Тема 1. Основні концепції, еволюція, різновиди операційних систем

Вступ в операційні системи.

Історія та еволюція операційних систем.

Властивості операційних систем та їх класифікація.

Вступ в операційні системи.

Причиною появи операційних систем була необхідність створення зручних у використанні комп'ютерних систем , що являють собою сукупність апаратного і програмного забезпечення комп'ютера. Вони призначені для розв'язання практичних задач користувачів. Оскільки робити це за допомогою лише апаратного забезпечення дуже складно і не зручно, тому що машинні команди дуже примітивні, а їхня рутинність виходить за межі можливостей людини, то були створені прикладні програми. Для таких програм знадобилися загальні операції керування апаратним забезпеченням, розподілу апаратних ресурсів тощо. Ці операції згрупували в рамках окремого рівня програмного забезпечення, який і стали називати операційною системою.

Далі можливості операційних систем вийшли далеко за межі базового набору операцій, необхідних прикладним програмам, але проміжне становище таких систем між прикладними програмами й апаратним забезпеченням залишилося незмінним. Можна дати таке визначення операційної системи.

Операційна система – це комплекс керуючих програм які забезпечують технічне функціонування обчислювальної системи, діагностику несправностей, планування використання ресурсів системи та вирішення задач по завданню користувачів. Операційні системи здійснюють також ввід/вивід інформації та обмін даними між різними компонентами системи. Як правило, операційну систему розглядають як продовження апаратної частини комп’ютера. Тому ще однією задачею ОС є керування виконанням

завдань користувачів з метою максимального підвищення продуктивності обчислювальної системи.

Історія та еволюція операційних систем.

Операційні системи почали розроблятися для ЕОМ 2 покоління, тому що збільшувався об’єм інформації який потрібно було обробляти за короткий проміжок часу. Крім того ЕОМ ширше почали використовувати для наукових і економічних розрахунків. З іншої сторони розвиток електроніки давав змогу значно збільшувати швидкодію апаратного забезпечення. Тому актуальною була задача створювати програмні засоби які б дозволяли автоматизувати роботу обслуговуючого персоналу пов’язану з завантаженням ЕОМ завданнями користувачів. Таким чином в ЕОМ 2 покоління були розроблені окремі елементи системного програмного забезпечення які дозволяли вирішити такі задачі.

Перші повномасштабні операційні системи з’явилися в ЕОМ 3 покоління. Вони представляли собою повномасштабні пакети програм які постійно знаходилися в ОЗП машини. Окрема частина ОС яка постійно знаходиться в ОЗП називається ядром системи. Інша частина, як правило, зберігається на зовнішніх носіях зберігання інформації і тому називається транзитною.

Операційні системи в ЕОМ 3 покоління працювали в мультипрограмному режимі. При якому в ЕОМ поступала велика кількість задач користувача, що для своєї роботи вимагала ресурсів ЕОМ. До ресурсів відноситься все, що обчислювальна система може дати користувачу:

- час процесора;
- місце в ОЗП;
- зовнішня пам’ять пристроїв вводу/виводу;
- консоль;
- окремі системні програми, тощо.

В ЕОМ 4 покоління, зокрема в ПК, основним завданням ОС та СПЗ в цілому є забезпечення виконання задач одного користувача.

Операційні системи для ЕОМ 4 покоління (ПК) розділяються на 3 групи:

1) найпростіші ОС призначені для обслуговування тільки одного користувача. В кожний момент часу в пам'яті тільки одна програма. В склад ядра ОС входять декілька найпростіших програм;

2) інструментальні однокористувацькі ОС. Як правило, розрахований на роботу з ОЗП великої ємності. В склад ядра ОС входять програми керування периферійними пристроями та файловою системою. Для таких ОС розроблено великий комплекс системного програмного забезпечення;

3) багатокористувацькі і багатозадачні ОС. Для високопродуктивних комп'ютерів. Можуть виконувати багато задач одночасно та взаємодіяти з багатьма користувачами. На сьогодні мають розроблений великий фонд СПЗ.

Для виконання програми необхідно як мінімум два ресурси: оперативна пам'ять, для зберігання коду і даних команд та мікропроцесор, для її виконання.

Завдання ОС – розподіл ресурсів обчислювальної системи, керування апаратним забезпеченням та організація виконання програм.

Операційна система може працювати в одному з 3 режимів:

1) однопрограмний – всі ресурси обчислювальної системи надаються лише одній програмі, яка здійснює обробку даних;

2) багатопрограмний – в цьому режимі кілька незалежних одна від одної програм здійснюють обробку даних одночасно (паралельно). При цьому ці програми розділяють деякі ресурси обчислювальної між собою. Основою для такого режиму є суміщення в часі виконання програми мікропроцесором та звернення програми до периферійних пристроїв. Перевагою мультипрограмного режиму є більш ефективне використання

ресурсів обчислювальної системи та збільшення її пропускної здатності (середня кількість програм виконаних за одиницю часу). В мультипрограмному режимі в оперативній пам'яті обчислювальної системи одночасно знаходяться декілька програм, однак в кожний момент часу процесор виконує тільки одну з них;

3) багатозадачний – виконання декількох задач є скоординованою з метою досягнення однієї спільної мети.

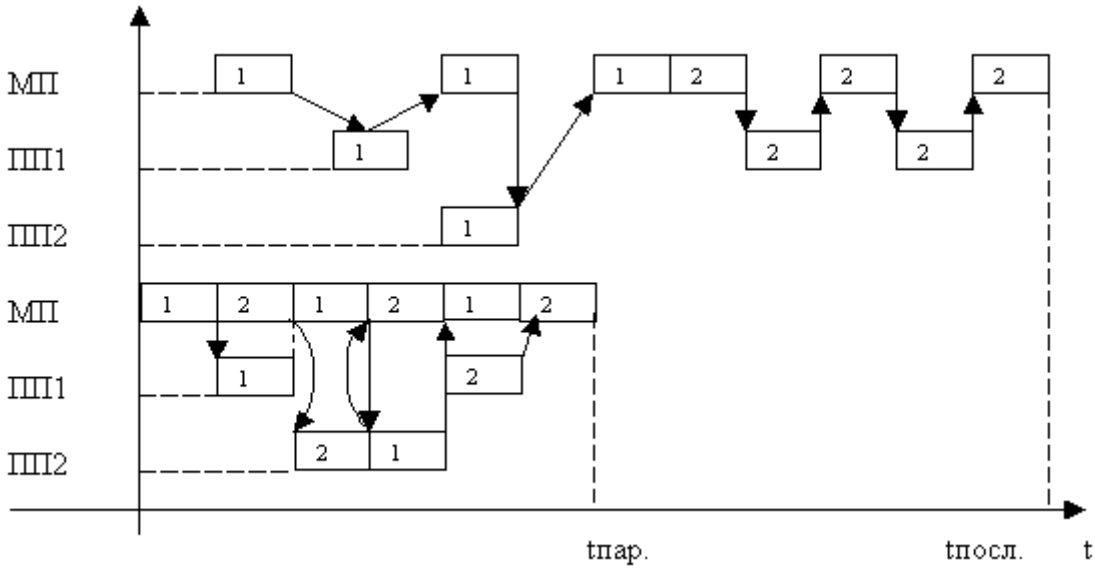

Рис. 1. Однопрограмний та багатопрограмний режим роботи ОС.

Для того щоб ініціалізувати обмін програми з периферійними пристроями вона звертається до ОС. Реагуючи на таке звернення ОС передає периферійному пристрою команду почати обмін даними з відповідними програмами, а потім перемикає мікропроцесор на виконання іншої програми. Після завершення периферійними пристроями обміну даними програма, яка ініціювала цей обмін знову готова до її виконання на мікропроцесорі.

Виконання обчислювальною машиною програми, яка здійснює обробку даних називається обчислювальним процесом. Часто замість терміну процес використовують термін задача. У випадку багатопрограмного режиму роботи кожна із задач є логічно незалежною від іншої задачі. Задачі в процесі виконання не взаємодіють між собою.

Багатозадачний режим здійснюється так. Для цього в складі ОС повинні бути засоби, які дозволяють задачам взаємодіяти між собою. В цьому режимі програми можуть виконуватись як паралельно, так і послідовно, в залежності від конкретних умов, які виникають в результаті взаємодії процесів. Таким чином один з основних призначень будь – якої ОС є забезпечення роботи обчислювальної системи, в одному з описаних вище режимах, динамічний розподіл ресурсів обчислювальної системи та керування обчислювальними системними процесами в залежності від вимог користувача.

Ресурс – всякий об’єкт, який може бути розподіленим між обчислювальними процесами.

До апаратних ресурсів обчислювальних систем відносяться: мікропроцесор; периферійні пристрої. До програмних – системне програмування; засоби керування пристроями та файлами; бібліотеки програм; засоби керування задачами.

Операційна система розподіляє ресурси у відповідності з запитами користувачів, можливостями апаратного забезпечення та взаємодії обчислювальних процесів. Ресурс працює в режимі розподілу, якщо кожен з обчислювальних процесів і займає його протягом певного інтервалу часу. ОС забезпечує взаємодію користувача і обчислювальної системи. Також операційна система звільняє користувача від обов’язку розподілу ресурсів та керування ними. Крім цього ОС здійснює аналіз запитів користувачів і забезпечує їх виконання. Запит відображає необхідні ресурси ЕОМ, необхідні дії та представляється послідовністю команд по внутрішній мові ОС. Така послідовність команд називається завданням. Завдання в свою чергу розподіляється на задачі.

Властивості операційних систем та їх класифікація.

В залежності від режиму виконання запиту користувача, операційні системи поділяються на:

1) ОС пакетної обробки – це система, яка обробляє пакет завдань, які підготовлені одним або декількома користувачами. Пакет завдань поступає на обробку з периферійних пристроїв, взаємодія користувача зі своїм завданням під час обробки неможлива. Обчислювальні системи під керуванням ОС пакетної обробки можуть працювати в одно та багатопрограмних режимах.

2) ОС розподілу часу – забезпечують одночасне обслуговування багатьох користувачів, дозволяючи при цьому кожному з них взаємодіяти зі своїм завданням в режимі діалогу. Ефект одночасного обслуговування досягається розподілом процесорного часу та інших ресурсів обчислювальної системи між кількома обчислювальними процесами, які відповідають окремим завданням користувачів. ОС надає ресурс комп'ютера кожному обчислювальному процесу на певний невеликий проміжок часу (квант часу). Якщо обчислювальний процес не закінчив своє виконання до кінця свого певного кванту часу, його виконання переривається і він заноситься в чергу процесів віддавши ресурси ЕОМ іншому процесу. ОС розподілу часу працюють як правило, як правило, в багатопрограмному режимі.

3) ОС реального часу – повинні гарантувати виконання запитів користувачів за певний заданий відрізок часу. Запити в таких системах надходять як від користувачів, так і від зовнішніх пристроїв і при цьому швидкість виконання обчислювальних процесів повинна бути узгоджена зі швидкістю процесів, які виконуються поза обчислювальною системамою, що керує ними. Час відповіді на запит не повинен перевищувати певних заданих значень реального часу (сек., хв., год.). Необхідний час відповіді системи визначається властивостями об'єктів які вона обслуговує. Обчислювальні системи під керівництвом таких ОС, як правило, працюють у багатозадачному режимі.

4) Діалогові ОС – отримали широке розповсюдження для ПК. Їхньою метою є забезпечення зручної форми діалогу з користувачем. Для виконання частовиконуваних задач вони мають можливість пакетної обробки

під керуванням діалогової ОС, коли може працювати в одному з 3-х вище перелічених режимів. Операційна система в загальному випадку складається з ядра та набору системних програм і даних. Ядро здійснює організацію взаємодій користувачів з обчислювальними системами, керування розподілів ресурсів та забезпечення необхідного режиму функціонування ЕОМ, завантаження і контроль виконання програм, обмін даними з периферійними пристроями та керування файлами. Системні програми обслуговують зовнішні пристрої, здійснюють безпосередні файлові операції, підготовку та ввід вхідної інформації, вивід результатів обчислень, а також зберігання і виконання програм.

Тема 2. Архітектура та ресурси операційних систем

Принципи організації операційних систем. Функції сучасних операційних систем. Архітектура операційних систем та її зв'язок з апаратними засобами. Управління вводом/виводом. Організація пам'яті. Переривання.

Принципи організації операційних систем.

Розглянемо 10 принципів побудови системних програм:

1) Частотний принцип.

Базується на розділенні програм і даних по частоті використання. Для операцій які часто використовуються створюються умови їх швидкого виконання. Найбільш часто виконувані операції створюються найкоротшими. До найбільш часто виконуваних даних забезпечується найшвидший доступ.

2) Принцип модульності.

Модуль – це функціональний елемент певної системи, який має певне оформлення, закінчення та наповнення в рамках даної системи, а також засоби взаємозв'язку з аналогічними модулями, з модулями більш високого рівня, або з модулями іншої системи. По своєму визначенню модуль вказує на легкий спосіб його заміни іншим при наявності певних програмних

інтерфейсів. Системні програми розділяються на модулі по функціональному призначенню. Як правило, модулі впорядковані ієрархічно, що дозволяє значно спростити експлуатацію та розробку програм, зменшити кількість проектних та інших помилок.

3) Принцип функціональної вибраності.

Є логічним продовження попередніх двох принципів. Частина важливих модулів повинна бути постійно в активному стані з метою ефективної реалізації обчислювального процесу. Така частина модулів системної програми називається ядром. При формуванні ядра повинні бути забезпечені дві протилежності: з однієї сторони в склад ядра мають входити програми які використовуються найчастіше, з іншої сторони – розмір ядра повинен бути мінімальний.

4) Принцип генерованості.

Дозволяє настроювати системні програми, виходячи з конкретної апаратної конфігурації обчислювальної системи та певного кола вирішуваних задач. Цей принцип реалізується окремою програмою чи утилітою, в результаті чого отримається повна версія системної програми.

5) Принцип функціональної надлишковості.

Системна програма повинна мати можливість виконувати одну і ту ж саму дію різними способами.

6) Принцип по замовчуванню.

Полягає у зберіганні в складі системної програми певних базових описів модулів, конфігурацій, даних які визначають прогнозовані параметри апаратного та програмного забезпечення.

7) Принцип переміщуваності.

Полягає в проектуванні модулів системної програми таким чином, що їх виконання незалежить від їх розміщення в ОЗП. Налаштування модуля на конкретне розміщення в ОЗП відбувається перед виконанням програми і полягає у визначенні фактичних адрес команд програми, в залежності від типу використовуваної обчислювальної системи та моделі пам’яті.

8) Принцип захисту.

Визначається необхідністю розробки засобів, які захищають програми і дані користувача від ушкоджень та будь – якого впливу інших користувачів або інших програм. Програми повинні бути захищені як на етапі виконання, так і під час зберігання. Цей принцип в тій чи іншій мірі реалізований в кожній мультипрограмній ОС.

9) Принцип незалежності програм від зовнішніх пристроїв.

Дозволяє здійснювати управління та обмін даними із зовнішніми пристроями, незалежно від їх конкретних фізичних характеристик. Цей принцип в багатьох сучасних системах реалізований за допомогою механізму драйверів.

10) Принцип відкритості і нарощуваності.

Відкрита системна програма доступна для аналізу спеціаліста. Нарощувана програма дозволяє виконувати не лише принцип генерованості, але й дозволяє вводити в склад системи нові модулі і нарощувати існуючі.

Призначення операційної системи.

Операційні системи забезпечують, по-перше, зручність використання комп'ютерної системи, по-друге, ефективність і надійність її роботи.

Перша функція ОС це реалізація розширеної машини замість реальної фізичної машини, друга – це розподіл апаратних ресурсів між усіма виконуваними програмами.

Операційна система як розширена машина.

За допомогою операційної системи у прикладного програміста (а через його програми і в користувача) має створюватися враження, що він працює з розширеною машиною

Апаратне забезпечення комп'ютера недостатньо пристосоване до безпосереднього використання у програмах. Наприклад, для роботи із пристроями введення-виведення набір команд відповідних контролерів дуже обмежений, а для багатьох пристроїв - примітивний (є навіть вислів:

«апаратне забезпечення потворне»). Операційна система приховує такий інтерфейс апаратного забезпечення, замість нього програмістові пропонують інтерфейс прикладного програмування (рис. 1.1), що використовує поняття вищого рівня (їх називають абстракціями).

Наприклад, при роботі з диском типовою абстракцією є файл. Працювати з файлами простіше, ніж безпосередньо з контролером диска (не потрібно враховувати переміщення головок дисковода, запускати й зупиняти мотор тощо), внаслідок цього програміст може зосередитися на суті свого прикладного завдання. За взаємодію з контролером диска відповідає операційна система.

Виділення абстракцій дає змогу досягти того, що код ОС і прикладних програм не потребуватиме зміни при переході на нове апаратне забезпечення. Наприклад, якщо встановити на комп'ютері дисковий пристрій нового типу (за умови, що він підтримується ОС), всі його особливості будуть враховані на рівні ОС, а прикладні програми продовжуватимуть використовувати файли, як і раніше. Така характеристика системи називається апаратною незалежністю. Можна сказати, що ОС надають апаратно-незалежне середовище для виконання прикладних програм.

Операційна система як розподілювач ресурсів

Одна із основних функцій ОС - це ефективний розподіл ресурсів. Під ресурсами розуміють процесорний час, дисковий простір, пам'ять, засоби доступу до зовнішніх пристроїв. Операційна система виступає в ролі менеджера цих ресурсів і надає їх прикладним програмам на вимогу.

Розрізняють два основні види розподілу ресурсів – просторовий і часовий. У разі просторового розподілу ресурс доступний декільком споживачам одночасно, при цьому кожен із них може користуватися частиною ресурсу (так розподіляється пам'ять). У разі часового розподілу система ставить споживачів у чергу і згідно з нею надає їм змогу

користуватися всім ресурсом обмежений час (так розподіляється процесор в однопроцесорних системах).

При розподілі ресурсів ОС розв'язує можливі конфлікти, запобігає несанкціонованому доступу програм до тих ресурсів, на які вони не мають прав, забезпечує ефективну роботу комп'ютерної системи.

Історія розвитку операційних систем.

Перші операційні системи з'явилися в 50-ті роки і були системами пакетної обробки. Такі системи забезпечували послідовне виконання програм у пакетному режимі (без можливості взаємодії з користувачем). У певний момент часу в пам'яті могла перебувати тільки одна програма (системи були однозадачними), усі програми виконувалися на процесорі від початку до кінця. За такої ситуації ОС розглядали просто як набір стандартних служб, необхідних прикладним програмам і користувачам.

Наступним етапом стала підтримка багатозадачності. У багатозадачних системах у пам'ять комп'ютера стали завантажувати кілька програм, які виконувалися на процесорі навперемінно. При цьому розвивалися два напрями: багатозадачна пакетна обробка і розподіл часу. У багатозадачній пакетній обробці завантажені програми, як і раніше, виконувалися в пакетному режимі. У режимі розподілу часу із системою могли працювати одночасно кілька користувачів, кожному з яких надавався діалоговий термінал (пристрій, що складається із клавіатури і дисплея).

Підтримка багатозадачності потребувала реалізації в ОС засобів координації задач. Можна виділити три складові частини такої координації.

1.Захист критичних даних задачі від випадкового або навмисного доступу інших задач.

2.Забезпечення обміну даними між задачами.

3.Надання задачам справедливої частки ресурсів (пам'яті, процесора, дискового простору тощо).

Ще одним етапом стала поява ОС персональних комп'ютерів. Спочатку ці системи, як і ОС першого етапу, були однозадачними й надавали базовий набір стандартних служб (на цьому етапі важливим було впровадження графічного інтерфейсу користувача). Подальший розвиток апаратного забезпечення дав змогу використати в таких системах засоби, розроблені для більших систем, насамперед багатозадачність і, як наслідок, координацію задач.

Класифікація сучасних операційних систем.

Залежно від області застосування ОС діляться на ОС великих ЕОМ (мейнфреймів), серверні ОС, персональні ОС, ОС реального часу, вбудовані ОС.

Основною характеристикою апаратного забезпечення, для ОС великих ЕОМ (мейнфреймів) є продуктивність введення-виведення: великі ЕОМ оснащують значною кількістю периферійних пристроїв (дисків, терміналів, принтерів тощо). Такі комп'ютерні системи використовують для надійної обробки значних обсягів даних, при цьому ОС має ефективно підтримувати цю обробку (в пакетному режимі або в режимі розподілу часу). Прикладом ОС такого класу може бути OS/390 фірми IBM.

До наступної категорії можна віднести серверні ОС. Головна характеристика таких ОС - здатність обслуговувати велику кількість запитів користувачів до спільно використовуваних ресурсів. Важливу роль для них відіграє мережна підтримка. Є спеціалізовані серверні ОС, з яких виключені елементи, не пов'язані з виконанням їхніх основних функцій (наприклад, підтримка застосувань користувача). Нині для реалізації серверів частіше застосовують універсальні ОС (UNIX або системи лінії Windows XP).

Наймасовіша категорія - персональні ОС. Деякі ОС цієї категорії розробляли з розрахунком на непрофесійного користувача (лінія Windows 95/98/Me фірми Microsoft, які називають Consumer Windows), інші є спрощеними версіями універсальних ОС. Особлива увага в персональних ОС

приділяється підтримці графічного інтерфейсу користувача і мультимедіа-технологій.

Виділяють також ОС реального часу. У такій системі кожна операція має бути гарантовано виконана в заданому часовому діапазоні. ОС реального часу можуть керувати польотом космічного корабля, технологічним процесом або демонстрацією відеороликів. Існують спеціалізовані ОС реального часу, такі як QNX і VxWorks.

Ще однією категорією є вбудовані ОС. До них належать керуючі програми для різноманітних мікропроцесорних систем, які використовують у військовій техніці, системах побутової електроніки, смарт-картах та інших пристроях. До таких систем ставлять особливі вимоги: розміщення в малому обсязі пам'яті, підтримка спеціалізованих засобів введення-виведення, можливість прошивання в постійному запам'ятовувальному пристрої. Часто вбудовані ОС розробляються під конкретний пристрій; до універсальних систем належать Embedded Linux і Windows CE.

Тема. Функціональні компоненти операційних систем

1.Функціональні компоненти ОС
2.Керування процесами й потоками
3.Керування пам`ятю
4.Керування введенням - виведенням
5.Керування файлами та файлові системи
6.Мержні та розподілені системи
7.Безпека даних
8.Інтерфейс користувача

1. Операційну систему можна розглядати як сукупність функціональних компонентів кожен з яких відповідає за реалізацію певної функції системи. Основними функціями ОС є керування процесами й

потоками, керування пам`яттю, керування введенням – виведенням, керування файлами та файлові системи, мережна підтримка, безпека даних, взаємодія з користувачем.

Спосіб побудови системи зі складових частин та їхній взаємозв'язок визначає архітектура операційної системи.

2. Однією з найважливіших функцій ОС є виконання прикладних програм. Код і дані прикладних програм зберігаються в комп'ютерній системі на диску в спеціальних виконуваних файлах. Після того як користувач або ОС вирішать запустити на виконання такий файл, у системі буде створено базову одиницю обчислювальної роботи, що називається процесом (process). Процес - це програма під час її виконання.

Операційна система розподіляє ресурси між процесами. До таких ресурсів належать процесорний час, пам'ять, пристрої введення-виведення, дисковий простір у вигляді файлів. При розподілі пам'яті з кожним процесом пов'язується його адресний простір - набір адрес пам'яті, до яких йому дозволено доступ. В адресному просторі зберігаються код і дані процесу. При розподілі дискового простору для кожного процесу формується список відкритих файлів, аналогічним чином розподіляють пристрої введення-виведення.

Процеси забезпечують захист ресурсів, якими вони володіють. Наприклад, до адресного простору процесу неможливо безпосередньо звернутися з інших процесів (він є захищеним), а при роботі з файлами може бути задано режим, що забороняє доступ до файла всім процесам, крім поточного.

Розподіл процесорного часу між процесами необхідний через те, що процесор виконує інструкції одну за одною, послідовно в часі (тобто в конкретний момент часу на ньому може фізично виконуватися тільки один процес), а для користувача процеси мають виглядати як послідовності інструкцій, виконувані паралельно. Щоб домогтися такого ефекту, ОС надає

процесор кожному процесу на деякий короткий час, після чого перемикає процесор на інший процес; при цьому виконання процесів відновлюється з того місця, де їх було перервано. У багатопроцесорній системі процеси можуть виконуватися паралельно на різних процесорах.

Сучасні ОС крім багатозадачності можуть підтримувати багатопотоковість (multithreading), яка передбачає в рамках процесу наявність кількох окремих послідовностей інструкцій (потоків, threads), які для користувача виконуються паралельно, подібно до самих процесів в ОС. На відміну від процесів потоки не забезпечують захисту ресурсів (наприклад, вони спільно використовують адресний простір свого процесу).

3. Під час виконання програмного коду процесор бере інструкції й дані з оперативної (основної) пам'яті комп'ютера. При цьому така пам'ять відображається у вигляді масиву байтів, кожен з яких має адресу.

Основна пам'ять є одним із видів ресурсів, розподілюваних між процесами. ОС відповідає за виділення пам'яті під захищений адресний простір процесу і за вивільнення пам'яті після того, як виконання процесу буде завершено. Обсяг пам'яті, доступний процесу, може змінюватися в ході виконання, у цьому разі говорять про динамічний розподіл пам'яті.

ОС повинна забезпечувати можливість виконання програм, які окремо або в сукупності перевищують за обсягом доступну основну пам'ять. Для цього в ній має бути реалізована технологія віртуальної пам'яті. Така технологія дає можливість розміщувати в основній пам'яті тільки ті інструкції й дані процесу, які потрібні в поточний момент часу, при цьому вміст іншої частини адресного простору зберігається на диску.

4.Операційна система відповідає за керування пристроями введення-виведення, підключеними до комп'ютера. Підтримка таких пристроїв в ОС здійснюється на двох рівнях. До першого, нижчого, рівня належать драйвери пристроїв - програмні модулі, які керують пристроями конкретного типу з

урахуванням усіх їхніх особливостей. До другого рівня належить універсальний інтерфейс введення-виведення, зручний для використання у прикладних програмах.

ОС має реалізовувати загальний інтерфейс драйверів введення-виведення, через який вони взаємодіють з іншими компонентами системи. Такий інтерфейс дає змогу спростити додавання в систему драйверів для нових пристроїв.

Сучасні ОС надають великий вибір готових драйверів для конкретних периферійних пристроїв. Що більше пристроїв підтримує ОС, то більше в неї шансів на практичне використання.

5. Для користувачів ОС і прикладних програмістів дисковий простір надається у вигляді сукупності файлів, організованих у файлову систему.

Файл - це набір даних у файловій системі, доступ до якого здійснюється за іменем. Термін «файлова система» може вживатися для двох понять: принципу організації даних у вигляді файлів, які доступні користувачеві і конкретного набору структур даних (зазвичай відповідної частини диска - розділу), організованих відповідно до такого принципу, які користувачеві недоступні. У рамках ОС може бути реалізована одночасна підтримка декількох файлових систем.

Файлові системи розглядають на логічному і фізичному рівнях. Логічний рівень визначає зовнішнє подання системи як сукупності файлів (які звичайно перебувають у каталогах), а також виконання операцій над файлами і каталогами (створення, вилучення тощо). Фізичний рівень визначає принципи розміщення структур даних файлової системи на диску або іншому пристрої.

6. Мережні системи це сучасні операційні системи, які пристосовані до роботи в мережі. Їх називають ме-режними операційними системами . Засоби мережної підтримки дають ОС можливість:

♦ надавати локальні ресурси (дисковий простір, принтери тощо) у загальне користування через мережу, тобто функціонувати як сервер;

♦ звертатися до ресурсів інших комп'ютерів через мережу, тобто функціонувати як клієнт.

Реалізація функціональності сервера і клієнта базується на транспортних засобах, відповідальних за передачу даних між комп'ютерами відповідно до правил, обумовлених мережними протоколами.

Мережні ОС не приховують від користувача наявність мережі, мережна підтримка в них не визначає структуру системи, а збагачує її додатковими можливостями. Є також розподілені ОС, які дають змогу об'єднати ресурси декількох комп'ютерів у розподілену систему. Вона виглядає для користувача як один комп'ютер з декількома процесорами, що працюють паралельно. Розподілені та багатопроцесорні системи є двома основними категоріями ОС, які використовують декілька процесорів і мають багато спільного.

7. Під безпекою даних в ОС розуміють забезпечення надійності системи (захист даних від втрати у разі збоїв) і захист даних від несанкціонованого доступу (випадкового чи навмисного).

Для захисту від несанкціонованого доступу ОС має забезпечувати наявність засобів аутентифікації користувачів (такі засоби дають змогу з'ясувати, чи є користувач тим, за кого себе видає; зазвичай для цього використовують систему паролів) та їхньої авторизації (дозволяють перевірити права користувача, що пройшов аутентифікацію, на виконання певної операції).

8. Розрізняють два типи засобів взаємодії користувача з ОС: командний інтерпретатор (shell) і графічний інтерфейс користувача (GUI).

Командний інтерпретатор дає змогу користувачам взаємодіяти з ОС, використовуючи спеціальну командну мову (інтерактивно або через запуск на виконання командних файлів). Команди такої мови змушують ОС

виконувати певні дії (наприклад, запускати програми, працювати із файлами).

Графічний інтерфейс користувача надає йому можливість взаємодіяти з ОС, відкриваючи вікна і виконуючи команди за допомогою меню або кнопок. Підходи до реалізації графічного інтерфейсу доволі різноманітні: наприклад, у Windows-системах засоби його підтримки вбудовані в систему, а в UNIX вони є зовнішніми для системи і спираються на стандартні засоби керування введенням-виведенням.

Тема. Базові поняття архітектури операційних систем

1.Механізми і політика в ОС
2.Ядро системи та режими роботи процесора
3.Системне програмне забезпечення
4.Монолітні ОС
5.Багаторівневі ОС
6.Системи з мікроядром
7.Концепція віртуальних машин

1. Операційну систему можна розглядати як сукупність компонентів, кожен з яких відповідає за певні функції. Набір таких компонентів і порядок їхньої взаємодії один з одним та із зовнішнім середовищем визначається архітектурою операційної системи.

В ОС насамперед необхідно виділити набір фундаментальних можливостей, які надають її компоненти; ці базові можливості становлять механізм (mechanism). Механізми ОС можна використовувати по різному, отже спосіб використання механізму визначає політику (policy) ОС. Таким чином, механізм показує, що реалізовано компонентом, а політика - як це можна використати.

Коли за реалізацію механізму і політики відповідають різні компоненти (механізм відокремлений від політики), спрощується розробка системи і підвищується її гнучкість. Компонентам, що реалізують механізм, не повинна бути доступна інформація про причини та цілі його застосування; усе, що потрібно від них, - це виконувати призначену їм роботу. Для таких компонентів використовують термін «вільні від політики» (policy-free). Компоненти, відповідальні за політику, мають оперувати вільними від неї компонентами як будівельними блоками, для них недоступна інформація про деталі реалізації механізму.

Прикладом відокремлення механізму від політики є керування введенням-виведенням. Базові механізми доступу до периферійних пристроїв реалізують драйвери. Політику використання цих механізмів задає програмне забезпечення, що здійснює введення-виведення.

2. Базові компоненти ОС, які відповідають за найважливіші її функції, перебувають у пам'яті постійно і виконуються у привілейованому режимі, їх називають ядром операційної системи (operating system kernel).

До найважливіших функцій ОС, виконання яких покладають на ядро, належать обробка переривань, керування пам'яттю, керування введенням-виведенням. До надійності та продуктивності ядра висувають підвищені вимоги.

Основною характерною ознакою ядра є те, що воно виконується у привілейованому режимі. Особливості цього режиму полягають в тому, що для забезпечення ефективного керування ресурсами комп'ютера ОС повинна мати певні привілеї щодо прикладних програм. Треба, щоб прикладні програми не втручалися в роботу ОС, і водночас ОС повинна мати можливість втрутитися в роботу будь-якої програми, наприклад для перемикання процесора або розв'язання конфлікту в боротьбі за ресурси. Для реалізації таких привілеїв потрібна апаратна підтримка: процесор має підтримувати два режими роботи — привілейований (захищений режим,

режим ядра, kernel mode) і режим користувача (user mode). У режимі користувача недопустимі команди, які є критичними для роботи системи (перемикання задач, звертання до пам'яті за заданими межами, доступ до пристроїв введення-виведення тощо).

Після завантаження ядро перемикає процесор у привілейований режим і отримує повний контроль над комп'ютером. Кожне застосування запускається і виконується в режимі користувача, де воно не має доступу до ресурсів ядра й інших програм. Коли потрібно виконати дію, реалізовану в ядрі, застосування робить системний виклик (system call). Ядро перехоплює його, перемикає процесор у привілейований режим, виконує дію, перемикає процесор назад у режим користувача і повертає результат застосування.

Системний виклик виконується повільніше за виклик функції, реалізованої в режимі користувача, через те що процесор двічі перемикається між режимами. Для підвищення продуктивності в деяких ОС частина функціональності реалізована в режимі користувача, тому для доступу до неї системні виклики використовувати не потрібно.

3. Окрім ядра, важливими складовими роботи ОС є також застосування режиму користувача, які виконують системні функції. До такого системного програмного забезпечення належать:

- системні програми (утиліти), наприклад: командний інтерпретатор, програми резервного копіювання та відновлення даних, засоби діагностики й адміністрування;
- системні бібліотеки, у яких реалізовані функції, що використовуються у застосуваннях користувача.

Системне програмне забезпечення може розроблятися й постачатися окремо від ОС. Наприклад, може бути кілька реалізацій командного інтерпретатора, засобів резервного копіювання тощо. Системні програми і бібліотеки взаємодіють з ядром у такий самий спосіб, як і прикладні програми.

4. ОС, у яких усі базові функції сконцентровані в ядрі, називають монолітними системами. У разі реалізації монолітного ядра ОС стає продуктивнішою (процесор не перемикається між режимами під час взаємодії між її компонентами), але менш надійною (весь її код виконується у привілейованому режимі, і помилка в кожному з компонентів є критичною).

Монолітність ядра не означає, що всі його компоненти мають постійно перебувати у пам'яті. Сучасні ОС дають можливість динамічно розміщувати в адресному просторі ядра фрагменти його коду (модулі ядра). Реалізація модулів ядра дає можливість також досягти його розширюваності (для додання нової функціональності досить розробити і завантажити у пам'ять відповідний модуль).

5. Компоненти багаторівневих ОС утворюють ієрархію рівнів (шарів, layers), кожен з яких спирається на функції попереднього рівня. Найнижчий рівень безпосередньо взаємодіє з апаратним забезпеченням, на найвищому рівні реалізуються системні виклики.

У традиційних багаторівневих ОС передача керування з верхнього рівня на нижній реалізується як системний виклик. Верхній рівень повинен мати права на виконання цього виклику, перевірка цих прав виконується за підтримки апаратного забезпечення. Прикладом такої системи є ОС Multics, розроблена в 60-ті роки. Практичне застосування цього підходу сьогодні обмежене через низьку продуктивність.

Рівні можуть виділятися і в монолітному ядрі; у такому разі вони підтримуються програмно і ведуть до спрощення реалізації системи. У монолітном ядрі визначають рівні, перелічені нижче.

♦ Засоби абстрагування від устаткування, які взаємодіють із апаратним забез печенням безпосередньо, звільняючи від такої взаємодії інші компоненти системи.

♦ Базові засоби ядра, які відповідають за найфундаментальніші, найпростіші дії ядра, такі як запис блоку даних на диск. За допомогою цих засобів виконуються вказівки верхніх рівнів, пов'язані з керуванням ресурсами.

♦ Засоби керування ресурсами (або менеджери ресурсів), що реалізують основні функції ОС (керування процесами, пам'яттю, введенням-виведенням тощо). На цьому рівні приймаються найважливіші рішення з керування ресурсами, які виконуються з використанням базових засобів ядра.

♦ Інтерфейс системних викликів, який служить для реалізації зв'язку із системним і прикладним програмним забезпеченням.

Розмежування базових засобів ядра і менеджерів ресурсів відповідає відокремленню механізмів від політики в архітектурі системи. Базові засоби ядра визначають механізми функціонування системи, менеджери ресурсів реалізують політику.

6. Один із напрямів розвитку сучасних ОС полягає в тому, що у привілейованому режимі реалізована невелика частка функцій ядра, які є мікроядром (microkernel). Інші функції ОС виконуються процесами режиму користувача (серверними процесами, серверами). Сервери можуть відповідати за підтримку файлової системи, за роботу із процесами, пам'яттю тощо.

Мікроядро здійснює зв'язок між компонентами системи і виконує базовий розподіл ресурсів. Щоб виконати системний виклик, процес (клієнтський процес, клієнт) звертається до мікроядра. Мікроядро посилає серверу запит, сервер виконує роботу і пересилає відповідь назад, а мікроядро переправляє його клієнтові (рис. 2.1). Клієнтами можуть бути не лише процеси користувача, а й інші модулі ОС.

Перевагами мікроядрового підходу є:

♦ невеликі розміри мікроядра, що спрощує його розробку й налагодження;

♦ висока надійність системи, внаслідок того що сервери працюють у режимі користувача і в них немає прямого доступу до апаратного забезпечення;

♦ більша гнучкість і розширюваність системи (непотрібні компоненти не займають місця в пам'яті, розширення функціональності системи зводиться до додавання в неї нового сервера);

♦ можливість адаптації до умов мережі (спосіб обміну даними між клієнтом і сервером не залежить від того, зв'язані вони мережею чи перебувають на одному комп'ютері).

♦ Головним недоліком мікроядрового підходу є зниження продуктивності. Замість двох перемикань режиму процесора у разі системного виклику відбувається чотири (два — під час обміну між клієнтом і мікроядром, два - між сервером та мікроядром).

Зазначений недолік є, швидше, теоретичним, на практиці продуктивність і надійність мікроядра залежать насамперед від якості його реалізації. Так, в ОС QNX мікроядро займає кілька кілобайтів пам'яті й забезпечує мінімальний набір функцій, при цьому система за продуктивністю відповідає ОС реального часу.

7. У системах віртуальних машин програмним шляхом створюють копії апаратного забезпечення (відбувається його емуляція). Ці копії (віртуальні машини) працюють паралельно, на кожній із них функціонує програмне забезпечення, з яким взаємодіють прикладні програми і користувачі.

Уперше концепція віртуальних машин була реалізована в 70-ті роки в операційній системі VM фірми IBM. У СРСР варіант цієї системи (VM/370) був широко розповсюджений у 80-ті роки і мав назву Система віртуальних машин ЄС ЕОМ (СВМ ЄС). Ядро системи, яке називається монітором віртуальних машин (VM Monitor, МВМ), виконуєтьсяся на фізичній машині, безпосередньо взаємодіючи з її апаратним забезпеченням. Монітор (ситемна програма) забезпечує набір віртуальних машин (ВМ). Кожна ВМ є точною

копією апаратного забезпечення, на ній може бути запущена будь-яка ОС, розроблена для цієї архітектури. Найчастіше на ВМ встановлювали спеціальну однокористувацьку ОС CMS (підсистема діалогової обробки, ПДО). На різних ВМ могли одночасно функціонувати різні ОС.

Коли програма, написана для ПДО, виконувала системний виклик, його перехоплювала копія ПДО, запущена на відповідній віртуальній машині. Потім ПДО виконувала відповідні апаратні інструкції, наприклад інструкції введення-виведення для читання диска. Ці інструкції перехоплював МВМ і перетворював їх на апаратні інструкції фізичної машини.

Віртуальні машини спільно використовували ресурси реального комп'ютера; наприклад, дисковий простір розподілявся між ними у вигляді віртуальних дисків, названих мінідисками. ОС, запущена у ВМ, використовувала мінідиски так само, як фізичні диски.

Тема. Операційна система та її оточення

1.Взаємодія ОС і апаратного забезпечення
2.Засоби апаратної підтримки операційних систем
3.Апаратна незалежність і здатність до перенесення ОС
4.Взаємодія ОС і виконуваного програмного коду
5. Системні виклики та інтерфейс програмування застосувань
6.Програмна сумісність

1.Із означення ОС випливає, що вона реалізує зв'язок між апаратним забезпеченням комп'ютера (через інтерфейс апаратного забезпечення) і програмами користувача (через інтерфейс прикладного програмування). Інтерфейс апаратного забезпечення слід розуміти, як набір машинних команд центрального процесора та всіх контролерів зовнішніх пристроїв ситеми. Інтерфейс прикладного програмування це набір правил (команд, директив)

для виклику системних функцій ОС в прикладну програму, що значно полегшує створення прикладних програм.

2.Сучасні апаратні архітектури комп'ютерів забезпечують базові засоби підтримки операційних систем. До них належать: система переривань, привілейований режим процесора, засоби перемикання задач, підтримка керування пам'яттю (механізми трансляції адрес, захист пам'яті), системний таймер, захист пристроїв введення-виведення, базова система введення-виведення (BIOS). Розглянемо ці засоби докладніше.

Система переривань є основним механізмом, що забезпечує функціонування ОС. За допомогою переривань процесор отримує інформацію про події, не пов'язані з його основною роботою. Переривання бувають двох типів: апаратні і програмні.

Апаратне переривання - це спеціальний сигнал (запит переривання, IRQ), що передається процесору від апаратного пристрою.

До апаратних переривань належать:

- переривання введення-виведення, що надходять від контролера периферійного пристрою; наприклад, таке переривання генерує контролер клавіатури при натисканні на клавішу;
- переривання, пов'язані з апаратними або програмними помилками (такі переривання виникають, наприклад, у разі збою контролера диска, доступу до забороненої області пам'яті або ділення на нуль).

Програмні переривання генерує прикладна програма, виконуючи спеціальну інструкцію переривання. Така інструкція є в системі команд більшості процесорів. Обробка програмних переривань процесором не відрізняється від обробки апаратних переривань.

Якщо переривання відбулося, то процесор негайно передає керування спеціальній процедурі - оброблювачеві переривання. Після виходу з оброблювача процесор продовжує виконання інструкцій перерваної програми. Розрізняють два типи переривань залежно від того, яка інструкція

буде виконана після виходу з оброблювача: для відмов (faults) повторюється інструкція, що спричинила переривання, для пасток (traps) — виконується наступна інструкція. Усі переривання введення-виведення і програмні переривання належать до категорії пасток, більшість переривань через помилки є відмовами.

За встановлення оброблювачів переривань зазвичай відповідає ОС. Можна сказати, що сучасні ОС керовані перериваннями (interrupt-driven), бо, якщо ОС не зайнята виконанням якої-небудь задачі, вона очікує на переривання, яке й залучає її до роботи.

Для реалізації привілейованого режиму процесора в одному з його регістрів передбачено спеціальний біт (біт режиму), котрий показує, у якому режимі перебуває процесор. У разі програмного або апаратного переривання процесор автоматично перемикається у привілейований режим, і саме тому ядро ОС (яке складається з оброблювачів переривань) завжди отримує керування в цьому режимі. За будь-якої спроби безпосередньо виконати привілейовану інструкцію в режимі користувача відбувається апаратне переривання.

Засоби перемикання задач дають змогу зберігати вміст регістрів процесора (контекст задачі) у разі припинення задачі та відновлювати дані перед її подальшим виконанням.

Механізм трансляції адрес забезпечує перетворення адрес пам'яті, з якими працює програма, в адреси фізичної пам'яті комп'ютера. Апаратне забезпечення генерує фізичну адресу, використовуючи спеціальні таблиці трансляції.

Захист пам'яті забезпечує перевірку прав доступу до пам'яті під час кожної спроби його отримати. Засоби захисту пам'яті інтегровані з механізмами трансляції адрес: у таблицях трансляції утримується інформація про права, необхідні для їхнього використання, і про ліміт (розміри ділянки пам'яті, до якої можна отримати доступ з їхньою допомогою). Неможливо

одержати доступ до пам'яті поверх ліміту або за відсутності прав на використання таблиці трансляції.

Системний таймер є апаратним пристроєм, який генерує переривання таймера через певні проміжки часу. Такі переривання обробляє ОС; інформацію від таймера найчастіше використовують для визначення часу перемикання задач.

Захист пристроїв введення-виведення ґрунтується на тому, що всі інструкції введення-виведення визначені як привілейовані. Прикладні програми здійснюють введення-виведення не прямо, а за посередництвом ОС.

Базова система введення-виведення (BIOS) — службовий програмний код, що зберігається в постійному запам'ятовувальному пристрої і призначений для ізоляції ОС від конкретного апаратного забезпечення. Зазначимо, що засоби BIOS не завжди дають змогу використати всі можливості архітектури: наприклад, процедури BIOS для архітектури IA-32 не працюють у захищеному режимі. Тому сучасні ОС використовують їх тільки для початкового завантаження системи.

3.Компоненти ядра, які відповідають за безпосередній доступ до апаратного забезпечення, виділено в окремий рівень абстрагування від устаткування, що взаємодіє з іншою частиною системи через стандартні інтерфейси. Тим самим спрощується досягнення апаратної незалежності ОС.

Рівень абстрагування від устаткування відображає такі особливості архітектури, як число процесорів, типи їхніх регістрів, розрядність і організація пам'яті тощо. Що більше відмінностей між апаратними архітектурами, для яких призначена ОС, то складніша розробка коду цього рівня.

Крім рівня абстрагування від устаткування, від апаратного забезпечення залежать драйвери зовнішніх пристроїв. Такі драйвери проектують заздалегідь як апаратно-залежні, їх можна додавати та вилучати

за потребою; для доступу до них зазвичай використовують універсальний інтерфейс.

Здатність до перенесення ОС визначається обсягом робіт, необхідних для того, щоб система могла працювати на новій апаратній платформі. ОС з такими властивостями має відповідати певним вимогам.

- Більша частина коду операційної системи має бути написана мовою високого рівня (звичайно для цього використовують мови С і С++, компілятори яких розроблені для більшості архітектур). Використання мови асемблера допустиме лише тоді, коли продуктивність компонента є критичною для системи.
- Код, що залежить від апаратного забезпечення (рівень абстрагування від устаткування) має бути відокремлений від іншої частини системи так, щоб у разі переходу на іншу архітектуру потрібно було переписувати тільки цей рівень.

4.У роботі в режимі користувача часто необхідне виконання дій, реалізованих у ядрі ОС (наприклад, під час запису на диск із прикладної програми). Для цього треба забезпечити взаємодію коду режиму користувача та ОС. Програмною основою такої взаємодії є системні виклики та інтерфейс програмування застосувань (API).

5.Системний виклик - це засіб доступу до певної функції ядра операційної системи із прикладних програм. Набір системних викликів визначає дії, які ядро може виконати за запитом процесів користувача. Системні виклики задають інтерфейс між прикладною програмою і ядром ОС.

Розглянемо послідовність виконання системного виклику.

1. Припустимо, що для процесу, який виконується в режимі користувача, потрібна функція, реалізована в ядрі системи.

2. Для того щоб звернутися до цієї функції, процес має передати керування ядру ОС, для чого необхідно задати параметри виклику і виконати програмне переривання (інструкцію системного виклику). Відбувається перехід у привілейований режим.

3. Після отримання керування ядро зчитує параметри виклику і визначає, що потрібно зробити.

4. Після цього ядро виконує потрібні дії, зберігає в пам'яті значення, які слід повернути, і передає керування програмі, що його викликала. Відбувається перехід назад у режим користувача.

5. Програма зчитує з пам'яті повернені значення і продовжує свою роботу.

6. Кожний системний виклик спричиняє перехід у привілейований режим і назад (у мікроядровій архітектурі, як було зазначено вище, таких переходів може бути і більше).

Є два основних способи передачі параметрів у системний виклик.

- передача параметрів у регістри процесора;
- занесення параметрів у певну ділянку пам'яті й передача покажчика на неї в регістрі процесора.

Системні виклики призначені для безпосереднього доступу до служб ядра ОС. На практиці вони не вичерпують (а іноді й не визначають) ті функції, які можна використати у прикладних програмах. Для доступу до засобів системних бібліотек ОС використовують інтерфейс програмування застосувань (Application Programming Interface, API).

Взаємозв'язок між функціями API і системними викликами неоднаковий у різних ОС.

По-перше, кожному системному виклику може бути поставлена у відповідність бібліотечна функція, єдиним завданням якої є виконання цього виклику. Таку функцію називають пакувальником системного виклику (system call wrapper). Для програміста в цьому разі набір функцій API виглядає як сукупність таких пакувальників і додаткових функцій,

реалізованих бібліотеками повністю або частково в режимі користувача. Це рішення прийняте за основу в UNIX; у такому разі прийнято говорити про використання системних викликів у прикладних програмах (насправді у програмах викликають пакувальники системних викликів).

По-друге, можна надати для використання у прикладних програмах універсальний інтерфейс програмування застосувань (API режиму користувача) і повністю сховати за ним набір системних викликів. Для програміста кожна функція такого API є бібліотечною функцією режиму користувача, пакувальника в цьому разі немає, відомості про системні виклики є деталями реалізації ОС. Це властиве Windows-системам, де подібний універсальний набір функцій називають Win32 API .

6.Дотепер ми розглядали виконання в ОС програм, розроблених спеціально для неї. Іноді буває необхідно виконати в середовищі ОС програми, розроблені для інших ОС і, можливо, для іншої апаратної архітектури. У цьому разі виникає проблема програмної сумісності.

Програмна сумісність означає можливість виконувати в середовищі однієї операційної системи програми, розроблені для іншої ОС. Розрізняють сумісність на рівні вихідних текстів (можливість перенесення вихідних текстів) та бінарну сумісність (можливість перенесення виконуваного коду).

Для сумісності на рівні вихідних текстів необхідно, щоб для всіх ОС існувала реалізація компілятора мови і API, що його використовує програма.

Сьогодні таку сумісність забезпечує стандартизація розробки програмного забезпечення, а саме:

- наявність стандарту на мови програмування (насамперед на C і C++) і стандартних компіляторів;
- наявність стандарту на інтерфейс операційної системи (API).

Робота зі стандартизації інтерфейсу операційних систем відбувається у рамках проекту POSIX (Portable Operating System Interface). Найбільш важливим стандартом є POSIX 1003.1 , який описує набір бібліотечних

процедур (таких, як відкриття файла, створення нового процесу тощо), котрі мають бути реалізовані в системі. Цей процес стандартизації триває дотепер, останньою редакцією стандарту є базова специфікація Open Group/IEEE . Зазначені стандарти відображають традиційний набір засобів, реалізованих в UNIX-сумісних системах.

Завдання забезпечення бінарної сумісності виникає тоді, коли потрібно запустити на виконання файл прикладної програми у середовищі іншої операційної системи. Таке завдання значно складніше в реалізації, найпоширеніший підхід до його розв'язання - емуляція середовища виконання. У цьому разі програма запускається під керуванням іншої програми — емулятора, який забезпечує динамічне перетворення інструкцій програми в інструкції потрібної архітектури. Прикладом такого емулятора є програма wine, яка дає можливість запускати програми, розроблені для Win32 API, у середовищі Linux через емуляцію функцій Win32 API системними викликами Linux.

Лекція №3. Тема: особливості архітектури: UNIX I Linux

План:

1.Базова архітектура UNIX

2.Архітектура Linux: призначення ядра Linux та його особливості, модулі ядра, особливості системних бібліотек, застосування користувача

1.UNIX є прикладом досить простої архітектури ОС. Більша частина функціональності цієї системи міститься в ядрі, ядро спілкується із прикладними програмами за допомогою системних викликів.

Система складається із трьох основних компонентів: підсистеми керування процесами, файлової підсистеми та підсистеми введення-виведення.

Підсистема керування процесами контролює створення та вилучення процесів, розподілення системних ресурсів між ними, міжпроцесову взаємодію, керування пам'яттю.

Файлова підсистема забезпечує єдиний інтерфейс доступу до даних, розташованих на дискових накопичувачах, і до периферійних пристроїв. Такий інтерфейс є однією з найважливіших особливостей UNIX. Одні й ті самі системні виклики використовують як для обміну даними із диском, так і для виведення на термінал або принтер (програма працює із принтером так само, як із файлом). При цьому файлова система переадресовує запити відповідним модулям підсистеми введення-виведення, а ті - безпосередньо периферійним пристроям. Крім того, файлова підсистема контролює права доступу до файлів, які значною мірою визначають привілеї користувача в системі.

Підсистема введення-виведення виконує запити файлової підсистеми, взаємодіючи з драйверами пристроїв. В UNIX розрізняють два типи пристроїв: символьні (наприклад, принтер) і блокові (наприклад, жорсткий диск). Основна відмінність між ними полягає в тому, що блоковий пристрій допускає прямий доступ. Для підвищення продуктивності роботи із блоковими пристроями використовують буферний кеш - ділянку пам'яті, у якій зберігаються дані, зчитані з диска останніми. Під час наступних звертань до цих даних вони можуть бути отримані з кеша.

Сучасні UNIX-системи дещо відрізняються за своєю архітектурою.

♦ У них виділено окремий менеджер пам'яті, відповідальний за підтримку віртуальної пам'яті.

♦ Стандартом для реалізації інтерфейсу файлової системи є віртуальна файлова система, що абстрагує цей інтерфейс і дає змогу організувати підтримку різних типів файлових систем.

♦ У цих системах підтримується багатопроцесорна обробка, а також багатопото-ковість.

Базові архітектурні рішення, такі як доступ до всіх пристроїв введення-виведення через інтерфейс файлової системи або організація системних викликів, залишаються незмінними в усіх реалізаціях UNIX.

2. В ОС Linux можна виділити три основні частини:

- ядро, яке реалізує основні функції ОС (керування процесами, пам'яттю, вве-денням-виведенням тощо);
- системні бібліотеки, що визначають стандартний набір функцій для використання у застосуваннях (виконання таких функцій не потребує переходу в привілейований режим);
- системні утиліти (прикладні програми, які виконують спеціалізовані задачі).

Призначення ядра Linux і його особливості

Linux реалізує технологію монолітного ядра. Весь код і структури даних ядра перебувають в одному адресному просторі. У ядрі можна виділити кілька функціональних компонентів [63].

- Планувальник процесів - відповідає за реалізацію багатозадачності в системі (обробка переривань, робота з таймером, створення і завершення процесів, перемикання контексту).
- Менеджер пам'яті - виділяє окремий адресний простір для кожного процесу і реалізує підтримку віртуальної пам'яті.
- Віртуальна файлова система - надає універсальний інтерфейс взаємодії з різними файловими системами та пристроями введення-виведення.
- Драйвери пристроїв - забезпечують безпосередню роботу з периферійними пристроями. Доступ до них здійснюється через інтерфейс віртуальної файлової системи.
- Мережний інтерфейс - забезпечує доступ до реалізації мережних протоколів і драйверів мережних пристроїв.

- Підсистема міжпроцесової взаємодії - пропонує механізми, які дають змогу різним процесам у системі обмінюватися даними між собою.

Деякі із цих підсистем є логічними компонентами системи, вони завантажуються у пам'ять разом із ядром і залишаються там постійно. Компоненти інших підсистем (наприклад, драйвери пристроїв) вигідно реалізовувати так, щоб їхній код міг завантажуватися у пам'ять на вимогу. Для розв'язання цього завдання Linux підтримує концепцію модулів ядра.

Модулі ядра

Ядро Linux дає можливість на вимогу завантажувати у пам'ять і вивантажувати з неї окремі секції коду. Такі секції називають модулями ядра (kernel modules) і виконують у привілейованому режимі. Модулі ядра надають низку переваг.

- Код модулів може завантажуватися в пам'ять у процесі роботи системи, що спрощує налагодження компонентів ядра, насамперед драйверів.
- З'являється можливість змінювати набір компонентів ядра під час виконання: ті з них, які в цей момент не використовуються, можна не завантажувати у пам'ять.
- Модулі є винятком із правила, за яким код, що розширює функції ядра, відповідно до ліцензії Linux має бути відкритим. Це дає змогу виробникам апаратного забезпечення розробляти драйвери під Linux, навіть якщо не заплановано надавати доступ до їхнього вихідного коду.
- Підтримка модулів у Linux складається із трьох компонентів.
- Засоби керування модулями дають можливість завантажувати модулі у пам'ять і здійснювати обмін даними між модулями та іншою частиною ядра.
- Засоби реєстрації драйверів дозволяють модулям повідомляти іншу частину ядра про те, що новий драйвер став доступним.
- Засоби розв'язання конфліктів дають змогу драйверам пристроїв резервувати апаратні ресурси і захищати їх від випадкового використання іншими драйверами.

Один модуль може зареєструвати кілька драйверів, якщо це потрібно (наприклад, для двох різних механізмів доступу до пристрою).

Модулі можуть бути завантажені заздалегідь — під час старту системи (завантажувальні модулі) або у процесі виконання програми, яка викликає їхні функції. Після завантаження код модуля перебуває в тому ж самому адресному просторі, що й інший код ядра. Помилка в модулі є критичною для системи.

Особливості системних бібліотек

Системні бібліотеки Linux є динамічними бібліотеками, котрі завантажуються у пам'ять тільки тоді, коли у них виникає потреба. Вони виконують ряд функцій:

- реалізацію пакувальників системних викликів;
- розширення функціональності системних викликів (до таких бібліотек належить бібліотека введення-виведення мови С, яка реалізує на основі системних викликів такі функції, як prlntf ( ));
- реалізацію службових функцій режиму користувача (сортування, функції обробки рядків тощо).

Застосування користувача

Застосування користувача в Linux використовують функції із системних бібліотек і через них взаємодіють із ядром за допомогою системних викликів.

Тема. Особливості архітектури: Windows XP

1.Компоненти режиму ядра Windows XP: рівень абстрагування від устаткування, ядро, виконавча система, драйвери пристроїв, віконна і графічна підсистеми

2.Компоненти режиму користувача: бібліотека системного інтерфейсу, підсистеми середовища,наперед визначені системні процеси, застосування користувача

3.Об`єктна архітектура Windows XP: структура заголовка об`акта, об`єкти типу, методи об`єктів, простір імен об`єктів

1.Деякі компоненти Windows XP виконуються у привілейованому режимі, інші компоненти - у режимі користувача.

У традиційному розумінні ядро ОС складають усі компоненти привілейованого режиму, однак у Windows XP поняття ядра закріплене тільки за одним із цих компонентів.

Рівень абстрагування від устаткування

У Windows XP реалізовано рівень абстрагування від устаткування (у цій системі його називають HAL, hardware abstraction layer). Для різних апаратних конфігурацій фірма Microsoft або сторонні розробники можуть постачати різні реалізації HAL.

Хоча код HAL є дуже ефективним, його використання може знижувати продуктивність застосувань мультимедіа. У такому разі використовують спеціальний пакет DirectX, який дає змогу прикладним програмам звертатися безпосередньо до апаратного забезпечення, обминаючи HAL та інші рівні системи.

Ядро

Ядро Windows XP відповідає за базові операції системи. До його основних функцій належать:

- перемикання контексту, збереження і відновлення стану потоків;
- планування виконання потоків;
- реалізація засобів підтримки апаратного забезпечення, складніших за засоби HAL (наприклад, передача керування оброблювачам переривань).

Ядро Windows XP відповідає базовим службам ОС і надає набір механізмів для реалізації політики керування ресурсами.

Основним завданням ядра є якомога ефективніше завантаження процесорів системи. Ядро постійно перебуває в пам'яті, послідовність

виконання його інструкцій може порушити тільки переривання (під час виконання коду ядра багатозадачність не підтримується). Для прискорення роботи ядро ніколи не перевіряє правильність параметрів, переданих під час виклику його функцій.

Windows XP не можна віднести до якогось певного класу ОС. Хоча за функціональністю ядро системи відповідає поняттю мікроядра, для самої ОС не характерна класична мікроядрова архітектура, оскільки у привілейованому режимі виконуються й інші її компоненти.

Виконавча система

Виконавча система (ВС) Windows XP (Windows XP Executive) - це набір компонентів, відповідальних за найважливіші служби ОС (керування пам'яттю, процесами і потоками, введенням-виведенням тощо).

Компонентами ВС є передусім базові засоби підтримки. Ці засоби використовують у всій системі.

♦ Менеджер об'єктів — відповідає за розподіл ресурсів у системі, підтримуючи їхнє універсальне подання через об'єкти.

♦ Засіб локального виклику процедур (LPC) — забезпечує механізм зв'язку між процесами і підсистемами на одному комп'ютері.

Інші компоненти ВС реалізують найважливіші служби Windows XP. Зупинимося на деяких із них.

♦ Менеджер процесів і потоків — створює та завершує процеси і потоки, а також розподіляє для них ресурси.

♦ Менеджер віртуальної пам'яті — реалізує керування пам'яттю в системі, насамперед підтримку віртуальної пам'яті.

♦ Менеджер введення-виведення — керує периферійними пристроями, надаючи іншим компонентам апаратно-незалежні засоби введення-виведення. Цей менеджер реалізує єдиний інтерфейс для драйверів пристроїв.

♦ Менеджер кеша — керує кешуванням для системи введення-виведення. Часто використовувані блоки диска тимчасово зберігаються в

пам'яті, наступні операції введення-виведення звертаються до цієї пам'яті, внаслідок чого підвищується продуктивність.

♦ Менеджер конфігурації - відповідає за підтримку роботи із системним реєстром (registry) - ієрархічно організованим сховищем інформації про налаштування системи і прикладних програм.

♦ Довідковий монітор захисту - забезпечує політику безпеки на ізольованому комп'ютері, тобто захищає системні ресурси.

Драйвери пристроїв

У Windows XP драйвери не обов'язково пов'язані з апаратними пристроями. Застосування, якому потрібні засоби, доступні в режимі ядра, завжди варто оформляти як драйвер. Це пов'язане з тим, що для зовнішніх розробників режим ядра доступний тільки з коду драйверів.

Віконна і графічна підсистеми

Віконна і графічна підсистеми відповідають за інтерфейс користувача - роботу з вікнами, елементами керування і графічним виведенням.

- Менеджер вікон - реалізує високорівневі функції. Він керує віконним виведенням, обробляє введення з клавіатури або миші й передає застосуванням повідомлення користувача.
- Інтерфейс графічних пристроїв (Graphical Device Interface, GDI) — складається з набору базових операцій графічного виведення, які не залежать від конкретного пристрою (креслення ліній, відображення тексту тощо).
- Драйвери графічних пристроїв (відеокарт, принтерів тощо) — відповідають за взаємодію з контролерами цих пристроїв.

Під час створення вікон або елементів керування запит надходить до менеджера вікон, який для виконання базових графічних операцій звертається до GDI. Потім запит передається драйверу пристрою, затим - апаратному забезпеченню через HAL.

2.Компоненти режиму користувача не мають прямого доступу до апаратного забезпечення, їхній код виконується в ізольованому адресному просторі. Більша частина коду режиму користувача перебуває в динамічних бібліотеках, які у Windows називають DLL (dynamic –link libraies).

Бібліотека системного інтерфейсу

Для доступу до засобів режиму ядра в режимі користувача необхідно звертатися до функцій бібліотеки системного інтерфейсу (ntdll.dll). Ця бібліотека надає набір функцій-перехідників, кожній з яких відповідає функція режиму ядра (системний виклик). Застосування зазвичай не викликають такі функції безпосередньо, за це відповідають підсистеми середовища.

Підсистеми середовища

Підсистеми середовища надають застосуванням користувача доступ до служб операційної системи, реалізуючи відповідний API. Ми зупинимося на двох підсистемах середовища: Win32 і POSIX.

Підсистема Win32, яка реалізує Win32 API, є обов'язковим компонентом Windows XP. До неї входять такі компоненти:

♦ процес підсистеми Win32 (csrss.exe), що відповідає, зокрема, за реалізацію текстового (консольного) введення-виведення, створення і знищення процесів та потоків;

♦ бібліотеки підсистеми Win32, які надають прикладним програмам функції Win32 API. Найчастіше використовують бібліотеки gdi32.dll (низькорівневі графічні функції, незалежні від пристрою), user32.dll (функції інтерфейсу користувача) і kernel32.dll (функції, реалізовані у ВС і ядрі).

Після того як застосування звернеться до функції Win32 API, спочатку буде викликана відповідна функція з бібліотеки підсистеми Win32. Розглянемо варіанти виконання такого виклику.

1.Якщо функції потрібні тільки ресурси її бібліотеки, виклик повністю виконується в адресному просторі застосування без переходу в режим ядра.

2.Якщо потрібен перехід у режим ядра, з коду бібліотеки підсистеми виконується системний виклик. Так відбувається у більшості випадків, наприклад під час створення вікон або елементів керування.

3.Функція бібліотеки підсистеми може звернутися до процесу підсистеми Win32, при цьому:

♦ коли потрібна тільки функціональність, реалізована даним процесом, переходу в режим ядра не відбувається;

♦ коли потрібна функціональність режиму ядра, процес підсистеми Win32 виконує системний виклик аналогічно до варіанта 2.

Зазначимо, що до виходу Windows NT 4.0 (1996 рік) віконна і графічна підсистеми працювали в режимі користувача як частина процесу підсистеми Win32 (тобто виклики базових графічних функцій Win32 API оброблялися відповідно до варіанта 3). Надалі для підвищення продуктивності реалізацію цих підсистем було перенесено в режим ядра [14].

Підсистема POSIX працює в режимі користувача й реалізує набір функцій, визначених стандартом POSIX 1003.1. Оскільки застосування, або прикладні програми (арріісатіопз), написані для однієї підсистеми, не можуть використати функції інших, у POSIX-програмах не можна користуватися засобами Win32 API (зокрема, графічними та мережними функціями), що знижує важливість цієї підсистем-и. Підсистема POSIX не є обов'язковим компонентом Windows XP.

Наперед визначені системні процеси

Ряд важливих процесів користувача система запускає автоматично до закінчення завантаження. Розглянемо деякі з них.

♦Менеджер сесій (Sesion Manager, smss.exe) створюється в системі першим. Він запускає інші важливі процеси (процес підсистеми Win32, процес реєстрації в системі тощо), а також відповідає за їхнє повторне виконання під час аварійного завершення.

♦ Процес реєстрації в системі (winlogon.exe) відповідає за допуск користувача в систему. Він відображає діалогове вікно для введення пароля,

після введення передає пароль у підсистему безпеки і в разі успішної його верифікації запускає засоби створення сесії користувача.

♦ Менеджер керування службами (Services Control Manager, services.exe) відповідає за автоматичне виконання певних застосувань під час завантаження системи. Застосування, які будуть виконані при цьому, називають службами (services). Такі служби, як журнал подій, планувальник задач, менеджер друкування, постачають разом із системою. Крім того, є багато служб сторонніх розробників; так зазвичай реалізовують серверні застосування (сервери баз даних, веб-сервери тощо).

Застосування користувача

Застосування користувача можуть бути створені для різних підсистем середовища. Такі застосування використовують тільки функції відповідного API. Виклики цих функцій перетворюються в системні виклики за допомогою динамічних бібліотек підсистем середовища.

3.Керування ресурсами у Windows XP реалізується із застосуванням концепції об'єктів. Об'єкти надають універсальний інтерфейс для доступу до системних ресурсів, для яких передбачено спільне використання, зокрема таких, як процеси, потоки, файли і розподілювана пам'ять. Концепція об'єктів забезпечує такі переваги:

- імена об'єктів організовані в єдиний простір імен, де їх легко знаходити.
- доступ до всіх об'єктів здійснюється однаково. Після створення нового об'єкта або після отримання доступу до наявного менеджер об'єктів повертає у застосування дескриптор об'єкта (oЬiect handle).
- забезпечено захист ресурсів. Кожну спробу доступу до об'єкта розглядає підсистема захисту - без неї доступ до об'єкта, а отже і до ресурсу, отримати неможливо.

Менеджер об'єктів відповідає за створення, підтримку та ліквідацію об'єктів, задає єдині правила для їхнього іменування, збереження й

забезпечення захисту. Підсистеми середовища звертаються до менеджера об'єктів безпосередньо або через інші сервіси ВС. Наприклад, під час запуску застосування підсистема Win32 викликає менеджер процесів для створення нового процесу. В свою чергу менеджер процесів звертається до менеджера об'єктів для створення об'єкта, що представляє процес.

Об'єкти реалізовано як структури даних в адресному просторі ядра. При перезавантаженні системи вміст усіх об'єктів губиться.

Структура заголовка об'єкта

Об'єкти складаються з двох частин: заголовка і тіла об'єкта. У заголовку міститься інформація, загальна для всіх об'єктів, у тілі - специфічна для об'єктів конкретного типу.

До атрибутів заголовка об'єкта належать:

- ім'я об'єкта і його місце у просторі імен;
- дескриптор захисту (визначає права, необхідні для використання об'єкта);
- витрата квоти (ціна відкриття дескриптора об'єкта, дає змогу регулювати кількість об'єктів, які дозволено створювати);
- список процесів, що дістали доступ до дескрипторів об'єкта.

Менеджер об'єктів здійснює керування об'єктами на підставі інформації з їхніх заголовків.

Об'єкти типу

Формат і вміст тіла об'єкта визначається його типом. Новий тип об'єктів може бути визначений будь-яким компонентом ВС. Існує визначений набір типів об'єктів, які створюються під час завантаження системи (такі об'єкти, наприклад, відповідають процесам, відкритим файлам, пристроям введення-виведення).

Частина характеристик об'єктів є загальними для всіх об'єктів цього типу. Для зберігання відомостей про такі характеристики використовують спеціальні об'єкти типу (type objects). У такому об'єкті, зокрема, зберігають:

- ім'я типу об'єкта («процес», «потік», «відкритий файл» тощо);

- режими доступу (залежать від типу об'єкта: наприклад, для файла такими режимами можуть бути «читання» і «запис»).

Об'єкти типу недоступні в режимі користувача.

Методи об'єктів

Коли компонент ВС створює новий тип об'єкта, він може зареєструвати у диспетчері об'єктів один або кілька методів. Після цього диспетчер об'єктів викликає ці методи на певних етапах життєвого циклу об'єкта. Наведемо деякі з методів об'єктів:

- open - викликається при відкритті дескриптора об'єкта;
- close - викликається при закритті дескриптора об'єкта;
- delete - викликається перед вилученням об'єкта з пам'яті.

Покажчики на код реалізації методів також зберігаються в об'єктах типу.

Простір імен об'єктів

Усі імена об'єктів у ВС розташовані в глобальному просторі імен, тому будь-який процес може відкрити дескриптор об'єкта, вказавши його ім'я. Простір імен об'єктів має ієрархічну структуру, подібно до файлової системи. Аналогом каталогу файлової системи в такому просторі імен є каталог об'єктів. Він містить імена об'єктів (зокрема й інших каталогів). Перелічимо деякі наперед визначені імена каталогів:

- Device — імена пристроїв введення-виведення;
- Driver - завантажені драйвери пристроїв;
- ObjectTypes- об'єкти типів.

Простір імен об'єктів, як і окремі об'єкти, не зберігається після перезавантаження системи.

Висновки до розділу 1

- Операційна система — це рівень програмного забезпечення, що перебуває між рівнями прикладних програм й апаратного забезпечення

комп'ютера. Головне її призначення -зробити використання комп'ютерної системи простішим і підвищити ефективність її роботи.

♦ До основних функціональних компонентів ОС належать: керування процесами, керування пам'яттю, керування введенням-виведенням, керування файлами і підтримка файлових систем, мережна підтримка, забезпечення захисту даних, реалізація інтерфейсу користувача.

▪ Архітектура ОС визначає набір її компонентів, а також порядок їхньої взаємодії один з одним та із зовнішнім середовищем.

▪ Найважливішим для вивчення архітектури ОС є поняття ядра системи. Основною характеристикою ядра є те, що воно виконується у привілейованому режимі.

▪ Основними типами архітектури ОС є монолітна архітектура й архітектура на базі мікроядра. Монолітна архітектура вимагає, щоб головні функції системи були сконцентровані в ядрі, найважливішою її перевагою є продуктивність. У системах на базі мікроядра в привілейованому режимі виконуються тільки базові функції, основними перевагами таких систем є надійність і гнучкість.

▪ Операційна система безпосередньо взаємодіє з апаратним забезпеченням комп'ютера. Сучасні комп'ютерні архітектури пропонують багато засобів підтримки роботи операційних систем. Для зв'язку з апаратним забезпеченням в ОС виділяється рівень абстрагування від устаткування.

▪ Операційна система взаємодіє із прикладними програмами. Вона надає набір системних викликів для доступу до функцій, реалізованих у ядрі. Для прикладних програм системні виклики разом із засобами системних бібліотек доступні через інтерфейс програмування застосувань (API).

Контрольні запитання та завдання

1. Які основні функції операційної системи? Чи немає між ними протиріч?

2. Наведіть кілька прикладів просторового і часового розподілу ресурсів комп'ютера. Від чого залежить вибір того чи іншого методу розподілу?

3. У чому полягає основна відмінність багатозадачних пакетних систем від систем з розподілом часу? Як можна в рамках однієї системи об'єднати можливості обох зазначених систем?

4. Чому більшість вбудованих систем розроблено як системи реального часу? Наведіть приклади вбудованих систем, для яких підтримка режиму реального часу не є обов'язковою.

5. Що спільного й у чому відмінності між мережною і розподіленою операційними системами? Яка з них складніша в реалізації і чому?

6. Перелічіть причини, за якими ядро ОС має виконуватися в привілейованому режимі процесора.

7. Чи може процесор переходити у привілейований режим під час виконання програми користувача? Чи може така програма виконуватися виключно в привілейованому режимі?

8. У чому полягає головний недолік традиційної багаторівневої архітектури ОС? Чи має такий недолік архітектура з виділенням рівнів у монолітному ядрі?

9. Чому перехід до використання мікроядрової архітектури може спричинити зниження продуктивності ОС?

10. Автор Linux Лінус Торвальдс стверджує, що мобільність Linux має поширюватися на системи з «прийнятною» (reasonable) архітектурою. Які апаратні засоби повинна підтримувати така архітектура?

11. Наведіть переваги і недоліки реалізації взаємодії прикладної програми з операційною системою в Linux і Windows XP.

12. Чи не суперечить використання модулів ядра принципам монолітної архітектури Linux? Поясніть свою відповідь.

13. Перелічіть переваги і недоліки архітектури ОС, відповідно до якої віконна і графічна підсистеми в Windows XP виконуються в режимі ядра.

14. Чому деякі діагностичні утиліти Windows XP складаються з прикладної програми і драйвера пристрою?

Розділ3. Керування процесами і потоками та їх планування

Лекція №1. Тема: базові поняття процесів і потоків

1.Процеси і потоки в сучасних ОС
2.Моделі процесів і потоків
3.Складові елементи процесів і потоків
4.Поняття паралелізму та його види
5.Переваги і недоліки багатопотоковості

1.У сучасній операційній системі одночасно виконуються код ядра (що належить до його різних підсистем) і код програм користувача. При цьому відбуваються різні дії: одні програми і підсистеми виконують інструкції процесора, інші зайняті введенням-виведенням, ще деякі очікують на запити від користувача або інших застосувань. Для спрощення керування цими діями в системі доцільно виділити набір елементарних активних елементів і визначити інтерфейс взаємодії ОС із цими елементами. Якщо активний елемент системи зв'язати із програмою, що виконується, то отримаємо абстракцію, яка називається процесом.

Під процесом розуміють абстракцію ОС, яка об'єднує все необхідне для виконання однієї програми в певний момент часу. Власне процес це програма, що виконується в обчислювальній системі.

Програма - це деяка послідовність машинних команд, що зберігається на диску, в разі необхідності завантажується у пам'ять і виконується. Можна сказати, що під час виконання програму представляє процес.

Однозначна відповідність між програмою і процесом встановлюється тільки в конкретний момент часу: один процес у різний час може виконувати код декількох програм, код однієї програми можуть виконувати декілька процесів одночасно.

Для успішного виконання програми потрібні певні ресурси. До них належать:

- ресурси, необхідні для послідовного виконання програмного коду (передусім процесорний час);
- ресурси, що дають можливість зберігати інформацію, яка забезпечує виконання програмного коду (регістри процесора, оперативна пам'ять тощо).

Ці групи ресурсів визначають дві складові частини процесу:

- послідовність виконуваних команд процесора;
- набір адрес пам'яті (адресний простір), у якому розташовані ці команди і дані для них.

Виділення цих частин виправдане ще й тим, що в рамках одного адресного простору може бути кілька паралельно виконуваних послідовностей команд, що спільно використовують одні й ті ж самі дані. Необхідність розмежування послідовності команд і адресного простору підводить до поняття потоку.

Потоком (потік керування, нитка, thread) називають набір послідовно виконуваних команд процесора, які використовують загальний адресний простір процесу. Оскільки в системі може одночасно бути багато потоків, завданням ОС є організація перемикання процесора між ними і планування їхнього виконання. У багатопроцесорних системах код окремих потоків може виконуватися на окремих процесорах.

Тепер можна дати ще одне означення процесу.

Процесом називають сукупність одного або декількох потоків і захищеного адресного простору, у якому ці потоки виконуються.

Захищеність адресного простору процесу є його найважливішою характеристикою. Код і дані процесу не можуть бути прямо прочитані або перезаписані іншим процесом; у такий спосіб захищаються від багатьох програмних помилок і спроб несанкціонованого доступу. Природно, що неприпустимим є тільки прямий доступ (наприклад, запис у пам'ять за допомогою простої інструкції перенесення даних); обмін даними між процесами принципово можливий, але для цього мають бути використані спеціальні засоби, які називають засобами міжпроцесової взаємодії. Такі засоби складніші за прямий доступ і працюють повільніше, але при цьому забезпечують захист від випадкових помилок у разі доступу до даних.

На відміну від процесів потоки розпоряджаються загальною пам'яттю. Дані потоку не захищені від доступу до них інших потоків за умови, що всі вони виконуються в адресному просторі одного процесу. Це надає додаткові можливості для розробки застосувань, але ускладнює програмування.

Захищений адресний простір процесу задає абстракцію виконання коду на окремій машині, а потік забезпечує абстракцію послідовного виконання команд на одному виділеному процесорі.

Адресний простір процесу не завжди відповідає адресам оперативної пам'яті. Наприклад, у нього можуть відображатися файли або регістри контролерів введення-виведення, тому запис за певною адресою в цьому просторі призведе до запису у файл або до виконання операції введення-виведення. Таку технологію називають відображенням у пам'ять (memory mapping).

2.Максимально можлива кількість процесів (захищених адресних просторів) і потоків, які в них виконуються, може варіюватися в різних системах.

- В однозадачних системах є тільки один адресний простір, у якому в кожен момент часу може виконуватися один потік.
- У деяких вбудованих системах теж є один адресний простір (один процес), але в ньому дозволене виконання багатьох потоків. У цьому разі можна організовувати паралельні обчислення, але захист даних застосувань не реалізовано.
- У системах, подібних до традиційних версій UNIX, допускається наявність багатьох процесів, але в рамках адресного простору процесу виконується тільки один потік. Це традиційна однопотокова модель процесів. Поняття потоку в даній моделі не застосовують, а використовують терміни «перемикання між процесами», «планування виконання процесів», «послідовність команд процесу» тощо (тут під процесом розуміють його єдиний потік).
- У більшості сучасних ОС (таких, як лінія Windows XP, сучасні версії UNIX) може бути багато процесів, а в адресному просторі кожного процесу — багато потоків. Ці системи підтримують багатопотоковість або реалізують модель потоків. Процес у такій системі називають багатопотоковим процесом.

Надалі для позначення послідовності виконуваних команд вживатимемо термін «потік», за винятком ситуацій, коли обговорюватиметься реалізація моделі процесів.

3.До елементів процесу належать:

- захищений адресний простір;
- дані, спільні для всього процесу (ці дані можуть спільно використовувати всі його потоки);
- інформація про використання ресурсів (відкриті файли, мережні з'єднання тощо);
- інформація про потоки процесу. Потік містить такі елементи:

- стан процесора (набір поточних даних із його регістрів), зокрема лічильник поточної інструкції процесора;
- стек потоку (ділянка пам'яті, де перебувають локальні змінні потоку й адреси повернення функцій, що викликані у його коді).

4.Використання декількох потоків у застосуванні означає внесення в нього паралелізму (concurrency). Паралелізм — це одночасне (з погляду прикладного програміста) виконання дій різними фрагментами коду застосування. Така одночасність може бути реалізована на одному процесорі шляхом перемикання задач (випадок псевдопаралелізму), а може ґрунтуватися на паралельному виконанні коду на декількох процесорах (випадок справжнього паралелізму). Потоки абстрагують цю відмінність, даючи можливість розробляти застосування, які в однопроцесор-них системах використовують псевдопаралелізм, а при доданні процесорів — справжній паралелізм (такі застосування масштабуються зі збільшенням кількості процесорів).

Види паралелізму

Можна виділити такі основні види паралелізму:

- паралелізм багатопроцесорних систем;
- паралелізм операцій введення-виведення;
- паралелізм взаємодії з користувачем;
- паралелізм розподілених систем.

Перший з них є справжнім паралелізмом, тому що у багатопроцесорних системах інструкції виконують декілька процесорів одночасно. Інші види паралелізму можуть виникати і в однопроцесорних системах тоді, коли для продовження виконання програмного коду необхідні певні зовнішні дії.

Паралелізм операцій введення-виведення

Під час виконання операції введення-виведення (наприклад, обміну даними із диском) низькорівневий код доступу до диска і код застосування не можуть виконуватись одночасно. У цьому разі застосуванню треба

почекати завершення операції введення-виведення, звільнивши на цей час процесор. Природним вважається зайняти на цей час процесор інструкціями іншої задачі.

Багатопотокове застосування може реалізувати цей вид паралелізму через створення нових потоків, які виконуватимуться, коли поточний потік очікує операції введення-виведення. Так реалізується асинхронне введення-виведення, коли застосування продовжує своє виконання, не чекаючи на завершення операцій введення-виведення.

Паралелізм взаємодії з користувачем

Під час інтерактивного сеансу роботи користувач може виконувати різні дії із застосуванням (і очікувати негайної реакції на них) до завершення обробки попередніх дій. Наприклад, після запуску команди «друкування документа» він може негайно продовжити введення тексту, не чекаючи завершення друкування.

Щоб розв'язати це завдання, можна виділити окремі потоки для безпосередньої взаємодії із користувачем (наприклад, один потік може очікувати введення з клавіатури, інший — від миші, додаткові потоки — відображати інтерфейс користувача). Основні задачі застосування (розрахунки, взаємодія з базою даних тощо) у цей час виконуватимуть інші потоки.

Паралелізм розподілених застосувань

Розглянемо серверне застосування, яке очікує запити від клієнтів і виконує дії у відповідь на запит. Якщо клієнтів багато, запити можуть надходити часто, майже водночас. Якщо тривалість обробки запиту перевищує інтервал між запитами, сервер буде змушений поміщати запити в чергу, внаслідок чого знижується продуктивність. При цьому використання потоків дає можливість організувати паралельне обслуговування запитів, коли основний потік приймає запити, відразу передає їх для виконання іншим потокам і очікує нових.

5.За допомогою потоків вирішуються такі проблеми:

- потоки дають змогу реалізувати різні види паралелізму і дозволяють застосуванню масштабуватися із ростом кількості процесорів.
- для підтримки потоків потрібно менше ресурсів, ніж для підтримки процесів (наприклад, немає необхідності виділяти для потоків адресний простір).
- для обміну даними між потоками може бути використана спільна пам'ять (адресний простір їхнього процесу). Це ефективніше, ніж застосовувати засоби міжпроцесової взаємодії.

Незважаючи на перелічені переваги, використання потоків не є універсальним засобом розв'язання проблем паралелізму, і пов'язане з деякими труднощами:

- розробляти і налагоджувати багатопотокові програми складніше, ніж звичайні послідовні програми; досить часто впровадження багатопотоковості призводить до зниження надійності застосувань. Організація спільного використання адресного простору декількома потоками вимагає, щоб програміст мав високу кваліфікацію.
- використання потоків може спричинити зниження продуктивності застосувань. Переважно це трапляється в однопроцесорних системах (наприклад, у таких системах спроба виконати складний розрахунок паралельно декількома потоками призводить лише до зайвих витрат на перемикання між потоками, кількість виконаних корисних інструкцій залишиться тією ж самою).

Переваги і недоліки використання потоків потрібно враховувати під час виконання будь-якого програмного проекту. Насамперед доцільно розглядати можливість розв'язати задачу в рамках моделі процесів. При цьому, однак, варто брати до уваги не лише поточні вимоги замовника, але й необхідність розвитку застосування, його масштабування тощо. Можливо, з урахуванням цих факторів використання потоків буде виправдане.

Лекція №2. Тема: Способи реалізації моделі процесів і потоків та їх опис

1.Способи реалізації моделі потоків

2.Стани процесів і потоків

3.Опис процесів і потоків

4.Керуючі блоки процесів і потоків

5.Образи процесів і потоків

1.В ОС використовують два типи потоків – потоки користувача і потоки ядра

Потік користувача — це послідовність виконання команд в адресному просторі процесу. Ядро ОС не має інформації про такі потоки, вся робота з ними виконується в режимі користувача. Засоби підтримки потоків користувача надають спеціальні системні бібліотеки; вони доступні для прикладних програмістів у вигляді бібліотечних функцій. Бібліотеки підтримки потоків у сучасних ОС реалізують набір функцій, визначений стандартом POSIX (відповідний розділ стандарту називають POSIX.1b); тут прийнято говорити про підтримку потоків POSIX.

Потік ядра - це послідовність виконання команд в адресному просторі ядра. Потоками ядра управляє ОС, перемикання ними можливе тільки у привілейованому режимі. Є потоки ядра, які відповідають потокам користувача, і потоки, що не мають такої відповідності.

Співвідношення між двома видами потоків визначає реалізацію моделі потоків.

Схема багатопотоковості M:1 (є найранішою) реалізує багатопотоковість винятково в режимі користувача. При цьому кожен процес може містити багато потоків користувача, однак про наявність цих потоків ОС не відомо, вона працює тільки із процесами. За планування потоків і перемикання контексту відповідає бібліотека підтримки потоків. Схема вирізняється ефективністю керування потоками (для цього немає потреби

переходити в режим ядра) і не потребує для реалізації зміни ядра ОС. Проте нині її практично не використовують через два суттєвих недоліки, що не відповідають ідеології багатопотоковості.

- Схема M:1 не дає змоги скористатися багатопроцесорними архітектурами, оскільки визначити, який саме код виконуватиметься на кожному із процесорів, може тільки ядро ОС. У результаті всі потоки одного процесу завжди виконуватимуться на одному процесорі.
- Оскільки системні виклики обробляються на рівні ядра ОС, блокувальний системний виклик (наприклад, виклик, який очікує введення даних користувачем) зупинятиме всі потоки процесу, а не лише той, що зробив цей виклик.

Схема багатопотоковості 1:1 ставить у відповідність кожному потоку користувача один потік ядра. Планування і перемикання контексту виконується на рівні потоків ядра. у режимі користувача ці функції не реалізовані. Оскільки ядро ОС має інформацію про потоки, ця схема вільна від недоліків попередньої (різні потоки можуть виконуватися на різних процесорах, а при зупиненні одного потоку інші продовжують роботу). Вона проста і надійна в реалізації і сьогодні є найпоширенішою. Хоча схема передбачає, що під час керування потоками треба постійно перемикатися між режимами процесора, на практиці втрата продуктивності внаслідок цього виявляється незначною.

Схема багатопотоковості M:N. У цій схемі присутні як потоки ядра, так і потоки користувача, які відображаються на потоки ядра так, що один потік ядра може відповідати декільком потокам користувача. Число потоків ядра може бути змінене програмістом для досягнення максимальної продуктивності. Розподіл потоків користувача по потоках ядра виконується в режимі користувача, планування потоків ядра - у режимі ядра. Схема є складною в реалізації і сьогодні здає свої позиції схемі 1:1.

2. Для потоку дозволені такі стани:

- створення (new) — потік перебуває у процесі створення;
- виконання (running) — інструкції потоку виконує процесор (у конкретний момент часу на одному процесорі тільки один потік може бути в такому стані);
- очікування (waiting) — потік очікує деякої події (наприклад, завершення операції введення-виведення); такий стан називають також заблокованим, а потік - припиненим;
- готовність (ready) — потік очікує, що планувальник перемкне процесор на нього, при цьому він має всі необхідні йому ресурси, крім процесорного часу;
- завершення (terminated) — потік завершив виконання (якщо при цьому його ресурси не були вилучені з системи, він переходить у додатковий стан — стан зомбі).

Перехід потоків між станами очікування і готовності реалізовано на основі планування задач, або планування потоків. Під час планування потоків визначають, який з потоків треба відновити після завершення операції введення-виведення, як організувати очікування подій у системі.

Для здійснення переходу потоків між станами готовності та виконання необхідне планування процесорного часу. На основі алгоритмів такого планування визначають, який з готових потоків потрібно виконувати в конкретний момент, коли потрібно перервати виконання потоку, щоб перемкнутися на інший готовий потік тощо.

Відносно систем, які реалізують модель процесів, прийнято говорити про стани процесів, а не потоків, і про планування процесів; фактично стани процесу в цьому разі однозначно відповідають станам його єдиного потоку.

У багатопотокових системах також можна виділяти стани процесів. Наприклад, у багатопотоковості, реалізованій за схемою M:1, потоки змінюють свої стани в режимі користувача, а процеси — у режимі ядра.

3.Одним із основних завдань операційної системи є розподіл ресурсів між процесами і потоками. Такими ресурсами є насамперед процесорний час (його розподіляють між потоками під час планування), засоби введення-виведення й оперативна пам'ять (їх розподіляють між процесами).

Для керування розподілом ресурсів ОС повинна підтримувати структури даних, які містять інформацію, що описує процеси, потоки і ресурси. До таких структур даних належать:

- таблиці розподілу ресурсів: таблиці пам'яті, таблиці введення-виведення, таблиці файлів тощо;
- таблиці процесів і таблиці потоків, де міститься інформація про процеси і потоки, присутні у системі в конкретний момент.

4.Інформацію про процеси і потоки в системі зберігають у спеціальних структурах даних, які називають керуючими блоками процесів і керуючими блоками потоків. Ці структури дуже важливі для роботи ОС, оскільки на підставі їхньої інформації система здійснює керування процесами і потоками.

Керуючий блок потоку (Thread Control Block, TCB) відповідає активному потоку, тобто тому, який перебуває у стані готовності, очікування або виконання. Цей блок може містити таку інформацію:

- ідентифікаційні дані потоку (зазвичай його унікальний ідентифікатор);
- стан процесора потоку: користувацькі регістри процесора, лічильник інструкцій, покажчик на стек;
- інформацію для планування потоків.

Таблиця потоків — це зв'язний список або масив керуючих блоків потоку. Вона розташована в захищеній області пам'яті ОС.

Керуючий блок процесу (Process Control Block, PCB) відповідає процесу, що присутній у системі. Такий блок може містити:

- ідентифікаційні дані процесу (унікальний ідентифікатор, інформацію про інші процеси, пов'язані з даним);

- інформацію про потоки, які виконуються в адресному просторі процесу (наприклад, покажчики на їхні керуючі блоки);
- інформацію, на основі якої можна визначити права процесу на використання різних ресурсів (наприклад, ідентифікатор користувача, який створив процес);
- інформацію з розподілу адресного простору процесу;
- інформацію про ресурси введення-виведення та файли, які використовує процес.

Зазначимо, що для систем, у яких реалізована модель процесів, у керуючому блоці процесу зберігають не посилання на керуючі блоки його потоків, а інформацію, необхідну безпосередньо для його виконання (лічильник інструкцій, дані для планування тощо).

Таблицю процесів організовують аналогічно до таблиці потоків. Як елементи в ній зберігатимуться керуючі блоки процесів.

5.Сукупність інформації, що відображає процес у пам'яті, називають образом процесу (process image), а всю інформацію про потік - образом потоку (thread image). До образу процесу належать:

- керуючий блок процесу;
- програмний код користувача;
- дані користувача (глобальні дані програми, загальні для всіх потоків);
- інформація образів потоків процесу.

Програмний код користувача, дані користувача та інформація про потоки завантажуються в адресний простір процесу. Образ процесу звичайно не є безперервною ділянкою пам'яті, його частини можуть вивантажуватися на диск. Ці питання будуть розглянуті в розділі 9.

До образу потоку належать:

- керуючий блок потоку;

- стек ядра (стек потоку, який використовується під час виконання коду потоку в режимі ядра);
- стек користувача (стек потоку, доступний у користувацькому режимі).

Лекція №3. Тема: перемикання контексту й обробка переривань

1.Організація перемикання контексту

2.Обробка переривань

3.Створення процесів та їхня ієрархія

4.Особливості завершення процесів

5.Синхронне й асинхронне виконання процесів

6.Створення і завершення потоків

1.Найважливішим завданням операційної системи під час керування процесами і потоками є організація перемикання контексту - передачі керування від одного потоку до іншого зі збереженням стану процесора.

Загальних принципів перемикання контексту дотримуються у більшості систем, але їхня реалізація обумовлена конкретною архітектурою. Для перемикання контексту потрібно виконати такі операції:

- зберегти стан процесора потоку в деякій ділянці пам'яті (області зберігання стану процесора потоку);
- визначити, який потік слід виконувати наступним;
- завантажити стан процесора цього потоку із його області зберігання;
- продовжити виконання коду нового потоку.

Перемикання контексту здійснюється із залученням засобів апаратної підтримки. Можуть бути використані спеціальні регістри та ділянки пам'яті, які дають можливість зберігати інформацію про поточну задачу (коли розглядають апаратне забезпечення, аналогом поняття «потік» є поняття

«задача»), а також спеціальні інструкції процесора для роботи з цими регістрами та ділянками пам'яті.

В архітектурі IA-32 для збереження стану процесора кожної задачі (вмісту пов'язаних із нею регістрів процесора) використовують спеціальну ділянку пам'яті - сегмент стану задачі TSS. Адресу цієї області можна одержати з регістра задачі TR (це системний адресний регістр).

Для перемикання задач досить завантажити нові дані в регістр TR. У результаті значення регістрів процесора поточної задачі автоматично збережуться в її сегменті стану, після чого в регістри процесора буде завантажено стан процесора нової (або раніше перерваної) задачі й почнеться виконання її інструкцій.

Наступний потік для виконання вибирають відповідно до принципів планування потоків.

2.У процесі виконання потік може бути перерваний не лише для перемикання контексту на інший потік, але й у зв'язку із програмним або апаратним перериванням (перемикання контексту теж пов'язане із перериваннями, власне, із перериванням від таймера). Із кожним перериванням надходить додаткова інформація (наприклад, його номер). На підставі цієї інформації система визначає, де буде розміщена адреса процедури оброблювача переривання (список таких адрес зберігають у спеціальній ділянці пам'яті і називають вектором переривань).

Наведемо приклад послідовності дій під час обробки переривання:

- збереження стану процесора потоку;
- встановлення стека оброблювача переривання;
- початок виконання оброблювача переривання (коду операційної системи); для цього з вектора переривання завантажується нове значення лічильника команд;
- відновлення стану процесора потоку після закінчення виконання оброблювача і продовження виконання потоку.

Передача керування оброблювачеві переривання, як і перемикання контексту, може відбутися практично у будь-який момент. Основна відмінність полягає в тому, що адресу, на яку передається керування, задають на основі номера переривання і зберігають у векторі переривань, а також у тому, що код оброблювача не продовжується з місця, де було перерване виконання, а починає виконуватися щораз заново.

3.Засоби створення і завершення процесів дають змогу динамічно змінювати в операційній системі набір застосувань, що виконуються. Засоби створення і завершення потоків є основою для створення багатопотокових програм.

Процеси можуть створюватися ядром системи під час її ініціалізації. Наприклад, в UNIX-сумісних системах таким процесом може бути процес ініціалізації системи init, у Windows XP - процеси підсистем середовища (Win32 або POSIX). Таке створення процесів, однак, є винятком, а не правилом.

Найчастіше процеси створюються під час виконання інших процесів. У цьому разі процес, який створює інший процес, називають предком, а створений ним процес - нащадком.

Нові процеси можуть бути створені під час роботи застосування відповідно до його логіки (компілятор може створювати процеси для кожного етапу компіляції, веб-сервер - для обробки прибулих запитів) або безпосередньо за запитом користувача (наприклад, з командного інтерпретатора, графічної оболонки або файлового менеджера).

Розрізняють два типи процесів з погляду їхньої взаємодії із користувачем - інтерактивні та фонові.

- Інтерактивні процеси взаємодіють із користувачами безпосередньо, приймаючи від них дані, введені за допомогою клавіатури, миші тощо. Прикладом інтерактивного процесу може бути процес текстового редактора або інтегрованого середовища розробки.

- Фонові процеси із користувачем не взаємодіють безпосередньо. Зазвичай вони запускаються під час старту системи і чекають на запити від інших застосувань. Деякі з них (системні процеси) підтримують функціонування системи (реалізують фонове друкування, мережні засоби тощо), інші виконують спеціалізовані задачі (реалізують веб-сервери, сервери баз даних тощо). Фонові процеси також називають службами (services, у системах лінії Windows XP) або демонами (daemons, в UNIX).

Ієрархія процесів

Після того як процес-предок створив процес-нащадок, потрібно забезпечити їхній взаємозв'язок. Є різні варіанти розв'язання цього завдання.

Можна організувати на рівні ОС однозначний зв'язок «предок-нащадок» так, щоб для кожного процесу завжди можна було визначити його предка. Наприклад, якщо процеси визначені унікальними ідентифікаторами, то для реалізації цього підходу в керуючому блоці процесу-нащадка повинен завжди зберігатися ідентифікатор процесу-предка або посилання на його керуючий блок.

Таким чином формується ієрархія (дерево) процесів, оскільки нащадки можуть у свою чергу створювати нових нащадків і т. ін. У таких системах існує спеціальний вихідний процес (в UNIX-системах його називають init), з якого починається побудова дерева процесів (його запускає ядро системи). Якщо предок завершить виконання свого процесу перед своїм нащадком, функції предка бере на себе вихідний процес.

З іншого боку, зв'язок «предок-нащадок» можна не реалізовувати на рівні ОС. При цьому всі процеси виявляються рівноправними. Якщо зв'язок «предок-нащадок» для конкретної пари процесів все ж таки потрібен, за його підтримку відповідають самі процеси (процес-предок, наприклад, може сам зберегти свій ідентифікатор у структурі даних нащадка у разі його створення).

Взаємозв'язок між процесами не обмежується лише відношеннями «предок-нащадок». Наприклад, у деяких ОС є поняття сесії (session). Така

сесія поєднує всі процеси, створені користувачем за час інтерактивного сеансу його роботи із системою.

4.Існує три варіанти завершення процесів.

Процес коректно завершується самостійно після виконання своєї задачі (для інтерактивних процесів це нерідко відбувається з ініціативи користувача, який, приміром, скористався відповідним пунктом меню). Для цього код процесу має виконати системний виклик завершення процесу. Такий виклик у POSIX-систе-мах називають exit (). Він може повернути процесу, що його викликав, або ОС код повернення, який дає змогу оцінити результат виконання процесу.

Процес аварійно завершується через помилку. Такий вихід може бути передбачений програмістом (під час обробки помилки приймається рішення про те, що далі продовжувати виконання програми неможливо), а може бути наслідком генерування переривання (ділення на нуль, доступу до захищеної області пам'яті тощо).

Процес завершується іншим процесом або ядром системи. Наприклад, перед закінченням роботи ОС ядро припиняє виконання всіх процесів. Процес може припинити виконання іншого процесу за допомогою системного виклику, який у POSIX-системах називають kill ().

Після того як процес завершить свою роботу, пам'ять, відведена під його адресний простір, звільняється і може бути використана для інших потреб. Усі потоки цього процесу теж припиняють роботу. Якщо у даного процесу є нащадки, їхня робота переважно не припиняється слідом за роботою предка. Інтерактивні процеси завершуються у разі виходу користувача із системи.

5.Коли у системі з'являється новий процес, для старого процесу є два основних варіанти дій:

- продовжити виконання паралельно з новим процесом - такий режим роботи називають асинхронним виконанням;
- призупинити виконання доти, поки новий процес не буде завершений, — такий режим роботи називають синхронним виконанням. (У цьому разі використовують спеціальний системний виклик, який у POSIX-системах називають wait().)

Вибір того чи іншого режиму залежить від конкретної задачі. Так, наприклад, веб-сервер може створювати процеси-нащадки для обробки запитів (якщо наявного набору нащадків недостатньо для такої обробки). У цьому разі обробка має бути асинхронною, бо відразу після створення нащадка предок має бути готовий до отримання наступного запиту. З іншого боку, компілятор C під час запуску процесів, що відповідають етапам компіляції, має чекати завершення кожного етапу, перш ніж перейти до наступного, - у такому разі використовують синхронну обробку.

6.Особливості створення потоків

Процедура створення потоків має свої особливості, пов'язані насамперед із тим, що потоки створюють у рамках вже існуючого адресного простору (конкретного процесу або, можливо, ядра системи). Існує кілька ситуацій, у яких може бути створений новий потік.

Якщо процес створюється за допомогою системного виклику fork(), після розподілу адресного простору автоматично створюється потік усередині цього процесу (найчастіше це — головний потік застосування).

Можна створювати потоки з коду користувача за допомогою відповідного системного виклику.

У багатьох ОС є спеціальні потоки, які створює ядро системи (код ядра теж може виконуватися в потоках).

Під час створення потоку система повинна виконати такі дії.

1.Створити структури даних, які відображають потік в ОС.

2.Виділити пам'ять під стек потоку.

3.Встановити лічильник команд процесора на початок коду, який буде виконуватися в потоці; цей код називають процедурою або функцією потоку, покажчик на таку процедуру передають у системний виклик створення потоку.

Слід відзначити, що під час створення потоків, на відміну від створення процесів немає необхідності виділяти нову пам'ять під адресний простір, тому зазвичай потоки створюються швидше і з меншими затратами ресурсів.

Локальні змінні функції потоку розташовані у стеку потоку і доступні лише його коду, глобальні змінні доступні всім потокам.

Особливості завершення потоків

Під час завершення потоку його ресурси вивільняються (насамперед, стек); ця операція звичайно виконується швидше, ніж завершення процесу. Потік може бути завершений, коли керування дійде до кінця процедури потоку; є також спеціальні системні виклики, призначені для дострокового припинення виконання потоків.

Як і процеси, потоки можуть виконуватися синхронно й асинхронно. Потік, створивши інший потік, може призупинити своє виконання до його завершення. Таке очікування називають приєднанням потоків (thread joining, очікує той, хто приєднує). Після завершення приєднаного потоку потік, який очікував його завершення, може дістати статус виконання. Під час створення потоку можна визначити, чи підлягає він приєднанню (якщо потік не можна приєднати, його називають від'єднаним — detached). Якщо потік не є від'єднаним (nondetached або joinable, такий потік називатимемо приєднуваним), після завершення його обов'язково потрібно приєднувати, щоб уникнути витікання пам'яті.

Рекомендації щодо розробки багатопотокових програм:

- Не можна визначати порядок виконання потоків, не подбавши про його підтримку, тому що швидкість виконання потоків залежить від багатьох факторів, більшість із яких програміст не контролює. Наприклад,

якщо запустити набір потоків усередині функції f () і не приєднати всі ці потоки до завершення f (),деякі з них можуть не встигнути завершити своє виконання до моменту виходу з f (). Можливо навіть, що частина потоків не завершиться й до моменту наступного виклику функції, якщо програміст виконає його достатньо швидко.

- Для потоків не підтримується така ієрархія, як для процесів. Потік, що створив інший потік, має рівні з ним права. Розраховувати на те, що такий потік сам буде продовжувати своє виконання після завершення потоків-нащадків, немає сенсу.
- Стек потоку очищається після виходу із функції потоку. Щоб цьому запобігти, не слід, наприклад, передавати нікуди покажчики на локальні змінні такої функції. Якщо необхідні змінні, значення яких мають зберігатися між викликами функції потоку, але при цьому вони не будуть доступні іншим потокам, треба скористатися локальною пам'яттю потоку (Thread Local Storage, TLS) - певним чином організованими даними, доступ до яких здійснюється за допомогою спеціальних функцій.

Розділ4. Міжпроцесова взаємодія

Лекція №1. Тема: види міжпроцесової взаємодії

1.Проблеми міжпроцесової взаємодії

2.Види міжпроцесової взаємодії

a) методи роз поділюваної пам’яті

b) методи передавання повідомлень

c) технологія відображуваної пам’яті

3.Міжпроцесова взаємодія на базі спільної пам’яті

1.Головною особливістю взаємодії потоків одного процесу є простота технічної реалізації обміну даними між ними — усі потоки одного процесу

використовують один адресний простір, а отже, можуть вільно отримувати доступ до спільно використовуваних даних, ніби вони є їх власними. Оскільки технічних труднощів із реалізацією обміну даними тут немає, основною проблемою, яку потрібно вирішувати в цьому випадку, є синхронізація потоків.

Кожен потік виконується в рамках адресного простору свого процесу, тому часто постає задача організації взаємодії між потоками різних процесів. Ідеться власне про міжпроцесову взаємодію (interprocess communication, IPC) [37]. Ця технологія з'явилася задовго до поширення багатопотоковості.

Для потоків різних процесів питання забезпечення синхронізації теж є актуальними, але вони в більшості випадків не ґрунтуються на понятті спільно використовуваних даних (такі дані за замовчуванням для процесів відсутні). Крім того, додається досить складна задача забезпечення обміну даними між захищеними адресними просторами. Підходи до її розв'язання визначають різні види міжпроцесової взаємодії.

2.Реалізація міжпроцесової взаємодії здійснюється трьма основними методами: передавання повідомлень, розподілюваної пам'яті та відображуваної пам'яті. Ще одним методом IPC також можна вважати технологію сигналів.Їхнє використання не зводиться тільки до організації IPC (синхронні сигнали є засобом оповіщення процесу про виняткову ситуацію); без них складно пояснити ряд базових понять керування процесами (наприклад, очікування завершення процесу).

Методи розподілюваної пам'яті

Методи розподілюваної пам'яті (shared memory) дають змогу двом процесам обмінюватися даними через загальний буфер пам'яті. Перед обміном даними кожний із тих процесів має приєднати цей буфер до свого адресного простору з використанням спеціальних системних викликів (перед цим перевіряють права). Буфер доступний у системі за допомогою процедури іменування, термін його існування звичайно обмежений часом роботи всієї системи. Дані в ньому фактично є спільно використовуваними, як і для

потоків. Жодних засобів синхронізації доступу до цих даних розподілювана пам'ять не забезпечує, програміст, так само, як і при розробці багатопотокових застосувань, має організовувати її сам.

Методи передавання повідомлень

В основі методів передавання повідомлень (message passing) лежать різні технології, що дають змогу потокам різних процесів (які, можливо, виконуються на різних комп'ютерах) обмінюватися інформацією у вигляді фрагментів даних фіксованої чи змінної довжини, котрі називають повідомленнями (messages). Процеси можуть приймати і відсилати повідомлення, при цьому автоматично забезпечується їхнє пересилання між адресними просторами процесів одного комп'ютера або через мережу. Важливою особливістю технологій передавання повідомлень є те, що вони не спираються на спільно використовувані дані - процеси можуть обмінюватися повідомленнями, навіть не знаючи один про одного.

Технологія відображуваної пам'яті

Ще однією категорією засобів міжпроцесової взаємодії є відображувана пам'ять (mapped memory). У ряді ОС відображувана пам'ять є базовим системним механізмом, на якому ґрунтуються інші види міжпроцесової взаємодії та системні вирішення. Звичайно відображувану пам'ять використовують у поєднанні з інтерфейсом файлової системи, в такому разі говорять про файли, відображувані у пам'ять (memory-mapped files).

Ця технологія зводиться до того, що за допомогою спеціального системного виклику (зазвичай це mmapO) певну частину адресного простору процесу однозначно пов'язують із вмістом файла. Після цього будь-яка операція записування в таку пам'ять спричиняє зміну вмісту відображеного файла, яка відразу стає доступною усім застосуванням, що мають доступ до цього файла. Інші застосування теж можуть відобразити той самий файл у свій адресний простір і обмінюватися через нього даними один з одним.

Файли, відображувані у пам'ять, будуть докладно розглянуті в розділі 11.

Особливості міжпроцесової взаємодії

Тепер можна порівняти характеристики міжпроцесової взаємодії із характеристиками взаємодії потоків одного процесу.

- Проблема організації обміну даними є актуальною тільки для міжпроцесової взаємодії, оскільки потоки обмінюються даними через загальний адрес простір. Обмін даними між потоками схожий на використання розподілюваної пам'яті, але не потребує підготовчих дій.
- Проблема синхронізації доступу до спільно використовуваних даних є актуальною для взаємодії потоків і для міжпроцесової взаємодії із використанням розподілюваної пам'яті. Використання механізму передавання повідомлень не ґрунтується на спільно використовуваних даних і не потребує синхронізації

3.Для вирішення проблеми міжпроцесової синхронізації необхідно:

- по-перше, організувати спільну пам'ять між процесами (це може бути розподілювана пам'ять або файл, відображений у пам'ять);
- по-друге, розмістити в цій пам'яті стандартні синхронізаційні об'єкти (семафори, м'ютекси, умовні змінні);
- по-третє, використовуючи ці об'єкти, працювати зі спільно використовуваними даними, як це робилося у разі використання потоків.

Такий підхід широко застосовують на практиці. На жаль, досить складно запропонувати спосіб його реалізації для міжпроцесової синхронізації у більшості систем, оскільки різні системи пропонують різний набір засобів організації спільної пам'яті та засобів сигналізації, які можуть працювати в такій пам'яті. Універсальним рішенням у даному разі є застосування семафорів.

Лекція №2. Тема: механізм передачі повідомлень

1.Основи передавання повідомлень

2.Примітиви передавання повідомлень

3.Синхронне й асинхронне предавання повідомлень

1.Усі методи взаємодії, які було розглянуто дотепер, ґрунтуються на читанні й записуванні спільно використовуваних даних. На практиці така взаємодія не завжди можлива (наприклад, робота зі спільно використовуваними даними проблематична, якщо для процесів немає спільної фізичної пам'яті, а є тільки мережний зв'язок між комп'ютерами, на яких вони виконуються). У таких випадках можна використати засоби взаємодії, які не ґрунтуються на спільно використовуваних даних, передусім засоби передавання повідомлень .

Як було вже згадано, засоби передавання повідомлень ґрунтуються на обміні повідомленнями - фрагментами даних змінної довжини. Основою такого обміну є не спільна пам'ять, а канал зв'язку (communication channel). Він забезпечує взаємодію між процесами (для того, щоб спілкуватися, вони повинні створити канал зв'язку) і є абстрактним відображенням мережі зв'язку. Абстрактність каналу дає змогу реалізувати його не тільки на основі мережної взаємодії, але й спільної пам'яті (коли процеси перебувають на одному комп'ютері). При цьому такі зміни в реалізації будуть сховані від процесів, що взаємодіють.

Виокремлюють такі характеристики каналів зв'язку: спосіб задання; кількість процесів, які можуть бути з'єднані одним каналом; кількість каналів, які можуть бути створені між двома процесами; пропускна здатність каналу (кількість повідомлень, які можуть одночасно перебувати в системі й бути асоційованими з цим каналом); максимальний розмір повідомлення; спрямованість зв'язку через канал (двобічний або однобічний зв'язок).

В однобічному зв'язку для конкретного процесу допускають передавання даних тільки в один бік.

2.Основна особливість передавання повідомлень полягає в тому, що процеси спільно використовують тільки канали. Немає необхідності забезпечувати взаємне виключення процесів під час доступу до спільно використовуваних даних, замість цього досить визначити примітиви передавання повідомлень — спеціальні операції обміну даними через канал, які забезпечують не лише обмін даними, але й синхронізацію.

Є два примітиви передавання повідомлень: send (для відсилання повідомлення каналом) і гесеі ve (для отримання повідомлення з каналу).

Розглянемо, як особливості реалізації send і гесеі ve дають змогу виділити різні класи методів передавання повідомлень.

Зазначені примітиви передавання повідомлень можуть задавати прямий і непрямий обмін даними. При прямому обміні даними необхідно явно вказувати процес, з яким необхідно обмінюватись інформацією. Непрямий обмін здійснюють через спеціальний об'єкт (поштову скриньку, порт); процеси можуть поміщати повідомлення в поштову скриньку і отримувати їх звідти. Зазвичай кілька процесів мають доступ до однієї поштової скриньки, застосовуючи під час її пошуку методи іменування. Більшість сучасних технологій обміну повідомленнями використовує непрямий обмін даними. Прикладом прямого обміну є традиційні сигнали.

3.Зупинимося на основних питаннях синхронізації під час передавання повідомлень. Можна виокремити різні групи методів передавання повідомлень залежно від того, як вони дають можливість відповісти на два запитання.

1.Чи може потік бути призупинений під час виконання операції send, якщо повідомлення не було отримане?

2.Чи може потік бути призупинений під час виконання операції receive, якщо повідомлення не було відіслане?

У реальних системах відповідь на друге запитання практично завжди буде позитивною неблокувальне приймання повідомлень спричиняється до того, що вони губляться. Варіанти відповідей на перше запитання визначають два класи передавання повідомлень - синхронне і асинхронне.

Під час синхронного передавання повідомлень операція send призупиняє процес до отримання повідомлення, а під час асинхронного передавання повідомлень вона не призупиняє процес (тобто є неблоісувальною); після відсилання повідомлення процес продовжує своє виконання, не чекаючи отримання результату. Найзручніше в цьому випадку використати непряму адресацію через поштові скриньки.

Реалізація синхронного й асинхронного передавання повідомлень залежить від низки характеристик каналу й обміну повідомленнями, насамперед від пропускної здатності каналу.

- Якщо пропускна здатність дорівнює нулю (повідомлення не можуть очікувати в системі), відправник завжди має очікувати, поки одержувачу не надійде повідомлення, а одержувач має очікувати, поки повідомлення йому не буде відіслано. Два процеси мають явно домовлятися про майбутній обмін.
- Якщо пропускна здатність обмежена (у системі можуть перебувати максимум n повідомлень для цього каналу), відправник має очікувати тільки тоді, коли черга повідомлень для цього каналу переповнена (у ній перебуває рівно n повідомлень), одержувач має очікувати, якщо ця черга порожня.
- Якщо пропускна здатність необмежена, очікування можливе тільки для одержувача за порожньої черги.

Під час обміну повідомленнями необхідне підтвердження їх отримання. Деякі методи обміну повідомленнями не вимагають підтвердження зовсім, в інших випадках можлива ситуація, коли відправника після виконання операції send блокують доти, поки одержувач не надішле

йому інше повідомлення із підтвердженням отримання; таку технологію називають обміном повідомленнями із підтвердженням отримання.

Лекція №3. Тема: технології передавання повідомлень

1.Методи передавання повідомлень за допомогою каналів і черг

2.Методи обміну повідомленнями за допомогою сокетів

3.Віддалений виклик процедур

1.Канал — це найпростіший засіб передавання повідомлень. Він є циклічним буфером, записування у який виконують за допомогою одного процесу, а читання — за допомогою іншого. У конкретний момент часу до каналу має доступ тільки один процес. Операційна система забезпечує синхронізацію згідно правилу: якщо процес намагається записувати в канал, у якому немає місця, або намагається зчитати більше даних, ніж поміщено в канал, він переходить у стан очікування.

Розрізняють безіменні та поіменовані канали.

До безіменних каналів немає доступу за допомогою засобів іменування, тому процес не може відкрити вже наявний безіменний канал без його дескриптора. Це означає, що такий процес має отримати дескриптор каналу від процесу, що його, створив, а це можливо тільки для зв'язаних процесів.

До поіменованих каналів (named pipes) є доступ за іменем. Такому каналу може відповідати, наприклад, файл у файловій системі, при цьому будь-який процес, який має доступ до цього файла, може обмінюватися даними через відповідний канал. Поіменовані канали реалізують непрямий обмін даними.

Обмін даними через канал може бути однобічним і двобічним.

Черги повідомлень

Іншою технологією асинхронного непрямого обміну даними є застосування черг повідомлень (message queues). Для таких черг виділяють

спеціальне місце в системній ділянці пам'яті ОС, доступне для застосувань користувача. Процеси можуть створювати нові черги, відсилати повідомлення в конкретну чергу й отримувати їх звідти. Із чергою одночасно може працювати кілька процесів. Повідомлення — це структури даних змінної довжини. Для того щоб процеси могли розрізняти адресовані їм повідомлення, кожному з них присвоюють тип. Відіслане повідомлення залишається в черзі доти, поки не буде зчитане. Синхронізація під час роботи з чергами схожа на синхронізацію для каналів.

2.Найрозповсюдженішим методом обміну повідомленнями є використання сокетів (sockets). Ця технологія насамперед призначена для організації мережного обміну даними, але може бути використана й для взаємодії між процесами на одному комп'ютері (власне, мережну взаємодію можна розуміти як узагальнення IPC).

Сокет — це абстрактна кінцева точка з'єднання, через яку процес може відсилати або отримувати повідомлення. Обмін даними між двома процесами здійснюють через пару сокетів, по одному на кожен процес. Абстрактність сокету полягає в тому, що він приховує особливості реалізації передавання повідомлень — після того як сокет створений, робота з ним не залежить від технології передавання даних, тому один і той самий код можна без великих змін використовувати для роботи із різними протоколами зв'язку.

Особливості протоколу передавання даних і формування адреси сокету визначає комунікаційний домен; його потрібно зазначати під час створення кожного сокету. Прикладами доменів можуть бути домен Інтернету (який задає протокол зв'язку на базі TCP/IP) і локальний домен або домен UNIX, що реалізує зв'язок із використанням імені файла (подібно до поіменованого каналу). Сокет можна використовувати у поєднанні тільки з одним комунікаційним доменом. Адреса сокету залежить від домену (наприклад, для сокетів домену UNIX такою адресою буде ім'я файла).

Способи передавання даних через сокет визначаються його типом. У конкретному домені можуть підтримуватися або не підтримуватися різні типи сокетів.

Наприклад, і для домену Інтернет, і для домену UNIX підтримуються сокети таких типів:

- потокові (stream sockets) — задають надійний двобічний обмін даними суцільним потоком без виділення меж (операція читання даних повертає стільки даних, скільки запитано або скільки було на цей момент передано);
- дейтаграмні (datagram sockets) - задають ненадійний двобічний обмін повідомленнями із виділенням меж (операція читання даних повертає розмір того повідомлення, яке було відіслано).

Під час обміну даними із використанням сокетів зазвичай застосовується технологія клієнт-сервер, коли один процес (сервер) очікує з'єднання, а інший (клієнт) з'єднують із ним.

Перед тим як почати працювати з сокетами, будь-який процес (і клієнт, і сервер) має створити сокет за допомогою системного виклику socket( ). Параметрами цього виклику задають комунікаційний домен і тип сокету. Цей виклик повертає дескриптор сокету - унікальне значення, за яким можна буде звертатися до цього сокету.

Подальші дії відрізняються для сервера і клієнта. Для сервера виконуються такі дії:

1.Сокет пов'язують з адресою за допомогою системного виклику bind. Для сокетів домену UNIX як адресу задають ім'я файла, для сокетів домену Інтернету - необхідні характеристики мережного з'єднання. Далі клієнт для встановлення з'єднання й обміну повідомленнями має буде вказати цю адресу.

2.Сервер дає змогу клієнтам встановлювати з'єднання, виконавши системний виклик listen( ) для дескриптора сокету, створеного раніше.

3.Після виходу із системного виклику listen() сервер готовий приймати від клієнтів запити на з'єднання. Ці запити вишиковуються в чергу. Для отримання запиту із цієї черги і створення з'єднання використовують системний виклик accept( ). Внаслідок його виконання в застосування повертають новий сокет для обміну даними із клієнтом. Старий сокет можна використовувати далі для приймання нових запитів на з'єднання. Якщо під час виклику accept( ) запити на з'єднання в черзі відсутні, сервер переходить у стан очікування.

Для клієнта послідовність дій після створення сокету зовсім інша. Замість трьох кроків досить виконати один - встановити з'єднання із використанням системного виклику connect( ). Параметрами цього виклику задають дескриптор створеного раніше сокету, а також адресу, подібну до вказаної на сервері для виклику bind().

Після встановлення з'єднання (і на клієнті, і на сервері) з'явиться можливість передавати і приймати дані з використанням цього з'єднання. Для передавання даних застосовують системний виклик send( ), а для приймання - recv( ).

Зазначену послідовність кроків використовують для встановлення надійного з'єднання. Якщо все, що нам потрібно, - це відіслати і прийняти конкретне повідомлення фіксованої довжини, то з'єднання можна й не створювати зовсім. Для цього як відправник, так і одержувач повідомлення мають попередньо зв'язати сокети з адресами через виклик bind(). Потім можна скористатися викликами прямого передавання даних: sendto( ) - для відправника і recvfrom( ) - для одержувача. Параметрами цих викликів задають адреси одержувача і відправника, а також адреси буферів для даних.

3.Технологія віддаленого виклику процедур (Remote Procedure Call, RPC) є прикладом синхронного обміну повідомленнями із підтвердженням отримання. Розглянемо послідовність кроків, необхідних для обміну даними в цьому разі.

1.Операцію send оформляють як виклик процедури із параметрами.

2.Після виклику такої процедури відправник переходить у стан очікування, а дані (ім'я процедури і параметри) доставляються одержувачеві. Одержувач може перебувати на тому самому комп'ютері, чи на віддаленій машині; технологія RPC приховує це. Класичний віддалений виклик процедур передбачає, що процес-одержувач створено внаслідок запиту.

3.Одержувач виконує операцію гесеі ve і на підставі даних, що надійшли, виконує відповідні дії (викликає локальну процедуру за іменем, передає їй параметри і обчислює результат).

4.Обчислений результат повертають відправникові як окреме повідомлення.

5.Після отримання цього повідомлення відправник продовжує своє виконання, розглядаючи обчислений результат як наслідок виклику процедури.

Розділ 5. Керування оперативною пам’яттю

Лекція №1. Тема: Основні поняття та вимоги до керування оперативною пам`яттю

1.Загальні положення керування оперативною пам’яттю

2.Спільне використання фізичної пам’яті процесами та передумови введення віртуальної пам’яті

3.Поняття віртуальної пам’яті

4.Проблема фрагментації пам’яті

5.Логічна і фізична адресація пам’яті

6.Спільне використання пам’яті за допомогою базового та межового регістрів

1.Під пам'яттю розуміють ресурс комп'ютера, призначений для зберігання програмного коду і даних. Пам'ять зображають як масив машинних слів або байтів з їхніми адресами. У фон-нейманівській архітектурі комп'ютерних систем процесор вибирає інструкції і дані з пам'яті та може зберігати в ній результати виконання операцій.

Різні види пам'яті організовані в ієрархію. На нижніх рівнях такої ієрархії перебуває дешевша і повільніша пам'ять більшого обсягу, а в міру просування ієрархією нагору пам'ять стає дорожчою і швидшою (а її обсяг стає меншим). Найдешевшим і найповільнішим запам'ятовувальним пристроєм є жорсткий диск комп'ютера. Його називають також допоміжним запам'ятовувальним пристроєм (secondary storage). Швидшою й дорожчою є оперативна пам'ять, що зберігається в мікросхемах пам'яті, встановлених на комп'ютері, - таку пам'ять називають основною пам'яттю (main memory). Ще швидшими засобами зберігання даних є різні кеші процесора, а обсяг цих кешів ще обмеженіший.

Керування пам'яттю в ОС - досить складне завдання. Потрібної за характеристиками пам'яті часто виявляють недостатньо, і щоб це не заважало роботі користувача, необхідно реалізовувати засоби координації різних видів пам'яті. Так, сучасні додатки можуть не вміщатися цілком в основній пам'яті, тоді невикористовуваний код застосування може тимчасово зберігатися на жорсткому диску.

2.Найпростіший з можливих способів спільного використання фізичної пам'яті кількома процесами це неперервний і послідовний її розподіл між усіма виконуваними програмами.

За цієї ситуації кожний процес завантажують у свою власну неперервну ділянку фізичної пам'яті, ділянка наступного процесу починається відразу після ділянки попереднього. На рис. 8.1 праворуч позначені адреси фізичної пам'яті, починаючи з яких завантажуються процеси.

Якщо проаналізувати особливості розподілу пам'яті на основі цього підходу, можуть виникнути такі запитання.

- Як виконувати процеси, котрим потрібно більше фізичної пам'яті, ніж встановлено на комп'ютері?
- Що відбудеться, коли процес виконає операцію записування за невірною адресою (наприклад, процес P2 - за адресою 0x7500)?
- Що робити, коли процесу (наприклад, процесу P1) буде потрібна додаткова пам'ять під час його виконання?
- Коли процес отримає інформацію про конкретну адресу фізичної пам'яті, що з неї розпочнеться його виконання, і як мають бути перетворені адреси пам'яті, використані в його коді?
- Що робити, коли процесу не потрібна вся пам'ять, виділена для нього?

Пряме завантаження процесів у фізичну пам'ять не дає змоги дати відповіді на ці запитання. Очевидно, що потрібні деякі засоби трансляції пам'яті, які давали б змогу процесам використовувати набори адрес, котрі відрізняються від адрес фізичної пам'яті. Перш ніж розібратися в особливостях цих адрес, коротко зупинимося на особливостях компонування і завантаження програм.

Програма зазвичай перебуває на диску у вигляді двійкового виконуваного файла, отриманого після компіляції та компонування. Для свого виконання вона має бути завантажена у пам'ять (адресний простір процесу). Сучасні архітектури дають змогу процесам розташовуватися у будь-якому місці фізичної пам'яті, при цьому одна й та сама програма може відповідати різним процесам, завантаженим у різні ділянки пам'яті. Заздалегідь невідомо, в яку ділянку пам'яті буде завантажена програма.

Під час виконання процес звертається до різних адрес, зокрема в разі виклику функції використовують її адресу (це адреса коду), а звертання до глобальної змінної відбувається за адресою пам'яті, призначеною для зберігання значення цієї змінної (це адреса даних).

Програміст у своїй програмі звичайно не використовує адреси пам'яті безпосередньо, замість них вживаються символічні імена (функцій, глобальних змінних тощо). Внаслідок компіляції та компонування ці імена прив'язують до переміщуваних адрес (такі адреси задають у відносних одиницях, наприклад «100 байт від початку модуля»). Під час виконання програми переміщувані адреси, своєю чергою, прив'язують до абсолютних адрес у пам'яті. По суті, кожна прив'язка — це відображення одного набору адрес на інший.

До адрес, використовуваних у програмах, ставляться такі вимоги.

♦ Захист пам'яті. Помилки в адресації, що трапляються в коді процесу, повинні впливати тільки на виконання цього процесу. Коли процес P2 зробить операцію записування за адресою 0x7500, то він і має бути перерваний за помилкою. Стратегія захисту пам'яті зводиться до того, що для кожного процесу виділяється діапазон коректних адрес, і кожна операція доступу до пам'яті перевіряється на приналежність адреси цьому діапазону.

♦ Відсутність прив'язання до адрес фізичної пам'яті. Процес має можливості виконуватися незалежно від його місця в пам'яті та від розміру фізичної пам'яті. Адресний простір процесу виділяється як великий статичний набір адрес, при цьому кожна адреса такого набору є переміщуваною. Процесор і апаратне забезпечення повинні мати змогу перетворювати такі адреси у фізичні адреси основної пам'яті (при цьому та сама переміщувана адреса в різний час або для різних процесів може відповідати різним фізичним адресам).

3.Віртуальна пам'ять — це технологія, в якій вводиться рівень додаткових перетворень між адресами пам'яті, використовуваних процесом, і адресами фізичної пам'яті комп'ютера. Такі перетворення мають забезпечувати захист пам'яті та відсутність прив'язання процесу до адрес фізичної пам'яті.

Завдяки віртуальній пам'яті фізична пам'ять адресного простору процесу може бути фрагментованою, оскільки основний обсяг пам'яті, яку займає процес, більшу частину часу залишається вільним. Є так зване правило «дев'яносто до десяти», або правило локалізації, яке стверджує, що 90 % звертань до пам'яті у процесі припадає на 10 % його адресного простору. Адреси можна переміщати так, щоб основній пам'яті відповідали тільки ті розділи адресного простору процесу, які справді використовуються у конкретний момент.

При цьому невикористовувані розділи адресного простору можна ставити у відповідність повільнішій пам'яті, наприклад простору на жорсткому диску, а в цей час інші процеси можуть використовувати основну пам'ять, у яку раніше відображалися адреси цих розділів. Коли ж розділ знадобиться, його дані завантажують з диска в основну пам'ять, можливо, замість розділів, які стали непотрібними в конкретний момент (і які, своєю чергою, тепер збережуться на диску). Дані можуть зчитуватися з диска в основну пам'ять під час звертання до них.

У такий спосіб можна значно збільшити розмір адресного простору процесу і забезпечити виконання процесів, що за розміром перевищують основну пам'ять.

4.Основна проблема, що виникає у разі використання віртуальної пам'яті, стосується ефективності її реалізації. Оскільки перетворення адрес необхідно робити під час кожного звертання до пам'яті, недбала реалізація цього перетворення може призвести до найгірших наслідків для продуктивності всієї системи. Якщо для більшості звертань до пам'яті система буде змушена насправді звертатися до диска (який у десятки тисяч разів повільніший, ніж основна пам'ять), працювати із такою системою стане практично неможливо. Питання підвищення ефективності реалізації віртуальної пам'яті буде розглянуто в розділі 9.

Ще однією проблемою є фрагментація пам'яті, що виникає за ситуації, коли неможливо використати вільну пам'ять. Розрізняють зовнішню і внутрішню фрагментацію пам'яті.

Зовнішня зводиться до того, що внаслідок виділення і наступного звільнення пам'яті в ній утворюються вільні блоки малого розміру - діри (holes). Через це може виникнути ситуація, за якої неможливо виділити неперервний блок пам'яті розміру N, оскільки немає жодного неперервного вільного блоку, розмір якого S>N, хоча загалом обсяг вільного простору пам'яті перевищує N. Так, для виконання процесу P5 місця через зовнішню фрагментацію не вистачає.

Внутрішня фрагментація зводиться до того, що за запитом виділяють блоки пам'яті більшого розміру, ніж насправді будуть використовуватися, у результаті всередині виділених блоків залишаються невикористовувані ділянки, які вже не можуть бути призначені для чогось іншого.

5.Найважливішими поняттями концепції віртуальної пам'яті є логічна і фізична адресація пам'яті.

Логічна або віртуальна адреса — адреса, яку генерує програма, запущена на деякому процесорі. Адреси, що використовують інструкції конкретного процесора, є логічними адресами. Сукупність логічних адрес становить логічний адресний простір.

Фізична адреса - адреса, якою оперує мікросхема пам'яті. Прикладна програма в сучасних комп'ютерах ніколи не має справи з фізичними адресами. Спеціальний апаратний пристрій MMU (memory management unit - пристрій керування пам'яттю) відповідає за перетворення логічних адрес у фізичні. Сукупність усіх доступних фізичних адрес становить фізичний адресний простір. Отже, якщо в комп'ютері є мікросхеми на 128 Мбайт пам'яті, то саме такий обсяг пам'яті адресують фізично. Логічно зазвичай адресують значно більше пам'яті.

Алгоритм перетворення логічних адрес у фізичні визначає принцип організації та керування пам`яттю.

6.Під час реалізації віртуальної пам'яті необхідно забезпечити захист пам'яті, переміщення процесів у пам'яті та спільне використання пам'яті кількома процесами.

Одним із найпростіших способів задовольнити ці вимоги є підхід базового і межового регістрів. Для кожного процесу в двох регістрах процесора зберігають два значення - базової адреси (base) і межі (bounds). Кожний доступ до логічної адреси апаратно перетворюється у фізичну адресу шляхом додавання логічної адреси до базової. Якщо отримувана фізична адреса не потрапляє в діапазон (base, base+bounds), вважають, що адреса невірна, і генерують помилку (рис. 8.4).

Такий підхід є найпростішим прикладом реалізації динамічного переміщення процесів у пам'яті. Усі інші підходи, які буде розглянуто в цьому розділі, є різними варіантами розвитку цієї базової схеми. Наприклад, те, що кожний процес у разі використання цього підходу має свої власні значення базового і межового регістрів, є найпростішою реалізацією концепції адресного простору процесу, яка ґрунтується на тому, що кожний процес має власне відображення пам'яті.

Для організації захисту пам'яті в цій ситуації необхідно, щоб застосування користувача не могли змінювати значення базового і межового регістрів. Достатньо інструкції такої зміни зробити доступними тільки у привілейованому режимі процесора.

До переваг цього підходу належать простота, скромні вимоги до апаратного забезпечення (потрібні тільки два регістри), висока ефективність. Однак сьогодні його практично не використовують через низку недоліків, пов'язаних насамперед з тим, що адресний простір процесу все одно відображається на один неперервний блок фізичної пам'яті: незрозуміло, як

динамічно розширювати адресний простір процесу; різні процеси не можуть спільно використовувати пам'ять; немає розподілу коду і даних.

За такого підходу для процесу виділяють тільки одну пару значень «базова адреса-межа». Природним розвитком цієї ідеї стало відображення адресного простору процесу за допомогою кількох діапазонів фізичної пам'яті, кожен з яких задають власною парою значень базової адреси і межі. Так виникла концепція сегментації пам'яті.

Лекція №2. Тема: Сегментна та сторінкова організація пам`яті

1. Особливості сегментації пам’яті
2. Реалізація сегментації в архітектурі
3. Базові принципи сторінкової організації пам’яті
4. Порівняльний аналіз сторінкової організації пам’яті та сегментації

1.Сегментація пам'яті дає змогу зображати логічний адресний простір як сукупність незалежних блоків змінної довжини, які називають сегментами. Кожний сегмент містить дані одного призначення, наприклад в одному може бути стек, в іншому - програмний код і т. д.

У кожного сегмента є ім'я і довжина (для зручності реалізації поряд з іменами використовують номери). Логічна адреса складається з номера сегмента і зсуву всередині сегмента; з такими адресами працює прикладна програма. Компілятори часто створюють окремі сегменти для різних даних програми (сегмент коду, сегмент даних, сегмент стека). Під час завантаження програми у пам'ять створюють таблицю дескрипторів сегментів процесу, кожний елемент якої відповідає одному сегменту і складається із базової адреси, значення межі та прав доступу. Під час формування адреси її сегментна частина вказує на відповідний елемент таблиці дескрипторів сегментів процесу. Якщо зсув більший, ніж задане значення межі (або якщо права доступу процесу не відповідають правам, заданим для сегмента), то

апаратне забезпечення генерує помилку. Коли ж усе гаразд, сума бази і зсуву в разі чистої сегментації дасть у результаті фізичну адресу в основній пам'яті. Якщо сегмент вивантажений на диск, спроба доступу до нього спричиняє його завантаження з диска в основну пам'ять. У підсумку кожному сегменту відповідає неперервний блок пам'яті такої самої довжини, що перебуває в довільному місці фізичної пам'яті або на диску.

Переваги сегментації пам'яті.

- З'явилася можливість організувати кілька незалежних сегментів пам'яті для процесу і використати їх для зберігання даних різної природи. При цьому права доступу до кожного такого сегмента можуть бути задані по-різному.
- Окремі сегменти можуть спільно використовуватися різними процесами, для цього їхні таблиці дескрипторів сегментів повинні містити однакові елементи, що описують такий сегмент.
- Фізична пам'ять, що відповідає адресному простору процесу, тепер не обов'язково має бути неперервною. Справді, сегментація дає змогу окремим частинам адресного простору процесу відображатися не в основну пам'ять, а на диск, і довантажуватися з нього за потребою, забезпечуючи виконання процесів будь-якого розміру.
- Цей підхід не позбавлений і недоліків.
- Необхідність введення додаткового рівня перетворення пам'яті (перерахунок логічних адрес у фізичні спричиняє зниження продуктивності (цей недолік властивий будь-якій повноцінній реалізації віртуальної пам'яті). Для ефективної реалізації сегментації потрібна відповідна апаратна підтримка.
- Керування блоками пам'яті змінної довжини з урахуванням необхідності їхнього збереженні* на диску може бути досить складним.
- Вимога, щоб кожному сегменту відповідав неперервний блок фізичної пам'яті відповідного розміру, спричиняє зовнішню фрагментацію пам'яті. Внутрішньої фрагментації у цьому разі не виникає, оскільки

сегменти мають змінну довжину і завжди можна виділити сегмент довжини, необхідної для виконання програми.

Сьогодні сегментацію застосовують доволі обмежено передусім через фрагментацію і складність реалізації ефективного звільнення пам'яті та обміну із диском. Ширше використання отримав розподіл пам'яті на блоки фіксованої довжини - сторінкова організація пам'яті.

2.В архітектурі IA-32 догічні адреси в програмі формуються із використанням сегментації й мають такий вигляд: «селектор-зсув». Значення селектора завантажують у спеціальний регістр процесора (сегментний регістр) і використовують як індекс у таблиці дескрипторів сегментів, що перебуває в пам'яті та є аналогом таблиці сегментів, описаної раніше. В архітектурі IA-32 підтримуються шість сегментних регістрів. Це означає, що виконуваний код в один і той самий час може адресувати шість незалежних сегментів.

Селектор містите індекс дескриптора в таблиці, біт індикатора локальної або глобальної таблиці та необхідний рівень привілеїв.

Для системи задають спільну глобальну таблицю дескрипторів (Global Descriptor Table, GDT), а для кожної задачі - локальну таблицю дескрипторів (Local Descriptor Table, LDT).

Дескриптори в IA-32 мають довжину 64 біти. Вони визначають властивості програмних об'єктів (наприклад, сегментів пам'яті або таблиць дескрипторів).

Дескриптор містить значення бази (base), яке відповідає адресі об'єкта (наприклад, початок сегмента); значення межі (limit); тип об'єкта (сегмент, таблиця дескрипторів тощо); характеристики захисту.

Звертання до таблиць дескрипторів підтримується апаратно. Якщо задані в дескрипторі характеристики захисту не відповідають рівню привілеїв, визначеному селектором, отримати доступ до пам'яті за його допомогою буде неможливо. Так забезпечують захист пам'яті.

Проте жодного разу не було згадано, що в дескрипторі зберігають фізичну адресу. Річ у тому, що для архітектури IA-32 внаслідок перетворення логічної адреси отримують не фізичну адресу, а ще один вид адреси, який називають лінійною адресою.

3.До основних технологій реалізації віртуальної пам'яті крім сегментації належить сторінкова організація пам'яті (paging). її головна ідея - розподіл пам'яті блоками фіксованої довжини. Такі блоки називають сторінками.

Ця технологія є найпоширенішим підходом до реалізації віртуальної пам'яті в сучасних операційних системах.

У разі сторінкової організації пам'яті логічну адресу називають також лінійною, або віртуальною, адресою. Такі адреси належать одній множині (наприклад, лінійною адресою може бути невід'ємне ціле число довжиною 32 біти).

Фізичну пам'ять розбивають на блоки фіксованої довжини — фрейми, або сторінкові блоки (frames). Логічну пам'ять, у свою чергу, розбивають на блоки такої самої довжини — сторінки (pages). Коли процес починає виконуватися, його сторінки завантажуються в доступні фрейми фізичної пам'яті з диска або іншого носія.

Сторінкова організація пам'яті повинна мати апаратну підтримку. Кожна адреса, яку генерує процесор, ділиться на дві частини: номер сторінки і зсув сторінки. Номер сторінки використовують як індекс у таблиці сторінок.

Таблиця сторінок — це структура даних, що містить набір елементів (page-table entries, PTE), кожен із яких містить інформацію про номер сторінки, номер відповідного їй фрейму фізичної пам'яті (або беспосередньо його базову адресу) та права доступу. Номер сторінки використовують для пошуку елемента в таблиці. Після його знаходження до базової адреси відповідного фрейму додають зсув сторінки, чим і визначають фізичну адресу.

Розмір сторінки є ступенем числа 2, у сучасних ОС використовують сторінки розміром від 2 до 8 Кбайт. У спеціальних режимах адресації можна працювати зі сторінками більшого розміру.

Для кожного процесу створюють його власну таблицю сторінок. Коли процес починає своє виконання, ОС розраховує його розмір у сторінках і кількість фреймів у фізичній пам'яті. Кожну сторінку завантажують у відповідний фрейм, після чого його номер записують у таблицю сторінок процесу.

Відображення логічної пам'яті для процесу відрізняється від реального стану фізичної пам'яті. На логічному рівні для процесу вся пам'ять зображується неперервним блоком і належить тільки цьому процесові, а фізично вона розосереджена по адресному простору мікросхеми пам'яті, чергуючись із пам'яттю інших процесів. Процес не може звернутися до пам'яті, адреса якої не вказана в його таблиці сторінок (так реалізований захист пам'яті).

ОС повинна мати інформацію про поточний стан фізичної пам'яті (про зайнятість і незайнятість фреймів, їхню кількість тощо). Цю інформацію звичайно зберігають у таблиці фреймів. Кожний її елемент відповідає фрейму і містить всі відомості про нього.

4.Сторінкова організація пам'яті та сегментація мають більше спільних рис, аніж відмінностей. Основна відмінність між цими двома підходами полягає в тому, що всі сторінки мають фіксовану довжину, а сегменти змінну. Інші базові моменти (відсутність вимоги неперервності фізичної пам'яті, можливість вивантаження блоків пам'яті на диск, необхідність підтримувати таблиці перетворення тощо) принципово не відрізняються.

Розглянемо основні переваги сторінкової організації пам'яті порівняно із сегментацією. Вони визначаються насамперед тим, що всі сторінки мають одну й ту саму довжину.

- Реалізація розподілу і звільнення пам'яті спрощується. Усі сторінки з погляду процесу рівноправні, тому можна підтримувати список вільних сторінок і в разі необхідності виділяти першу сторінку із цього списку, а після звільнення повертати сторінку в список. Із сегментами так чинити не можна, оскільки кожен сегмент можна використати лише за його призначенням (спроба використати сегмент для іншої мети призведе швидше за все до того, що виникне потреба у сегменті іншої довжини).
- Реалізація обміну даними з диском також спрощується. Для організації такого обміну ділянка на диску, яку використовують для зберігання інформації про сторінки, вивантажені з пам'яті {простір підтримки, backing store) може бути теж розбита на блоки фіксованого розміру, рівного розмірові фрейму.

Сторінкова організація пам'яті не позбавлена й недоліків.

- Насамперед цей підхід спричиняє внутрішню фрагментацію, пов'язану з тим, що розмір сторінки завжди фіксований, і в разі необхідності виділення блоку пам'яті конкретної довжини його розмір буде кратним розміру сторінки. У середньому розмір невикористовуваної пам'яті становить приблизно половину сторінки для кожного виділеного блоку пам'яті (аналогічного до сегмента). Така фрагментація може бути знижена зменшенням кількості та збільшенням розміру блоків, що виділяються.
- Таблиці сторінок мають бути більші за розміром, ніж таблиці сегментів. Так, для виділення неперервного діапазону пам'яті розміром 100 Кбайт знадобиться один елемент таблиці сегментів, що описує сегмент, виділений для цього діапазону. З іншого боку, у разі використання сторінок розміром 4 Кбайт для опису такого діапазону нам знадобиться 25 елементів таблиці сторінок - по одному елементу для кожної сторінки.

Лекція №3. Тема: Таблиці сторінок та сторінково-сегментна організація пам`ті

1. Багаторівневі таблиці сторінок
2. Реалізація таблиць сторінок в архітектурі IA-32
3. Асоціативна пам’ять
4. Сторінково-сегментна організація пам’яті

1.Щоб адресувати логічний адресний простір значного обсягу за допомогою однієї таблиці сторінок, її доводиться робити дуже великою. Наприклад, в архітектурі IA-32 за стандартного розміру сторінки 4 Кбайт (для адресації всередині такої сторінки потрібні 12 біт) на індекс у таблиці залишається 20 біт, що відповідає таблиці сторінок на 1 мільйон елементів.

Щоб уникнути таких великих таблиць, запропоновано технологію багаторівневих таблиць сторінок. Таблиці сторінок самі розбиваються на сторінки, інформацію про які зберігають в таблиці сторінок верхнього рівня. Кількість рівнів рідко перевищує 2, але може доходити й до 4.

Коли є два рівні таблиць, логічну адресу розбивають на індекс у таблиці верхнього рівня, індекс у таблиці нижнього рівня і зсув.

Ця технологія має дві основні переваги. По-перше, таблиці сторінок стають менші за розміром, тому пошук у них можна робити швидше. По-друге, не всі таблиці сторінок мають перебувати в пам'яті у конкретний момент часу. Наприклад, якщо процес не використовує якийсь блок пам'яті, то вміст усіх сторінок нижнього рівня невикористовуваного блоку може бути тимчасово збережений на диску.

2.Архітектура IA-32 використовує дворівневу сторінкову організацію, починаючи з моделі Intel 80386.

Таблицю верхнього рівня називають каталогом сторінок (page directory), для кожної задачі повинен бути заданий окремий каталог сторінок, фізичну адресу якого зберігають у спеціальному керуючому регістрі сг3 і куди він автоматично завантажується апаратним забезпеченням при

перемиканні контексту. Таблицю нижнього рівня називають просто таблицею сторінок (page table).

Лінійна адреса поділяється на три поля:

♦ каталогу (Directory) - визначає елемент каталогу сторінок, що вказує на потрібну таблицю сторінок;

♦ таблиці (Table) - визначає елемент таблиці сторінок, що вказує на потрібний фрейм пам'яті;

♦ зсуву (Offset) — визначає зсув у межах фрейму, що у поєднанні з адресою фрейму формує фізичну адресу.

Розмір полів каталогу і таблиці становить 10 біт, що дає таблиці сторінок, які містять 1024 елементи, розмір поля зсуву - 12 біт, що дає сторінки і фрейми розміром 4 Кбайт. Одна таблиця сторінок нижнього рівня адресує 4 Мбайт пам'яті (1 Мбайт фреймів), а весь каталог сторінок — 4 Гбайт.

Елементи таблиць сторінок всіх рівнів мають однакову структуру. Виокремимо такі поля елемента:

♦прапорець присутності (Present), дорівнює одиниці, якщо сторінка перебуває у фізичній пам'яті (їй відповідає фрейм); рівність цього прапорця нулю означає, що сторінки у фізичній пам'яті немає, при цьому операційна система може використати інші поля елемента для своїх цілей;

♦ 20 найбільш значущих бітів, які задають початкову адресу фрейму, кратну 4 Кбайт (може бути задано 1 Мбайт різних початкових адрес);

♦ прапорець доступу (Accessed), який покладають рівним одиниці під час кожного звертання пристрою сторінкової підтримки до відповідного фрейму;

♦ прапорець зміни (Dirty), який покладають рівним одиниці під час кожної операції записування у відповідний фрейм;

♦ прапорець читання-записування (Read/Write), що задає права доступу до цієї сторінки або таблиці сторінок (для читання і для записування або тільки для читання);

♦ прапорець привілейованого режиму (User/Supervisor), який визначає режим процесора, необхідний для доступу до сторінки. Якщо цей прапорець дорівнює нулю, сторінка може бути адресована тільки із привілейованого режиму, якщо одиниці - доступна також і з режиму користувача;

Прапорці присутності, доступу і зміни можна використовувати ОС для організації віртуальної пам'яті.

3.Під час реалізації таблиць сторінок для отримання доступу до байта фізичної пам'яті доводиться звертатися до пам'яті кілька разів. У разі використання дворівневих сторінок потрібні три операції доступу: до каталогу сторінок, до таблиці сторінок і безпосередньо за адресою цього байта, а для трирівневих таблиць — чотири операції. Це сповільнює доступ до пам'яті та знижує загальну продуктивність системи.

Як уже зазначалося, правило «дев'яносто до десяти» свідчить, що більша частина звертань до пам'яті процесу належить до малої підмножини його сторінок, причому склад цієї підмножини змінюється досить повільно. Засобом підвищення продуктивності у разі сторінкової організації пам'яті є кешування адрес фреймів пам'яті, що відповідають цій підмножині сторінок.

Для розв'язання цієї проблеми було запропоновано технологію асоціативної пам'яті або кеша трансляції (translation look-aside buffers, TLB). У швидкодіючій пам'яті (швидшій, ніж основна пам'ять) створюють набір із кількох елементів (різні архітектури відводять під асоціативну пам'ять від 8 до 2048 елементів, в архітектурі IA-32 таких елементів до Pentium 4 було 32, починаючи з Pentium 4 — 128). Кожний елемент кеша трансляції відповідає одному елементу таблиці сторінок.

Тепер під час генерування фізичної адреси спочатку відбувається пошук відповідного елемента таблиці в кеші (в IA-32 — за полем каталогу, полем таблиці та зсуву), і якщо він знайдений, стає доступною адреса відповідного фрейму, що негайно можна використати для звертання до пам'яті. Якщо ж у кеші відповідного елемента немає, то доступ до пам'яті

здійснюють через таблицю сторінок, а після цього елемент таблиці сторінок зберігають в кеші замість найстарішого елемента.

На жаль, у разі перемикання контексту в архітектурі IA-32 необхідно очистити весь кеш, оскільки в кожного процесу є своя таблиця сторінок, і ті ж самі номери сторінок для різних процесів можуть відповідати різним фреймам у фізичній пам'яті. Очищення кеша трансляції є дуже повільною операцією, якої треба всіляко уникати .

Важливою характеристикою кеша трансляції є відсоток влучень, тобто відсоток випадків, коли необхідний елемент таблиці сторінок перебуває в кеші і не потребує доступу до пам'яті. Відомо, що при 32 елементах забезпечується 98 % влучень. Зазначимо також, що за такого відсотку влучень зниження продуктивності у разі використання дворівневих таблиць сторінок порівняно з однорівневими становить 28 %, однак переваги, отримувані під час розподілу пам'яті, роблять таке зниження допустимим.

4.Базові принципи.Оскільки сегменти мають змінну довжину і керувати ними складніше, чиста сегментація зазвичай не настільки ефективна, як сторінкова організація. З іншого боку, видається цінною сама можливість використати сегменти як блоки пам'яті різного призначення змінної довжини.

Для того щоб об'єднати переваги обох підходів, у деяких апаратних архітектурах (зокрема, в IA-32) використовують комбінацію сегментної та сторінкової організації пам'яті. За такої організації перетворення логічної адреси у фізичну відбувається за три етапи.

1.У програмі задають логічну адресу із використанням сегмента і зсуву.

2.Логічну адресу перетворюють у лінійну (віртуальну) адресу за правилами, заданими для сегментації.

3.Віртуальну адресу перетворюють у фізичну за правилами, заданими для сторінкової організації.

Таку архітектуру називають сторінково-сегментною організацією пам'яті.

Перетворення адрес в архітектурі IA-32

Розглянемо особливості реалізації описаних трьох етапів перетворення адреси в архітектурі IA-32.

1.Машинна мова архітектури IA-32 (а, отже, будь-яка програма, розроблена для цієї архітектури) оперує логічними адресами. Логічна адреса, як було зазначено раніше, складається із селектора і зсуву.

2.Лінійна або віртуальна адреса — це ціле число без знака завдовжки 32 біти. За його допомогою можна дістати доступ до 4 Гбайт комірок пам'яті. Перетворення логічної адреси в лінійну відбувається всередині пристрою сегментації (segmentation unit) за правилами перетворення адреси на базі сегментації, описаними раніше.

3.Фізичну адресу використовують для адресації комірок пам'яті в мікросхемах пам'яті. її теж зображають 32-бітовим цілим числом без знака. Перетворення лінійної адреси у фізичну відбувається всередині пристрою сторінкової підтримки (paging unit) за правилами для сторінкової організації пам'яті (лінійну адресу розділяють апаратурою на адресу сторінки і сторінковий зсув, а потім перетворюють у фізичну адресу із використанням таблиць сторінок, кеша трансляції тощо).

Необхідність підтримки сегментації в IA-32 значною мірою є даниною традиції (це пов'язано з необхідністю зворотної сумісності зі старими моделями процесорів, у яких була відсутня підтримка сторінкової організації пам'яті). Сучасні ОС часто обходять таку сегментну організацію майже повністю, використовуючи в системі лише кілька загальних сегментів, причому кожен із них задають селектором, дескрипторі якого поле base дорівнює нулю, а поле limit - максимальній адресі лінійної пам'яті. Зсув логічної адреси завжди буде рівний лінійній адресі, а отже, лінійну адресу можна буде формувати у програмі, фактично переходячи до чисто сторінкової організації пам'яті.

Лекція №3. Тема: Організація пам`яті в Linux

1. Використання сегментації в Linux, формування логічних адрес
2. Сторінкова адресація в Linux
3. Розташування ядра у фізичній пам’яті
4. Особливості адресації процесів і ядра
5. Використання асоціативної пам’яті

1.Як уже зазначалося, необхідність підтримки сегментації призводить до того, що програми стають складнішими, оскільки задача виділення сегментів і формування коректних логічних адрес лягає на програміста. Цю проблему в Linux вирішують доволі просто - ядро практично не використовує засобів підтримки сегментації архітектури IA-32. У системі підтримують мінімальну кількість сегментів, без яких неможлива коректна адресація пам'яті процесором (сегменти коду і даних ядра та режиму користувача). Код ядра і режиму користувача спільно використовує ці сегменти.

Сегменти коду використовують під час формування логічних адрес коду (для виклику процедур тощо); такі сегменти позначають як доступні для читання і виконання. Сегменти даних призначені для формування логічних адрес даних (глобальних змінних, стека тощо) і позначаються як доступні для читання і записування. Сегменти режиму користувача доступні з режиму користувача, сегменти ядра - тільки з режиму ядра.

Усі сегменти, які використовуються у Linux, визначають межу зсуву, що дає змогу створити в рамках кожного з них 4 Гбайт логічних адрес. Це означає, що Linux фактично передає всю роботу з керування пам'яттю на рівень перетворення між лінійними і фізичними адресами (оскільки кожна логічна адреса відповідає лінійній).

Далі в цьому розділі вважатимемо логічні адреси вже сформованими (на базі відповідного сегмента) і перетвореними на лінійні адреси.

2.У ядрі Linux версії 2.4 використовують трирівневу організацію таблиць сторінок. Підтримуються три типи таблиць сторінок: глобальний (Page Global Directory, PGD); проміжний каталог сторінок (Page Middle Directory, PMD); таблиця сторінок (Page Table).

Кожний глобальний каталог містить адреси одного або кількох проміжних каталогів сторінок, а ті, своєю чергою, - адреси таблиць сторінок. Елементи таблиць сторінок (PTE) вказують на фрейми фізичної пам'яті.

Кожний процес має свій глобальний каталог сторінок і набір таблиць сторінок. Під час перемикання контексту Linux зберігає значення регістра сг3 у керуючому блоці процесу, що передає керування, і завантажує в цей регістр значення з керуючого блоку процесу, що починає виконуватися. Отже, коли процес починає виконуватися, пристрій сторінкової підтримки вже посилається на коректний набір таблиць сторінок.

Тепер розглянемо роботу цієї трирівневої організації для архітектури IA-32, яка дає можливість мати тільки два рівні таблиць. Насправді ситуація досить проста — проміжний каталог таблиць оголошують порожнім, водночас його місце в ланцюжку покажчиків зберігають для того, щоб той самий код міг працювати для різних архітектур. У цьому разі PGD відповідає каталогу сторінок IA-32 (його елементи містять адреси таблиць сторінок), а під час роботи із покажчиком на PMD насправді працюють із покажчиком на відповідний йому елемент PGD, відразу отримуючи доступ до таблиці сторінок. Між таблицями сторінок Linux і таблицями сторінок IA-32 завжди дотримується однозначна відповідність.

Для платформо-незалежного визначення розміру сторінки в Linux використовують системний виклик getpagesize( ):

3.Ядро Linux завантажують у набір зарезервованих фреймів пам'яті, які заборонено вивантажувати на диск або передавати процесу користувача, що захищає код і дані ядра від випадкового або навмисного ушкодження.

Завантаження ядра починається із другого мегабайта пам'яті (перший мегабайт пропускають, тому що в ньому є ділянки, які не можуть бути використані, наприклад відеопам'ять текстового режиму, код BIOS тощо). Із ядра завжди можна визначити фізичні адреси початку та кінця його коду і даних.

4.Лінійний адресний простір кожного процесу поділяють на дві частини: перші 3 Гбайт адрес використовують у режимі ядра та користувача; вони відображають захищений адресний простір процесу; решту 1 Гбайт адрес використовують тільки в режимі ядра.

Елементи глобального каталогу процесу, що визначають адреси до 3 Гбайт, можуть бути задані самим процесом, інші елементи мають бути однаковими для всіх процесів і задаватися ядром.

Потоки ядра (див. розділ 3) не використовують елементів глобального каталогу першого діапазону. На практиці, коли відбувається передавання керування потоку ядра, не змінюється значення регістра сг3, тобто потік ядра використовує таблиці сторінок процесу користувача, що виконувався останнім (оскільки йому потрібні тільки елементи, доступні в режимі ядра, а вони в усіх процесах користувача однакові).

Адресний простір ядра починається із четвертого гігабайта лінійної пам'яті. Для прямого відображення на фізичні адреси доступні перші 896 Мбайт цього простору (128 Мбайт, що залишилися, використовується переважно для динамічного розподілу пам'яті ядром).

5. Під час роботи з асоціативною пам'яттю основне завдання ядра полягає у зменшенні потреби її очищення. Для цього вживають таких заходів.

- Під час планування невелику перевагу має процес, який використовує той самий набір таблиць сторінок, що й процес, який повертає

керування (під час перемикання між такими процесами очищення кеша трансляції не відбувається).

- Реалізація потоків ядра, котрі використовують таблиці сторінок останнього процесу, теж призводить до того, що під час перемикання між процесом і потоком ядра очищення не відбувається.

Лекція №4. Тема: Організація пам`яті у Windows XP

1. Сегментація у Windows XP
2. Сторінкова адресація у Windows XP
3. Особливості адресації процесів і ядра
4. Структура адресного простору процесів і ядра
5. Висновки концепції керування оперативною пам’яттю

1.Система Windows XP використовує загальні сегменти пам'яті подібно до того, як це робиться в Linux. Для всіх сегментів у програмі задають однакові значення бази і межі, тому роботу з керування пам'яттю аналогічним чином передають на рівень лінійних адрес (які є зсувом у цих загальних сегментах).

2.Під час роботи з лінійними адресами у Windows XP використовують дворівневі таблиці сторінок, повністю відповідні архітектурі IA-32. У кожного процесу є свій каталог сторінок, кожен елемент якого вказує на таблицю сторінок. Таблиці сторінок усіх рівнів містять по 1024 елементи таблиці сторінок, кожний такий елемент вказує на фрейм фізичної пам'яті. Фізичну адресу каталогу сторінок зберігають у блоці KPROCESS.

Розмір лінійної адреси, з якою працює система, становить 32 біти. З них 10 біт відповідають адресі в каталозі сторінок, ще 10 — це індекс елемента в таблиці, останні 12 біт адресують конкретний байт сторінки (і є зсувом).

Розмір елемента таблиці сторінок теж становить 32 біти. Перші 20 біт адресують конкретний фрейм (і використовуються разом із останніми 12 біт лінійної адреси), а інші 12 біт описують атрибути сторінки (захист, стан сторінки в пам'яті, який файл підкачування використовує). Якщо сторінка не перебуває у пам'яті, то в перших 20 біт зберігають зсув у файлі підкачування.

3.Лінійний адресний простір процесу поділяється на дві частини: перші 2 Гбайт адрес доступні для процесу в режимі користувача і є його захищеним адресним простором; інші 2 Гбайт адрес доступні тільки в режимі ядра і відображають системний адресний простір.

Зазначимо, що таке співвідношення між адресним простором процесу і ядра відрізняється від прийнятого в Linux (3 Гбайт для процесу, 1 Гбайт для ядра).

Деякі версії Windows XP дають можливість задати співвідношення 3 Гбайт/1 Гбайт під час завантаження системи.

4.В адресному просторі процесу можна виділити такі ділянки:

- перші 64 Кбайт (починаючи з нульової адреси) — це спеціальна ділянка, доступ до якої завжди спричиняє помилки;
- усю пам'ять між першими 64 Кбайт і останніми 136 Кбайт (майже 2 Гбайт) може використовувати процес під час свого виконання;
- далі розташовані два блоки по 4 Кбайт: блоки оточення потоку (TEB) і процесу (PEB) (див. розділ 3);
- наступні 4 Кбайт — ділянка пам'яті, куди відображаються різні системні дані (системний час, значення лічильника системних годин, номер версії системи), тому для доступу до них процесу не потрібно перемикатися в режим ядра;
- останні 64 Кбайт використовують для запобігання спробам доступу за межі адресного простору процесу (спроба доступу до цієї пам'яті дасть помилку).

Системний адресний простір містить велику кількість різних ділянок. Найважливіші з них такі:

- Перші 512 Мбайт системного адресного простору використовують для завантаження ядра системи.
- 4 Мбайт пам'яті виділяють під каталог сторінок і таблиці сторінок процесу.
- Спеціальну ділянку пам'яті розміром 4 Мбайт, яку називають гіперпростором (hyperspace), використовують для відображення різних структур даних, специфічних для процесу, на системний адресний простір (наприклад, вона містить список сторінок робочого набору процесу).
- 512 Мбайт виділяють під системний кеш.
- У системний адресний простір відображаються спеціальні ділянки пам'яті - вивантажуваний пул і невивантажуваний пул, які розглянемо в розділі 10.
- Приблизно 4 Мбайт у самому кінці системного адресного простору виділяють під структури даних, необхідні для створення аварійного образу пам'яті, а також для структур даних HAL.

5.Керування пам'яттю є однією з найскладніших задач, які стоять перед операційною системою. Щоб нестача пам'яті не заважала роботі користувача, потрібно розв'язувати задачу координації різних видів пам'яті. Можна використовувати повільнішу пам'ять для збільшення розміру швидшої (на цьому ґрунтується технологія віртуальної пам'яті), а також швидшу - для прискорення доступу до повільнішої (на цьому ґрунтується кешування).

- Технологія віртуальної пам'яті передбачає введення додаткових перетворень між логічними адресами, які використовують програми, та фізичними, що їх розуміє мікросхема пам'яті. На основі таких перетворень може бути реалізований захист пам'яті; крім того, вони дають змогу процесу розміщатися у фізичній пам'яті не неперервно і не повністю (ті його частини, які в цей момент часу не потрібні, можуть бути збережені на жорсткому

диску). Ця технологія спирається на той факт, що тільки частина адрес процесу використовується в конкретний момент часу, тому коли зберігати в основній пам'яті тільки її, продуктивність процесу залишиться прийнятною.

- Базовими підходами до реалізації віртуальної пам'яті є сегментація і сторінкова організація. Обидва ці підходи дають можливість розглядати логічний адресний простір процесу як сукупність окремих блоків, кожен з яких може бути відображений на основну пам'ять або на диск. Головна відмінність полягає в тому, що у випадку сегментації блоки мають змінну довжину, а у разі сторінкової організації — постійну. Сьогодні часто трапляється комбінація цих двох підходів (сторінково-сегментна організація пам'яті).
- У разі сторінкової організації пам'яті логічна адреса містить номер у спеціальній структурі даних - таблиці сторінок, а також зсув відносно початку сторінки. Розділення адреси на частини відбувається апаратно. Елемент таблиці сторінок містить адресу початку блоку фізичної пам'яті, у який відображається сторінка (такий блок називають фреймом) і права доступу. Він може також відповідати сторінці, відображеній на диск. Таблиці сторінок можуть містити кілька рівнів. Таблицю верхнього рівня називають каталогом сторінок. Кожний процес має свій набір таких таблиць. Для прискорення доступу останні використані елементи таблиць сторінок кешуються в асоціативній пам'яті.

Розділ 5. Логічна та фізична організація файлових систем

Лекція №1. Тема: Організація інформації у файловій системі

1. Поняття файла і файлової системи
2. Організація інформації розділами та каталогами у файловій системі
3. Зв’язок розділів і структури каталогів
4. Зв’язки між іменами файлів

5. Атрибути файлів

6. Операції над файлами і каталогами

1.Файл - це набір даних, до якого можна звертатися за іменем. Файли організовані у файлові системи. З погляду користувача файл є мінімальним обсягом даних файлової системи, з яким можна працювати незалежно. Наприклад, користувач не може зберегти дані на зовнішньому носії, не звернувшися при цьому до файла. Розглянемо особливості використання файлів.

▪ Файли є найпоширенішим засобом зберігання інформації в енергонезалежній пам'яті. Така пам'ять надійніша, й інформація на ній може зберігатися так довго, як це необхідно. Зазначимо, що більшість збоїв у роботі ОС не руйнує інформації, що зберігається у файлах на диску. Для забезпечення збереження даних підвищеної цінності вживають додаткових заходів (гаряче резервування, резервне копіювання тощо).

▪ Файли забезпечують найпростіший варіант спільного використання даних різними застосуваннями. Це пов'язано з тим, що файли відокремлені від програм, які їх використовують: будь-яка програма, якій відоме ім'я файла, може отримати доступ до його вмісту. Якщо одна програма запише у файл, а інша його потім прочитає, то ці дві програми виконають обмін даними.

Файлова система - це підсистема ОС, що підтримує організований набір файлів, здебільшого у конкретній ділянці дискового простору (логічну структуру); низькорівневі структури даних, використовувані для організації цього простору у вигляді набору файлів (фізичну структуру); програмний інтерфейс файлової системи (набір системних викликів, що реалізують операції над файлами).

Файлова система надає прикладним програмам абстракцію файла. Прикладні програми не мають інформації про те, як організовані дані файла,

як знаходять відповідність між ім'ям файла і його даними, як пересилають дані із диска у пам'ять тощо - усі ці операції забезпечує файлова система.

Важливо зазначити, що файлові системи можуть надавати інтерфейс доступу не тільки до диска, але й до інших пристроїв. Є навіть файлові системи, які не зберігають інформацію, а генерують її динамічно за запитом. Втім, для прикладних програм усі такі системи мають однаковий вигляд.

До головних задач файлової системи можна віднести: організацію її логічної структури та її відображення на фізичну організацію розміщення даних на диску; підтримку програмного інтерфейсу файлової системи; забезпечення стійкості проти збоїв; забезпечення розподілу файлових ресурсів за умов багатозадачності та захисту даних від несанкціонованого доступу.

Типи файлів

Раніше ОС підтримували файли різної спеціалізованої структури. Сьогодні є тенденція взагалі не контролювати на рівні ОС структуру файла, відображаючи кожен файл простою послідовністю байтів. У цьому разі застосування, які працюють із файлами, самі визначають їхній формат.

Такий спрощений підхід справедливий не для всіх файлів. Є спеціальні файли, що їх операційна система інтерпретує особливим чином. Структуру таких файлів ОС підтримує відповідно до тих задач, які з їхньою допомогою розв'язуються.

Ще однією категорією файлів є виконувані файли. Хоч їх звичайно не розглядають разом зі спеціальними файлами, вони мають жорстко заданий формат, який розпізнає операційна система. Часто буває так, що ОС може працювати із виконуваними файлами різних форматів.

Ще одним варіантом класифікації є поділ на файли із прямим і послідовним доступом. Файли із прямим доступом дають змогу вільно переходити до будь-якої позиції у файлі, використовуючи для цього поняття покажчика поточної позиції файла (seek pointer), що може переміщатися у будь-якому напрямку за допомогою відповідних системних викликів. Файли

із послідовним доступом можуть бути зчитані тільки послідовно, із початку в кінець. Сучасні ОС звичайно розглядають усі файли як файли із прямим доступом.

Імена файлів

Важливою складовою роботи із файлами є організація доступу до них за іменем.

Різні системи висувають різні вимоги до імен файлів. Так, у деяких системах імена є чутливими до регістра (myfile.txt і MYFILE.TXT будуть різними іменами), а в інших - ні.

Операційна система може розрізняти окремі частини імені файла. Кілька останніх символів імені, відокремлених від інших символів крапкою, у деяких системах називають розширенням файла, яке може характеризувати його тип. В інших системах обов'язкове розширення не виділяють, при цьому деякі програми можуть, однак, розпізнавати потрібні їм файли за розширеннями (наприклад, компілятор С може розраховувати на те, що вихідні файли програм матимуть розширення .c).

Важливою характеристикою файлової системи є максимальна довжина імені файла. У минулому багато ОС різним чином обмежували довжину імен файлів. Широко відоме було обмеження на 8 символів у імені файла і 3 - у розширенні, присутнє у файловій системі FAT до появи Windows 95. Сьогодні стандартним значенням максимальної довжини імені файла є 255 символів.

2.Кожний розділ може мати свою файлову систему (і, можливо, використовуватися різними ОС). Для поділу дискового простору на розділи використовують спеціальну утиліту, яку часто називають fdisk.

Для генерації файлової системи на розділі потрібно використати операцію високорівневого форматування диска. У деяких ОС під томом (volume) розуміють розділ із встановленою на ньому файловою системою.

Реалізація розділів дає змогу відокремити логічне відображення дискового простору від фізичного і підвищує гнучкість використання файлових систем.

Каталоги

Розділи є основою організації великих обсягів дискового простору для розгортання файлових систем. Для організації файлів у рамках розділу зі встановленою файловою системою було запропоновано поняття файлового каталогу (file directory) або просто каталогу.

Каталог - це об'єкт (найчастіше реалізований як спеціальний файл), що містить інформацію про набір файлів. Про такі файли кажуть, що вони містяться в каталозі. Файли заносяться в каталоги користувачами на підставі їхніх власних критеріїв, деякі каталоги можуть містити дані, потрібні операційній системі, або її програмний код.

Каталог можна уявити собі як символьну таблицю, що реалізує відображення імен файлів у елементи каталогу (зазвичай в таких елементах зберігають низько-рівневу інформацію про файли). Подивимося, як може бути реалізоване таке відображення.

Деревоподібна структура каталогів

Базовою ідеєю організації даних за допомогою каталогів є те, що вони можуть містити інші каталоги. Вкладені каталоги називають підкаталогами (subdirectories). Таким чином формують дерево каталогів. Перший каталог, створений у файловій системі, встановленій у розділі (корінь дерева каталогів), називають кореневим каталогом (root directory).

Поняття шляху

Розглянемо, яким чином формують ім'я файла з урахуванням багаторівневої структури каталогів.

Для файла, розташованого всередині каталогу недостатньо його імені для однозначного визначення, де він перебуває - в іншому каталозі може бути файл із тим самим ім'ям. Тепер для визначення місцезнаходження файла потрібно додавати до його імені список каталогів, де він перебуває. Такий

список називають шляхом (path). Каталоги у шляху перераховують зліва направо - від меншої глибини вкладеності до більшої. Роздільник каталогів у шляху відрізняється для різних систем: в UNIX прийнято використовувати прямий слеш «/», а у Windows-системах - зворотний «\».

Абсолютний і відносний шляхи

Є два шляхи до файла: абсолютний і відносний. Абсолютний (або повний) повністю й однозначно визначає місце розташування файла. Такий шлях обов'язково має містити кореневий каталог. Ось приклад абсолютного шляху для UNIX-систем: /usr/local/bin/myfile.

Якщо застосування використовує тільки абсолютні шляхи, йому зазвичай бракує гнучкості. Наприклад, у разі перенесення в інший каталог потрібно буде вручну відредагувати всі шляхи, замінивши їх новими.

Відносний - шлях, відлічуваний від деякого місця в ієрархії каталогів. Щоб його організувати, потрібно визначитися із точкою відліку, для чого використовують поняття поточного каталогу. Такий каталог задають для кожного процесу, і він може бути змінений у будь-який момент командою cd або системним викликом chdir ( ). Відносний шлях може відлічуватися від поточного каталогу і звичайно кореневий каталог не включає. Прикладом відносного шляху до файла /usr/ local /bin/myfile (за умови, що поточним каталогом є /usr/local) буде bin/myfile, а в ситуації, коли поточним є каталог файла (/usr/local/bin), відносним шляхом буде просто ім'я файла: myfile.

Для спрощення побудови відносного шляху кожний каталог містить два спеціальні елементи:

- «.»- посилання на поточний каталог
- «..» - посилання на каталог рівнем вище-батьківський.

З урахуванням цих елементів можуть бути задані такі відносні шляхи, як ../../bin/myfile (за умови, що поточний каталог - /usr/local/lib/mylib) або ./myfile (вказує на елемент у поточному каталозі).

Застосування, що обмежується тільки відносними шляхами під час доступу до файлів (особливо, якщо вони не виходять за межі каталогу цього

застосування), може бути без змін перенесене в інший каталог тієї самої структури.

Є й інші можливості полегшити задання шляхів доступу до файлів у каталогах. Одним із найпоширеніших способів є використання змінної оточення PATH, що містить список часто використовуваних каталогів. У разі доступу до файла за іменем його пошук спочатку виконуватиметься в каталогах, заданих за допомогою PATH.

3.Залишилося з'ясувати важливе питання про взаємозв'язок розділів і структури каталогів файлових систем. Розрізняють два основні підходи до реалізації такого взаємозв'язку, які істотно відрізняються один від одного.

Єдине дерево каталогів. Монтування файлових систем

Перший підхід в основному використовується у файловій системі UNIX і полягає в тому, що розділи зі встановленими на них файловими системами об'єднуються в єдиному дереві каталогів ОС.

Стандартну організацію каталогів UNIX зображують у вигляді дерева з одним коренем - кореневим каталогом, який позначають «/». Файлову систему, на якій перебуває кореневий каталог, називають завантажувальною або кореневою. У більшості реалізацій вона має містити файл із ядром ОС.

Додаткові файлові системи об'єднуються із кореневою за допомогою операції монтування (mount). Під час монтування вибраний каталог однієї файлової системи стає кореневим каталогом іншої. Каталог, призначений для монтування файлової системи, називають точкою монтування (mount point). Весь вміст файлової системи, приєднаної за допомогою монтування, виглядає для користувачів системи як набір підкаталогів точки монтування.

У цьому разі на диску є два розділи. На кожному з них встановлена файлова система (типи файлових систем можуть бути різними - це не є обмеженням; у каталозі системи одного типу можна змонтувати систему іншого типу за умови, що цей тип підтримує ОС). На рисунку точкою монтування ми вибрали каталог /usr першої файлової системи. Для

користувача системи практично не помітно, що насправді каталог / і каталог /usr відповідають різним файловим системам. Відмінності можуть виявлятися, наприклад, під час спроби перенесення файла: виконання звичайної операції перенесення (mv у UNIX) між файловими системами не дозволяється.

Розглянемо деякі наслідки застосування єдиного каталогу для організації файлової системи.

- Будь-який файл може бути адресований побудовою відносного шляху від будь-якого каталогу.
- Від користувача прихована структура розділів жорсткого диска, яка йому у більшості випадків не потрібна.
- Адміністрування системи спрощується. Наприклад, якщо додамо ще один диск і захочемо перенести на нього каталог /home, достатньо буде виконати кілька простих дій: відформатувати цей диск, задавши на ньому один розділ; змонтувати цей розділ у довільному місці; перенести на нього каталог /home (стерши весь його вміст на вихідному диску); заново змонтувати цей розділ у каталозі /home кореневої файлової системи.

Внаслідок цих дій всі застосування, які використовують каталог /home, працюватимуть у колишньому режимі; на їхню роботу не вплине той факт, що каталог тепер відповідає новій файловій системі, а /home став точкою монтування.

Літерні позначення розділів

Другий підхід, що в основному поширений в лініях Consumer Windows і Windows XP, припускає, що кожний розділ зі встановленою файловою системою є видимим для користувача і позначений буквою латинського алфавіту. Такий розділ звичайно називають томом. Позначення томів нам знайомі — цеC:, D: тощо.

Особливості такої реалізації наведені нижче.

- Вміст кожного розділу не пов'язаний з іншими розділами; відносний шлях можна побудувати тільки за умови, що поточний каталог перебуває на тому самому томі, що і файл.
- Структура логічних розділів видима для користувача.
- Перенос каталогу на новий розділ призводить до того, що шлях до цього каталогу зміниться (оскільки такий шлях завжди включає літерне позначення тому). У підсумку програмне забезпечення, яке використовує цей шлях, може перестати працювати.
- У разі необхідності додавання або вилучення дискового пристрою у системах лінії Consumer Windows користувач не може впливати на те, які літери система присвоює розділам (фактично це залежить від порядку підключення апаратних пристроїв); у системах лінії Windows XP користувач може вільно змінювати літерні позначення під час роботи системи.

Зазначимо, що нині в ОС лінії Windows XP реалізована підтримка монтування (для файлової системи NTFS), що вирішує більшість перелічених проблем. Ця підтримка вперше з'явилась у Windows 2000 .

4.Структура каталогів файлової системи не завжди є деревом. Багато файлових систем дає змогу задавати кілька імен для одного й того самого файла. Такі імена називають зв'язками (links). Розрізняють жорсткі та символічні зв'язки.

Жорсткі зв'язки

Ім'я файла не завжди однозначно пов'язане з його даними. За підтримки жорстких зв'язків (hard links) для файла допускається кілька імен. Усі жорсткі зв'язки визначають одні й ті самі дані на диску, для користувача вони не відрізняються: не можна визначити, які з них були створені раніше, а які - пізніше.

Підтримка жорстких зв'язків у POSIX

Для створення жорстких зв'язків у POSIX призначений системний виклик link().Першим параметром він приймає ім'я вихідного файла, другим — ім'я жорсткого зв'язку, що буде створений:

Зазначимо, що стандартні засоби вилучення даних за наявності жорстких зв'язків працюватимуть саме з ними, а не безпосередньо із файлами. Замість системного виклику вилучення файла використовують виклик вилучення зв'язку (який зазвичай називають unlink ( ), що вилучатиме один жорсткий зв'язок для заданого файла. Якщо після цього зв'язків у файла більше не залишається, його дані також вилучаються.

Підтримка жорстких зв'язків у Windows XP

Жорсткі зв'язки здебільшого реалізовані в UNIX-сумісних системах, їх підтримують також у системах лінії Windows XP для файлової системи NTFS . Для створення жорсткого зв'язку в цій системі необхідно використати функцію Create-HardLink ( ), ім'я зв'язку задають першим параметром, ім'я файла - другим, а третій дорівнює нулю:

CreateHardLink("myfile_hardlink.txt". "myfile.txt". 0);

Для вилучення жорстких зв'язків у Win32 API використовують функцію

DeleteFile( ):

DeleteFi1e("myfile_hardlink.txt");

Зазначимо, що для файлових систем, які не підтримують жорстких зв'язків, виклик DeleteFile( ) завжди спричиняє вилучення файла.

Жорсткі зв'язки мають певні недоліки, які обмежують їх застосування:

- не можуть бути задані для каталогів;
- усі жорсткі зв'язки одного файла завжди мають перебувати на одному й тому самому розділі жорсткого диска (в одній файловій системі);
- вилучення жорсткого зв'язку потенційно може спричинити втрати даних файла.

Символічні зв'язки

Символічний зв'язок (symbolic link) - зв'язок, фізично відокремлений від даних, на які вказує. Фактично, це спеціальний файл, що містить ім'я файла, на який вказує.

Наведемо властивості символічних зв'язків.

- Через такий зв'язок здійснюють доступ до вихідного файла.
- При вилученні зв'язку, вихідний файл не зникне.
- Якщо вихідний файл перемістити або вилучити, зв'язок розірветься, і доступ через нього стане неможливий, якщо файл потім поновити на тому самому місці, зв'язком знову можна користуватися.
- Символічні зв'язки можуть вказувати на каталоги і файли, що перебувають на інших файлових системах (на іншому розділі жорсткого диска). Наприклад, якщо створити в поточному каталозі зв'язок system-docs, що вказує на каталог /usr/doc, то перехід у каталог system-docs призведе до переходу в каталог /usr/doc.

Підтримка символічних зв'язків на рівні системних викликів.

Для задания символічного зв'язку у POSIX визначено системний виклик symlink (), параметри якого аналогічні до параметрів link ():

symlink ("myfile.txt". "myfile-symlink.txt");

Для отримання шляху до файла або каталогу, на який вказує символічний зв'язок, використовують системний виклик readlink( ).

// PATHMAX - константа, що задає максимальну довжину шляху

char filepath[PATH_MAX+l];

readlink("myfile-symlink.txt\ filepath, sizeof(filepath));

//yfilepathбуде шлях до myfile.txt

Символічні зв'язки вперше з'явилися у файлових системах UNIX, у Windows XP вони підтримуються файловою системою NTFS під назвою точок з'єднання (junction points), але засоби API для їхнього використання не визначені [87].