових потоків.

Існує безліч різних алгоритмів планування потоків, які виконують вищезгадані завдання.

Саме особливості реалізації планування потоків найбільшою мірою визначають специфіку ОС, зокрема, чи є вона системою пакетної обробки, системою поділу часу або системою реального часу.

Планування може бути **динамічним** або **статичним**.

При **динамічному плануванні** рішення приймаються під час роботи системи на основі аналізу поточної ситуації.

ОС працює в умовах невизначеності - потік і процеси з'являються у випадкові моменти часу й також непередбачене завершується. Динамічні планувальники можуть гнучко пристосовуватися до ситуації, що змінюється. Тут ОС для пошуку оптимальних рішень повинна прикладати значні зусилля.

**Планувальник** називається **статичним**, якщо він ухвалює рішення щодо плануванні не під час роботи системи, а заздалегідь.

Результатом роботи статичного планувальника є таблиця, названа **розкладом**, у якій вказується, якому потоку (процесу), коли й на який час повинен бути наданий процесор.

**Диспетчеризація** полягає в реалізації знайденого в результаті планування (динамічного або статичного) рішення, тобто в перемиканні процесора з одного потоку на іншій. Перш, ніж перервати виконання потоку, ОС запам'ятовує його контекст для того, щоб згодом використати цю інформацію для наступного відновлення виконання даного потоку.

Контекст відбиває:

* стан апаратур комп'ютера в момент переривання потоку: значення лічильника команд, вміст регістрів загального призначення, режим роботи процесора, прапори, маски й інші параметри;
* параметри операційного середовища (посилання на відкриті файли, дані про незавершені операції введення - виведення, коди помилок, виконуваних даним потоком системних викликів тощо).

Диспетчеризація зводиться до наступного :

* збереження контексту поточного потоку, що потрібно змінити;
* завантаження контексту нового потоку, обраного в результаті планування;
* запуск нового потоку на виконання.

У мультипрограмній системі потік може перебувати в одному із трьох станів:

* виконання - виконується процесором:
* очікування - чекає здійснення деякої події;
* готовність - має всі необхідні для виконання ресурси, готовий виконуватися, але процесор зайнятий виконанням іншого потоку.

Помітимо, що стани виконання й очікування можуть бути віднесені й до завдань, що виконуються в однопрограмному режимі, а стан **готовності** характерний тільки для режиму мультипрограмування.

Граф станів потоку в багатозадачному середовищі можна представити як на малюнку:

Потік очікує завершення

введення/виведення (чи іншої події)

Потік завершено або помилка

введення/виведення завершено(подія здійснилась)

Потік витиснутий

Обраний на виконання потік

Тільки но створений потік

Рис.7.1. Граф станів потоку в багатозадачному середовищі

**«Витиснення потоку»** означає припинення його виконання процесором, наприклад, внаслідок вичерпання відведеного для виконання кванта часу.

У стані виконання в однопроцесорній системі може перебувати не більше одного потоку, а в кожному зі станів очікування й готовності – кілька потоків. Ці потоки утворюють черги **очікуючих** і **готових** потоків.

# **Алгоритми планування**

## Витісняючі і не витісняючі алгоритми планування

Всю безліч алгоритмів планування можна розділити на два класи: ті, що витісняють і ті, що не витісняють.

**Невитісняючі** засновані на тому, що активному потоку дозволено виконуватися доти, доки він сам не вирішить віддати керування ОС.

**Витісняючі** – такі, у яких рішення про перемикання процесора з виконання одного потоку на інший приймається ОС.

При мультипрограмуванні, що не витісняє, механізм планування розподілений між ОС і прикладними програмами. Прикладна програма, одержавши керування від ОС, сама визначає момент завершення чергового циклу свого виконання й тільки потім передає керування ОС за допомогою якого-небудь системного виклику.

Тому розроблювачі програм (програмісти) для ОС із **невитісняючою** багатозадачністю змушені брати на себе частину функцій планувальника й створювати програми так, щоб вони виконували свої завдання невеликими частинами. Це може бути як недоліком, так і перевагою, якщо, наприклад, наперед відомий набір постійно розв'язуваних завдань.

Майже всі сучасні ОС, такі як UNIX, Windows NT/2000, OS-2, Windows 95/98 реалізують витісняючі алгоритми планування потоків.

## Алгоритми планування, засновані на квантуванні

В основі багатьох витісняючих алгоритмів планування лежить концепція **квантування**. Відповідно до неї, кожному потоку по черзі для роботи виділяється обмежений безперервний відрізок часу – **квант**.

Зміна активного потоку відбувається, якщо:

* потік завершився й покинув систему;
* відбулася помилка;
* потік перейшов у стан очікування;
* вичерпано квант процесорного часу.

Потік, що вичерпав свій квант, переводиться в стан готовності й очікує в черзі. Граф станів потоку представлений на малюнку:

Потік завершено або помилка

Введення - виведення завершено

Потік вичерпав

квант

Потоку наданий квант

Потік ініціював

введення - виведення

Рис.8.1. Граф станів потоку

Кванти для потоків можуть бути однаковими або різними.

Черга може бути проста або із пріоритетами. Наприклад, якщо потоки не повністю використовують кванти часу через операції введення - виведення, з них можна утворити пріоритетну чергу як відповідну компенсацію за не повністю використані кванти.

Відповідний граф станів потоку можна представити як:

Потік завершено або помилка

Квант вичерпаний

Потрібно введення - виведення

Наданo квант

Введення-виведення завершено

Знову створені потоки

Рис.8.2. Граф станів потоку

## Алгоритми планування, засновані на пріоритетах

Пріоритет – це число, що характеризує ступінь привілейованості потоку при використанні ресурсів. У більшості ОС пріоритет потоку пов'язаний із пріоритетом процесу, у рамках якого виконується даний потік. Значення пріоритетів включається в описувач процесу. ОС може змінювати пріоритети потоку залежно від ситуації. В останньому випадку пріоритети називаються **динамічними**, на відміну від незмінних, які називаються **статичними** (або фіксованими).

В ОС Windows NT визначено 32 рівня пріоритетів і два класи потоків - потоки реального часу й потоки зі змінними пріоритетами.

Діапазон від 1 до 15 відведений для потоків зі змінними пріоритетами, а від 16 до 32 - для більш критичних вчасно потоків реального часу.

## Змішані алгоритми планування

У ряді ОС алгоритми планування побудовані з використанням як концепції квантування, так і пріоритетів. Наприклад, в основі планування лежить квантування, але величина кванта й порядок вибору потоків із черги готових визначається пріоритетами потоків.

Так зроблено в Windows NT, у ній квантування сполучається з динамічними абсолютними пріоритетами. На виконання вибирається потік з найвищим пріоритетом. Йому виділяється квант часу. Якщо під час виконання в черзі готових з'являється потік з більш високим пріоритетом, то він витісняє виконуваний потік. Витиснутий потік повертається в чергу готових, причому він стає попереду всіх інших потоків, що мають такий же пріоритет.

## Планування в системах реального часу

У системах реального часу головним критерієм є забезпечення часових характеристик обчислювального процесу.

Будь-яка система реального часу повинна реагувати на сигнали керованого об'єкта протягом заданих тимчасових обмежень. Необхідність ретельного планування робіт полегшується тим, що в системах реального часу весь набір виконуваних завдань відомий заздалегідь.

Крім того, у системі є інформація про час виконання завдань, моменти активації, граничні припустимі строки очікування відповіді тощо. Ці дані можуть бути використані планувальником для створення статичного розкладу або для побудови адекватного алгоритму динамічного планування.

Якщо наслідки невиконання тимчасових обмежень системою катастрофічні (наприклад, прокатний стан), система називається **жорсткою**. Якщо невиконання обмежень не настільки серйозне (наприклад, система продажу авіаквитків), система називається **м'якою**.

***Моменти перепланування***

Для реалізації алгоритму планування ОС повинна одержувати керування щоразу, коли в системі відбувається подія, що вимагає перерозподілу процесорного часу.

До таких подій відносять наступні:

* переривання від таймера, яке сигналізує, що час, відведений активному завданню, закінчився;
* активне завдання виконало системний виклик, пов'язаний із запитом на введення - виведення або на доступ до ресурсу, що у даний момент зайнятий (наприклад, файл даних);
* активне завдання виконало системний виклик, пов'язаний зі звільненням ресурсу. Планувальник перевіряє, чи не очікує цей ресурс яке-небудь завдання. Якщо так, то завдання переводиться зі стану очікування в стан готовності й перевіряється, чи має воно найвищий пріоритет. Якщо немає - можливе перепланування;
* зовнішнє апаратне переривання. Воно сигналізує про переклад відповідного поточного завдання в чергу готовності й виконується планувальник;
* внутрішнє переривання повідомляє про помилку в поточному завданні. Планувальник знімає завдання й виконує перепланування.

При виникненні кожної із цих подій планувальник виконує перегляд черг і вирішує питання про те, яке завдання буде виконуватися наступним.

Нагадаємо, що стратегія планування визначає, які процеси ми плануємо до виконання, для того, щоб досягти поставленої мети.

Відомо дасить багато стратегій визначення процесу, якому треба віддати процесор. Найбільш поширеними є такі стратегії:

* по можливості завершити процеси у тому порядку, в якому вони були розпочаті;
* віддати перевагу найбільш коротким, за часом виконання, процесам;
* надати усім користувачам (тобто процесам) ознакові послуги щодо часу читання у відповідних чергах.

Усі дисципліни диспетчеризації, не зважаючи на їх велику різноматність можна поділити на два великих класи: безпріоритетні та пріоритетні.

При безпріоритетному обслуговуванні вибір процесу на виконання визначається у деякому, наперед встановленому, порядку (скажімо, визначеному планувальником) без урахування їх значимості та часових обмежень.

При реалізації пріоритетних дисциплін окремим процесам надається переважне право на викачання.

Перелік та класифікацію дисциплін обслуговування можно навести як на малюнку:

Дисципліни диспетчерезації

Приорітетні

Безприорітетні

Лінійні

У порядку черги

Циклічні

З циклічним пріоритетом

З фіксованим пріоритетом

Випадковий вибір процесу

Багатопріоритетний циклічний

Цикліний алгоритм

З абсолютним пріоритетом

З відносним пріоритетом

Пріоритет залежить від часу обслуговування

Пріоритет у залежності від часу очікування

Адаптивне обслуговування

Рис.8.3. Перелік та класифікація дисциплін обслуговування

Про пріоритети треба запам’ятати таке:

* пріоритет процесу може бути постійним;
* пріоритет процесу може бути змінений при його виконанні.

Слід завжди пам’ятати, що використання динамічних пріоритетів потребує додаткових витрат на обчислення пріоритетів у конкретний час, тому у більшості випадків реального часу використовуються методи диспетчеризації на основі фіксованих пріоритетів.

Розглянемо деякі найбільш типові дисципліни диспетчеризації.

Найпростішою є дисципліна FCFS (First Come – First S), тобто обслуговування у порядку черги. Ті процеси, що були заблоковані у процесі виконання (скажімо, чекають завершення введення - виведення) стають у чергу готовності перед тими процесами, які ще не починали виконуватись. Тобто створюється дві черги: одна з нових процесів, друга – з процесів, що вже виконувались, але перейшли в режим очікування. Такий перехід дозволяє реалізувати тезу про завершення процесів у порядку їх виникнення.

Ця дисципліна обслуговування не потребує втручання в хід виконання процесів, перерозподіл процесорного часу не відбувається.

Графічно цю дисципліну можна відобразити як на малюнку:

Процесор

Черга процесів готових до виконання

Черга нових процесів

Рис.8.4. Дисципліна FCFS

Перевагами цієї дисципліни є простота реалізації та малі витрати системних ресурсів та формування черги процесів.

Однак, недолік у тому, що при збільшенні завантаження обчислювальної системи зростає середній час очікування обслуговування, причому невеликі за часом виконання процеси мають очікувати стільки ж, як і великі завдання.

Цей недолік нівелюють дисципліни SJN та SRT.

Дисципліна SJN (Shortest Job next) означає: наступним буде виконане найкоротше завдання. Ясно, що для цього диспетчер має порівнювати заявлений на виконання час та час фактичного виконання, та у випадку перевищення по даному параметру процес має бути поставлено у кінець черги, або може використовуватись інша система штрафів.

Дисципліна SJN – визначає, що є тільки одна черга завдань, готових до виконання. Завдання, які були тимчасово заблоковані знову стають у кінець черги поряд з повними процесами.

Це теж призводить до того, що процеси, яким потрібно мало часу для виконяння в кінці-кінців також вимушені очікувати процесор поряд з довшими процесами.

Для викорінення цього недоліку була запропонована дисципліна SRT (Shortest Remaining Time), наступне завдання потребує для завешення найменше часу.

Дисципліна обслуговування RR передбачає, що кожний процес отримує процесорний час частками (квантами).

Після закінчення кванту процес знімається з процесору. Процесор передається іншому процесу. Процес, що був знятий з виконання стає у чергу процесів, готових до виконання. Ця дисципліна проілюстрована на малюнку:

Процесор

Нові задачі

Рис.8.5. Дисципліна обслуговування RR

Її ще називають карусельною дисципліною диспетчеризації. Для оптимальної роботи системи необхідно обрати закон, за яким кванти часу надаються процесам.

Величина кванту визначається як компроміс між бажаним часом реакції системи на запит користувачів та накладеними витратами на часту заміну контексту процесів.

Нагадаємо, що при невитісняючому типуванні диспетчеризація виконується без перерозподілу часу процесора. Дисципліни FCFS, SJN, SRT відносяться до невитісняючих.

Для порівняння різних алгоритмів диспетчеризації використовуються наступні критерії:

* завантаженість процесора. Вона змінюється від 2 до 3 відсотків у персональному комп’ютері, і до 90 – 100 відсотків для сильно завантажених серверів.
* Пропускна здатність. Вона вимірюється числом процесів, які можуть виконуватись за одиницю часу.
* Час обороту (turnaround time). Для деяких процесів важливим критерієм є повний час виконання, тобто інтервал від моменту появи процесу у черзі до моменту його завершення.
* Час очікування. Під цим розуміють сумарний час знаходження процесу у черзі готових процесів.
* Час відгуку (response time). Цей критерій важливий для інтерактивних програм. Під ним мається на увазі час відгуку або час, який проходить від моменту постановки процесу у чергу до моменту першого звернення до терміналу.

Треба відмітити, що правильне визначення планування процесів сильно впливає на продуктивність усієї системи.

Головними причинами, що призводять до зменшення продуктивності системи є:

* Накладні витрати на переключення процесора (сюди відносять переключення контекстів, переміщення сторінок віртуальної пам’яті, заміни даних у КЕШі тощо.)
* Переключення на інший процес, якщо попередній виконує критичну секцію. У цьому випадку втрати найбільші.

## Використання динамічних пріоритетів

При роботі у режимі реального часу, що виконують задачі контролю та управління, може виникнути така ситуація, коли одна або декілька задач не можуть бути виконані протягом досить великого проміжку часу. Втрати, пов’язані з невиконанням таких задач можуть бути досить значними. При цьому доцільно тимчасово змінити пріоритети „аварійних” задач. Після їх виконання відповідні пріоритети можна поновити. Тому, як правило, у ОС реального часу є засоби зміни пріоритету програми.

Одним з таких засобів є наступний:

При обчисленні приймають участь два поля дескриптора процесу. Перше з них визначається користувачем явно або формується по замовчуванню за допомогою системи програмування. Друге поле формується планувальником і зветься системною складовою або поточним пріоритетом. Тобто процес має два атрибути пріоритету: поточний пріоритет, за яким виконується планування та заказний відносний пріоритет. Наприклад, більш високому значенню поточного пріоритету може відповідати більш низький фактичний пріоритет планувальника.

Розподіл між пріоритетами режиму ядра та задачі також залежить від версії ОС. Пріоритети процесів, що виконуються у режимі задачі мають менший пріоритет ніж у режимі ядра. Наприклад, пріоритети режиму задачі змінюються у діапазоні 0-65, для режиму ядра – 66-95, а процеси, пріоритети яких лежать у діапазоні 96-127 – процеси з фіксованим пріоритетом. Вони не змінюються ОС та визначені для підтримки реального часу.

Як правило, для процесу, що чекає недоступного у даний час ресурсу, система визначає значення пріоритету, який вибирається ядром із діапазону системних пріоритетів, пов’язаних із цим випадком, що призвів до такого стану. Коли процес пробуджується, ядро встановлює значення поточного пріоритету, що дорівнює пріоритету сну. Оскільки пріоритет такого процесу знаходиться у системному діапазоні та вище, ніж пріоритет задачі, вірогідність надання процесу обчислювальних ресурсів значно збільшується.

Скажімо, такий підхід дозволяє швидко завершити системний виклик, виконання якого можуть блокувати деякі системні ресурси.

Після завершення системного виклику перед поверненням у режим задачі, ядро поновить пріоритет режиму задачі, що зберігся перед виконанням системного виклику. Це може призвести до зниження пріоритету, що у свою чергу може викликати переключення контексту.

Наприклад, операційна система OS/2 самостійно змінює пріоритет виконання програми незалежно від рівня, який був встановлений прикладною задачею. Цей механізм зветься механізмом підвищення пріоритету.

ОС змінює пріоритети задачі у трьох випадках:

* Збільшується пріоритет активної задачі. Пріоритет автоматично збільшується, коли вона стає активною. Це зменшує час реакції активного додатку (приложения (рос.)) на дії користувача у порівнянні із фоновими програмами
* Збільшується пріоритет введення - виведення. По завершенні операцій введення - виведення процес отримує найбільший рівень пріоритету його класу. Таким чином забезпечується завершення усіх незакінчених операцій введення - виведення.
* Збільшення пріоритету „забутих” задач. Якщо задача (або процес) не отримує процесор досить довго (цей проміжок часу задається наприклад у OS/2 у файлі CONFIG.SYS), диспетчер задач OS/2 тимчасово присвоює їй рівень пріоритету, що не перевищує критичний. У результаті, час переключення на таку „забуту” програму пришвидшується. Після використання одного кванту часу пріоритет знову зменшується до попереднього.

Такий механізм у сильно завантажених системах дозволяє програмам з маленьким пріоритетом теж іноді працювати. При відсутності такого механізму, такі програми взагалі б не працювали.

# Мультипрограмування на основі переривань

## Переривання

Переривання відбувається в довільній точці потоку команд програми, яку програміст не може спрогнозувати.

Переривання мають деяку подібність із процедурою в тому, що в обох випадках виконується деяка підпрограма, що обробляє спеціальну ситуацію, а потім триває виконання основної гілки програми.

Залежно від джерела, переривання діляться на три більших класи:

* зовнішні;
* внутрішні;
* програмні;

**Зовнішні**переривання виникають у результаті дій користувача або в результаті надходження сигналів від апаратних пристроїв. Даний клас переривань є **асинхронним**стосовно потоку інструкцій перериває програми.

Апаратура процесора працює так, що асинхронні переривання виникають між виконанням двох сусідніх інструкцій, при цьому система після обробки переривання продовжує виконання процесу, уже починаючи з наступної інструкції.

**Внутрішні**переривання відбуваються **синхронно**щодовиконання програми з появою аварійної ситуації в ході виконання деякої інструкції програми. Прикладами є ділення на нуль, помилки захисту пам'яті, звертання до неіснуючої адреси. Переривання виникають усередині виконання команди.

**Програмні**переривання відрізняються від попередніх двох класів тим, що вони по своїй суті не є «щирими» перериваннями. Програмне переривання виникає при виконанні особливої команди процесора, виконання якої імітує переривання, тобто перехід на нову послідовність інструкцій.

Перериванням приписується пріоритет, за допомогою якого вони ранжуються по ступені важливості й терміновості. Про переривання, які мають однакове значення пріоритету, говорять, що вони відносяться до **одного рівня пріоритету**переривань.

Процедури, викливані перериваннями, звичайно називають **оброблювачами переривань,**або **процедурами обслуговування переривань***.*

Апаратні переривання обробляються драйверами відповідних зовнішніх пристроїв, внутрішні переривання - спеціальними модулями ядра, а програмні переривання - процедурами ОС, що обслуговують системні виклики.

Крім цих модулів в ОС може перебувати так званий диспетчер переривань, що координує роботу окремих оброблювачів переривань.

Узагальнено послідовність дій апаратних і програмних засобів по обробці переривання можна описати в такий спосіб:

1.При виникненні сигналу (для апаратних переривань) або умови (для внутрішніх переривань) переривання, відбувається первинне апаратне розпізнавання типу переривання. Якщо переривання даного типу в даний момент заборонені (пріоритетною схемою або механізмом маскування), то процесор продовжує підтримувати природний хід виконання команд.

У іншому випадку залежно від інформації, що надійшла в процесор, відбувається автоматичний виклик процедури обробки переривання, адреса якої перебуває в спеціальній таблиці ОС, розташованої або в регістрах процесора, або в певному місці оперативної пам'яті.

1. Автоматично зберігається деяка частина контексту перерваного потоку, що дозволить ядру відновити виконання потоку процесу після обробки переривання. У цю підмножину включаються значення лічильника команд, слова стану машини, що зберігає ознаки основних режимів роботи процесора (приклад слова - регістр EFLAGS в Intel Pentium), а також декількох регістрів загального призначення, які потрібні програмі обробки переривання. Може бути збережений і повний контекст процесу, якщо ОС обслуговує дане переривання зі зміною процесу.
2. Одночасно із завантаженням адреси процедури обробки переривань у лічильник команд, може автоматично виконуватися завантаження нового значення слова стану машини, що визначає режими роботи процесора при обробці переривання, у тому числі роботу в привілейованому режимі.
3. Тимчасово забороняються переривання даного типу, щоб не утворилася черга вкладених один в одного потоків однієї й тієї ж процедури. Деталі виконання цієї операції залежать від особливостей апаратної платформи, наприклад може використовуватися механізм маскування переривань..
4. Після того як переривання оброблене ядром ОС, перерваний контекст відновлюється, і робота потоку відновлюється з перерваного місця.

Частина контексту відновлюється апаратно по команді повернення з переривань (наприклад, адреса наступної команди й слово стану машини), а частина - програмним способом, за допомогою явних команд добування даних зі стека. При поверненні з переривання блокування повторних переривань даного типу знімається.

## Програмні переривання

Програмне переривання реалізує один зі способів переходу на підпрограму за допомогою спеціальної інструкції процесора, такий як INT у процесорах Intel Pentium, trap у процесорах Motorola тощо.

При виконанні команди програмного переривання процесор відпрацьовує ту ж послідовність дій, що й при виникненні зовнішнього або внутрішнього переривання, але відбувається це тільки в передбачуваній точці програми *—* там, де програміст помістив дану команду.

Майже усі сучасні процесори мають у системі команд інструкції програмних переривань.

Програмні переривання часто використовуються для виконання обмеженої кількості викликів функцій ядра операційної системи, тобто системних викликів.

## Диспетчеризація й пріоритезація переривань в ОС

Переривання виконують корисну для обчислювальної системи функцію - дозволяють реагувати на асинхронні, стосовно обчислювального процесу, події.

Переривання створюють додаткові труднощі для ОС в організації обчислювального процесу.

Для впорядкування роботи оброблювачів переривань в ОС застосовується той же механізм, що й для впорядкування роботи користувальницьких процесів – механізм пріоритетних черг. Всі джерела переривань діляться на кілька класів, причому кожному класу присвоюється пріоритет. В ОС виділяється програмний модуль, що займається диспетчеризацією оброблювачів переривань. Цей модуль у різних ОС називається по-різному, ми на самому початку визначили його, як *диспетчер переривань.*

При виникненні переривання диспетчер переривань викликається першим. Він забороняє на якийсь час всі переривання, і з'ясовує причину переривання. Потім рівняється пріоритет джерела переривання з поточним пріоритетом потоку команд, виконуваних процесором. Якщо пріоритет нового запиту вищий поточного, то виконання поточного потоку припиняється й виконується обробка переривання. У іншому випадку запит ставиться в чергу.

## Процедури обробки переривань і поточний процес

Важливою особливістю процедур, виконуваних по запитам переривань, є те, що вони виконують роботу, найчастіше ніяк не пов'язану з поточним процесом.

Наприклад, драйвер диска може одержати керування після того, як контролер диска записав у відповідні сектори інформацію, отриману від процесу А, але цей момент часу, не співпаде з періодом чергової ітерації виконання процесу А або його потоку.

У найбільш типовому випадку процес А буде перебувати в стані очікування завершення операції введення - виведення (при синхронному режимі виконання цієї операції) і драйвер диска перерве який-небудь інший процес.

У деяких випадках взагалі важко однозначно визначити, для якого процесу виконує роботу той або інший програмний модуль ОС, наприклад планувальник потоків. Тому для такого роду процедур вводяться обмеження - вони не мають права використовувати ресурси (пам'ять, відкриті файли й т.п.), з якими працює поточний процес.

Процедури обробки переривань працюють із ресурсами, які були виділені їм при ініціалізації відповідного драйвера або ініціалізації самої операційної системи. Ці ресурси належать ОС, а не конкретному процесу. Так пам'ять драйверам виділяється із системної області. Тому звичайно говорять, що процедури обробки переривань працюють поза контекстом процесу.

Диспетчеризація переривань є важливою функцією ОС, і ця функція реалізована практично у всіх мультипрограмних ОС. Як правило, в ОС реалізується дворівневий механізм планування робіт. Верхній рівень планування виконується диспетчером переривань, що розподіляє процесорний час між потоком вступників запитів на переривання різних типів - зовнішніх, внутрішніх і програмних. Процесорний час, що залишився, розподіляється іншим диспетчером - диспетчером потоків, на підставі дисциплін квантування й інших, які ми розглядали.

## Системні виклики

Системний виклик дозволяє додатку звернутися до ОС із проханням виконати ту або іншу дію, оформлена як процедура (або набір процедур) кодового сегмента ОС.

Для прикладного програміста ОС виглядає як якась бібліотека, що реалізує корисні функції, що полегшують керування прикладним завданням або виконання дій, заборонених у користувальницькому режимі, наприклад обмін даними із пристроєм введення - виведення.

Реалізація системних викликів повинна задовольняти наступним вимогам:

* забезпечувати перемикання в привілейований режим;
* мати високу швидкість виклику процедур ОС;
* забезпечувати однакове звертання до системних викликів для всіх апаратних платформ, на яких працюють ОС;
* допускати легке розширення набору системних викликів;
* забезпечувати контроль із боку ОС за коректним використанням системних викликів.

У більшості ОС системні виклики обслуговуються за централізованою схемою, заснованою на існуванні диспетчера системних викликів.

При будь-якому системному виклику додаток виконує програмне переривання з певним й єдиним номером вектора.

Перед виконанням програмного переривання додаток передає ОС номер системного виклику. Спосіб передачі залежить від реалізації. Наприклад, номер можна помістити в певний регістр процесора або передати через стек. Також деяким способом передаються аргументи системного виклику, вони можуть поміщатися як у регістри загального призначення, так і передаватися через стек або масив оперативної пам'яті.

|  |
| --- |
| **…** |
| Адреса процедури 21h |
| Адреса процедури 22h |
| Адреса процедури 23h |
| **…** |

Системный

виклик

Вектор = 80h,

RØ = 21h

**ОС Linux**

Таблиця переривань

системи

|  |
| --- |
|  |
| Адреса диспетчера  системних викликів |
|  |

|  |
| --- |
|  |
| Диспетчер  системних викликів |
|  |
| Процедура обробки  Системного виклику 21h |
|  |
| Процедура обробки  Системного виклику 22h |
|  |
| Процедура обробки  Системного виклику 23h |
|  |

Рис.9.1 Централізована схема обробки системних викликів.

Після завершення роботи системного виклику керування повертається диспетчерові, при цьому він одержує також код завершення цього виклику. Диспетчер відновлює регістри процесора, поміщає в певний регістр код повернення й виконує інструкцію повернення з переривання, що відновлює непривілейований режим роботи процесора.

Описаний табличний спосіб організації системних викликів прийнятий практично у всіх операційних системах. Він дозволяє легко модифікувати вміст системних викликів, просто додавши в таблицю нову адресу й розширивши діапазон припустимих номерів викликів.

ОС може виконувати системні виклики в **синхронному**або **асинхронному**режимах.

**Синхронний системний виклик**означає, що процес, що зробив такий виклик, припиняється доти, доки системний виклик не виконає всю необхідну роботу*.* Після цього планувальник переводить процес у стан готовності.

**Асинхронний системний виклик**не приводить до переводу процесу в режим очікування після виконання деяких початкових системних дій, наприклад, запуску операції введення - виведення, керування повертається прикладному процесу.

Більшість системних викликів в ОС є синхронними.

Очікуванняя

Прикладний

процес

Код

системного

виклику

Системний

виклик

б) Синхронний системний

виклик

Обробка системного

виклику

Перевірка події:

завершення системного виклику

Прикладний

процес

а) Асинхронний системний

виклик

Обробка системного

виклику

Рис.9.2.а) Асинхронний системний виклик; б) Синхронний системний виклик.

# Керування пам'яттю

## Функції ОС по керуванню пам'яттю

Під **пам'яттю**розуміється оперативна пам'ять комп'ютера. На відміну від пам'яті жорсткого диску, що називають **зовнішньою пам'яттю***,* оперативна пам'ять для збереження інформації вимагає постійного електроживлення.

Особлива роль пам'яті в тому, що процесор може виконувати інструкції програми тільки в тому випадку, якщо вони перебувають у пам'яті.

Пам'ять розподіляється як між модулями прикладних програм, так і між модулями самої операційної системи.

Функціями ОС по керуванню пам'яттю в мультипрограмній системі є:

* відстеження вільної й зайнятої пам'яті;
* виділення пам'яті процесам і звільнення пам'яті по завершенні процесів;
* витиснення кодів і даних процесів з оперативної пам'яті на диск, коли розміри основної пам'яті не достатні для розміщення в ній всіх процесів, і повернення їх в оперативну пам'ять, коли в ній звільнюється місце;
* настроювання адрес програми на конкретну область фізичної пам'яті.

Під час роботи ОС доводиться створювати нові службові інформаційні структури, такі як описувачі процесів і потоків, різні таблиці розподілу ресурсів, буфери для обміну даними тощо. Всі ці системні об'єкти вимагають пам'яті.

У деяких ОС під час установки резервується деякий фіксований обсяг пам'яті для системних потреб. В інших же ОС використовується більш гнучкий підхід, при якому пам'ять для системних цілей виділяється динамічно.

**Захист пам'яті** — ще одне важливе завдання ОС. Вона полягає в тому, щоб не дозволити виконуваному процесу записувати або читати дані з пам'яті, призначеної іншому процесу.

## Типи адрес

Для ідентифікації змінних і команд на різних етапах життєвого циклу програми використовуються символьні імена, віртуальні адреси й фізичні адреси.

Символьні

імена

Віртуальні

адреси

Фізичні

адреси

Транслятор

Ідентифікатори змінних

у програмі

алгоритмічною мовою

Умовні адреси,

вироблювані транслятором

Номера комірок

фізичної пам'яті

Рис.10.1. Ідентифікація змінних і команд на різних етапах життєвого циклу програми.

* **Символьні імена –**привласнює користувач при написанні програми.
* **Віртуальні (умовні) адреси** *–* виробляє транслятор, що перекладає програму на машинну мову.
* **Фізичні адреси** відповідають номерам комірок оперативної пам'яті, де в дійсності розташовані змінні й команди.

Сукупність віртуальних адрес називається **віртуальним адресним простором***.*

Діапазон можливих адрес віртуального простору у всіх процесів є тим самим. Проте, кожен процес має власний віртуальний адресний простір - транслятор привласнює віртуальні адреси змінним і кодам кожній програмі незалежно.

У різних ОС використовуються різні способи структуризації адресного простору.

1. ***Лінійна***послідовності віртуальних адрес. Така структура адресного простору називають також **плоскою (flat***).* При цьому віртуальною адресою є єдине число, що представляє собою зсув відносно початку віртуального адресного простору. Адресу такого типу називають лінійною віртуальною адресою.

2. Віртуальний адресний простір ділиться на частини, названі сегментами. Віртуальна адреса, що представляє собою **пари**чисел (**n**, **m**)*,* де **n**визначає сегмент, a **m** — зсув усередині сегмента*.*

3. Є більш складні способи структуризації, коли віртуальна адреса утворюється трьома (або навіть більше) числами.

Існують два принципово різних підходи до перетворення віртуальних адрес у фізичні.

1. Заміна віртуальних адрес на фізичні виконується один раз для кожного процесу під час початкового завантаження програми на згадку. Виконує це системна програма — *переміщуючий* **завантажник***,* - на підставі наявних у неї вихідних даних про початкову адресу фізичної пам'яті, а також інформації, наданої транслятором про адресно-залежні елементи програми, виконує завантаження програми, сполучаючи її із заміною віртуальних адрес фізичними.

2. Програма завантажується на згадку в незміненому вигляді у віртуальних адресах. При завантаженні ОС фіксує зсув дійсного розташування програмного коду щодо віртуального адресного простору. Під час виконання програми при кожному звертанні до оперативної пам'яті виконується перетворення віртуальної адреси у фізичну.

Останній спосіб є більш гнучким: у той час як переміщуючий завантажник, жорстко прив'язує програму до спочатку виділеної їй ділянці пам'яті, динамічне перетворення віртуальних адрес дозволяє переміщувати програмний код процесу протягом усього періоду його виконання.

Як правило, обсяг віртуального адресного простору перевищує доступний обсяг оперативної пам'яті. У такому випадку ОС для зберігання даних віртуального адресного простору використовує програмну зовнішню пам'ять.

Однак співвідношення обсягів віртуальної й фізичної пам'яті може бути й зворотним.

Варто пам'ятати, що механізми віртуального адресного простору, і віртуальної пам'яті в загальному випадку - це не одне й теж. Можна уявити собі ОС, у якій підтримуються віртуальні адресні простори для процесів, але відсутній механізм віртуальної пам'яті. Це можливо тільки в тому випадку, якщо розмір віртуального адресного простору кожного процесу менше обсягу фізичної пам'яті.

Вміст призначеного процесу віртуального адресного простору, являє собою **образ процесу***.*

Під час роботи процесу постійно виконуються переходи від прикладних кодів до кодів ОС, які або явно викликаються із прикладних процесів як системні функції, або викликаються як реакція на зовнішні події. Для того щоб спростити передачу керування від прикладного коду до коду ОС, а також для легкого доступу модулів ОС до прикладних даних, у більшості ОС її сегменти розділяють віртуальний адресний простір із прикладними сегментами активного процесу.

Тобто віртуальний адресний простір процесу ділиться на дві безперервні частини: системну й користувальницьку. У деяких ОС (наприклад, Windows NT) ці частини мають однаковий розмір - по 2 Гбайти.

Частина віртуального адресного простору кожного процесу, що відводить під сегменти ОС, є ідентичною для всіх процесів. Тому при зміні активного процесу замінюється тільки друга частина віртуального адресного простору. Наприклад, у процесорах Intel Pentium існує два типи системних таблиць: одна - для опису сегментів, загальних для всіх процесів, інша - для опису індивідуальних сегментів даного процесу. При зміні процесу перша таблиця залишається незмінною, а друга - замінюється новою.

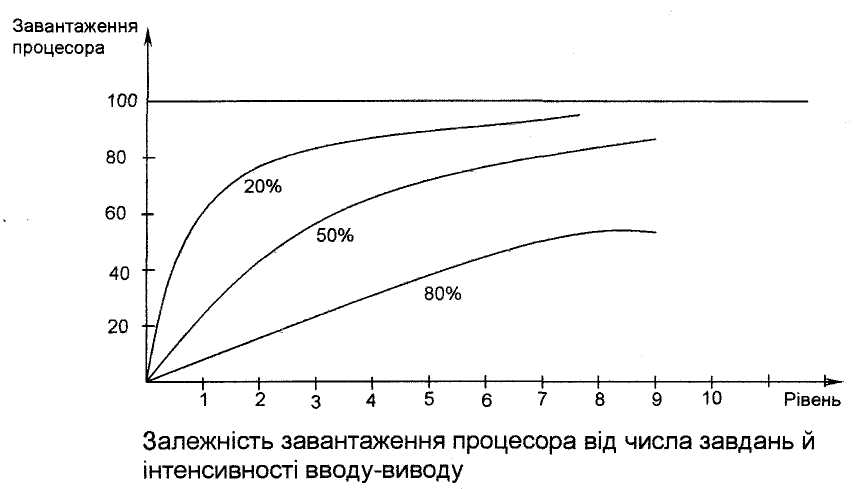


Рис.10.2. Залежність навантаження процесора від числа завдань й інтенсивності введення – виведення.

Велика кількість завдань вимагає великих обсягів оперативної пам'яті.

В умовах, коли для забезпечення прийнятного рівня мультипрограмування наявної пам'яті виявляється недостатньо, був запропонований метод організації обчислювального процесу, при якому образи деяких процесів цілком або частково тимчасово вивантажуються на диск.

У мультипрограмному режимі крім активного процесу, є також припинені процеси, що очікують завершення введення - виведення або звільнення ресурсів, а також процеси в стані готовності, що стоять у черзі до процесора.

Образи таких неактивних процесів можуть бути тимчасово, до наступного циклу активності, вивантажені на диск. Незважаючи на це, ОС «знає» про існування процесів і враховує це при розподілі процесорного часу й інших ресурсів. До моменту, коли підходить черга виконання вивантаженого процесу, його образ повертається з диска в оперативну пам'ять. Якщо при цьому виявляється, що вільного місця в оперативній пам'яті не вистачає, то на диск вивантажується інший процес.

Така підміна **(віртуалізація)** оперативної пам'яті дисковою дозволяє підвищити рівень мультипрограмування.

**Віртуальним** називається ресурс, що користувачеві або користувальницькій програмі представляється володіючим властивостями, якими він, у дійсності, не володіє.

Ясно, що робота віртуальної оперативної пам'яті відбувається повільніше, ніж реальної.

Віртуалізація оперативної пам'яті здійснюється сукупністю програмних модулів ОС й апаратних схем процесора й включає рішення наступних завдань:

• розміщення даних у запам'ятовувальних пристроях різного типу;

• вибір образів процесів або їхніх частин для переміщення з оперативної пам'яті на диск і назад;

• переміщення в міру необхідності даних між пам'яттю й диском;

• перетворення віртуальних адрес у фізичні.

Помітимо, що із проблемою розміщення в оперативній пам'яті програм, які перевищують її розміри, програмісти зіштовхнулися досить давно. Одним із рішень була розбивка програми на частини, які називалися **оверлеями.** Коли один оверлей закінчував своє виконання, він викликав інший оверлей. Всі оверлеї зберігалися на диску й переміщувалися між пам'яттю й диском засобами ОС на підставі директив програміста, що містяться в програмі.

Незважаючи на зовнішню подібність, має принципову відмінність від віртуальної пам'яті. Воно полягає в тому, що при оверлеї розбивка програми на частини й планування їхнього завантаження в оперативну пам'ять виконуються програмістом заздалегідь під час написання програми.

Віртуалізація пам'яті може бути здійснена на основі двох різних підходів:

• **свопінг** — образи процесів вивантажуються на диск і повертаються в оперативну пам'ять *цілком;*

• **віртуальна пам'ять** — між оперативною пам'яттю й диском переміщаються *частими* (сегменти, сторінки) образів процесів.

**Свопінг** - окремий випадок віртуалізації. Це найпростіший спосіб спільного використання оперативної пам'яті й диска.

Недолік - надмірність. Коли ОС вирішує активізувати процес, для його виконання, часто не потрібно завантажувати в оперативну пам'ять всі його сегменти повністю. Аналогічно при звільненні пам'яті для завантаження нового процесу дуже часто не потрібно вивантажувати інший процес на диск повністю. Досить вивантажити тільки частину його образу. Переміщення надлишкової інформації сповільнює роботу системи й при цьому неефективно використовується оперативна пам'ять.

Ще один *недолік.* Системи, що підтримують свопінг не здатні завантажити для виконання процес, віртуальний адресний простір якого перевищує наявну вільну пам'ять.

У сучасних ОС свопінг застосування не знаходить.

*Ключовою проблемою віртуальної пам'яті,* що виникає в результаті багаторазової зміни місця розташування в оперативній пам'яті образів процесів або їхніх частин, є *перетворення* ***віртуальних*** *адресу фізичні.*

У цей час вся безліч реалізацій віртуальної пам'яті може бути представлена трьома класами:

• **Сторінкова віртуальна пам'ять.** Переміщення даних здійснюється сторінками - частинами віртуального адресного простору, фіксованого й порівняно невеликого розміру.

• **Сегментна віртуальна пам'ять** *-переміщення* здійснюється сегментами — частинами віртуального адресного простору довільного розміру.

• **Сегментно-сторінкова віртуальна пам'ять.** Тут використовується дворівневий розподіл: віртуальний адресний простір ділиться на сегменти, а потім сегменти діляться на сторінки. Одиницею переміщення тут є сторінка.

Для тимчасового зберігання сегментів і сторінок на диску виділяється або спеціальна ділянка, або спеціальний файл, що зветься сторінковий файл.

Поточний розмір сторінкового файлу є важливим параметром ОС, чим він більше, тим більше програм може одночасно виконувати ОС. Однак, зі збільшенням сторінкового файлу збільшується час на перекачування інформації й загальна корисна продуктивність системи зменшується.

Розмір сторінкового файлу в сучасних ОС є параметром, що налаштовується.

Тобто, віртуальний адресний простір процесу ділиться на дві неперервні частини: системну та призначену для користувача. В деяких ОС (наприклад, Windows NT) ці частини мають однаковий розмір – по 2 ГБайти.

Частина віртуального адресного простору кожного процесу, яка відводиться під сегменти ОС, являється ідентичною для всіх процесів. Тому при зміні активного процесу замінюється тільки друга частина віртуального адресного простору. Наприклад, в процесорах Intel Pentium існує два типи системних таблиць: одна – для опису сегментів, загальних для всіх процесів, інша – для опису індивідуальних сегментів даного процесу. При зміні процесу перша таблиця залишається незмінною, а друга – замінюється новою.

## Алгоритми розподілу пам'яті

Алгоритми розподілу пам'яті розділені на два класи: алгоритми, у яких використовується переміщення сегментів процесів між оперативною пам'яттю й диском, і алгоритми, у яких зовнішня пам'ять не залучається.

Методи розподілу пам'яті

Без використання

зовнішньої пам'яті

З використанням

зовнішньої пам'яті

Фіксованими

розділами

Динамічними

розділами

Переміщуваними

розділами

Сторінковий

розподіл

Сегментний

розподіл

Сегментно-

сторінковий

розподіл

Рис.10.2. Класифікація методів розподілу пам’яті

**Розподіл пам'яті фіксованими розділами**

Це найпростіший спосіб керування пам'яттю. Пам'ять розбивається на кілька областей фіксованої величини, називаних *розділами.*

Ця розбивка може бути виконана вручну оператором під час старту системи або її установки. Після цього границі розділів не змінюються.

Новий процес, що надійшов на виконання, вміщується або в загальну чергу, або в чергу до деякого розділу*.*

ОС

Розділ 1

Розділ 2

Розділ 3

Черга завдань

– вільна пам'ять

а) Загальна черга

ОС

Розділ 1

Розділ 2

Розділ 3

Черги

до розділів

– вільна пам'ять

б) Окремі черги

Рис.10.3. Розподіл пам'яті фіксованими розділами

Система керування пам'яті вирішує наступні завдання:

* Порівнює обсяг пам'яті, необхідний для нового процесу, з розмірами вільних розділів і обирає підходящий розділ.
* Здійснює завантаження програми в один з розділів і налаштування адрес.

Уже на етапі трансляції розроблювач програми може задати розділ, у якому її варто виконувати. Це дозволяє відразу, без використання переміщуючого завантажника, одержати машинний код, налаштований на конкретну область пам'яті.

Істотний недолік - жорсткість.

Рівень мультипрограмування заздалегідь обмежений числом розділів. Незалежно від розміру програми вона буде займати весь розділ, з іншої сторони процес, що вимагає кілька розділів, не може бути виконаний.

Цей найпростіший метод розподілу пам'яті зараз знаходить застосування тільки в системах реального часу, завдяки детермінованості обчислювального процесу.

## Розподіл пам'яті динамічними розділами

Кожному новому вступникові на виконання процесу виділяється вся необхідна йому пам'ять, якщо її не вистачає, процес не запускається.

Функції ОС, для реалізації цього методу:

* Ведення таблиць вільних і зайнятих областей, у яких вказуються початкові адреси й розміри ділянок пам'яті.
* При створенні нового процесу - аналіз вимог до пам'яті, перегляд таблиці вільних областей і вибір розділу. Вибір розділу може здійснюватися за різними правилами: перший розділ, що попався, достатнього розміру, розділ, що має найменший достатній розмір, розділ, що має найбільший достатній розмір.
* Завантаження програми у виділений їй розділ і коректування таблиць вільних і зайнятих областей.
* Коректування таблиць вільних і зайнятих областей.

Для цього методу характерним є серйозний недолік - фрагментація пам'яті.

Фрагментація – наявність великої кількості несумісних ділянок вільної пам'яті маленького розміру. Такого, що жоден із вступників програм не може поміститися в жодній з ділянок, хоча сумарний обсяг фрагментів може скласти величину, що перевищує необхідний обсяг пам'яті.

Прикладом цього методу є популярна в минулому OS/360 й EC - ЕОМ.

## Переміщувані розділи

Боротьба із фрагментацією - переміщення всіх зайнятих ділянок у бік старших або молодших адрес, так, щоб вся вільна пам'ять утворила вільну єдину область.

На додаток до функцій, які виконує ОС із динамічними розділами, вона ще час від часу копіює вміст розділів з одного місця пам'яті в інше, корегуючи таблиці вільних і зайнятих областей. Ця процедура називається *стиском.*

Стиск виконується або при кожному завершенні процесу, або коли для створюваного процесу немає вільного розділу достатнього розміру.

Програми переміщуються в оперативній пам'яті в ході свого виконання, то в цьому випадку неможливо виконати настроювання адрес за допомогою переміщуючого завантажника. Тут необхідна реалізація динамічного перетворення адрес.

Хоча цей метод приводить до більш ефективного використання пам'яті, але вимагає значних тимчасових витрат, що переважує переваги даного методу.

Метод використовується в ранніх версіях ОС OS/2.

ОС

П1

*a*

П2

П2

П4

*b*

П4

Процедура

стиску

*c*

ОС

*a+b+c*

П1

Рис.10.4 Розподіл пам'яті переміщуваними розділами

## Свопінг і віртуальна пам'ять

Обсяг оперативної пам'яті, що є в комп'ютері, істотно позначається на протіканні обчислювального процесу. Він обмежує число одночасно, що виконуються програм, і розміри їх віртуальних адресних просторів.

Якщо всі завдання мультипрограмної суміші є обчислювальними (мало операцій введення - виведення), для гарного завантаження процесора може виявитися достатнім усього 3-5 завдань.

Якщо обчислювальна система завантажена виконанням інтерактивних завдань, для ефективного використання процесора може знадобитися вже кілька десятків, а те й сотень завдань.

## Сторінковий розподіл

Віртуальний адресний простір кожного процесу ділиться на частини однакового, фіксованого для даної системи розміру – *віртуальні сторінки.*

У загальному випадку розмір віртуального адресного простору процесу не кратний розміру сторінки. Тому остання сторінка кожного процесу доповнюється фіктивною областю.

Вся оперативна пам'ять машини також ділиться на частини такого ж розміру, названі **фізичними сторінками**.

Розмір сторінки вибирається рівним щодо степеню двійки: 512, 1024, 4096 байт тощо. Це дозволяє спростити механізм перетворення адрес.

0

1

2

3

4

0

1

2

3

4

5

Віртуальний адресний

Простір процесу 1

Віртуальний адресний

Простір процесу 2

Віртуальні

сторінки

Віртуальні

сторінки

№стр

5

ВП

ВП

10

2

Стор. 4, п. 1

0

1

2

3

4

Упр. Инф.

Таблиця сторінок

процесу 1

№стр

8

ВП

ВП

ВП

ВП

0

1

2

3

4

Упр. Инф.

Таблиця сторінок

процесу 2

5

11

Стор. 0, п. 1

Стор. 0, п. 2

Стор. 3, пр. 1

Стор. 5, пр. 2

Сторінковий обмін

Рис.10.5. Сторінковий розподіл

Для кожного процесу ОС створює **таблицю сторінок** *—* інформаційну структуру, що містить записи про всі віртуальні сторінки процесу.

Запис таблиці, названий **дескриптором сторінки**йвключає наступну інформацію:

* номер фізичної сторінки, у яку завантажена дана віртуальна сторінка;
* ознака присутності, установлюється в одиницю, якщо віртуальна сторінка перебуває в оперативній пам'яті;
* ознака модифікації сторінки, що встановлюється в одиницю щораз, коли робиться запис за адресою, що ставиться до даної сторінки;
* ознака звертаннядо сторінки, названа також бітом доступу*,* що встановлюється в одиницю при кожному звертанні за адресою, що ставиться до даної сторінки.

Ознаки присутності, модифікації й звертання в більшості сучасних процесорів установлюються апаратно при операціях з пам'яттю. Інформація з таблиць сторінок використовується для рішення питань про необхідність переміщення тієї або іншої сторінки між пам'яттю й диском, а також для перетворення віртуальної адреси у фізичну.

Таблиці сторінок розміщуються в оперативній пам'яті. Адреса таблиці сторінок включається в контекст відповідного процесу. При активізації чергового процесу ОС завантажує адресу його таблиці сторінок у спеціальний регістр процесора.

При кожному звертанні до пам'яті виконується пошук номера віртуальної сторінки, що містить необхідну адресу, потім по цьому номеру визначається потрібний елемент таблиці сторінок, і з нього витягується інформація, що описує сторінку. Далі аналізується ознака присутності, і якщо сторінка перебуває в оперативній пам'яті, то виконується перетворення віртуальної адреси у фізичну. Якщо ж потрібна віртуальна сторінка в цей момент вивантажена на диск, то відбувається *сторінкове переривання.* Процес, що виконується, переводиться в стан очікування, і активізується інший процес із черги процесів, що перебувають у стані готовності. Паралельно програма обробки сторінкового переривання знаходить на диску необхідну віртуальну сторінку й намагається завантажити її в оперативну пам'ять. Якщо в пам'яті є вільна фізична сторінка, то завантаження виконується негайно, якщо ж вільних сторінок немає, то на підставі прийнятої в даній системі стратегії заміщення сторінок, вирішується питання про те, яку сторінку варто вивантажити з оперативної пам'яті.

Після визначення віртуальної сторінки, присвоюється нуль її біту присутності й аналізується її ознака модифікації. Якщо за час останнього перебування в оперативній пам'яті була модифікована, її нова версія повинна бути переписана на диск. Якщо ні, то ніякого запису на диск не робиться (вона й так є). Фізична сторінка оголошується вільною.

Віртуальна адреса при сторінковому розподілі може бути представлена у вигляді пари (р, sv), де p — порядковий номер віртуальної сторінки процесу (нумерація сторінок починається з 0), a sv — зсув у межах віртуальної сторінки.

Фізична адреса також може бути представлена у вигляді пари (n, Sf), де n — номер фізичної сторінки, a Sf — зсув у межах фізичної сторінки. Завдання підсистеми віртуальної пам'яті полягає у перетворенні пари (р, sv) в (n, Sf).

Розглянемо дві базисні властивості сторінкової організації.

**Перша** з них полягає в тому, що обсяг сторінки обирається рівним щодо степеню двійки — 2k . Із цього випливає, що зсув s може бути отримано простим відділенням k молодших розрядів у двійковому записі адреси, а старші розряди адреси, що залишилися, являють собою двійковий запис номера сторінки.

Наприклад:

Розмір сторінки 1 Кбайт (210) двійкової адреса **101 000 111 0012**. Він належить сторінці 102 і зміщений відносно її початку на 1 000 111 0012 байт.

**Друга** властивість полягає в тому, що в межах сторінки безперервна послідовність віртуальних адрес однозначно відображається в безперервну послідовність фізичних адрес, тобто sv = Sf.

Звідси випливає проста схема перетворення віртуальних адрес у фізичні.

Молодші розряди фізичної адреси, що відповідають зсуву, виходять переносом такої ж кількості молодших розрядів з віртуальної адреси. Старші розряди фізичної адреси, що відповідають номеру фізичної сторінки, визначаються з таблиці сторінок, у якій указується відповідність віртуальних і фізичних сторінок.

Якщо відбувається звертання до пам'яті по деякій віртуальній адресі. Апаратними схемами процесора виконуються наступні дії:

* Зі спеціального регістра процесора витягається адреса AT таблиці сторінок активного процесу. На підставі початкової адреси таблиці сторінок, номера віртуальної сторінки p (старші розряди віртуальної адреси) і довжини окремого запису в таблиці сторінок L (системна константа) визначається адреса потрібного дескриптора в таблиці сторінок: а =АТ+(рх).
* Із цього дескриптора витягається номер відповідної фізичної сторінки - n.
* До номера фізичної сторінки приєднується зсув s (молодші розряди віртуальної адреси).

k двійкових розрядів

Віртуальна адреса

Номер віртуальної

сторінки - p

Зсув у віртуальный

сторінці- S

Номера фізичних

сторінок

n

Регістр процесора

AT+(pxL)

Початкова адреса

таблиці сторінок - АТ

Зсув у фізичній сторінці – S

Номер фізичної

сторінки - n

Фізична адреса

k двійкових розрядів

Рис.10.6. Віртуальна сторінка

Апаратний механізм одержання фізичної адреси по віртуальному, передбачений для зменшення часу перетворення адрес.

Іншим важливим фактором, що впливає на продуктивність системи, є частота сторінкових переривань. На неї, у свою чергу, впливають розмір сторінки й прийняті в даній системі правила вибору сторінок для вивантаження й завантаження.

При виборі сторінки на вивантаження зміст критерію зводиться до одного: на диск виштовхується сторінка, до якої довше всього не буде звертань.

Однак, точно передбачити хід обчислювального процесу неможливо, а отже, і точно визначити сторінку, що підлягає вивантаженню. Рішення приймається на основі якихось емпіричних критеріїв.

Наприклад, якщо сторінка не використовувалася довгий час, робиться висновок про те, що вона не буде потрібна й зараз.

Найбільш популярним критерієм є число звертань до сторінки за останній період часу. ОС веде по кожній сторінці програмний лічильник. Коли виникає необхідність видалити яку-небудь сторінку з пам'яті, ОС знаходить сторінку, лічильник звертань якої має найменше значення. Всі лічильники періодично обнуляються.

Інтенсивність сторінкового обміну може бути знижена за рахунок, так названого **попереджуючого завантаження***,* відповідно до якого при виникненні сторінкового переривання у пам’ять завантажується не одна сторінка, а відразу декілька прилягаючих до неї сторінок.

Концептуально передбачається, що якщо звертання відбулося по деякій адресі, то велика ймовірність звертання по сусідніх адресах.

Найважливіше завдання - вибір оптимального розміру сторінки.

Якщо сторінка велика, то великі втрати з фіктивних областей (у середньому половина останньої сторінки). Вибір розміру сторінки - складне оптимізаційне завдання. На практиці розмір сторінки вибирають у кілька кілобайт. Наприклад, у процесорах Pentiumи ОС. На них, підтримують сторінки розміром 4096 байт (4 Кбайт).

Розмір сторінки також впливає на кількість записів у таблицях сторінок процесів. Так у сучасних процесорах максимальний обсяг віртуального адресного простору процесу, не менше 4 Гбайт (232), то при розмірі сторінки 4 Кбайт (212) і довжині запису 4 байти для зберігання таблиці сторінок потрібно 4 Мбайт пам'яті!

Вирішенням виниклої ситуації є зберігання в пам'яті тільки тієї частини таблиці сторінок, що активно використовується в даний період часу, тобто таблицю сторінок можна тимчасово витісняти з оперативної пам'яті.

Саме такий результат може бути досягнутий шляхом ускладнення структуризації віртуального адресного простору, при якому вся безліч віртуальних адрес процесу ділиться на розділи, а розділи діляться на сторінки.

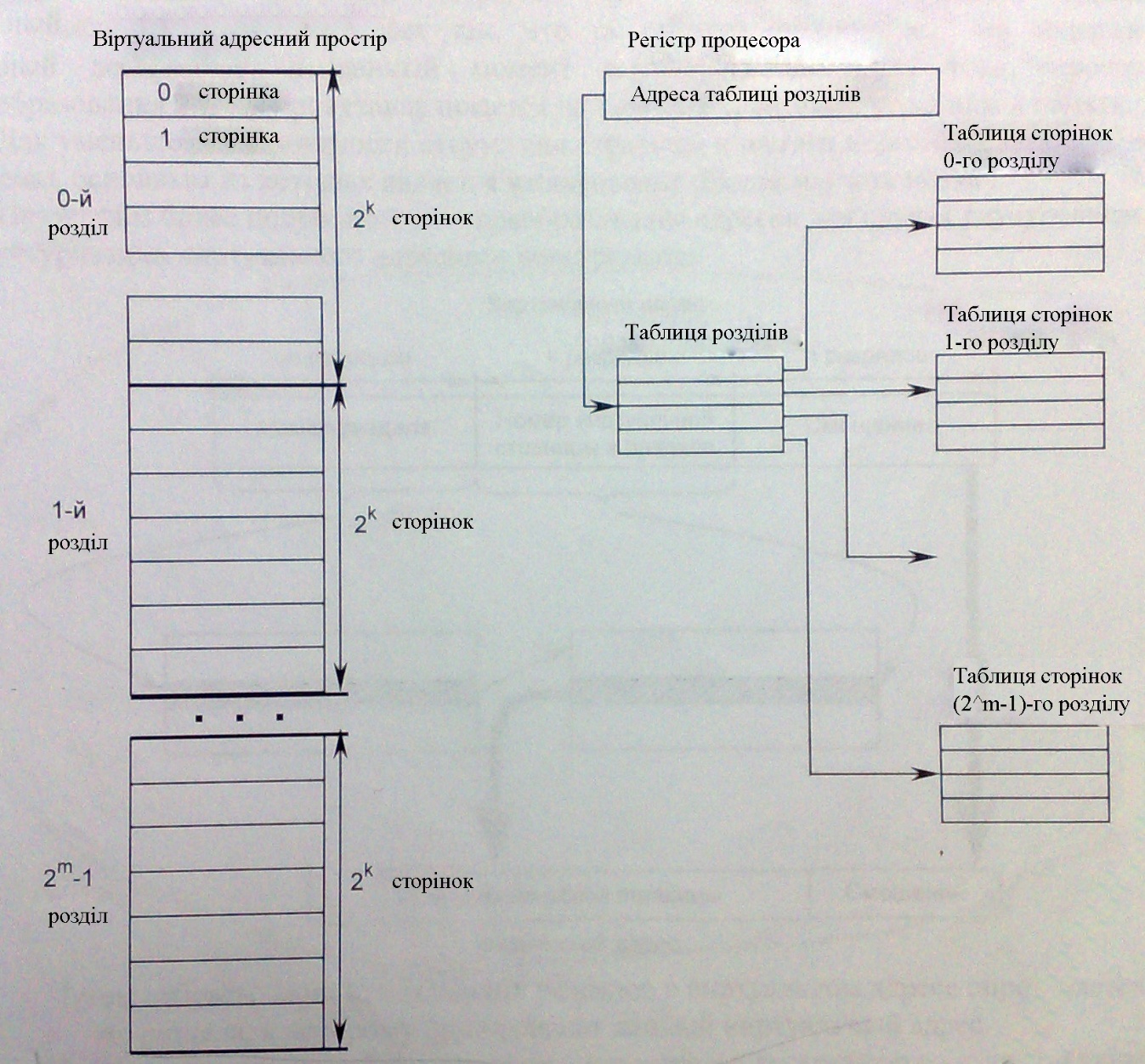


Рис.10.7.

Всі сторінки мають однаковий розмір, а розділи містять однакову кількість сторінок.

Якщо розмір сторінки й кількість сторінок у розділі вибрати рівними щодо степеню двійки (2k й 2n відповідно), то належність віртуальної адреси до розділу й сторінки, а також зсув усередині сторінки визначається дуже просто: молодші k двійкових розрядів дають *зсув*, наступні n розрядів – номер віртуальної сторінки, а старші розряди, що залишилися, містять номер розділу.

Для кожного розділу будується власна таблиця сторінок. Кількість дескрипторів у таблиці і їхній розмір підбираються таким чином, щоб обсяг таблиці виявився рівним обсягу сторінки.

Наприклад, у процесорі Pentium при розмірі сторінки 4 Кбайт довжина дескриптора становить 4 байти й кількість записів у таблиці сторінок, що міститься на сторінку, дорівнює 1024. Кожна таблиця сторінок описується дескриптором, структура якого повністю збігається зі структурою дескриптора звичайної сторінки. Ці дескриптори зведені в таблицю розділів, названу також каталогом сторінок. Фізична адреса таблиці розділів активного процесу міститься в спеціальному регістрі процесора й тому завжди відомий ОС. Сторінка, що містить таблицю розділів, ніколи не вивантажується з пам'яті.

Вивантаження сторінок з таблицями сторінок дозволяє заощаджувати пам'ять, але при цьому приводить до додаткових витрат при одержанні фізичної адреси. Дійсно, може трапитися так, що та таблиця сторінок, що містить потрібний дескриптор, у цей момент вивантажена на диск, тоді процес перетворення адреси припиняється до завантаження необхідної сторінки на згадку.

Для зменшення ймовірності відсутності сторінки в пам'яті використовують різні прийоми, основним з яких є кешування (будемо вивчати пізніше).

Простежимо більш докладно схему перетворення адрес для випадку дворівневої структуризації віртуального адресного простору:

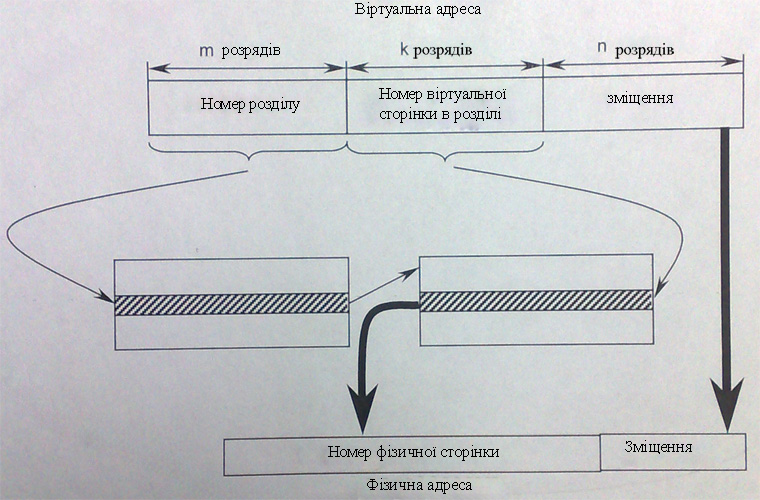


Рис.10.8.Віртуальна адреса

1. Шляхом відкидання k+n молодших розрядів у віртуальній адресі визначається номер розділу, до якого належить дана віртуальна адреса.
2. По цьому номеру з таблиці розділів витягається дескриптор відповідної таблиці сторінок. Перевіряється, чи перебуває дана таблиця сторінок у пам'яті. Якщо ні, відбувається сторінкове переривання й система завантажує потрібну сторінку з диска.
3. Далі із цієї таблиці сторінок витягається дескриптор віртуальної сторінки, номер якої міститься в середніх **n** розрядах преутвореної віртуальної адреси. Знову виконується перевірка наявності даної сторінки в пам'яті й при необхідності її завантаження.
4. З дескриптора визначається номер (базова адреса) фізичної сторінки, у яку завантажена дана віртуальна сторінка. До номера фізичної сторінки пристиковується зсув, узяте з k молодших розрядів віртуальної адреси. У результаті виходить шукана фізична адреса.

**Помітимо** що сторінковий розподіл пам'яті може бути реалізований в спрощеному варіанті, без вивантаження сторінок на диск. У цьому випадку всі віртуальні сторінки всіх процесів постійно перебувають в оперативній пам'яті. Такий варіант не має переваг роботи з віртуальною пам'яттю великого обсягу, але дозволяє успішно боротися із фрагментацією фізичної пам'яті. Ніколи не утворяться невикористовувані залишки. Такий режим використовується в спеціалізованих ОС, коли потрібна висока реактивність системи.

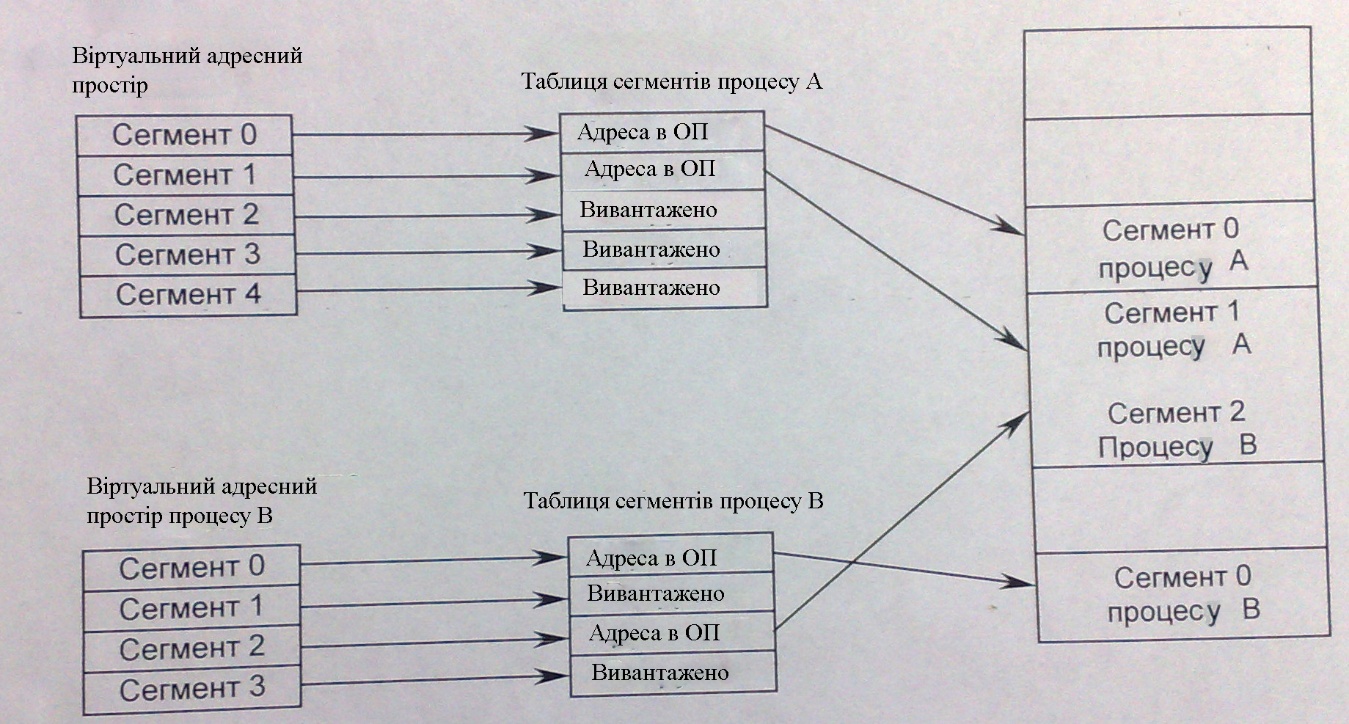
## Сегментний розподіл

При сторінковій організації віртуальний адресний простір процесу ділиться на рівні частини *механічно,* без урахування змісту значення даних.

Однак, розділення віртуального адресного простору на «осмислені» частини робить принципово можливим спільне використання фрагментів програм різними процесами. Наприклад, двом процесам потрібно та сама підпрограма, що до того ж має властивість реентерабельності. Тоді коди цієї підпрограми можуть бути оформлені у вигляді окремого сегмента й включені у віртуальні адресні простори обох процесів.

При відображенні у фізичну пам'ять сегменти, що містять коди підпрограми з обох віртуальних просторів, проектуються на ту саму область фізичної пам'яті. Таким чином, обидва процеси одержать доступ до однієї й тій же копії підпрограми.

Віртуальний адресний простір процесу ділиться на частини — *сегменти.*

**

## Рис.10.9. Сегментний розподіл

Їхній розмір визначається з урахуванням значеннєвого значення інформації, що міститься в них. Окремий сегмент може являти собою підпрограму, масив даних тощо. Ділення віртуального адресного простору на сегменти здійснюється компілятором на основі вказівок програміста або за замовчуванням, на основі прийнятих у системі угод. Максимальний розмір сегмента визначається розрядністю віртуальної адреси, наприклад при 32-розрядній організації процесора він дорівнює 4 Гбайт. При цьому максимально можливий віртуальний адресний простір процесу являє собою набір з N віртуальних сегментів, кожен розміром по 4 Гбайт. У кожному сегменті віртуальні адреси перебувають у діапазоні від 0000000016 до FFFFFFFF16.

Сегменти не впорядковуються відносно один одного, так що загальних для сегментів лінійної віртуальної адреси не існує, віртуальна адреса задається парою чисел: номером сегмента й лінійною віртуальною адресою.

При завантаженні процесу в оперативну пам'ять міститься тільки частина його сегментів, повна копія віртуального адресного простору перебуває в дисковій пам'яті. Для кожного сегмента, що завантажує, ОС підшукує безперервну ділянку вільної пам'яті достатнього розміру. Суміжні у віртуальній пам'яті сегменти одного процесу можуть займати в оперативній пам'яті несуміжні ділянки (це очевидно). Якщо під час виконання процесу відбувається звертання по віртуальній адресі( що відноситься до сегмента), яка у цей момент відсутня у пам'яті, то відбувається переривання. ОС припиняє активний процес, запускає на виконання наступний процес із черги, а паралельно організує завантаження потрібного сегмента. При відсутності в пам'яті місця, ОС вибирає сегмент на вивантаження.

На етапі створення процесу під час завантаження його образа в оперативну пам'ять система створює *таблицю сегментів* процесу (аналогічну таблиці сторінок), у якій для кожного сегмента вказується:

* базова фізична адреса сегмента в оперативній пам'яті;
* розмір сегмента;
* правила доступу до сегмента;
* ознаки модифікації, присутності й звертання до даного сегмента, а також деяка інша інформація

Якщо віртуальні адресні простори декількох процесів включають той самий сегмент, то в таблицях сегментів цих процесів робляться зсилання на ту саму ділянку оперативної пам'яті, у які даний сегмент завантажується в єдиному екземплярі.

Механізми перетворення адрес у сторінковому й сегментному способах організації пам'яті досить схожі. Однак у них є й істотні відмінності. Ці відмінності є наслідком того, що сегменти, на відміну від сторінок мають довільний розмір.

Віртуальна адреса при сегментній організації можна представити парою (**g**, **s**), де **g** - номер сегмента, as- зсув.

Сторінки мають однаковий розмір, і виходить, їхні початкові адреси кратні розміру сторінок. Тому ОС заносить у таблиці сторінок не повні адреси, а номери фізичних сторінок, які збігаються зі старшими розрядами базових адрес.

Сегмент може в загальному випадку розташовуватися у фізичній пам'яті, починаючи з будь-якої адреси. Отже, для визначення місця розташування в пам'яті необхідно задавати його повну початкову фізичну адресу.

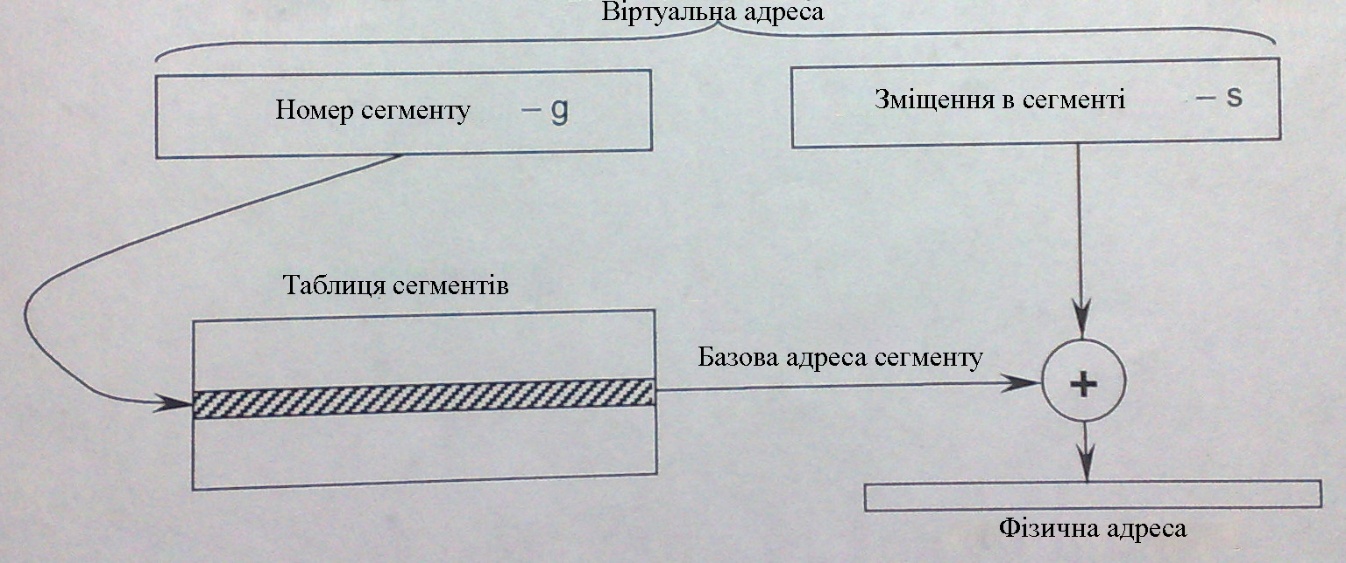


Рис.10.10. Процедура перетворення віртуальної адреси у фізичну

Використання операції додавання сповільнює процедуру перетворення віртуальної адреси у фізичну у порівнянні зі сторінковою організацією (там просте об'єднання старших і молодших розрядів).

Іншим недоліком сегментного розподілу є надмірність. При сегментній організації одиницею переміщення між пам'яттю й диском є сегмент, об’єм якого, як правило, більший, ніж сторінка. Але в багатьох випадках для роботи програми не потрібно завантажувати весь сегмент цілком, досить було б однієї або двох сторінок. Аналогічно при відсутності вільного місця не слід вивантажувати цілий сегмент, якщо можна обмежиться декількома сторінками.

Але головний недолік - це фрагментація, що виникає через непередбачуваність розмірів сегментів.

Найбільш істотною відмінністю сегментної організації від сторінкової є можливість завдання диференційованих прав доступу процесу до його сегментів.

Наприклад, сегмент даних вихідної інформації може мати права доступу «тільки читання», а сегмент результатів, - «читання й запис». Ця властивість визначає перевагу перед сторінковою.

## Сегментно-сторінковий розподіл

Даний метод - спроба об'єднати переваги обох методів.

Тут, як і при сегментній організації пам'яті, віртуальний адресний простір процесу розділений на сегменти. Це дозволяє визначати різні права доступу до різних даних. Переміщення даних між пам'яттю й диском здійснюється не сегментами, а сторінками. Для цього кожен сегмент і фізична пам'ять діляться на сторінки рівного розміру, що дозволяє мінімізувати фрагментацію.

У більшості сучасних реалізацій сегментно-сторінкової організації діапазонів адрес, при сегментній організації, всі віртуальні сегменти утворюють один безперервний лінійний віртуальний адресний простір.

FF…F

00…0

d2

d1

d3

Оперативна пам'ять

Віртуальний адресний простір процесу

Таблиця сегментів процесу

d1

…..

d2

…..

d3

…..

Рис.10.11. Сегментно-сторінкова організація

Координати байта при цьому можна задати двома способами. По-перше, лінійною віртуальною адресою, що дорівнює зсуву даного байта щодо границі загального лінійного віртуального простору, по-друге, парою чисел, одне з яких є номером сегмента, а інше - зсувом щодо сегмента.

При цьому на відміну від сегментної моделі для однозначного завдання віртуальної адреси другим способом необхідно вказати також початкову віртуальну адресу сегмента з даним номером.

Найчастіше використовується другий спосіб, тому що він дозволяє безпосередньо визначити приналежність адреси деякому сегменту й перевірити права доступу процесу до нього.

Для кожного процесу ОС створює окрему таблицю сегментів, у якій утримуються дескриптори (описувачі ) всіх сегментів процесу. Опис сегмента включає призначені йому права доступу й інші характеристики, подібні тим, які містяться в дескрипторах сегментів при сегментній організації пам'яті.

Однак є відмінності. У полі базової адреси вказується не початкова фізична адреса сегмента, а початкова лінійна віртуальна адреса сегмента в просторі віртуальних адрес. Його наявність дозволяє однозначно перетворити адресу, задану парою (g, s) у лінійну віртуальну адресу байта, що потім перетвориться у фізичну адресу сторінковим механізмом.

Розподіл загального лінійного віртуального адресного простору процесу й фізичної пам'яті на сторінки здійснюється так само, як це робиться при сторінковій організації пам'яті.

Базові адреси таблиці сегментів і таблиці сторінок процесу є частиною його контексту. При активізації процесу ці адреси завантажуються в спеціальні регістри процесів і використовуються механізми перетворення адрес.

Перетворення віртуальної адреси у фізичну відбувається у два етапи.

Вихідна віртуальна адреса

номер сегмента

зсув у сегменті

Дескриптор сегмента

таблиця сегментів процесу

базова віртуальна адреса сегмента

Перший етап

логічна віртуальна адреса

номер віртуальної сторінки

зсув у сторінці

Дескриптор сторінки

таблиця сторінок процесу

базова фізична адреса сторінки

шукана фізична адреса

Другий етап

Регістр процесора

Адреса таблиці

сегментів

Рис.10.12. Перетворення віртуальної адреси у фізичну

1. На першому етапі працює механізм сегментації. Вихідна віртуальна адреса пари (g, s) перетворюється в лінійну віртуальну адресу. Для цього на підставі базової адреси сегментів і номера сегмента обчислюється поле дескриптора й виконується перевірка можливості виконання заданої операції.

Якщо доступ до сегмента дозволений, то обчислюється лінійна віртуальна адреса шляхом додавання базової адреси сегмента, витягнутого з дескриптора, і зсуву, заданого у вихідній віртуальній адресі.

2. Другий етап використовує сторінковий механізм. Лінійна віртуальна адреса перетворюється у фізичну. У результаті перетворення лінійна віртуальна адреса представляється в тому виді, у якому вона використовується при сторінковій організації пам'яті (пари: номер сторінки, зсув у сторінці). Далі як уже вивчали раніше.

Старші розряди лінійної віртуальної адреси, що містять номер віртуальної сторінки, заміняються номером фізичної операції, узятим з таблиці сторінок, а молодші розряди віртуальної адреси, що містять зсув, залишаються без зміни.

З розгляду випливає, що механізм сегментації й сторінковий механізм діють досить незалежно друг від друга.

Тому не важко уявити собі реалізацію пам'яті, коли механізм сегментації виконується, як було показано, а сторінковий механізм реалізується за дворівневою схемою: віртуальний адресний простір ділиться спочатку на розділи, а вже потім на сторінки.

У такому випадку перетворення віртуальної адреси у фізичну відбувається у кілька етапів.

Спочатку механізм сегментації звичайним чином, використовуючи таблицю сегментів, обчислює лінійну віртуальну адресу. Потім із цієї адреси обчислюється номер розділу, номер сторінки й зсув. По номеру розділу з таблиці розділів визначається адреса таблиці сторінок, а потім по номеру віртуальної сторінки з таблиці сторінок визначається номер фізичної сторінки, до якої приписується зсув.

Саме цей підхід реалізований у процесорах i386, i486 й Pentium.

Розглянемо ще одну схему керування пам'яттю, засновану на сегментно-сторінковому підході.

Відмінність цього підходу полягає в тому, що віртуальні сторінки нумеруються не в межах усього адресного простору процесу, а в межах сегмента.

Віртуальна адреса в цьому випадку виражається трійкою (номер сегмента, номер сторінки, зсув у сторінці).

Завантаження процесу виконується посторінково. Для кожного процесу створюється своя таблиця сегментів, а для кожного сегмента - своя таблиця сторінок.

Адреса таблиці сегментів завантажується в спеціальний регістр процесора.

Таблиці сторінок повністю аналогічні таблиці сторінок у попередньому випадку.

У таблиці сегментів є істотні відмінності. Вона складається з дескрипторів сегментів, які замість інформації про розташування сегментів у віртуальному адресному просторі містить опис розташування таблиць сторінок у фізичній пам'яті.

.

.

.

таблиця сторінок сегмента g

таблиця сегментов

номер сегмента - g

номер віртуальної сторінки - p

зміщення - s

g

p

n

номер фізичної сторінки

n

зміщення s

Віртуальна адреса (g, p, s)

фізична адреса

Рис.10.13. Схема перетворення віртуальної адреси у фізичну для другого випадку

1. По номеру сегмента, заданому у віртуальній адресі, з таблиці сегментів витягається фізична адреса відповідної таблиці сторінок.

2. По номеру віртуальної сторінки, заданої у віртуальній адресі, з таблиці сторінок витягається дескриптор, у якому зазначений номер фізичної сторінки.

3. До номера фізичної сторінки пристиковується молодша частина віртуальної адреси - зсув.

## Поділювані сегменти пам'яті

Прикладом застосування поділюваної області пам'ять може бути використання її як буфер при міжпроцесному обміні даними. Один процес пише в поділювану область, інший - читає.

Для організації поділюваного сегмента при наявності системи віртуальної пам'яті досить помістити його у віртуальний адресний простір кожного процесу, якому потрібний доступ до даного сегмента, а потім налаштувати параметри відображення цих віртуальних сегментів так, щоб вони відповідали одній й тій же області оперативної пам'яті.

Деталі такого налаштування залежать від пам'яті.

«Влучення» віртуальних сегментів на загальну частину оперативної пам'яті досягається або за рахунок погодженого налаштування ОС численних дескрипторів для процесів, або приміщенням єдиного розділового віртуального сегмента в загальну частину віртуального адресного простору процесів, що звичайно використовують для модулів ОС. У цьому випадку налаштування дескриптора сегмента й дескрипторів сторінок виконується один раз, а всі процеси користуються таким будуванням й спільно використовують частину оперативної пам'яті.

При роботі з поділюваними сегментами пам'яті необхідно дотримуватися загальних правил використання поділюваних ресурсів - семафорів, моніторів тощо.

Для відмінності поділюваних сегментів пам'яті від індивідуальних. Дескриптор сегмента повинен містити поле, що має два значення shared(поділюваний) або private(індивідуальний).

Поділювані сегменти вивантажуються на диск системою віртуальної пам'яті по тому ж алгоритму і за допомогою тих же механізмів що й індивідуальні.

## Кешування даних

Пам'ять ЕОМ - це ієрархія ЗП, що відрізняються середнім часом доступу до даних, обсягом і вартістю зберігання 1 біта інформації.

Фундамент цієї піраміди - пам'ять на жорстких дисках, але час доступу до диска обчислюється мілісекундами.

Оперативна пам'ять – має час доступу 10-20 наносекунд (від декількох мегабайт до декількох гігабайт)

Надоперативна пам'ять (десятки сотні кілобайт) - 8нсек.

Внутрішні регістри процесора - кілька десятків байт з часом доступу 2-3 наносекунди.

**Кеш пам'ять** або просто **кеш** спосіб спільного функціонування двох типів запам'ятовувальних пристроїв, що відрізняються часом доступу й вартістю зберігання даних, що за рахунок динамічного копіювання у швидкі ЗП найбільш часто використовуваної інформації з «повільного» ЗП дозволяє, з одного боку, зменшити середній час доступу до даних, з іншого боку - заощадити більш дорогу швидкодіючу пам'ять.

Особливістю кешування є те, що система не вимагає ніякої зовнішньої інформації про інтенсивність використання даних, ні користувачі, ні програми не приймають ніякої участі в переміщенні даних із ЗП одного типу в ЗП іншого типу, все це робиться автоматично системними засобами.

Кешем часто називають не тільки спосіб організації двох типів запам'ятовувальних пристроїв, але й один із пристроїв - «швидке ЗП». Воно дорожче й порівняно невеликого обсягу на противагу «повільному» ЗП - оперативної пам'яті.

Якщо **кешування** використовують для зменшення середнього часу **доступу до оперативної пам'яті**, то в якості КЕШа використовують більш дорогу й швидкодіючу статичну пам'ять.

Якщо **кешування** використовується системою введення - виведення для прискорення доступу до даних, що **зберігається на диску,** роль кеш-пам'яті грають буфери, реалізовані в оперативній пам'яті.

**Віртуальну пам'ять** теж можна розглядати як окремий випадок **кешування**.

**Принцип дії кеш-пам'яті**

Вміст кеш-пам'яті являє собою сукупність записів про всі завантажені в неї елементи даних з основної пам'яті. Кожен запис включає:

* елементи даних;
* адресу, яку цей елемент має в основній пам'яті;
* додаткову інформацію, що використовується для реалізації алгоритму, включає ознаку модифікації й ознаку дійсності даних.

При кожному звертанні до основної пам'яті по фізичній адресі проглядається вміст кеш-пам'яті з метою визначення, чи не перебувають там потрібні дані.

Кеш-пам'ять не є адресуємою, тому пошук потрібних даних здійснюється по вмісту - по взятому із запиту значенню поля адреси в оперативній пам'яті.

Далі можливі два варіанти розвитку:

* дані виявлені в кеш-пам'яті, тобто зроблено кеш-потрапляння(cache-hit), вони зчитуються й передаються джерелу запиту;
* потрібні дані відсутні, тобто відбувся кеш-перенос (cache-miss), вони зчитуються з основної пам'яті, передаються джерелу запиту й одночасно копіюються в кеш-пам'ять.

Ефективність кешувания залежить від імовірності влучення в кеш.

швидка відповідь (кеш – влучення)

джерело запитів до основної пам'яті

КЕШ

основна пам’ять

повільна відповідь (кеш - промах)

Рис.10.14. Принцип дії кеш-пам'яті

Якщо позначити ймовірність кеш-влучання через p, а час доступу до основної пам'яті через t1, час доступу через t2, то по формулі повної ймовірності середній час доступу буде дорівнює:

t=t2p+t1(1-p)

Якщо p=1, час доступу дорівнює t2.

Імовірність виявлення даних у кеші залежить від різних факторів, таких як:

* обсяг кеша;
* обсяг кешуємої пам'яті;
* алгоритм заміщення даних у кеші;
* особливостей виконуваної програми й т.п.

На практиці відсоток влучень виявляється досить високим - близько 90%. Такий відсоток обумовлюється наявністю в даних об'єктивних властивостей таких як просторової й тимчасової локальності.

**Просторова локальність**. Якщо відбулося звертання по деякій адресі, то з високою ймовірністю найближчим часом відбудеться звертання по сусідніх адресах.

**Часова локальність**. Якщо відбулося звертання по деякій адресі, то наступне звертання по тій же адресі з великою ймовірністю відбудеться найближчим часом.

Оскільки при виконанні програми дуже висока ймовірність, що команди вибираються з пам'яті одна за іншою із сусідніх комірок, має сенс завантажувати в кеш цілий фрагмент програми. Аналогічно й з масивами даних.

У процесі роботи вміст кеш-пам'яті постійно оновлюється. Витиснення даних означає або просто оголошення вільної деякої області кеш-пам'яті (скидання біта дійсності) мінялися або на додаток до цього копіювання даних в основну пам'ять, якщо вони модифікувалися.

Наявність у комп'ютері двох копій даних - в основній пам'яті й у кеші - породжує проблему узгодження даних.

Існує два підходи до рішення цієї проблеми:

* наскрізний запис (write through). При запиті до основної пам'яті (у тому числі при записі) проглядається кеш. Якщо дані по запитуваній адресі відсутні, запис виконується тільки в основну пам'ять. Якщо дані перебувають у кеші, запис робиться як у кеш, так і у пам'ять.
* зворотний запис (write back). Виконується перегляд кешу, якщо даних там немає, то запис робиться в основну пам'ять. У іншому випадку запис робиться тільки в кеш. При цьому встановлюється ознака модифікації. При витисненні даних з кешу ці дані будуть переписані в основну пам'ять.

**Способи відображення основної пам'яті на кеш**

Алгоритми пошуку й заміщення даних у кеші залежать від того, як основна пам'ять відображається на кеш.

Використовуються дві схеми відображення:

* випадкове відображення;
* детерміноване відображення.

При випадковому відображенні елемент оперативної пам'яті може бути розміщений у довільному місці кеш пам'яті. Він вміщується туди разом зі своєю адресою в оперативній пам'яті. Пошук інформації здійснюється по цій адресі. Процедури простого перебору адрес при великих тимчасових витратах.

Тому використовується так званий асоціативний пошук, при якому порівняння виконується не послідовно з кожним записом у кеші, а паралельно з усіма його записами. Ознака, по якій виконується порівняння, називається **тегом** (tag). У цьому випадку тегом є адреса даних в оперативній пам'яті.

Електронна реалізація такого пошуку робить кеш-пам'ять значно дорожчою. Тому цей метод використовується для забезпечення високого відсотка влучення при невеликому обсязі кеш-пам'яті.

У кешах на основі випадкового відображення витиснення старих даних відбувається тільки в тому випадку, коли вся кеш-пам'ять заповнена і немає вільного місця. Вибір даних на вивантаження ґрунтується на тих же принципах, що й при заміщенні сторінок (давно немає звертань, найменше звертань і т.д.).

При **детермінованому** способі відображення будь-який елемент основної пам'яті відображається в те саме місце кеш-пам'яті. У цьому випадку кеш-пам'ять розділена на рядки, кожен з яких призначений для зберігання одного запису про один елемент даних і має свій номер.

Між номерами рядків кеш-пам'яті й адресами оперативної пам'яті встановлюється відповідність «**один до багатьох**»: одному номеру рядка відповідає трохи (досить багато) адрес оперативної пам'яті.

Як відображаюча функція може використовуватися просте виділення декількох розрядів з адреси оперативної пам'яті, які інтерпретуються як номер рядка кеш-пам'яті. Таке відображення називається **прямим**. Наприклад, кеш розрахований на 1024 записи (1024 рядка). Тоді будь-яка адреса оперативної пам'яті може бути відображена на адресу кеш-пам'яті простим відділенням 10 двійкових розрядів.

джерело запитів до основної пам'яті

oсновна

пам'ять

Адреса ОП із запиту

у цьому випадку 10 розрядів

тег

номер елемента кеш – пам’яті

шукані дані

Пам’ять тегів

Пам’ять даних

КЕШ

Рис.10.15.Пряме відображення основної пам'яті на кеш

При пошуку даних у кеші використовується швидкий прямий доступ до запису по номеру рядка, отриманому з адреси оперативної пам'яті із запиту. Крім того, виконується додаткова перевірка на збіг тегу з відповідною частиною адреси із запиту.

При збігу тегу з відповідною частиною адреси із запиту констатується кеш влучення. Якщо немає, констатується кеш-промах і дані зчитуються з OП і копіюються в кеш.

У багатьох сучасних процесорах кеш-пам'ять будується на основі сполучення цих двох підходів, що дозволяє знайти компроміс.

При змішаному підході довільна адреса оперативної пам'яті відображується не на одну адресу кеш-пам'яті (як це характерно для прямого відображення), і не на будь-яку адресу кеш-пам'яті (як це робиться при випадковому відображенні), а на деяку групу адрес. Всі групи пронумеровані. Пошук у кеші здійснюється спочатку по номеру групи, отриманому з адреси ОП із запиту, а потім у межах групи шляхом асоціативного перегляду всіх записів групи на предмет збігу старших частин адрес ОП

1010111

0010

Адреса запиту

тег

номер групи

Пам’ять тегов

Пам’ять даних

1010111

шукані дані

0 група

1 група

2 група

(0010)

n група

.

.

.

.

.

.

Рис.10.16. Комбінування прямого й випадкового відображення

При промаху дані копіюються по будь-якій вільній адресі з однозначно заданої групи.

Якщо вільних адрес у групі немає, то виконується витиснення даних. Оскільки кандидатів на вивантаження кілька - всі записи з даної групи, то алгоритм заміщення може врахувати інтенсивність звертання до даних і тим самим підвищити ймовірність влучень у майбутньому.

Як ми бачили, кеш проглядається тільки з метою узгодження вмісту кеша й основної пам'яті. Якщо відбувається промах, то запити на запис не викликають ніяких змін кеша.

У деяких реалізаціях кеш-пам'яті при відсутності даних у кеші вони копіюються туди з основної пам'яті незалежно від того, виконується запит на читання або запис.

Відповідно до описаної логіки роботи кеш пам'яті, при виникненні запиту спочатку проглядається кеш, а потім, якщо відбувся промах, виконується звертання до основної пам'яті.

Однак, часто реалізується й інша схема роботи кеша: **пошук** у кеші й основній пам'яті **починається одночасно,** потім у результаті перегляду кеша, операція в основній пам'яті або триває, або переривається.

У ряді обчислювальних систем використовується дворівневе кешування.

Кеш першого рівня має менший обсяг і більш високу швидкодію, ніж кеш другого рівня. Кеш другого рівня відіграє роль основної пам'яті стосовно кешу першого рівня.

Кеш 1

Кеш 2

ОП

Рис.10.17. Дворівневе кешування

Спочатку робиться спроба виявити дані в кеші 1-го рівня. Якщо відбувся промах, пошук триває в кеші другого рівня. Якщо потрібні дані відсутні й тут, тоді відбувається зчитування даних з основної пам'яті.

При зчитуванні даних з ОП відбувається їхнє копіювання в кеш другого рівня, а якщо дані зчитуються з кеша другого рівня, то вони копіюються в кеш першого рівня.

Кеши різних рівнів можуть погоджувати дані різними способами. Наприклад, у процесорі Pentium кеш першого рівня використовує кеш першого рівня - зворотний запис.

Помітимо, що ми розглядали системи, у яких на кожному рівні є тільки один кеш. Однак існує цілий ряд систем - розподілених систем обробки інформації, у яких на кожному рівні є кілька кешів.

# Введення - виведення і файлова система

Основними компонентами підсистеми введення - виведення є драйвери, керуючі зовнішніми пристроями, і файлові системи.

До **підсистеми введення - виведення** можна також віднести й **диспетчер переривань**; але це досить умовно, тому що він обслуговує не тільки модулі підсистеми введення - виведення але також планувальник або диспетчер потоків.

Файлову систему доцільно розглядати разом з іншими компонентами введення - виведення з двох причин:

* файлова система активно використовує введення - виведення;
* модель файлу може бути в основі більшості механізмів доступу до пристроїв введення - виведення.

## Завдання ОС по керуванню файлами й пристроями

Підсистема введення - виведення мультипрограмної ОС вирішує наступні основні завдання:

* організація паралельної роботи пристроїв введення - виведення й процесора;
* узгодження швидкостей обміну й кешування даних;
* поділ пристроїв і даних між процесами;
* забезпечення зручного логічного інтерфейсу між пристроями й іншою частиною системи;
* підтримка широкого спектру драйверів з можливістю простого включення в систему нового драйвера;
* динамічне завантаження й вивантаження драйверів;
* підтримка декількох файлових систем;
* підтримка синхронних й асинхронних операцій введення - виведення.

## Основні поняття та концепція організації введення - виведення

Складність реалізації введення - виведення полягає у тому, що перед проектувальниками постає задача підтримки широкої частини пристроїв з одного боку, а з другого боку – створена прийняттю та ефективного віртуального інтерфейсу незалежно від специфіки пристроїв введення - виведення та принципів їх розподілу між виконуваними задачами.

Тобто, система введення - виведення повинна бути універсальною, щоб об’єднувати різні пристрої від миші до графічних дисплеїв та накопичувачів на магнітних дисках. З іншого боку доступ до цих пристроїв повинен бути таким, щоб паралельно виконувані задачі не заважали одна одній, якщо вони використовують одні і ті самі пристрої.

Тому завжди використовують такий найголовніший принцип, операції введення - виведення оголошуються привілейованими і виконуються тільки кодами даної ОС.

Для забезпечення цього принципу вводять режим супервізора та користувача. Як правило, виконання введення - виведення – розподілювальні ресурси, тому їх використання потребує синхронізації у загальному випадку, хоча можуть існувати і нероздільні пристрої, наприклад, принтер.

Можна назвати три основні принципи за якими не можна кожній окремій користувацькій програмі звертатись до зовнішнього пристрою безпосередньо:

1. Необхідність вирішення можливих конфліктів доступу до пристроїв. Наприклад, якщо дві або декілька програм будуть виводити свої результати на папір, то окрім проблеми звичайної синхронізації доступу виникає додаткова проблема: як розділити між собою ті результати, які будуть друкуватися у перемішку.
2. Бажання збільшити ефективність використання цих ресурсів. Наприклад у накопичувача на магнітних дисках час допуску до необхідної доріжки та звертання до відповідного сектору може на декілька порядків перевищувати час пересилки даних.

Тому, якщо задачі звертаються до циліндрів по черзі, які далеко розташовані один від одного, ефективний час роботи накопичувача буде суттєво зменшений.

1. Похибки у програмах введення - виведення можуть призвести до зупинки усіх процесів, бо частина операції введення - виведення виконується для самої ОС. Власне у ряді ОС операції введення - виведення для потреб ОС мають суттєві привілеї перед такими операціями для потреб звичайних процесів. Системний код, що керує операціями введення - виведення ретельно відпрацьовується для підвищення надійності та ефективного використання обладнання.

Тобто управління введенням - виведенням виконується операційно. Системою, власне її частиною, яка називається супервізором введення - виведення.

Він виконує наступні функції:

1. Отримання запитів на введення – виведення від прикладних програм та власних модулів ОС. Ці запити перевіряються на коректність. Якщо він виконаний відповідно до специфікацій та не має помилок,він оброблюється. У протилежному випадку видається відповідне діагностичне повідомлення.
2. Виклик розподілювачів каналів та контролерів планування введення - виведення. Він визначає чергу надання пристроїв задачам або процесам. Їх запити виконуються або ставляться у чергу на виконання.
3. Ініціює операції введення - виведення (передає керування відповідним драйверам) і у випадку, коли керування введенням - виведенням використовує переривання, надає процесор диспетчеру задач для того, щоб надати його задачі, що стоїть першою у черзі на виконання.
4. При отриманні сигналів переривання від пристроїв введення – виведення, ідентифікує їх та передає керування відповідної задачі обробки переривань.
5. Виконує передачу повідомлень про похибки введення - виведення.
6. Надсилає повідомлення про завершення операції введення - виведення процесу, який її запросив, та знімає його зі стану очікування введення - виведення, якщо процес очікував на її завершення.

Якщо пристрій є ініціативним ( це зовнішні пристрої, окрім стандартних пристроїв введення - виведення, наприклад, датчики), керування зі сторони супервізора введення - виведення буде полягати у активізації відповідного обчислювального процесу ( або переведення його у стан готовності).

Тобто прикладні програми безпосередньо не зв'язуються з пристроями введення – виведення, незалежно від виду використання – монопольного чи сумісного. Вони це роблять централізовано, через використання супервізора введення – виведення.

## Режим керування введенням-виведенням

Існують два основних режими введення - виведення:

1. режим обміну з опитуванням готовності пристрою введення - виведення;
2. режим обміну з привілеями



Рис.11.1. Керування введенням-виведенням

Розглянемо Рис.11.1. Нехай, керування виконує центральний процесор. Тобто використовується програмний канал обміну між зовнішнім пристроєм та оперативною пам'яттю (ще можливий обмін між зовнішнім пристроєм та каналом прямого доступу у пам'ять за допомогою спеціального об’єднання).

Центральний процесор надає пристрою керування команду та виконання пристроєм введення - виведення. Останнє виконує команду, перетворюючи її у послідовність сигналів, зрозумілих пристрою введення - виведення. Але швидкодія пристрою введення - виведення на декілька порядків менша ніж у процесора. Тому сигнал прийняття чергової команди треба досить довго чекати, постійно опитуючи відповідну лінію інтерфейсу на проходження відповідного сигналу.

У режимі опитування готовності драйвер, що керує процесом обміну дає виконати команду «перевірити наявність сигналу готовності». До тих під доки останній не з’явиться, задарма витрачається час процесора.

Значно вигідніше тимчасово забути про пристрій введення - виведення після видачі команди введення - виведення. Саме ці сигнали готовності і будуть сигналами запиту на поривання.

Режим обміну з перериваннями є режимом асинхронного керування. Для того щоб не загубити зв'язок з пристроєм після того як процес видав чергову команду по керуванню обміном і перейшов на виконання інших програм, може бути організовано відлік часу за який пристрій має виконати команду та видати сигнал переривання. Максимальне значення такого часу називають **установкою таймаута**. Якщо цей час витрачено, то роблять висновок, що зв'язок із пристроєм втрачено і керування неможливе, про що користувач або задача отримують відповідне повідомлення.

Драйвери що працюють у режимі переривань являють собою комплекс програмних модулів, що мають декілька секцій: секцію запуску, одну або декілька секцій продовження та секцію завершення.

**Секція запуску** ініціалізує операцію введення - виведення. Вона включає пристрій введення - виведення або ініціалізує чергову ітерацію введення - виведення.

**Секція продовження** виконує основну роботу по передачі даних. Вона і є основним обробником переривання. Узагальненим випадком використання інтерфейс може вимагати декілька послідовностей команд керування, а сигнал переривання з пристрою, як правило тільки один. Тому після виконання чергової секції переривання супервізор переривань при черговому сигналі готовності повинен передати керування черговій секції. Якщо ж така секція тільки одна, вона сама передає керування певному модулю обробки.

**Секція завершення** виключає пристрій виведення або просто завершує операцію.

Програми, що керують введенням - виведенням у режимі переривань створювати досить складно. Тому, наприклад, драйвер для другу через паралельний порт в ОС Windows працює не в режимі переривань, а в режимі запитів готовності, що призводять до 100% завантаження процесора. Виконання інших завдань виконується винятково за рахунок витісняючи стратегій, коли процесор примусово передається для виконання інших задач.

## Закріплення пристроїв. Загальні пристрої введення - виведення

Цілий ряд пристроїв не допускає паралельного використання. Якщо такі пристрої закріпити за певним процесом, то може з’ясуватись, що деякі процеси взагалі виконуватись не зможуть. Для організації паралельного використання таких пристроїв вводять так звані віртуальні пристрої. Сюди ж тісно прилягає поняття спулінга, тобто імітації паралельного розподілу пристрою введення - виведення з послідовним добутком.

У цьому випадку кожному процесу надається, скажімо для друку, не реальний, а віртуальний принтер і потік символів, що виводяться, направляються не на пристрій друку, а в спеціальні спул-файли на диску. По закінченню віртуального друку, можна вивести цей файл на пристрій друку. Системний процес, що керує спул-файлом, зветься **спулером**.

## Основні системні таблиці введення - виведення

Кожна ОС має свої таблиці введення - виведення. Їх кількість та склад можуть сильно різнитися. Існують ОС, у яких замість таблиць використовують списки, але використання таблиць призводить до більшої продуктивності.

Базуючись на принципі керування введенням - виведенням через супервізор та те, що драйвери використовують механізм переривань можна зробити висновок про необхідність створення хоча б трьох системних таблиць.

**Перша таблиця** зберігає інформацію про всі пристрої введення - виведення, що підключені до даної системи. Вона носить назву **таблиці обладнання**. Кожний елемент такої таблиці має назву UCB (Unit Control Block).

UCB, як правило, має таку інформацію:

* тип пристрою, конкретна модель, символьне ім’я та характеристика пристрою;
* через який інтерфейс підключено, до якого з’єднування, які порти та лінії запиту переривань використовуються;
* номер та адреса каналу, якщо вони використовуються для керування;
* вказівка на драйвер, який має управляти цим пристроєм, адреса секції запуску та секцій проводження;
* інформація про те, використовується буферизація чи ні при обміні даними з цим пристроєм, ім’я буферу, якщо він виділений у алгоритмічній області пам'яті;
* установка тайм-аута та комірка для лічильника тайм-аута;
* стан пристрою;
* поле вказівника для зв'язку задач, що очікують пристрій, можливо ще якісь дані;

Оскільки драйвери можуть мати властивості реєнтерабельності ( один екземпляр програми може забезпечувати паралельне обслуговування декількох пристроїв одного типу), то у елементі UCB повинна зберігатись сама інформація про даний стан пристрою та самі змінні для реєнтерабельної обробки, або вказівку на ту адресу, де таку інформацію можна відшукати.

Дуже важливим компонентом UCB є вказівник на дескриптор тієї задачі, яка на даний час використовує пристрій введення - виведення.

Якщо пристрій вільний – відповідне поле вказівника буде мати нульове значення. Якщо пристрій введення - виведення вже зайнятий і відповідно вказівник не нульовий, то нові запити до пристрою фіксуються через створення списку з дескрипторів цих задач, що очікують даний пристрій.

**Друга таблиця** призначена для реалізації принципу візуалізації. Бажано, щоб програміст міг би враховувати тільки найбільш загальні можливості даного класу пристроїв введення - виведення. Наприклад, принтер має виводити на друк символи або графічні зображення. А накопичувач на дисках мав би використовуватись через файлову систему. Тому запити на введення - виведення програми має логічне ім’я пристрою, абстрагуючись від його конкретних особливостей та характеристик. Конкретний пристрій, який відповідає віртуальному (логічному) обирає супервізор за допомогою цієї другої таблиці. Ця таблиця дістала назву - таблиця опису віртуальних пристроїв (DRT, device reference table). Тобто призначення цієї таблиці – встановлення зв'язку між віртуальними (логічними) пристроями та реальними пристроями, описи яких задані у першій таблиці. Друга таблиця власне дозволяє супервізору перенаправляти запит на введення - виведення із прикладної задачі на ті програмні модулі та структури даних, які зберігаються у відповідному елементі першої таблиці.

У багатокористувацьких системах така таблиця не одна, а декілька – по одній на кожного користувача.

**Третя таблиця** необхідна для організації зворотнього зв'язку між центральною частиною та пристроями введення - виведення. Це таблиця **переривань.** Вона вказує для кожного запиту на переривання той елемент UCB, який співставляється даному пристрою, підключеному так, що воно використовує дану лінію (сигнал) переривання. Взаємозв'язок між таблицями наведено на рис. 2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  | | | | |

Рис.11.2. Взаємозв’язок між таблицями введення-виведення.

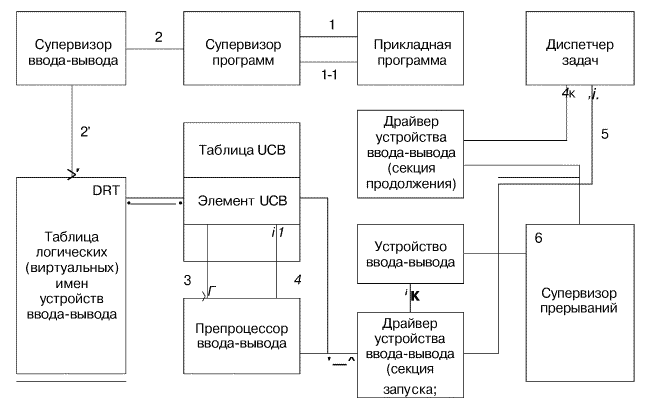


Рис.11.3. Процес керування введенням-виведенням

Запит на операцію введення - виведення від прикладної програми (1) надходить на супервізор. Він перевіряє системний виклик на відповідність специфікації та у випадку похибки повертає задачі відповідне повідомлення (1-1). Якщо запит конкретний, то він відправляється на супервізор введення - виведення (2). Останній за логічним ім’ям за допомогою таблиці логічних імен знаходить елемент UCB у таблиці обладнання. Якщо пристрій зайнятий, то описувач задачі, запит якої зараз опрацьовує супервізор введення - виведення, включається у список задач, що очікують даний пристрій введення - виведення. Якщо пристрій введення - виведення вільний, супервізор введення - виведення визначає з UCB тип пристрою та при необхідності запускає препроцесор, за допомогою якого отримується послідовність управляючих кодів та даних, яку може правильно сприйняти та відпрацювати пристрій (3). Коли програма управління операцією введення - виведення буде готова, супервізор введення - виведення передає керування відповідному драйверу на секцію запуску (4). Драйвер ініціалізує операцію керування, обнуляє лічильник тайм-аута і повертає керування супервізору (диспетчеру задач) для того, щоб він встановив на процесор готову для виконання задач (5). Система працює своєю чередою, але коли пристрій введення - виведення відпрацює відправлену йому команду, воно виставить сигнал запиту преривання, по якому через таблицю переривань управління буде передано на секцію продовження драйверу (6). Отримавши нову команду, пристрій знову починає її опрацьовувати, а керування процесом передається диспетчеру задач і процесор продовжує корисну роботу.

Тобто виконується паралельне виконання процесів на фоні виконання введення - виведення.

## Синхронне та асинхронне введення - виведення

Задача, що видала запит на операцію введення - виведення переводиться супервізором у стан очікування завершення цієї операції. Коли супервізор отримує сигнал від секції завершення, що операцію завершено, він переводить задачу (процес) у стан готовності до виконання і їй може бути надано процесор. Це так зване синхронне введення - виведення. Воно використовується у більшості ОС. Для того, щоб збільшити швидкодію виконання процесів використовують асинхронний принцип введення - виведення.

Найпростіша реалізація цього принципу полягає у тому, що виведення буферизується. Тобто інформація для виведення передається не на пристрій безпосередньо, а у спеціальний системний буфер. У цьому випадку логічна операція виведення для процесу вважається зразу виконаною і процес (задача) може не чекати його фактичного виконання. Процесом реального виведення у цьому разі із системного буфера займається супервізор введення - виведення.

Виділенням буферу із системної області пам’яті займається спеціальний системний процес за вказівкою супервізора введення - виведення.

У результаті для останнього випадку виведення буде асинхронним, якщо у запиті на введення - виведення була вказівка на необхідність буферизації та відповідний пристрій введення - виведення допускає такі асинхронні операції, про які вказано у таблиці UCB.

Введення буферизації як засобу інформаційної взаємодії висуває додаткову проблему керування цими системними буферами. Вона вирішується засобами супервізорної частини ОС. При цьому супервізор вирішує не тільки задачі по виділенню та звільненню буферів у системній області пам'яті, але й задачі синхронізації процесів по заповненню та звільненню буферів, а також побудову черг при відсутності вільних буферів. Як правило супервізор введення - виведення при цьому використовує стандартні засоби синхронізації, які використовує ОС для вирішення інших задач синхронізації.

## Керування та буферизація при роботі з магнітними дисками

Середня швидкість процесора з оперативною пам'яттю на 2-3 порядки вища ніж середня швидкість передачі даних із зовнішньої пам'яті на магнітних дисках в оперативну пам'ять. Для подолання такої невідповідності у продуктивності використовується буферизація та кешування даних.

Найпростіший варіант прискорення дискових операцій читання даних – використання подвійної буферизації. При цьому доки в один буфер записується інформація, із другого вона читається та передається по запиту. Аналогічно при записі. Буферизація використовується практично в усіх операційних системах.

Кешування дуже корисне у тому випадку, коли програма (процес) багаторазово читає з диску одні і тіж самі дані. Після того як вони один раз були розміщені у кеші, звернень до диску більше не треба і швидкодія суттєво підвищується.

У даному випадку під кешем можна розуміти деякий пул буферів, який керується системним процесом. Якщо ми запитуємо деяку кількість секторів, то відповідна інформація, проходячи через кеш, там і залишається. Якщо буде потрібне повторне читання, то дані можуть бути прочитані безпосередньо з оперативної пам'яті без звертання до диску.

Кількість буферів, що складають кеш, обмежена, тому виникають ситуації, коли заново прочитані або записані сектори мають бути замінені у цих буферах. Власне у принципі робота такої кеш-пам'яті не відрізняється від роботи звичайної кеш-пам'яті., яку ми розглядали раніше. Власне тут процес кешування, як ми і казали раніше – це спосіб сумісного використання двох запам’ятовуючих пристроїв із різною швидкодією.

Тому точно, як і зазвичай, кешування дискових операцій може бути покращено за рахунок техніки упередженого читання, коли з диску зчитається значно більша кількість даних, ніж тих, що конкретно запросила операція.

У ряді ОС є можливість вказати у явному вигляді параметри кешування. У деяких – за це відповідає сама ОС. Так при використанні Windows NT такої можливості немає, а у Windows 95/98 є. Можна вказати обсяг пам'яті, що відводиться для кешування та об’єм порції даних (буфер або chunk), з яких набирається кеш. У файлі System.ini у секції [VCACHE] можна прописати, наприклад:

[VCACHE]

Min FileCache = 4096

Max FileCache = 32768

ChunkSize = 512

Мінімальний обсяг кешу – 4 Мбайт, макс. Обсяг – 32 Мбайт, об’єм буфера, яким оперує менеджер кешу дорівнює об’єму одного сектора, тобто 513 байт.

У мультипрограмних системах при виконанні багатьох задач запити на читання та запис даних ідуть таким потоком, що створюються черги. Якщо обслуговувати такі черги у порядку постановки у чергу, то завдяки випадковому характеру звертань до тих чи інших секторів мають місце великі втрату часу на пошук. Тому, враховуючи те, що переупорядкування черги з метою зменшення непродуктивних втрат часу можливо використовувати відносно швидко, можна запропонувати деякі дисципліни обслуговування таких черг з метою економії непродуктивного використання часу. Наведемо деякі із запропонованих дисциплін.

SSTF (Shortest seek time-first) найменший час пошуку – перший. У відповідності з цією дисципліною задовольняється запит, що потребує мінімальної кількості кроків для перепозиціонування запитуючих голівок на диску.

SCAN (сканування). Згідно з цією дисципліною голівки зчитувача переміщуються то в одному, то в другому напряму, обслуговуючи на своєму шляху усіх, хто виставив вимоги і потрапляє на такий шлях.

NEXT-Step- Scan – відміна від попередньої дисципліни у тому, що обслуговуються тільки ті запити, які вже існували на момент початку переміщення.

C-Scan (циклічне обслуговування). Відповідно до цієї дисципліни головки переміщуються циклічно з зовнішньої до внутрішньої поверхонь, обслуговуючи усі запити, що трапляються на шляху такого переміщення.

## Організація доступу до зовнішних пристроїв.

З точки зору центрального процесора зовнішні пристрої – це набір спеціалізованих комірок пам’яті або пристрої є наборами спеціалізованих елементів пам'яті або, якщо завгодно, регістрів. У процесорів загального призначення регістри пристроїв підключаються до адресних шин та шини даних. Пристрій має адресний дешифратор. Якщо виставлена на шині адреса відповідає адресі одного з регістрів пристрою, дешифратор підключає відповідний регістр до шини даних (рис. 4). Таким чином, регістри пристрою отримують адреси у фізичному адресному просторі процесора.

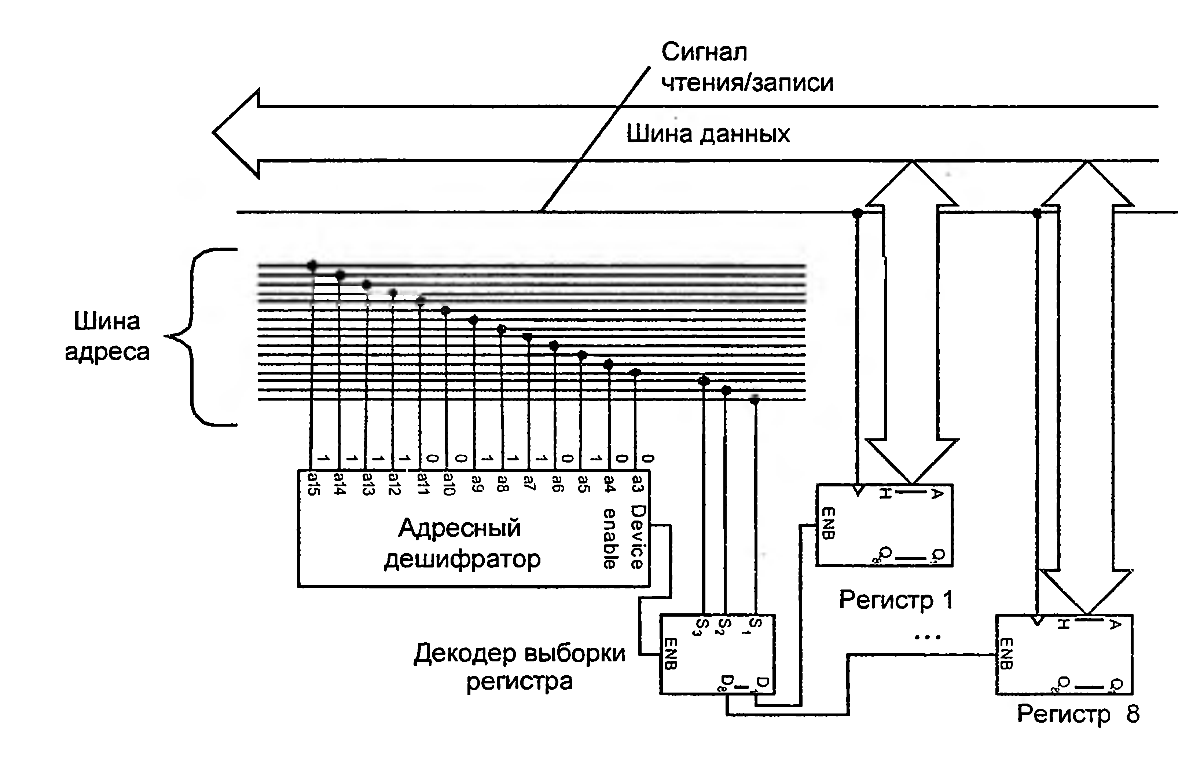


Рис. 11.4. Підключення зовнішнього пристрою до шини

Таким чином регістри отримують адреси у фізичному адресному просторі.

Існує два підходи до адресації цих регістрів:

- окремий адресний простір введення – виведення;

- відображення у пам’яті введення - виведення (memory maped I/О).

У останньому випадку пам’ять та регістри зовнішних пристроїв розміщені у одному адресному просторі.

У першому випадку для звернення до регістрів пристроїв використовуються спеціальні команди, наприклад, in та out.

У другому випадку можна використовувати які завгодно команди, що дозволяють працювати з операндами у пам’яті.

Як правило, навіть у випадку різних адресних просторів, для обміну даними між пам’яттю та зовнішними пристроями процессор використовує одні й тіж самі шини адрес та дані, але має додатковий сигнал шини адрес, що вказує на те, який з адресних просторів використовується у конкретному випадку.

Існує два основні підходи до виділення адрес зовнішнім пристроям — це *фіксована адресація,* коли один і той самий пристрій завжди має одні і ті ж адреси регістрів, і *географічна адресація,* коли кожному роз'єднувачу периферійної (або системної шини) відповідає свій діапазон адрес.

Географічно можна розподіляти не лише адреси регістрів, але і інші ресурси — лінії запиту переривань, канали прямого доступу до пам’яті, тощо.

Проте цей спосіб розподілу адресного простору зручний тим, що унеможливлює конфлікт адрес між пристроями різних виробників або між двома однотипними пристроями (з цією проблемою має бути знайомий кожен, хто намагався одночасно встановити в комп'ютер мережеву і звукову карти конструктиву ISA). Більшість*периферійних шин* сучасних міні- і мікрокомп'ютерів, такі, як PCI, S-Bus і ін., реалізують географічну адресацію.

Багато сучасних конструктивів вимагають, аби окрім регістрів управління і даних, пристрої мали також конфігураційні регістри, через звернення до яких ОС може отримати інформацію про пристрій: фірму-виробника, модель, версію, кількість регістрів тощо. Наявність таких регістрів дозволяє ОС без втручання (або з мінімальним втручанням) з боку адміністратора визначити встановлене в системі устаткування і автоматично підвантажити відповідні модулі управління.

## Порти введення-виведення.

Цей пристрій – стандартний компонент більшості мікропроцесорних систем.

**Порт виведення** – представляє собою регістр та входи і виходи. Кількість виходів порту відповідає кількості біт регістру. Нулю у розряді відповідає низький рівень напруги (скажімо 0), а одиниці – високий рівень (скажімо напруги блоку живлення).

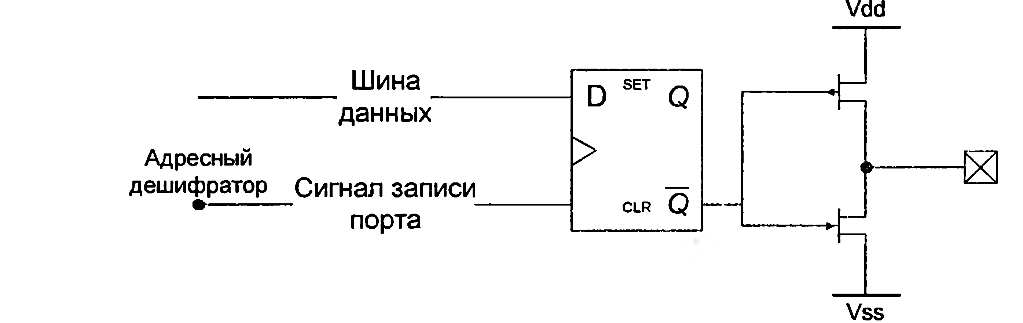


Рис.11.5 Структура порту виведення

**Порт введення** теж складається з регістру та декількох вхідних ліній, що відповідають бітам регістра. Біт регістра має значення 0, коли на вхід подають низьку напругу і, навпаки, одиниці – високу. Регістр зберігає прийнятий код.

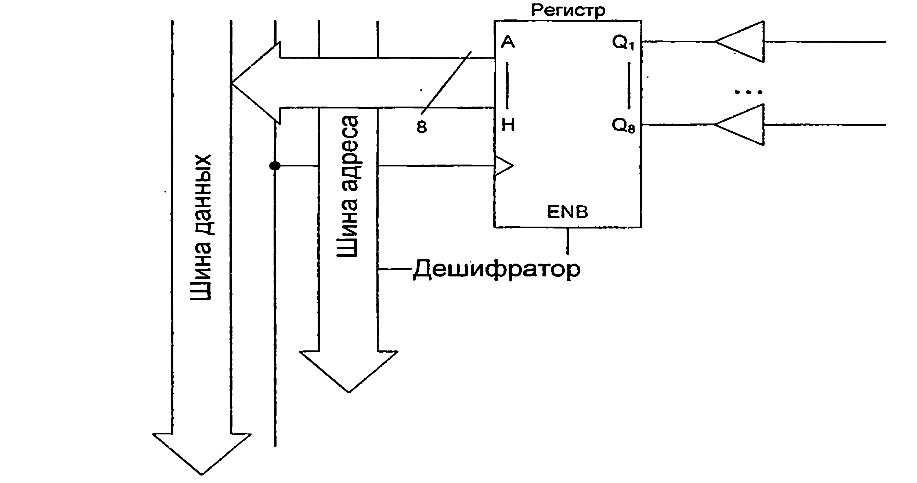


Рис.11.6. Структура порту введення.

Розробники мікропроцесорів часто роблять сумісними порти введення - виведення.

Головна проблема використання простих портів полягає у тому, що приймаючий пристрій має знати чи передавач виставив на своїх проводах(дротах?) нову порцію даних чи ні. Для рішення цієї проблеми є три підходи:

- синхронна передача даних,

- асинхронна передача даних,

- ізохорна передача.

При **синхронній передачі** ми або передаємо додатковий сигнал – строб, або передаємо синхросигнали по тим же лініям передачі сигналу, що і дані.

Передача стробу потребує прокладки додаткових проводів, але можливо суміщення синхросигналу та даних і тому це широко використовується.

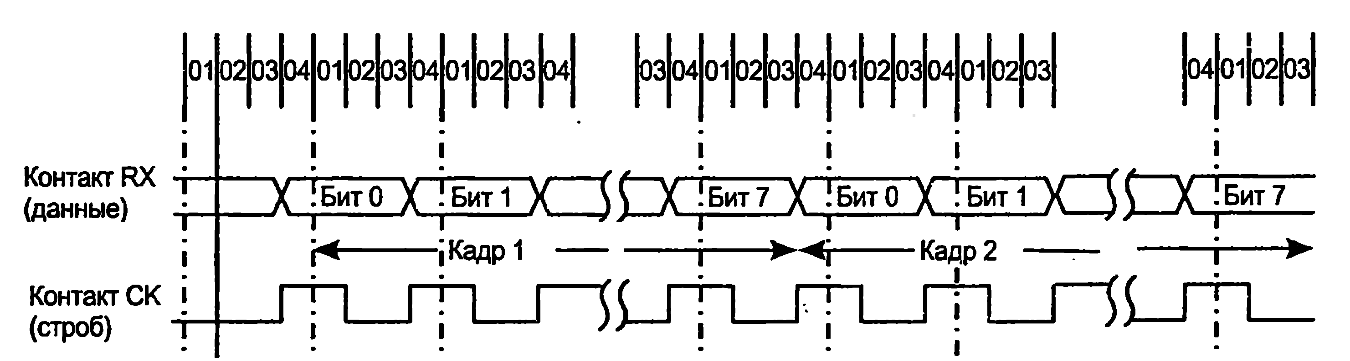


Рис.11.7. Часова діаграма порта зі стробом

Більшість портів із стробом асиметричні : оне з пристроїв **ведуче** (master – хазяїн), генерує стробовий сигнал, а друге **ведоме** (slave – раб) користується цим сигналом для прийому чи передачі.

Часто замість одиничного стробу використовується декілька різних сигналів. Наприклад, один виставляється передавачем та сповіщає, що чергова порція даних готова, а другий сигнал – приймачем, сповіщає, що приймач прийняв ці дані та готовий прийняти чергові дані.

Додаткові сигнали також можуть вирішити ці питання про те, яке з пристроїв у даному циклі буде приймачем, а яке передавачем.

При **асинхронному** обміні даними пристрій – передавач пересилає стартовий силавол, сигналізуючий про те, що зараз підуть дані та із фіксованим інтервалом виставляє на своїх виходах символи даних. Переданий за один раз блок даних звичайно невеликий за обсягом – з причин можливої небезпеки того, що годинники приймача та передавача, за якими вони відміряють свої інтервали між черговими порціями даних, неточні та можуть розбігтися.

Зазвичай блок даних складається із фіксованої кількості символів і називається кадром або фреймом (frame). Кадр завершується одним або декількома стоповими символами. Не знайшовши цих символів приймач може зрозуміти, що його годинник розійшовся з годинником передавача.

Асинхронна передача дозволяє економити на проводах за рахунок стробуючих сигналів та при цьому запобігти складних способів кодування, характерних для суміщеної синхронної передачі. Однак стартові та стопові символи складають значну частку потоку даних, що значно збільшує накладні витрати.

Окрім цього при передачі великого об’єму даних, у вигляді тісно слідуючих один за одним кадрів, виникає небезпека, що приймач загубить заголовок чергового кадру та не зможе поновнити структуру потоку.

Щоб запобігти цьому багато асинхронних протоколів потребують паузи між сусідніми кадрами.

Асинхронна передача кадрів доцільна у ситуаціях, коли обсяги передачі невеликі, а потреба у їх передачі виникає у випадкові, непередбачувані моменти часу. Зазвичай, асинхронні порти працюють на невисоких швидкостях, не більше декількох кілобіт на секунду.

**Ізохорна передача** даних нагадує асинхронну з тією різницею, що при обміні даними приймач і передавач користуються високо стабільними, незалежними тактовими генераторами. Тому вони можуть обмінюватися кадрами великого розміру. В ідеалі ізохорна передача дуже якісна, але дуже складно забезпечити стабільність та калібровку тактових генераторів, тому ізохорна передача використовується досить рідко.

Як синхронні, так і асинхронні порти бувають наступних типів:

- симплексні (simplex – передавати дані може тільки один пристрій);

- напівдуплексні (half-duplex – обидва пристрої можуть приймати та передавати дані, але неспроможні робити це одночасно, наприклад, тому, що прийом та передача використовують один провід);

- повнодуплексні (full-duplex), або просто дуплексні (обидва пристрої можуть одночасно передавати та приймати дані, найчастіше по різним проводам).

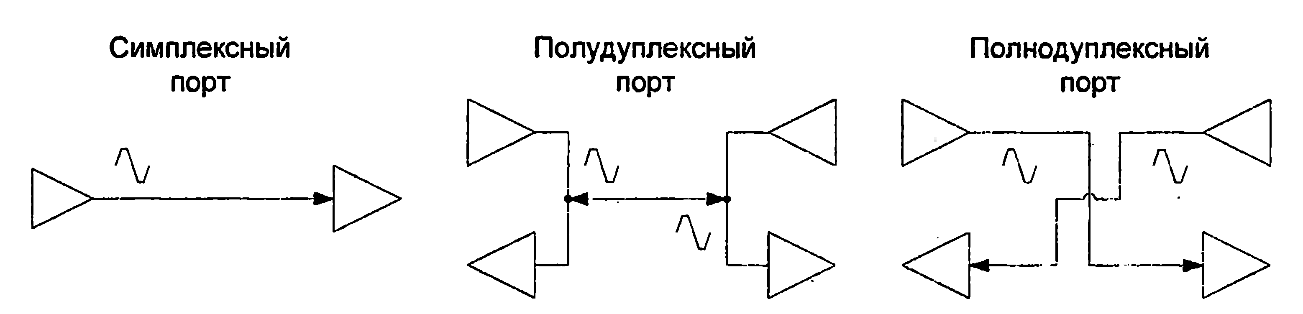


Рис.11.8. Різновиди портів.

Ще одна класифікація портів передачі даних:

* послідовні порти;
* паралельні порти.

**Послідовний** порт має один провід, по якому послідовно передаються біти даних, а також можливо синхронізуючі або стартові стопові біти.

**Паралельний порт** має декілька ліній передачі даних, як правило 8, для передачі одного байта, а іноді і більше.

Як правило, послідовні порти – асинхронні або синхронні із суміщеною передачею синхросигналу.

Паралельні порти – синхронні, стробіруємі(стробуємі?). До восьми шин додається ще одна – дев`ята для передачі строба. Тож додаткові витрати у процентному відношенні невеликі у цьому випадку.

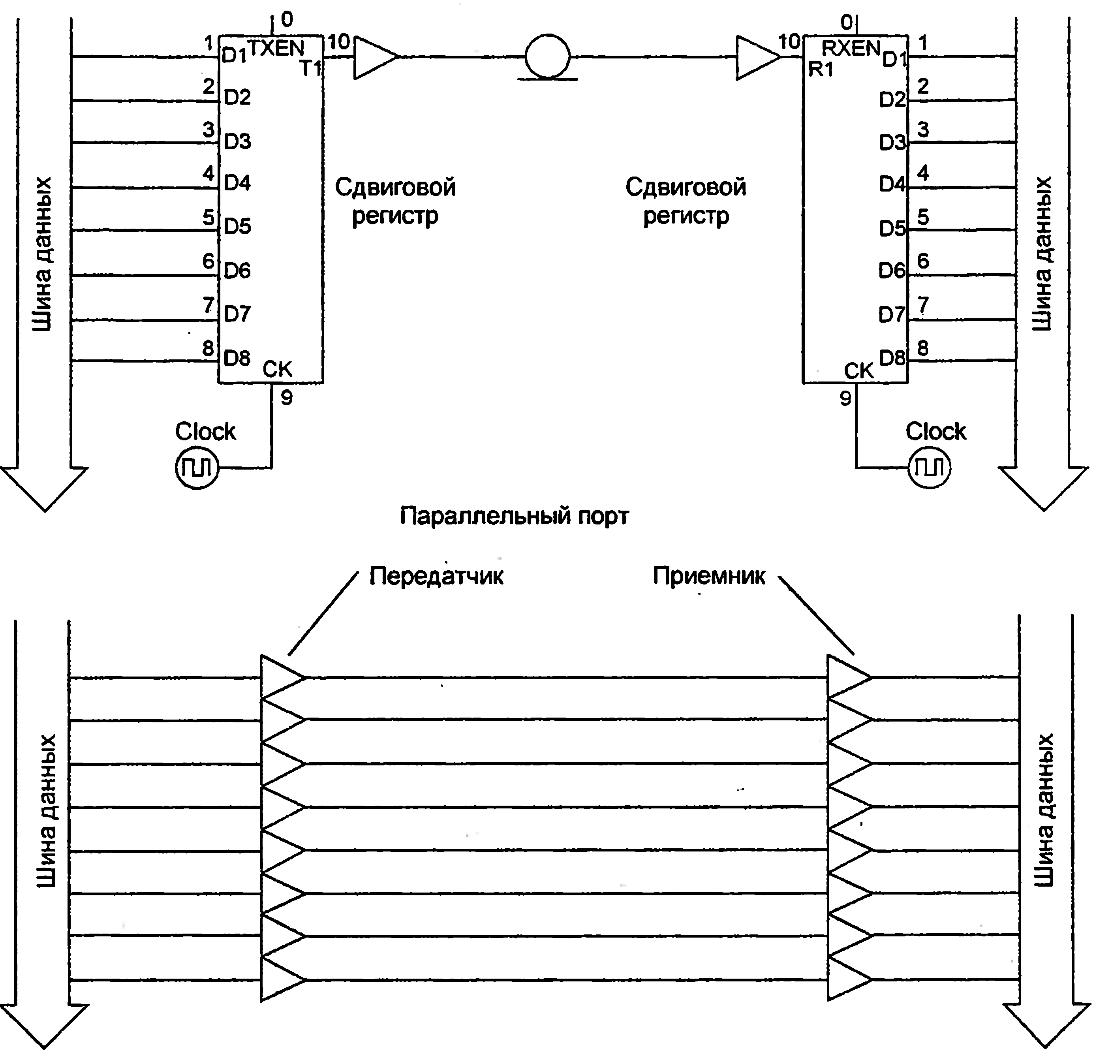


Рис. 11.9 Послідовний порт.

Розглянемо порт RS232, який широко використовується для передачі даних за границі корпусу комп`ютера. RS232 використовує у якості «1» напругу у діапазоні від -25 до -3 В, а в якості «0», відповідно, від +3 до +25В. RS232 передбачає двосторонній обмін даними. Для цього передбачені дві лінії даних – для прийому та передачі, які зазначені Тх та Rx.

У відповідності із стандартом, пристрої поділяються на два типи: «комп`ютери» та «термінали». Різниця між ними полягає у тому, що «комп`ютер» передає дані по лінії Тх, а отримує їх по Rx. «Термінали» - навпаки. Для поєднання двох комп`ютерів необхідно мати спеціальний, так званий нуль - модемний кабель, у якому проводи Тх та Rx перехрещені.

Обмін даними виконується кадрами, що складаються із стартового біта, семи чи восьми бітів даних (молодший біт передається першим), можливий контрольний біт парності та одного чи двох стопових бітів.

Мінімальна швидкість передачі 300 біт/с, максимальна – більше 115200 біт/с. Швидкість та варіації формату кадру визначаються налаштуваннями приймача та передавача. Необхідно, щоб у поєднаних портом пристроїв ці налаштування співпадали.

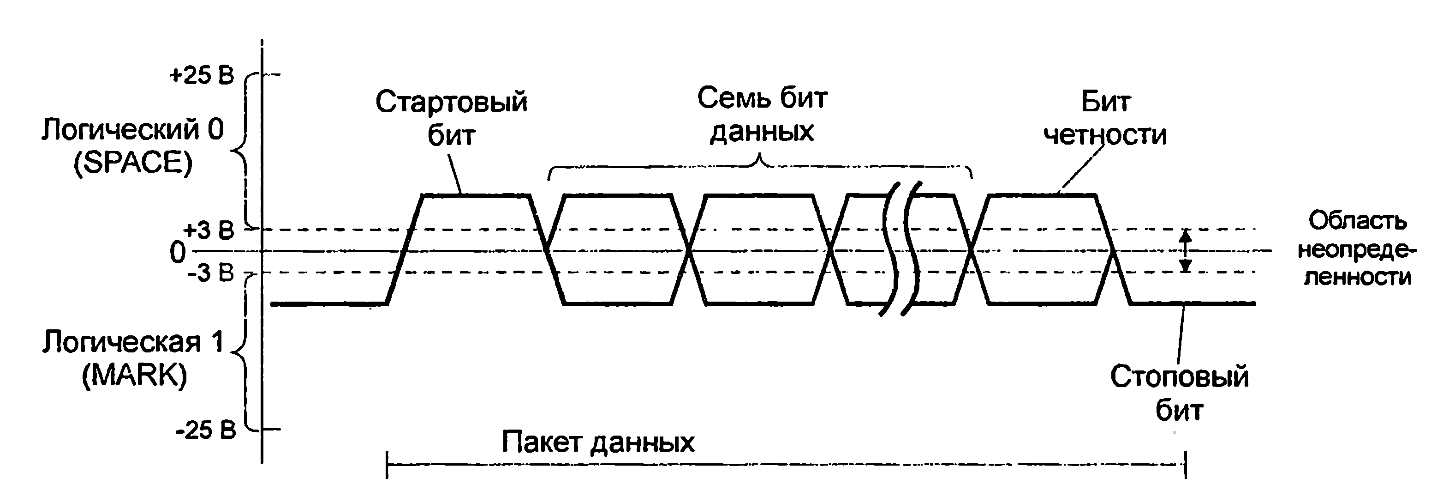


Рис.11. 11. Діаграма напруг

## Шини

Часто виникає потреба у підключенні до одного порту передачі даних декількох пристроїв. При цьому кожна порція переданих даних має супроводжуватись вказівкою, до якого з підключених пристроїв вона призначена – адресою або селектором пристрою.

Такі, багатоточкові порти звуться **шинами** (bus).

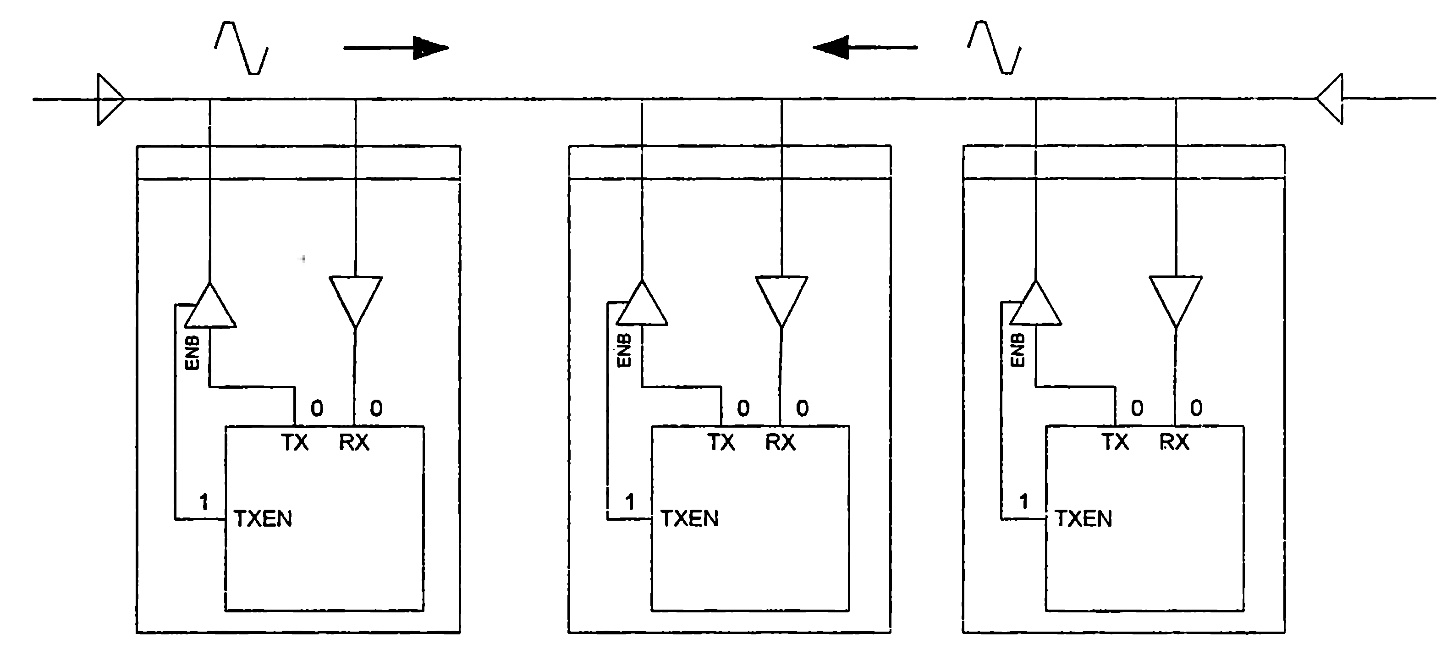


Рис.11.10. Шина

Як і двоточкові порти, шини бувають синхронні та асинхронні, а також послідовні та паралельні. Однак, терміни синхронний та асинхронний використовуються у іншому визначенні, ніж при описі портів.

Асинхронними називають шини, у яких відомий пристрій не виставляє сигнали завершення операції, а синхронними, відповідно, шини, де відомий зобов`язаний це робити.

Підключення N пристроїв двоточковими портами вимагає, щоб центральний процесор мав N прийомопередатчиків.

Використання багатоточкового порту дозволяє обходитись одним. За рахунок цього виникає можливість зменшити кількість виходів мікросхеми процесора або переферійного контролера. Окрім цього, при вдалому розміщенні пристроїв можна отримати значний виграш у загальній кількості проводів та зменшити кількість доріжок на друкованій платі. У багатьох випадках це призводить до здешевлення апаратури, хоча і ускладнює протокол передачі даних.

Основний недолік шини полягає у тому, що тільки один пристрій на шині може передавати інформаціюу кожен момент часу. Якщо у двоточкових портах часто доцільно реалізувати повнодуплексний обмін даними за допомогою двох комплектів ліній (один на прийом, інший – на передачу, як у випадку RS232), то у випадку шинної топології це неможливо. Тому шини бувають тільки **напівдуплексн**і або **симплексні.**

Неможливість паралельно виконувати обмін з двома пристроями може призвести до падіння продуктивності порівняно з власним портом обміну у кожного пристрою. Але якщо пристрої не займають пропускну здатність каналів передачі даних повністю, програш виявляється не таким великим. Тому шини широко використовують навіть тоді, коли один пристрій має можливість ініціювати обмін даними.

Якщо передачу даних можуть ініціювати декілька пристроїв (наприклад, системна шина з декількома процесорами), шинна технологія найбільш прийнятна. Така технологія, однак, потребує вирішення однієї проблеми: забезпечення арбітражного доступу до шини з боку можливих ініціаторів обміну.

Методи такого арбітражу відрізняються великою різноманітністю.

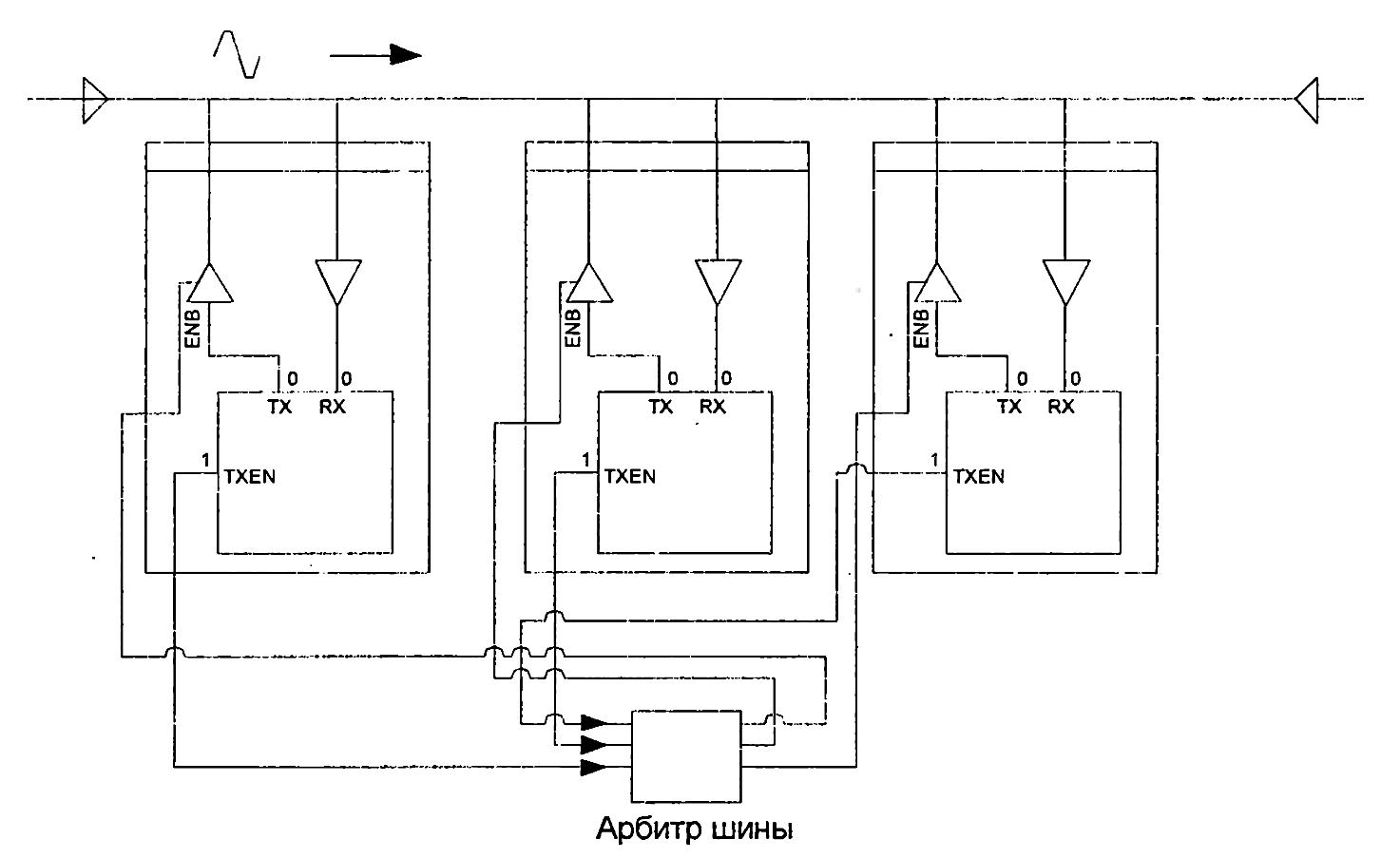


Рис11.12. Шина з декількома за датчиками

Необхідність визначення колізій та арбітражу накладає обмеження на фізичні розміри шини. Сигнали розповсюджуються по шині з кінцевою швидкістю, а будь-які два конфліктуючі пристрої мають дізнатися про колізію за час, що менше, ніж мінімальний цикл передачі даних. Для внутрішніх шин мікропроцесора ця проблема може бути неактуальною, однак, для локальних мереж, довжина яких може становити метри і кілометри, чи для системних шин комп`ютерів з великою кількістю процесорів та банків пам`яті – це серьйозна проблема.

Одним з основних шляхів подалання цієї проблеми – заміна шини центральним комутатором або системою комутаторів. Пристрої з`єднуються з найближчим комутатором повнодуплексним двоточковим каналом, канали також поєднують комутатори один з одним, а усі колізії виникають та вирішуються тільки усередені комутаторів.

Внутрішня шина комутатора має велику пропускну здатність (значно більшу, ніж зовнішні з`єднання), окрім того, комутатори, як правило, мають декілька внутрішніх шин. Тому вирішення колізій всередині комутатора часто зводиться до відправки даних іншим шляхом (Див. Рис. 13).

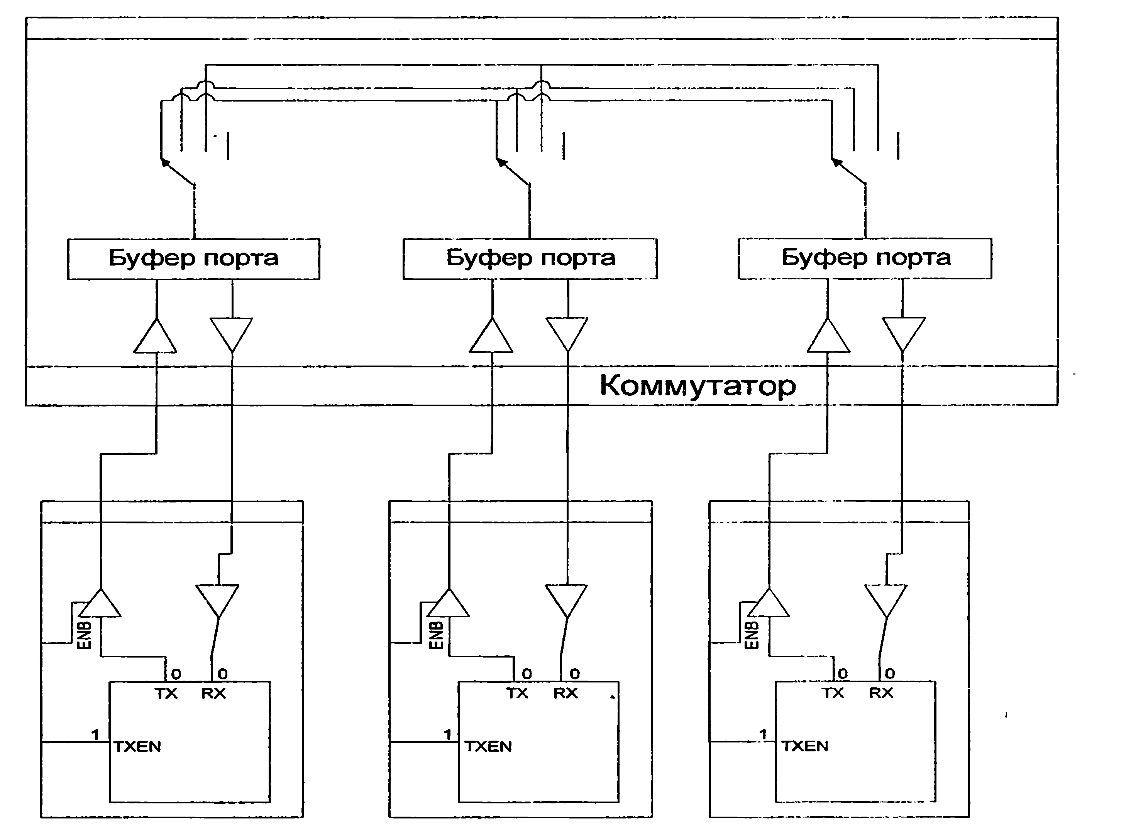


Рис. 11.13. Комутатор з декількома внутрішніми шинами.

Таким чином, магістраль, що комутується, дозволяє зменшити втрати продуктивності, що виникає завдяки колізіям, та підвищити реальну пропускну здатність магістралей. Однак, при цьому суттєво збільшується загальна вартість системи, що далеко не завжди може бути виправдано.

У обчислювальних системах загального призначення часто використовується топологія з двома або більше шинами. Процесори та пам`ять зв`язані системною шиною. До цієї шини також прив`язують декілька адаптерів периферійних шин, до яких підключають зовнішні пристрої. Головні переваги такого рішення полягають у тому, що одні і тіж периферійні пристрої можуть бути використані у обчислювальних системах з різними системними шинами.

Більш детальна інформація по темі зовнішніх пристроїв, каналів зв`язку та драйверів зовнішніх пристроїв може бути отримана із рекомендованої літератури.

## Організація паралельної роботи пристроїв введення - виведення й процесора.

Кожен пристрій введення - виведення постачається блоком керування - контролером. Контролер взаємодіє із драйвером - системним програмним модулем, керуючим даним пристроєм.

Контролер періодично приймає від драйвера виведену на пристрій інформацію й команди керування, які визначають, що робити із цією інформацією.

Пристрій введення - виведення працює під управлінням контролера в інтервалах між видачею команд незалежно від ОС.

Від підсистем введення - виведення потребується спланувати в реальному масштабі часу запуск і припинення драйверів, забезпечивши прийнятний час реакції кожного драйвера на незалежні події контролера.

Для цього всі драйвери розподіляються по декількох пріоритетних рівнях відповідно до вимог за часом реакції й тимчасових витрат процесорного часу.

Для реалізації цього процесу звичайно використовується диспетчер переривань ОС.

## Узгодження швидкостей обміну й кешування даних

При обміні даними виникає задача узгодження швидкостей. Вона досягається за рахунок буферизації даних в ОП і синхронізації доступу процесів до буфера.

Але буферизація тільки на основі ОП у підсистемі введення - виведення виявляється недостатньою - різниця між швидкістю обміну з ОП і швидкістю роботи зовнішнього пристрою виявляється занадто великою. З іншого боку, при великих обсягах введення - виведення ОП просто може не вистачити.

Для таких випадків як буфер використовується spool (шпулька). Типовий приклад - виведення на принтер. Друк документа в кілька десятків мегабайт - не рідкість.

Інше рішення проблеми - більш буферна пам'ять у контролерах зовнішніх пристроїв. Приклад - контролери графічних дисплеїв. Їх ОП порівняна з ОП процесора.

## Розподіл пристроїв і даних між процесами

Пристрої введення - виведення можуть надаватися процесам, як у монопольному, так й у поділюваному режимах.

ОС повинна забезпечувати контроль доступу тими ж способами, що й при доступі процесів до інших ресурсів ВР.

ОС може контролювати доступ не тільки до пристрою в цілому, але й до **окремих порцій даних**. Наприклад, при виводі на графічний дисплей - інформація з окремих вікон екрана.

Тому для організації спільного доступу до частин пристрою або частин даних, неодмінною умовою є завдання режиму спільного використання пристроїв або даних у цілому.

ОС надає пристрою, відслідковуючи процедури захвату й звільнення використання пристроїв, оптимізуючи послідовність операцій введення - виведення для різних процесів з метою підвищення загальної продуктивності, якщо це можливо.

Наприклад, при обміні даними декількох процесів з дисками можна впорядкувати послідовність операцій так.

## Забезпечення зручного логічного інтерфейсу між пристроями й іншою частиною системи

Всі сучасні ОС підтримують як основу такого інтерфейсу файлову модель периферійних пристроїв. При такому підході будь-який пристрій виглядає як набір байт, з якими можна працювати за допомогою уніфікованих системних викликів (наприклад, read, write), задаючи ім'я файлу - пристрою й зсув від початку послідовності байт.

Привабливість такої моделі заключається в її простоті й уніфікованості для пристроїв різного типу.

Іноді для спеціальних застосувань цей інтерфейс використовується як базовий і вимагає додаткової доробки. Наприклад, при програмуванні операцій мережного обміну або виведенні на дисплей графічної інформації.

## Підтримка широкого спектру драйверів і простота включення нового драйвера в систему

Наявність різноманітних драйверів для всіх типів зовнішніх пристроїв є важливою характеристикою ОС. Наприклад, така прекрасна в багатьох відносинах ОС як ОС/2 була витиснута ОС Windows завдяки багатству драйверів.

Тому відкритість інтерфейсу драйверів, тобто доступність його опису для незалежних розроблювачів ПО - необхідне

Драйвер взаємодіє, з однієї сторони з модулями ядра ОП (підсистемою введення - виведення, системними викликами, керування процесами й пам'яттю), з іншого боку, з контролерами зовнішніх пристроїв.

Тому існує 2 типи інтерфейсів:

* драйвер - ядро;
* драйвер - пристрій.

Інтерфейс драйвер - ядро повинен бути стандартизований у кожному разі.

Інтерфейс драйвер - пристрій має сенс стандартизувати тоді, коли підсистема введення - виведення не дозволяє драйверу безпосередньо взаємодіяти із апаратурами.

Екранування драйвера від апаратур є досить корисною функцією, тому що драйвер стає незалежним від апаратної платформи.

Для підтримки процесу розробки драйверів до ОС звичайно випускається пакет ДДК (Diver Development Kit), що представляє собою інструментарій - бібліотеки, компілятори й відладчики.

## Динамічне завантаження й вивантаження драйверів

Включення драйвера до складу модулів працюючої ОС являє собою самостійну проблему.

Тому що набір потенційно підтримуваних даною ОС периферійних пристроїв завжди ширше конкретної системи, то дуже цінною властивістю ОС є можливість динамічно завантажувати в ОП необхідний драйвер (без зупинки ОС) і вивантажувати його після того, як потреба в підтримці пристрою минула. Це може істотно заощаджувати системну область пам'яті.

Альтернативою динамічному завантаженню драйверів при змінах поточної конфігурації зовнішніх пристроїв є повторна компіляція коду ядра з необхідним набором драйверів, що створює між всіма компонентами ядра статичні зв'язки замість динамічних. Зміни в процесі роботи ОС неможливі.

**Підтримка декількох файлових систем**

Диски - особливий рід периферійних пристроїв, тому що на них зберігається більша частина користувальницьких і системних даних. Ці дані організуються у файлові системи, властивості яких багато в чому визначають властивості ОС (відмовостійкість, швидкодія, макс. обсяг збережених даних). Гарна файлова система звичайно «кочує» з однієї ОС в іншу.

Так файлова система FAT спочатку була розроблена для MS-DOS, потім перекочувала в OS/2 й MS Windows (3.1).

Важливо, щоб архітектура підсистеми введення - виведення дозволяла досить просто включати в її склад нові типи файлових систем без необхідності переписування коду.

Для цього в ОС передбачається спеціальний шар ПО, що відповідає за рішення даного завдання. Наприклад, шар VFS (Virtual File System) у версіях UNIX.

## Підтримка синхронних й асинхронних операцій введення - виведення

Операції введення - виведення можуть виконуватися в синхронному й асинхронному режимах.

Синхронний режим означає, що процес, який запросив операцію, припиняє свою роботу до тих пір, поки операція введення - виведення не завершиться.

При асинхронному режимі процес виконується в мультипрограмному режимі одночасно з операцією введення - виведення.

Підсистема введення - виведення надає своїм клієнтам (користувальницьким процесам й ядру ОС) можливість виконувати як синхронні, так й асинхронні операції введення - виведення залежно від потреб визивної сторони.

Системні виклики введення - виведення частіше оформлюються як синхронні процедури у зв'язку з тим, що такі операції тривають довго й користувальницькому процесу або потоку однаково прийдеться чекати результатів, щоб продовжити роботу.

Внутрішні виклики з модулів ядра ОС виконуються як асинхронні процедури, тому що ядру потрібна свобода у виборі подальшого поводження після запиту операції введення - виведення.

## Багатошарова модель підсистеми введення - виведення

При великій розмаїтості пристроїв введення - виведення, що досить відрізняються, дозволяє дотриматись балансу між двома суперечливими вимогами:

* необхідністю обліку всіх особливостей кожного пристрою;
* єдине логічне подання й уніфікований інтерфейс для пристроїв всіх типів.

Нижні шари підсистеми введення - виведення повинні включати індивідуальні драйвера конкретних фізичних пристроїв, а верхні шари повинні узагальнювати процедури керування цими пристроями, надаючи, по можливості, загальний інтерфейс.

## Менеджер введення - виведення

Модулі підсистеми введення - виведення, які організують злагоджену роботу всіх компонентів підсистеми й взаємодію з користувальницькими процесами й іншими підсистемами ОС утворюють як би оболонку підсистеми.

Ця оболонка називається менеджером введення - виведення.

**Верхній шар менеджера** підтримує **користувальницький інтерфейс введення - виведення**, тобто приймає запити на введення - виведення і переадресує їхнім драйверам, а також повертає процесам результати операцій введення - виведення.

**Нижній шар** взаємодіє з контролерами зовнішніх пристроїв, екрануючи драйвери від особливостей апарату введення - виведення, системи переривань тощо. Цей шар приймає від драйверів запити на обмін даними з регістрами контролерів у деякій узагальненій формі з користуванням незалежних від шини введення - виведення адресації й формату, перетворюючи ці запити у формат, зрозумілий апаратній платформі.

Ще однією функцією менеджера введення - виведення є організація взаємодії модулів введення - виведення з модулями інших підсистем ОС, таких як підсистеми керування процесами, віртуальною пам'яттю та ін.

Наявність стандартного віртуального міжмодульного інтерфейсу підвищує стійкість і поліпшує розширюваність підсистеми введення - виведення, хоча й сповільнює її роботу.

## Багаторівневі драйвери.

Традиційні особливості й функції, виконувані драйвером полягають у наступному:

* входить до складу ядра ОС, працюючи в привілейованому режимі;
* безпосередньо управляє зовнішнім пристроєм, взаємодіючи з його контролером за допомогою команд введення - виведення комп'ютера;
* обробляє переривання від контролера;
* надає програмістові зручний логічний інтерфейс, екрануючи від непотрібних деталей і подробиць;
* взаємодіє з іншими модулями.

Традиційні драйвери не ділилися на шари. З розвитком ОС поряд із традиційними з'явилися **високорівневі драйвери**, які розташовуються над традиційними драйверами. При цьому традиційні драйвери стали називати апаратними. За допомогою **високорівневого** драйвера підвищується гнучкість і розширюваність функцій по керуванню пристроєм - замість жорсткого набору функцій, які зосереджені в низкорівневому драйвері, адміністратор ОС може вибрати необхідний набір функцій, установивши потрібний **високорівневий** драйвер.

Якщо різним додаткам необхідно працювати з різними логічними модулями того самого фізичного пристрою, то для цього досить установити в системі кілька драйверів на одному рівні, що працюють над одним апаратним драйвером.

Кількість рівнів драйверів звичайно не обмежується. На практиці використовується від 2 до 5 рівнів.

В**исокорівневі** драйвери оформляються по тим же правилам і дотримуються тих же внутрішніх інтерфейсів, що й апаратні драйвери за винятком того, що вони не викликаються по перериваннях, тому що взаємодіють.

Як загальні принципи побудови багаторівневих драйверів можуть бути реалізовані стосовно до конкретних пристроїв можна розглянути на прикладі керування дисками.

системні виклики

VFS

Прикладний програмний інтерфейс

Блок - орієнтований інтерфейс

VFS

Дисковий кеш

Драйвер

HD

Диспетчер переривання, функції доступу до апаратур

NTFS

FAT

Драйвер

FD

Рис.11.14. Високорівневі драйвери

Апаратні драйвери підтримують для верхніх рівнів подання диска як послідовного набору блоків однакового розміру, перетворюючи разом з контролером номер блоку в більш складну адресу з номере циліндра, головки й сектора. Поняття «файлу» і файлової системи апаратні драйвери не підтримують

Ці абстракції створюються на більш високому рівні . Вони можуть підтримувати кілька файлових систем одночасно. Для цього в ОС встановлюється декілька високорівневих драйверів (UFS, NTFS, FAT). Вони працюють із загальними апаратними драйверами, але по-своєму організовують файлову систему користувачам і прикладними процесам.

Для уніфікації подання різних файлових систем може використовуватися загальний драйвер верхнього рівня VFS (Virtual File System). Такий драйвер використовується, наприклад, у системах UNIX.

В уніфікацію драйверів великий внесок зробила ОС UNIX. У ній всі драйвери розділені на 2 великих класи; блок - орієнтовані й байт - орієнтовані.

Наприклад, драйвера графічних і мережних пристроїв ставляться до байт - орієнтованих, а драйвери, керуючими пристроями прямого доступу, які зберігають інформацію в блоках фіксованого розміру, - до блок - орієнтованих (диск).

Адресуємість блоків приводить до того, що для пристроїв прямого доступу є можливість кешування даних в оперативній пам'яті. Це впливає на загальну організацію введення - виведення таких пристроїв.

## Спеціальні файли

Спеціальні файли, які іноді називають наборами даних, що зберігаються на дисках. Вони використовуються для уніфікованого подання пристроїв введення - виведення .

Зі спеціальним файлом можна працювати так само, як і зі звичайним - відкривати, читати або записувати байти, а після завершення операції закривати.

Для цього використовуються ті ж системні виклики, що й для роботи зі звичайними файлами; open, create, read, write, close.

Традиційно спеціальні файли містяться в каталозі /dev, хоча ніщо не заважає створити їх у будь-якому каталозі файлової системи. З появою нового пристрою й, відповідно, нового драйвера адміністратор системи може створити новий запис (наприклад, за допомогою команди mknod).

# Файлові системи

## Логічна організація файлової системи

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

## Мета й завдання файлової системи

**Файл** – це іменована область зовнішньої пам'яті, в яку можна записувати і з якої можна зчитувати дані. Зазвичай файли зберігаються на енергонезалежній пам'яті - диску. Однак є й виключення. Наприклад:

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 - **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 - однокористувальницька, однопрограмна. До їх числа належить MS-DOS.

Завдання наступні:

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

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

У цьому випадку файл - загальнодоступний ресурс із усіма проблемами, що випливають звідси.

У багатокористувальницьких системах додається ще одне завдання; захист файлів від несанкціонованого доступу.

## Типи файлів

**Звичайні файли –** містять інформацію довільного характеру. Більшість сучасних ОС (UNIX, Windows, OS/2) ніяк не обмежують і не контролюють зміст 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, об’єднаних користувачами по якійсь неформальній ознаці (наприклад, один документ і т.п.).

У багатьох ОС до каталогу можуть входити:

**Спеціальні файли**  - це фіктивні файли, асоційовані з пристроями введення - виведення. Вони використовуються з метою уніфікації механізму доступу до файлів і зовнішніх пристроїв. Спеціальні файли дозволяють користувачу 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 у команди керування відповідним пристроєм.

## 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. 12.1.Дерево Рис. 12.2. Мережа

Графік описуючий ієрархію каталогів може бути деревом або мережею. Якщо файлу дозволено входити тільки в один каталог, файли утворять дерево. Мережа - файл може входити відразу в декілька каталогів.

Наприклад, в MS-DOS й Windows каталоги утворюють деревоподібну структуру, а в UNIX - мережну.

У деревоподібній структурі кожен **файл** є **листом.** Каталог найвищого рівня називається **кореневим** **каталогом** або **коренем**.

## Імена файлів

У файлових системах використовуються 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. У файловій системі FAT максимальна довжина імені обмежується схемою 8.3 (8 символів ім'я, 3-розширення), 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 показує шлях 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 існує відповідність: **один файл – багато повних імен.**

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

## 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 розміщується 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 називається **монтуванням.**

При цьому ОС виділяє один дисковий пристрій, називаний **системним**. Нехай є дві файлові системи, розташовані на різних логічних дисках, причому один з них є системним.

Файлова система, розташована на системному диску, призначається кореневій. 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 монтуєма файлова система приєднується, як піддерево до загального дерева.

Файлова система 1

dev

2

USER

man

local

2

Файлова система 2

man1

man2

Звичайний файл

USER

каталог

2

пристрій (спец.файл)

man1

man2

Загальна файлова система

dev

2

USER

man

local

2

Рис.12.3

## Атрибути файлів

Атрибути -інформація, що описує властивості файлу. Приклади можливих атрибутів:

* тип файлу (звичайний файл, каталог, спец.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 «схований 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.

зроблено в MS-DOS

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | | 3 | 1 | | | | 4 |
| Ім'я файлу | | Розширення | R | A | H | S | Резервні |
| Резервні | Час | Дата | Номер першого набору | | | | Розмір |

Рис.12.4

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.

|  |  |
| --- | --- |
| 2 | 14 |
| № індексного дескриптора | Ім'я файлу |

Рис.12.5. 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.

## 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 можливо на програму, або цю роботу може на себе брати файлова система.

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

Модель файлу, відповідно до якої вміст файлу представляється неструктурованою послідовністю (потоком) байт, широко використовується в 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 інтерпретувати дані, 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 того, щоб виділити будь-який запис. ФС надає додатку доступ до запису, а вся подальша обробка даних, що міститься в цьому записі, виконується програмою.

Розвитком цього

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 (позиціюється файл на конкретний запис)

## Фізична організація файлової системи

Подання про файлову систему, як про ієрархічно організовану множину інформаційних об'єктів має мало загального 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 знаходяться поряд 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 **пакета 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 диска. Коли диск обертається, головка зчитує дані з магнітної доріжки або записує їх на доріжку.

Головка може позиціонуватися над заданою доріжкою, переміщаючись над поверхнею диска дискретними кроками. Кожен крок відповідає зрушенню на 1 доріжку. У деяких дисках замість однієї головки є по головці на кожну доріжку.

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

Всі доріжки мають рівне число секторів, у яких можна максимально записати одне і те ж число байт.

Сектор має фіксований для конкретної системи розмір, що виражається ступенем двійки. Найчастіше - 512 байт. Оскільки доріжки різного радіуса мають однакове число секторів, щільність запису на доріжках різна - більша до центра.

**Сектор** – найменша адресуєма одиниця обміну даними з оперативною пам'яттю.

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

Тому що прикладній програмі потрібен не сектор, а деяка кількість байт, не обов'язково кратна розміру сектора. Типовий запит включає читання декількох секторів, які містять необхідну інформацію й одного або двох секторів з надлишковими даними.

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

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

Для визначення кордонів блоків на диск записується ідентифікаційна інформація.

**Низькорівневий формат** диска не залежить від типу ОС, котрий цей диск використовує.

Розмітку диска 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 кордони 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** – безперервна частина фізичного диска. Його ОС представляє як **логічний пристрій** (логічний диск).

Тому що файлова система, з якою працює одна ОС, не може в загальному випадку інтерпретуватися ОС іншого типу, логічні диски не можуть бути використані ОС різного типу.

На кожному **логічному диску** може створюватися тільки **одна файлова система.**

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

Всі розділи одного диска мають однаковий розмір низькорівневого форматування. Але при високорівневому форматуванні на 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 завантажуваний (або активний). Саме з нього зчитується завантажник ОС.

**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.

**Безперервне розміщення –** найпростіший варіант фізичної організації. При цьому файлу надається послідовність кластерів диска, що утворюють безперервну ділянку дискової пам'яті. **Основна перевага**– висока швидкість доступу, тому що витрати на пошук і зчитування кластерів файлу мінімальні. Мінімальний і обсяг адресної інформації - досить зберегти тільки номер першого кластера й обсяг файлу. Максимально можливий розмір файлу не обмежується.

Цей варіант має істотний недолік:

* неможливо пророчити, якого розміру повинна бути безперервна область, якщо файл при кожній модифікації може збільшувати розмір;
* велика проблема фрагментації (багато вільних областей малого розміру);

**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.

**Використання зв'язаного списку адрес**

Файлу виділяється пам'ять у вигляді зв'язаного списку кластерів. Номер першого кластера запам'ятовується в записі каталогу, де зберігаються характеристики цього файлу. Інша адресна інформація відділена від кластерів файлу. З кожним кластером диска зв'язується деякий елемент – **індекс**. Індекси розташовуються в окремій області диска. Ця таблиця займає один кластер. Коли пам'ять вільна, всі індекси мають нульове значення.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 3 |  | 5 |  |  |  | Область  індексів |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| 1 | | | | | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |

Рис.12.6.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, указуючи на новопризначений кластер.

Цей спосіб використовується у ФС FAT.

**Ще один спосіб складається в 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.

**Достоїнство** – 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.**

Логічний розділ, відформатований під ФС FAT, складається з наступних областей (див. мал.):

* 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 в 32 рядка (16 Кбайт), що дозволяє зберігати 512 записів про файли й каталоги, бо кожен запис каталогу складається з 32 байт.
* область даних. Призначена для розміщення всіх файлів і всіх каталогів, крім кореневого каталогу.

Завантажувальний сектор (512 байт)

0 1 2 3

FAT1

FAT2 (копія)

. Корнєвій каталог

.

.

0 1 2 3 …

Дані

елементи (покажчики) FAT

32 байта - один запис каталогу

кластери даних

Рис.12.7. Фізична організація

FAT підтримує всього два типи файлів; звичайний файл і каталог. ФС розподіляє пам'ять тільки з області даних, причому використовує в якості  мінімальної одиниці дискового простору – кластер.

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.

Таблиця FAT - загальна 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, починаючи з початку, шукає 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 значення «останній кластер файлу». Якщо розмір файлу більше, 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 кластери робити невеликими, а для скорочення обсягу адресної інформації й підвищення швидкості обміну -навпаки.

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

Існує кілька різновидів FAT індексних покажчиків, що відрізняються розрядністю, що використовується як умовне позначення 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 про те, що запис вільний, а в усі індексні покажчики файлу заноситься ознака "кластер вільний".

Інші дані в записі каталогу, у тому числі номер першого кластера файлу, залишаються недоторканими, що залишає 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 ( по різних причинах), губиться 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 питання фізичної організації файлових систем, ознайомимось більш детально з ФС FAT-32 та NTFS.

## Файлова система FAT

Абревіатура FAT (File allocation table) або таблиця розмноження файлів відноситься до лінійної таблиці із 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. На відміну від системної область даних доступна через користувацький інтерфейс DOS. Системна область складається з наступних компонентів, які розміщені у логічному адресному просторі підряд:

* + - Завантажувальний;
    - Запис(або сектор);
    - Зарезервованих секторів;
    - Таблиці розміщення файлів;
    - Кореневого каталогу;

**Таблиця розміщення файлів**

Ця таблиця уособлює собою карту або образ області даних, у якій прописано стан кожної частини 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 FAT-16 використовується 16 бітове слово, тобто можна мати 65536 кластерів. Файли або каталог займає ціле число кластерів. Останній кластер може бути задіяний не повністю, що приводить 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. Так:

|  |  |  |
| --- | --- | --- |
| Ємність розділу | Кількість секторів у кластері | Розмір кластерів(Кбайт) |
| 16-127 | 4 | 2 |
| 128-255 | 8 | 4 |
| 256-511 | 16 | 8 |
| …. | …. | …. |
| 1024-2047 | 64 | 32 |

Рис.12.8. Таблиця розміщення файлів.

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

Логічна розбивка 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 розмір у 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 на малюнку:

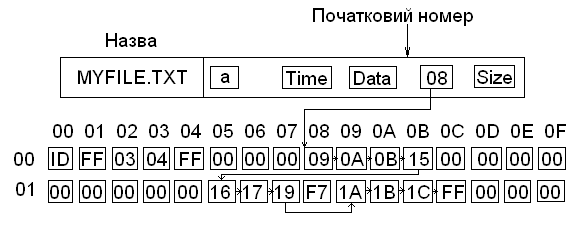


Рис.12.9. Розміщення файлової системи з використанням таблиць.

Файл починається з восьмого кластеру. Всього файл займає 11 кластерів, 18 кластер позначено спеціальним кодом F7 як поганий, 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.

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.

|  |  |
| --- | --- |
| Розмір поля даних(байт) | Зміст поля даних |
| 11 | Ім’я файлу |
| 1 | Атрибути файлу |
| 1 | Резервне поле |
| 3 | Час створення |
| 2 | Дата створення |
| 2 | Дата останнього доступу |
| 2 | Зарезервовано |
| 2 | Час останньої модифікації |
| 2 | Дата останньої модифікації |
| 2 | Номер початкового кластера FAT |
| 4 | Розмір файла |

Рис.12.10. Кореневий каталог.

Структура системи файлів – ієрархічна. У ній елементом каталогу може бути такий файл, який сам може бути каталогом:

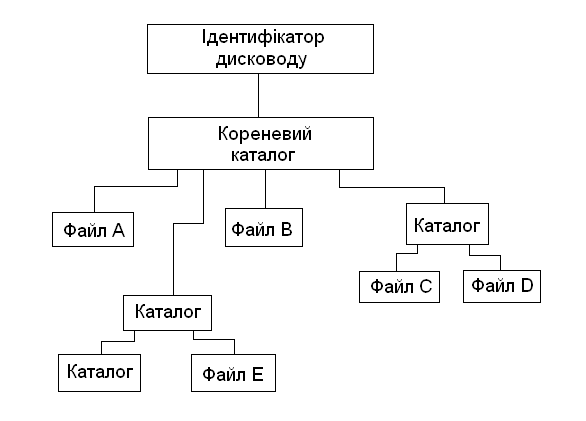


Рис.12.11. Ієрархічна структура системи файлів.

## Файлові системи VFAT та FAT-32

Ми бачимо, що дуже важливою характеристикою FAT було використання імен файлів формату «8.3», де 8 символів відводять для вказівки назви файлу, а 3 для його розширення. Пізніше до стандартної FAT-16 додали ще 2 різновиди: VFAT(віртуальний FAT) та FAT 32.

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 FAT 16, тобто підтримує формат «8.3».

Основні недоліки FAT та VFAT полягають у тому, що існують великі втрати на кластеризацію при великих розмірах логічного диску та обмеження на розмір самого логічного диску. Це стало причиною розробки нової файлової системи з використанням старої ідеї використання таблиць FAT-32.

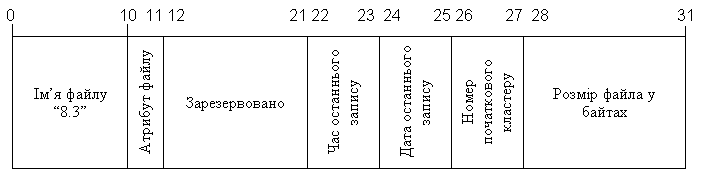
FAT-32 – 32 розрядна файлова система, яка дещо доповнена деякими удосконаленнями у порівнянні з FAT-16 та VFAT.

Принциповим є те, що FAT-32 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 FAT-32 може використовувати 4 Кбайтові кластери. В результаті економія у порівнянні з FAT-16 досягла 10-15%.

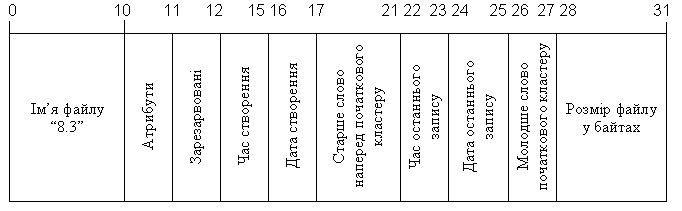
FAT-32 може переміщувати кореневий каталог та використовувати резервну копію замість стандартної. Кореневий каталог FAT-32 задається у вигляді звичайного ланцюжка кластерів. Тобто кореневий каталог може знаходитись у довільному місцезнаходженні диску, що знімає старе обмеження на розмір кореневого каталогу у 512 елементів.

Окрім збільшення ємності FAT до 4 Тбайт (тера байт – 1024 Гбайт), FAT-32 має необхідні вдосконалення у структурі кореневого каталогу. FAT-16 вимагала, щоб вся інформація кореневого каталогу була розміщена в одному дисковому кластері. При цьому вся його інформація не мала 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.

**Елемент каталогу для короткого імені FAT-16**

Рис.12.12. Елемент каталогу для короткого імені FAT-16

**Елемент каталогу для короткого імені FAT 32**

**** Рис.12.13. Елемент каталогу для короткого імені FAT-32

**Елемент каталогу для довгого імені FAT 32**

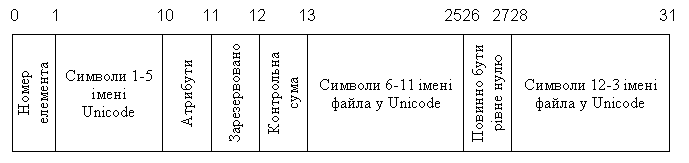
****

Рис.12.14. Елемент каталогу для довгого імені FAT 32

FAT-32 успішно справляється з проблемою довгих імен у кореневому каталозі, але проблема 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).

## Файлова система NTFS

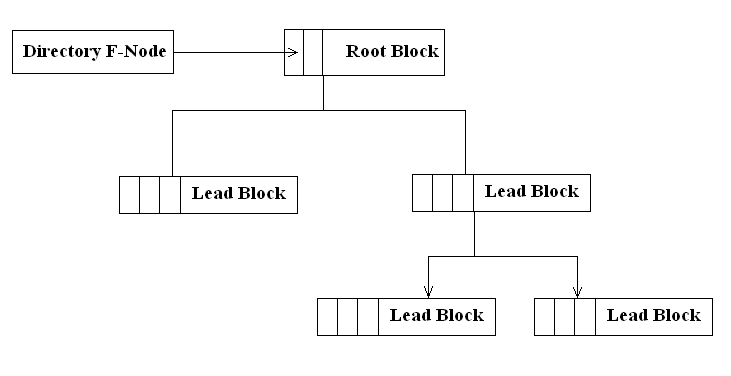
Файли зберігаються у каталогах(здебільше їх називають папками). На відміну від FAT її робота з дисками великого розміру виконується більш ефективно. У її розпорядженні також існують засоби для обмеження доступу до файлів та каталогів, введені механізми для підвищення надійності файлової системи, знято 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≈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 ми послідовно проглядаємо його із самого початку.

У випадку B-Tree(двійкове дерево, Н. Вірт Алгоритми та структури даних) структура каталогу являє собою збалансоване дерево з записами, упорядкованому згідно алфавіту. Кожен запис, що входить до цього дерева, має атрибути файлу, вказівник на відповідний файловий вузол, інформацію про час та дату створення, час і дату останнього обновлення та звертання, даних, у складі яких є розширені атрибути, лічильник звертань до файлу, довжина імені файлу, саме ім’я та інша інформація.

**Збалансоване двійкове дерево**

Рис.12.15. Збалансоване двійкове дерево

Файлова система при пошуку файлу у каталозі проглядає тільки необхідні гілки двійкового дерева. Такий метод більш ефективний, ніж послідовне читання усіх записів у каталозі, що було присутнє 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 буде необхідно виконати значно меншу кількість операцій читання з диску.

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

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 порівняні з FAT потрібно буде у найгіршому випадку більш ніж 4000 секторів.

Одним з основних понять, що використані 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. Увесь дисковий простір поділено на 2 нерівні частини, як на малюнку.

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

Рис.12.16. Дисковий простір

Перші 12 % диску відводиться 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 знаходити 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 1 Кбайт, кожний запис відповідає конкретному файлу. Перші 16 файлів носять службовий характер і недоступні ОС, вони звуться метафайлами, причому самий перший з них – сам MFT.

Ці перші 16 елементів в MFT – єдина частина диску, що має строго фіксоване положення. Копія цих же 16 записів зберігається у середині тому для надійності, тому що 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 дану основу – за перший елемент MFT.

Вище 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 «$». У таблиці 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 та інше.

$AttrDel Список стандартних атрибутів файлів на томі.

$ Кореневий каталог.

$Bitmap Карта вільного місця тому.

$Boot Завантажувальній сектор.

$Quota Файл, у якому 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 уся 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 мало, то використовується декілька записів, причому не обов’язково, щоб вони йшли підряд.

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

Файл в томі з NTFS ідентифікується з так званою файловою посилкою (посиланням, File Rederence), яка визначається як 64 розрядне число. Вона складається з номеру файлу, який відповідає позиції його файлового запису у MTF, та номера послідовності. Цей номер збільшується кожний раз, коли дана позиція у MTF використовується повторно, що дозволяє файловій системі NTFS виконувати внутрішні перевірки цілісності.

Кожен файл задано за допомогою потоків (streams), тобто у нього немає «просто даних», а є «потоки».

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

Таким чином визначається, що базова сутність у файлу тільки одна – номер у MTF, усе інше включають потоки – опції.

Даний підхід може бути ефективно використаний – наприклад, файлу можна додавати потоки та записувати у них будь-які данні.

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 із яких складається файл, файлова вказівка на файловий запис в MTF, у якій розміщено кожний з атрибутів. Останній використовується, якщо файлу необхідно мати більш ніж один запис у MTF.

- ім’я файлу

У символах Unicode файл може мати декілька атрибутів – імен файлу.

- декстриптор 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.

Готовий каталог диску – кореневий – нічим не відрізняється від звичайних каталогів, окрім спеціальної вказівки на нього із 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. Ми починаємо з такого запитання до середнього елементу. Кожна відповідь звужує зону пошук у 2 рази. Якщо файли відсортовані по алфавіту, то відповідь на питання виконується порівнянням початкових букв.

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

## Основні відмінності FAT та NTFS

Накладні витрати на зберігання службової інформації FAT відрізняються від NTFS більшою компактністю та меншою складністю. У більшості томів FAT на 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 ми вже згадували, що місткість 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 приводять 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, FAT-32 до 4 Тбайт, однак завдяки своїй внутрішній побудові, розділи FAT краще всього працюють при розмірах 200 Мбайт та менше.

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

Розміри FAT можуть бути використані практично з усіма операційними системами. З розділами NTFS можна працювати лише з ОС Windows NT, хоча існує утиліта NTFS DOS, що дозволяє читати дані NTFS на комп’ютері, що моделює режим DOS.

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.

Windows NT має у своєму розпорядженні спеціальну утиліту CONVERT.EXE, що дозволяє перетворити томи FAT у еквівалентні томи NTFS, однак зворотне перетворення неможливе.

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 FAT-32 або NTFS.

В 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 у алфавітному порядку. 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 обновлення 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 значно більш ефективний, ніж послідовне читання усіх записів у каталозі, що має місце у FAT.

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

Збалансованість дерева зменшує глибину дерева і як наслідок теж зменшує довжину пошуку.

## Файлові операції

Файлова система надає користувачам певний набір операцій по роботі з файлами.

Але які би операції не вироблялися ОС необхідно виконати ряд операцій, універсальних для всіх виконуваних дій

• по символьному імені файлу знайти його характеристики, які 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 та в кінці послідовності операцій, а для кожної проміжної операції виконуються тільки унікальні дії.

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

Зате перший спосіб більш стійкий до збоїв у роботі системи, тому що кожна операція самодостатня й не залежить від результату попередньої.

**Відкриття файлу**

Системний виклик **ореn** працює із двома аргументами: символьним ім'ям файлу, що відкриває, і режимом відкриття файлу. Режим відкриття говорить системі, які операції будуть виконуватися над файлом у послідовності операцій до закриття файлу системному виклику **close**. Наприклад, тільки читання, тільки запис або читання й запис.

**Обмін даними з файлом**

Для обміну даними з файлом (попередньо відкритим) використовуються процедури геаd й 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 становлять прикладний інтерфейс ОС.

**Блокування файлів**

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

Багатокористувальницькі 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.

**Контроль доступу до файлів.**

Файли - приватний вид розділених 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 довільно, 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 користувач віднесений.

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 строгу 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.

**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 та 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 зкожним процесом, запущеним оболонкою для даного користувача. Говорять, що процес виступає від імені даних користувача й даних груп користувачів.

Визначити права доступу до ресурсу - значить 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 самому загальному випадку права доступу можуть бути описані матрицею прав доступу, у якому стовпці відповідають всім файлам системи, рядка всім користувачам, а на перетинанні рядків і стовпців указуються різні операції.



Рис.12.16. Матриця прав доступу.

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 на виконання операцій користувачів і групи користувачів стосовно цього файлу й каталогу. Список управителя доступу є частиною характеристик файлу або каталогу й зберігається на диску у відповідній області.

Однак, не всі файлові системи підтримують список керування доступом, наприклад, його не підтримує ФС FАТ, так вона розроблялася для однокористувальницької МS -DОS

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

# Архітектурні особливості побудови ОС

Ми вивчали складові операційних систем, тому на закінчення курсу хотілось би познайомитись з основними архітектурними особливостями побудови ОС.

У цьому плані найбільший інтерес являє ОС Unix. Це мультипрограмна та багатокористувацька ОС. У свій час вона проектувалась, як засіб для розробки програмного забезпечення. Унікальність її полягає в тому, що вона була розроблена всього двома розробниками (Кен Томпсон та Денніс Рітчі). Вони націлювали її для використання на міні-ЕОМ і тому орієнтувалась на досить скромні обчислювальні ресурси. Вона має досить ефективну командну мову та незалежну від пристроїв файлову систему.

Вся ОС була написана на мові програмування високого рівня (мова С), а тому легко переносима.

Дуже важливою особливістю ОС є те, що користувачі мають можливість направляти виходи однієї програми безпосередньо на вхід другої шляхом реалізації так званих каналів (ріре). В результаті великі програмні комплекси можна створювати шляхом композиції існуючих програм, а не шляхом розробки нових. Це поклало початок новому напрямку в програмуванні – створенню технологій програмування по типу маршрутних технологічних карт подібно для виготовлення механічних деталей.

Власне ОС Unix поставляється з великим набором системних та прикладних прграм, включаючи редактори листів, програмовані інтерпретатори командної мови, компілятори, включаючи С, С++, асемблер та багато інших мов.

Зупинимось на наборі основних понять, щоб зрозуміти основні відмінності її від звичного для нас Windows.

## Віртуальна машина

ОС – багатокористувацька. Кожному користувачеві надається віртуальний комп’ютер, у якому є всі необхідні ресурси. Процесор розподіляє час на основі циклічної дисципліни обслуговування (або так званої карусельної диспетчеризації з використанням динамічних приорітетів, щоб забезпечити рівність в обслуговуванні).

Плинний стан такого віртуального комп’ютера зветься образом. Власне процес у нашій термінології – це виконання образу.

Такий образ має такі складові:

* образ пам’яті;
* стан загальних регістрів процесора;
* стан відкритих файлів;
* плинна директорія, каталог файлів та іншу інформацію.

Образ процесу під час виконання розміщується в основній пам’яті. Як правило підтримується сторінковий механізм організації віртуальної пам’яті.

Власне образ пам’яті поділяється на три логічні сегменти.

* сегмент реєнтерабельних процедур (розміщується починаючи з нульової адреси у віртуальному адресному просторі процеса).
* сегмент даних (розташований безпосередньо за сегментом процедур). Він може зростати у бік більших адрес.
* сегмент стека. Він використовується при викликах підпрограм та при перериваннях.

## Користувач

Unix орієнтована на мультитермінальну роботу. Для початку роботи людина-користувач має увійти в систему, ввести з вільного терміналу своє ім’я, а можливо і пароль. Кожен користувач має реєстраційний запис і зветься зареєстрованим користувачем системи. Реєстрацію здійснює адміністратор системи. Користувач не може змінити своє ім’я, але може змінити свій пароль.

Всі користувачі працюють з файлами. Файлова система має деревоподібну організацію. Проміжні вузли є каталогами, що мають посилання на інші каталоги, листи дерева відповідають листям або пустим каталогам.

Кожному зареєстрованому користувачу відповідає відповідальний каталог, який зветься «домашнім» (home) користувача. Користувач має необмежений доступ до свого домашнього каталогу, до всіх каталогів і файлів. Він може модифікувати та створювати нові файли. Потенційно він може мати доступ і до інших файлів, але це залежить від прав доступу.

## Інтерфейс користувача

Традиційний спосіб взаємодії між користувачем та ОС базується на використанні командних мов, а також має місце використання графічного інтерфейсу типу Windows. Після входження користувача в систему запускається один із командних інтерпретаторів, яких може бути декілька. Загальна назва для такого інтерпретатора – оболонка (shell), оскільки вона являє собою зовнішнє середовище ядра ОС.

Командний інтерпретатор видає запрошення на введення командного рядку, що являє собою просту команду, конвеєр команд або послідовність команд. Після виконання командного рядку на екран терміналу або у відповідний файл оболонка (shell) знову видає запрошення доти, доки користувач не вирішить вийти з системи.

Одним з різновидів управління є вказівка командного файлу, який у своєму складі містить такі ж командні рядки.

## Привілейований користувач

Ядро ОС ідентифікує його по його ідентифікатору (user identefier), яким позначають користувача. Крім того кожний користувач відноситься до деякої окремої групи (group identifier). Ці ідентифікатори для кожного зареєстрованого користувача зберігаються в залікових файлах системи і приписані процесу, у якому виконується командний інтерпретатор. Ці ідентифікатори наслідуються кожним новим процесом, що запущено від імені конкретного користувача.

Адміністратор має значно більші можливості, ніж звичайні користувачі. Такий користувач має назву (Superuser) суперкористувач. Він має право контролю над системою.

Для звичайних користувачів встановлені обмеження: максимальний розмір файлу, максимальна кількість сегментів поділюваної пам’яті, розмір віртуального процесору і т.п. Суперкористувач має право змінити обмеження для інших користувачів.

## Команди та командний інтерпретатор

Оболонка (shell) – це механізм взаємодії між користувачем та системою. Інтерпретатор аналізує рядки та виконує відповідні функції. Склад командного рядка наступний: команда (перелік аргументів поділених пропусками), оболонка поділяє командний рядок на компоненти. Відповідний файл завантажується і йому надається доступ до вказаним у команді аргументам.

Кожен з командних мов shell складається з трьох частин:

* службові конструкції, що дозволяють маніпулювати із рядками тексту та будувати складні команди на базі простих;
* вбудовані команди, що виконуються безпосередньо інтерпретатором;
* команд, що зображуються окремо виконуваними файлами.

Команди останньої категорії складаються із стандартних команд (системні утиліти) та команд, що створені користувачами. Для того щоб файл, що створений користувачем можна було запускти як команду оболонки (shell) достатньо визначити в одному із вихідних файлів функцію з іменем main, яке у свою чергу має бути глобальним, тобто перед ним має не бути вказано слово static. Командний інтерпретатор створить новий процес запустить на виконання програму, починаючи зі слова main. Тіло (або склад) функції main може бути довільним, але відповідно треба виконувати деякі правила. Наприклад можна використовувати тексти на мові С.

## Процеси

Під цим терміном розуміється процес у класичному його розумінні, тобто як прогама, що виконується у власному віртуальному просторі. Коли користувач входитьу систему автоматично створюється процес, у якому виконується програма командного інтерпретатора. Якщо командному інтерпретатору зустрічається команда, що відповідає виконуваному файлу, то він створює новий процес та запускає відповідну програму, починаючи зі слова main. Ця програма у свою чергу може створити процес та запустити в ньому іншу програму.

Для створення нового процесу та запуску в ньому програми використовується два системні виклики fork та exec (<ім’я файлу>). Системний виклик fork створює адресний простір процесу. Для дочірнього процесу заводяться копії всіх сегментів даних. Кожен процес має своїх батьків, але в свою чергу може бути батьком багатьох процесів. Початковий (нульовий процес) є особливим, він створюється в результаті завантаження системи. Тобто ця ОС підтримує структурований спосіб організації процесів у вигляді дерева.

**Виконання процесів.**

Процес може виконуватись в одному із двох станів: користувацькому та системному. У користувацькому стані процес виконує користувацьку програму та має доступ до користувацького сегменту даних. У системному стані процес виконує програми ядра та має доступ до системного сегменту даних.

Коли користувацькому процесу треба виконати системну функцію він виконує системний виклик. З моменту появи системного виклику процес вважається системним, оскільки фактично виконується виклик ядра системи, як підпрограми. Тобто користувацький та системний процеси являють собою дві фази одного й того ж процесу, але вони ніколи не перехрещуються між собою. Кожна фаза використовує свій власний стек. У цих ОС виконується розподіл часу, тобто кожному процесу надається квант часу. Процес або сам закінчується до завершення відведеного йому кванту або він відкладається до завершення кванту.

Механізм диспетчеризації характеризується таким розподілом часу, щоб процеси закінчувались у порядку їх виникнення. Користувацьким процесам приписують приорітети в залежності від кількості отримуємого ними процесорного часу. Процесам, що отримали велику кількість процесорного часу визначають більш низькі приорітети, у той же час процеси процеси, які отримали невелику кількість часу процесора, навпаки підвищують приорітет.

## Підсистема введення - виведення

Функції цієї підсистеми задаються в основному пам’яттю системними викликами: open, close, read, write, seek.

При читанні можливі три операції, у кожній з яких читання виконується послідовно:

* якщо це перше читання з файлу, то воно виконується послідовно з самого початку файла;
* якщо операції читання передувала інша операція читання з цього файлу, то виконувана операція надасть нам дані, безпосередньо наступні за попередніми;
* якщо читанню передувала операція seek, то читання виконується послідовно від точки зміщення, що вказана у команді seek.

Звертаю вашу увагу на те, що всі ці виклики відносяться до послідовного доступу, а ефект прямої адресації досягається за допомогою команди seek, що зміщує дану позицію файлу.

Існує ще декілька примітивів, що дозволяють отримувати та задавати інформацію про файли та термінали. Фізичні пристрої представлені спеціальними файлами у єдиній структурі файлової системи.

**Перенаправлення введення - виведення**

Механізм пере направлення введення - виведення один з найбільш простих механізмів.

Unix – інтерактивна система. Історично склалось так, що програми звичайно вводили текстові рядки з терміналу та виводили результати на екран терміналу. Для того, щоб забезпечити більш гнучкий спосіб використання таких програм треба було зуміти забезпечити їм виведення з файлу та виведення інших програм та направити їх виведення у файл або виведення іншим програмам. Реалізація цього механізму базується на таких властивостях ОС. По-перше, будь-яке введення - виведення трактується як введення з деякого файлу і виведення у деякий файл. Клавіатура та екран терміналу теж інтерпретуються як файли. По-друге, доступ до будь-якого файлу виконується через його дескриптор. Фіксуються три значення дескрипторів файлу. Файл з дескриптором 1 зветься файлом стандартного введення (stdin), файл з дескриптором 2 – файлом стандартного виведення (stdout) та файл з дескриптором 3 – файлом стандартного виведення діагностичних повідомлень (stder). По-третє програма, запущена в деякому процесі «наслідує» від породившого процесу всі дескриптори відкритих файлів. У головному процесі інтерпретатора командної мови файлом стандартного введення є клавіатура терміналу користувача, а файлом стандартного введення - виведення діагностичних повідомлень – екран терміналу. Однак, при запуску будь-якої команди можна повідомити інтерпретатору (засобами командної мови) який файл або виведення якої програми має служити файлом стандартного введення для запуску – мої прогами і який файл або виведення якої програми має слугувати файлом стандартного виведення або виведення діагностичних повідомлень для програми, яка запускається.

Інтерпретатор перед виконанням системного виклику exec відкриває вказані файли, підмінюючи зміст дескрипторів 1, 2, 3.

## Файлова система

Файл у Unix – множина символів з довільним доступом. У файлах розміщуються довільні дані. Файл має таку структуру, яку вимагає від нього користувач.

Інформація на дисках розміщується по блокам по 512 байт у кожному блоці. Блок спеціально обрано відповідно до розміру сектора.

Диск поділений на наступні області:

* невикористовуваний блок;
* управляючий або суперблок у якому зберігається розмір диску та розміри інших областей;
* і-список, складений з опису файлів, що названі і-вузлами;
* область для зберігання вмісту файлів.



## Рис.13.1. Файлова система

Організація файлів в Unix диску. Кожний вузол утримує:

* ідентифікацію користувача;
* ідентифікацію групи власника;
* біти захисту;
* фізичні адреси на диску, де знаходиться вміст файлу;
* розмір файлу;
* час створення файлу;
* час останнього використання файлу;
* час останнього змінення атрибутів;
* кількість зв’язок-вказівників, що вказують на файл;
* індикацію, являє собою файл директорію, є звичайним файлом чи спеціальним файлом.

За і-списком ідуть блоки, що визначені для зберігання вмістимого файлів. Простір на диску, що залишився вільним від файлів створює зв’язаний список вільних блоків.

Файлова система Unix структура даних, яка має у своєму складі управляючий суперблок, у якому визначена файлова система в цілому; масив і-вузлів, де визначені всі файли, самі файли та сукупність вільних блоків. Виділення простору під дані виконується блоками фіксованого розміру.

Кожен файл однозначно ідентифікується старшим номером пристрою, молодшим номером пристрою та і-номером (індексом і-вузла даного файлу у масиві і-вузлів). Коли викликають драйвер пристрою по старшому номеру індексується масив вхідних точок в драйвери. По молодшому номеру драйвер обирає один пристрій з групи ідентичних фізичних пристроїв. Файл-директорія, у якому перераховані імена файлів, дозволяє встановити відповідність між іменами та самими файлами. Директорії створюють деревоподібну структуру. На кожний звичайний файл або файл-пристрій може мати посилання у різних вузлах цієї структури. У не привілейованих програмах запис в директорії не дозволений, але при наявності відповідних дозволів вони можуть читати їх. Додаткових зв’язків між директивами немає.

Велика кількість системних директорій ОС використовує для своїх власних потреб. Одна з них – коренева директорія є базою для всієї структури директорій, і йдучи від неї, можна знайти розміщення всіх файлів. В інших системних директоріях знаходяться програми та команди, що надаються користувачам та файли пристроїв. Імена файлів задаються послідовністю імен директорій, що поділяються похилою.

Файл, що не є директорією може зустрічатися у різних директоріях, можливо під різними іменами. Це має назву пов’язуванням. Елемент в директорії, що відноситься до одного файлу – зветься зв’язком. У ОС Unix всі такі зв’язки мають однаковий статус. Файли же належать до директорій. Вони існують незалежно від елементів директорій, а зв’язки в директоріях вказують на фізичні файли. Файл зникає, коли зникає останній зв’язок, що вказує на нього.

## Захист файлів

Захист файлів виконується за допомогою номеру, що ідентифікує користувача та встановлення десяти бітів захисту – атрибутів доступу. Права доступу поділяють на три типи: читання (read), запис (write) та виконання (execute). Ці типи прав доступу можуть бути надані трьом класам користувачів: власнику файла, групі у яку входить власник та всім іншим користувачам. Дев’ять з цих бітів керують захистом по читанню, запису і виконанню для власника файлу, інших членів групи, у яку входить власник та всіх інших користувачів. Файл завжди пов’язаний з конкретним користувачем – своїм власником – і з визначеною групою, тобто у нього вже є відоме нам UID (user ID, ідентифікатор користувача) та GID (group ID, ідентифікатор групи).

Змінити права доступу до файлу може тільки його власник, змінити власника файлу може суперкористувач, групу – суперкористувач та власник файлу.

Програма, що виконується у системі завжди зупускається від імені якогось користувача та деякої групи, але зв’язок процесів із користувачами та групами організований складніше: тут розрізняють ідентифікатор доступу до файлової системи (FSUID – file system access user ID, FSGID – file system access group ID) та ефективний ідентифікатор (EUID – effective user ID, EGID – effective group ID), а при доступі до файлу враховуються ще й повноваження (capabilities) приписані до самого процесу.

При створенні файл отримує UID (ідентифікатор користувача), що співпадає з FSUID (ідентифікатор доступу до файлової системи) процесу, який його створює, і як правило GID (ідентифікатор групи), що співпадає з FSGID (ідентифікатор групи доступу до файлової системи).

Атрибути доступу визначають, що дозволено робити з конкретним файлом даної категорії користувачів. Мають місце всього три операції: читання, запис, виконання.

При створенні файлу (або ще одного імені для вже існуючого файлу) модифікується не сам файл, а каталог, у якому з’являються нові посилання на вузли. Знищення файлу – це знищення посилання. Тобто право на створення чи знищення файлу – це право на запис у каталог.

Право на виконання каталогу інтерпретується, як право на пошук у ньому (власне, проходження через нього). Воно дозволяє звернутись до файлу на шляху, що вміщує даний каталог, навіть, тоді, коли каталог не дозволено читати, а тому список всіх його файлів є недоступним.

Окрім трьох названих основних атрибутів існують додаткові, що використовуються в таких випадках.

Атрибути SUID та SGID суттєві при запуску програми на виконання: вони вимагають, щоб програма виконувалась не від імені користувача, що запустив програму, а від імені власника (або групи) того файлу, в якій вона знаходиться. Завдяки цьому користувачі отримують можливість запускати системну програму, яка створює свої робочі файли у закритих для них каталогах.

Окрім того, якщо процес створює файл у каталозі, що має атрибути SGID, то файл отримує GID не по FSGID, а по GID каталогу. Це зручно для колективної роботи: усі файли і підкаталоги у каталозі автоматично прив’язуються до однієї й тієї ж групи, хоча створювати їх можуть різні користувачі.

І ще один атрибут CVTX, який відноситься до каталогів. Він показує, з каталогу, що має атрибут, посилання на файл може знищити тільки власник файлу.

Існують дві стандартні форми запису до прав доступу – символьна та восьмирічна. Символьна – це ланцюг з десяти знаків. Восьмирічний запис – це шестизначне число, перші два знаки якого визначають тип файлу, далі атрибути, останні три – права власника, групи та всіх інших.

## Міжпроцесорні комунікації

ОС Unix повністю відповідає технології «клієнт-сервер». Ця універсальна модель слугує основою побудови систем довільної складності в тому числі й мережених. Для цього в ОС існують наступні механізми:

* сигнали;
* семафори;
* програмні канали;
* черги повідомлень;
* сегменти поділюваної пам’яті;
* виклики віддалених процедур.

Багато з цих засобів нам добре відомі, тому розглянемо їх досить коротко.

Сигнали. Якщо розглядати виконання процесу у віртуальному комп’ютері, який надається кожному користувачеві, то в такій системі повинна існувати система переривань, що відповідає наступним вимогам:

* обробка виняткових ситуацій;
* засоби обробки зовнішніх та внутрішніх переривань;
* засоби управління системою переривань (маскування та демаскування).

ОС здатна не тільки приймати та обробляти сигнали, але й породжувати їх та посилати на інші машини або процеси. Сигнали можуть бути синхронними, коли ініціатор сигналу сам процес, та асинхронними, коли ініціатором є користувач за терміналом. Джерелом асинхронних сигналів може бути також ядро, коли воно контролює визначені стани апаратури, що розглядаються як помилкові.

Сигнали можна розглядати, як найпростішу форму міжпроцесорної взаємодії, що повідомляють процеси або ядро про виникнення конкретних подій.

Семафори. Механізм семафорів – узагальнення класичного механізму семафорів, які ми з вами розглядали.

Семафор в ОС Unix складається з таких елементів:

* значення семафору;
* ідентифікатор процесу, який хронологічно останнім працював з семафором;
* кількість процесів, що чекають збільшення значення семафору;
* кількість процесів, що чекають нульового значення.

Для роботи з семафорами є наступні системні виклики:

* semget - для створення та отримання доступу;
* semop – для маніпуляції значеннями семафорів (використовується для синхронізації процесів);
* semctl – для виконання різних керуючих операцій над набором семафорів;

Основним системним викликом для маніпуляції семафором є semop:

Oldval = semop (id, oplist, count);

id – раніше отриманий дескриптор групи семафорів;

oplist – масив описувачів операцій над семафорами групи;

count – розмір цього масиву.

Значення, що повертаються системним викликам є значення останнього опрацьованого семафора. Кожний з елементів oplist має наступну структуру:

* номер семафору у вказаному наборі семафорів;
* операція;
* флаги.

Якщо перевірка прав доступу проходить нормально та вказані в масиві oplist номера семафорів не виходять за межі розміру набору семафорів, то системний виклик виконується наступним способом. Для кожного елементу масиву oplist значення відповідного семафору змінюється у відповідності до значення поля «операція»:

* якщо значення поля позитивне, то значення семафору збільшується на одиницю, а всі процеси, що чекають збільшення значення семафору, активізуються (або «пробуджуються» в термінах Unix);
* якщо значення поля операції дорівнює нулю, то якщо значення семафору теж дорівнює нулю, обирається наступний елемент масиву oplist. Якщо ж значення семафору відмінне від нуля, то ядро збільшує на одиницю кількість процесів, що очікують нульового значення семафору, а той процес, що звернувся переходить в стан очікування («усипляється» в термінології Unix).
* Якщо значення поля операції від’ємне та його абсолютне значення менше чи дорівнює значенню семафора, то ядро додає це від’ємне значення до значення семафора. Якщо в результаті значення семафору стало нульовим, то ядро активізує (пробуджує) всі процеси, що чекали нульового значення цього семафору. Якщо ж значення семафору менше абсолютної величини поля операції, то ядро збільшує на одиницю кількість процесів, що очікують збільшення значення семафору та відкладає («всипляє») плинний процес до настання цієї події.

Цікаво, що приводом для введення масових операцій над семафорами, як пояснюють розробники Unix, було бажання уникати тупикових ситуацій.

Програмні канали. Програмні канали (ріре) в ОС Unix є важливим засобом взаємодії та синхронізації процесів. Теоретично програмний канал дозволяє взаємодіяти довільній кількості процесів, забезпечуючи дисципліну FIFO (стек). Тобто процес, що читає з програмного каналу – прочитає найдавніші дані, що записані в програмний канал. При традиційній реалізації програмних каналів для збереження даних використовувались файли. В сучасних версіях Unix для реалізації програмних каналів застосовуються інші засоби IPC (зокрема черги повідомлень).

Розрізняють два види програмних каналів – поіменовані і не поіменовані. Іменований може слугувати для сполучення та синхронізації довільних процесів, що знають ім’я програмного каналу та мають відповідні права доступу.

Неіменованим програмним каналом можуть користуватися лише процес, який створив цей канал, та його потомки (не обов’язково прямі).

Для створення іменованого програмного каналу (або отримання до нього доступу) використовується звичайний системний файловий виклик open.

Для створення неіменованого програмного каналу існує спеціальний системний виклик ріре. Однак після отримання відповідних дескрипторів обидва види програмних каналів використовуються однаково за допомогою стандартних файлових системних викликів read, write та close.

Системний виклик ріре має наступний синтаксис:

pipe (fdptr);

тут fdptr – вказівник на масив з двох цілих чисел, в які після створення не поіменованого програмного каналу будуть записані дескриптори, визначені для читання з програмного каналу (за допомогою системного виклику read) та запису у програмний канал (за допомогою системного виклику write). Дескриптори не поіменованого програмного каналу – це звичайні дескриптори файлів, тобто такому програмному каналу відповідають два елементи таблиці відкритих файлів процесу. Тому при наступному використанні системних викликів read та write процес не відрізняє випадки використання програмних каналів від випадка використання звичайних файлів.

Для створення іменованих програмних каналів (або для отримання доступу до вже існуючих) використовується звичайний системний виклик open. Основною відмінністю від випадку відкриття звичайного файлу є те, що коли іменований програмний канал відкривається до запису та ні один з процесів не відкриває той же самий канал для читання, то процес, що звернувся, блокується до тих пір доки деякий процес не відкриє деякий програмний канал для читання. Аналогічно виконується обробка і до читання. Закінчення роботи процесу з прогамних каналом виконується за допомогою системного виклику close.

Черги повідомлень. Для забезпечення можливості обміну повідомленнями між процесами цей механізм підтримується наступними системними викликами:

* msgget для створення нової черги повідомлень або отримання дескриптора існуючої черги;
* msgsnd для відправлення повідомлення (вірніше для постановки його у вказану чергу);
* msgrev для отримання повідомлення (вірніше для виборки повідомлення з черги повідомлень);
* msgct для виконання ряду управляючих функцій.

Системний виклик msgget має стандартний для сімейства get синтаксис:

msggid = msgget (key, flag)

При виконанні цього виклику ОС або створює нову чергу повідомлень, поміщаючи його в заголовок таблиці черг повідомлень та повертаючи користувачу дескриптор заново створеної черги або знаходить елемент таблиці черг повідомлень, які мають вказаний ключ та повертають відповідний дескриптор черги.

Ми не будемо далі розшифровувати синтаксис інших системних викликів, бо це е є зараз метою нашого дослідження.

Поділювана пам’ять. Для роботи з поділюваною пам’яттю використовуються чотири системні виклики:

* shmget – створює новий сегмент поділюваної пам’яті або знаходить існуючий сегмент з тим же ключем;
* shmat - підключає сегмент із вказаним дескриптором до віртуальної пам’яті процесу, що звертається;
* shmdt – відключає від віртуальної пам’яті підключений до неї сегмент за вказаною початковою віртуальною адресою;
* shmct – слугує для управління різнорідними параметрами, пов’язаними з існуючим сегментом.

Після того як сегмент поділюваної пам’яті підключено до віртуальної пам’яті процесу, він може звертатися до відповідних елементів пам’яті з використанням звичайних машинних команд читання та запису без звернення до додаткових системних викликів.

Ми ж будемо розглядати синтаксис системних викликів shm, оскільки це не стосується напряму мети нашого дослідження.

Виклики віддалених процедур(RPC). У багатьох випадках взаємодія процесів носить характер «клієнт-сервер». Один з процесів запитує послугу і не продовжує своє виконання до тих пір поки не отримає цю послугу. Сегментно такий режим дуже схожий до виклику процедури. Звідтіля і відповідна назва. ОС Unix ідеально підходить для того, щоб бути мережевою операційною системою. Однак залишається проблема різного представлення даних у комп’ютерах з різною архітектурою. Тому основна ідея RPC є автоматичне забезпечення перетворення форматів даних при взаємодії процесів, що виконуються на різних комп’ютерах.

Реалізація технології віддалених процедур досить складна, оскільки цей механізм має забезпечити взаємодію процесів, що реалізуються на різних комп’ютерах. Якщо у випадку звернення до процедури у тому ж самому комп’ютері процес використовує стек або загальні області пам’яті, то при віддаленому виклику передача параметрів процедури перетворюється у передачу запиту по мережі, а це вже інша проблема.

**Операційна система LINUX**

Це подібна до Unix ОС. Це багатозадачна та багатокористувацька ОС. Вона достатньо добре сумісна з Unix на рівні вхідних текстів. Більшість програм, орієнтованих на Unix придатна для Linux без змін. Linux підтримує різні типи файлових систем для зберігання даних. Реалізована також система управління файлами на базі FAT, що дозволяє безпосередньо звертатись до файлів, що знаходяться у цій файловій системі. Підтримується файлова система ISO 9660 CD-ROM для роботи з дисками CD-ROM. Система управління файлами FAT32 теж може бути використана.

Linux забезпечує повний набір протоколів TCP/IP для мережевої роботи.

Ядро Linux було створено з урахуванням можливостей захищеного режиму процесорів 80386 та 80486. Linux підтримує завантаження тільки потрібних сторінок. Є можливість використання однієї сторінки, фізично один раз завантаженої в пам'ять одразу декількома процесами.

Програми, що виконуються використовують динамічно пов’язані бібліотеки, тобто програми, що виконуються, мають можливість сумісно використовувати бібліотечну програму, що представлена одним файлом на диску. Це дозволяє виконуваним файлам займати менше місця на диску, особливо тим файлам, які багаторазово використовують бібліотечні функції.

Є можливість також використовувати статично пов’язані бібліотеки, якщо користувач бажає користуватись відладкою на рівні об’єктних кодів або мати «повні» програми, що не потребують бібліотек, які поділяються.

Linux дозволяє бібліотеки, що поділяються, динамічно, пов’язувати під час виконання, що дозволяє програмісту підміняти бібліотечні модулі своїми власними.