Лекція №7

Операційні системи

План

1. Поняття ОС. Різноманітність ОС.
2. Процеси, потоки. Пам’ять. Файлові системи.
3. Класифікація ОС за архітектурою ядра.

1. План ОС. Різноманітність ОС

Що таке операційна система? Є два рівноважливі погляди:

Погляд з боку користувача:
ОС як розширена машина

• ОС як посередник між користувачем (та його програмами) і апаратним забезпеченням.

• Надання абстракцій замість низькорівневих деталей доступу:

- файли та каталоги замість секторів і доріжок дисків;

- адресний простір замість фізичних мікросхем RAM.

• Надання стандартизованого прикладного інтерфейсу програмування (API).

• Результат: робота стає зручнішою та передбачуванішою.

Погляд з боку комп'ютера:
ОС як менеджер ресурсів

• ОС як спеціальне системне програмне забезпечення, що керує апаратними компонентами.

• Комп'ютер має обмежений обсяг ресурсів, на які одночасно претендують багато програм.

• Головні задачі ОС як менеджера ресурсів:

- справедливий та ефективний розподіл ресурсів;

- запобігання конфліктам за ресурси між процесами;

- захист даних одних програм від несанкціонованого втручання інших;

- облік використання ресурсів, балансування навантаження.

1. План ОС. Різноманітність ОС

Системні ресурси та принципи керування ними

Основні системні ресурси комп'ютера

• Центральний процесор (ЦП / CPU):

- процесорний час;

- регістри ЦП.

• Оперативна пам'ять (RAM / main memory):

- зберігає виконуваний код та дані виконуваних програм.

• Накопичувачі (storage):

- жорсткі диски (HDD), твердотільні накопичувачі (SSD) для тривалого зберігання даних.

• Пристрої введення-виведення (I/O devices):

- клавіатура, миша, дисплей, мережні інтерфейси, принтери тощо.

Два способи керування ресурсами

1. Керування у часі.

• Різні програми використовують ресурс почергово.

• Програма отримує монопольний доступ на певний квант часу, потім черга переходить до іншої.

Приклади: центральний процесор (квантування часу), принтер.

2. Керування у просторі.

• Ресурс ділиться на фізичні частини між кількома програмами одночасно.

Приклади: оперативна пам'ять (кожен процес має власний адресний простір), дисковий простір (квоти).

1. План ОС. Різноманітність ОС

Цілі функціонування та чинники еволюції ОС

Три фундаментальні цілі розробки ОС [W. Stallings]

1. Зручність:

зробити використання комп'ютера простим і комфортним для людей та розробників ПЗ.

2. Ефективність:

забезпечити якомога ефективніше використання обладнання.

3. Здатність до розвитку:

архітектура ОС має дозволяти впровадження нових функцій без руйнування наявних.

Чому операційні системи постійно змінюються?

- Розвиток апаратного забезпечення:
  - поява нових, потужніших ЦП, еволюція архітектур ЦП;
  - накопичувачі NVMe, енергонезалежна пам'ять NVDIMM;
  - поява нових відеокарт;
  - підтримка нових шин і периферійних контролерів тощо.
- Поява нових технологій та користувацьких сервісів:
  - віртуалізація, хмарні обчислення, контейнеризація;
  - мобільні та сенсорні інтерфейси, біометрія тощо.
- Виправлення помилок та уразливостей:
  - безпекові оновлення (security patches);
  - виправлення критичних помилок ядра.

1. План ОС. Різноманітність ОС

Класифікація ОС за сферою застосування (Слайд 1 з 4)

ОС мейнфреймів

Де застосовуються:

великі корпоративні центри обробки даних, банки, фондові біржі.

Ключові акценти:

• велика пропускна здатність введення-виведення;

• одночасна обробка великої кількості транзакцій.

Приклади:

IBM z/OS, OS/390, Linux для мейнфреймів

Серверні ОС

Де застосовуються:

сервери мереж, хмарні провайдери (AWS, Azure, GCP), дата-центри, студентські проєкти.

Ключові акценти:

• надання спільного доступу до файлів, баз даних, сервісів тощо;

• підвищені вимоги до безпеки та стабільності.

Приклади:

Linux (Ubuntu Server, RHEL), Windows Server, FreeBSD, Solaris тощо.

1. План ОС. Різноманітність ОС

Класифікація ОС за сферою застосування (Слайд 2 з 4)

Настільні ОС

Де застосовуються:

домашні та офісні ПК, робочі станції, ноутбуки, нетбуки, моноблоки.

Ключові акценти:

• графічний інтерфейс користувача (GUI), швидка реакція на дії користувача;

• підтримка мультимедіа, ігор, пристроїв введення-виведення різних виробників.

Приклади:

Microsoft Windows, macOS, настільні версії дистрибутивів Linux.

Мобільні ОС

Де застосовуються:

смартфони, планшети, розумні годинники, електронні рідери тощо.

Ключові акценти:

• економія заряду акумулятора;

• сенсорний інтерфейс;

• робота з бездротовими модулями (LTE, 5G, Wi-Fi, Bluetooth, GPS, NFC).

Приклади:

Android, iOS, HarmonyOS тощо.

1. План ОС. Різноманітність ОС

Класифікація ОС за сферою застосування (Слайд 3 з 4)

Вбудовані ОС (embedded OS)

Де застосовуються:

автомобільні бортові комп'ютери, пральні машини, маршрутизатори, телевізори, дрони тощо.

Ключові акценти:

• робота на специфічному апаратному забезпеченні;

• обмежений розмір пам’яті;

• висока надійність (ціна краху системи висока або неприпустимо висока).

Приклади:

Embedded Linux, OpenWrt, QNX ([к’юніск]), VxWorks.

ОС реального часу RTOS (real-time OS)

Де застосовуються:

космічні апарати, авіаційні системи керування, промислові конвеєри, медичне обладнання тощо.

Ключові акценти:

• операція має гарантовано виконатися у чітко визначений проміжок часу;

• висока надійність (ціна краху системи висока або неприпустимо висока).

Приклади:

• FreeRTOS, QNX, VxWorks, eCos.

1. План ОС. Різноманітність ОС

Класифікація ОС за сферою застосування (Слайд 3 з 4)

Операційні системи сенсорних вузлів

Де застосовуються:

мережі автономних сенсорів (IoT, моніторинг пожеж у лісах, сейсмологія, військова розвідка тощо).

Ключові акценти:

• економія електроенергії (наприклад, елемент живлення, сонячна батарея);

• дуже малий обсяг пам'яті.

Приклади:

TinyOS, Contiki.

Операційні системи смарт-карток

Де застосовуються:

банківські платіжні картки (Mastercard, Visa), SIM-картки телефонів, ID-паспорти, транспортні картки.

Ключові акценти:

• чіп не має власної батареї;

• дуже малий обсяг пам’яті;

• можлива підтримка Java-аплетів.

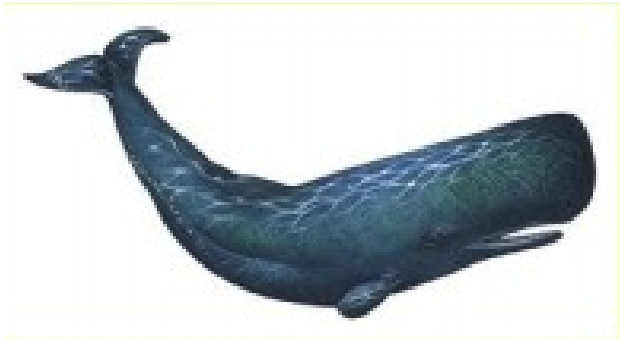
Приклади:

Java Card (JCOP), PayFlex, MULTOS.

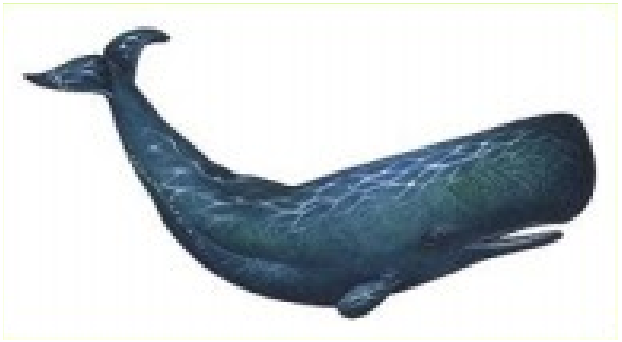
2. Процеси, потоки. Пам’ять. Файлові системи

Три «кити» операційної системи (Слайд 1 з 2)

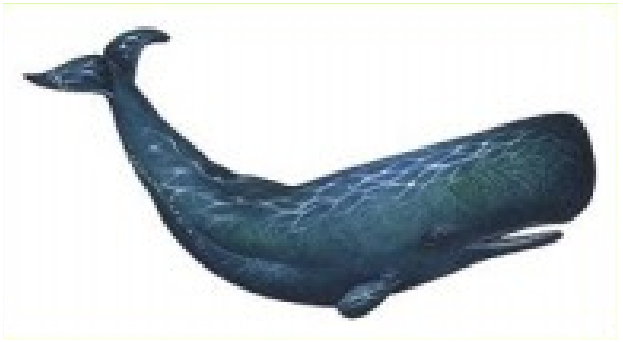
Операційна система



Процеси та потоки



Адресні простори



Файлові системи

2. Процеси, потоки. Пам’ять. Файлові системи

Три «кити» операційної системи (Слайд 2 з 2)

«Кит» #1: процеси та потоки

Спеціалізація «кита»:
ЦП, обчислення

• **Процес** — абстракція, яка представляє програму під час виконання.

• **Потік** — код програми, виконуваний на ЦП.

• Абстрагують процесор, надаючи кожній програмі ілюзію власного віртуального процесора 🦄

• Реалізують *багатозадачність* (multitasking).

«Кит» #2: адресні простори

Спеціалізація «кита»:
пам’ять, ізоляція

• **Адресний простір** — множина адрес у пам'яті, доступних процесу.

• Абстрагує фізичну оперативну пам'ять (RAM).

• Забезпечує захист та ізоляцію: процес не може перезаписати чужі дані або код.

• Реалізує *віртуальну пам'ять*.

«Кит» №3: файлові системи

Спеціалізація «кита»:
довгострокове зберігання даних

• **Файл** — називають впорядкований набір даних, що зберігається у комп'ютерній системі під спільним ім'ям.

• Абстрагує дискові пристрої (HDD, SSD, flash).

• Забезпечує незалежність від фізичної будови носіїв та спільне використання даних.

2. Процеси, потоки. Пам’ять. Файлові системи

Процес як базова абстракція

Процес *vs* програма

? *Чим процес відрізняється від програми?*

• Програма — статична сутність (файли на диску, які містять програмний код, дані, налаштування).

• Процес — динамічна сутність (програма під час виконання).

• Один файл програми може породити декілька дочірніх процесів (наприклад, кілька вікон текстового редактора).

• Процес споживає ресурси: час ЦП, регістри ЦП, оперативна та дискова пам'ять, відкриті файли тощо.

Блок керування процесом (Process Control Block, PCB)

Сутність у пам'яті для керування процесом (наприклад, структура даних, об’єкти).

• Ідентифікація: PID (Process ID), PPID (Parent PID), SID/UID/GID власника.

• Стан процесу: виконання, готовність, очікування, створення, завершення (типові).

• Інформація стану процесу (контекст ЦП): збережені значення лічильника команд (PC) та регістрів під час переривань.

• Керування пам'яттю: вказівники на таблиці сторінок.

• Інформація для обліку: використаний час ЦП, пріоритети планування.

• Стан введення-виведення: відкриті файли, пристрої.

2. Процеси, потоки. Пам’ять. Файлові системи

П’ятистанова модель станів процесів*

П'ять базових станів процесу

Створення (New):

процес ініціалізується, ОС створює PCB та виділяє ресурси.

Готовність (Ready):

процес готовий до роботи, чекає, поки планувальник вибере його для виконання на ЦП.

Виконання (Running):

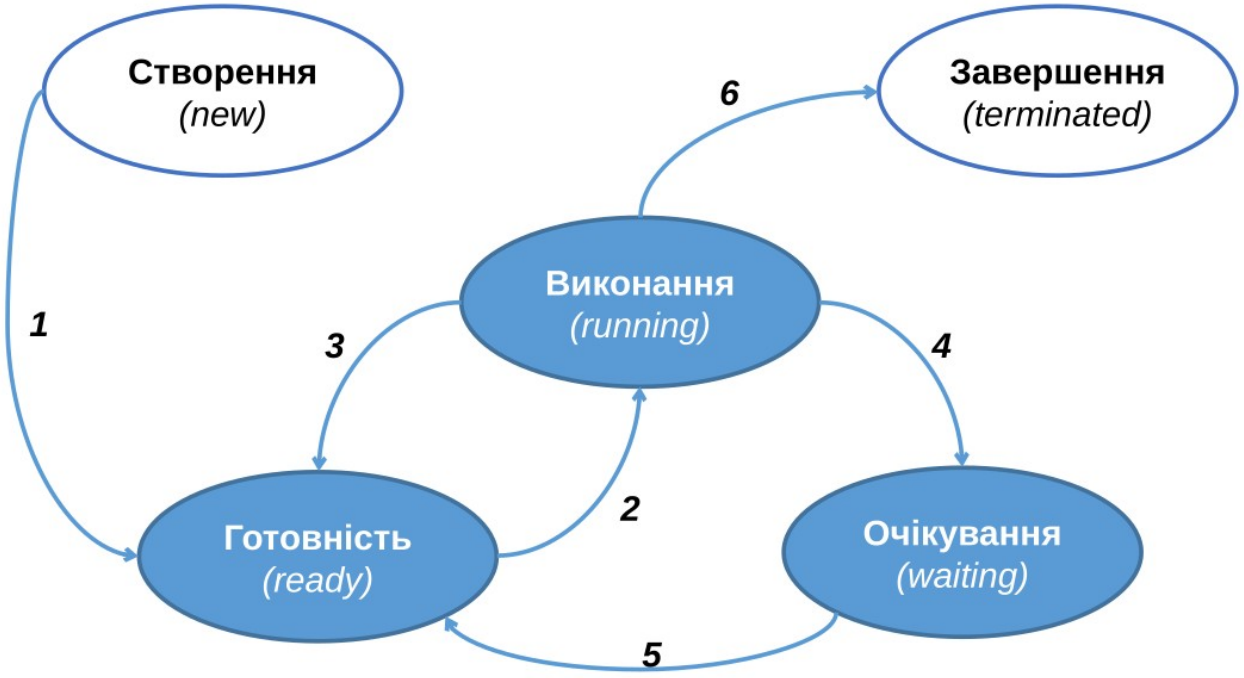
інструкції процесу безпосередньо виконуються на ЦП.

Очікування (Waiting / Blocked):

процес чекає на подію (наприклад, завершення I/O, надходження даних).

Завершення (Terminated):

процес виконав роботу, звільняє пам'ять.



** Може застосовуватися і до потоків (наприклад, лежить в основі моделі станів процесів у Windows)*

2. Процеси, потоки. Пам’ять. Файлові системи

Перемикання контексту (context switching)

Що таке контекст процесу і навіщо його зберігати?

? Якщо процес А переривається заради процесу Б, як не втратити проміжні результати процесу А?

• **Перемикання контексту** (context switching) — передача керування від одного процесу до іншого зі збереженням даних пов'язаних з виконанням процесу на ЦП.

До збереженого контексту входить, зокрема:

• вміст регістрів ЦП, зокрема лічильника команд (яку інструкцію виконувати наступною);

• відомості про розподіл пам'яті, зокрема вказівник вершини стеку.

Ціна перемикання контексту

Часте перемикання контексту уповільнює систему 😵‍💫

Тому планувальник має балансувати між:

• швидким відгуком системи (більше перемикань контексту);

• вищою продуктивністю виконання процесів (менше перемикань контексту).

Під час планування також враховуються умови, в яких працює ОС, зокрема що важливіше:

• швидкий відгук системи? (інтерактивні системи)

• швидке виконання окремих процесів (пакетна обробка);

• дотримання дедлайнів (системи реального часу).

2. Процеси, потоки. Пам’ять. Файлові системи

Потоки (threads)

Поняття потоку

? Чим потік відрізняється від процесу?

• **Потік (потік виконання, нитка, thread)** — це код програми, виконуваний на ЦП.

• **Процес (узагальнене означення)** — один або декілька потоків виконання, а також сукупність пов’язаних з ними ресурсів.

Що у кожного потоку ***своє***:

• лічильник команд;

• значення регістрів ЦП;

• стек (дані викликів підпрограм);

• стан (але процес також може мати стан).

Що у потоків одного процесу ***спільне***:

• адресний простір процесу:

- скомпільований програмний код;

- глобальні змінні;

- динамічно виділена пам'ять (купа / heap).

• системні ресурси:

- відкриті файли та сокети;

- права доступу;

- системні сигнали тощо.

Процесам для обміну даними потрібні засоби міжпроцесової взаємодії (бо адресні простори процесів захищені).

Потоки мають спільний адресний простір, тому можуть обмінюватися даними швидше, ніж процеси.

2. Процеси, потоки. Пам’ять. Файлові системи

Багатопотоковість: переваги, моделі реалізації

Переваги багатопотоковості

? *Чому сучасні програми проєктуються багатопотоковими?*

• Оперативність реакції програми (app responsiveness):

окремий потік, відповідальний за взаємодію з користувачем.

• Економія часу та пам'яті:

- створення потоку швидше за створення процесу;

- перемикання контексту між потоками теж швидше.

• Масштабованість:

- можливість залучити всі фізичні ядра багатоядерного ЦП для паралельної роботи.

Основні моделі багатопотоковості

1. Модель Many-to-one («Багато до одного»)

• потоки рівня користувача, створюються без системних викликів;

• якщо один потік здійснює I/O, то блокується весь процес.

Приклади: рання Solaris.

2. Модель One-to-one («Один до одного»)

• потоки рівня ядра;

• справжній апаратний паралелізм на ядрах ЦП.

Приклади: Linux Pthreads, Windows Threads.

3. Модель Many-to-one («Багато до багатьох»)

• *M* потоків користувача транслюються в *N* потоків ядра.

Приклади: на рівні бібліотек.

2. Процеси, потоки. Пам’ять. Файлові системи

Адресний простір процесу: організація пам’яті

Що таке захищений адресний простір

• **Адресний простір** — це набір адрес пам'яті, до яких процес має доступ.

• Адресний простір процесу є *захищеним*:

- кожен процес ізольований у власному адресний просторі;

- за замовчуванням інші процеси не мають права читати чи записувати в цю ділянку;

- спроба вийти за межі породжує помилку.

• Якщо потрібен обмін даними — залучаються механізми міжпроцесова взаємодія (IPC).

Як влаштовано адресний простір

1. Програмний код

скомпільовані машинно-орієнтовані інструкції програми (read-only).

2. Глобальні дані

ініціалізовані та неініціалізовані глобальні змінні.

3. Динамічна пам'ять (купа / heap)

• пам'ять, яку програма просить під час роботи (malloc / new);

• купа динамічно ***росте вгоду*** (у бік вищих адрес)

4. Стек

• зберігає параметри виклику функцій, адреси повернення, локальні змінні;

• стек динамічно ***росте вниз*** (у бік нижчих адрес)

2. Процеси, потоки. Пам’ять. Файлові системи

Фізичні та віртуальні адреси

Чому не можна працювати лише з фізичними адресами?

? *Що буде, якщо програми писатимуть безпосередньо у фізичну RAM?*

• Проблема прив'язки:

програміст не знає, які саме фізичні адреси будуть вільні в комп'ютері користувача під час запуску.

• Проблема захисту та надійності:

помилка або зловмисна дія в одній програмі перезапише пам'ять ядра чи сусіднього процесу.

• Проблема розміру:

процес не може виконуватися, якщо йому потрібно більше пам'яті, ніж фізично встановлено RAM.

• Проблема зовнішньої фрагментації.

Поняття віртуальної пам’яті

• Поверх фізичної пам'яті створюється додатковий рівень абстракції — віртуальні (логічні) адреси.

• Кожен процес вважає, що володіє власним адресним простором, що починається з нульової адреси.

• Апаратний блок керування пам'яттю (MMU, memory management unit) автоматично перетворює віртуальну адресу у фізичну.

• Реальні комірки пам'яті процесу можуть бути розкидані по різних ділянках RAM або тимчасово вивантажені на диск.

2. Процеси, потоки. Пам’ять. Файлові системи

Сторінкова організація пам'яті (paging)

Базові ідеї сторінкової пам’яті

• Сторінка:

віртуальний адресний простір процесу ділиться на блоки однакового фіксованого розміру — сторінки.

• Фрейм:

фізична оперативна пам'ять ділиться на блоки точно такого ж розміру — фрейми.

• Принципи:

- розмір сторінки — 2^n;

- будь-яка віртуальна сторінка процесу може бути розміщена в будь-якому вільному фізичному фреймі;

- сусідні сторінки не зобов'язані лежати у сусідніх фреймах.

Перетворення адреси

Структура віртуальної адреси:

• номер сторінки;

• зсув всередині сторінки.

Механізм перетворення адреси:

1) номер сторінки використовується як індекс у таблиці сторінок;

2) з таблиці зчитується номер відповідного фрейму;

3) перевірка наявності прав доступу;

4) *фізична адреса = номер сторінки* * 2^n + *зсув*

Для прискорення перетворення: кеш трансляції (TLB) зберігає часто використовувані адреси.

2. Процеси, потоки. Пам’ять. Файлові системи

Підкачування та алгоритми заміщення сторінок

Підкачування (swapping)

• Більшість сторінок процесу використовуються рідко.

• Неактивні сторінки вивантажуються на диск у файл або розділ підкачування.

Сторінкова відмова (page fault):

1. Процес звертається до віртуальної сторінки, якої немає в RAM.

2. MMU сигналізує апаратне переривання (page fault).

3. ОС призупиняє процес, зчитує сторінку з диска у вільний фрейм RAM, оновлює таблицю сторінок.

4. Процес продовжує роботу, не знаючи про затримку 🦄

Алгоритми заміщення сторінок

Яку сторінку витіснити на диск? (приклади алгоритмів)

• FIFO (First-In, First-Out):

витісняє найстарішу сторінку; простий, але неефективний (ризик витіснити активно використовувану сторінку).

• LRU (Least recently used):

витісняє сторінку, до якої не зверталися найдовше; ефективний, але складно реалізувати, існують наближення.

• Clock («Годинник»):

наближення до LRU за допомогою біта використання сторінки (reference bit).

• Оптимальний алгоритм:

витісняє сторінку, що знадобиться пізніше за всіх (неможливо реалізувати, використовується як еталон для оцінювання ефективності інших алгоритмів).

2. Процеси, потоки. Пам’ять. Файлові системи

Файлові системи: призначення та рівні організації

Базові функції файлової системи

Файлова система виконує дві головні ролі:

• організовує довготермінове зберігання даних на носіях (HDD, SSD, flash);

• надає користувачам та програмам доступ до даних через зручну абстракцію — файл.

Файлова система об'єднує в собі:

• самі файли з їхнім вмістом;

• службові структури даних (метадані, каталоги, списки вільних блоків тощо);

• системні програмні засоби керування файлами (file management system).

Два рівні організації файлової системи

1. Фізичний рівень

• Має справу з реальними секторами та блоками накопичувача.

• Керує розміщенням блоків, плануванням черги дискових запитів.

2. Логічний рівень

• Має справу з файлами, папками, атрибутами та правами доступу.

• Зв'язок між рівнями: саме службові структури ФС перетворюють розрізнені фізичні блоки на диску в єдиний логічний файл.

2. Процеси, потоки. Пам’ять. Файлові системи

Поняття файлу, атрибуту, каталогу

Файл та його очікуванні властивості

Файл — це впорядкований набір даних, що зберігається в комп'ютерній системі під спільним ім'ям.

Три очікувані характеристики файлів [W. Stallings]:

• Довготермінове існування^

інформація зберігається на енергонезалежному носії після вимкнення живлення чи перезавантаження.

• Спільне використання:

доступ багатьох процесів на основі прав доступу та блокувань.

• Структурованість:

внутрішня організація байтів/записів та включення в ієрархічні структури (дискові томи, каталоги тощо).

Каталог: дві рівноважливі точки зору

Як дивитися на каталог (директорію, папку)?

1. Позиція користувача:

каталог — це логічна група файлів для зручної організації та навігації.

2. Позиція внутрішньої будови ОС:

• каталог — це особливий файл, що містить список записів про інші файли та підкаталоги.

2. Процеси, потоки. Пам’ять. Файлові системи

Іменування, журналювання та різноманітність файлових систем

Іменування файлів

• Абсолютне ім'я (повний шлях від кореня):

- Linux: `/home/user/docs/file.txt` (єдине дерево від /);

- Windows: `C:\Users\user\docs\file.txt` (корені для дисків).

• Відносне ім'я: відлік ведеться від поточного (робочого) каталогу.

Віртуальна файлова система (VFS):

• Абстрактний рівень, що уніфікує доступ до різних ФС.

• Завдяки VFS системні виклики читання та запису однаково працюють з ext4, NTFS, NFS, ISO9060, USB-флешкою FAT32 тощо.

Журналювання (journaling)

Механізм журналювання транзакцій

• Зміни у файловій системі фіксуються в спеціальному журналі перед фактичним записом на диск.

• Якщо під час запису стався збій, транзакція:

- або завершується,

- або скасовується (відкочується);

- але НЕ виконується частково (це б порушило цілісність ФС).

Приклади файлових систем

• Linux: ext4 (журнальована), Btrfs, XFS.

• Windows: NTFS (журнальована), FAT32, exFAT.

• Apple: APFS, HFS+.

3. Класифікація ОС за архітектурою ядра

Режим ядра *vs* режим користувача

Режим ядра (kernel mode, привілейований режим)

? *Що дозволено в режимі ядра?*

• Процес має доступ до ВСІХ команд, інструкцій та регістрів процесора.

• Необмежений доступ до будь-якої фізичної пам'яті та портів пристроїв.

• У режимі ядра працюють критичні компоненти ОС (ядро, драйвери).

• Наслідки збою:

будь-який критичний збій у режимі ядра призводить до краху всієї системи (kernel panic у Linux, blue screen of beath / BSOD у Windows).

Режим користувача (user mode)

? *Що дозволено в режимі користувача?*

• Процес має доступ лише до обмеженої підмножини інструкцій.

• Відсутній прямий доступ до апаратури, портів введення-виведення та чужої пам'яті.

• У режимі користувача працюють усі прикладні програми, утиліти, інтерфейси користувача (GUI / CLI).

• Наслідки збою:

помилка у програмі впливає на цей конкретний процес, а операційна система продовжує працювати.

3. Класифікація ОС за архітектурою ядра

Системні виклики та API

Системний виклик (system call)

? *Як програмі з режиму користувача зберегти файл чи вивести текст?*

• Програма виконує системний виклик (механізм звернення до ядра ОС за «послугою»).

Послідовність виконання системного виклику:

1) програма готує параметри виклику і викликає спеціальну апаратну інструкцію;

2) ЦП перемикається з режиму користувача в режим ядра;

3) код ядра перевіряє права, виконує дію з обладнанням та повертає результат;

4) ЦП перемикається назад у режим користувача.

API (Application Programming Interface)

Для залучення системного виклику використовуються API.

• POSIX API (Unix-подібні системи: Linux, macOS, BSD):

- відкриття файлу — `open()`;
- читання файлу — `read()`;
- створення процесу — `fork()`;
- очікування — `wait()`.

• Windows API / Win32 API (ОС Windows):

- створення файлу — `CreateFile()`;
- читання файлу — `ReadFile()`;
- створення процесу — `CreateProcess()`;
- очікування — `WaitForSingleObject()`.

• Важливо: перемикання режимів займає процесорний час, тому часті системні виклики знижують швидкодію.

3. Класифікація ОС за архітектурою ядра

Монолітна архітектура ядра (monolithic kernel)

Принципи побудови монолітного ядра

Найпоширеніший історичний та практичний спосіб реалізації ядра ОС.

• Усі базові компоненти ОС працюють в режимі ядра, зокрема:

- планувальник процесів і потоків;
- підсистема керування пам’яттю;
- драйвери апаратних пристроїв;
- мережні протоколи;
- файлові системи.

• Взаємодія підсистем всередині ядра відбувається через швидкі звичайні виклики функцій у пам'яті (без системних викликів).

Переваги та недоліки монолітного ядра

Переваги монолітних ядер:

+ швидкодія (немає накладних витрат на передачу повідомлень між компонентами ядра).

Недоліки класичних монолітних ядер:

- зниження надійності (помилка в будь-якому сторонньому драйвері може призвести до краху ОС).

Модульні ядра

• Сучасні монолітні ядра мають модульну структуру.

• Ядро Linux, FreeBSD не компілюється як єдине ціле. Драйвери та протоколи є динамічними модулями, що завантажуються за потреби без перезавантаження ОС.

3. Класифікація ОС за архітектурою ядра

Мікроядерна архітектура (Microkernel)

Принципи побудови мікроядра

• У ядрі залишається мінімум функцій, наприклад:

- диспетчеризація та планування низькорівневих потоків;

- базова обробка апаратних переривань;

- механізм міжпроцесової взаємодії.

• Решта компонентів винесені у простір користувача як окремі сервери, наприклад:

- файлова система;
- мережний стек;
- драйвери тощо.

Переваги та недоліки мікроядра

Переваги мікроядерної архітектури:

+ підвищена надійність (помилка драйвера не зачіпає ядро).

Недоліки класичної мікроядерної архітектури:

- зниження продуктивності через (накладні витрати на перемикання контексту та міжпроцесову взаємодію).

Приклади:

QNX (автомобілебудування, авіоніка), MINIX 3, seL4, L4, Integrity, PikeOS.

3. Класифікація ОС за архітектурою ядра

Гібридні ядра. Багаторівневі ядра

Гібридні ядра (hybrid kernels)

Поєднання сильних сторін моноліту та мікроядра:

• за структурою система проєктується як мікроядро (модульність).

• заради продуктивності більшість критичних серверів (драйвери, графіка, файлова система) виконуються в режимі ядра.

Приклади:

• Windows NT (Windows 10/11, Windows Server 20xx):

мікроядро NT з виконавчою підсистемою та графічним інтерфейсом у режимі ядра — велиике мікроядро

• macOS та iOS (ядро XNU):

поєднання мікроядра Mach та монолітних компонентів FreeBSD.

Багаторівневі ОС (layered systems)

Історична архітектура (наприклад, ОС THE Едсгера Дейкстри)

• Замість двох режимів — ієрархія з багатьох рівнів захисту (від 0 до 5):

- рівень 0 — апаратне забезпечення;
- рівень 1 — планування процесора;
- рівень 2 — керування пам'яттю;
- рівень 3 — зв'язок консолі й оператора;
- рівень 4 — введення-виведення;
- рівень 5 — інтерфейс користувача.

• Кожен рівень спирається на функції нижчого.

• Плюс — модульність; мінус — значні витрати на проходження рівнів.

Рекомендовані джерела

Підручники:

• Silberschatz A., Galvin P., Gagne G. Operating System Concepts, 10th ed. — Wiley, 2018.

• Stallings W. Operating Systems: Internals and Design Principles, 9th ed. — Pearson, 2017.

• Tanenbaum A. S., Bos H. Modern Operating Systems, 4th ed. — Pearson, 2014.

• Шеховцов В. А. Операційні системи. — К.: BHV, 2005.

Навчальний посібник:

• Головня О. С. Основи операційних систем: навч. посібник. — Житомир: Державний університет «Житомирська політехніка», 2023. — 126 с.