МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МИКОЛАЇВСЬКИЙ НАЦІОНАЛЬНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ МЕНЕДЖМЕНТУ

Кафедра економічної кібернетики, комп'ютерних наук та інформаційних технологій

ОПЕРАЦІЙНІ СИСТЕМИ ТА СИСТЕМНЕ ПРОГРАМУВАННЯ

конспект лекцій для здобувачів першого (бакалаврського) рівня вищої освіти ОПП «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» денної форми здобуття вищої освіти

Миколаїв
2025

УДК 004.451
О-60

Друкується за рішенням науково-методичної комісії факультету менеджменту Миколаївського національного аграрного університету (протокол № 1 від 28 серпня 2025 року)

Укладачі:

С. І. Тищенко – к.п.н., доцент, доцент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

О. Ю. Пархоменко – к.ф.-м.н., доцент, доцент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

С. І. Ємельянов – PhD, старший викладач кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

Т. С. Кучмійова – к.е.н., доцент, доцент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

О. О. Жебко – асистент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

Ю. В. Волосюк - к.т.н., доцент, заступник сільського голови з питань відбудови та цифрової трансформації Шевченківської територіальної громади Миколаївської області;

А. М. Коломієць – асистент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету

Рецензенти

Р. В. Кураченко – Head o Production / Learning Experience at Interaction Design Foundation (IxDF)

О. С. Садовий – канд. техн. наук, доцент, завідувач кафедри агроінженерії Миколаївського національного аграрного університету

ЗМІСТ

ПЕРЕДМОВА 4
МОДУЛЬ 1. ОСНОВИ ОПЕРАЦІЙНИХ СИСТЕМ ТА ЇХ АРХІТЕКТУРА 5
МОДУЛЬ 2. СИСТЕМНЕ ПРОГРАМУВАННЯ: ПРОЦЕСИ, ПОТОКИ І СИГНАЛИ 21
МОДУЛЬ 3. УПРАВЛІННЯ ПАМ’ЯТТЮ В ОПЕРАЦІЙНИХ СИСТЕМАХ ТА РОБОТА ПРИКЛАДНИХ ПРОГРАМ 45
МОДУЛЬ 4. СИСТЕМНЕ ПРОГРАМУВАННЯ ВВЕДЕННЯ/ВИВЕДЕННЯ ТА ФАЙЛОВИХ СИСТЕМ 62
МОДУЛЬ 5. СУЧАСНІ ТЕНДЕНЦІЇ В ОС ТА НИЗЬКОРІВНЕВЕ ПРОГРАМУВАННЯ 80
ТЕМА 5.1. ТЕНДЕНЦІЇ РОЗВИТКУ ОС 80
ТЕМА 5.2. МЕРЕЖІ Й МЕРЕЖЕВІ ОПЕРАЦІЙНІ СИСТЕМИ ТА СИСТЕМНЕ ПРОГРАМУВАННЯ 87
СПИСОК РЕКОМЕНДОВАНИХ ТА ВИКОРИСТАНИХ ДЖЕРЕЛ 100

ПЕРЕДМОВА

Курс лекцій з дисципліни «Операційні системи та системне програмування» має важливе значення у теоретичній підготовці майбутніх фахівців і є обов'язковою складовою математичної та професійної підготовки здобувачів вищої освіти спеціальності «Комп'ютерні науки». Сучасний світ неможливо уявити без операційних систем, які становлять фундамент роботи всіх комп'ютерних процесів. Їх значущість визначається не лише можливістю застосування сучасних технологій у навчальному процесі, а й інтеграцією у різні сфери повсякденного життя. Володіння принципами операційних систем та системного програмування забезпечує ефективне управління ресурсами комп'ютера, оптимізацію виконання програм і обробку даних.

Застосування операційних систем неможливе без глибокого розуміння їх структури та принципів роботи. Майбутній фахівець повинен не лише знати основні команди та інтерфейси, а й розуміти, як система реагує на події, як розподіляє ресурси та забезпечує захист даних. Така підготовка дає змогу ефективно використовувати можливості операційних систем для розв'язання актуальних завдань та адаптуватися до динамічного розвитку інформаційних технологій.

У конспекті лекцій висвітлено основні теми курсу «Операційні системи та системне програмування», визначені державними освітніми стандартами за спеціальністю «Комп'ютерні науки». Наведено методи та підходи, що застосовуються для розв'язання практичних задач, пов'язаних із функціонуванням та розробленням системного програмного забезпечення.

Для успішного засвоєння дисципліни здобувачам достатньо базових знань з організації та обробки електронної інформації, об'єктно-орієнтованого програмування та інтелектуального аналізу даних.

Опрацювання матеріалу курсу сприяє формуванню у здобувачів визначених компетентностей та досягненню результатів навчання, передбачених освітньою програмою. Конспект підготовлено на основі ряду відомих підручників і навчальних матеріалів [2, 3, 4, 6, 7, 8], а його особливістю є доступне викладення теоретичних положень у поєднанні з їх практичною інтерпретацією.

МОДУЛЬ 1. ОСНОВИ ОПЕРАЦІЙНИХ СИСТЕМ ТА ЇХ АРХІТЕКТУРА

План

1. Призначення, функції і класифікація операційних систем
2. Архітектура, етапи розвитку, еволюція та сучасні платформи операційних систем

1.1. Призначення, функції і класифікація операційних систем

В залежності від того, які з перерахованих вище функцій операційні системи та системне програмування виконують, їх можна розділити на:

ДОС (Дискові операційні системи та системне програмування)

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 в пам'ять і 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 до якогось іншого стану, ДОС нічим 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 подібних систем обумовлено їхньою простотою й малих ресурсів для їх реалізації.

Ще одна причина їх можливого використання навіть на досить потужних машинах – вимога програмної сумісності з ранніми моделями того ж самого сімейства комп'ютерів.

ОС загального призначення

Це системи, що виконують всі перераховані функції, наприклад, IBM DOS й ОС/360 і наші аналоги ОС ЄС.

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

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

Системи віртуальних машин

Ці ОС допускають одночасну роботу декількох програм, але створюють при цьому для кожної програми ілюзію того, що машина перебуває в повному

її розпорядженні, як і при роботі під керуванням ДОС.

Віртуальні машини – цінний 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 використовуються для промислового застосування. У системах віртуальних машин приділяється велика увага емуляції роботи апаратури.

Часто ці системи є підсистемами ОС загального призначення, наприклад підсистема W_0W в Windows NT, Windows 2000, емулятор RT-II в VAX тощо.

Системи реального часу

Ці системи призначені для полегшення розробки так званих програм **реального часу** – програм, які управляють некомп'ютерним устаткуванням, часто із жорсткими часовими обмеженнями.

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

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

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

На сьогодні існує кілька визначень систем реального часу (СРЧ) (real time operating systems (RTOS)), однак більшість з них суперечить одне одному.

Наведемо деякі з цих визначень для демонстрації різних поглядів на призначення і основні завдання СРЧ:

1. Системою реального часу називається система, в якій успішність роботи будь-якої програми залежить не тільки від її логічної правильності, а й від часу, за який вона отримала результат. Якщо часові обмеження не витримані, тоді фіксується збій в роботі систем.

Таким чином, часові обмеження повинні бути гарантовано витримані. Це вимагає від системи бути передбачуваною, тобто, незалежно від свого поточного стану і завантаженості, видавати потрібний результат за необхідний час. При цьому бажано, щоб система забезпечувала якомога більший відсоток використання наявних ресурсів. Прикладом завдання, де потрібна СРЧ, є управління роботом, що бере деталь зі стрічки конвеєра. Деталь рухається, і робот має лише невелике часове вікно, коли він може її взяти. Якщо він

запізниться, то деталь вже не буде на потрібній ділянці конвеєра, і, отже, робота не буде зроблена, незважаючи на те, що робот знаходиться в правильному місці. Якщо він позиціонується раніше, то деталь ще не встигне під'їхати, і він заблокує їй шлях.

Іншим прикладом може бути космічний апарат, що знаходиться на автопілоті. Сенсорні серводатчики повинні постійно передавати в керуючий комп'ютер результати вимірювань. Якщо результат будь-якого вимірювання буде пропущено, то це може привести до неприпустимої невідповідності між реальним станом систем космічного апарату і інформацією про нього в керуючій програмі. Розрізняють жорсткі (hard) і слабкі (soft) вимоги реального часу. Якщо запізнення програми призводить до повного порушення роботи керованої системи, тоді говорять про жорсткі вимоги реального часу (жорсткі СРЧ).

Якщо ж запізнювання призводить тільки до втрати продуктивності, тоді говорять про слабкі вимоги реального часу (м'які СРЧ). Більшість програмного забезпечення орієнтоване на м'який реальний час, а завдання хорошої СРЧ – забезпечити рівень безпечного функціонування системи, навіть якщо керуюча програма ніколи не закінчить своєї роботи.

2. Стандарт POSIX 1003.1 визначає СРЧ наступним чином: реальний час в операційних системах – це здатність операційної системи забезпечити необхідний рівень сервісу в заданий проміжок часу.

3. Іноді системами реального часу називають системи постійної готовності (on-line системи), або інтерактивні системи з достатнім часом реакції. Звичайно це роблять фірми-виробники з маркетингових міркувань. Якщо інтерактивну програму називають працюючою в реальному часі, то це означає, що вона встигає обробляти запити від людини, для якої затримка в сотні мілісекунд навіть непомітна.

4. Часто поняття «система реального часу» ототожнюють з поняттям «швидка система». Це не завжди правильно. Час затримки реакції СРЧ на подію вже не так і важливий (він може досягати декількох секунд). Головне, щоб цього часу було достатньо для конкретного додатку і гарантовано. Часто алгоритм з гарантованим часом роботи менш ефективний, ніж алгоритм, що такою властивістю не володіє. Наприклад, алгоритм «швидкого» сортування (quicksort) в середньому працює значно швидше багатьох інших алгоритмів сортування, але його гарантована оцінка складності значно гірша.

5. У багатьох важливих сферах виконання додатків СРЧ вводяться свої поняття «реального часу». Так, про процес цифрової обробки сигналу кажуть, що він йде в «реальному часі», якщо аналіз (при введенні) і / або генерація (при виведенні) даних може бути проведено за той самий час, що і аналіз та / або

генерація тих самих даних без цифрової обробки сигналу. Наприклад, якщо при обробці аудіо даних потрібно 2,01 секунди для аналізу 2,00 секунди звуку, тоді це не процес реального часу. Якщо ж потрібно 1,99 секунди, тоді це – процес реального часу. Виходячи з вищезазначеного, визначення системи реального часу можна подати в наступній інтерпретації.

Визначення. Система реального часу реагує в передбачуваний час на непередбачувану появу зовнішніх подій.

Основні відмінності між ОС РЧ та звичайними ОС можна продемонструвати за допомогою табл. 1.1.

Таблиця 1.1. Відмінності ОС РЧ

	ОС РЧ	Звичайна ОС
Основна задача	Можливість реагувати на події, що надходять зовні	Оптимально розподілити ресурси комп'ютера між користувачами та задачами
Як позиціонується	Інструмент для створення конкретного програмно-апаратного комплексу	Сприймається користувачем як сукупність додатків, готових до використання
На що орієнтована	Обробка зовнішніх подій	Обробка дій користувача
Кому призначена	Кваліфікованому розробнику	Розробнику середньої кваліфікації

Засоби крос-розробки

Це алгоритми, що 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:

- 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

Існують системи, які важко віднести до якогось одного з перерахованих типів. 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, і навіть забезпечують деяку подібність 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 ПЗ, крім того, запозиченню окремих вдалих рішень.

На основі такої спадковості можна побудувати “генеалогічні дерева” ОС. Так, можна виділити кілька сімейств нині експлуатованих ОС, наприклад,

сімейство UNIX, у тому числі ОС Linux, а також сімейств, які вже вимерли або вимирають, наприклад, системи для великих комп'ютерів IBM OS-360, IBM-Vm, OS/2.

1.2. Архітектура, етапи розвитку, еволюція та сучасні платформи операційних систем

Найбільш загальним підходом до структуризації операційної системи є поділ усіх 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 **прикладне програмне середовище.** Додатки звертаються до ядра із запитами – **системними викликами** – для виконання тих чи інших дій (наприклад, відкриття й читання файлу, виводу графіки, одержання системного часу й т.п.).

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

Функції, виконувані модулями ядра, є найчастіше використовуваними, тому швидкість їхнього виконання визначає продуктивність всієї системи в цілому. Саме тому більша частина модулів ядра є резидентною, тобто постійно перебуває 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 якась програма частиною ОС чи ні, приймає виробник ОС.

Допоміжні модулі ОС поділяють на наступні групи:

- утиліти – програми, що вирішують окремі завдання керування й

супроводу комп'ютерної системи (наприклад, програми стискування дисків, архівування даних);

• системні оброблюючі програми – текстові й графічні редактори, компілятори, компонувальники, відладчики і т. п.;

• 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 (рис.1.1).

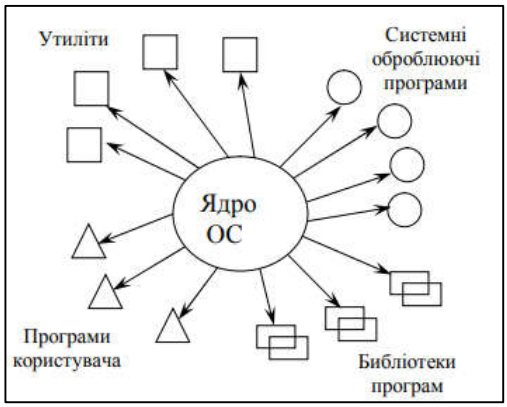

Рис. 1.1. Загальна структура ОС

Допоміжні модулі ОС завантажуються в оперативну пам'ять тільки на час свого виконання, 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 привілеї 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 підлеглий стан 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 трирівневу 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 користувацького режиму, стала класичної. Її використовують багато відомих 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, звертання програм до ядра здійснюються швидше, але при цьому відсутній надійний апаратний захист пам'яті.

За своєю внутрішньою архітектурою ОС можна умовно поділити на монолітні ОС, ОС на основі мікроядра та об’єктно-орієнтовані ОС. ОС з монолітною архітектурою можна представити у вигляді:

- прикладний рівень – прикладні процеси;
- системний рівень – монолітне ядро ОС , що, у свою чергу, складається:
  - ❖ з інтерфейсу між додатками та ядром;
  - ❖ власне ядра ОС;
  - ❖ інтерфейсу між драйверами та пристроями.

Основною перевагою монолітної архітектури є більша швидкодія, у порівнянні з іншими архітектурами. Однак, ця перевага досягається, в основному, за рахунок написання основної частини програми ОС на асемблері.

Недоліки монолітної архітектури:

- системні виклики, що потребують переключення рівня привілеїв, повинні реалізовувати інтерфейси між додатками та ядром як переривання. Це значно збільшує час їх виконання;

- ядро не може бути перерване додатком. Це може призвести до того, що високопріоритетна задача може не отримати процесор, зайнятий виконанням низькопріоритетної задачі. Наприклад, низькопріоритетна задача запросила виділити пам’ять, зробила системний виклик, до закінчення якого сигнал на активізацію високопріоритетної задачі не може її активізувати;

- складність переходу на нову архітектуру процесора при невідповідності асемблерів;

- негнучкість та складнощі з розвитком, оскільки зміна частини ядра потребує його повної перекомпіляції.

ОС на основі модульної організації прибирає недоліки, пов’язані з

інтерфейсом між додатками та ядром, полегшує модернізацію ОС та її встановлення на нові мікропроцесори.

При модульній архітектурі інтерфейс відіграє тільки одну роль – забезпечує зв'язки додатків із спеціальним модулем – менеджером процесів. А ядро – подвійну роль:

- керує взаємодією частин системи (менеджерів процесів та файлів);
- забезпечує неперервність виконання коду системи (тобто переключення задач відсутнє під час виконання функцій ядра).

В модульній архітектурі недоліки майже залишились, хоча вони перейшли з рівня інтерфейсу на рівень ядра.

Вище вивчались складові операційних систем, а на закінчення курсу хотілось би познайомитись з основними архітектурними особливостями побудови ОС. У цьому сенсі, найбільший інтерес представляє ОС Unix.

Це мультипрограмна та багатокористувацька ОС. У свій час вона проектувалась, як засіб для розробки програмного забезпечення. Вона має досить ефективну командну мову та незалежну від пристроїв файлову систему.

Вся ОС була написана мовою програмування високого рівня (мова С), а тому вона легко переноситься. Дуже важливою особливістю ОС є те, що користувачі мають можливість направляти виходи однієї програми безпосередньо на вхід другої шляхом реалізації так званих каналів (pipe).

Як результат, великі програмні комплекси можна створювати шляхом композиції існуючих програм, а не шляхом розробки нових. Це поклало початок новому напрямку в програмуванні – створенню технологій програмування подібно маршрутних технологічних карт для виготовлення механічних деталей.

Власне ОС Unix поставляється з великим набором системних та прикладних програм, включаючи редактори текстів, програмовані інтерпретатори командної мови, компілятори С, С++, асемблера та багато інших мов. Зупинимось на наборі основних понять, щоб зрозуміти основні відмінності Unix від звичного для багатьох ОС 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 пріоритетів, щоб забезпечити рівні права в обслуговуванні).

Поточний стан такого віртуального комп'ютера називають образом. Власне процес у нашій термінології – це виконання образу. Такий образ має наступні складові:

- образ пам'яті;
- стан загальних регістрів процесора;

- стан відкритих файлів;
- поточна 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 реєнтабельних процедур;
- сегмент даних;
- сегмент стеку.

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, листя дерева відповідають листям або порожнім каталогам.

Кожному зареєстрованому користувачу відповідає свій каталог, який зветься «домашнім» (home) каталогом 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 ввід 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 містить такі ж командні рядки.

Привілейований користувач

Ядро ОС ідентифікує користувача за його ідентифікатором (user identifier), яким позначають користувача. Крім того, кожний користувач

відноситься до деякої окремої групи (group identifier). Ці ідентифікатори для кожного зареєстрованого користувача зберігаються у спеціальних 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 вказаних у команді аргументам.

Кожен з командних мов shell складається з трьох частин:

- службові інструкції, 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 створений користувачем файл можна було запускати 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 не має бути вказано слово static.

Командний інтерпретатор створить новий процес, запустить на виконання програму, починаючи зі слова main. Тіло (або вміст)
функції main може бути довільним, але потрібно виконувати деякі правила. Наприклад, можна використовувати тексти мовою С.

Процеси

Під цим терміном розуміється процес у класичному його розумінні, тобто як програма, що виконується у власному віртуальному просторі. Коли користувач входить у 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 використовуються два системні виклики fork та exec (<ім'я файлу>). Системний виклик fork створює адресний простір процесу.

Для дочірнього процесу створюються 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 процесорного часу, що вони отримують. Процесам, що отримали багато процесорного часу, визначають більш низькі приорітети; у той же час процесам, які отримали невелику кількість часу процесора, навпаки, підвищують приорітет.

Підсистема введення-виведення

Функції цієї підсистеми задаються 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 файлу;
- якщо операції читання передувала інша операція читання з цього файлу, то виконувана операція надасть дані, що безпосередньо наступними за

попередніми;

- якщо читанню передувала операція seek, то читання виконується послідовно від точки зміщення, що вказана у команді seek.

Звертаємо увагу на те, що всі ці виклики відносяться до послідовного доступу, а ефект прямої адресації досягається за допомогою команди seek, що зміщує поточну 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 перенаправлення введення-виведення є одним 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 вивід інформації з файлів інших програм і направити її у файл або ввод-вивід конкретної програми.

Реалізація цього механізму базується на наступних властивостях ОС. По перше, будь-який введення-виведення трактується як ввід з деякого файлу і вивід 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 вводу (stdin), файл з дескриптором 2 – файлом стандартного виводу (stdout) та файл з дескриптором 3 – файлом стандартного виводу діагностичних повідомлень (stder). По-третє, програма, запущена в деякому процесі «наслідує» від породжуваного процесу всі дескриптори відкритих файлів.

У головному процесі інтерпретатора командної мови файлом стандартного вводу є клавіатура терміналу користувача, а файлом стандартного виводу та виводу діагностичних повідомлень – екран терміналу.

Однак, при запуску будь-якої команди можна повідомити інтерпретатору (засобами командної мови), який саме файл або вивід якої програми має слугувати файлом стандартного вводу для запуску якої програми, і який файл або вивід якої програми має слугувати файлом стандартного виводу або виводу 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 блоками – по 512 байт у кожному блоці. Блок спеціально обрано відповідно до розміру сектора. Диск поділено на наступні області (рис. 1.2):

- невикористовуваний блок;
- керуючий або суперблок, у якому зберігається розмір диску та розміри інших областей;
- і-список, складений з опису файлів, що називають і-вузлами; область для зберігання вмісту файлів.

Рис. 1.2. Поділ диску на області

Організація файлів в Unix диску. Кожний вузол містить:

- 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 останньої змінення атрибутів;
- кількість зв’язок-вказівників, що вказують на файл;
- індикацію, що являє собою файл-директорію, і є звичайним файлом чи спеціальним файлом.

За і-списком йдуть блоки, що визначені для зберігання вмісту 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

керуючий 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, за старшим номером індексується масив вхідних точок в драйвери.

За молодшим номером, 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 бути посилання у різних вузлах цієї структури. У непривілейованих програмах запис в директорії не дозволено, але при наявності відповідних дозволів вони можуть читати їх. Додаткових зв'язків між директивами немає.

Велику 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.

Файл, що не є директорією, може зустрічатися у різних директоріях, можливо під різними іменами. Це має назву **зв'язування.** Елемент в директорії, що відноситься до одного файлу, звуть 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 щодо читання, запису і виконання для власника файлу, інших членів групи, у яку входить власник, та всіх інших користувачів. Файл завжди зв'язаний з конкретним користувачем – своїм власником – і з визначеною групою, тобто у нього вже є відоме 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.

Виконувана в системі програма завжди запускається 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 самому 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 у таких випадках. Атрибути SUID та SGID є суттєвими 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

вісімкова. Символьна – це ланцюг з десяти знаків. Вісімковий запис – це шестизначне число, перші два знаки якого визначають тип файлу, далі атрибути, останні три – права власника, групи та всіх інших.

Питання до самоконтролю:

1. Призначення операційних систем і системного програмування.
2. Основні функції дискових операційних систем (ДОС).
3. Характерні приклади операційних систем загального призначення.
4. Призначення і особливості систем віртуальних машин.
5. Відмінності між системами реального часу і звичайними ОС.
6. Класифікація систем реального часу на жорсткі та м’які.
7. Архітектура ОС, поділ модулів на ядро та допоміжні модулі.
8. Принципи забезпечення привілеїв ядра та користувацького режиму.
9. Основні відмінності монолітної та модульної архітектури ОС.
10. Структура та особливості файлової системи Unix.

МОДУЛЬ 2. СИСТЕМНЕ ПРОГРАМУВАННЯ: ПРОЦЕСИ, ПОТОКИ І СИГНАЛИ

План

1. Процеси: класифікація, управління і взаємодія.
2. Потоки, синхронізація і блокування.
3. Сигнали та механізми обробки переривань в ОС.

2.1. Процеси: класифікація, управління і взаємодія

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

Велику послідовну програму, написану алгоритмічною мовою, можна розділити на менші послідовні програми, що звуть 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 щоразу під час своєї 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 відображується й на програмному забезпеченні. ОС виконує безліч робіт, які виконуються майже незалежно: введення, виведення, 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 забезпечення. У системах з декількома арифметичними процесорами й значною кількістю каналів, що працюють паралельно, організація робіт стає складним завданням.

На додаток до багатьох апаратних пристроїв, що працюють паралельно, Може бути ще кілька завдань, які також виконуються паралельно. Ці паралельно виконувані завдання взагалі не повинні враховувати існування одне одного. Коли якесь завдання віддає наказ “друкувати”, його не стосується те, що інше завдання в цей самий час здійснює запис на диск тощо.

Для отримання адекватної моделі такого функціонування, потрібно

ввести одиницю роботи, яка б відображала цю паралельність, а це і є **процес**. Оскільки роботи виконуються незалежно, і процеси, що їх представляють, працюють з різними швидкостями, повинні допускатися довільні співвідношення швидкостей виконання процесів.

Отже, **процеси** – незалежні роботи, які виконуються паралельно та з різними швидкостями. Роботам, іноді, все ж таки необхідно обмінюватися інформацією. Тому вони не є цілком незалежними. Однак спілкування між процесами повинно проходити тільки за чіткими, чітко визначеними схемами, щоб уникнути плутанини й невизначеності.

Крім того, робиться припущення, що операції всередині процесу виконуються в чітко визначеній послідовності.

Варто спробувати дати формальне визначення поняття процесу, оскільки існує дуже багато його визначень. Нехай $X = \{x_0, x_1, ..., x_N, ...$ – набір змінних, що характеризують стан, який називають набором змінних станів. **Стан** описується завданням 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, називають **роботою**.

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

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 задано процес P, який має дві змінні x й y. Роботу процесу P можна описати послідовністю станів: {(2,1 ,(4,2 ,(6,3) 8,4 ... 2i,i ... .

Роботу процесу можна також описати, вказавши простір станів {(i, j), де i, j – натуральні числа}, один з початкових станів (2,1) і функцію дії, що виконує дії над всіма станами: (x,y)→(x+2,y+1).

Кожен стан процесу – це “миттєвий знімок” ходу роботи, що виконує процес. Нехай програма разом зі своїми змінними перебуває в оперативній

пам'яті. Можна слідкувати за виконанням програми, спостерігаючи за пов'язаними з нею комірками й регістрами. Послідовність станів описує хід роботи програми в даному середовищі.

Очевидно, що при наявності окремих наборів змінних станів у двох процесів, вони не можуть взаємодіяти. Спілкування між процесами забезпечується поділюваними змінними.

У ряді ОС, крім поняття «процес», виділяється дрібніша одиниця, що має назву **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.

Це 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 будемо, з метою спрощення, ототожнювати ці поняття й користуватися терміном **процес**.

2.2. Потоки, синхронізація і блокування

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

Мультипрограмування або багатозадачність – спосіб організації обчислювального процесу, при якому на одному процесорі почергово виконуються одразу декілька програм.

Найбільш характерними критеріями ефективності обчислювальних систем є:

- Пропускна здатність – кількість завдань, виконуваних обчислювальною системою в одиницю часу.
- Зручність роботи користувачів – користувачі 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 поділяють на:

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

Системи пакетної обробки

Призначалися для рішення завдань обчислювального характеру, що не потребує швидкого одержання результатів. Головною метою й критерієм ефективності систем пакетної обробки є максимальна пропускна здатність. Для досягнення цієї мети формується пакет завдань, кожне завдання містить вимоги

до системних ресурсів, із цих завдань формується мультипрограмна суміш, тобто безліч одночасно виконуваних завдань.

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

Спеціалізований процесор введення-виведення

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

- установити магнітну головку;
- надрукувати рядок тощо.

Канальні програми можуть зберігатися в тій самій оперативній пам'яті, що й програми центрального процесора. В системі команд центрального процесора передбачається спеціальна інструкція, за допомогою якої каналу передаються параметри й вказівки на програму введення-виведення, яку він повинен виконати (рис. 2.1).

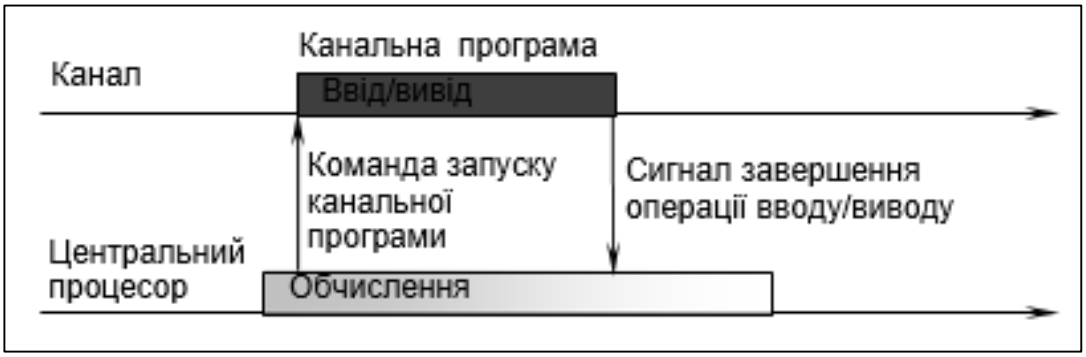

Рис. 2.1. Спеціалізований процесор введення-виведення

Зовнішні пристрої, керовані контролерами. Кожний зовнішній пристрій (або група) має свій власний контролер, який опрацьовує команди від центрального процесора. Контролер і ЦП працюють асинхронно.

Контролер повідомляє ЦП про свою готовність прийняти наступну команду сигналом переривання або ЦП довідається про це, періодично опитуючи стан контролера (рис. 2.2).

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.

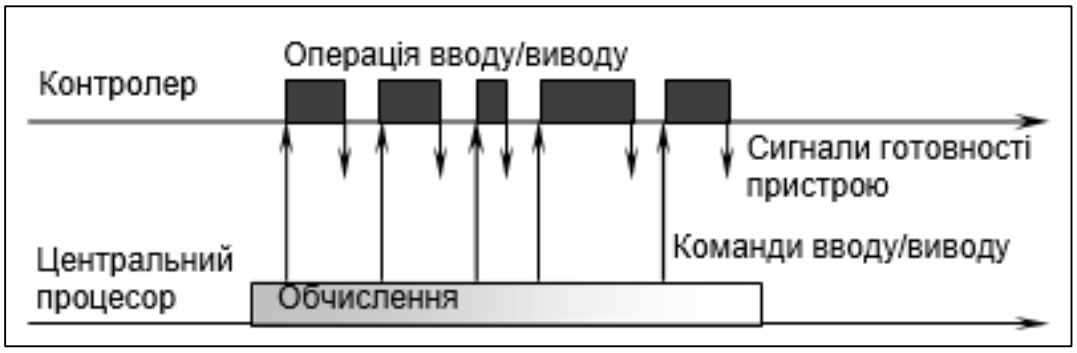

Рис. 2.2. Управління контролером

Системи поділу часу

Системи поділу часу усувають основний недолік пакетної обробки – ізоляцію користувачів-програмістів від процесу виконання їх завдань. Кожному користувачеві виділяють окремий 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 програмістові).

Системи реального часу

Основний критерій – здатність системи витримати заздалегідь задані інтервали часу між запуском програми й одержанням результату. Цей час називають **часом реакції системи**, а відповідну 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 за перериваннями (наприклад, виходячи з поточного стану об'єкта керування) або згідно до розкладу 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.

На даний час стало звичайним явищем включення декількох процесорів в архітектуру персонального комп'ютера.

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

Мультипроцесорні системи визначають як **симетричні** або як **несиметричні**, в залежності від того, до якого аспекту обчислювальної системи це відноситься:

- до архітектури;
- до способу організації обчислювального процесу.

Симетрична архітектура дозволяє 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 ж оперативною пам'яттю та розташовуються в одному корпусі. Це **масштабування по вертикалі**.

В 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, що надається їм у системі.

Масштабування в асиметричній архітектурі реалізується інакше, ніж у симетричній. Система може складатися з декількох пристроїв, кожний з яких містить один або кілька процесорів. Це **масштабування по горизонталі**.

Кожен такий пристрій називають **кластером**, а всю систему звичайно називають **кластерною**. Спосіб організації обчислювального процесу в мультипроцесорній системі визначається ОС.

Асиметричне мультипроцесування є найпростішим способом організації. Цей спосіб іноді називають «ведучий - ведений».

На «провідному» процесорі працює ОС, що управляє всіма іншими «веденими» процесорами. Він бере на себе функції розподілу завдань і ресурсів, а «ведені» працюють тільки як оброблюючі пристрої, й ніякі дії з організації роботи обчислювальної системи не виконують. Така ОС не набагато складніша за 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 реалізовано 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

реєнтерабельності (повторної входимості). ОС повністю децентралізована. Як тільки процесор завершує виконання чергового завдання, він передає керування планувальникові, що вибирає 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.

Кожен програмний процес однозначно визначається інформаційною структурою, яку називають **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, але зворотно не завжди вірно. Дескриптор й область пам'яті, з якої складається програмний процес, повинні бути виділені з наявних у машині ресурсів.

Є два підходи до створення програмних процесів.

Можна побудувати систему з фіксованою кількістю програмних процесів, які існують завжди. У такому випадку, програмні процеси буде створено 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, прийдеться очікувати, коли йому буде наданий програмний процес для виконання роботи. *Інший шлях* полягає в тому, що в системі передбачають механізм для створення і знищення програмних процесів при надходженні відповідного запиту.

Цей механізм, який сам не є програмним процесом, дає системі можливість маніпулювати програмними процесами. Його називають **стрижнем**.

У процесі може викликатися створення або знищення інших процесів. Так один процес може зажадати, щоб стрижень утворив інший процес. Стрижень створює дескриптор нового процесу, виділяє для процесів пам'ять і повідомляє про завершення цієї діяльності утворюючому процесу.

Тоді цей процес поміщає в пам'ять нового процесу програму (тобто подання деякої функції дії). У стрижні передбачена команда “запустити”, що виділяє процесу процесор.

Аналогічно команда “зупинити” відбирає процесор у процесу, 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 одна з одною одиниці, або можуть бути зв'язані особливими відносинами, утворюючи структури. Якщо система не передбачає такої структурної організації, тоді супервізор зобов'язаний, якщо потрібно, створювати процеси.

З появою завдання супервізор створить процес для виконання завдання. Коли завдання виконано, процес знищується. При такій організації всі процеси рівні.

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

У системі з деревоподібною структурою процес називають **батьком** всіх процесів, які він створить, і процес називають **сином** того процесу, що його створив.

Тут має місце визначення предків і нащадків. **Генеалогічне дерево** процесів у системі являє собою 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 в будь-який момент часу.

У системі з деревоподібною структурою (рис. 2.3) можуть бути дуже чіткі правила щодо передачі ресурсів й організації керування.

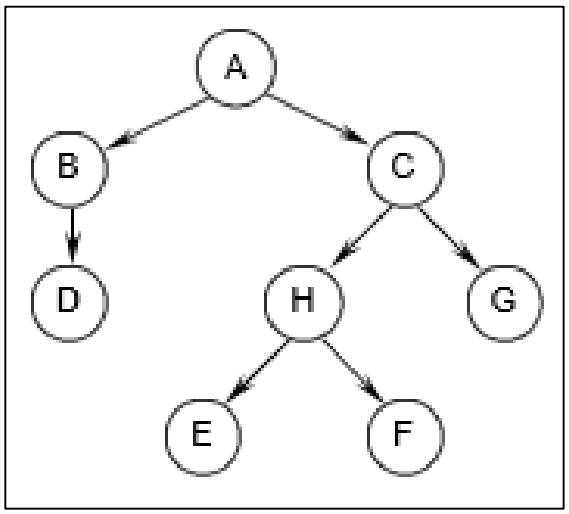

Рис. 2.3. Дерево створення процесів

Наприклад, при створенні 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.

Така структурована система має переваги:

- Розподіл ресурсів перебуває під строгим контролем. Ресурси всякого процесу колись були власністю кожного з його предків. Якщо процес хоче звільнити деякі зі своїх ресурсів, він може віддати їх тільки своєму батькові. Ніколи не буває незрозумілим, який ресурс належить якому з процесів.

- Вся структура, в цілому, така, що одні з процесів мають більше прав, ніж інші, тобто в ній передбачено простий механізм поділу роботи.

- Завдяки структурі завжди зрозуміло, якому процесу належить керування: батько управляє синами.

У безструктурній системі все знає й виконує супервізор. Хоча централізоване керування допускає вільне розбиття завдань, це ускладнює ведення обліку, оскільки супервізору доводиться стежити за кожним процесом у системі.

Для ефективного використання апаратних й програмних ресурсів їх поділяють. Розглянемо завдання користувача, що виконується в системі, як єдиний програмний процес. Рідко буває, коли процесу протягом усього часу його існування потрібен деякий апаратний пристрій для одноосібного користування.

Припустимо, що він використовує канал для роботи з периферійним пристроєм. Якщо канал обслуговує вимоги процесу швидше, ніж останній їх породжує, є сенс організувати поділ єдиного каналу між декількома процесами.

Для зниження втрат через неефективне використання програмних засобів також можна 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 декілька процесів, тоді таке розмноження, з погляду ресурсів пам'яті, є неоптимальним. Доцільно розділити за часом один екземпляр компілятора. Зауважимо, що апаратні і програмні ресурси мають загальні властивості. Розглянемо систему керування файлами, яка у кожен момент часу може обслуговувати запити тільки одного процесу.

Якщо ця система – програмний ресурс, її поділ організується так само, як і поділ апаратних ресурсів. Фізичні пристрої називаються фізичними або природними ресурсами. Частина програмного забезпечення, що поводиться подібно фізичному ресурсу, називають логічним ресурсом. Процесу неважливо, яким саме ресурсом він користується: фізичним чи логічним; важливо, щоб ресурс робив те, що повинен.

Дотепер 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 цю дрібну одиницю називають **потоком**.

При реалізації потоків з'являється можливість організації паралельних обчислень в межах процесу.

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

Із цього виходить, що поряд з механізмами керування процесами, в ОС потрібний інший механізм розпаралелювання обчислень, який враховував би тісні зв'язки між окремими гілками обчислень одного й того ж додатку.

Для цього у ряді сучасних ОС використовується механізм **багатопоточної обробки**. Вводиться нова одиниця роботи – **потік виконання**, а поняття «процес» до деякої міри змінює вміст.

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, чи перебуває він в активному або пасивному стані, чи образ процесу перебуває в оперативній пам'яті або вивантажено на диск.

Образ – сукупність кодів команд і даних процесу.

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

У дескрипторі прямо або посередньо (через покажчики) зберігається 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, інформацію про всі відкриті даним процесом файли і незавершені операції введення-виведення й інші дані,

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 відноситися до процесів у цілому, якщо ОС не підтримує потоки).

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

Роботу з визначення того, у який момент часу необхідно перервати потік і якому саме потоку надати можливість виконуватися, називають **плануванням**.

При плануванні можуть прийматися до уваги **пріоритет потоків**, **час їхнього очікування** в черзі, накопичений час виконання, інтенсивність звертання до пристроїв введення-виведення й ін.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 вирішують згадані вище 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 – потоки і процеси з'являються у випадкові моменти часу й також непередбачено завершуються. Динамічні планувальники можуть гнучко пристосовуватися до ситуації, що змінюється. І для пошуку оптимальних рішень ОС повинна витрачати значні зусилля.

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

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

Диспетчеризація полягає в реалізації знайденого в результаті

планування (динамічного або статичного) рішення, тобто в перемиканні процесора з одного потоку на інший. Перш, ніж перервати виконання потоку, ОС запам'ятовує його контекст, щоб згодом використати цю інформацію для наступного поновлення виконання даного потоку.

Контекст визначає:

- стан апаратури 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, виконувані даним потоком системні виклики тощо).
- Диспетчеризація зводиться до наступного :
- збереження контексту поточного потоку, що потрібно замінити;
- 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 до виконання, але процесор зайнятий виконанням іншого потоку.

Зауважимо, що стани виконання й очікування можуть бути віднесені й до завдань, що виконується в однопрограмному режимі, а стан **готовності** характерний тільки для режиму мультипрограмування. Граф станів потоку в багатозадачному середовищі можна представити як на рис. 2.4.

Рис. 2.4. Граф станів потоку в багатозадачному середовищі

Витіснення 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 потоків, що очікують, і готових потоків.

2.3. Сигнали та механізми обробки переривань в операційних системах

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

Існують три основні напрямки політики запобігання тупиків:

- запобігання тупиків;
- автоматичне виявлення;
- виявлення за участю оператора.

Метод автоматичного виявлення тупиків допускає потрапляння системи в тупикову ситуацію, проте з можливістю виявити це програмним шляхом. Потім система відбирає ресурси в інших процесів і віддає їх на те, щоб зрушити з місця процеси, які потрапили в тупик.

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

Розглянемо систему, в якій процеси конкурують через стрічкопротягувальні пристрої. Якщо процесу надати всі необхідні йому пристрої, він буде працювати протягом кінцевого відрізка часу, після чого повертає всі надані йому пристрої.

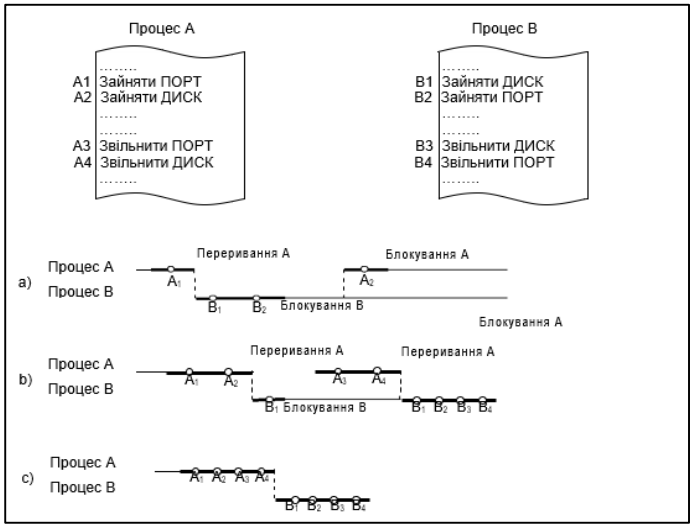

Рис. 2.5. Виникнення взаємного блокування процесів і програм

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

роздруківки інформації, що надходить модемним зв'язком.

Представимо фрагмент двох програм (рис. 2.5):

Залежно від співвідношення швидкостей процесів, вони можуть або взаємно блокувати один одного (дедлок а, б), або утворювати черги до поділюваних ресурсів, або зовсім незалежно використати поділювані ресурси (с).

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 запросити.

На рис. 2.6. а) зображено певний стан системи, а на б) – стан системи після зміни. Припустимо, процес С запросив ще два пристрої, і його задовольнили. Тоді, якщо один із процесів запросить максимально можливу для нього кількість пристроїв або більше одного для завершення, буде дедлок.

а)

Ім'я процесу	Максимальна потреба	Виділено	Залишок
A	4	2	2
B	6	3	3
C	8	2	6

б)

Ім'я процесу	Максимальна потреба	Виділено	Залишок
A	4	2	2
B	6	3	3
C	8	4	4

Рис. 2.6. Стани системи

Тому задовольняти запит С небезпечно.

Для визначення того, чи приведе задоволення запиту до небезпечного стану, існують спеціальні алгоритми. Один з них має назву “Алгоритм банкіра”.

Кожному процесу поставлено у відповідність ціле число i (i = 1 ÷ N). Процесу і відповідають його максимальні потреби в пристроях МАКС(i), кількість виділених йому в цей момент пристроїв – ОТРИМ(i), потрібний ще йому залишок пристроїв – ЗАЛИШОК(i). Також, ознака ЗАВЕРШ(i) = 1, якщо процес може завершитися успішно, і дорівнює 0 – якщо ні.

Система заводить глобальні змінні ЗАГПР, що визначають загальне число наявних у системі пристроїв, та ВІЛЬНПР – вільних на даний момент пристроїв. На початку роботи невідомо, чи може якийсь процес закінчитися МОЖЕ НЕ ЗАКІНЧИТИ(i):= truедля всіх i

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

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

ВІЛЬНПР:=ЗАГПР; for i:=1 to Ndo
Begin ВІЛЬНПР:=ОТРИМ(i);
ЗАЛИШОК(i):=МАКС(i)- ОТРИМ(i);
ЗАВЕРШ(i):=false {Процес може не завершитися} end;
flag: = true;{Ознака продовження аналізу} while flag do
Begin
flag:= false;
for i:=1 to N do Begin
if(not(ЗАВЕРШ(i)and(ЗАЛИШОК(i)<=ВІЛЬНПР) then
Begin ЗАВЕРШ(i):=true;
ВІЛЬНПР:=ВІЛЬНПР +ОТРИМ(i); flag:=true
end
end
end;
if ВІЛЬНПР = ЗАГПР then
(СТАН СИСТЕМИ БЕЗПЕЧНИЙ, МОЖНА ВИДАТИ ПРИСТРІЙ) else
(СТАН СИСТЕМИ НЕБЕЗПЕЧНИЙ, ПРИСТРІЙ ВИДІЛЯТИ НЕ МОЖНА) end.

Попередньо, при вивченні процесів вирішувались 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 їх на два класи:

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

У літературі їх умовно позначають як SR (System Resource) та CR (Consumable Resource).

Ці два види ресурсів характеризуються певними особливостями:

1. Системні ресурси SR:

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

Стосовно проблеми тупиків, то у якості SR ресурсів можуть розглядатися:

основна пам’ять, зовнішня пам’ять, периферійні пристрої, процесори, файли даних і т. п.

2. 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.

Використані одиниці ресурсу, у загальному випадку, не повертаються ресурсу, а використовуються.

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

Для вивчення проблеми тупиків, пов’язаних з вивченням паралельних процесів, розроблено декілька підходів. Одним 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 стан системи за допомогою запиту отримання або відмови від ресурсу.

У графічній формі процеси та ресурси позначають квадратиками та кружками відповідно.

Кожний кружок має деяку кількість маркерів, що відповідає числу одиниць цього ресурсу. Дуга, що йде від процесу до ресурсу, означає запит однієї одиниці цього ресурсу. Дуга, що йде від ресурсу до процесу, означає виділення ресурсу процесу.

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

Така модель зветься графом повторно використовуваних ресурсів. Наприклад:

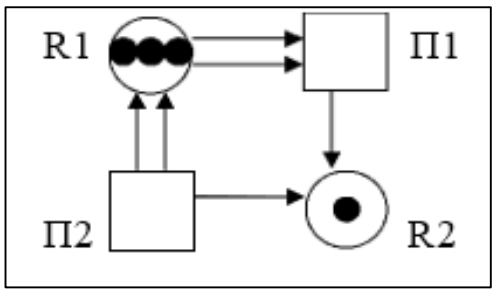

Рис. 2.7. Граф повторно використовуваних ресурсів

Нехай процес П2 запитує дві одиниці ресурсу R1 і одну одиницю ресурсу R2. Процесу П1 належать дві одиниці ресурсу R1, та йому потрібна одна одиниця ресурсу R2. Якщо процес П2 отримає тепер одиницю ресурсу R2, і прийнято правило, згідно з яким, для звільнення хоча б однієї одиниці якогось ресурсу він має отримати всі потрібні йому ресурси, це призведе до тупикової ситуації: П1 не зможе продовжити роботу, доки П2 не звільнить одиницю ресурсу R2, а П2 не звільнить одиницю R2, доки П1 не поверне дві одиниці R1. Ясно, що причиною такого дедлоку є неупорядковане використання ресурсів. ***Запобігання тупиків*** зумовлено дуже великою вартістю такого стану, тому він має бути виключений. Запобігання треба розглядати як заборону виникнення цих станів.

Запобігання тупиків має гарантувати, що жодна з нижче перерахованих ситуацій не зможе виникнути:

1. ***Ситуація взаємного виключення***. Можна запобігти шляхом дозволу необмеженого поділу ресурсів. Це легко виконати для програм із повторним входом, але зовсім не підходить до сумісно використовуваних змінних у критичних інтервалах.

2. ***Ситуація очікування***. Можна запобігти попереднім виділенням ресурсів. При цьому процес може почати виконання тільки після отримання усіх необхідних ресурсів заздалегідь. Тому значна кількість замовлених паралельними процесами ресурсів повинна не перевищувати можливостей системи. Складнощі виникають ще й тоді, коли попереднє виділення ресурсів досить часто є неможливим, оскільки необхідні ресурси стають відомими процесу тільки після початку його виконання.

3. ***Ситуація відсутності перерозподілу***. Для цього в ОС повинен бути передбачений механізм відбирання і перерозподілу ресурсів. Перерозподіл деяких ресурсів, наприклад, процесорного часу, достатньо легко зробити, проте перерозподіл, наприклад, периферійних пристроїв досить проблематичний.

4. ***Ситуація кругового очікування***. Цю умову можна виключити забороною утворення ланцюгів запитів, що забезпечується шляхом ієрархічного виділення ресурсів. Всі ресурси мають деяку ієрархію. Процес, що запросив ресурси на одному рівні, згодом може запросити ресурси тільки на більш високому рівні. Він може звільнити ресурси на даному рівні тільки після вивільнення усіх ресурсів на всіх більш високих рівнях. Тільки після того, як процес отримав, а потім звільнив ресурси даного рівня, він може запросити ресурси на тому ж самому рівні.

Наприклад, є два види ресурсів: R1 та R2, причому R2 знаходиться на більш високому рівні. Є також 2 процеси П1 і П2. Якщо П1 захопив R1, то П2 не зможе захопити R2, оскільки доступ до нього проходить через R1, який вже

отримав П1. Таким чином, замкнене коло виключається.

Обхід тупиків. Його можна інтерпретувати як заборону входження у небезпечний стан. Приклад – алгоритм банкіра. Цей алгоритм має багато недоліків:

1. Вважається, що кількість ресурсів, що підлягають поділу, є сталою величиною.

2. Вимагає попередньої вказівки процесами своїх максимальних потреб у ресурсі. Часто це неможливо.

3. Кількість працюючих процесів має залишатись незмінною. Це нереально, особливо у мультипрограмних системах.

Знаходження тупика за допомогою редукції графа повторно використовуваних ресурсів.

1. Граф повторно використовуваних ресурсів зменшується процесом Пі, який не є ані заблокованим, ані ізольованою вершиною, шляхом видалення усіх дуг, що входять та виходять з Пі. Ця процедура еквівалентна отриманню процесом Пі деяких ресурсів, на які він раніше видавав запити, а потім звільняв усі ресурси. Тоді вершина Пі стає ізольованою.

2. Граф повторно використовуваних ресурсів не редукується (не скорочується), якщо він не може бути скороченим жодним процесом.

3. Граф ресурсів типу SR є повністю скорочуваним, якщо існує послідовність скорочень, яка видаляє усі дуги графу.

Існує лема. Для ресурсів SR порядок скорочень є несуттєвим. Послідовності призводять до одного й того ж нескорочуваного у подальшому графу.

Теорема про тупик. Стан S є станом тупику тоді й тільки тоді, якщо граф повторно використовуваних ресурсів у стані S не є повністю редукованим.

Висновок 1. Процес Пі не знаходиться у тупику тоді й тільки тоді, коли деяка серія скорочень призводить до стану, у якому Пі не заблоковано.

Висновок 2. Якщо S є станом тупику з ресурсами SR, то, у крайньому випадку, два процеси знаходяться у тупику в S.

Із цієї леми безпосередньо випливає алгоритм пошуку тупику. Треба просто спробувати скоротити граф як можна ефективніше. Якщо граф повністю не скорочується, то стан, що був до скорочення, був станом тупику для тих процесів, вершини яких залишились у нескорочуваному графі.

Приклад тупику з ресурсами типу SR розглянуто. Тепер розглянемо приклад тупику з ресурсами CR.

Нехай існують три процеси П1, П2 та П3, які генерують повідомлення М1, М2 та М3. Ці повідомлення не що інше, як CR ресурси. Нехай П1 “користувач” отримує повідомлення М3, процес Р2 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), як на рис. 2.8.

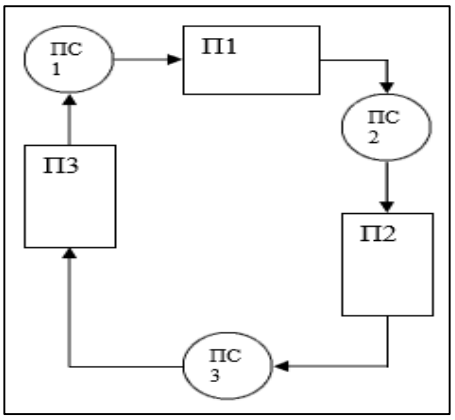

Рис. 2.8. Приклад тупику з ресурсами CR

Якщо цей зв’язок за допомогою повідомлень відтворюється відповідно до наступного порядку:

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)
……..

тоді все гаразд. Проте, переставивши місцями ці дві процедури, прийдемо до тупику:

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 у розпорядженні є 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) звільнення 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 в очікуванні 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 спеціально вивчати цей предмет, то знадобилося б дуже багато часу. Тому зупинимось на найбільш розповсюдженій моделі – сітці Петрі (окрім вже розглянутої моделі Холта).

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

Взаємодію подій у паралельних асинхронних дискретних системах описують як ситуації, при яких деяка подія може статися. При цьому глобальні ситуації у системі формуються за допомогою локальних операцій, які називають 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 сітки Петрі був використаний Холтом. Існує декілька формальних представлень мереж Петрі:

- теоретично-множинне;
- графова – біхроматичне (орієнтований граф);
- матричне.

Ці сітки можуть бути використані з точки зору аналізу системи на можливість виникнення тупикових ситуацій. Цей аналіз використовується за допомогою дослідження простору можливих станів мережі. Й найчастіше для цього використовують графову модель. Цей підхід базується на побудові редукованого 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 такого дерева (за винятком початкової), то шлях до цієї вершини від кореня дерева (від початкового маркування до заданого маркування) буде послідовністю виконання переходів.

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

Сітку Петрі називають “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 (досягнута бажане кінцеве маркування) або ж зависла (має місце тупикова ситуація).

У графічному представленні сітки (рис. 3.5), переходи зображено вертикальними або горизонтальними “лініями”, а позиції – “кружечками”. Умови – позиції та події-переходи, пов’язані відношеннями безпосередньої залежності (безпосереднього причинно-наслідкового зв’язку), як зображено за допомогою направлених дуг, що ведуть з позицій до переходів і переходів до позицій. Позиції, з яких ведуть дуги на даний перехід, звуть вхідними позиціями, а позиції, на які ведуть дуги з даного переходу – вихідними позиціями.

Виконання умови здійснюється розміткою відповідної позиції, власне

поміщенням числа N або зображенням N маркерів (фішок) у те місце, де N > 0 – ємкість умови.

Переміщення маркерів сіткою здійснюється за допомогою виконання її переходів. Виконання збуреного переходу 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.

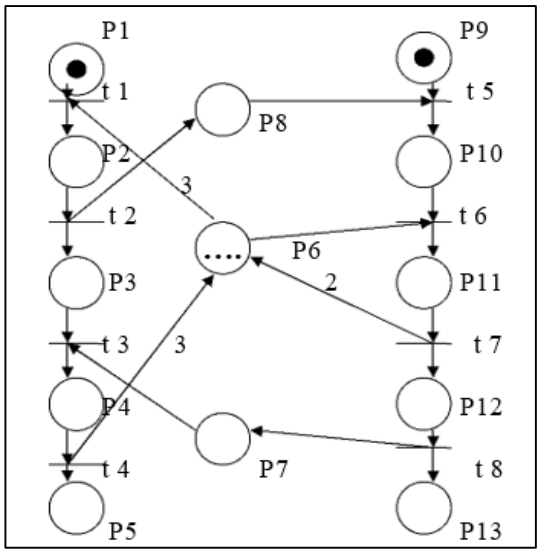

Рис. 2.9. Приклад сітки Петрі для двох взаємодіючих процесів

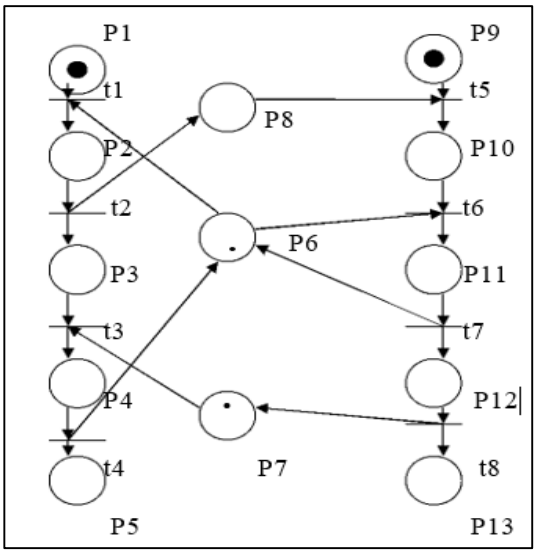

Рис. 2.10. Приклад тупикової ситуації

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

Початкове маркування (1, 0, 0, 0, 0, 4, 0, 0, 1, 0, 0, 0, 0). Тут позиція P2 означає, що P1 отримав три одиниці ресурсу R. Дуга, що поєднує позицію P6 (число в ній відповідає кількості доступних одиниць ресурсу R), має вагу 3, при спрацюванні переходу t1 процес P1 отримає три одиниці ресурсу. Перехід t2

відповідає відправленню повідомлення P2. Перехід t5 – прийом. Поява маркеру у позиції P7 означає, що P2 обробив та послав відповідь P2. Виконання переходу t4 – це повернення 3 одиниць ресурсу, якими володів P1.

Сітка на рис.2.10. не буде “живою”, бо в ній будуть “мертві” переходи: t2, t3, t6, t7, t8 (тупикова ситуація).

Розмітка M сітки – це функція, що ставить у відповідність міткам позицій невід’ємні цілі числа. Розмітка приписує кожній позиції деяку кількість міток відповідно до функції розмітки.

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

Розмітку M можна представити як вектор з n(P) елементів, у якому i-ий елемент визначає кількість міток в i-ій позиції.

Розмітка змінюється за умови запуску переходу. Перехід запускається, якщо для кожної позиції x, з якої стрілка вказує на t, існує хоча б одна мітка (якщо стрілка одинична).

Розглянемо роботу з сіткою Петрі. Сітка Петрі, як на рис. 2.11.

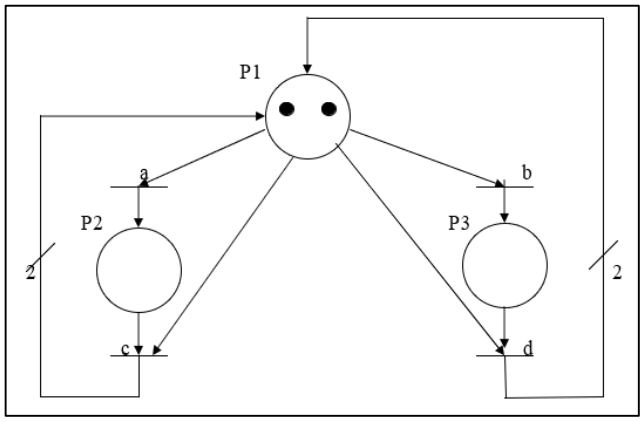

Рис. 2.11. Приклад сітки Петрі

На рис. 2.11. переходи a та b можна запустити, M = [2,0,0]. Якщо зробити послідовність запусків a, b, отримаємо: M = [1,1,0]. Тепер можна запустити переходи b, c. Якщо запустити c, отримаємо M = [2,0,0]. Можна, наприклад, запустити a та b одночасно, тоді буде тупик. M = [0,1,1]. Далі “живих” переходів немає. Якщо запустити b, буде M = [1,0,1]. Далі запускаємо d, отримуємо M = [2,0,0]. Мережа “жива”.

Питання до самоконтролю:

1. Основна одиниця роботи в операційній системі.

2. Відмінність відповідності між процедурами й виконуваними роботами у ОС.

3. Паралельність роботи периферійних пристроїв і центрального процесора.

4. Формальне визначення процесу через простір станів, функцію дії та початковий стан.

5. Різниця між процесом і потоком виконання.

6. Критерії ефективності обчислювальних систем: пропускна здатність, зручність користувачів, реактивність.

7. Види операційних систем: пакетної обробки, поділу часу, реального часу.

8. Симетричні та асиметричні мультипроцесорні системи й їх особливості.

9. Види ресурсів системи та політика запобігання тупікам.

МОДУЛЬ 3. УПРАВЛІННЯ ПАМ’ЯТТЮ В ОПЕРАЦІЙНИХ СИСТЕМАХ ТА РОБОТА ПРИКЛАДНИХ ПРОГРАМ

План

1. Віртуальна пам’ять та пам'ять прикладних програм.
2. Технології розподілу пам’яті

3.1. Віртуальна пам’ять та пам'ять прикладних програм.

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

Особлива роль пам'яті полягає 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;
- витіснення кодів і даних процесів з оперативної пам'яті на диск, коли розміру основної пам'яті недостатньо для розміщення в ній всіх процесів, і повернення їх в оперативну пам'ять, коли в ній звільняється місце;
- настроювання адреси 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 й т.п. 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. Воно полягає в тому, щоб не дозволити виконуваному процесу записувати або читати дані з пам'яті, призначеної іншому процесу.

Для ідентифікації змінних і команд на різних етапах життєвого циклу програми використовуються символьні імена, віртуальні адреси й фізичні адреси (рис. 3.1):

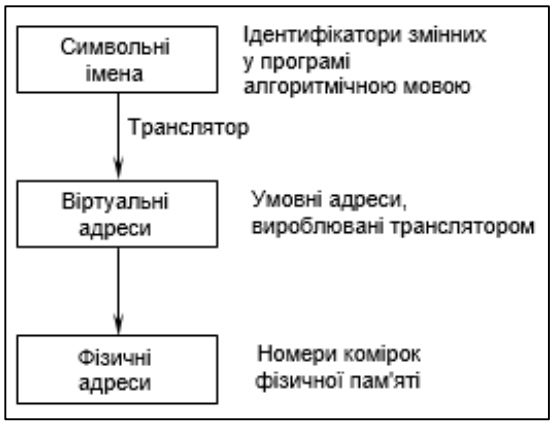

Рис. 3.1. Типи адрес

• **Символьні імена** надає користувач при написанні програми.

• **Віртуальні (умовні) адреси***,* виробляє транслятор, що перетворює програму на машинну мову.

• **Фізичні адреси** відповідають номерам комірок оперативної пам'яті, де в дійсності розташовані змінні й команди.

Сукупність віртуальних адрес називають **віртуальним адресним простором**. Діапазон можливих адрес віртуального простору у всіх процесів є однаковим. Проте, кожен процес має власний віртуальний адресний простір - транслятор привласнює віртуальні адреси змінним і кодам кожної програмі незалежно.

У різних ОС використовуються різні способи структуризації адресного простору:

1. ***Лінійна*** послідовність віртуальних адрес. Таку структуру адресного простору називають також **пласкою (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 поділяють на частині, називані сегментами. Віртуальна адреса представляє собою **пари** чисел (**n**, **m**)*,* де **n** визначає сегмент, а **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 у пам'ять. Виконує це системна програма – *переміщуючий* **завантажувач.** На підставі наявних у неї вихідних даних про початкову адресу фізичної пам'яті, а також інформації, наданої транслятором про адресно-залежні елементи програми, він виконує

завантаження програми, поєднуючи із заміною віртуальних адрес фізичними.

2. Програма завантажується у пам'ять 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 завантажувач жорстко прив'язує програму до початку виділеної їй ділянки пам'яті, динамічне перетворення віртуальних адрес дозволяє переміщати програмний код процесу протягом усього періоду його виконання.

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

віртуального адресного простору ОС використовує програмну зовнішню пам'ять. Однак, співвідношення обсягів віртуальної й фізичної пам'яті може бути й зворотним.

Варто пам'ятати, що, у загальному випадку, механізми віртуального адресного простору і віртуальної пам'яті – це не одне й теж саме. Можна уявити собі ОС, у якій підтримуються віртуальні адресні простори для процесів, але відсутній механізм віртуальної пам'яті. Це можливо тільки у 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 простого 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 неперервні частини: системну й користувацьку. У деяких ОС (наприклад, Windows NT), ці частини мають однаковий розмір – по 2 Гбайти.

Частина віртуального адресного простору кожного процесу, що відводять під сегменти ОС, є ідентичною для всіх процесів. Тому при зміні активного процесу заміняється тільки друга частина віртуального адресного простору. Наприклад, у процесорах Intel Pentium існують 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.

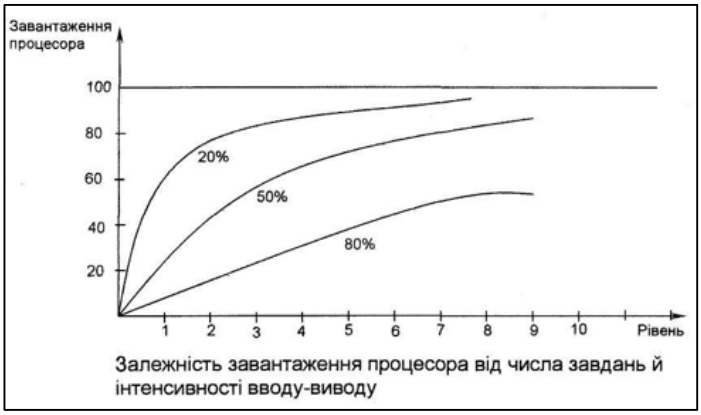

Рис. 3.2. Завантаження процесора

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 призупинені процеси, що очікують завершення введення-виведення або звільнення ресурсів, а також процеси у 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 надходить 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 називають ресурс, який представляється користувачу або користувальницькій програмі таким, що володіє властивостями, якими він насправді 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 і включає рішення наступних завдань:

- розміщення даних у запам'ятовуючих пристроях різного типу;
- вибір образів процесів або їхніх частин для переміщення з оперативної пам'яті на диск і зворотно;
- переміщення в міру необхідності даних між пам'яттю й диском;
- перетворення віртуальних адрес у фізичні.

Зауважимо, що з проблемою розміщення в оперативній пам'яті програм, розміри яких перевищують розміри оперативної пам'яті, програмісти зіштовхнулися досить давно.

Одним із рішень було розбиття програми на частини, які називали **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 містилися у програмі.

Незважаючи на зовнішню подібність, ця процедура має принципову відмінність від віртуальної пам'яті. Вона полягає в тому, що при оверлеї розбиття 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 *повністю;*
- **віртуальна пам'ять** - між оперативною пам'яттю й диском переміщаються *частини* (сегменти, сторінки) образів процесів.

Свопінг – окремий випадок віртуалізації. Це найбільш простий спосіб спільного використання оперативної пам'яті й диску.

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 повністю вивантажувати інший процес на диск. 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 стопінг, 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** адрес у фізичні.*

У цей час вся безліч реалізацій віртуальної пам'яті може бути представлена трьома наступними 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**. Використовується дворівневий поділ: віртуальний адресний простір ділять на сегменти, а потім сегменти ділять на сторінки. Одиницею переміщення тут є сторінка.

Для тимчасового зберігання сегментів і сторінок на диску виділяють або спеціальну ділянку, або спеціальний файл, що називають сторінковим файлом.

Поточний розмір сторінкового файлу є важливим параметром ОС – чим він більше, тим більше програм ОС може одночасно виконувати. 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 настроюється.

3.2. Технології розподілу пам’яті

Алгоритми розподілу пам'яті ділять 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 (рис. 3.3).

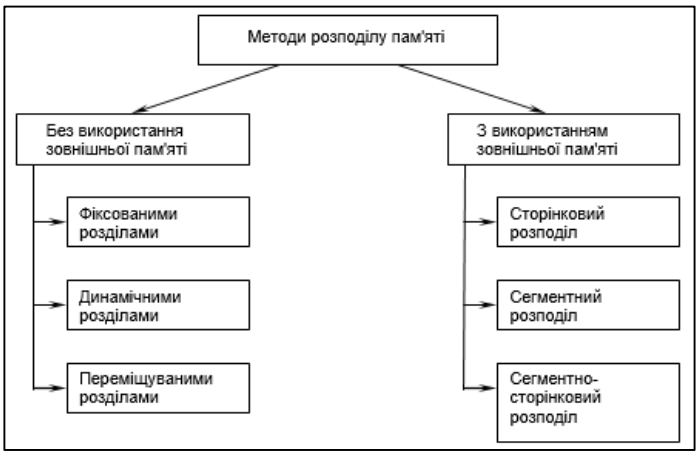

Рис. 3.3. Класифікація методів розподілу пам’яті

Розподіл пам'яті фіксованими розділами

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

Це розбиття може бути виконано вручну оператором під час старту системи або її установки. Після цього границі розділів не змінюються.

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

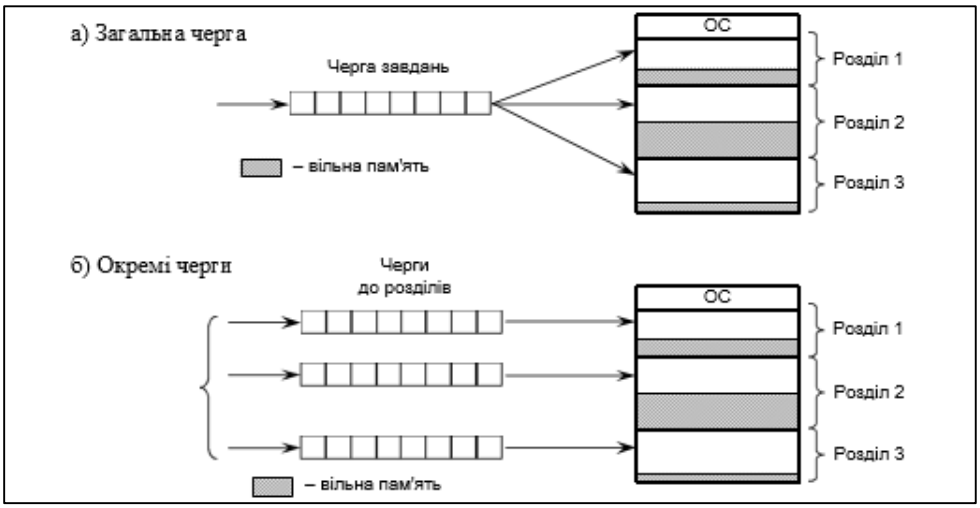

Рис. 3.4. Розподіл пам'яті фіксованими розділами

Система керування пам'яті вирішує наступні завдання:

• Порівнює обсяг пам'яті, необхідний для нового процесу, з розмірами вільних розділів і вибирає підходящий розділ.

• Здійснює завантаження програми в один з розділів і настроювання адрес.

Уже на етапі трансляції розроблювач програми може задати розділ, у якому її варто виконувати. Це дозволяє відразу, без використання переміщуючого завантажувача, одержати машинний код, настроєний на конкретну область пам'яті. Істотний недолік – жорсткість.

Рівень мультипрограмування заздалегідь обмежений числом розділів. Незалежно від розміру програми, вона буде займати весь розділ; з іншого боку, 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 процесу, що надходить на виконання, виділяється вся необхідна йому пам'ять; якщо її не вистачає, процес не запускається.

Для реалізації методу потрібні наступні функції ОС:

• 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 знайдений розділ достатнього розміру; розділ з найменшим достатнім розміром; розділ з найбільшим достатнім розміром.

• Завантаження програми у виділений їй розділ і коригування таблиць вільних і зайнятих областей.

• Коригування таблиць вільних і зайнятих областей.

Цьому методу властивий серйозний недолік – фрагментація пам'яті. Фрагментація – наявність великої кількості несуміжних ділянок вільної пам'яті маленького розміру.

Такого, що жодна з програм, що надходять на виконання, не може помістится в жодній з ділянок, хоча сумарний обсяг вільних фрагментів може скласти величину, що перевищує необхідний програмі обсяг пам'яті. Прикладом застосування цього методу є популярна в минулому OS/360 й ЕС ЕОМ.

Переміщувані розділи

Боротьба із фрагментацією здійснюється шляхом переміщення всіх зайнятих ділянок убік старших або молодших адрес таким чином, щоб вся вільна пам'ять утворила єдину вільну область (рис. 3.5).

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, коригуючи таблиці вільних і зайнятих областей. Цю процедура називають **стискуванням.**

Стискування виконується або при завершенні кожного процесу, або у випадку відсутності для створюваного процесу вільного розділу достатнього розміру.

Оскільки програми переміщюються в оперативній пам'яті в процесі свого виконання, то, у цьому випадку, неможливо виконати настроювання адрес за допомогою переміщуючого завантажувача. Тут необхідна реалізація динамічного перетворення адрес.

Хоча цей метод і призводить до більш ефективного використання пам'яті, він вимагає значних часових витрат, що нівелює переваги даного методу. Метод використовувся в ранніх версіях ОС OS/2.

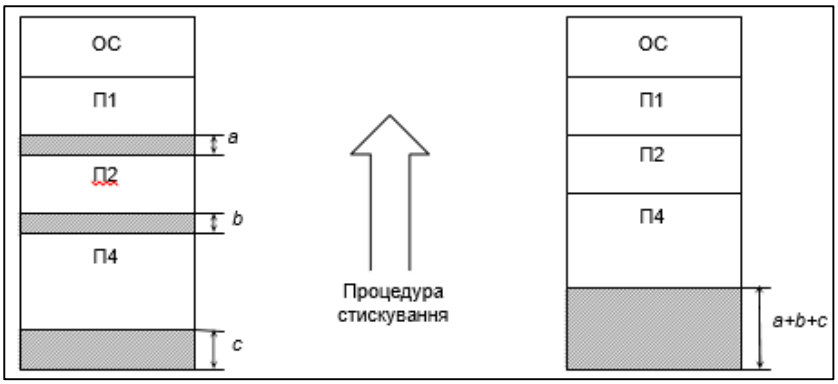

Рис. 3.5. Розподіл пам'яті переміщуваними розділами

Свопінг і віртуальна пам'ять. Обсяг оперативної пам'яті, що є в комп'ютері, істотно позначається на протіканні обчислювального процесу.

Він обмежує кількість одночасно виконуваних програм і розміри їх віртуальних адресних просторів. Якщо всі завдання мультипрограмної суміші є

обчислювальними (мало операцій введення-виведення), для оптимального 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 то й сотень завдань.

Сторінковий розподіл

Віртуальний адресний простір кожного процесу ділиться на частини однакового, фіксованого для даної системи розміру – **віртуальні сторінки** (рис. 3.6).

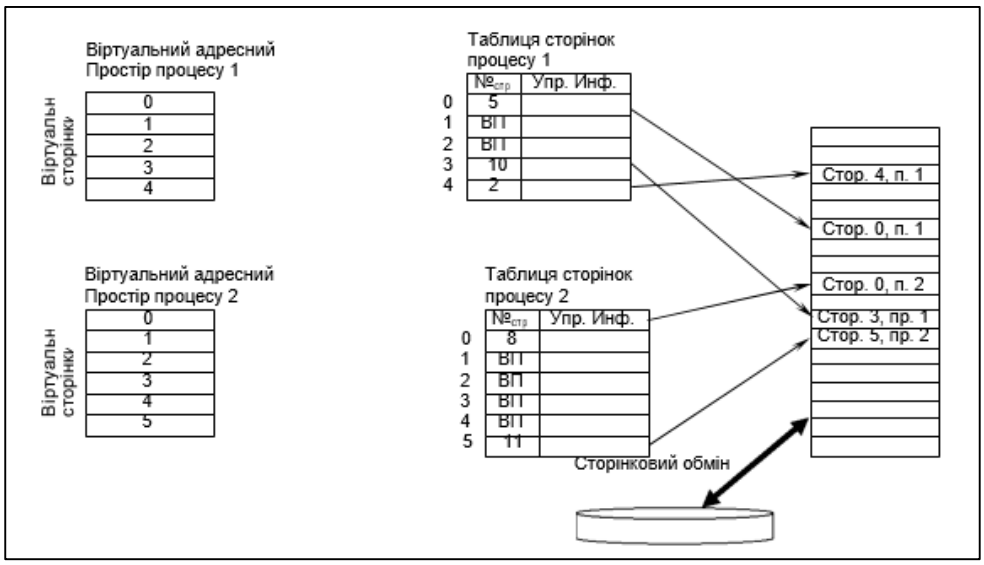

Рис. 3.6. Сторінковий розподіл

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 частині такого ж розміру – **фізичні сторінки**. Розмір сторінки вибирається рівним степеням двійки: 512, 1024, 4096 байт і т.д, Це дозволяє спростити механізм перетворення адрес.

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

Запис таблиці має назву **дескриптора сторінки.** Він містить 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 щоразу, коли здійснюється запис за адресою, що відповідає даній сторінці;
- ознака звертання до сторінки, називана також бітом доступу, що

встановлюється в одиницю при кожному звертанні за адресою, що відповідає даній сторінці.

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

Таблиці сторінок розміщують 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 за цим номером визначається потрібний елемент таблиці сторінок, і з нього читається інформація, що описує сторінку. Далі аналізується ознака присутності, і, якщо сторінка перебуває в оперативній пам'яті, 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 саме 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 вона 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 ($\mathbf{p, s_v}$), де $\mathbf{p}$ – порядковий номер віртуальної сторінки процесу (нумерація сторінок починається з 0), а $\mathbf{s_v}$ – зсув у межах віртуальної сторінки.

Фізична адреса також може бути представлена у вигляді пари ($\mathbf{n, S_f}$), де $\mathbf{n}$ – номер фізичної сторінки, а $\mathbf{S_f}$ – зсув у межах фізичної сторінки. Завдання підсистеми віртуальної пам'яті полягає у перетворенні пари ($\mathbf{p, s_v}$) в ($\mathbf{n, S_f}$).

Розглянемо дві базисних властивості сторінкової організації.

Перша з них полягає в тому, що обсяг сторінки обирається рівним степені

двійки – $\mathbf{2^k}$. Із цього випливає, що зсув **s** може бути отримано простим відділенням **k** молодших розрядів у двійковому записі адреси, а решта старших розрядів адреси являє собою двійковий запис номера сторінки.

Наприклад:

Розмір сторінки 1 Кбайт (2^{10}). Двійкова адреса $\mathbf{1\,000\,111\,001_2}$. Вона належить сторінці 102 і зміщена відносно її початку на $1\,000\,111\,001_2$ байт. **Друга** властивість полягає в тому, що в межах сторінки неперервна послідовність віртуальних адрес однозначно відображається у неперервну послідовність фізичних адрес, тобто $s_v = S_f$.

Рис. 3.7. Перетворення віртуальних сторінок на фізичні

Пам'ять ЕОМ – це ієрархія запам’ятовуючих пристроїв (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 диску обчислюється мілісекундами. Оперативна пам'ять має час доступу 10-20 наносекунд (від декількох мегабайт до декількох гігабайт)

Надоперативна пам'ять (десятки - сотні кілобайт) – 8 нсек. Внутрішні регістри процесора – кілька десятків байт з часом доступу 2-3 наносекунди.

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

Особливістю кешування є те, що система не вимагає ніякої зовнішньої інформації про інтенсивність використання даних. Ані користувачі, ані

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'ятовуючих пристроїв, але й один із пристроїв – «швидкий» ЗП. Він дорожчий й порівняно невеликого обсягу на противагу «повільному» ЗП – оперативній 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 **зберігаються на диску,** роль кеш-пам'яті виконують буфери, реалізовані в оперативній пам'яті. **Віртуальну пам'ять** теж можна розглядати як окремий випадок **кешування**.

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 елемента має в основній пам'яті;
- додаткову інформацію, що використовується для реалізації алгоритму та містить ознаку модифікації й ознаку дійсності даних.

При кожному звертанні до основної пам'яті за фізичною адресою 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 за вмістом – за взятим із запиту значенням поля адреси в оперативній пам'яті.

Далі можливі два варіанти розвитку (рис. 3.8):

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

Ефективність кешування залежить від ймовірності влучення в кеш. Якщо позначити ймовірність кеш-влучання через p, а час доступу до основної пам'яті через t1, час доступу до кеш через t2, то за формулою повної ймовірності середній час доступу буде дорівнювати: $t=t2p+t1(1-p)$

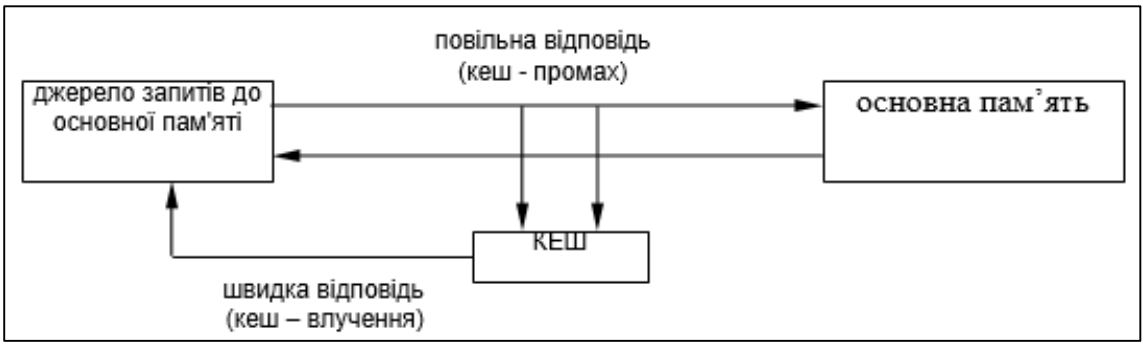

Рис. 3.8. Принцип кешування

Якщо p=1, час доступу дорівнює t2.

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

- обсяг кешу;
- обсяг кешуємої пам'яті;
- алгоритму заміщення даних у кеші;
- особливостей виконуваної програми й т.п.

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

Просторова локальність. Якщо відбулося звертання за деякою адресою, то з високою ймовірністю найближчим часом відбудеться звертання за сусідніми адресами.

Часова локальність. Якщо відбулося звертання за деякою адресою, то наступне звертання за тією ж адресою з великою ймовірністю відбудеться найближчим часом.

Оскільки при виконанні програми дуже висока ймовірність, що команди вибираються з пам'яті одна за одною 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. Витіснення даних означає або просто оголошення вільною деякої області кеш-пам'яті (скидання біта дійсності), якщо дані не змінювалися, або, на додаток до цього, копіювання даних в основну пам'ять, якщо вони були модифіковані.

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

Існують два підходи до вирішення цієї проблеми:

- наскрізний запис (write through). При запиті до основної пам'яті (у тому числі при записі) переглядають кеш. Якщо дані за запитуваною адресою відсутні, запис виконується тільки в основну пам'ять. Якщо дані перебувають у кеші, запис робиться у кеш й у пам'ять.
- зворотний запис (write back). Виконується перегляд кешу, якщо даних

там немає, то запис робиться в основну пам'ять. У противному випадку, запис робиться тільки в кеш. При цьому встановлюється ознака модифікації. При витісненні даних з кешу вони будуть переписані в основну пам'ять.

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

- випадкове відображення;
- детерміноване відображення.

При випадковому відображенні елемент оперативної пам'яті може бути розміщений у довільному місці кеш пам'яті. Він розміщується там разом зі своєю адресою в оперативній пам'яті. Пошук інформації здійснюється за цією адресою. Процедури простого перебору адрес потребують великих часових витрат.

Тому використовується асоціативний пошук, при якому порівняння виконується не послідовно з кожним записом у кеші, а паралельно з усіма його записами. Ознака, за якою виконується порівняння, називають **тегом** (tag). У даному випадку, тегом є адреса даних в оперативній пам'яті.

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

У кешах на основі випадкового відображення витіснення старих даних відбувається тільки в тому випадку, коли вся кеш пам'ять заповнена і немає вільного місця. Вибір даних на вивантаження ґрунтується на тих самих 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 одне й те ж 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 декілька (досить багато) адрес оперативної пам'яті.

Як функцію відображення можна використовувати просте виділення декількох розрядів з адреси оперативної пам'яті, які інтерпретуються як номер рядка кеш-пам'яті. Таке відображення називають **прямим** (рис. 3.9). Наприклад, кеш розраховано на 1024 записи (1024 рядків). Тоді будь-яка адреса оперативної пам'яті може бути відображена на адресу кеш пам'яті простим відділенням 10 двійкових розрядів.

При пошуку даних у кеші використовується швидкий прямий доступ до запису за номером рядка, отриманим 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 збігу, констатується кеш-промах, і дані зчитуються з ОП і копіюються в кеш.

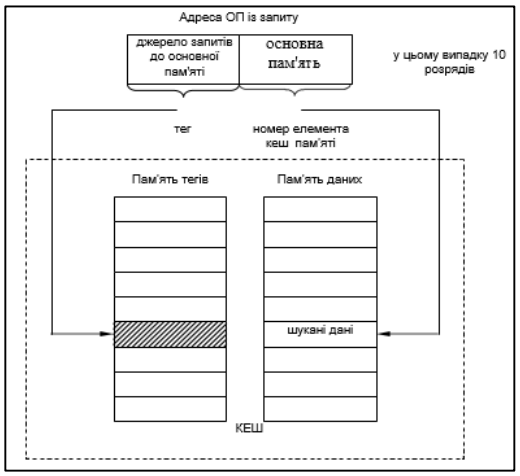

Рис. 3.9. Пряме відображення пам’яті на кеш

У багатьох сучасних процесорах кеш пам'ять будується на основі поєднання цих двох підходів, що дозволяє знайти певний компроміс у швидкодії.

При **змішаному підході** (рис. 3.10) довільна адреса оперативної пам'яті відображається не на одну адресу кеш пам'яті (як це характерно для прямого відображення) і не на довільну адресу кеш пам'яті (як це робиться при випадковому відображенні), а на певну групу адрес.

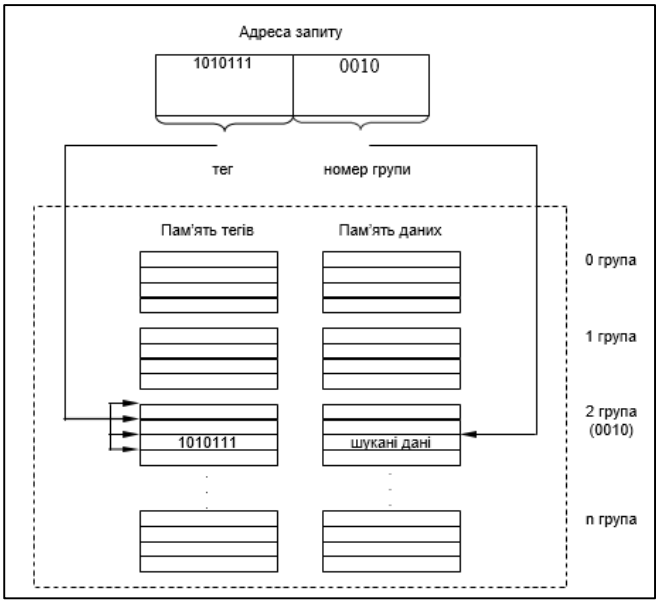

Рис. 3.10. Комбінування прямого й випадкового відображення пам’яті у кеш

Всі групи пронумеровані. Пошук у кеші здійснюється спочатку за номером групи, отриманим з адреси ОП із запиту, а потім у межах групи шляхом асоціативного перегляду всіх записів групи на предмет збігу старших

частин адрес ОП.

При промаху дані копіюються за будь-якою вільною адресою з визначеної групи. Якщо вільних адрес у групі немає, то виконується витіснення даних. Оскільки кандидатів на вивантаження кілька – всі записи з даної групи – алгоритм заміщення може врахувати інтенсивність звертання до даних і, тим самим, підвищити ймовірність влучень у майбутньому.

Як бачимо, 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 чи 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 (рис. 3.11).

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

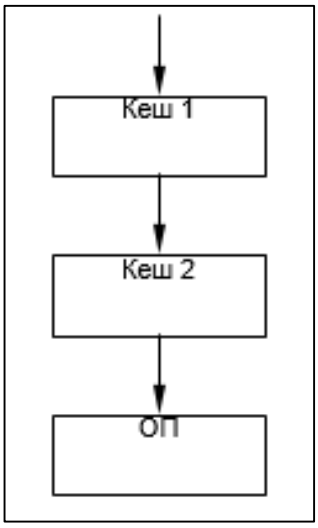

Рис. 3.11. Дворівневе кешування

Спочатку робиться спроба виявити дані в кеші першого 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.

Зауважимо, що 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.

Питання до самоконтролю:

1. Визначення оперативної пам'яті та її відмінності від зовнішньої пам'яті.
2. Роль пам'яті у виконанні інструкцій процесором.
3. Розподіл пам'яті між модулями прикладних програм і ОС.
4. Основні функції ОС у керуванні пам'яттю в мультипрограмній системі.
5. Поняття захисту пам'яті та його призначення.
6. Відмінності між символьними іменами, віртуальними і фізичними адресами.
7. Сукупність віртуальних адрес і принцип їх виділення для процесів.
8. Способи структуризації віртуального адресного простору: лінійна, сегментна, багаторівнева.
9. Підходи до перетворення віртуальних адрес у фізичні: переміщуючий завантажувач і динамічне перетворення.
10. Розподіл віртуального адресного простору на системну і користувацьку частини.
11. Призначення вивантаження неактивних процесів на диск і роль свопінгу.
12. Відмінності між оверлеями і механізмом віртуальної пам'яті.
13. Основні методи організації віртуальної пам'яті: свопінг, сторінкова, сегментна, сегментно-сторінкова.
14. Принципи роботи кеш-пам'яті та визначення кеш-влучання і кеш-промаху.
15. Методи узгодження даних між кешем і основною пам'яттю: наскрізний і зворотний запис.

МОДУЛЬ 4. СИСТЕМНЕ ПРОГРАМУВАННЯ ВВЕДЕННЯ/ВИВЕДЕННЯ ТА ФАЙЛОВИХ СИСТЕМ

План

1. Завдання ОС по керуванню файлами й пристроями.
2. Основні поняття та концепція організації введення-виведення.
3. Режими керування введенням-виведенням.
4. Закріплення пристроїв. Загальні пристрої введення-виведення.
5. Основні системні таблиці введення-виведення.
6. Синхронне та асинхронне введення-виведення.
7. Мета і завдання файлової системи.
8. Логічна модель файлової системи.
9. Фізична організація файлової системи.
10. Фізична організація й адресація файлу.

4.1. 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 спектра 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.

4.2. Основні поняття та концепція організації введення-виведення

Складність реалізації введення-виведення полягає у тому, що перед проектувальниками постає задача підтримки широкого різноманіття пристроїв, з одного боку, а, з другого боку, – сприяння створенню 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 ж самі пристрої.

Тому завжди використовують наступний найголовніший принцип: операції ведення-виведення об’являють привілейованими і виконують тільки кодами даної ОС.

Для забезпечення цього принципу вводять режим супервізора та користувача. Як правило, виконання введення-виведення - це розподілювальні ресурси. Тому, у загальному випадку, їх використання потребує синхронізації, 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 друкувати свої результати, то, окрім проблеми звичайної синхронізації доступу, виникає додаткова проблема: як розділити між собою ті результати, які будуть друкуватися упереміш.

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 пересилання даних.

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

3) 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 ж операціями для потреб звичайних процесів. Системний код, що керує операціями введення-виведення, дуже щільно відпрацьовується для підвищення надійності та ефективного використання обладнання.

Тобто управління введенням-виведенням виконується операційною 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 запит виконаний відповідно до специфікацій та не має помилок, він обробляється. 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 надання його задачі, що стоїть першою у черзі на виконання.

4. При отриманні сигналів переривання від пристроїв введення-виведення ідентифікує їх та передає керування відповідній задачі обробки переривань.

5. Виконує передачу повідомлень про похибки введення-виведення.

6. Надсилає повідомлення про завершення операції введення-виведення процесу, що її запитав, та знімає його зі стану очікування введення-виведення, якщо процес очікував її завершення.

Якщо пристрій є ініціативним (це зовнішні пристрої, окрім стандартних пристроїв введення-виведення, наприклад, датчики), керування з боку 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.

4.3. Режими керування введенням-виведенням

Існують два основних режими введення-виведення:

1) режим обміну з опитуванням готовності пристрою введення-виведення;

2) режим обміну з привілеями.

Нехай керування виконує центральний процесор (рис.8.1). 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 з’єднання). Центральний процесор надає пристрою керування команду на виконання введення-виведення. Останній виконує команду, перетворюючи її у послідовність сигналів, зрозумілих пристрою введення-виведення.

Але швидкодія пристрою введення-виведення на декілька порядків менше ніж у процесора. Тому сигнал щодо 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 пір, доки останній не з’явиться, даремно

витрачається час процесора.

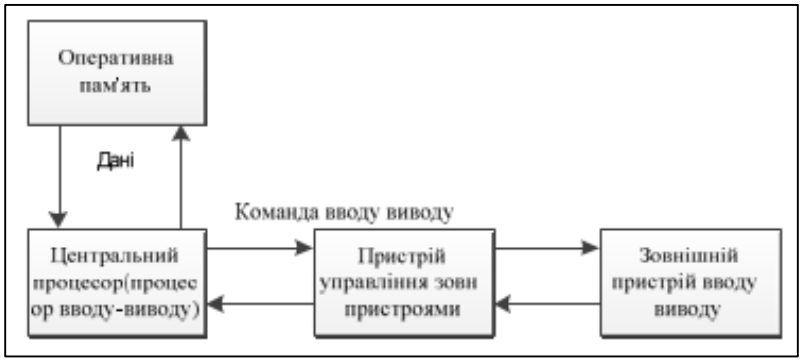

Рис. 4.1. Керування введенням-виведенням

Значно вигідніше тимчасово забути про пристрій введення-виведення після видачі команди введення-виведення. Саме ці сигнали готовності і будуть сигналами запиту на переривання.

Режим обміну з перериваннями у режимі асинхронного керування. Для того, щоб не загубити зв'язок з пристроєм після надання процесом чергової команди з керування обміном і переходу на виконання інших програм, можна організувати відлік часу, за який пристрій має виконати команду та видати сигнал переривання. Максимальне значення такого часу називають **установкою тайм-аута**. 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 вмикає пристрій введення-виведення або ініціалізує чергову ітерацію введення-виведення.

Секція продовження виконує основну роботу з передачі даних. Вона і є основним обробником переривання. У загальному випадку, використання інтерфейсу може вимагати декількох 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 введення-виведення або просто завершує операцію. Програми, що керують введенням-виведенням у режимі переривань, створювати досить складно. Тому, наприклад, драйвер для друку через паралельний порт в ОС Windows працює не в режимі переривань, а в режимі запитів готовності, що призводить до 100% завантаження процесора. Виконання інших завдань виконується винятково за рахунок витісняючих

стратегій, коли процесор примусово передається для виконання інших задач.

4.4. 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. До цього ж має відношення поняття спулінга, тобто імітації паралельного розподілу пристрою введення-виведення з послідовною програмою.

У цьому випадку, кожному процесу надається, наприклад для друку, не реальний, а віртуальний принтер, а потік символів, що виводяться, направляється не на пристрій друку, а у спеціальні спул-файли на диску. По закінченню віртуального друку, можна вивести цей файл на пристрій для друку. Системний процес, що керує спул-файлом, має назву **спулер.**

4.5. Основні системні таблиці введення-виведення

Кожна ОС має свої таблиці введення-виведення. Їх кількість та склад можуть дуже 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 використанні драйверами механізму 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, містить наступну інформацію:

- тип пристрою, конкретна модель, символьне ім’я та характеристика пристрою;
- через який інтерфейс підключено, до якого з’єднання, які порти та лінії запиту переривань використовуються;
- номер та адреса каналу, якщо вони використовуються для керування;
- вказівка на драйвер, який має керувати цим пристроєм, адреса секції запуску та секцій проводження;
- інформація про те, чи використовується буферизація при обміні даними з цим пристроєм; ім’я буферу, якщо він виділений у області пам'яті, що займає програма;
- установка тайм-аута та комірка для лічильника тайм-аута;
- стан пристрою;
- поле вказівника для чергу задач, що очікують пристрій; можливо ще

деякі дані.

Оскільки драйвери можуть мати властивості реєнтерабельності (один екземпляр програми може забезпечувати паралельне обслуговування декількох пристроїв одного типу), то у елементі UCB повинна зберігатись інформація про стан даного пристрою та самі змінні для реєнтерабельної обробки, або вказівку на адресу, де цю 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 дескрипторів тих задач, що очікують даний пристрій.

Друга таблиця визначена для реалізації принципу візуалізації. Бажано, щоб програміст міг 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 містять логічне ім’я пристрою, абстрагуючись від його конкретних особливостей та характеристик.

Конкретний пристрій, який відповідає віртуальному (логічному), обирає супервізор за допомогою цієї другої таблиці. Вона отримала назву – таблиця опису віртуальних пристроїв (DRT, device reference table). Тобто її призначення – 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 зворотного 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 таким чином, щоб він використовав дану лінію (сигнал) переривання. Взаємозв'язок між таблицями наведено на рис. 4. 2.

Запит на операцію введення-виведення від прикладної програми (1) надходить на супервізор (рис. 4.3). Він перевіряє системний виклик на відповідність специфікації та, у випадку похибки, повертає задачі відповідне повідомлення (1-1). Якщо запит коректний, то він відправляється на супервізор

введення-виведення (2).

Супервізор введення-виведення за логічним ім’ям і за допомогою таблиці логічних імен знаходить елемент UCB у таблиці обладнання.

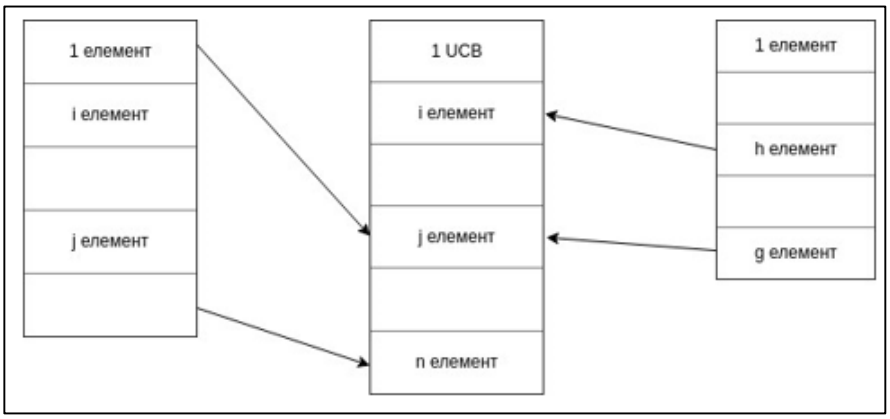

Рис. 4.2. Взаємозв’язок між таблицями введення-виведення

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 задачу (5).

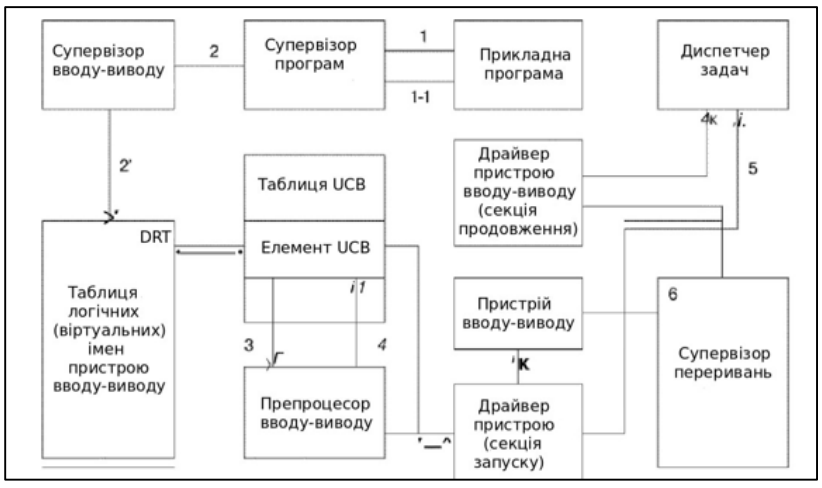

Рис. 4.3. Керування введенням-виведенням

Система продовжує працювати, але коли пристрій введення-виведення відпрацює відправлену йому команду, вона виставить сигнал запиту на переривання, за яким 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.

4.6. 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 про завершення операції, 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 збільшення швидкодії виконання процесів використовують асинхронний принцип введення-виведення.

Найпростіша реалізація цього принципу полягає у буферизації виводу. Тобто інформація для виведення передається не на пристрій безпосередньо, а у спеціальний системний буфер. У цьому випадку, логічна операція виведення для процесу вважається одразу 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 вивід буде асинхронним, якщо запит на введення-виведення мав вказівку щодо необхідності 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 ті 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 за запитом. Аналогічно, при записі. Буферизація використовується практично в усіх операційних системах.

Кешування дуже корисне у тому випадку, коли програма (процес) багаторазово читає з диску одні й ті ж 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, якими керує системний процес.

Якщо 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, принцип роботи такої кеш пам'яті не відрізняється від роботи звичайної кеш пам'яті, яку розглядали раніше. Процес кешування, як і вказувалось раніше – це спосіб сумісного використання двох запам’ятовуючих пристроїв із різною швидкодією.

Тому, як і звичайно, кешування дискових операцій може бути покращено за рахунок техніки упередженого читання, коли з диску зчитується значно більша кількість даних, ніж операція запрошувала.

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, максимальний обсяг – 32 Мбайт, об’єм буфера, яким оперує менеджер кешу, дорівнює об’єму одного сектора, тобто 512 байт.

У мультипрограмних системах при виконанні багатьох задач запити на читання та запис даних надходять таким потоком, що створюються черги. Якщо обслуговувати ці черги у порядку надходження у чергу, то, завдяки випадковому характеру звертань до тих чи інших секторів, мають місце великі

втрати часу на пошук. Тому, враховуючи те, що переупорядкування черги з метою зменшення непродуктивних втрат часу можливо виконувати відносно швидко, можна запропонувати деякі дисципліни обслуговування таких черг з метою економії непродуктивного використання часу. Наведемо деякі із них.

SSTF (Shortest Seek Time First) найменший час пошуку – перший. У відповідності з цією дисципліною? задовольняється запит, що потребує мінімальної кількості кроків для перепозиціонування записуючих голівок на диску.

SCAN (сканування). Згідно цієї дисципліни голівки зчитувача переміщуються то в одному, то в другому напряму, обслуговуючи на своєму шляху усіх, хто виставив вимоги і потрапляє на такому шляху.

NEXT Step Scan – відмінність 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 поверхні, обслуговуючи усі запити, що трапляються на шляху такого переміщення.

4.7. Мета і завдання файлової системи

Файл – це іменована область зовнішньої пам'яті, в яку можна записувати й з якої можна зчитувати дані. Звичайно файли зберігаються в енергонезалежній пам'яті – диску. Однак є й виключення. Основні цілі використання файлу:

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

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

Файлова система – частина ОС, що включає:

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

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

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

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

ЇЇ функції наступні:

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

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

У цьому випадку, файл – поділюваний ресурс із усіма проблемами, що звідси з'являються. У багатокористувацьких системах додається ще одне завдання – захист файлів від несанкціонованого доступу.

4.8. Логічна модель файлової системи

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

Типи файлів

Звичайні файли містять інформацію довільного характеру. Більшість сучасних ОС (UNIX, Windows, OS/2) ніяк не обмежують і не контролюють вміст і структуру файлу. Всі ОС повинні розпізнавати хоча б один тип файлів – їх власні виконувані файли.

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

У багатьох ОС у каталог можуть входити **спеціальні файли** – це фіктивні файли, що асоціюються із пристроями вводу–виводу. Вони використовуються з метою уніфікації механізму доступу до файлів і зовнішніх пристроїв.

Спеціальні файли дозволяють користувачеві виконувати операції введення-виведення за допомогою звичайних команд запису у файл або читання з файлу. Ці команди обробляються спочатку програмами файлової системи, а потім на деякому етапі виконання запиту перетворюються ОС у команди керування відповідним пристроєм.

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. 4.4).

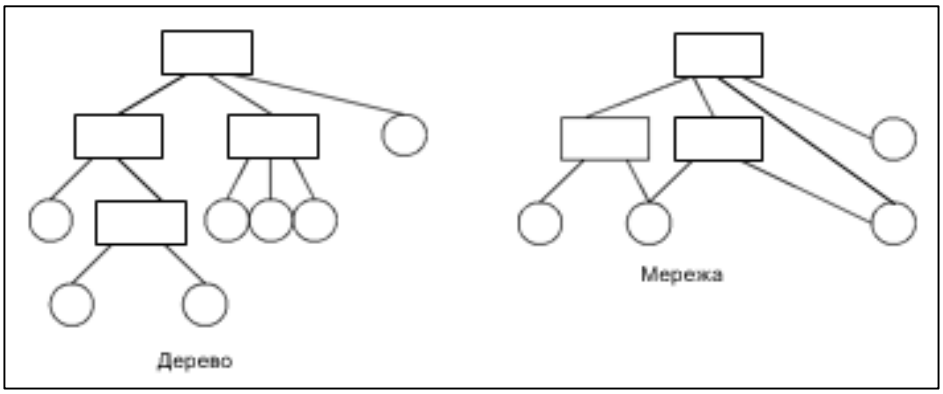

Рис. 4.4. Графи ієрархії каталогів

Граф, описуючий ієрархію каталогів, може бути деревом або мережею. Якщо файлу дозволено входити тільки в один каталог, файли утворять дерево. Якщо мережа – файл може входити відразу у декілька каталогів.

Наприклад, в MS DOS й Windows каталоги утворюють деревоподібну структуру, а в UNIX – мережну. У деревоподібній структурі кожен **файл** є **листом.** каталог самого верхнього рівня називається **кореневим каталогом** або **коренем**.

Імена файлів

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

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

В ієрархічних файлових системах різним файлам дозволено мати однакові прості символьні імена за умови, що вони належать різним каталогам. Для однозначної ідентифікації в таких системах використовується так зване **повне ім'я.**

Повне ім'я являє собою ланцюжок, який є шляхом від кореня до даного файлу. У деревоподібній файловій системі між файлом і його повним ім'ям є взаємно однозначна відповідність **один файл – одне повне ім'я**. У випадку мережної структури має місце відповідність: **один файл – багато повних імен.**

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

Монтування

Обчислювальна система може мати кілька дискових пристроїв. Більш

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

Перше рішення. На кожному із пристроїв розміщається автономна файлова система. Тобто є два незалежних дерева каталогів. Тут у повне ім'я файлу входить ідентифікатор відповідного логічного диску. **Друге рішення.** Файлові системи поєднуються в єдину файлову систему, що описується єдиним деревом каталогів. Така операція називають **монтуванням.**

При цьому ОС виділяє один дисковий пристрій, який називають **системним**. Нехай є дві файлові системи, розташовані на різних логічних дисках, причому один з них є системним. Файлова система, що розташована на системному диску, призначається кореневою.

Для зв'язку ієрархій файлів у кореневій файловій системі вибирається деякий існуючий каталог. Після виконання монтування обраний каталог стає кореневим каталогом другої файлової системи. Через цей каталог файлова система, що монтують, приєднується як піддерево до загального дерева (рис. 4.5).

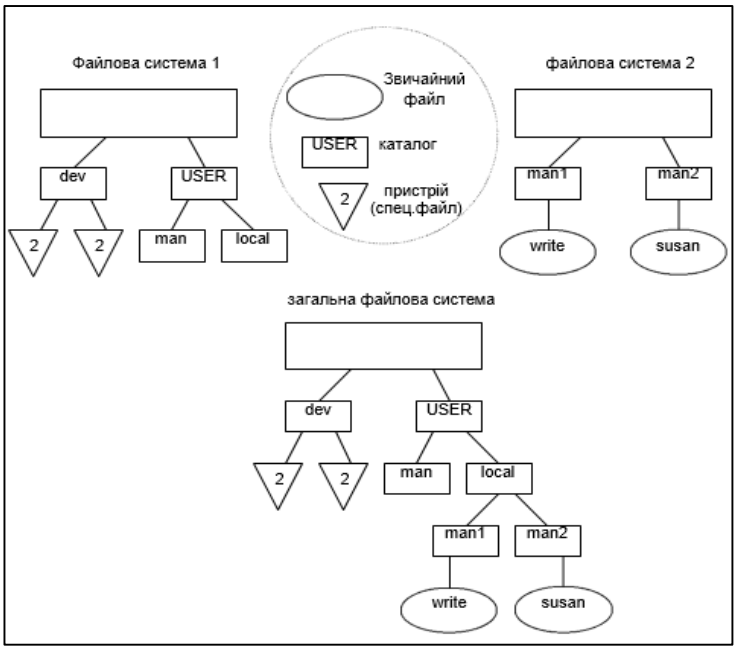

Рис. 4.5. Приклад монтування

Атрибути файлів

Атрибути – інформація, що описує властивості файлу. Можливі атрибути:

- тип файлу (звичайний файл, каталог, спеціальний файл);
- власник файлу;
- творець файлу;
- пароль для доступу до файлу;
- інформація про дозволені операції доступу до файлу;

- часи створення, останнього доступу й останньої зміни;
- поточний розмір файлу;
- максимальний розмір файлу;
- ознака «тільки для читання»;
- ознака «прихований файл»;
- ознака «системний файл»;
- ознака «архівний файл»;
- ознака «двійковий/символьний»;
- ознака «тимчасовий» (видалити по завершенні процесу);
- ознака блокування;
- ознака запису у файл;
- покажчик на ключове поле в записі;
- довжина ключа.

Конкретний перелік атрибутів визначається специфікою файлової системи. Наприклад, в MS-DOS зроблено як на рис.4.6.

Рис.4.6. Перелік атрибутів файлів

Іншим варіантом є розміщення атрибутів у спеціальних таблицях, коли в каталогах містяться тільки посилання на ці таблиці. У такій файловій системі (ОС UNIX) структура каталогу дуже проста (рис. 4.7).

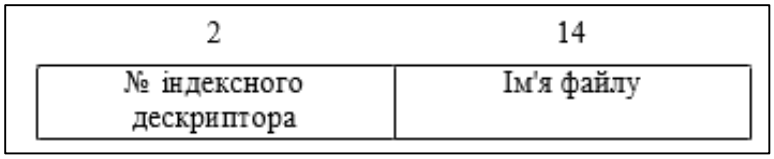

Рис. 4.7. Індексний дескриптор файлу

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 покладена на програму, або, в тій або іншій мірі, цю роботу може брати на себе файлова система.

У першому випадку, файл представляється ФС як неструктурована

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

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

ФС надає додатку доступ до запису, а вся подальша обробка даних, що міститься в цьому записі, виконується програмою. Розвитком цього є системи, що підтримують складні структури даних і взаємозв'язок між ними. Логічний запис –- найменший елемент даних, яким оперує програміст при обміні із зовнішнім пристроєм.

ФС може забезпечувати два способи доступу до логічних записів:

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

4.9. Фізична організація файлової системи

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

Файл може бути розкиданий «шматочками» по всьому диску. Логічно об'єднані файли з одного каталогу зовсім не обов'язково є сусідніми на диску. Принципи розміщення файлів, каталогів і системної інформації на реальному пристрої описуються **фізичною організацією файлової системи.** Основним носієм є жорсткий диск, що складається з **пакету пластин.**

На кожній стороні пластини розміщені тонкі концентричні кільця – доріжки (tracks), на яких зберігаються дані, Кількість доріжок залежить від типу диска. Нумерація доріжок починається з 0 від зовнішнього краю до центру. Коли диск обертається, головка зчитує дані з магнітної доріжки або записує їх

на доріжку дискретними кроками. Кожен крок зсувається на 1 доріжку. У деяких дисках замість однієї головки є по головці на кожну доріжку.

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

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

Сектор – найменша одиниця обміну даними з оперативною пам'яттю, яку можна адресувати. Для того, щоб контролер міг знайти на диску потрібний сектор, необхідно задати йому всі складові адреси сектора: номер циліндра, номер доріжки й номер сектора.

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

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

Доріжки й сектори створюються в результаті виконання процедури фізичного (або низькорівневого) форматування. Форматування передує використанню диска. Для визначення границь блоків на диск записується ідентифікаційна інформація. **Низькорівневий формат** диску не залежить від типу ОС, що цей диск використовує.

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

Перш, ніж форматувати диск під певну файлову систему, він може бути розбитий на розділи. **Розділ** – неперервна частина фізичного диску. ОС представляє його як **логічний пристрій** (логічний диск). Оскільки файлова система, з якою працює одна ОС, не може, у загальному випадку, інтерпретуватися ОС іншого типу, логічні диски не можуть використовуватись

ОС різного типу.

На кожному **логічному диску** може створюватися тільки **одна файлова система.** На **різних логічних дисках** того самого фізичного диску можуть розташовуватися файлові системи **різного типу**. Всі розділи одного диску мають однаковий розмір низькорівневого форматування.

Але при високорівневому форматуванні в різних логічних дисках можуть бути встановлені файлові системи, розміри кластерів яких мають різні розміри.

ОС може підтримувати різні статуси логічних дисків, особливим чином їх розрізняючи. Одні розділи (диски) можна використати для завантаження модулів ОС, в інші можна встановлювати тільки програми й зберігати файли даних. Один з розділів диска позначається як завантажувальний (або активний). Саме з нього зчитується завантажник ОС.

4.10. Фізична організація й адресація файлу

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

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

Неперервне розміщення – найпростіший варіант фізичної організації. При цьому файлу надається послідовність кластерів диску, що утворюють неперервну ділянку дискової пам'яті.

Основне достоїнство – висока швидкість доступу, тому що витрати на пошук і зчитування кластерів файлу мінімальні. Мінімальний й обсяг адресної інформації – досить зберігати тільки номер першого кластера й обсяг файлу. Максимально можливий розмір файлу не обмежується.

Цей варіант має істотний недолік:

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

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 (рис. 4.9). Номер першого кластера запам'ятовується у записі каталогу, де зберігаються характеристики цього файлу. Інша адресна інформація відділена від кластерів файлу.

З кожним кластером диска зв'язується деякий елемент – **індекс**. Таблиця індексів розташовується в окремій області диску і займає один кластер. Коли пам'ять вільна, всі індекси мають нульове значення.

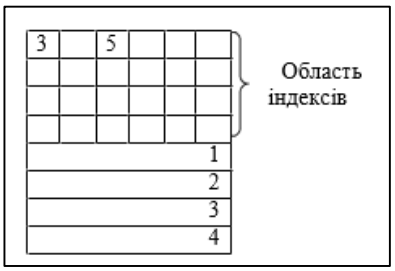

Рис. 4.9. Зв’язаний список адрес

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

Ще один спосіб полягає у простому перерахуванні номерів кластерів, займаних файлом. Цей перелік і служить адресою файлу. **Недолік** очевидний – довжина адреси залежить від розміру файлу й для великого файлу може скласти значну величину. **Достоїнство** – висока швидкість доступу до довільного кластера, тому що тут використовується пряма адресація, а не перегляд ланцюжків.

Питання до самоконтролю:

1. Основні завдання підсистеми введення-виведення мультипрограмної ОС.
2. Причини складності реалізації введення-виведення в мультипрограмних ОС.
3. Принципи привілейованого виконання операцій введення-виведення.
4. Функції супервізора введення-виведення.
5. Режими керування введенням-виведенням і їх особливості.
6. Призначення віртуальних пристроїв та спулінга.
7. Основні системні таблиці введення-виведення та їх роль.
8. Відмінності між синхронним і асинхронним введенням-виведення.
9. Фізична організація файлової системи і принципи розміщення файлів на диску.

МОДУЛЬ 5. СУЧАСНІ ТЕНДЕНЦІЇ В ОС ТА НИЗЬКОРІВНЕВЕ ПРОГРАМУВАННЯ

ТЕМА 5.1. ТЕНДЕНЦІЇ РОЗВИТКУ ОС

План

1. Узгодження швидкостей обміну і кешування даних.
2. Розподіл пристроїв і даних між процесами.
3. Забезпечення зручного логічного інтерфейсу між пристроями й іншою частиною системи.
4. Підтримка широкого спектра драйверів і простота включення нового драйвера в систему.
5. Динамічне завантаження і вивантаження драйверів.
6. Підтримка декількох файлових систем.
7. Підтримка синхронних і асинхронних операцій введення–виведення.
8. Багатошарова модель підсистеми введення-виведення.
9. Менеджер введення-виведення.
10. Багаторівневі драйвери.
11. Спеціальні файли.

Кожен пристрій введення-виведення забезпечується блоком керування - контролером. Контролер взаємодіє із драйвером – системним програмним модулем, що керує 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 проміжках часу між видачею команд незалежно від ОС.

Від підсистем введення-виведення вимагається спланувати в реальному масштабі часу запуск і призупинення драйверів, забезпечивши прийнятний час реакції кожного драйвера на незалежні дії контролера.

Для цього всі драйвери розподіляють на декілька пріоритетних рівнів відповідно до вимог за часом реакції й витрат процесорного часу. Для реалізації цього процесу звичайно використовується диспетчер переривань ОС.

5.1. Узгодження швидкостей обміну і кешування даних

При обміні даними виникає задача узгодження швидкостей. Вона вирішується за рахунок буферизації даних в оперативній пам’яті (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 спул (spool). Типовий приклад – вивід на принтер. Друк документа у кілька десятків мегабайт не є рідкістю.

Інше рішення проблеми – більша буферна пам'ять у контролерах зовнішніх пристроїв. Наприклад, контролери графічних дисплеїв. Їх ОП порівнянна з ОП процесора.

5.2. 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 обчислювача.

ОС може контролювати доступ не тільки до пристрою в цілому, але й до **окремих порцій даних**. Наприклад, при виводі на графічний дисплей – інформація в окремих вікон екрана.

Тому для організації спільного доступу до частин пристрою або частин даних неодмінною умовою є задання режиму спільного використання пристроїв або даних у цілому.

ОС надає пристрої, відслідковуючи процедури захоплення й звільнення 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.

5.3. 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 байтів, з якими можна працювати за допомогою уніфікованих системних викликів (наприклад, read, write), задаючи ім'я файлу - пристрою й зсув від початку послідовності байтів.

Привабливість такої моделі полягає 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.

5.4. Підтримка широкого спектра драйверів і простота включення нового драйвера в систему

Наявність різноманітних драйверів для всіх типів зовнішніх пристроїв є важливою характеристикою ОС. Наприклад, така гарна у багатьох відношеннях ОС як ОС/2 була витіснута ОС Windows завдяки багатству драйверів.

Інтерфейсу драйверів необхідна відкритість, тобто доступність його опису для незалежних розроблювачів ПО. 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 існують 2 типи інтерфейсів:

- драйвер - ядро;
- драйвер - пристрій.

Інтерфейс драйвер - ядро повинен бути обов’язково стандартизований. Інтерфейс драйвер - пристрій має сенс стандартизувати тоді, коли підсистема введення-виведення не дозволяє драйверу безпосередньо взаємодіяти з апаратурою.

Екранування драйвера від апаратних засобів є досить корисною функцією, тому що драйвер стає незалежним від апаратної платформи. Для підтримки процесу розробки драйверів для ОС додатково випускається пакет DDK (Driver Development Kit), що представляє собою необхідний інструментарій: бібліотеки, компілятори й відладчики.

5.5. Динамічне завантаження і вивантаження драйверів

Включення драйвера до складу модулів працюючої ОС являє собою самостійну проблему. Оскільки набір потенційно підтримуваних даною ОС периферійних пристроїв завжди ширший ніж набір 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 зникає.

Це може істотно заощадити 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 усіма компонентами ядра статичні зв'язки замість динамічних. При цьому зміни в процесі роботи ОС неможливі.

5.6. Підтримка декількох файлових систем

Диски – це особливий вид периферійних пристроїв, тому що на них зберігається більша частина користувацьких і системних даних. Ці дані

організуються у файлові системи, властивості яких багато в чому визначають властивості ОС (відмовостійкість, швидкодію, максимальний обсяг збережених даних). Хороша файлова система звичайно переноситься з однієї ОС в іншу.

Так файлова система FAT спочатку була розроблена для MS-DOS, потім перейшла в OS/2 й MS Windows . Важливо, щоб архітектура підсистеми введення-виведення дозволяла досить просто включати у свій склад нові типи файлових систем без необхідності переписування коду. Для цього в ОС передбачається спеціальний шар ПО. Наприклад, шар VFS (Virtual File System) у версіях UNIX.

5.7. Підтримка синхронних і асинхронних операцій введення–виведення

Операції введення-виведення можуть виконуватися в синхронному й асинхронному режимах. Синхронний режим означає, що процес, що запросив операцію, припиняє свою роботу доти, 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 (користувацьким процесам й ядру ОС) можливість виконувати як синхронні, так й асинхронні операції введення-виведення залежно від потреб викликаючої сторони.

Системні виклики введення-виведення найчастіше оформлюються як синхронні процедури у зв'язку з тим, що такі операції тривають довго, й користувацькому процесу або потоку однаково прийдеться чекати результатів, щоб продовжити роботу. Внутрішні виклики з модулів ядра ОС виконуються як асинхронні процедури, тому що ядру потрібен вільний вибір подальшого поводження після запиту операції вводу - виводу.

5.8. Багатошарова модель підсистеми введення-виведення

При великій розмаїтості пристроїв введення-виведення, що значно відрізняються, необхідно дотримуватись балансу між двома суперечливими вимогами:

- необхідністю обліку всіх особливостей кожного пристрою;
- єдиним логічним поданням й уніфікованим інтерфейсом для пристроїв всіх типів.

Нижні шари підсистеми введення-виведення повинні включати індивідуальні драйвери конкретних фізичних пристроїв, а верхні шари повинні узагальнювати процедури керування цими пристроями, надаючи, якщо можливо, загальний інтерфейс.

5.9. Менеджер введення-виведення

Модулі підсистеми введення-виведення, які організують погоджену роботу всіх компонентів підсистеми й взаємодію з користувацькими процесами й іншими підсистемами ОС, утворюють начебто оболонку підсистеми. Ця оболонка називається менеджером введення-виведення.

Верхній шар менеджера підтримує **користувальницький інтерфейс вводу–виводу**, тобто приймає запити на введення-виведення і переадресовує їх драйверам, а також повертає процесам результати операцій вводу–виводу.

Нижній шар взаємодіє з контролерами зовнішніх пристроїв, екрануючи драйвери від особливостей апаратури вводу–виводу, системи переривань тощо. Цей шар приймає від драйверів запити на обмін даними з регістрами контролерів у деякій узагальненій формі з використанням незалежних від шини вводу–виводу 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 й ін. Наявність стандартного віртуального міжмодульного інтерфейсу підвищує стійкість і поліпшує розширюваність підсистеми введення-виведення, хоча й сповільнює її роботу.

5.10. 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;
- 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 керування пристроєм – замість жорсткого набору функцій, які зосереджені у низкорівневому драйвері, адміністратор ОС може вибрати необхідний набір функцій, установивши потрібний **високорівневий**

драйвер.

Якщо різним додаткам необхідно працювати з різними логічними модулями одного й того ж фізичного пристрою, то для цього достатньо установити в системі на одному рівні кілька драйверів, що працюють над одним апаратним драйвером.

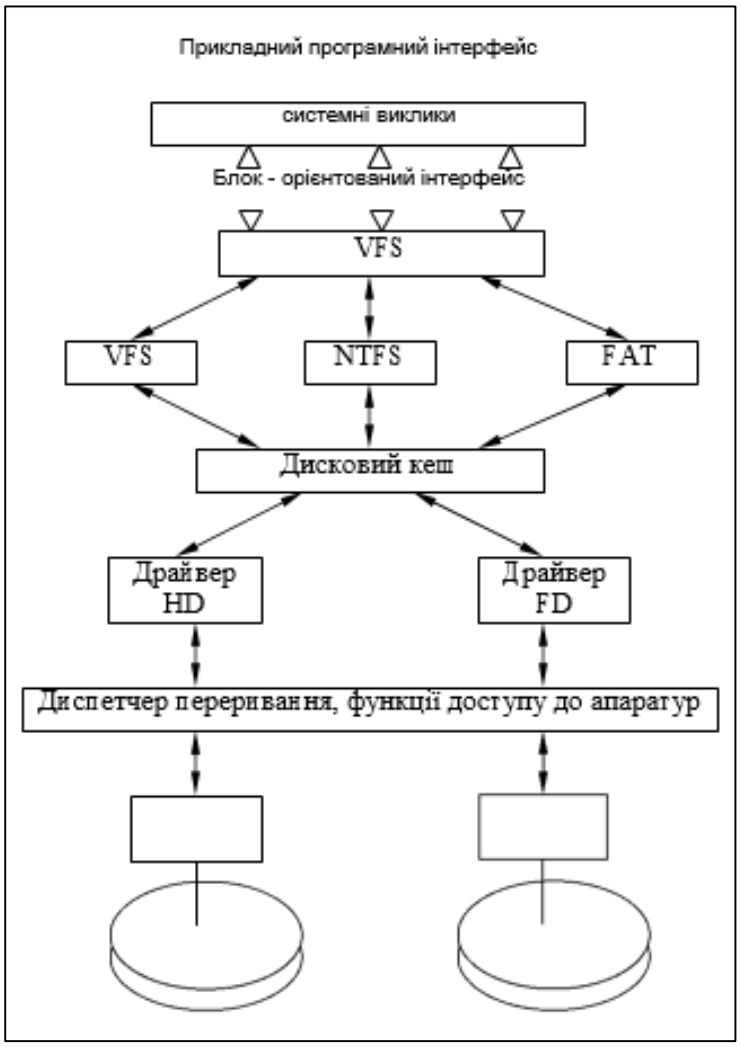

Рис. 5.1. Використання високорівневих драйверів

Кількість рівнів драйверів звичайно не обмежується. На практиці використовується від 2 до 5 рівнів. Високорівневі драйвери оформляються за тими же правилами і дотримуються тих же внутрішніх інтерфейсів, що й апаратні драйвери, за винятком того, що вони не викликаються за перериваннями, тому що взаємодіють з апаратурою.

Як саме загальні принципи побудови багаторівневих драйверів можуть бути реалізовані відповідно до конкретних пристроїв, розглянемо на прикладі керування дисками.

Апаратні драйвери підтримують для верхніх рівнів подання диску як послідовного набору блоків однакового розміру, перетворюючи разом з контролером номер блоку в більш складну адресу з номером 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 установлюється декілька високорівневих драйверів (UFS, NTFS, FAT). Вони працюють із загальними апаратними драйверами, але по-своєму організують файлову систему для користувачів і прикладних процесів.

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 внесла ОС UNIX. У ній всі драйвери розділені на 2 великих класи: блок-орієнтовані й байт-орієнтовані.

Наприклад, драйвери графічних і мережних пристроїв відносяться до байт-орієнтованих, а драйвери, керуючі пристроями прямого доступу, які зберігають інформацію в блоках фіксованого розміру, – до блок-орієнтованих (диск).

Можливість адресації блоків зумовлює для пристроїв прямого доступу можливість кешування даних в оперативній пам'яті. Це впливає на загальну організацію введення-виведення таких пристроїв.

5.11. Спеціальні файли

Спеціальними файлами називають іноді набори даних, які 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 самі 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*).

Питання до самоконтролю:

1. Роль контролера і взаємодія драйвера з пристроєм введення-виведення.
2. Принципи багаторівневих драйверів і їхнє значення для гнучкості ОС.
3. Підтримка синхронних і асинхронних операцій введення-виведення.
4. Підтримка декількох файлових систем і використання VFS для уніфікації подання.

ТЕМА 5.2. МЕРЕЖІ Й МЕРЕЖЕВІ ОПЕРАЦІЙНІ СИСТЕМИ ТА СИСТЕМНЕ ПРОГРАМУВАННЯ

План

1. Операційні системи та системне програмування мережевих пристроїв.
2. Мережеві ОС для комп'ютерних мереж.
3. Novell NetWare.
4. Solaris.
5. Мережеві операційні системи та системне програмування на базі ОС UNIX.
6. Мережеві операційні системи та системне програмування Microsoft.

Операційна система (ОС) – це базовий комплекс програм, що виконує керування апаратною складовою комп'ютера або віртуальної машини; забезпечує керування обчислювальним процесом і організовує взаємодію з користувачем.

Операційна система зазвичай складається з ядра операційної системи та базового набору прикладних програм.

Мережева операційна система – операційна система з вбудованими можливостями для роботи в комп'ютерних мережах. До таких можливостей можна віднести:

- підтримка мережевого обладнання;
- підтримка мережевих протоколів;
- підтримка протоколів маршрутизації;
- підтримка фільтрації зв'язку;
- підтримка доступу до віддалених ресурсів, таких як принтери, диски і т.п. по мережі;
- підтримка мережевих протоколів авторизації;
- наявність в системі мережних служб, які дозволяють віддаленим користувачам використовувати ресурси комп'ютера.

5.2.1. Операційні системи та системне програмування мережевих пристроїв

Мережева операційна система може бути вбудована в маршрутизатор або апаратний міжмережевий екран, який працює з функціями мережевого рівня. Прикладами можуть служити:

- JUNOS – використовується в маршрутизаторах і комутаторах виробника Juniper Networks;
- Cisco IOS – в продукції Cisco;

• TiMOS – в комутаторах від Alcatel-Lucent;

• VRP (Versatile Routing Platform) – в комутаторах Huawei;

• RouterOS – програмне забезпечення, що перетворює комп'ютер або обладнання MikroTik в маршрутизатор;

• ZyNOSruen – використовується пристроями компанії ZyXEL;

• Extensible Operating System – використовується комутаторами фірми Arista Networksruen;

• ExtremeXOSruen, або EXOS – в мережевих пристроях Extreme Networks; Linux для вбудованих систем – такі дистрибутиви, як OpenWrt і DD-WRT, що працюють на недорогих платформах;

• відкриті мережеві операційні системи та системне програмування представлені:

❖ Cumulus Linux від Cumulus_Networks – дистрибутив, який використовує повний TCP/IP стек з Лінукс;

❖ Dell Networking Operating System, або DNOS – нова назва системи для комутаторів фірми Dell Networking. Заснована на NetBSD;

❖ Open Network Operating System (ONOS);

❖ PicOS – заснована на Лінукс ОС фірми Pica;

❖ VyOS – відкритий форк пакету машрутизації Vyatta;

❖ OpenSwitch Linux Network Operating System від Hewlett-Packard.

5.2.2. Мережеві ОС для комп'ютерних мереж

Мережева ОС також включає в себе мережеві служби, що дозволяють віддаленим користувачам використовувати ті чи інші ресурси комп'ютера. Приклади мережевих операційних систем:

• Novell NetWare;

• LANtastic;

• Microsoft Windows (NT, XP, Vista, 7, 8, 8.1, 10);

• UNIX системи, такі як Solaris, FreeBSD;

• GNU/Linux системи.

Головними завданнями є доступ до ресурсів мережі (наприклад, дисковий просторір) і адміністрування мережі. За допомогою мережевих функцій системний адміністратор визначає колективні ресурси, задає паролі, визначає правадоступу для кожного користувача або групи користувачів. Звідси розподіл:

• мережеві ОС для серверів;

• мережеві ОС для користувачів.

Існують спеціальні мережеві ОС, яким надано функції звичайних систем

(наприклад, Windows NT) і звичайні ОС (наприклад, Windows XP), яким надано мережеві функції. Сьогодні практично всі сучасні ОС мають вбудовані мережеві функції.

5.2.3. Novell NetWare

NetWare – мережева операційна система і набір мережевих протоколів, які використовуються в цій системі для взаємодії з комп'ютерами-клієнтами, підключеними до мережі. Операційна система NetWare створена компанією Novell. NetWare є закритою операційною системою, що використовує кооперативну багатозадачність для виконання різних служб на комп'ютерах з архітектурою Intel x86. В основі мережевих протоколів системи лежить стек протоколів Xerox Network Systems (XNS).

В основу NetWare була покладена дуже проста ідея: один або кілька виділених серверів підключаються до мережі і надають для спільного використання свій дисковий простір у вигляді «томів». На комп'ютерах-клієнтах з операційною системою MS-DOS запускається кілька спеціальних резидентних програм, які дозволяють «призначати» літери дисків для цих томів.

Користувачам необхідно зареєструватися в мережі, щоб отримати доступ до томів і мати можливість призначати літери дисків. Доступ до мережевих ресурсів визначається ім'ям реєстрації. Користувачі можуть також підключатися до спільно використовуваних принтерів на виділеному сервері і виконувати друк на мережевих принтерах так само, як і на локальних.

Незважаючи на те, що в ранніх версіях NetWare всі модулі системи вважалися ненадійними, вона була дуже стабільною системою. Зафіксовано випадки, коли сервери NetWare працювали без втручання людини роками.

Система NetWare була створена в результаті роботи SuperSet Software – консалтингової групи, заснованої друзями Дрю Мейджером, Дейлом Найбауером, Кайлом Пауелом і Марком Херстом, в жовтні 1981 року.

У 1983 році Реймонд Ноорда приєднався до роботи групи SuperSet. Спочатку перед групою стояло завдання створення системи CP/M спільного використання дисків для мереж на основі обладнання CP/M, яке в той час продавала компанія Novell.

Всередині групи склалося переконання, що CP/M є приреченою на неуспіх платформою, і в результаті було запропоновано альтернативне рішення для щойно випущених IBM-сумісних ПК. Групою також було написано додаток Snipes – працююча в текстовому режимі гра, яку вони використовували для тестування нової мережі і демонстрації її можливостей. Snipes був першим мережевим додатком для IBM PC і фактично був попередником багатьох популярних багатокористувацьких ігор, таких як Doom та Quake.

Ця мережева операційна система пізніше була названа Novell NetWare. У NetWare використовується протокол NCP (NetWare Core Protocol – протокол ядра NetWare), який є протоколом передачі пакетів, що дозволяє клієнтам передавати запити на сервери NetWare і отримувати від них відповіді. Спочатку NCP був прив'язаний до протоколів IPX/SPX, тобто система NetWare сама по собі могла використовувати для взаємодії в мережі тільки IPX/SPX. Для зберігання інформації аутентифікації використовувалася вбудована система на основі СУБД Btrieve.

Перший програмний продукт з ім'ям NetWare був випущений в 1983 році. Він називався NetWare 68 (або Novell S-Net), працював на процесорі Motorola 68000 і використовував топологію зірка. Цей продукт був замінений в 1985 році на NetWare 86, який був написаний для роботи на процесорах Intel 8086. Після випуску процесора Intel 80286 компанія Novell випустила NetWare 286 (в 1986 році). У 1989 році, після випуску процесора Intel 80386, вийшла NetWare 386. Пізніше Novell переглянула нумерацію версій NetWare: NetWare 286 стала NetWare 2.x, а NetWare 386 стала NetWare 3.x.

З випуском в жовтні 1998 року NetWare 5, компанія Novell визнала вирішальну роль Інтернету і зробила для протоколу NCP підтримку стеку TCP/IP, а не IPX/SPX. Стек протоколів IPX/SPX підтримувався, але роль основного стеку став грати TCP/IP. Більшість утиліт Novell і продуктів інших компаній треба було переписувати для роботи з TCP/IP, а не з IPX/SPX. З NetWare 5 поставлялася перша версія графічної Java-консолі адміністрування – ConsoleOne, яку передбачалося використовувати разом з утилітами NWAdmin.

Основна відмінність NetWare 6 від попередниці пов'язано з черговим поворотом у світогляді менеджерів Novell: розуміючи безнадійність підтримки користувальницького інтересу до головного продукту в традиційному ключі компанії, було прийнято епохальне рішення портувати в ядро NetWare частину POSIX-коду з метою портування на платформу NetWare 6 популярних UNIX програм, таких як WEB Server Apache, SQL сервера MySQL, PHP, SSH та інших додатків. Саме це дозволило зрушити історію операційної системи з мертвої точки.

З випуском в жовтні 2001 року NetWare 6, зміни були продовжені: була додана поліпшена підтримка симетричній багатопроцесорної обробки (SMP – кілька процесорів в одному сервері), iFolder (синхронізація файлів локальної папки з сервером і надання захищеного доступу до них в локальній мережі і через Інтернет), iManager (веб-утиліта адміністрування NetWare та інших продуктів), Native File Access Pack (NFAP – компонент, що надає доступ до ресурсів сервера NetWare клієнтам Windows, Macintosh і UNIX-подібних систем за протоколами відповідних мереж), NetDrive (утиліта, що дозволяє

призначати літери дисків на HTTP- і FTP-ресурси, а також на сервери iFolder), а також веб-сервер за замовчуванням був замінений з Netscape Enterprise Server на Apache.

Також база даних Btrieve (використовувана з попередніх версіях NetWare) була замінена на Pervasive PSQL, що представляє собою розвиток того ж Btrieve.

5.2.4. Solaris

Solaris – операційна система, розроблена компанією Sun Microsystems для платформи SPARC, з 2010 року належить разом з активами Sun корпорації Oracle. Незважаючи на те, що Solaris – операційна система із закритим вихідним кодом, більша його частина відкрита і опублікована в проекті OpenSolaris.

На початку 1990-х років Sun Microsystems замінила засновану на BSD SunOS 4 на UNIX System V Release 4 (SVR4), що розробляється спільно з AT&T, а також змінила ім'я SunOS 5 на Solaris 2. Після виходу версії 2.6 Sun Microsystems відкинула з імені «2», і наступна версія називалася вже Solaris 7.

Фактично, Solaris – це операційна система SunOS з графічною оболонкою і деякими додатковими компонентами. Починаючи з версії Solaris 9 випускалася загальнодоступна (в бінарному вигляді, тобто з закритим вихідним кодом) некомерційна версія Solaris за ліцензією CDDL.

Від комерційної версії вона відрізнялася відсутністю технічної підтримки від Sun, друкованої документації і деякого об'єму додаткового пропрієтарного програмного забезпечення.

У червні 2005 року Sun Microsystems прийняла рішення відкрити значну частину вихідного коду останньої версії системи – Solaris 10 і запустити проект OpenSolaris. Стандартнібінарні збірки Solaris10 були розміщені на веб-сайті Sun Microsystems і доступні всім бажаючим за ліцензією CDDL після реєстрації на веб-сайті компанії.

Розробка наступної версії – Solaris 11 велося вже в співробітництві з спільнотою розробників OpenSolaris. До того ж до вихідного коду операційної системи Solaris, Sun Microsystems відкрила цілий ряд програмного забезпечення власної розробки для неї в рамках проекту OpenSolaris.

З квітня 2010 року новий власник інтелектуальної власності – корпорація Oracle, змінила умови ліцензування системи Solaris 10. За новими правилами, продуктом безкоштовно, в комерційних цілях, стало дозволено користуватися тільки протягом 90 днів, для подальшого комерційного використання Solaris 10 і випущеної в листопаді 2011 року версії 11 необхідно придбання контракту на технічну підтримку операційної системи, або цілком системи від Oracle. Бінарні збірки Solaris 10 і 11 як і раніше доступні на веб-сайті компанії після реєстрації,

але вже за умовами особливої ліцензії розробника (Oracle Technology Network Developer License), що дозволяє їх використання виключно для розробки і тестування додатків під платформу Solaris.

5.2.5. Мережеві операційні системи та системне програмування на базі ОС UNIX

UNIX – операційна система, розробку якої впродовж 1969-1970-х років здійснювала група співробітників підрозділу Bell Labs корпорації AT&T у складі Кена Томпсона, Денніса Рітчі та Дугласа Макілроя. В наш час існує велика кількість різних UNIX-систем, які в свою чергу об'єднуються в родини. У їх розробці в різний час брали участь AT&T, деякі комерційні фірми, а також некомерційні організації.

У 1973 р. було прийняте рішення переписати ядро системи на щойно створеній мові C. UNIX став першою операційною системою, майже повністю написаною на мові програмування високого рівня, що суттєво спростило портування системи на інші архітектури. 15 жовтня на черговому симпозіумі ACM була представлена четверта версія UNIX. Незабаром з'явилася UNIX Version 5, з 1974 року розпочалося розповсюдження безкоштовно серед університетів та академічних закладів.

До 1978 р. система використовувалася більш ніж на 600 машинах, перш за все, в університетах. Версія 7 було останньою єдиною версією UNIX. Саме у версії 7 з'явився близький до сучасного інтерпретатор командного рядка Bourne shell.

На початку 1980-тих компанія AT&T, якій належали Bell Labs, зрозуміла цінність UNIX та почала створення комерційної версії UNIX. Ця версія, яка надійшла у продаж у 1982 році, отримала назву UNIX System III та базувалася на сьомій версії системи. Трохи раніше, у 1977 р. лабораторія Білла Джоя в університеті Берклі створила власну версію UNIX, яка базувалась на UNIX Version 6. Ця версія отримала назву BSD (Berkeley Software Distribution).

Поворотним моментом у історії UNIX стала реалізація у 1980 р. стеку протоколів TCP/IP. До цього міжмашинна взаємодія в UNIX перебувала у зародковому стані – найбільш суттєвим способом зв'язку був UUCP (засіб копіювання файлів з однієї UNIX-системи у іншу, яке спочатку працювало через телефонні мережі за допомогою модемів).

Було запропоновано два інтерфейси програмування мережевих програм: Berkley sockets та інтерфейс транспортного рівня TLI (Transport Layer Interface). Інтерфейс Berkley sockets був розроблений в університеті Берклі та використовував стек протоколів TCP/IP, розроблений у цьому ж університеті.

TLI був створений AT&T згідно з визначенням транспортного рівня

моделі OSI та вперше з'явився у системі System V версії 3. Хоч ця версія містила TLI та потоки, першочергово у ній не було реалізації TCP/IP та інших мережевих протоколів, але подібні реалізації пропонувались сторонніми фірмами.

РеалізаціяTCP/IPофіційнотаостаточнобулавключенаубазовупоставкуSystem V версії 4. Це, також як і інші міркування (більшою частиною ринкові), призвело до остаточного розмежуванняміждвомагілкамиUNIX –BSD таSystemV.Потім багато компаній ліцензували System V у AT&T, і розробили власні комерційні різновиди UNIX, такі, як AIX, HP-UX, IRIX, Solaris.

Вільні UNIX-подібні операційні системи та системне програмування

У 1983 році Річард Столмен оголосив про створення проекту GNU – спроби створити вільну UNIX-подібну операційну системи з нуля, без використання оригінального початкового коду. Більша частина програмного забезпечення, розробленого в рамках цього проекту – такого, як GNU toolchain, Glibc (стандартна бібліотека мови C) та Coreutils – відіграють ключову роль в інших вільних операційних системах.Однак,роботизістворennязаміни дляядра UNIX, необхідного для повного виконання задач GNU, відбувались дуже повільно. На теперішній час GNU Hurd – спроба створити сучасне ядро на основі мікроядерної архітектури Mach – все ще далека від завершення.

У 1991 році, коли Лінус Торвальдс опублікував ядро Linux та залучив помічників, використання інструментів, розроблених в рамках проекту GNU, було очевидним вибором. Об'єднавшись з ядром Linux, програмне забезпечення GNU стало основою для UNIX-подібної операційної системи, відомою як Linux.

Дистрибутиви цієї системи (такі як Red Hat та Debian), які включають ядро, утиліти GNU та додаткове програмне забезпечення стали популярними як серед аматорів, так і серед фахівців. В результаті регулювання юридичної справи, відкритою UNIX Systems Laboratories проти університету Берклі та Berkeley Software Design Inc., було встановлено, що університет може розповсюджувати BSD UNIX, в тому числі і безкоштовно.

Після цього були відновлені експерименти, пов'язані з BSD-версією UNIX. Незабаром розробка BSD UNIX була продовжена у декількох напрямах одночасно, що призвело до появи проектів, відомих як FreeBSD, NetBSD, OpenBSD та DragonFlyBSD.

На теперішній час Linux та представники сімейства BSD швидко відвойовують ринок у комерційних UNIX-систем та одночасно проникають як на персональні комп'ютери користувачів, так і на мобільні вмонтовані системи.

Одним із свідчень даного успіху служить той факт, що коли фірма Apple шукала основу для своєї операційної системи, вона вибрала NEXTSTEP –

операційну систему з вільно розповсюджуваним ядром, розроблену фірмою NeXT та перейменованою у Darwin після придбання фірмою Apple. Ця система відноситься до сімейства BSD та базується на ядрі Mach. Застосування Darwin BSD UNIX у macOS робить його однією з найбільш розповсюджених версій UNIX.

Стандарти

Поки ОС UNIX не була комерційним продуктом, не було потреби в стандартизації засобів цієї ОС. Нечисленні висококваліфіковані користувачі ОС UNIX самі могли розібратися в особливостях і відмінах версії системи, якою вони користуються, та обрати ту підмножину її засобів, яке забезпечувало портованість програми.

Однак, з виходом ОС UNIX на комерційний ринок , переходом до широкого трактування системи та суттєвим збільшенням числа користувачів різних її варіантів, стало необхідним ввести можливість виробництва побудованих на основі ОС UNIX операційних систем, які були б дійсно сумісними.

Для цього необхідна стандартизація (інтерфейсів) засобів операційної системи на різних рівнях. Така робота триває вже близько 10 років, ще не завершена й навряд чи колись буде завершена у вигляді кінцевого набору стандартів де-юре.

Однак, навіть отримані результати дозволяють виробникам забезпечити користувачів різних апаратних платформ операційними системами, достатньо зручними для користування і дають можливість розробляти мобільні прикладні системи, які здатні виконуватись на комп'ютерах, що мають операційні системи та системне програмування з аналогічними властивостями.

Хоча більшість комерційних реалізацій UNIX базувалось на System V, UNIX BSD завжди був популярним в університетах, і громадськість потребувала визначення деякого інтерфейсу, який би був по суті об’єднанням засобів AT&T та BSD.

Ця робота була почата Асоціацією професійних програмістів Відкритих систем UniForum, а потім продовжена в спеціально створених робочих групах POSIX (Portable Operating System Interface). В робочих групах POSIX розробляється багато відкритих систем, але найбільш відомим і авторитетним є ухвалений ISO за клопотанням IEEE стандарт POSIX 1003.1, в якому визначені мінімальні необхідні засоби операційної системи.

5.2.6. Мережеві операційні системи та системне програмування Microsoft

MS-DOS (Microsoft Disk Operating System) – операційна система із родини

DOS, розроблена фірмою Microsoft – комерційна операційна система фірми Microsoft для IBM PC-сумісних персональних комп'ютерів. MS-DOS – найвідоміша ОС ізсімейства DOS, що уминулому встановлювалась на більшість IBM PC-сумісних комп'ютерів. З часом була замінена на ОС сімейства Windows 9x та Windows NT.

MS-DOS була створена в 1981 році і в ході її розвитку було випущено вісім великих версій (1.0, 2.0 і т.д.) та два десятки проміжних (3.1, 3.2 і т.п.), поки у 2000 році в Microsoft не припинила її розробку. За період використання MS-DOS – це був ключовий продукт фірми, що давав їй істотний прибуток і маркетинговий ресурс. В ході розвитку Microsoft перетворилась від розробника мови програмування (Basic) до великої компанії, що виробляє найрізноманітніше програмне забезпечення.

Windows – узагальнююча назва операційних систем для ЕОМ, розроблених корпорацією Microsoft. Перші версії були не повноцінними операційними системами, а лише оболонками до ОС MS-DOS.

Версії Windows можна умовно поділити на кілька груп.

Графічні інтерфейси і розширення для DOS

Ці версії Windows не були повноцінними операційними системами, а лише надавали графічну оболонку.Зодного боку,при роботі зцими версіями Windows користувачі мали змогу використовувати віконний інтерфейс, керування за допомогоюмиші та інші візуальні способи взаємодії зкомп'ютером. В той самий час ці версії Windows самі не мали змоги взаємодіяти з компонентами комп'ютера безпосередньо і використовували для цього можливості ОС MS-DOS.

Сімейство Windows 9x

Сімейство ОС, розроблених спеціально для процесорів з 32-бітною архітектурою. На відміну від попередніх версій, Windows цього сімейства вже є повноцінними операційними системами та не потребують для своєї роботи підтримки з боку MS-DOS.

Водночас спрямування системи на широкий споживацький ринок обумовили підвищені вимоги до зворотної сумісності, тобто можливість виконання широкого спектру програм, написаних для MS-DOS та ранніх версій Windows.

Це призвело до компромісів в архітектурі, що певним чином вплинуло на стабільність Windows цього сімейства. При цьому потреба охопити якнайширший парк встановлених у потенційних користувачів комп'ютерів, накладала досить жорсткі вимоги до швидкості роботи ОС. Пришвидшення роботи частково відбулось за рахунок архітектурних компромісів, що теж вплинуло на стабільність цих систем.

Сімейство Windows NT

Операційні системи та системне програмування цього сімейства працювали на процесорах з архітектурою IA32 та деяких менших RISC-процесорів: Alpha, MIPS (до версії 2000, що вийшла лише у версії для IA32).

Розробка Windows NT велась на тих самих засадах, що і Windows 9X, але NT із самого початку позиціонувалась не на домашнє використання, а на серверний ринок. Це дозволяло не звертати значної уваги на зворотну сумісність та накладало не такі жорсткі обмеження на швидкість роботи.

Таким чином в ОС цього сімейства з самого початку були повноцінно реалізовані механізми безпечної взаємодії між процесами, що позитивно вплинуло на їхню стабільність. Ціною були вищі вимоги до апаратного забезпечення та (особливо в ранніх версіях) обмежена можливість використання старих програм.

Операційні системи та системне програмування сімейства Windows 9x та Windows NT належать до операційних систем з витісняючою багатозадачністю. Поділ процесорного часу між потоками відбувається за принципом «каруселі».

Операційна система виділяє квант часу кожному потоку за чергою з врахуванням пріоритету. Після закінчення виділеного часу система перехоплює у потока керування та виділяє час наступному потоку за чергою. Також потік може відмовитись від виділеного йому кванту часу; в цьому випадку система перехоплює у нього керування (навіть якщо виділений квант часу триває) і передає цей квант іншому потоку.

При передачі керування система зберігає стан всіх регістрів процесора в особливій структурі пам’яті. Ця структура називається контекстом потоку. Збереження контексту потоку дає можливість для наступного поновлення його роботи.

Windows XP та Windows Server 2003

Windows XP (Windows NT 5.1) – операційна система сімейства Windows NT від компанії Microsoft. Вона була випущена 25 жовтня 2001 року і є розвитком Windows 2000 Professional. На відміну від попередньої системи Windows 2000, яка поставлялася як в серверному, такі в клієнтському варіантах, Windows XP є виключно клієнтською системою.

Її серверним варіантом є випущена пізніше система Windows Server 2003. Windows XP і Windows Server 2003 побудовані на основі одного і того ж ядра операційної системи, в результаті їх розвиток і оновлення йшов більш-менш паралельно.

Windows XP аналізує продуктивність системи з певними візуальними ефектами і залежно від цього активує їх чи ні, враховуючи можливе падіння або зростання продуктивності.

Користувачі також можуть змінювати дані параметри, використовуючи діалогові вікна налаштування, при цьому можна або гнучко вибрати активність тих або інших візуальних ефектів, або віддати це на керування системі або ж вибрати максимальну продуктивність або найкращий вид графічного інтерфейсу.

Windows Server 2003 (Windows NT 5.2) – операційна система родини Windows NT від компанії Microsoft, розроблялась для роботи на серверах. Побачила світ 24 квітня 2003 року.

Windows Server 2003 є розвитком Windows 2000 Server і серверним варіантом операційної системи Windows XP. Спочатку Microsoft планувала назвати цей продукт «Windows.NET Server» з ціллю рекламувати свою нову платформу Microsoft.NET. Але у подальшому ця назва була відкинута, щоб не провокувати помилкові уявлення про .NET на ринку програмного забезпечення.

Windows Server 2003 – перша із операційних систем Microsoft, яка надходила до користувача з оболонкою .NET Framework. Це дає системі виступати у ролі сервера додатків для платформи Microsoft .NET без встановлення додаткового програмного забезпечення.

Windows Server 2003 містить такі покращення для Active Directory – служби каталогів, що вперше з'явившись у Windows 2000:

- можливість перейменування домену Windows NT Active Directory після його розгортання.
- спрощені зміни схеми Active Directory – наприклад, відключення атрибутів і класів.
- покращений інтерфейс користувача для керування каталогом (стало можливо, наприклад, переміщувати об'єкти шляхом їх перетягування і водночас міняти властивості декількох об'єктів).
- покращені засоби керуванняя групової політики, включаючи програму Group Policy Management Console.

Windows 7 та Windows Server 2008 R2

Windows 7 – назва версії операційної системи Windows, яка вийшла 22 жовтня 2009 року, менше, ніж через три роки після випуску попередньої операційної системи, комерційно невдалої Windows Vista. Розробка Windows 7 під різними кодовими назвами велась з 2000 року.

Найвідчутнішим нововведенням розробник називає широке застосування інтерфейсу вводу, що керується дотиком (touchscreen), у варіанті реагування на кілька дотиків одночасно (multi-touch).

Істотно перероблений інтерфейс системи, панель завдань і меню «Пуск». Дістати доступ до основних функцій можна, не розгортаючи додаток, – достатньо натиснути на відповідну іконку в панелі завдань правою кнопкою

миші.

Наприклад, у випадку з мультимедійним плеєром користувач може відкрити список відтворення, а у випадку з браузером – переглянути зменшені зображення відкритих веб-сторінок і перемкнутися між ними. У меню «Пуск» з'явився доступ до тек, що найчастіше відкривалися.

У Windows 7 також вбудовано Windows Defender та брандмауер.

Windows Server 2008 R2 – нова серверна операційна система компанії Microsoft, що є вдосконаленою версією Windows Server 2008 (серверний версія Windows Vista). Надійшла у продаж 22 жовтня 2009. Як і Windows 7, Windows Server 2008 R2 використовує ядро Windows NT 6.1. Нові можливості включають поліпшену віртуалізацію, нову версію Active Directory, Internet Information Services 7.5 і підтримку до 256 процесорів. Система доступна тільки в 64-розрядному варіанті.

Windows 10 та Windows Server 2016

Windows 10 – операційна система від компанії Microsoft для персональних комп'ютерів, ноутбуків, планшетів і смартфонів. В компанії цю версію операційної системи називали останньою, так як надалі вона надається за моделлю «програмне забезпечення як послуга».

Реліз Windows 10 відбувся влітку 2015 року, а саме 29 липня в 190 країнах і 111-ма мовами. Протягом першого року після виходу системи користувачі мали змогу безкоштовно оновитися до Windows 10 на будь-якому пристрої під керуванням офіційних версій Windows 7, Windows 8.1 і Windows Phone 8.1, що відповідають певним вимогам.

Windows 10 працює на всіх сучасних пристроях (смартфон, планшет, комп'ютер, ноутбук, телевізор та приставка). Ще одна особлива функція в Windows 10 – якщо пристрій під'єднаний до Інтернету і ви під'єднали до комп'ютера невідомий пристрій для комп'ютера або ОС, то не потрібно буде шукати самому драйвера під цей пристрій, Windows його знайде, завантажить та повторно підключить пристрій до комп'ютера.

Windows 10 має вдосконалену функцію мультивіконності Snap Assist, яка допомагає розподіляти простір екрана між вікнами.

Вона дозволяє розташувати на робочому столі до чотирьох вікон одночасно. При цьому Windows 10 підказує, які ще додатки запущені в системі і як їх можна розмістити. В цій ОС додано функцію створення кількох робочих столів (подібну функцію можна побачити в macOS та Ubuntu). Windows Server 2016 – серверна операційна система компанії Microsoft. Вона належить до сімейства операційних систем Windows NT і розробляється паралельно із Windows 10.

Windows 11 та Windows Server 2022

Windows 11 – найновіша версія сімейства операційних систем Windows NT, що розроблюється Microsoft. Була представлена 24 червня 2021 року та вийшла 5 жовтня 2021 року.Windows 11 доступна як безкоштовне оновлення для сумісних пристроїв з Windows 10 та Windows 8.1, за допомогою служби Windows Update.

Операційна система має підтримку додатків Android, однак вимоги для роботи підтримки додатків Android вищі, ніж для самої операційної системи.

Windows Server 2022 – серверна операційна система від Microsoft, що є найновішою операційною системою сімейства Windows Server. Була випущена 18 серпня 2021 року, майже 3 роки після попередньої версії Windows Server, Windows Server 2019.

В ній з'явилася розширена багаторівнева система безпеки. Стало більше можливостей для роботи з Microsoft Azure та взагалі хмарними сервісами. Містить вдосконалення в роботі з контейнерами .NET, ASP.NET та IIS.

Питання до самоконтролю:

1. Основне призначення операційної системи та її складові.
2. Відмінні риси мережевої операційної системи та її функції.
3. Приклади мережевих ОС, вбудованих у маршрутизатори та комутатори.
4. Роль мережевих служб у мережевих операційних системах.
5. Основні характеристики та історія розвитку Novell NetWare.
6. Особливості операційної системи Solaris та її відкриті версії.
7. Внесок UNIX у розвиток мережевих ОС та системне програмування.
8. Вільні UNIX-подібні операційні системи, GNU-проект і Linux.
9. Історія розвитку операційних систем Microsoft, включаючи MS-DOS і Windows.
10. Новітні версії Windows і Windows Server та їх основні особливості.

СПИСОК РЕКОМЕНДОВАНИХ ТА ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Архітектура системного програмного забезпечення : підручник / Л. О. Левченко, Н. Г. Кучук, Ю. А. Тарнавський, В. П. Колумбет. Київ : КПІ ім. Ігоря Сікорського, 2022. 497 с. URL: https://ela.kpi.ua/server/api/core/bitstreams/d932fa9d-e1b7-4275-9dea-77b3af0c2c66/content

2. Задерейко О. В., Гура В. І., Толокнов А. А. Операційні системи та системне програмування : навчально-методичний посібник. Одеса : Фенікс, 2023. 298 с. URL: http://surl.li/sgiho

3. Задерейко О. В., Зіноватна С. Л., Толокнов А. А. Операційні системи та системне програмування : навчальний посібник. Одеса : Фенікс, 2022. 140 с. URL: http://surl.li/qlehe

4. Левченко Л. О., Тарнавський Ю. А. Операційні системи та системне програмування : навчальний посібник. Київ : КПІ ім. Ігоря Сікорського, 2023. 256 с. URL: https://ela.kpi.ua/server/api/core/bitstreams/e8b6f0e6-9381-4aeb-b699-dfccc92edddb/content

5. Мосіюк О. О., Федорчук А. Л. Операційні системи та системне програмування та системне програмування : навчально-методичний посібник. Житомир : ЖДУ ім. І. Франка, 2022. 76 с. URL: http://eprints.zu.edu.ua/33751/1/OS_ost_Feb_04.pdf

6. Операційні системи та системне програмування : навчальний посібник / І. М. Федотова-Півень та ін. ; за ред. В. М. Рудницького. Харків : ТОВ «ДІСА ПЛЮС», 2019. 216 с. URL: http://surl.li/aucbs

7. Операційні системи та системне програмування : навчальний посібник / уклад. В. Г. Зайцев, І. П. Дробязко. Київ : КПІ ім. Ігоря Сікорського, 2019. 240 с. URL: https://ela.kpi.ua/server/api/core/bitstreams/d9d83ff6-4004-4a8c-8772-62b624a2196b/content

8. Погребняк Б. І., Булаєнко М. В. Операційні системи та системне програмування : навчальний посібник. Харків : ХНУМГ ім. О. М. Бекетова, 2018. 104 с. URL: http://surl.li/auwlf

9. Сумець О. М. Проєктування операційних систем : підручник. Київ : Університет «КРОК», 2021. 322 с. https://library.krok.edu.ua/media/library/category/pidruchniki/sumets_0002.pdf

Навчальне видання

ОПЕРАЦІЙНІ СИСТЕМИ ТА СИСТЕМНЕ ПРОГРАМУВАННЯ

конспект лекцій

Укладачі:

Тищенко Світлана Іванівна
Пархоменко Олександр Юрійович
Ємельянов Святослав Ігорович
Кучмійова Тетяна Сергіївна
Жебко Олександр Олегович
Волосюк Юрій Вікторович
Коломієць Андрій Миколайович

Формат 60х84 1/16. Ум. друк. арк. 9,00.
Наклад 50 прим. Зам. № _____

Надруковано у видавничому відділі
Миколаївського національного аграрного університету
54020, м. Миколаїв, вул. Георгія Гонгадзе, 9

Свідоцтво суб'єкта видавничої справи ДК № 4490 від 20.02.2013 р.