МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ДНІПРОВСЬКА ПОЛІТЕХНІКА”

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра інформаційних технологій та
комп’ютерної інженерії

Гаркуша І.М.

Конспект лекцій
з дисципліни “Операційні системи”
для студентів галузі знань 12 “Інформаційні технології”
спеціальності 126 “Інформаційні системи та технології”

Дніпро
НТУ “ДП”
2020

УДК 004.451
Г20

Гаркуша І.М. Конспект лекцій з дисципліни “Операційні системи” для студентів галузі знань 12 “Інформаційні технології” спеціальності 126 “Інформаційні системи та технології”. – Д.: НТУ «ДП», 2020. – 73 с.

В конспекті лекцій з дисципліни “Операційні системи” для студентів спеціальності 126 “Інформаційні системи та технології” приведена базова частина курсу, яка викладається за новим навчальним планом у другому семестрі першого курсу.

Основна увага у базовій частині курсу приділяється питанням архітектур найвідоміших операційних систем, а також питанням файлових систем та певним командам.

Електронна версія подана у редакційному виправленні та доповнені автора у 2024 році.

Погоджено рішенням науково-методичної комісії спеціальності 126 Інформаційні системи та технології (протокол № 7 від 27.08.2020).

ЗМІСТ

ВСТУП

Метою дисципліни “Операційні системи” для студентів спеціальності 126 “Інформаційні системи та технології” є формування компетентностей щодо побудови, функціонування та основ роботи в сучасних операційних системах MS Windows, GNU/Linux-сумісних та macOS. В лабораторній частини курсу розглядаються створення сценаріїв обробки даних на базі скриптів під командні процесори Bash та PowerShell, опанування використання віртуальної машини.

Курс умовно розбитий на дві частини – базову, лекційна частина якої представлена цією навчально-методичною розробкою, та додаткову, в якій викладаються елементи системного програмування (створення та управління процесами, потоками, нитками) для різних операційних систем.

При складанні лекцій базового курсу використані різноманітні літературні джерела інформації – як книги відомих світових авторів, так і перевірені довідникові матеріали глобальної мережі Internet, довідникові матеріали компаній-розробників операційних систем.

Основними дисциплінарними результатами навчання, після завершення базової частини лекційного курсу “Операційні системи”, є:

– вміння дати класифікацію операційних систем за різними критеріями;

– знання загальної архітектури сучасних операційних систем;

– володіння певними знаннями по роботі з файловими системами та вміння пояснити головні відмінності провідних файлових систем;

– знання щодо побудови та функціонування сучасних операційних середовищ класу MS Windows, GNU/Linux, macOS;

– володіння певними консольними командами Bash в GNU/Linux-сумісних операційних середовищах.

Лекція 1. Введення до операційних систем

Комп'ютерні програми (програмне забезпечення, ПЗ) ділять на три категорії:

1. Прикладні програми, які безпосередньо забезпечують виконання необхідних користувачам робіт: редагування текстів, малювання картинок, обробка інформаційних масивів і т.д.

2. Системні програми, виконують різні допоміжні функції, наприклад, створення копій інформації, видачу довідкової інформації про комп'ютер, перевірку працездатності пристроїв комп'ютера і т.д.

3. Інструментальні системи (системи програмування), що забезпечують створення нових програм для комп'ютера.

Серед всіх системних програм особливе місце займає *операційна система* (ОС) – програма (частіше набір програм), яка завантажується при включенні комп'ютера. Вона ізолює апаратне забезпечення комп'ютера від прикладних програм користувачів, які взаємодіють з комп'ютером через інтерфейси ОС. Таким чином, ОС представляється користувачеві у вигляді розширеної або віртуальної машини, яку легше програмувати і з якою легше працювати, ніж безпосередньо з апаратурою, що становить реальну машину.

Основні функції ОС

1. Приймання від користувача завдань, або команд, сформульованих на відповідній мові, і їх обробка. Команди які пов'язані із запуском (припиненням, зупинкою) програм, з операціями над файлами (отримати перелік файлів в поточному каталозі, створити, перейменувати, скопіювати, перемістити файл і т.п.) і інші команди.

2. Завантаження в оперативну пам'ять програм, які підлягають виконанню.

3. Розподіл пам'яті, організація віртуальної пам'яті.

4. Запуск програми (передача їй управління, в результаті чого процесор виконує програму).

5. Ідентифікація всіх програм та даних.

6. Прийом і виконання різних запитів від програм, які виконуються. Звернення здійснюється за відповідними правилами, які і визначають інтерфейс прикладного програмування (*Application Programming Interface, API*) цієї ОС.

7. Обслуговування всіх операцій введення-виведення (Input/Output, I/O).

8. Забезпечення роботи систем управління файлами (СУФ) та/або систем управління базами даних (СУБД), що дозволяє різко збільшити ефективність всього ПЗ.

9. Забезпечення режиму мультипрограмування, тобто організація паралельного виконання двох або більш програм на одному процесорі, що створює видимість їх одночасного виконання.

10. Планування та диспетчеризація задач відповідно до заданих стратегії та дисциплін обслуговування.

11. Організація механізмів обміну повідомленнями і даними між програмами, які виконуються.

12. Для мережевих ОС характерною є функція забезпечення взаємодії пов'язаних між собою комп'ютерів.

13. Захист однієї програми від впливу іншої, забезпечення збереження даних, захист самої ОС від програм, які виконуються під її керівництвом.

14. Аутентифікація та авторизація користувачів. *Аутентифікація* – процедура перевірки імені користувача і його пароля на відповідність тим значенням, які зберігаються в його профілі. *Авторизація* означає, що відповідно до облікового запису користувача, який пройшов аутентифікацію, йому (і всім запитам, які він здійснює в ОС від свого імені), призначаються певні права (привілеї), що визначають дії, які він може виконувати на комп'ютері.

15. Задоволення жорстких обмежень на час відповіді в режимі реального часу (характерно для ОС реального часу).

16. Забезпечення роботи систем програмування.

17. Надання функцій відновлення системи після збою.

До найбільш важливих функцій можна віднести наступні:

1. Реалізація абстрактної машини, що дає можливість працювати кінцевому користувачеві.

2. Раціональний розподіл обчислювальних ресурсів (зовнішньої та внутрішньої пам'яті, процесорного часу).

3. Захист інформації від несанкціонованого доступу.

4. Організація функціонування та взаємодії обчислювальних процесів.

Типи ОС

1. ОС персональних електронно-обчислювальних машин (ОС ПЕОМ): MS-DOS, PC-DOS, DR-DOS і т.д., Windows 95/98/NT/2000, OS/2, Mac OS X. З розвитком мереж сюди можна внести й ОС робочих станцій (різновиди Unix\ Linux). Для даного класу ОС найважливішою є перша функція.

2. ОС управління виробничими процесами (ОС УПП). Головна функція 4-а (забезпечення синхронізації внутрішніх обчислювальних процесів і зовнішніх процесів виробництва). Приклад: FreeRTOS, QNX – Unix-подібна ОС. Оскільки Unix – це ОС яка не є системою реального часу, то у QNX ядро відрізняється від Unix.

3. ОС локальних обчислювальних мереж (ОС ЛВС). Приклади: MS Windows NT/2000, Novell NetWare, різновиди Unix\Linux. Головні функції – 2-а та 3-я.

4. ОС регіональних та глобальних ВС. Приклади: MS Windows NT Server/2000 Server, Free BSD, HPUX, AIX та ін. Головні функції – 2-а та 3-я.

5. ОС супер ЕОМ, мейнфреймів. Приклад: MVS (система віртуальних машин). Для цих ОС важливі в тій чи іншій мірі всі функції.

6. ОС смартфонів та комунікаторів. Приклад: Windows Mobile, Symbian OS, Palm OS, Android.

7. ОС нетбуків. Приклади: Ubuntu Netbook Remix, gOS, Apple iOS.

8. ОС для Internet of Thengs – інтернету речей (ОС IoT). Приклади: FreeRTOS, Windows 10 IoT, TinyOS, ARM Mbed OS.

Лекція 2. Класифікації ОС

ОС можуть відрізнятися особливостями реалізації внутрішніх алгоритмів керування основними ресурсами комп'ютера (процесорами, пам'яттю, пристроями), особливостями використаних методів проектування, типами апаратних платформ, областями використання та іншими властивостями. Нижче наведено класифікацію ОС по декількох найбільш основних ознаках.

Залежно від особливостей використовуваного алгоритму управління процесором, ОС ділять на:

1. Багатозадачні та однозадачні.
2. Багатокористувацькі та однокористувацькі.
3. З підтримкою багатонитевості та без її підтримки.
4. З підтримкою багатопроцесорності та ті що працюють з однопроцесорними архітектурами.

Багатозадачні ОС відрізняються від однозадачних наявністю функцій управління поділом ресурсів, які спільно використовуються, таких як процесор, оперативна пам'ять, файли та зовнішні пристрої.

Головною відмінністю багатокористувацьких систем від систем, що підтримують тільки одного користувача, є наявність засобів захисту інформації кожного користувача від несанкціонованого доступу інших користувачів.

Не кожна багатозадачна ОС є багатокористувацькою, і не кожна однокористувацька ОС є однозадачною.

Найважливішим ресурсом є процесорний час. Спосіб розподілу процесорного часу між декількома одночасно існуючими в системі процесами або нитками багато в чому визначає специфіку ОС. Багатозадачність забезпечує можливість паралельної (або псевдопаралельної) обробки декількох процесів. Серед безлічі існуючих варіантів реалізації багатозадачності можна виділити дві групи алгоритмів:

– невитісняюча багатозадачність (NetWare, Windows 3.x);

– витісняюча (витискальна) багатозадачність (Windows NT, OS/2, Unix, Mac OS).

Основною відмінністю між цими варіантами багатозадачності є ступінь централізації механізму планування процесів.

Невитісняюча багатозадачність (Non-preemptive multitasking) – це спосіб планування процесів (потоків), при якому активний процес (потік) виконується до того часу, доки він сам, за власною ініціативою, не віддасть керування

планувальнику ОС для того, щоб той вибрав з черги інший, готовий до виконання процес (потік).

Витісняюча багатозадачність (Preemptive multitasking) – це вид багатозадачності, при якому ОС може тимчасово перервати поточний процес (потік) без будь-якої допомоги з його боку. Завдяки цьому, програми, які довго експлуатують процесор (“підвислі”), як правило, не порушують роботу ОС.

Справжня багатозадачність ОС можлива тільки в багатопроцесорних, або кількаядерних системах, або в розподілених обчислювальних системах.

Особливістю багатониткової ОС є розподіл процесорного часу не між завданнями (процесами), а між їх окремими гілками (нитками).

Важливою властивістю сучасних ОС є відсутність або наявність в них засобів підтримки багатопроцесорної обробки – багатопроцесування. Воно призводить до ускладнення всіх алгоритмів керування ресурсами.

В системі з багатопроцесорною архітектурою багатопроцесорні ОС можуть класифікуватися за способом організації обчислювального процесу на:

– асиметричні ОС;

– симетричні ОС.

Асиметрична ОС цілком виконується тільки на одному з процесорів системи, розподіляючи прикладні завдання по іншим процесорам.

Симетрична ОС повністю децентралізована та використовує весь пул процесорів, поділяючи їх між системними і прикладними завданнями.

В основі новітніх версій сучасних ОС лежить архітектура симетричної багатопроцесорної обробки (Symmetric Multiprocessing, SMP), яка дозволяє значно скоротити час обробки поставлених завдань.

Багатозадачні ОС підрозділяються на три типи відповідно до критеріїв ефективності, які були використані при їх розробці:

1. Системи пакетної обробки (наприклад, ОС ЕС).
2. Системи поділу часу (наприклад: Unix, VMS).
3. Системи реального часу (наприклад: QNX, RT/11).

Системи пакетної обробки призначалися для вирішення завдань, в основному, обчислювального характеру, які не потребують швидкого отримання результатів. Головною метою та критерієм ефективності систем пакетної обробки є максимальна пропускна здатність, тобто рішення максимального числа завдань в одиницю часу. Для досягнення цієї мети в таких системах використовується наступна схема функціонування.

На початку роботи формується пакет завдань. Кожне завдання містить вимоги до системних ресурсів. З цього пакету завдань формується мультипрограмна суміш, тобто безліч завдань, які будуть одночасно

виконуватися. Для одночасного виконання вибираються завдання, які відрізняються вимогами до ресурсів, так, щоб забезпечувалася збалансоване завантаження всіх пристроїв обчислювальної машини.

Наприклад, в мультипрограмній суміші бажано одночасна присутність обчислювальних задач та задач з інтенсивним введенням-виведенням (I/O). Таким чином, вибір нового завдання з пакету завдань залежить від внутрішньої ситуації, що складається в системі, тобто вибирається "вигідне" завдання.

Отже, в таких ОС неможливо гарантувати виконання того чи іншого завдання протягом певного періоду часу. У системах пакетної обробки переключення процесора з виконання одного завдання на виконання іншого відбувається тільки в разі, якщо активна задача сама відмовляється від процесора, наприклад, через необхідність виконати операцію I/O. Тому одна задача може надовго зайняти процесор, що унеможливлює виконання інтерактивних завдань.

Недолік систем пакетної обробки – низька ефективність роботи користувача.

Системи поділу часу покликані виправити основний недолік систем пакетної обробки – ізоляцію користувача-програміста від процесу виконання його завдань. Кожному користувачеві системи поділу часу надається термінал, з якого він може вести діалог зі своєю програмою. Так як в системах поділу часу кожній задачі виділяється тільки квант процесорного часу, жодна задача не займає процесор надовго, і час відповіді виявляється прийнятним. Якщо квант обраний досить невеликим, то у всіх користувачів, що одночасно працюють на одній і тій же машині, складається враження, що кожен з них одноосібно використовує машину. Такі системи мають меншу пропускну спроможність, ніж системи пакетної обробки, так як на виконання приймається кожна запущена користувачем задача, а не та, яка "вигідна" системі, і, крім того, є накладні витрати обчислювальної потужності на більш часте переключення процесора з задачі на задачу.

Критерієм ефективності систем поділу часу є не максимальна пропускна здатність, а зручність та ефективність роботи користувача.

Системи реального часу застосовуються для керування різними технічними об'єктами, такими, наприклад, як верстат, супутник, наукова експериментальна установка або технологічними процесами, такими, як гальванічна лінія, доменний процес і т.п. У всіх цих випадках існує гранично допустимий час, протягом якого повинна бути виконана та чи інша програма, що керує об'єктом, в іншому випадку може статися аварія.

Критерієм ефективності для таких систем є їх здатність витримувати заздалегідь задані інтервали часу між запуском програми й одержанням результату (керувального впливу). Цей час називається часом реакції системи, а відповідна властивість системи – реактивністю.

Для цих систем мультипрограмна суміш являє собою фіксований набір заздалегідь розроблених програм, а вибір програми на виконання здійснюється виходячи з поточного стану об'єкта або відповідно до розкладу планових робіт.

Деякі ОС можуть поєднувати в собі властивості систем різних типів, наприклад, частина завдань може виконуватися в режимі пакетної обробки, а частина – в режимі реального часу або в режимі поділу часу. У таких випадках режим пакетної обробки часто називають *фоновим режимом*.

Лекція 3. Архітектура ОС

Найбільш загальним підходом до структуризації ОС є поділ усіх її модулів на дві групи[1]:

1. ядро – модулі, що виконують основні функції ОС;
2. модулі, що виконують допоміжні функції ОС.

Обчислювальну систему, що працює під управлінням ОС на основі ядра, можна розглядати як систему, що складається з трьох ієрархічно розташованих шарів: апаратура, ядро та утиліти з системними програмами та бібліотеками (рис. 3.1).

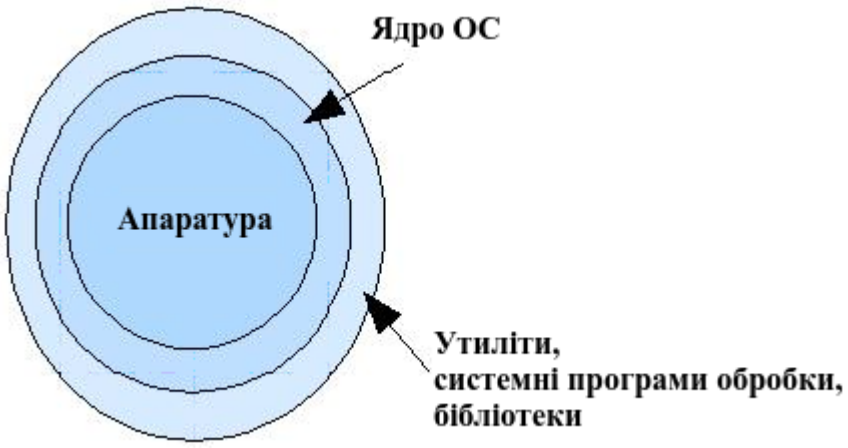


Рис. 3.1. Тришарова схема обчислювальної системи

Кожен шар обслуговує розміщений вище шар, виконуючи для нього певний набір функцій, які утворюють міжшаровий інтерфейс.

Модулі ядра виконують такі базові функції ОС, як управління процесами, пам'яттю, пристроями I/O і т.п.

До складу ядра входять функції, які вирішують внутрішньосистемні задачі організації обчислювального процесу, такі як перемикання контекстів виконання, завантаження/вивантаження сторінок пам'яті, обробка переривань. Ці функції не доступні для програм користувачів.

Інший клас функцій ядра служить для підтримки програм, створюючи для них так зване прикладне програмне середовище. Додатки можуть звертатися до ядра з запитами – системними викликами – для виконання тих чи інших дій, наприклад, для відкриття та читання файлу, виведення графічної інформації на монітор, отримання системного часу і т.п. Функції ядра, які можуть викликатися

1 В лекції використані матеріали з різних підручників, а також деякі рисунки з вільної енциклопедії Wikipedia.

програмами, утворюють інтерфейс прикладного програмування (*Application Programming Interface*) – API.

Функції, що виконуються модулями ядра, є функціями ОС, які найбільш часто використовують і тому швидкість їх виконання визначає продуктивність всієї системи в цілому.

Для забезпечення високої швидкості роботи ОС все модулі ядра або велика їх частина постійно знаходяться в оперативній пам'яті (оперативному запам’ятовуючому пристрої, ОЗП), тобто є резидентними.

Допоміжні модулі ОС звичайно підрозділяються на наступні групи:

– утиліти – програми, які вирішують окремі завдання управління і супроводу комп'ютерної системи, такі, наприклад, як програми стиснення дискового простору, архівування даних і ін.;

– системні програми обробки – текстові або графічні редактори, компілятори, компонувальники, відладчики;

– програми надання користувачу додаткових послуг – спеціальний варіант інтерфейсу користувача, калькулятор, ігри і т.д.

– бібліотеки процедур різного призначення, які спрощують розробку програм, наприклад бібліотека математичних функцій, функцій I/O і т.д.

Режими роботи ядра ОС

Для надійного управління ходом виконання програм, ОС повинна мати по відношенню до програм певні привілеї. Інакше некоректно працююча програма може втрутитися в роботу ОС і, наприклад, зруйнувати частину її кодів.

Забезпечити привілеї ОС неможливо без спеціальних засобів апаратної підтримки. Апаратура комп'ютера повинна підтримувати як мінімум два режими роботи:

– режим користувача (user mode);

– привілейований режим, який також називають режимом ядра (kernel mode), або режимом супервізора (supervisor mode).

Між кількістю рівнів привілеїв, реалізованих апаратно, і кількістю рівнів привілеїв, які підтримуються ОС, немає прямої відповідності. Так на базі 4-х рівнів, що забезпечуються процесорами компанії Intel, OS/2 будує трьохрівневу систему привілеїв, а Windows NT, різновиди Unix і деякі інші обмежуються дворівневою системою.

З іншого боку, якщо апаратура підтримує хоча б два рівня привілеїв, то ОС може на цій основі створити програмним способом як завгодно розвинену систему захисту.

Підвищення стійкості ОС, яке забезпечується переходом ядра в привілейований режим, досягається за рахунок деякого уповільнення виконання

системних викликів. Системний виклик привілейованого ядра ініціює перемикання процесора з призначеного для користувача режиму в привілейований, а при поверненні до програми – назад в режим користувача (рис. 3.2). У всіх типах процесорів через додаткову дворазову затримку перемикання перехід на процедуру зі зміною режиму виконується повільніше, ніж виклик процедури без зміни режиму.

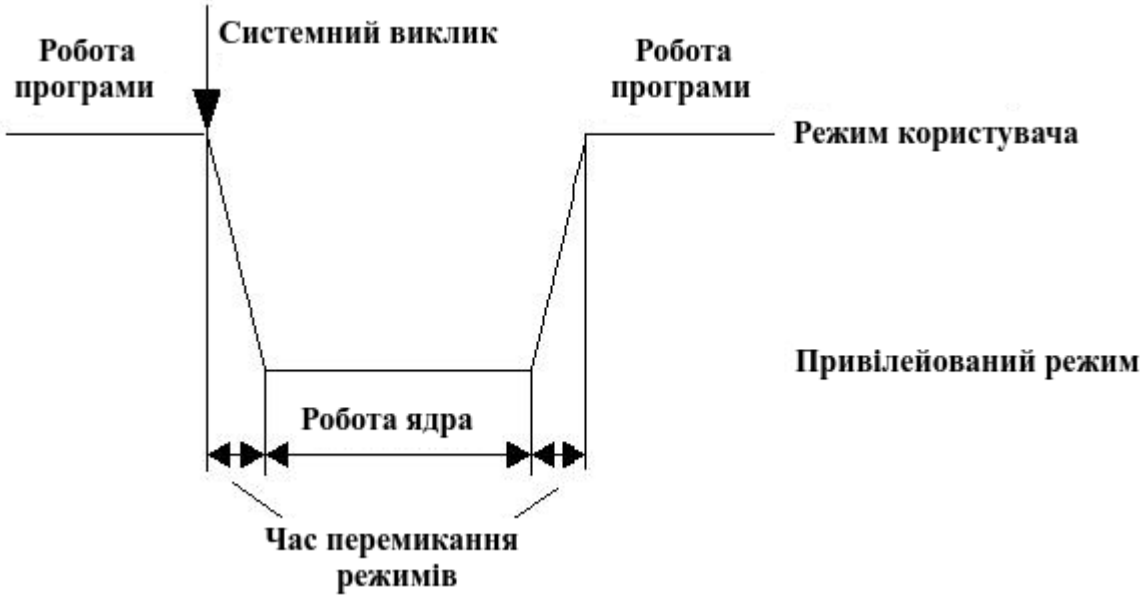


Рис. 3.2. Зміна режимів при виконанні
системного виклику до привілейованого ядру

Архітектура ОС, заснована на привілейованому ядрі та програмах для режиму користувача, стала, по суті, класичною. Її використовує багато ОС, наприклад, різні версії Unix, VAX VMS, IBM OS/390, OS/2, і з певними модифікаціями – Windows NT.

Розрізняють такі типи архітектур ядер ОС:

1. Монолітне ядро (BSD UNIX, Linux, MS-DOS, KolibriOS і ін.).
2. Модульне ядро.
3. Мікроядро (Windows CE, Symbian OS, OpenVMS, Mach, QNX, AIX, Minix
та ін.).
4. Екзоядро.
5. Наноядро (KeyKOS).
6. Гібридне ядро (деякі варіанти Unix/Linux, лінійка Windows NT).

Всі частини монолітного ядра працюють в одному адресному просторі. Переваги: швидкість роботи, спрощена розробка модулів.

Модульне ядро – це сучасна, вдосконалена модифікація архітектури монолітних ядер ОС. На відміну від “класичних” монолітних ядер, модульні

ядра, як правило, не вимагають повної перекомпіляції ядра при зміні складу апаратного забезпечення комп'ютера. Замість цього модульні ядра надають механізм підвантаження модулів ядра, які підтримують певне апаратне забезпечення (наприклад, драйвері). Підвантаження модулів може бути як динамічним (без перезавантаження ОС) так і статичним (виконується при перезавантаженні ОС).

Класичні мікроядра надають лише дуже невеликий набір низькорівневих примітивів, або системних викликів, що реалізують базові сервіси ОС. Основною перевагою мікроядерної архітектури є висока ступінь модульності ядра, а також стійкість до збоїв обладнання та помилок в компонентах ОС. Недолік: передача між процесами вимагає накладних витрат.

Екзоядро – ядро ОС, що надає лише функції для взаємодії між процесами, безпечного виділення і звільнення ресурсів. Передбачається, що API для прикладних програм будуть надаватися зовнішніми до ядра бібліотеками.

Наноядро – архітектура ядра ОС, в рамках якої вкрай спрощене і мінімалістське ядро виконує лише одну задачу – обробку апаратних переривань, що генеруються пристроями комп'ютера. Після обробки переривань від апаратури наноядро посилає інформацію про результати обробки вищерозміщеному ПЗ за допомогою того ж механізму переривань.

Гібридне ядро – модифіковані мікроядра, що дозволяють для прискорення роботи запускати "несуттєві" частини в просторі ядра. Наприклад, управління віртуальною пам'яттю і робота низькорівневих драйверів забезпечується мікроядром. Всі інші функції, в тому числі взаємодія з прикладними програмами, здійснюється монолітним ядром.

При описі ОС часто вказуються особливості її структурної організації та основні концепції, покладені в її основу. До таких базових концепцій відносяться:

1. Побудова ОС на базі монолітного ядра.
2. Використання багаторівневого підходу при побудові ядра.
3. Використання мікроядерного підходу.
4. Розподілена організація ОС на основі клієнт-серверної моделі.

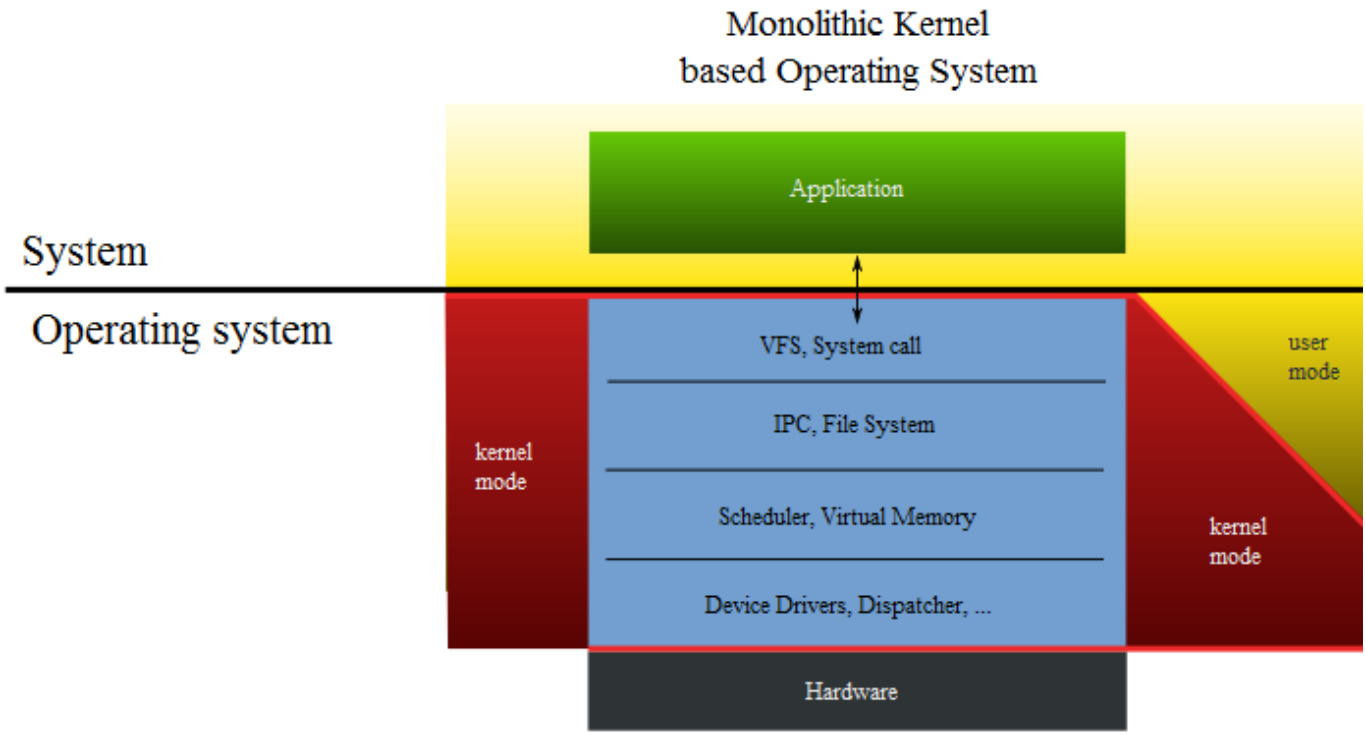


Рис. 3.3 Структура ОС на базі монолітного ядра

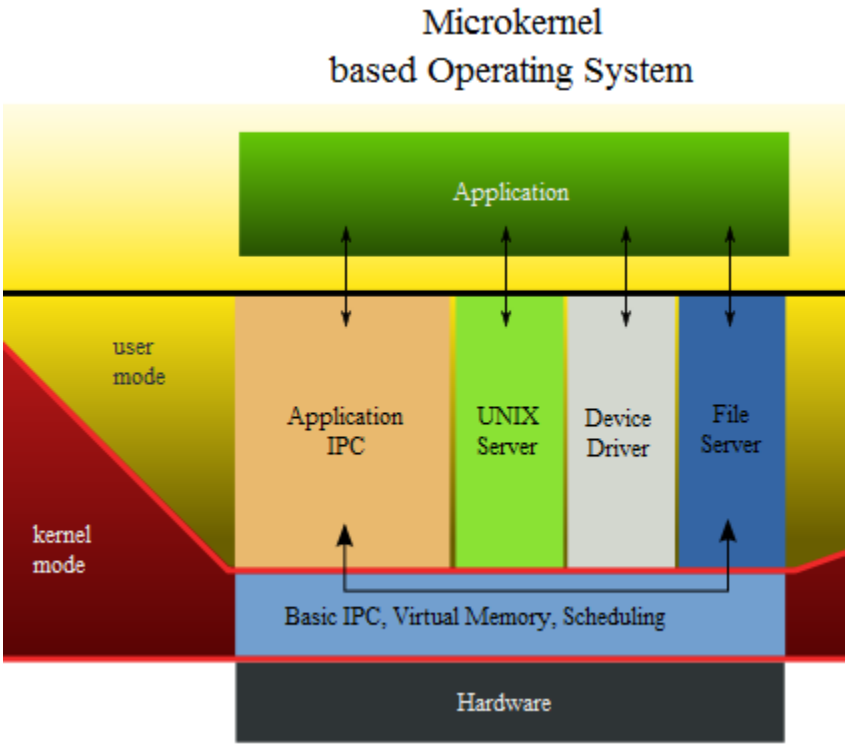


Рис. 3.4. Структура ОС на базі мікроядра

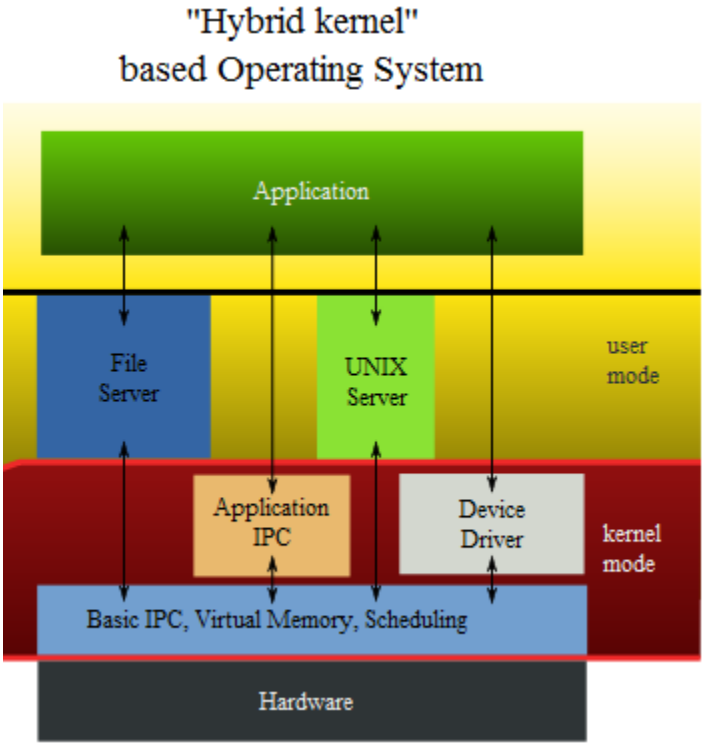


Рис. 3.5. Структура ОС на базі гібридного ядра

ОС на базі монолітного ядра

Організація ОС на монолітному ядрі є найпоширенішим та найстарішим способом – ОС працює як єдина програма в режимі ядра, написана у вигляді набору процедур. Служби, системні виклики, що надаються ОС, запитуються шляхом приміщення параметрів в чітко визначене місце, наприклад, в стек. Потім виконується спеціальна інструкція ОС, що перемикає машину з призначеного для користувача режиму в режим ядра і передає управління ОС. Потім ОС витягує параметри і визначає, який системний виклик повинен бути виконаний. Після цього ОС переміщається по індексу в таблиці на рядок стека, що містить покажчик на процедуру, яка виконує системний виклик.

Описана організація передбачає таку базову структуру ОС:

1. Основна програма, що викликає необхідну службову процедуру.
2. Набір службових процедур, що виконують системні виклики.
3. Набір допоміжних процедур, що сприяють роботі службових процедур.

У цій моделі для кожного системного виклику є одна відповідальна за нього службова процедура, яка його і виконує. Допоміжні процедури виконують дії, необхідні декільком службовим процедурам, зокрема вилучення даних з програм користувача (рис. 3.6).

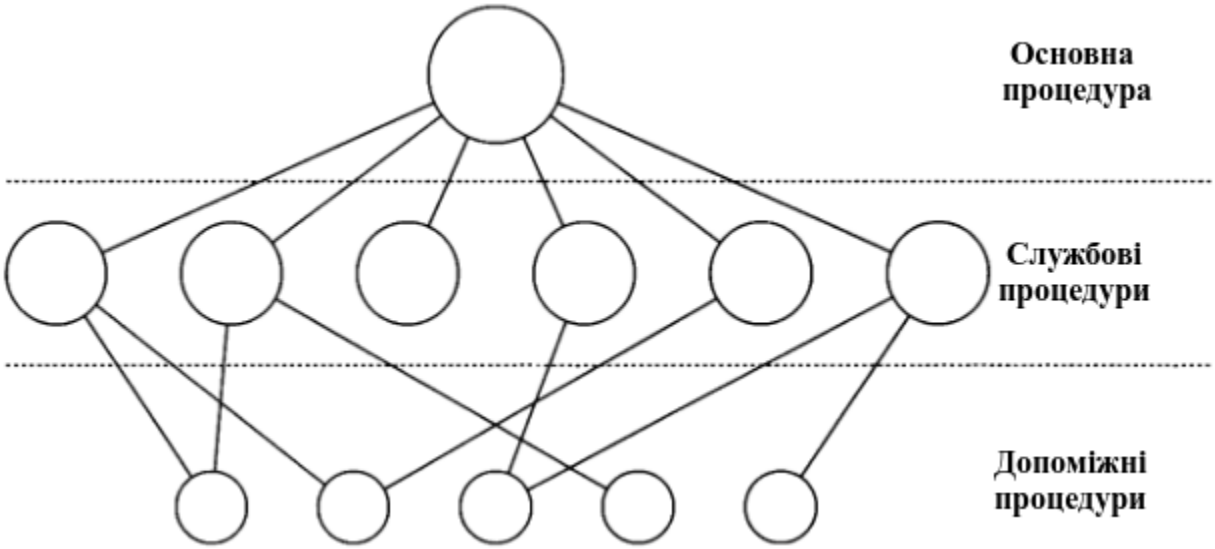


Рис. 3.6. Проста структурована модель монолітної системи

На додаток до основної ОС, що завантажується під час запуску комп'ютера, багато ОС підтримують завантаження розширення, в число яких входять драйвери пристроїв I/O та файлові системи (ФС). Такі компоненти завантажуються в міру потреби.

Багаторівневий підхід

Узагальненням підходу, показаного на рис. 3.6, є організація ОС у вигляді ієрархії рівнів, кожен з яких є надбудовою над нижнім рівнем. Наприклад, перший або нульовий рівень займається розподілом процесорного часу, перемиканням між процесами. Наступний рівень управляє пам'яттю. Третій рівень управляє зв'язком оператора з процесом. Наступний – I/O і т.д. Можливі й інші функції рівнів (рис. 3.7).

Рис. 3.7. Багатошарова структура ядра ОС

Мікроядерний підхід

При використанні багаторівневого підходу розробникам необхідно вибрати, де провести межу між режимами ядра і користувача. Традиційно всі рівні входять в ядро, але це необов'язково.

ОС на базі мікроядра мають високу надійність за рахунок розбиття ОС на невеликі модулі. Тільки один з них – мікроядро – запускається в режимі ядра (привілейованому режимі), а всі інші запускаються у вигляді відносно слабко наділених повноваженнями звичайних процесів користувачів. Якщо, наприклад, в драйвері будь-якого пристрою відбудеться помилка, то вона не призведе до повного зависання комп'ютера, тільки викличе неможливість доступу до цього пристрою.

Функції ОС більш високого рівня виконують спеціалізовані компоненти – сервери, що працюють в режимі користувача. При такій побудові ОС працює більш повільно, так як часто виконуються переходи між привілейованим режимом і режимом користувача, однак система виходить більш гнучкою – її функції можна модифікувати, модифікуючи сервери режиму користувача. Крім того, сервери добре захищені один від одного, як і будь-які призначені для користувача процеси. Найбільш часто цей принцип побудови закладений в ОС реального часу, що працюють в промислових пристроях, авіоніки і військовій техніці.

Клієнт-серверна модель

Існують реалізації ідеї мікроядер, виражені в відокремленні двох класів процесів: серверів, кожен з яких надає якусь службу, і клієнтів, які користуються цими службами. Зв'язок між клієнтами та серверами часто організовується за допомогою передачі повідомлень. Щоб скористатися службою, клієнтський процес готує повідомлення, в якому говориться, що саме йому потрібно, і відправляє його відповідній службі. Служба виконує певну роботу і відправляє назад відповідь. Клієнт-серверна модель є абстракцією, яка використовується як для окремо взятої машини, так і для машин, об'єднаних в мережу (рис. 3.8).

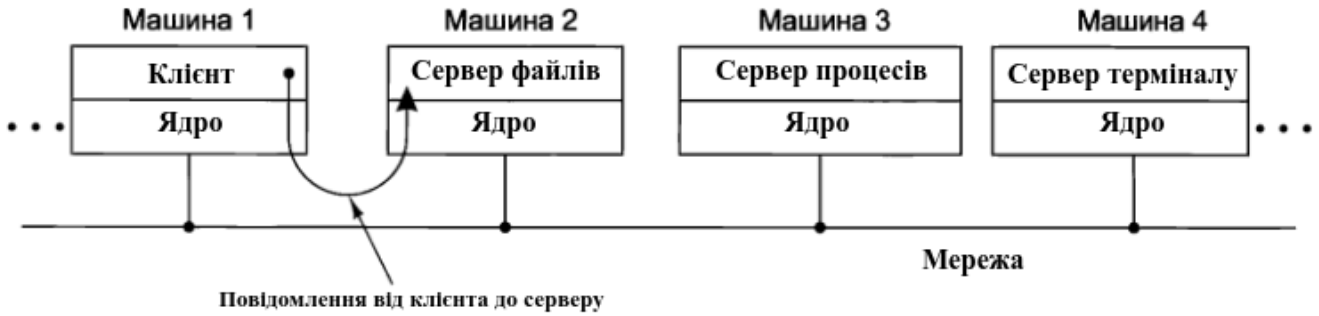


Рис. 3.8. Клієнт-серверна модель, реалізована за допомогою мережі

Розподілена організація ОС дозволяє спростити роботу користувачів в мережевих середовищах, представляє мережу у вигляді традиційного одно- або багатопроцесорного комп'ютера. Характерними ознаками розподіленої організації ОС є: наявність єдиної довідкової служби розподілених ресурсів, єдиної служби часу, використання механізму виклику віддалених процедур (Remote Procedure Call, RPC) для прозорого розподілу програмних процедур по машинах, багатониткової обробки.

Лекція 4. Архітектура Unix/Linux

Дворівнева модель системи представлена на рис. 4.1.

Рис. 4.1. Спрощена модель системи Unix

У центрі знаходиться ядро системи (kernel). Ядро безпосередньо взаємодіє з апаратною частиною комп'ютера, ізолюючи прикладні програми від її власної архітектури. Ядро має набір послуг, що надаються прикладним програмам. До послуг ядра відносяться операції введення/виведення (відкриття, читання, запису та управління файлами), створення та управління процесами, їх синхронізації і взаємодії між процесами. Ядро розподіляє пам'ять і забезпечує доступ до файлів та периферійних пристроїв. Всі програми запитують послуги ядра за допомогою *системних викликів*. Структурна схема ядра представлена на рис. 4.2.

Другий рівень в моделі системи Unix складають програму та завдання, як системні, що визначають функціональність системи, так і прикладні, що забезпечують інтерфейс користувача Unix. Однак, незважаючи на зовнішню різнорідність програм, схеми їх взаємодії з ядром однакові.

Ядро часто називають основною частиною (core) або контролером ОС. Типові компоненти ядра – обробники переривань, які обслуговують запити на переривання, планувальник, що розподіляє процесорний час між багатьма процесами, система управління пам'яттю, яка управляє адресним простором процесів, і системні служби, такі як мережева підсистема та підсистема взаємодії між процесами.

Стан системи, в якому знаходиться ядро, і область пам'яті, в якій знаходиться ядро, разом називаються *простором ядра* (або режимом ядра, kernel-space). Відповідно, призначені для користувача програми виконуються в *просторах завдань* (режим користувача, режим завдань, user-space). Програмам користувача доступним є лише деяка підмножина машинних ресурсів, вони не можуть виконувати деякі системні функції, безпосередньо звертатися до

апаратури і робити інші заборонені операції. При виконанні програмного коду ядра система знаходиться в просторі (режимі) ядра, на відміну від нормального виконання призначених для користувача програм, яке відбувається в режимі завдання.

Прикладні програми, що працюють в системі, взаємодіють з ядром за допомогою інтерфейсу *системних викликів* (system call) (рис. 4.3).

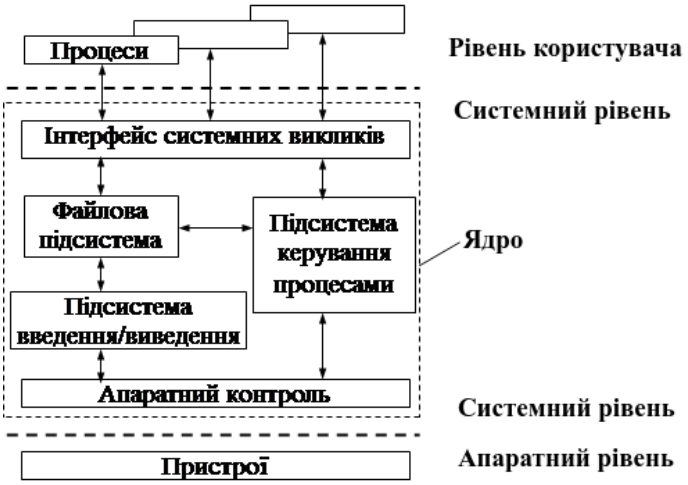


Рис. 4.2. Внутрішня структура ядра Unix

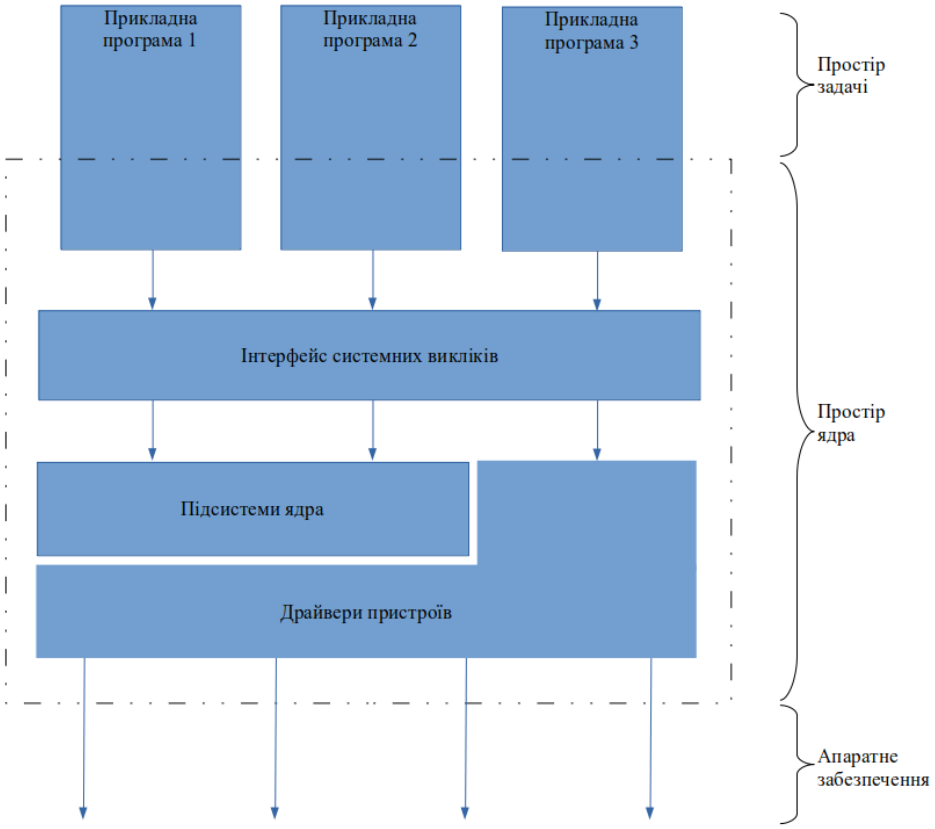


Рис. 4.3. Взаємодія між прикладними програмами, ядром та апаратним забезпеченням

Прикладна програма зазвичай викликає функції різних бібліотек, наприклад бібліотеки функцій мови С, які, в свою чергу, звертаються до інтерфейсу системних викликів для того, щоб віддати наказ ядру виконати певні дії від їхнього імені. Деякі бібліотечні виклики надають функції, для яких відсутній системний виклик, і тому звернення до ядра – це тільки один етап в більш складній функції. Наприклад, функція *printf( )* – забезпечує форматування та буферизацію даних і лише після цього один раз звертається до системного виклику *write( )* для виведення даних на консоль.

Деякі бібліотечні функції за іменем відповідають функціям ядра один до одного. Наприклад, бібліотечна функція *open( )* не робить нічого, крім виконання системного виклику *open( )*. У той же час деякі бібліотечні функції, як, наприклад, *strcpy( )*, взагалі не використовують звернення до ядра.

Коли прикладна програма виконує системний виклик, то кажуть, що ядро виконує роботу від імені прикладної програми. Більш того, говорять, що прикладна програма виконує системний виклик в просторі ядра, а ядро виконується в контексті процесу. Такий тип взаємодії, коли прикладна програма звертається до ядра через інтерфейс системних викликів, є фундаментальним способом виконання завдань.

У функції ядра входить також управління системним апаратним забезпеченням. Практично всі платформи, на яких працює ОС, використовують *переривання* (interrupt). Коли апаратному пристрою необхідно якось взаємодіяти з системою, воно генерує переривання, яке перериває роботу ядра в асинхронному режимі. Іншими словами, заздалегідь невідомо, в який момент часу ця подія відбудеться і в якому стані буде система в цей момент часу.

Загальну архітектуру ОС на базі ядра GNU/Linux часто розглядають як піраміду (рис. 4.4). На рис. 4.5 представлена детальна структура ядра GNU/Linux.

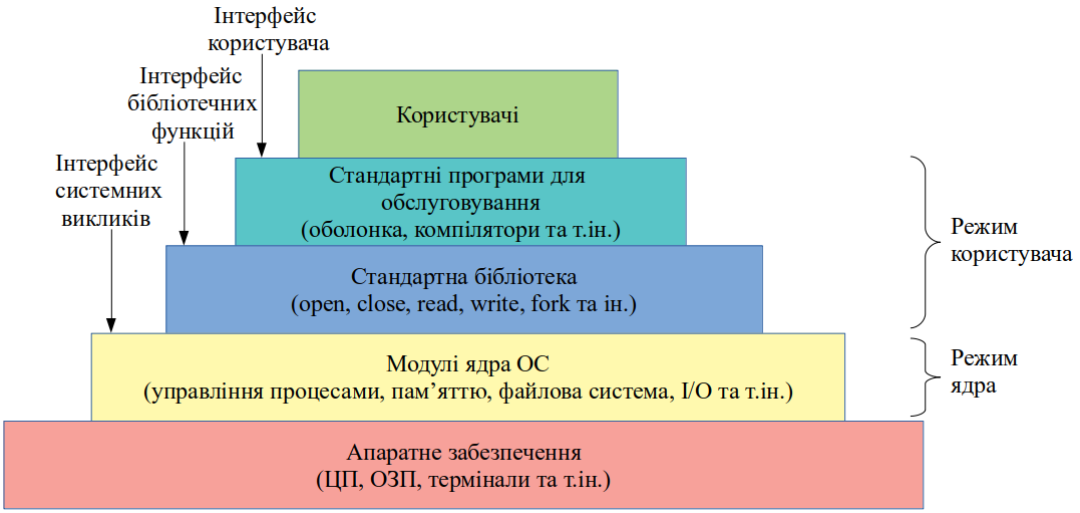


Рис. 4.4. Рівні ОС на базі ядра GNU/Linux

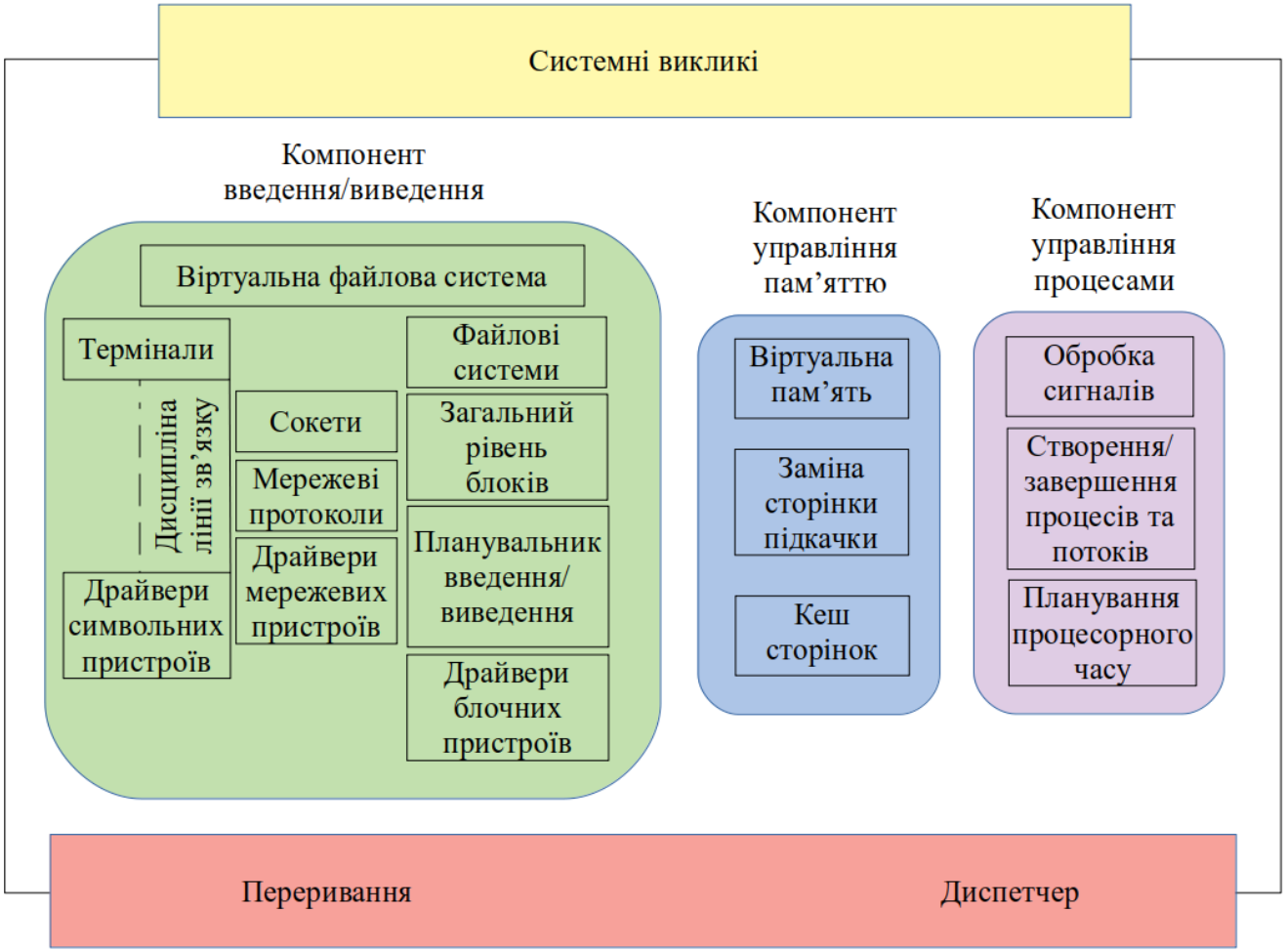


Рис. 4.5. Структура ядра GNU/Linux

Нижній рівень ядра складається з обробників переривань, що є основним засобом взаємодії з пристроями, і механізму диспетчеризації на низькому рівні. Диспетчеризація здійснюється при виникненні переривання. При цьому код низького рівня зупиняє виконання процесу, який виконується, зберігає його стан в структурах процесів ядра і запускає відповідний драйвер. Диспетчеризація процесів проводиться також і тоді, коли ядро завершує якусь операцію і необхідно знову запустити процес користувача.

На найнижчому рівні всі операції I/O проходять через певний драйвер пристрою. Всі драйвери в GNU/Linux класифікуються або як символьні драйвери пристроїв, або як блокові драйвери. Основна різниця полягає в тому, що пошук і довільний доступ дозволені тільки для блокових пристроїв. Мережеві пристрої, які виділені в окрему категорію, з технічної точки зору, є символьними, однак робота з ними ведеться дещо інакше. Вище рівня драйверів пристроїв код ядра для кожного типу пристроїв свій.

До завдань управління пам'яттю входять: обслуговування відображення віртуальної пам'яті на фізичну; підтримка кешу сторінок, до яких нещодавно

отримували доступ (реалізація стратегії заміни сторінок); доставку в пам'ять (на вимогу) нових сторінок з кодом і даними.

Основна сфера відповідальності компонента управління процесами – створення і завершення процесів. У ньому є планувальник процесів, який вибирає, який процес (потік) буде працювати наступним.

Представлені на рис. 4.5 компоненти ядра постійно взаємодіють між собою. Крім статичних компонентів ядро GNU/Linux підтримує модулі що динамічно завантажуються. Ці модулі можуть використовуватися для додавання або заміни драйверів пристроїв за замовчуванням, файлових систем, роботи в мережі, а також інших кодів ядра.

Лекція 5. Фізична і логічна організація файлової системи

Файлова система (ФС) – це частина ОС, що включає:

– сукупність усіх файлів на диску;

– набори структур даних, використовуваних для управління файлами, такі, наприклад, як каталоги файлів, дескриптори файлів, таблиці розподілу вільного і зайнятого простору на диску;

– комплекс системних програмних засобів, що реалізують різні операції над файлами, такі як створення, знищення, читання, запис, іменування і пошук файлів. Принципи розміщення файлів, каталогів і системної інформації на реальному пристрої описуються фізичної організацією ФС. Очевидно, що різні ФС мають різну фізичну організацію.

Жорсткий диск (вінчестер) складається з пластин[2] (рис. 5.1, 5.2). На кожній стороні кожної пластини розмічені тонкі концентричні кільця – *доріжки* (*tracks*), на яких зберігаються дані. Кількість доріжок залежить від типу диска. Нумерація доріжок починається з нуля від зовнішнього краю до центру диска.

Сукупність доріжок одного радіусу на всіх поверхнях всіх пластин називається *циліндром* (*cylinder*). Кожна доріжка розбивається на фрагменти – *сектори* (*sectors*), або *блоки* (*blocks*). Всі доріжки мають рівне число секторів, в які можна максимально записувати одне і те ж число байт. Сектор має фіксований для конкретної системи розмір, що виражається ступенем двійки. Найчастіше розмір сектора становить 512 байт. З огляду на, що доріжки різного радіусу мають однакове число секторів, щільність запису стає вище, чим ближче доріжка до центру.

Сектор – найменша одиниця обміну даними дискового пристрою з оперативною пам'яттю, що адресується (рис. 5.2). Для того, щоб контролер міг знайти на диску потрібний сектор, необхідно вказати йому всі складові адреси сектора: *номер циліндра*, *номер поверхні* та *номер сектору*.

ОС при роботі з диском використовує, як правило, власну одиницю дискового простору, яка називається *кластером* (*cluster*). При створенні файлу, місце на диску йому виділяється кластерами. Наприклад, якщо файл має розмір 2560 байт, а розмір кластера в ФС визначено в 1024 байт, то файлу буде виділено на диску 3 кластера.

Примітка: іноді кластер називають блоком (наприклад, в ОС Unix), що може призвести до термінологічної плутанини. Тому слід трактувати терміни з урахуванням контексту.

2 В лекції використані рисунки з мережі Internet.

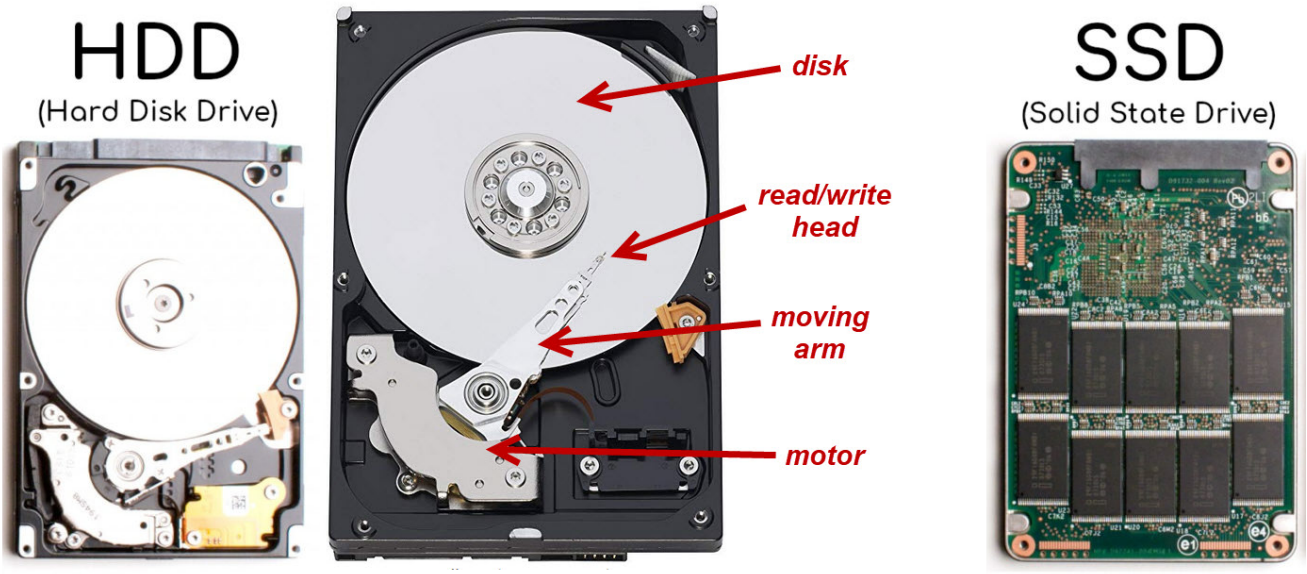


Рис. 5.1. Основні сучасні різновиди пристроїв накопичення даних

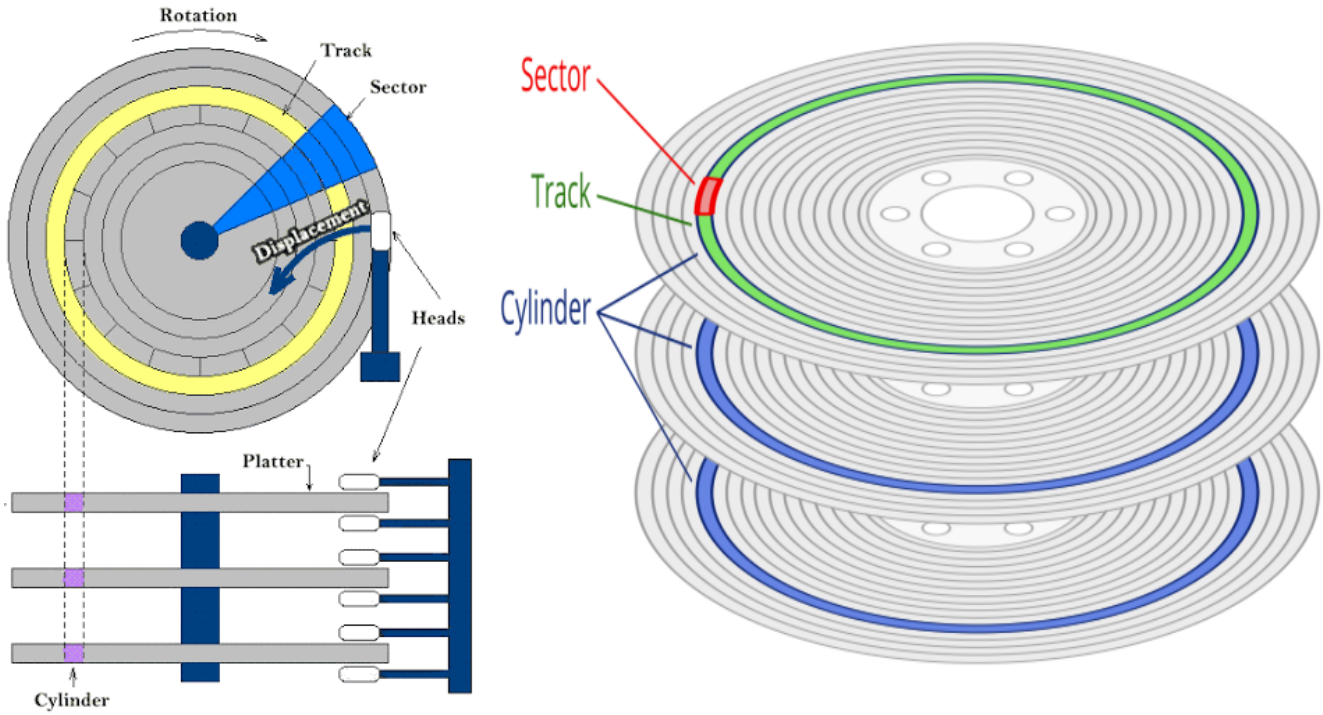


Рис. 5.2. Внутрішня побудова HDD (жорсткого диску) та фізична модель зберігання даних

Доріжки і сектори створюються в результаті виконання процедури *фізичного або низькорівневого форматування* диска, яке роблять (як правило на фабриках по виготовленню дисків) перед використанням диску. Низькорівневий формат диска не залежить від типу ОС, яка цей диск буде використовувати.

Розмітку диска під конкретний тип ФС виконують процедури *високорівневого або логічного форматування*. При цьому визначається розмір кластера та на диск записується інформація, необхідна для роботи ФС.

Перед форматуванням диску під певну ФС, він може бути розбитий на *розділи*.

Розділ – це безперервна частина фізичного диска, яку ОС надає користувачеві як *логічний пристрій* (використовуються також назви логічний диск, логічний розділ). Логічний пристрій функціонує так, як би це був окремий фізичний диск.

Логічний пристрій може бути створено і на базі декількох розділів, причому ці розділи не обов'язково повинні належати одному фізичному пристрою. Об'єднання декількох розділів в єдиний логічний пристрій виконується різними шляхами і переслідує різні цілі, основні з яких наступні.

1. Збільшення загального обсягу логічного розділу.
2. Підвищення продуктивності роботи диску.
3. Підвищення відмовостійкості диску.

Прикладами організації спільної роботи декількох дискових розділів є так звані RAID-масиви.

Важливим компонентом фізичної організації ФС є фізична організація файлу, тобто спосіб розміщення файлу на диску. Основними критеріями ефективності фізичної організації файлів є:

– швидкість доступу до даних;
– обсяг інформації файлу, яка адресується;
– ступінь фрагментації дискового простору;
– максимально можливий розмір файлу.

Однією з основних завдань ОС є надання зручностей користувачеві при роботі з даними, що зберігаються на дисках. Для цього ОС представляє фізичну структуру даних, що зберігаються, деякою зручною для користувача логічною моделлю. Така модель представляється, наприклад, ієрархічним деревом каталогів, що містять файли. У цьому випадку говорять про *логічну організацію* ФС.

Лекція 6. Найвідоміші різновиди файлових систем

FAT (File Allocation Table, таблиця розміщення файлів) – класична архітектура ФС, відома своєю простотою і широтою використання для флеш-накопичувачів, зовнішніх дисків і карт пам'яті. Розроблено в 1976-1977 рр. Має три різновиди: FAT12, FAT16, FAT32 – відрізняються розрядністю записів в дисковій структурі (кількістю біт, відведених для зберігання номера кластера). FAT12 використовувалася для дискет, FAT16 – використовувалася для дисків малого обсягу. Починаючи з Windows 95 з'явилася VFAT – розширення FAT, яке підтримує довгі імена (до 255 символів). FAT12 та FAT16 підтримують іменування у *форматі 8.3* (8 символів – ім’я файлу та 3 символи – розширення файлу).

В 2006 році Microsoft випустила під пропрієтарною ліцензією exFAT (Extensible File Allocation Table) для ОС Windows Embedded CE 6.0. Її іноді називають FAT64. ФС призначена для використання на USB flash-девайсах та SD-картах. Лише в 2019 році Microsoft відкрила специфікацію цієї ФС.

ФС	Макс. довжини імені файлу	Макс. довжина шляху файлу	Макс. розмір файлу	Макс. розмір тому
FAT12	8+3 (255 для VFAT)	260 байт	32 MiB	1MiB – 32MiB
FAT16	8+3 (255 для VFAT)	260 байт	2GiB	16MiB – 2GiB
FAT32	255 байт	?	4GiB	512MiB – 8TiB
exFAT	255 символів	?	128 PiB (теоретично 16 EiB – 1)	128 PiB (512 TiB рекомендований)

NTFS (New Technology File System, файлова система нової технології) – стандартна ФС для сімейства ОС MS Windows NT. Вона підтримує систему метаданих і використовує спеціалізовані структури даних для зберігання інформації про файли для поліпшення продуктивності, надійності та ефективності використання дискового простору, використовує систему журналювання. Розроблено на основі ФС HPFS (High Performance File System), створеної спільно Microsoft та IBM для OS/2.

Розрізняють такі версії NTFS: v.1.2 – MS Windows NT 3.51/4.0; v.3.0 – MS Windows 2000; v.3.1 – MS Windows XP/Vista/7/8/8.1/10, MS Windows Server 2003/2003 R2/2008/2008 R2/2012/2016/2019.

ФС	Макс. довжини імені файлу	Макс. довжина шляху файлу	Практичний макс. розмір файлу	Рекомендований макс. розмір тому
NTFS	255 символів	32767 символів	2^{44} байт мінус 64 кБ	не більше 2TB

ext4 або ext4fs (Fourth Extended File System, четверта версія розширеної файлової системи) – журнальована ФС, яка використовується в ОС з ядром Linux. Заснована на ФС ext3. Вперше випущена 10.10.2006 р Ендрю Мортоном. Основна особливість – збільшення максимального обсягу розділу диска до 1 ексабайту (2^{60} байт) при розмірі блоку 4 Kb і збільшенні розміру одного файлу до 16 терабайт. Нова ФС дозволяє за рахунок спеціальних алгоритмів значно зменшити фрагментацію файлів і підвищує продуктивність.

ФС	Макс. довжини імені файлу	Макс. розмір файлу	Макс. розмір тому
ext2	255 байт	16GiB – 2TiB	2TiB – 32TiB
ext3			
ext4		16GiB – 16TiB	1EiB

Також популярними останнім часом в GNU/Linux-сумісних системах окрім ext4 є ФС btrfs та zfs.

UFS (Unix File System) – ФС, яка створена для сімейства BSD та використовується в переробленому та доповненому вигляді в ОС FreeBSD, OpenBSD, NetBSD, PC-BSD, SunOS.

ФС	Макс. довжини імені файлу	Макс. розмір файлу	Макс. розмір тому
UFS1	255 байт	4GiB – 256TiB	256TiB
UFS2		512GiB – 32PiB	1YiB

HFS Plus або HFS+ або Mac OS Extended (Hierarchical File System, ієрархічна файлова система) – ФС, що розроблена Apple Inc. для заміни 16-ти бітної HFS на комп'ютерах Macintosh, яка використовувалася раніше, для розширення можливостей Mac OS. Представлена 19.01.1998 р. в Mac OS 8.1. У Mac OS 10.3 з'явилася ФС HFSX, головною відмінністю від HFS+ є режим роботи з урахуванням регістра імен. Сучасною ФС в ОС Apple macOS є APFS.

ФС	Макс. довжини імені файлу	Макс. розмір файлу	Макс. розмір тому
HFS	30 байт	?	?
HFS+ (Mac OS Extended)	255 символів	8 EiB	8 EiB
APFS (Apple File System)	?	8 EiB	?

XFS – високопродуктивна журнальована ФС, яка створена компанією Silicon Graphics для ОС IRIX (різновид Unix). Спочатку розрахована для використання на дисках великого обсягу (більш 2 ТВ). Добре адаптована в SuSE, Gentoo, Mandriva, Slackware, Ubuntu, Fedora та Debian. Одними з основних недоліків вважається відносно високе навантаження на CPU, а також можливість втрати даних під час запису при збої живлення, тому що в пам'яті зберігається велика кількість буферів.

ФС	Макс. довжини імені файлу	Макс. розмір файлу	Макс. розмір тому
XFS	255 байт	9EiB	9EiB

JFS (Journaled File System) – 64 бітна журнальована ФС компанії IBM для ОС AIX (різновид Unix). Існують два покоління – JFS1 та JFS2. Для OS/2 і Linux існує тільки в другому поколінні, яка просто зветься JFS. Також JFS називають ФС VxFS компанії Veritas Software, яка використовується в ОС HP-UX (різновид Unix). Основна мета системи – забезпечити високу продуктивність, надійність та масштабованість для багатопроцесорних комп'ютерів. Спочатку проектувалася як журнальована. JFS веде журнал метаданих, підтримуючи структуру ФС цілісною, але не обов'язково зберігає дані. Відключення живлення або крах системи може призвести до збереження застарілих копій файлів, однак самі файли залишаться придатними до використання.

ФС	Макс. довжини імені файлу	Макс. розмір файлу	Макс. розмір тому
JFS1	255 байт	8EiB	512TiB – 4Pib
JFS2		4PiB	32PiB

ISO 9660 – стандарт, що описує ФС для дисків CD-ROM. Метою стандарту є забезпечення сумісності носіїв під різними ОС – Unix/Linux, macOS, Windows та ін. Стандарт випущений в 1988 році. Розширення стандарту, що має назву Joliet, додає підтримку довгих імен файлів і не ASCII символів в іменах файлів. Розширення стандарту Rock Ridge Interchange Protocol – розширення ФС ISO 9660, яке розроблене для зберігання файлових атрибутів, які використовуються в ОС на базі стандарту POSIX.

UDF (Universal Disk Format, універсальний дисковий формат) – специфікація формату ФС, що не залежить від ОС для зберігання файлів на оптичних носіях. UDF є реалізацією стандарту ISO/IEC 13346. Цей стандарт покликаний замінити ISO 9660. Він розроблений і розвивається Optical Storage Technology Association. Існує кілька версій UDF.

ФС	Макс. довжини імені файлу	Макс. довжина шляху файлу	Макс. розмір файлу
ISO 9660 Level1	8.3	до 8 директорій	2ГБ
ISO 9660 Level2	32 символа	до 8 директорій	
ISO 9660 + Joliet	64 символа	?	
ISO 9660 + Rock Ridge	255 символів	?	
UDF	255 байт	1023 байта	16EiB

Існує безліч і інших ФС.

Представлені одиниці виміру інформації:

Вимірювання в байтах					
Десяткова приставка			Двійкова приставка		
Назва	Символ	Ступінь	Назва	Символ	Ступінь
		ДСТУ*			МЕК**
байт	B	10^0	байт	B байт	2^0
кілобайт	kB	10^3	кібібайт	KiB Кбайт	2^{10}
мегабайт	MB	10^6	мебібайт	MiB Мбайт	2^{20}
гігабайт	GB	10^9	гібібайт	GiB Гбайт	2^{30}
терабайт	TB	10^{12}	тебібайт	TiB Тбайт	2^{40}
петабайт	PB	10^{15}	пебібайт	PiB Пбайт	2^{50}
ексабайт	EB	10^{18}	эксбібайт	EiB Эбайт	2^{60}
зетабайт	ZB	10^{21}	зебібайт	ZiB Збайт	2^{70}
йотабайт	YB	10^{24}	йобібайт	YiB Йбайт	2^{80}

* – 1. ДСТУ IEC 80000-13:2016 (IEC 80000-13:2008, IDT). Величини та одиниці. [Чинний від 2018-01-11]. Вид. офіц. Київ : ДП «УкрНДНЦ», 2018. 26 с. (Частина 13. Інформатика та інформаційні технології).
2. Про затвердження визначень основних одиниць SI, назв та визначень похідних одиниць SI, десяткових кратних і частинних від одиниць SI, дозволених позасистемних одиниць, а також їх позначень та Правил застосування одиниць вимірювання і написання назв та позначень одиниць вимірювання і символів величин : Наказ Міністерства економічного розвитку і торгівлі України від 04.08.2015 р. № 914. URL: https://zakon.rada.gov.ua/laws/show/z1022-15/print (дата звернення: 02.02.2020).

** – Міжнародна електротехнічна комісія (МЕК) в березні 1999 року ввела стандарт МЕК 60027-2, в якому описано іменування двійкових чисел. Аналогічний стандарт IEEE 1541-2002 введений в 2008 р.

Лекція 7. Управління дисковими розділами та змінними носіями в GNU/Linux-подібних ОС

Номенклатура накопичувачів та їх розділів в Unix\Linux

Управління дисковими розділами тісно пов'язано з налаштуванням початкового завантажувача, оскільки кожна з завантажуваних ОС знаходиться, як правило, в своєму розділі. Так як диски, їх первинні та логічні розділи, а також будь-які інші носії представлені файлами з каталогу /dev, для них існує єдина система номенклатури.

Приводи гнучких дисків іменуються /dev/fd0 (дисковод A:) та /dev/fd1 (дисковод B:). Іменування змінних носіїв типу так званих супер-дискет (LS-120 та їх аналоги), що підключаються до інтерфейсу IDE, описується правилами для IDE-накопичувачів.

Будь-які носії інформації з інтерфейсом IDE/EIDE/ATAPI позначаються в формі hdL#, де L – літера, що однозначно ідентифікує носій фізично, а # – номер розділу (первинного або розширеного).

Так, наприклад, перший жорсткий диск на першому каналі IDE (Primary Master) позначається як /dev/hda, другий диск (або будь-який інший накопичувач) на першому каналі (Primary Slave) – як /dev/hdb, перший накопичувач на другому каналі (Secondary Master) – як /dev/hdc, другий накопичувач на другому каналі (Secondary Slave) – як /dev/hdd (таблиці 7.1 та 7.2).

Таблиця 7.1

Номенклатура накопичувачів IDE/EIDE/ATAPI

Канал IDE	*Перший*	*Другий*
Master	hda	hdc
Slave	hdb	hdd

Таблиця 7.2

Номенклатура накопичувачів IDE/EIDE/ATAPI
при наявності додаткового контролеру

Канал IDE	*Основний*		*Додатковий*	
	Перший	*Другий*	*Перший*	*Другий*
Master	hda	hdc	hde	hdg
Slave	hdb	hdd	hdf	hdh

Буквений ідентифікатор IDE-накопичувачів будь-якого типу (жорстких дисків, приводів CD-ROM R/RW, Zip, LS-120) зростає в залежності не від кількості

підключених пристроїв, а від його знаходження в структурі IDE-каналів. Так, єдиний жорсткий диск, підключений до другого каналу в якості першого пристрою (Secondary Master), буде називатися /dev/hdc, навіть якщо раніше ніяких накопичувачів підключено не було.

Дискові розділи IDE-пристроїв позначаються цифрами після літерного ідентифікатора. При цьому за первинними розділами (Primary Partition, яких на фізичному диску не може бути більше чотирьох) зарезервовані цифри від 1 до 4. Наприклад, якщо перший диск на першому IDE-каналі розбитий на чотири первинних розділу, вони будуть позначатися як /dev/hda1 , /dev/hda2, /dev/hda3, /dev/hda4 (таблиця 7.3).

Таблиця 7.3

Номенклатура первинних розділів на накопичувачах IDE

Розділ	*Primary Master*	*Primary Slave*	*Secondary Master*	*Secondary Slave*
Перший	hda1	hdb1	hdc1	hdd1
Другий	hda2	hdb2	hdc2	hdd2
Третій	hda3	hdb3	hdc3	hdd3
Четвертий	hda4	hdb4	hdc4	hdd4

Логічні томи (Volume) всередині розширеного розділу (так званий Extended Partition) отримують номери, починаючи з п'ятого, незалежно від кількості первинних розділів (і навіть якщо жодного первинного розділу на диску немає).

Змінні носії типу CD-ROM будуть ідентифікуватися тільки послідовностями символів без цифр: /dev/hdc, /dev/hdd і т.д., так як розділів на них зазвичай не буває. Нові, з коробки, диски ZIP мають розділ виду /dev/hdc4, відформатований, наприклад, в *vfat*.

Для накопичувачів з інтерфейсом SCSI, SATA та USB система номенклатури інша.

Такі жорсткі диски іменуються в порядку підключення до шини так: /dev/sda, /dev/sdb і так далі, а їх розділи – /dev/sda1, /dev/sda2, /dev/sda5 і т.д.

Правила для нумерації первинних та логічних розділів – ті ж самі, що і для IDE-дисків.

Поширене також іменування оптичних приводів, як пристроїв виду /dev/sr0, /dev/sr1 і т.д.

Слід зауважити, що в сучасних GNU/Linux-сумісних ОС використовується оновлена редакція специфікації, що надана в таблиці 7.3. В таблиці перелічені пристрої в такій послідовності, якби вони були всі разом

підключені у системі. Тобто разом би існували як старі інтерфейси IDE, так і сучасні SATA.

Таблиця 7.3

Оновлена специфікація накопичувачів в GNU/Linux-сумісних, FreeBSD та macOS

GNU/Linux-сумісні		*FreeBSD*	*macOS*	*Опис*
Стара редакція	*Нова редакція*			
/dev/fd0, /dev/fd1, . . .			–	Floppy drives
/dev/hda	/dev/sda	/dev/ada0	–	Primary IDE Master
/dev/hdb	/dev/sdb	/dev/ada1	–	Primary IDE Slave
/dev/hdc	/dev/sdc	/dev/ada2	–	Secondary IDE Master
/dev/hdd	/dev/sdd	/dev/ada3	–	Secondary IDE Slave
/dev/sda	/dev/sde	/dev/ada4	/dev/disk0	SATA 0
/dev/sdb	/dev/sdf	/dev/ada5	/dev/disk1	SATA 1
/dev/sdc	/dev/sdg	/dev/da0	/dev/disk2	USB drive
/dev/sr0, /dev/cdrom (/dev/hde)		/dev/cd0	/dev/disk3	CD/DVD drive

Останнім часом багато сучасних Linux-дистрибутивів використовують нарівні з іменами файлів пристроїв їх унікальні ідентифікатори UUID (Universally Unique Identifier), що представляють собою 128-бітові числа виду:

UUID=ce8c343c-7ff7-49c3-9a98-43903a65a5cd

Слід зауважити, що номенклатура іменування носіїв в Unix-подібних ОС, таких наприклад, як FreeBSD відрізняється в номенклатури іменування, яка розглянута вище.

Наприклад, розділ під FreeBSD називається "слайс" (slice). Слайси нумеруються з 1 по 4. Номери слайсів слідують за ім'ям пристрою, які супроводжуються малої *s*, починаючи з 1. Так "da0s1" – це перший слайс першого SCSI пристрої. На одному фізичному диску може бути тільки чотири фізичні слайси. Усередині фізичних слайсів можуть перебувати логічні. Ці додаткові слайси нумеруються починаючи з 5, так, наприклад, що "ad0s5" – це перший додатковий слайс на першому IDE диску. Слайси, "ексклюзивно виділені" фізичні пристрої та інші пристрої містять розділи, представлені буквами від a до h. Ці букви додаються до імені пристрою. "da0a" – це розділ a на першому пристрої da, який "ексклюзивно виділений". "ad1s3e" – це п'ятий розділ в третьому слайсі другого IDE диска. Коди дискових пристроїв в різних Unix можуть дещо відрізнятися.

LVM

В GNU/Linux та OS/2 відомий метод розподілу простору жорсткого диска по логічним томам, розмір яких можна легко змінювати, на відміну від звичайних розділів. Цей метод – *LVM* (Logical Volume Manager – менеджер логічних томів).

LVM є додатковим рівнем абстракції між фізичними/логічними дисками та ФС. Це досягається шляхом розбиття початкових розділів на невеликі блоки (екстенти, зазвичай від 4 до 32 МБайт) та об'єднання їх в єдиний віртуальний том (групу томів) – *volume group*, яка далі розбивається на логічні томи (logical volume).

Для ФС логічний том представлений як звичайний блоковий пристрій, хоча окремі екстенти томи можуть перебувати на різних фізичних пристроях.

LVM збільшує гнучкість ФС, однак, будучи просто проміжним шаром, не скасовує обмеження і використання інших шарів та ускладнює роботу. Тобто також потрібно створювати і змінювати розділи, форматувати їх. Зміна розміру має підтримуватися також і самою ФС.

В GNU/Linux використовується технологія LVM2. Робота з LVM проводиться за допомогою команди LVM або будь-яких менеджерів.

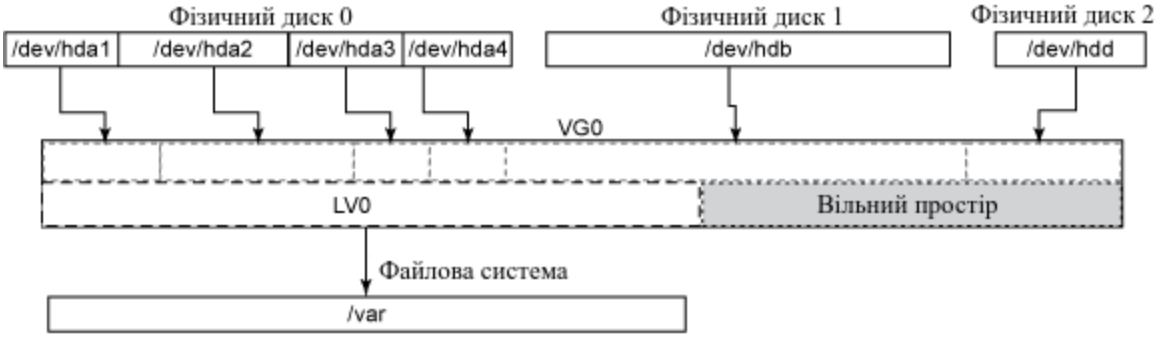


Рис. 7.1. Відповідність фізичних томів файловій системі

На рис. 7.1 представлений приклад LVM. Всі розділи фізичних дисків 0, 1 та 2 додані як фізичні томи в групу томів VG0. До групи входить один логічний том LV0 та залишено вільний простір або для розширення LV0, або для створення нового логічного тому в складі групи. Таким чином, в рамках LVM всі фізичні диски та розділи незалежно від їх розміру і розбиття можуть абстрагуватися і розглядатися як єдине сховище даних.

При роботі з LVM модуль відображення пристроїв створить вузли пристроїв в каталозі /dev наступного вигляду:

/dev/{vg_name}/{lv_name} -> /dev/mapper/{vg_name}{lv_name}

Наприклад: /dev/VG0/LV0 -> /dev/mapper/VG0-LV0

Додаткову інформацію можна отримати за посиланнями:

https://help.ubuntu.com/community/UbuntuDesktopLVM
https://wiki.ubuntu.com/Lvm

Створення розділів та ФС в GNU/Linux-сумісних

Інформацію про розділи жорстких дисків в Unix\Linux можна отримати за допомогою root-команди *fdisk* з опцією `-l` та із зазначенням фізичного пристрою в якості аргументу. Наприклад:

```
fdisk -l /dev/sda
```

або

```
fdisk -l
```

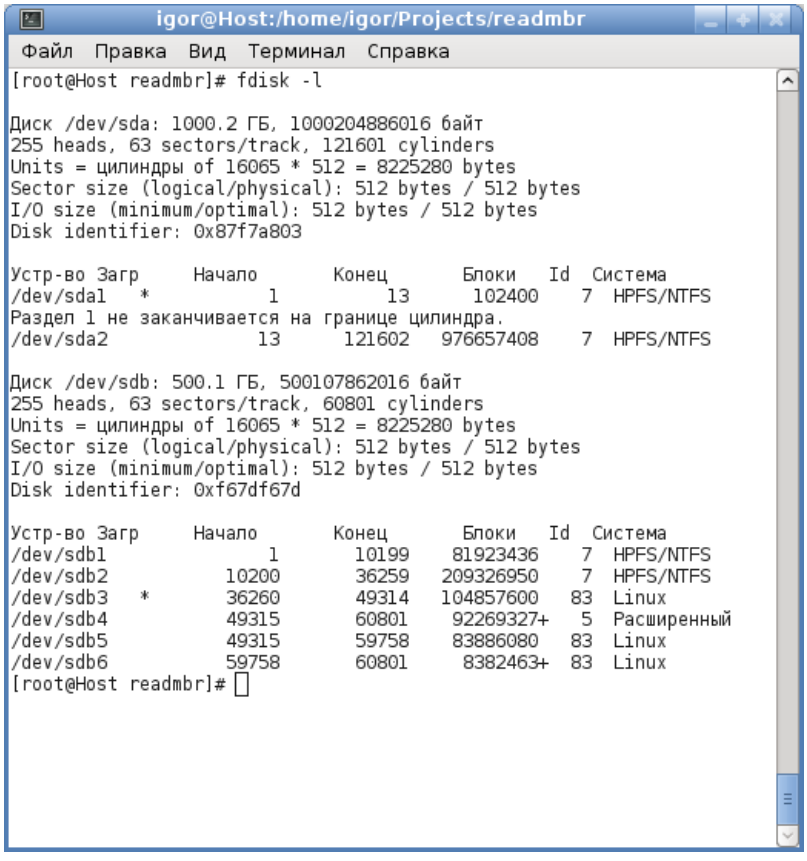


Рис. 7.2. Приклад виведення інформації *fdisk*

Для розбиття фізичного диска на розділи використовуються програми *fdisk*, *cfdisk*, *parted*, редактор дискових розділів з GUI: *GParted* (GNOME Partition Editor) та ін.

Для створення ФС на розділах жорстких дисків, використовуються команди *mkfs*, *mke2fs*, *mkdosfs* та їх розширення. Наприклад, наступна команда створює ФС Linux на першому розділі диска, який підключений як Master до другого каналу IDE:

```
mkfs -t ext2 /dev/hdc1
```

Довідка по команді *mkfs* наприкінці виведе додаткові посилання на різновиди команд *mkfs* в системі. Наприклад, *mkfs.ext4*, *mkfs.vfat* та ін.

Дізнатись які ФС підтримує ядро GNU/Linux в поточний час, можна переглянувши спеціальний файл:

```
cat /proc/filesystems
```

Розділ підкачки (swap-розділ, Linux Swap, за яким закріплений номер 82), створюється за допомогою тих же програм розбивки на розділи. Команда *mkswap* створює на зазначеному swap-розділі спеціальну ФС, а команда *swapon* активізує створений розділ підкачки. Наприклад:

```
mkswap /dev/hdc2
swapon /dev/hdc2
```

Монтування та демонтування ФС, робота з ISO-образами

Для монтування ФС в ієрархію каталогів загальної ФС, призначена команда *mount*. Наприклад, для монтування ФС пристрою /dev/hdc1 в точку монтування (певний каталог) /mnt/disk, використовують наступну команду:

```
mount /dev/hdc1 /mnt/disk
```

За допомогою опції *-t* можна вказати тип ФС, а за допомогою опції *-o* додаткові опції монтування. Наприклад, наступна команда виконує монтування ФС пристрою /dev/sda8 (перший SATA-диск, четвертий логічний розділ, починаючи з 5-го) в точку монтування /mnt/disk2 з додатковими опціями: rw – на читання та запис, для користувача user1 та групи user1, codepage – сприймати символи імен файлів та каталогів в кодовій сторінці 1251 (за замовчанням виставлена кодова сторінка 437), iocharset – вказує в якому кодуванні працює поточна локаль:

```
mount -t vfat /dev/sda8 /mnt/disk2 -o iocharset=utf8,codepage=1251,rw,uid=user1,gid=user1
```

Зауважимо, що частини рядка після опції *-o*, які перелічуються через кому, не повинні містити пробілів!

Ознайомитися з переліком номерів кодових сторінок та їх вмістом можна за посиланнями:

https://en.wikipedia.org/wiki/Code_page
https://en.wikipedia.org/wiki/Windows-1251
https://en.wikipedia.org/wiki/Windows-1252
https://en.wikipedia.org/wiki/Code_page_437

Найбільш відомі типи ФС, які використовуються з опцією *-t* команди *mount* наступні: vfat, ext, ext2, ext3, ext4, nfs, cifs, smbfs, ntfs, ntfs-3g, ufs, iso9660, hfs, hpfs, proc, msdos, umsdos , udf, autofs, reiserfs, tmpfs і ін.

Демонтування пристроїв виконується командою *umount* з вказанням точки монтування. Наприклад:

```
umount /mnt/disk
```

ISO-образи можуть бути вмонтовані в загальне дерево ФС в такий спосіб:

```
mount -t iso9660 -o loop /home/user1/mydisk.iso /mnt/loop
```

Швидко створити ISO-образ з певного дискового пристрою, наприклад з оптичного диску (CD, DVD), можна за допомогою, наприклад, команди (вважається, що пристрій /dev/cdrom та вказаний каталог існують):

```
dd if=/dev/cdrom of=/home/user1/mydisk.iso
```

Приклад запису в ISO-образ певного каталогу (повинен бути встановлений з репозиторію додатковий пакунок *genisoimage*). ISO-образ, в який збережений поточний каталог (файли, розташовані в каталозі будуть розміщені в корені ISO-образу):

```
genisoimage -o ./mydisk.iso -R -J -input-charset utf-8 -V mydisk ./"Курс ОС"
```

Опція -J – використання записів Joliet, для сумісності з Windows. Опція -R – використання записів Rock Ridge для сумісності з Unix/Linux. Опція -V – забезпечує ідентифікатор диска (mydsk) для Windows.

Налаштування ФС для постійного використання

Якщо ФС використовуються постійно, їх монтування можна автоматизувати за допомогою необхідних записів у файлі */etc/fstab*. Приклад вмісту файлу:

```
# <file system>    <mount point>          <type>     <options>     <dump>       <pass>
/dev/hde1          /                      ext2       defaults          0         1
/dev/hde2          none                   swap       sw                0         0
/dev/cdrom         /mnt/cdrom             auto       owner,noauto,ro   0         0
/dev/fd0           /mnt/floppy            auto       owner,noauto      0         0
/dev/hdd           /mnt/zip100.0          auto       noauto,owner      0         0
proc               /proc                  proc       defaults          0         0
UUID=7394f199-e983-446a-9acd-a053b12c6406 /home  ext4  defaults     0         2
```

Четверте поле запису – умови монтування пристроїв. Так, значення його для розділу Linux («defaults») означає, що він монтується автоматично, під час завантаження системи. І за попередньо визначеними умовами монтування може містити файли для виконання (параметр «exec»), бути доступним як для читання, так і для записів («rw»), такий що містить файли пристроїв («dev»), допускати асинхронне (тобто з кешування в оперативній пам'яті) введення/виведення («async») та ін.

Якщо яку-небудь з цих можливостей потрібно заборонити, це слід зробити в явному вигляді або відповідним параметром (наприклад, «ro» – лише для читання, «sync» – синхронний, без кешування введення/виведення), або заперечуючи параметром «no*» (наприклад, параметр «noexec» заблокує запуск файлів на виконання з вказаного пристрою!). Важливим параметром є «suid», який також автоматично включається при монтуванні пристроїв як при вказівці «defaults». Він означає можливість врахування прав доступу до записаних на нього файлів. Існують і інші параметри (див.: *man fstab* та *man mount*).

Останні два поля кожної запису визначають умови резервного копіювання даних пристроїв («<dump>») та перевірку їх ФС при завантаженні («<pass>»).

Поле «<pass>» використовується програмою *fsck* (check and repair a Linux filesystem) для визначення того, чи потрібно перевіряти цілісність ФС. Можливі значення: 0, 1 або 2. Значення 1 слід вказувати тільки для кореневої ФС (з точкою монтування /); для інших ФС, які потрібно перевіряти, слід використовувати значення 2, яке має менш високий пріоритет. Однак, слід також враховувати конкретні особливості ФС.

ФС, для яких значення полів «<dump>» та «<pass>» дорівнює нулю, не резервуються та не перевіряються.

Лекція 8. Завантажувачі ОС

Завантажувач ОС – системне ПО, що забезпечує завантаження ОС безпосередньо після включення комп'ютера. Він надає необхідні засоби для діалогу з користувачем комп'ютера; приводить апаратуру комп'ютера в стан, необхідний для старту ядра ОС і завантаження ядра ОС в оперативну пам’ять; формує параметри, що передаються ядру ОС; передає управління ядру ОС.

Найбільш відомі завантажувачі ОС:

– NTLDR – завантажувач ядра Windows NT;

– Windows Boot Manager (bootmgr.exe, winload.exe) – завантажувач ядра Windows Vista/7/8/10;

– LILO (LInux LOader) – завантажувач, в основному використовується для завантаження ядра Linux;

– GRUB (GRand Unified Bootloader) – застосовується для завантаження ядра Linux та Hurd;

– OS/2 Boot Manager – завантажувач ядра OS/2;

– Loadlin, Syslinux – завантажують Linux з під DOS або Windows;

– BOOTP – застосовується для завантаження по мережі;

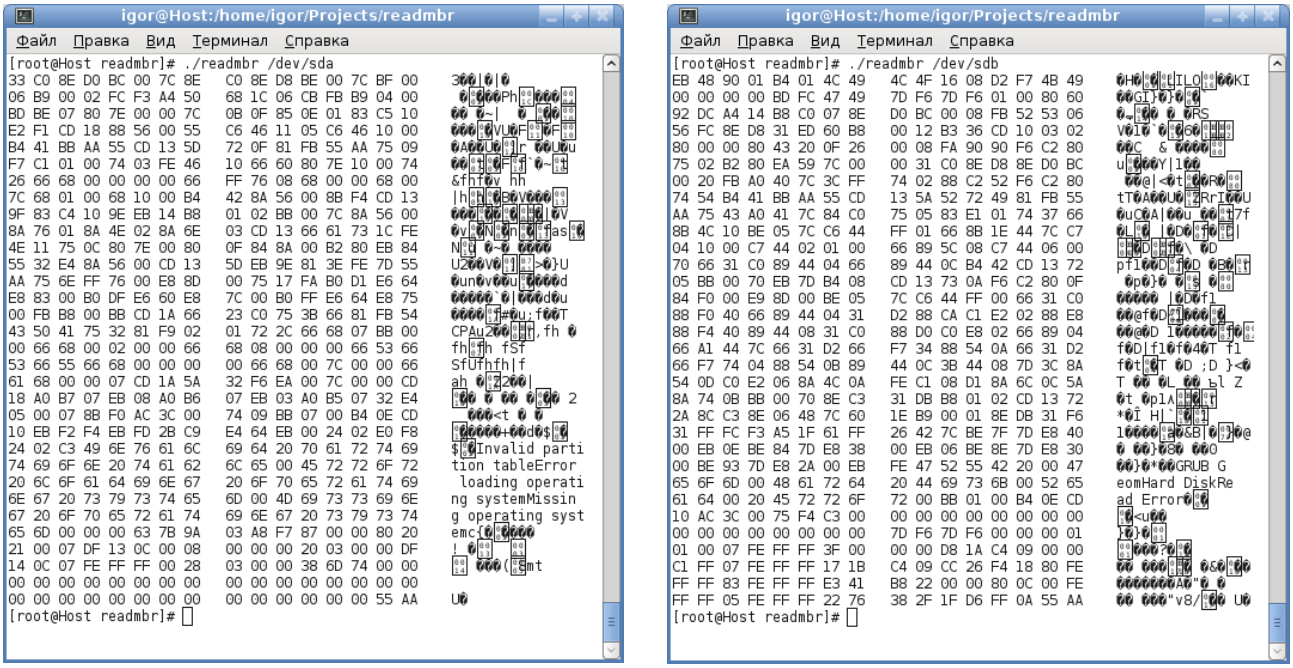
– BootX – завантажувач macOS.

MBR

MBR (Master Boot Record) – головний завантажувальний запис – це код і дані, необхідні для завантаження ОС, розташовані в перших фізичних секторах (найчастіше в найпершому) на жорсткому диску або іншому пристрої для збереження інформації.

MBR містить невеликий фрагмент коду, що виконується, таблицю розділів (partition table) і спеціальну сигнатуру. Мета MBR – не завантаження ОС, а всього лише вибір розділу жорсткого диска, з якого необхідно завантажувати ОС. Завантаження ОС відбувається на більш пізніх етапах. Простір між MBR та першим розділом, який становить близько 32 кб, багато завантажувачів використовують для власних цілей. У таких випадках під MBR розуміють весь завантажувальний код, а для перших 512 байт кажуть, що вони розташовані в MBS (Master Boot Sector) – головному завантажувальному секторі. Для ОС Microsoft поняття MBR та MBS збігаються.

На рис. 8.1 представлений вміст MBR, створений двома різними ОС.



а) б)

Рис. 8.1. Приклад вмісту MBR:
а) створеного MS Windows 7; б) створеного Fedora Linux

Приклад структури MBR (512 байт) для MS Windows:

Зміщення	Розмір	Призначення
000h	Змінний	Код завантажувача
1BBh	04h	Ідентифікатор диску
1BEh	10h	partition 1
1CEh	10h	partition 2
1DEh	10h	partition 3
1EEh	10h	partition 4
1FEh	02h	Ознака таблиці розділів, сигнатура 55h AAh

У процесі завантаження комп'ютера, після закінчення початкового тесту (Power On Self Test – POST), MBR завантажується базової системою введення-виведення (Basic Input/Output System – BIOS) в оперативну пам'ять і далі управління передається завантажувального коду, який знаходиться в MBR.

Адреси збережених байтів даних задаються або в CHS (Cylinder-Head-Sector – циліндр-голівка-сектор), або в LBA (Logical Block Address – логічний адресу блоку) форматах. Конкретний формат визначається типом розділу (Boot ID).

GPT

GUID Partition Table (GPT) – стандартний формат розміщення таблиць розділів на фізичному жорсткому диску. Він є частиною EFI (Extensible Firmware Interface, розширюваний мікропрограмний інтерфейс) – стандарту, покликаного замінити застарілий BIOS. EFI використовує GPT також само, як BIOS використовує MBR.

GUID (Globally Unique IDentifier – глобально унікальний ідентифікатор) – статично унікальний 128-бітний ідентифікатор. Він представлений рядком з 32 шістнадцяткових цифр, розбитих на групи дефісами. GUID можна зустріти в реєстрі Windows, а також при програмуванні під цю ОС в оточені фігурних дужок.

Приклади GUID в GPT:

– розділ даних Linux (або розділ основних даних Windows):

`EBD0A0A2-B9E5-4433-87C0-68B6B72699C7`

– завантажувальний розділ Apple в Mac OS X:

`426F6F74-0000-11AA-AA11-00306543ECAC`

Примітка 1: Порядок запису байтів в написаннях GUID – *little-endian*. Наприклад, GUID системного розділу EFI записаний як:

`C12A7328-F81F-11D2-BA4B-00A0C93EC93B`,

що відповідає послідовності 16 байтів:

`28 73 2A C1 1F F8 D2 11 BA 4B 00 A0 C9 3E C9 3B`

Байти пишуться в зворотній послідовності тільки в перших трьох блоках (`C12A7328-F81F-11D2`).

Примітка 2: Порядок запису байтів *little-endian* (від молодшого до старшого) прийнятий в пам'яті персональних комп'ютерів з x86-процесорами і іноді називається інтеловським або як *VAX order*. Поширений також порядок від старшого до молодшого – *big-endian*. Використовується в мережевих протоколах стеку TCP/IP, а також на комп'ютерах з процесорами IBM 360/370/390, Motorola 68000, SPARC.

Перед GPT в нульовому секторі диска також розміщена MBR. Її функція чисто захисна, оскільки далеко не всі системні утиліти які не підтримують GPT можуть правильно розпізнати такі диски. Щоб уникнути проблем, вказується наявність всього одного розділу, що охоплює весь GPT диск. Системний ідентифікатор для цього розділу в MBR встановлюється в значення *0xEE*, яке вказує, що застосовується GPT.

Завантажувач LILO

LILO не залежить від ФС і тому може завантажувати ОС з жорсткого диска, дискети або з USB. У зв'язку з цим LILO зберігає пункти меню і положення ядер, які завантажуються, безпосередньо в тілі завантажувача і потребує оновлення при кожній зміні конфігурації.

LILO підтримує до 16-ти пунктів меню при завантаженні. Завантажувач підтримує два види завантаження – завантаження ядра Linux з вибором опцій та передача управління іншому завантажувачу. LILO може бути встановлений в головному завантажувальному секторі MBR або завантажувальному сектору розділу. LILO використовує BIOS для доступу до жорстких дисків.

Для управління LILO в Linux використовується відповідна команда: *lilo*. Команда `lilo -q` виводить поточну карту завантаження ОС. Зірочкою вказується поточне початкове завантаження. Команда `lilo -D Win` або `lilo -D linux` змінює порядок завантаження ОС (назви *Win* та *linux* можуть бути іншими – дивитися за допомогою команди `lilo -q`). Команда *lilo* використовує конфігураційний файл */etc/lilo.conf*. Якщо було ручне редагування цього файлу, то потрібне перевстановлення завантажувача командою `lilo`.

Станом на листопад 2020 року, останні зміни датуються 2015 роком.

Завантажувач GRUB

Цей завантажувач є еталонною реалізацією специфікації *Multiboot* та може завантажити будь-яку сумісну з нею ОС – GNU/Linux, FreeBSD, Solaris та ін. Він вміє по ланцюжку передавати управління іншому завантажувачу, що дозволяє завантажувати, наприклад, MS Windows, MS-DOS, OS/2 та ін.

GRUB має велику кількість переваг в порівнянні з LILO, одною з яких є підтримка EFI, наявність інтерактивного командного рядку, модульна структура. Поточна модифікація завантажувача – GRUB 2.

Для основних змін потрібно правити файл */boot/grub/menu.lst* у випадку зі старою версією GRUB або файл */etc/default/grub* та файли в каталогу */etc/grub.d* у випадку з GRUB 2. Дізнатися версію GRUB можна за допомогою команди (в залежності від ОС):

```
grub --version
```

або

```
update-grub --version
```

У випадку з редагуванням при версії GRUB 2 для підтвердження поновлення опцій завантажувача необхідно використовувати команду `update-grub`.

Слід зазначити, що в GRUB 2 нумерація розділів відрізняється від нумерації розділів, що була в старої версії – вона починається з одиниці.

Наприклад, перший розділ першого диска (sda1) іменується як «hd0,1». У старій версії – «hd0,0».

Пункт меню, обраний за замовчуванням, контролюється опцією *default* (файл */boot/grub/menu.lst*) або *GRUB_DEFAULT* (файл */etc/default/grub* в GRUB 2). Час очікування меню на екрані контролюється опцією *timeout* або *GRUB_TIMEOUT* з відповідних файлів.

Завантажувач Bootmgr

Завантажувач в сучасних ОС MS Windows знаходиться та додатковому розділі диску (за замовчанням це перша Primary Partition). За замовчанням у цього розділу немає імені, хоча в програмі керування дисками це ім'я можна задати. Всі файли, які мають відношення до завантажувача й сам завантажувач за замовчанням, мають системні атрибути, які приховують їх відображення в файловому менеджері Windows Explorer та захищають від помилкового запису. Так, наприклад, *bootmgr* має атрибути *System*, *Hidden*, *Read only*.

Важливим є також каталог *Boot* і бінарний файл *BCD* (*Boot Configuration Data*).

Раніше в ОС MS Windows XP завантаження ОС можна було контролювати та налаштовувати опціями в спеціальному текстовому файлі *boot.ini*. Починаючи з Windows Vista всі параметри завантаження зберігаються в спеціальному бінарному файлі *BCD*. Редагування цього файлу проводиться через спеціальну системну утиліту *bcdedit.exe*, яка виконується в режимі адміністратора системи. За замовчанням без параметрів команда `bcdedit` виведе основне поточне налаштування завантажувача сучасної версії Windows.

Примітка: *bcdedit* працює з завантажувачем також в режимі прихованого системного розділу, що був зарезервований системою під завантажувач. Тобто для редагування BCD-файлу не потрібно показувати прихований розділ диску.

Для зміни значення *timeout* (часу на висвітлення стартового меню), можна скористатися командою (встановлює на висвітлення меню 10 секунд):

```
bcdedit /timeout 10
```

Для появи меню завантажувача з опціями та приховання стартового вікна можна ввести команди:

```
bcdedit -set {globalsettings} advancedoptions false
bcdedit -set {globalsettings} optionsedit false
bcdedit -set {globalsettings} bootuxdisabled on
bcdedit -set {bootmgr} displaybootmenu yes
bcdedit -set {bootmgr} timeout 10
```

Висвітлення додаткових опцій, проводиться командою:

```
bcdedit -set {globalsettings} advancedoptions true
```

Наступна команда призвиде до створення файлу-логу завантаження драйверів в файл *C:\Windows\ntbtlog.txt*:

```
bcdedit -set {current} bootlog yes
```

Приклади виведення інформації певних частин BCD:

```
bcdedit
bcdedit /enum {current}
bcdedit /enum {globalsetting}
bcdedit /enum {bootmgr}
bcdedit /enum osloader
bcdedit /enum all
```

Виведення додаткового меню на екран при запуску з можливістю входження в Safe-mode режим:

```
bcdedit -set {globalsettings} advancedoptions true
bcdedit -set {globalsettings} optionsedit false
bcdedit -set {globalsettings} bootuxdisabled off
bcdedit -set {bootmgr} displaybootmenu yes
bcdedit -set {bootmgr} timeout 10
bcdedit -set {current} bootlog no
bcdedit -set {current} sos yes
bcdedit -set {default} bootmenupolicy legacy
reg add hklm\software\microsoft\windows\currentversion\policies\system /t reg_dword /v verbosestatus /d 1
```

Для отримання детальної довідки по команді *bcdedit* можна скористатися посиланнями:

https://docs.microsoft.com/uk-ua/windows-hardware/manufacture/desktop/bcdedit-command-line-options

https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/bcdedit--set

Лекція 9. Архітектура ОС MS Windows

На рис. 9.1 представлена спрощена архітектура ОС MS Windows[3]. Вона не враховує певні особливості, пов'язані з мережевими компонентами та драйверами.

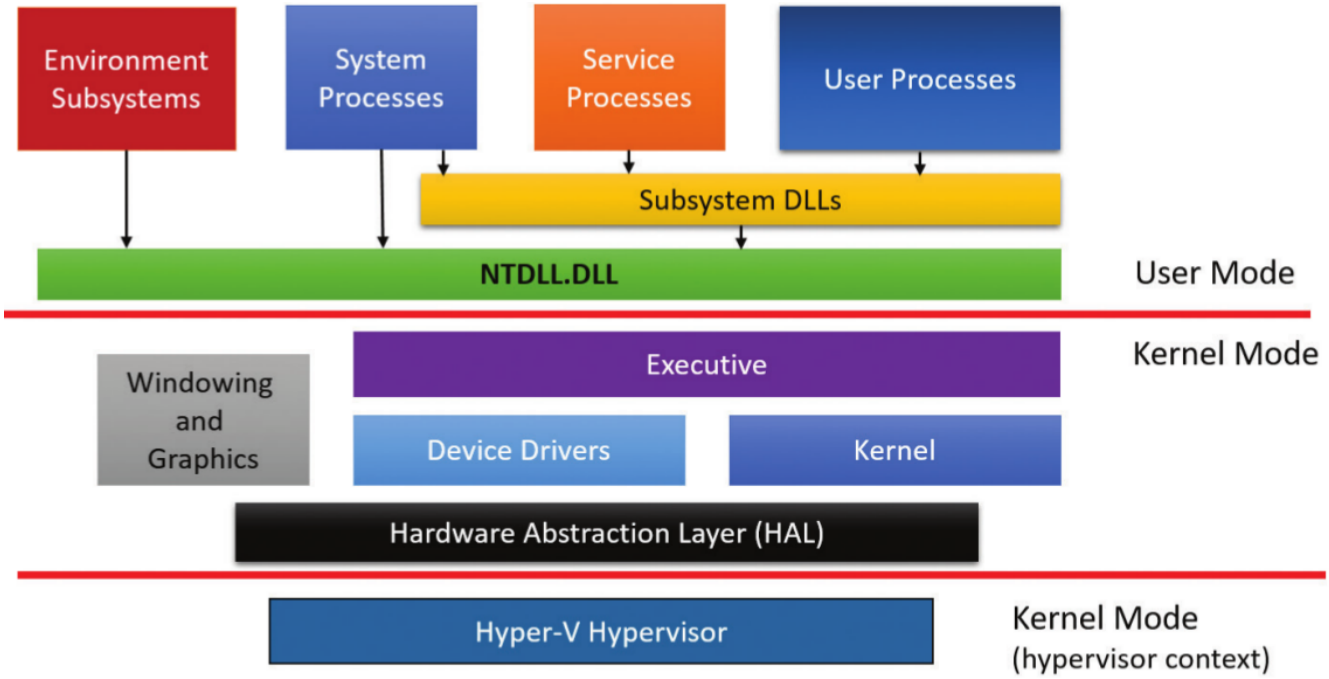


Рис. 9.1. Спрощена архітектура MS Windows

Системні процеси (*System Processes*), процеси служб (*Service Processes*), процеси користувача (*User Processes*) та підсистеми середовища (*Environment Subsystems*) мають власні закриті адресні простори.

Так в режимі користувача виділяють чотири базових типи процесів.

1. Процеси користувача – відносяться до одного з наступних типів: 32-розрядні або 64-розрядні програми Windows.

2. Процеси служб – процеси, що є хостами для служб Windows, наприклад, служби планувальника задач та диспетчера друку. Багато серверних програм (наприклад, MS SQL Server та MS Exchange Server) також включають компоненти, які виконуються як служби.

3. Системні процеси – фіксовані процеси (наприклад, диспетчер сеансів), які не є службами Windows, тобто не запускаються диспетчером служб.

4. Серверні процеси підсистем середи – процеси, що реалізують частину підтримки середи ОС, яка надається користувачу та програмісту.

---

3 В лекції використані матеріали з книги: Pavel Yosifovich, Alex Ionescu, Mark E. Russinovich, and David A. Solomon. Windows Internals. Part 1. System architecture, processes, threads, memory management, and more. 7th Edition. – Microsoft Press, 2017. – 800 p. ISBN-10: 9780735684188, ISBN-13: 978-0735684188.

До категорії режиму ядра відносяться наступні компоненти.

1. Виконавча система (*Executive*) – містить базові сервісні функції ОС: управління пам'яттю, управління процесами та потоками, безпека, введення/виведення, мережева підтримка та міжпроцесні комунікації.

2. Ядро Windows (*Kernel*) – низькорівневі функції ОС: планування потоків, диспетчеризація переривань та виключень, багатопроцесорна синхронізація. Ядро також надає набір функцій та базових об'єктів, які використовуються виконавчою системою для реалізації високорівневих конструкцій.

3. Драйвери пристроїв (*Device Drivers*) – сюди входять як драйвери фізичних пристроїв, які перетворюють виклики функцій введення/виведення користувача в конкретні запити введення/виведення до пристрою, так і драйвери пристроїв, які не мають відношення до фізичного обладнання, наприклад, драйвери ФС або мережеві драйвери.

4. Шар абстрагування обладнання (*Hardware Abstraction Layer*, HAL) – прошарок коду, який ізолює ядро, драйвери пристроїв та інший код виконання Windows від платформно-залежних різниць в роботі обладнання (наприклад, різниць між системними платами).

5. Віконна та графічна система (*Windowing and Graphics*) – реалізація функцій графічного інтерфейсу (GUI), також відомих як функції GDI: робота з вікнами, елементи інтерфейсу користувача та графічне виведення.

6. Рівень гіпервізору (*Hyper-V Hypervisor*) – складається з одного компоненту: гіпервізору. Він містить декілька внутрішніх рівнів та служб: власний диспетчер пам'яті, планувальник віртуальних процесів, управління перериваннями та таймером, функції синхронізації та ін.

Найважливішими системними файлами MS Windows є:

– Ntoskrnl.exe – виконавча система та ядро;

– Hal.dll – HAL;

– Win32k.sys – частина підсистеми Windows режиму ядра (GUI);

– Hvix64.exe (Intel), Hvax64.exe (AMD) – гіпервізор;

– Windows\System32\drivers*.sys – основні файли драйверів;

– Ntdll.dll – внутрішні допоміжні функції та заглушки диспетчеризації системних сервісних функцій;

– Kernel32.dll, Advapi32.dll, User32.dll, Gdi32.dll – бібліотеки основних підсистем.

Одна з головних цілей при проектуванні Windows полягала в тому, щоб система добре працювала на багатопроцесорних системах. Windows є ОС з симетричною багатопроцесорною обробкою (Symmentic Multiprocessing, SMP) – головного CPU в системі не існує. Планування виконання як системних процесів, так і процесів користувача здійснюється між всіма CPU системи.

В залежності від версії Windows, існують певні обмеження на використання кількості процесорів та об’єму пам’яті (таблиця 9.1).

Таблиця 9.1

Обмеження по кількості процесорів та об’єму пам’яті в деяких випусках Windows

Версія ОС	*Кількість фізичних CPU, що підтримуються (32-розрядна версія)*	*Об’єм фізичної пам’яті, що підтримуються (32-розрядна версія)*	*Кількість логічних/фізичних CPU, що підтримуються (64-розрядна версія)*	*Об’єм фізичної пам’яті, що підтримуються (64-розрядна версія)*
Windows 7 Home Basic	1	4 Гб	1 фізичний	8 Гб
Windows 7 Pro	2	4 Гб	2 фізичних	192 Гб
Windows 7 Ultimate	2	4 Гб	2 фізичних	192 Гб
Windows 8	1	4 Гб	1 фізичний	128 Гб
Windows 8 Pro	2	4 Гб	2 фізичних	512 Гб
Windows 8 Enterprise	2	4 Гб	2 фізичних	512 Гб
Windows 10 Home	1	4 Гб	1 фізичний	128 Гб
Windows 10 Pro	2	4 Гб	2 фізичних	2 Тб
Windows 10 Enterprise	2	4 Гб	4 фізичних	6 Тб
Windows Server 2012 R2 Standard	–	–	2 фізичних (на ліцензію)	4 Тб
Windows Server 2012 R2 Essentials	–	–	2 фізичних (на ліцензію)	64 Гб
Windows Server 2016 Standard	–	–	64 фізичних; логічних – необмежено	24 Тб
Windows Server 2016 Datacenter	–	–	64 фізичних; логічних – необмежено	24 Тб
Windows Server 2019 Standard	–	–	64 фізичних; логічних – необмежено	24 Тб
Windows Server 2019 Datacenter	–	–	64 фізичних; логічних – необмежено	24 Тб

Підсистеми середи та DLL середовища

Підсистема середовища потрібна, щоб надавати прикладним програмам деяку підмножину сервісних функцій базової виконавчої системи Windows.

Різні підсистеми відкривають доступ до різних підмножинам вбудованих сервісних функцій Windows. Це означає, що в програмі, яка побудована на базі

однієї підсистеми, можна робити те, що не може бути зроблено в програмі на базі іншої підсистеми.

Кожна програма (exe-файл) зв'язується з однією і тільки з однією підсистемою. При виконанні програми код створення процесу аналізує код типу підсистеми в заголовку програми, щоб оповістити правильну підсистему про новий процес. Код типу задається параметром компонувальника Microsoft Visual Studio (рис. 9.2): */SUBSYSTEM* (або за допомогою запису *SubSystem* в сторінці властивостей Linker/System властивостей проекту).

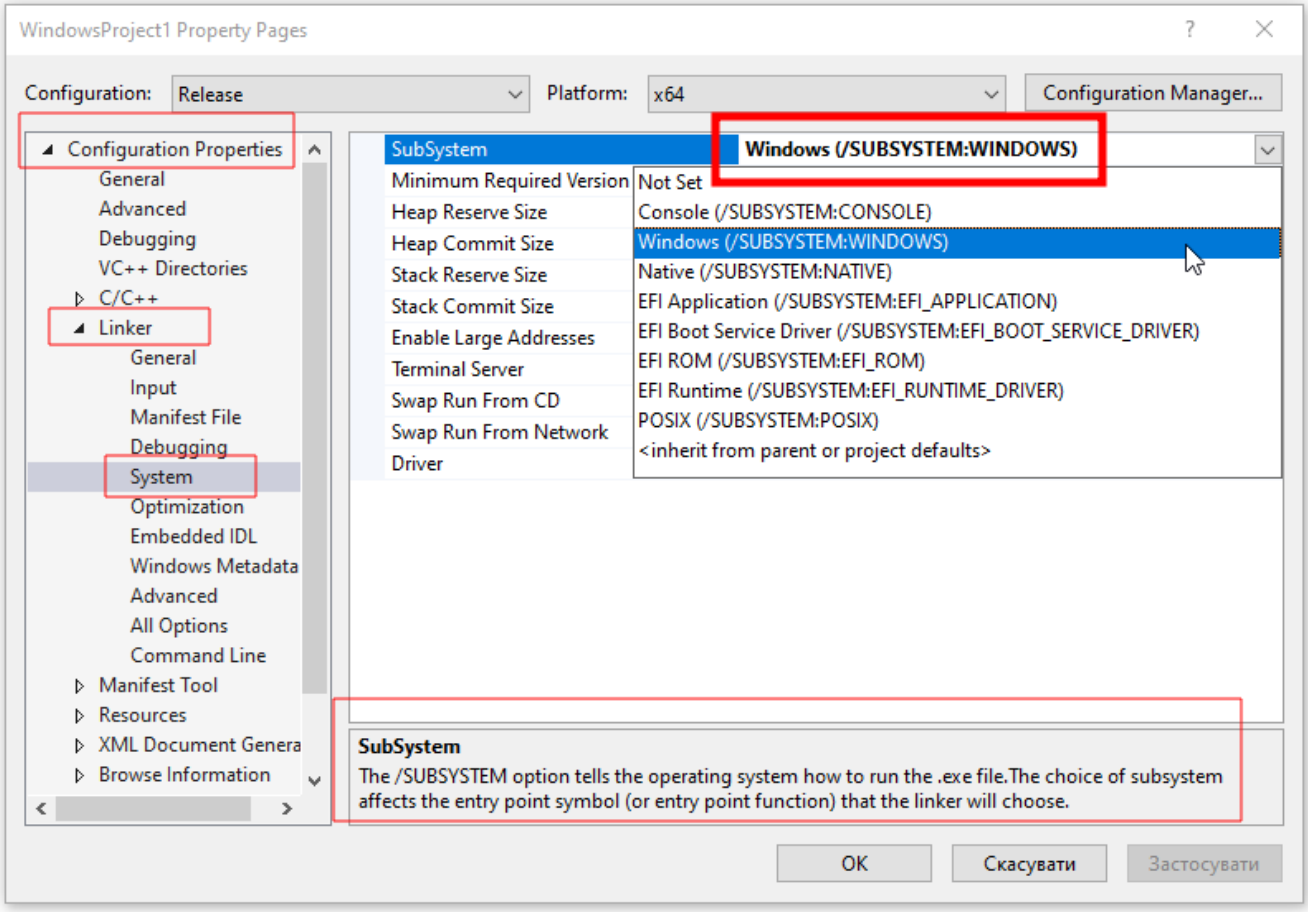


Рис. 9. 2. Встановлення типу підсистеми у властивостях проекту при розробці Windows програми в середовищі розробки MS Visual Studio Community 2019

При збірці Win32-програми в середовищі IDE Code::Blocks, автоматично при використанні компілятора GCC проекту MinGW-W64 буде прописана опція *-mwindows*, яка вказує на створення програми Windows з GUI. Тобто фактично значення SUBSYSTEM буде визначено типом проекту, який обраний для створення та додатковим пунктами в діалозі налаштувань цілей збірки (рис. 9.3).

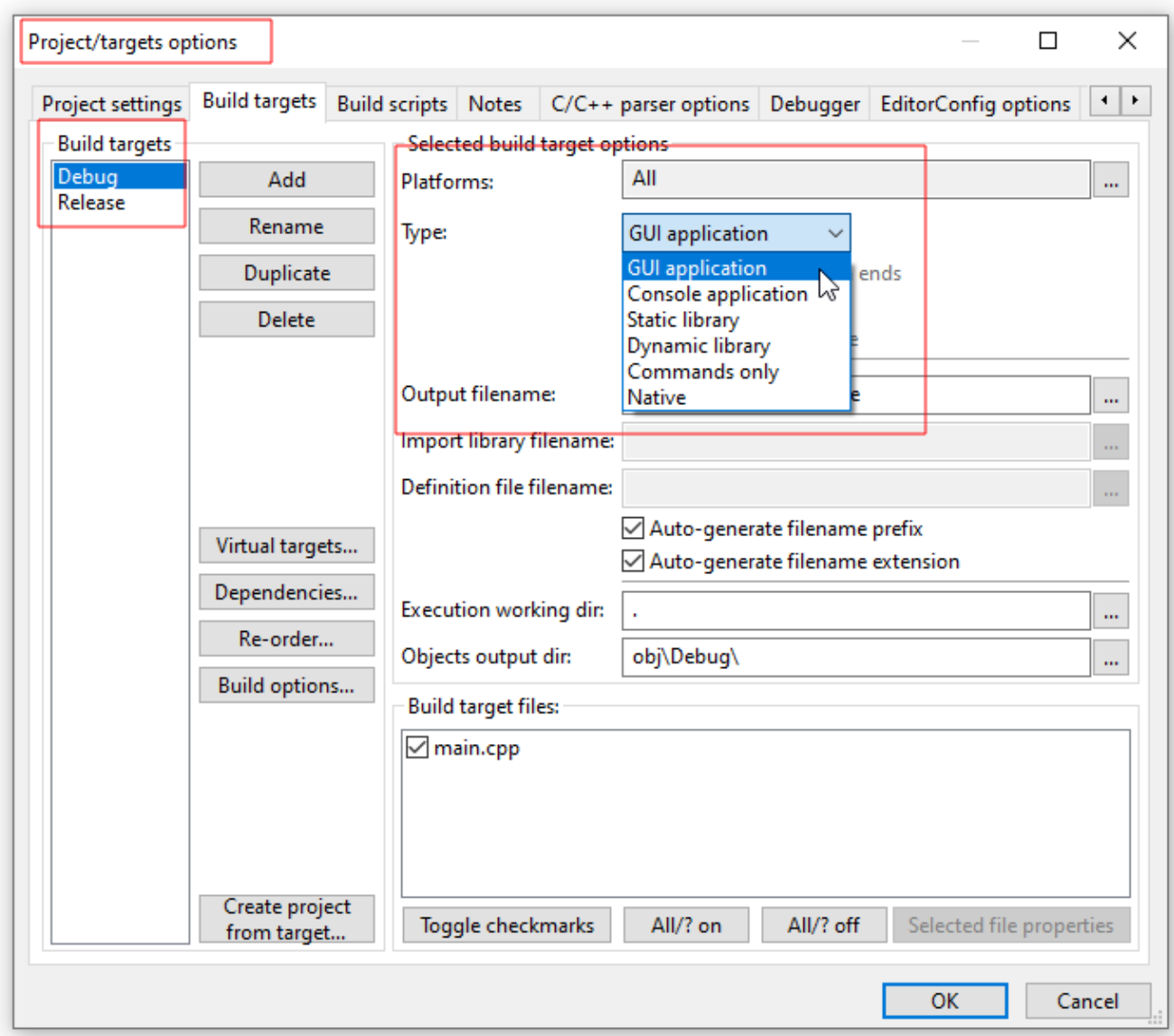


Рис. 9.3. Обрання типу створюваного додатку в середовищі Code::Blocks

Призначені для користувача програми не викликають системні функції Windows безпосередньо. Замість цього вони проходять через одну або кілька DLL підсистем. Ці бібліотеки експортують документований інтерфейс, який може викликатися програмами, скомпонованими з цієї підсистемою. Наприклад, DLL-бібліотеки підсистеми Windows (такі, як Kernel32.dll, Advapi32.dll, User32.dll та Gdi32.dll) реалізують функції Windows API. DLL-бібліотека підсистеми SUA (Subsystem for Unix-based Applications; Psxdll.dll) використовується для реалізації функцій SUA API (у версіях Windows з підтримкою стандарту POSIX).

Коли програма викликає функцію з DLL підсистеми, можливі три варіанти.

1. Функція повністю реалізована в призначеному для користувача режимі в DLL підсистемі. Іншими словами, процесу підсистеми середовища

повідомлення не надсилається, і сервісні функції виконавчої системи Windows не викликаються. Функція виконується в режимі користувача, а результати повертаються стороні, яка викликає функцію. До числа таких функцій відносяться *GetCurrentProcess( )* (завжди повертає -1 – значення, яким позначається поточний процес у всіх функціях, пов'язаних з процесами) та *GetCurrentProcessId( )* (ідентифікатор працюючого процесу не змінюється, тому значення читається з кешу, щоб обійтися без виклику ядра).

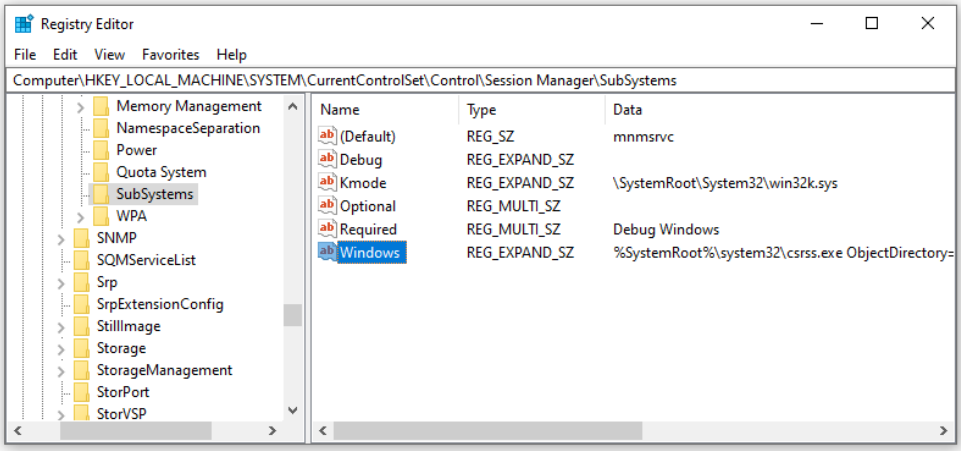
2. Функція вимагає одного або декількох викликів до виконавчого середовища Windows. Наприклад, функції Windows *ReadFile( )* та *WriteFile( )* вимагають виклику внутрішніх (і недокументованих для використання в режиму користувача) системних функцій введення/виведення Windows *NtReadFile( )* та *NtWriteFile( )* відповідно.

3. Функція вимагає виконання певної роботи в процесі підсистеми середовища (процеси підсистеми середовища, що працюють в режимі користувача, відповідають за підтримання стану клієнтських додатків, що працюють під їх керуванням). У цьому випадку запит до підсистеми середовища проводиться за допомогою повідомлення *ALPC* (Asynchronous Local Inter-Process Communication або Advanced Local Procedure Call – спеціальний механізм за допомогою якого повідомлення передаються між клієнтським та серверним процесами, що працюють на одному комп'ютері), яке відправляється підсистемі для виконання деякої операції. DLL підсистеми очікує відповіді перед тим, як повертати управління на сторону виклику.

Деякі функції є комбінаціями другого і третього пунктів. Наприклад, функції Windows *CreateProcess( )* та *ExitWindowsEx( )*.

Підсистеми запускаються процесом диспетчера сеансів (Smss.exe). Інформація для запуску підсистеми зберігається в розділі реєстру (рис. 9.4):

```
HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\SubSystems
```




```
%SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows SharedSection=1024,20480,768
Windows=On SubSystemType=Windows ServerDll=basesrv,1 ServerDll=winsrv:UserServerDllInitialization,
3 ServerDll=sxssrv,4 ProfileControl=Off MaxRequestThreads=16
```

Рис. 9.4. Приклад заповнення значеннями певної властивості диспетчера сеансів в ОС MS Windows 10 Professional

Параметр Windows визначає файл підсистеми Windows *Csrss.exe* (*Client/Server Runtime* Subsystem). Параметр реєстру *Kmode* містить ім'я файлу, який містить частину підсистеми Windows режиму ядра – win32k.sys. Параметр *Debug* залишений для сумісності з версією Windows XP та не використовується. Параметр *Optional* за замовчанням пустий та може містити підсистеми, які завантажуються за потреби.

Гіпервізор

Останні досягнення в сфері програм та програмного забезпечення, такі як поява хмарних сервісів та поширення пристроїв інтернету речей (IoT), привели до того, що ОС та виробникам устаткування доводиться шукати більш ефективні способи віртуалізації інших «гостей» ОС на обладнанні машини, будь то розміщення численних мешканців ферми серверів, робота ста ізольованих Web-сайтів на одному сервері або ж можливість тестування розробниками десятків різновидів ОС без покупки спеціалізованого обладнання.

Потреба в швидкою, ефективної та безпечної віртуалізації породила нові моделі обчислень і підходів до проектування програмних продуктів. Дуже часто сучасні програми працюють в контейнерах, які забезпечують повну ізоляцію віртуальних машин, призначених виключно для виконання одного стеку додатків або інфраструктури (як приклад можна привести технологію Docker, яка підтримується платформою MS Windows); таким чином, кордони між керуючою системою і гостями піднімаються на новий рівень.

Для надання сервісу віртуалізації майже всі сучасні рішення користуються послугами *гіпервізору* – спеціалізованого, дуже привілейованого компонента, який забезпечує віртуалізацію та ізоляцію всіх ресурсів машини, від віртуальної і фізичної пам'яті до переривань пристроїв, навіть пристроїв з різноманітними фізичними інтерфейсами.

Наприклад, в основі функціональності клієнта Hyper-V, що надається в Windows 8.1 і вище, лежить гіпервізор Hyper-V. Всі конкуруючи продукти – Xen, KVM, VMware та VirtualBox – реалізують власні гіпервізори, кожен з яких має свої переваги та недоліками.

Через свою привілейовану природу і через рівень доступу навіть більшого, ніж у самого ядра, гіпервізор має очевидну перевагу перед гостьовими екземплярами інших ОС: він може забезпечувати захист та збір керуючої інформації одного екземпляру хоста для надання гарантій, які перевищують можливості ядра. У Windows 10 компанія Microsoft використовує гіпервізор Hyper-V для надання нового набору сервісів віртуалізаційної безпеки VBS (Virtualization-Based Security). Крім того, гіпервізор Hyper-V забезпечує деякі ключові заходи захисту ядра від експлойтів та інших видів атак.

Лекція 10. Архітектура macOS

macOS являє собою поєднання оригінальних закритих програмних технологій компанії Apple з відкритими технологіями, що стали промисловими стандартами. Базова архітектура macOS[4] показана на рис. 10.1 та має ієрархічну структуру.

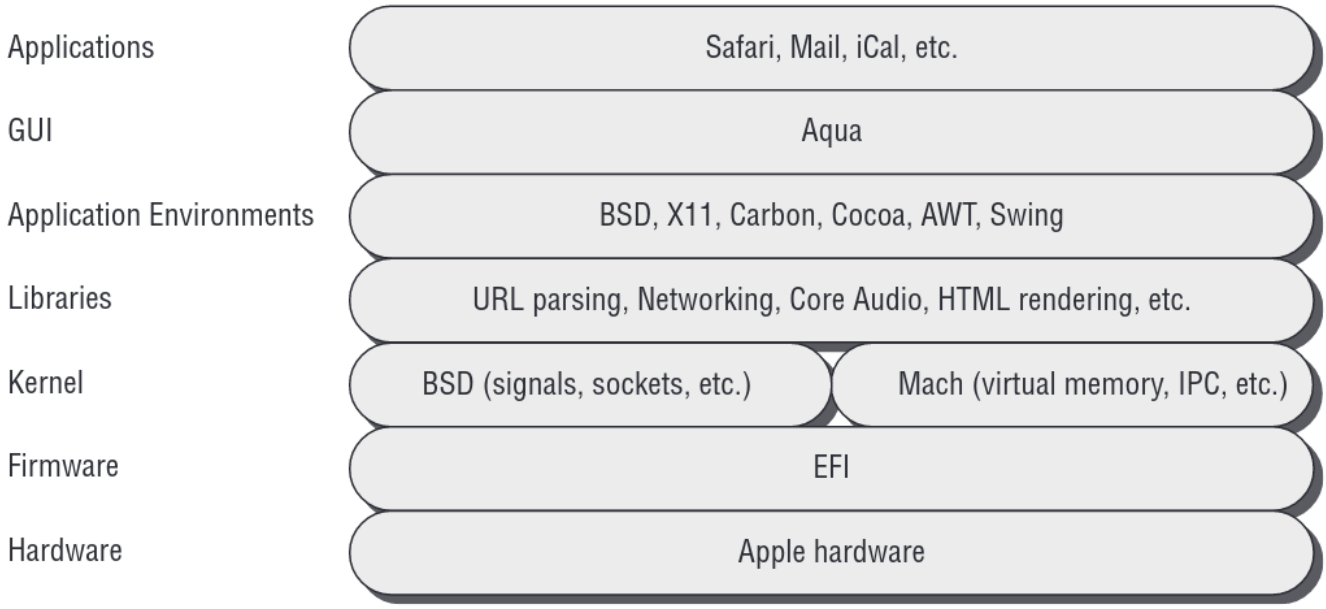


Рис. 10.1. Базова архітектура macOS

macOS побудована на базі мікроядра під назвою *Darwin*. Усередині *Darwin* знаходиться "ядро в ядрі" – мікроядро *Mach*.

Mach є "класичним" мікроядром, яке розробляється університетом Carnegie Mellon з 1985 року. Воно створено на основі BSD і є основою для ряду BSD Unix-подібних систем.

BSD (Berkeley Software Distribution) – система розповсюдження програмного забезпечення в початкових кодах, створена для обміну досвідом між навчальними закладами. Особливістю пакетів ПЗ BSD була спеціальна ліцензія BSD, яку коротко можна охарактеризувати так: весь вихідний код – власність BSD, всі правки – власність їх авторів[5]. На даний час термін BSD найчастіше вживається як синонім BSD-UNIX – загальна назва варіантів Unix, які є похідними від дистрибутивів університету Берклі. Наприклад, це ОС: FreeBSD, NetBSD, OpenBSD, ClosedBSD, MirBSD, DragonFly BSD, PC-BSD, DesktopBSD, SunOS, TrueBSD, Frenzy, Ultrix та частково XNU (ядро macOS, iOS, tvOS, watchOS, CarPlay, Darwin).

4 За матеріалами книги Charlie Miller, Dino Dai Zovi. The Mac Hacker's Handbook, 1st Edition. – Wiley, 2009. – 384 p. ISBN-10: 0470395362, ISBN-13: 978-0470395363.

5 За матеріалами вільної енциклопедії Wikipedia.

Відмінності від «класичної» системи Unix укладені в системі друку, ФС, відсутності перемикання рівнів виконання та командній оболонці. Команди, які призначені для користувача, практично ідентичні.

Aqua – дружній графічний інтерфейс macOS.

Ядра *Mach* та *Darwin* є відкритими продуктами і підтримуються організацією Open Group. *Mach* підтримує основні низькорівневі функції управління ресурсами, такі як:

– управління нитками;

– призначення ресурсів для процесів;

– управління віртуальною пам'яттю;

– обмін повідомленнями між завданнями;

– управління процесорами, I/O та іншими ресурсами.

Мікроядро *Darwin*, що є розширенням *Mach*, містить наступні основні компоненти:

– інструменти I/O – об'єктно-орієнтований каркас для розробки драйверів пристроїв та забезпечення для них необхідної інфраструктури;

– ФС – ґрунтується на віртуальній ФС VFS та забезпечує можливість додавати нові ФС;

– розширені засоби Network Kernel Extensions (NKE), що дозволяють розробникам як додавати підтримку нових протоколів, так і розширювати функціональність вже існуючих;

– BSD – оболонка BSD 4.4 навколо ядра, що включає в себе API POSIX та забезпечує модель процесів, базові політики безпеки та інші функції.

Ядро maxOS скорочено позначається як *XNU* – X is Not Unix, оскільки ядро ОС Apple скомбіновано з двох джерел, і лише його частина має відношення до Unix (рис. 10.2).

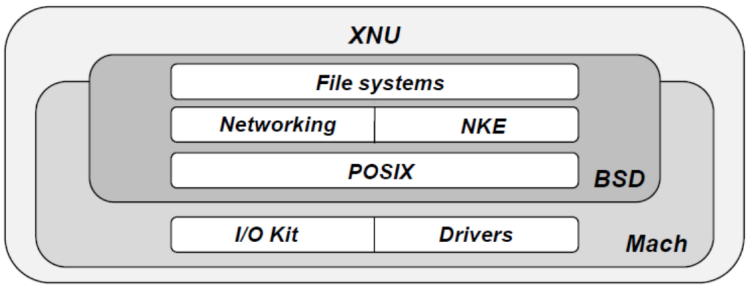


Рис. 10.2. Схематичне представлення ядра macOS

Служби ядра містять ті системні сервіси, які не пов'язані з GUI. Основними компонентами цих служб є:

– менеджер середовища Carbon – є загальносистемним та забезпечує низькорівневий сервіс для всіх прикладних середовищ, таких як: Component Manager, File Manager, Memory Manager, Multiprocessing Services та ін.

– Core Foundation – каркас, що забезпечує базові програмні служби, корисні для більш високих рівнів ПЗ. Він же забезпечує роботу різних сервісів системи.

– Open Transport – основні модулі для роботи в мережі.

Крім служб ядра, macOS дозволяє також використовувати розширення ядра. Система завантажує їх динамічно в міру необхідності. Часто в таких випадках говорять про гібридне ядро, однак експерти відносять ядро macOS скоріше до монолітного через особливості його будови.

Головне завдання прикладних служб macOS – забезпечення функціонування GUI.

Прикладні середовища macOS складаються з каркасів (framework), бібліотек та сервісів, що забезпечують виконання програм в різних моделях API. Наприклад:

– Carbon – розвиток спеціального API під Mac OS X для зворотної сумісності з Mac OS 8 та 9, 32-бітний (помічений як deprecated з 2012 р. з версії Mac OS X 10.8 Mountain Lion);

– Cocoa – об'єктно-орієнтоване середовище для мов Java та Objective-C;

– Java – дозволяє розробляти та виконувати в macOS додатки та аплети Java.

– Swift – багатопарадигмова компільована мова програмування, розроблена компанією Apple для того, щоб співіснувати з Objective-C та бути стійкішою до помилкового коду.

Та інші.

Станом на 2020 рік мова програмування Swift стала основною мовою розробки програм під платформ macOS та iOS.

Для розробки на Swift, C/C++, Objective-C потрібно встановити спеціалізоване середовище розробника Xcode.

Відправною точкою для розробників в середовищах Apple є ресурс:

https://developer.apple.com/

macOS додає власні спеціальні каталоги до дерева Unix починаючи з кореня системи:

/Applications – за замовчуванням в цей каталог встановлюються всі програми у системі;

/Developer – якщо встановлено Xcode, то це точка встановлення за замовчуванням для всіх інструментів розробника;

/Library – файли даних, довідка, документація тощо для системних програм;

/Network – віртуальний каталог для виявлення та доступу до мережевого вузла;

/System – використовується для системних файлів та містить підкаталог Library, який в свою чергу містить практично всі основні компоненти системи, такі як фреймворки (/System/Library/Frameworks), модулі ядра (/System/Library/Extensions), шрифти тощо;

/Users – домашня директорія для користувачів, де кожний користувач має свій власний каталог;

/Volumes – точка монтування для знімних носіїв та мережевих ФС;

/Cores – каталог дампів ядра, якщо така можливість увімкнена (основні дампи створюються при аварійному завершенні процесів, якщо це дозволяє команда ulimit, і містять основний образ віртуальної пам'яті процесу).

Bundles (набори, пакети) – це ключова ідея в macOS, яка виникла ще в NeXTSTEP (об'єктно-орієнтована, багатозадачна ОС компанії NeXT Computer, яка була придбана в 1997 році компанією Apple і надалі увійшла в основу при розробці власних ОС Apple) та у час використання вже мобільних додатків стала фактичним стандартом. Концепція пакету є основою для додатків, а також для фреймворків, плагінів, віджетів і навіть розширень ядра, упакованих у пакети. В залежності від того, до якої категорії входять пакети, вони мають відповідні розширення. Наприклад: .app, .framework, .kext, .plugin, .docset та інші.

Встановлювач програм в macOS найчастіше працює з файлами програм, дистрибутивів певного ПЗ, які містяться у файлах з розширеннями .dmg, .pkg та .xip.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Andrew S. Tanenbaum. Modern Operating Systems, 3rd Edition. – Pearson, 2007. – 1104 p. ISBN-10: 0136006639, ISBN-13: 978-0136006633.

2. Pavel Yosifovich, Mark Russinovich, David Solomon, Alex Ionescu. Windows Internals, Part 1: System architecture, processes, threads, memory management, and more, 7th Edition – Microsoft Press, 2017. – 800 p. ISBN-10: 9780735684188, ISBN-13: 978-0735684188.

3. Evi Nemeth. UNIX and Linux System Administration Handbook, 5th Edition / Evi Nemeth, Garth Snyder, Trent Hein, Ben Whaley, Dan Mackin. – Addison-Wesley Professional, 2017. – 1232 p. ISBN-10: 0134277554, ISBN-13: 978-0134277554.

4. Michael Kerrisk. The Linux Programming Interface: A Linux and UNIX System Programming Handbook. – No Starch Press, 2010. – 1552 p. ISBN-10: 1593272200, ISBN-13: 978-1593272203.

5. Kevin Wilson. MacOS Fundamentals: Catalina Edition: The Step-by-step Guide to Using your Mac. – Independently published, 2019. – 335 p. ISBN-10: 1708721118, ISBN-13: 978-1708721114.

6. Chris Johnson, Jayant Varma. Pro Bash Programming, Second Edition: Scripting the GNU/Linux Shell, 2nd Edition. – Apress, 2015. – 279 p. ISBN-10: 1484201221, ISBN-13: 978-1484201220.

7. Lee Holmes. Windows PowerShell Cookbook: The Complete Guide to Scripting Microsoft's Command Shell, Third edition. – O'Reilly Media, 2013. – 1036 p. ISBN-10: 1449320686, ISBN-13: 978-1449320683.

8. Зайцев, В. Г. Операційні системи [Електронний ресурс] : навчальний посібник для студентів спеціальності 123 «Комп’ютерна інженерія» / В. Г. Зайцев, І. П. Дробязко ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 2,22 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2019. – 240 с. – Назва з екрана.

9. Погребняк Б.І. Операційні системи: навч. посібник / Б.І.Погребняк, М.В.Булаєнко; Харків. нац. ун-т міськ. госп-ва ім. О.М. Бекетова. – Харків: ХНУМГ ім. О.М. Бекетова, 2018. – 104с.

10. Федотова-Півень І.М. Операційні системи: навчальний посібник. [за ред. В.М. Рудницького] / І.М. Федотова-Півень, І.В. Миронець, О.Б. Півень, С.В. Сисоєнко, Т.В. Миронюк; Черкаський державний технологічний університет. – Харків: ТОВ «ДІСА ПЛЮС», 2019. – 216 с.

11. Микитишин А.Г. Операційні системи: консп. лекц. / укл. А.Г. Микитишин, І.В. Чихіра. – Тернопіль: ТНТУ імені Івана Пулюя, 2016. – 107 с.

ДОДАТОК А. Деякі корисні консольні команди Unix/Linux-подібних ОС

Зауваження: більшість наведених нижче команд містять різні ключі. Крім того, в різних версіях Unix/Linux-подібних ОС ключі та їх значення у командах можуть відрізнятися. Вкрай рекомендується перед виконанням команд з ключами для рекурсивного використання або іншими можливостями, ознайомитися з ними за допомогою команди: *man команда*.

Команди ls, ls -l, ls -la, ls -ld, ls -li, ls -i, ls -d

Команда виводить список файлів і каталогів поточного каталогу. Опція *-l* дає можливість виконати детальне виведення, а опція *-la* дає можливість перегляду інформації по прихованим файлам. Опція *-d* дозволяє видавати імена каталогів, як ніби вони звичайні файли, тобто без показу їх вмісту. Опція *-i* передує виведення для кожного файлу його номером inode. Приклад виведення команди *ls -l*

```
drwxr-xr-x    2    root    root    112     2007-04-22 18:08    ifplugd/
-rw-r--r--    1    root    root    4476    2006-10-19 09:38    inetd.conf
 lrwxrwxrwx   1    root    root    12      2007-10-13 10:43    init.d -> rc.d/init.d/

1    2        3    4       5       6          7                8
```

1 – тип файлу (перший символ, див. нижче);
2 – права доступу (6-ть символів, див. нижче);
3 – кількість жорстких зв'язків;
4 – власник-користувач;
5 – власник-група;
6 – розмір в байтах;
7 – дата і час створення або останньої модифікації;
8 – ім'я файлу (каталогу).

Типи файлів:

b – блоковий файл пристрою;
c – символьний файл пристрою;
d – каталог;
l – файл символічної зв'язку;
s – сокет;
p – канал, FIFO;
- – звичайний файл.

Права доступу:

rwx rwx rwx
1 2 3

1 – права власника-користувача;
2 – права власника-групи;
3 – права інших користувачів;
- – відсутність прав (яких саме – залежить від знакомісця).
r – право на читання;
w – право на запис;
x – право на виконання (або відкриття каталогу)

Команда cd
Команда зміни поточного каталогу. Приклади:
cd ~
cd $HOME
cd /
cd /usr/bin
cd ..
cd ../ ..

Команда pwd
Команда визначення поточного каталогу. Приклад:

pwd

Команда which
Команда відображає повний шлях до вказаних команд або сценаріїв. Команда має низку ключів. Наприклад, за допомогою ключа *-a* команда виводить всі співпадаючі файли для виконання за вмістом в змінній оточення PATH, а не тільки перший з них. Приклади:

which java
which -a java

Команда touch
Команда змінює час останнього доступу та/або час останньої модифікації кожного заданого файлу. Якщо заданий файл не існує, то він створюється (якщо не задана опція -c). наприклад:

```
touch ./mynewfile
```

Команда cp

Команда копіювання файлів (каталогів). З ключем -r виконує рекурсивне копіювання каталогу (з усіма його підкаталогами). Приклади:

```
cp filename newcopyfilename
cp filename1 filename2 filename3 /home/igor/temp
cp filename* /home/igor/temp
cp /etc/samba/* /home/igor/temp
cp -r /home/user1 /docs/mnt/share
```

Команда mkdir

Команда створює новий каталог. Приклад:

```
mkdir mynewdir
```

Команда mv

Команда переміщення/перейменування файлів (каталогів). Приклади:

```
mv filename newfilename
mv filename /home/user1/targetdir
```

Команда rmdir

Видалення порожнього каталогу. Приклад:

```
rmdir mydeldir
```

Команда rm

Команда видалення файлів/каталогів. З опцією -r виконує рекурсивне видалення каталогу (з усіма підкаталогами). З опцією -f ігнорує імена неіснуючих файлів (каталогів) і не видає інтерактивних запитів користувачеві. Приклади:

```
rm filename
rm filename *
rm -f filename
rm -r dirname
```

Команда cat

Команда виведення вмісту файлу на консоль (за замовчуванням). Для перенаправлення виведення використовуйте символи '>' (для створення нового файлу) або '>>' (для додавання в кінець файлу). Приклади:

```
cat filename
cat filename > newfile
cat filename >> addtofile
cat filename1 filename2
cat filename1 filename2 > totalfile
```

Команда more

Команда забезпечує посторінкове виведення на екран. Як правило, застосовується з іншими командами при організації конвеєра команд. Приклади:

```
more ./readme
ls -l | more
ps -ef | more
cat bigfile | more
more bigfile
```

Команда less

Консольна програма для перегляду вмісту текстових файлів з можливістю прокрутки. Набагато швидше, ніж *vi*, здійснює перегляд великих файлів. Має ряд опцій для додаткових можливостей. Приклади:

```
less ./readme
```

Команда ln

Команда створює зв'язок (за замовчуванням жорсткий). Із зазначенням ключа -s – символічний зв'язок. Приклади:

```
ln mysourcefile newlinkfile
ln -s mysourcefile newslinkfile
```

Команда chmod

Команда зміни прав доступу до файлу/каталогу. Приклади:

```
chmod g+w ./test.txt
chmod g+w, o-r ./test.txt
```

```
chmod a+r ./test.txt
chmod a+rwx ./test.txt
```

Команда chown

Команда зміни власника-користувача (можливо і власника-групи) файла/каталога. При вказівці ключа -R виконується рекурсивна зміна прав на всі підкаталоги зазначеного каталогу. Приклади:

```
chown igor ./test.txt
chown -R user2 ./otherdirectory
chown igor:users ./test.txt
```

Команда chgrp

Команда зміни власника-групи файлу/каталогу. При вказівці ключа -R виконується рекурсивне зміна прав на всі підкаталоги зазначеного каталогу. Приклади:

```
chgrp users ./test.txt
chgrp -R users ./otherdirectory
```

Команда uname

Команда виводить інформацію про систему. З опцією -a виводить повну інформацію. З опцією -r – про версію ядра. Приклад:

```
uname
uname -a
uname -r
```

Команда eject

Команда здійснює вилучення оптичного CD/DVD-диску з пристрою. Приклад:

```
eject
```

Команда startx

Команда завантажує графічну оболонку, якщо вхід в систему був проведений в текстовому режимі консолі і налаштовані скрипти ініціалізації графічного старту. Оболонка буде завантажена тільки при правильному налаштуванні графічної підсистеми X Window. Приклад:

startx

Команда logout

Команда завершує роботу в сеансі Unix/Linux. Приклад:

logout

Команда reboot

Команда викликає перезавантаження системи. Вона є укороченою версією команди *shutdown -r*. Приклад:

reboot

Команда halt

Команда викликає останов системи (завершує роботу ОС і, якщо це можливо, вимикає живлення комп'ютера). Вона є укороченою версією команди *shutdown -h*. Приклад:

halt

Команда df

Команда інформує про наявність вільного місця на дискових носіях. При використанні з ключем -h виводить в форматі, зручному для розуміння людиною. Приклади:

df
df -h

Команда du

Команда оцінює, скільки займають місця файли/каталоги на дисках. Команда має безліч ключів. Наприклад, при використанні ключа -h команда здійснює виведення в форматі, зрозумілому людині. При використанні ключа -s команда видає тільки сумарну інформацію. При використанні ключа -c команда видає статистику про займання місця каталогами та файлами, а також підсумкове сумарне значення. Приклади:

```
du /etc
du -h /etc
du -hs /etc
du -hc /etc
du -h /etc | sort
du -hsc /usr/* | sort -r
```

Команда free

Команда інформує про стан оперативної пам'яті і використання розділу підкачки. При використанні з ключем -k виводить обсяги в КБ, з ключем -m – в МБ, з ключем -g – в ГБ. Приклади:

```
free
free -m
```

Команда last

Команда виводить на екран інформацію про дату і час реєстрації в системі користувача, показує ім'я терміналу, звідки проводилася реєстрація, ім'я хосту користувача, час і дату останньої реєстрації. Термінал з ім'ям :0 означає графічну підсистему X Window. Приклади:

```
last
last igor
last tty1
last pts/0
```

Команда find

Утиліта пошуку файлів в Unix/Linux-подібних ОС. Може здійснювати пошук в одній або декількох директоріях з використанням критеріїв, заданих користувачем. За замовчуванням повертає всі файли в поточному каталозі. Використовує велику кількість різноманітних опцій. Найбільш відомі опції:

name – пошук по імені файлу;
iname – пошук по імені файлу без урахування регістру;
type – тип шуканого: f – файл; d – каталог; l – посилання (link);
user – власник (ім'я користувача або UID);
group – власник (група користувача або GID);
perm – вказує права доступу;
size – розмір (вказується в 512-байтних блоках або байтах, символ 'c' за числом вказує на ознаку байтів);

atime – час останнього звернення до файлу;

ctime – час останньої зміни власника або прав доступу до файлу;

mtime – час останньої зміни файлу;

delete – видаляти знайдені файли;

print – показує на екрані знайдені файли;

exec command {} \; – виконує над знайденим файлом зазначену команду;

prune – використовується, коли необхідно виключити з пошуку певні каталоги.

Приклади.

1. Знайти всі файли, починаючи з поточної директорії, назва яких починається на 'my':

find . -name 'my*'

2. Знайти всі файли, починаючи з кореневої директорії, назва яких починається на 'my':

find / -name 'my*'

3. Пошук в директоріях /usr/local та /opt/local певних файлів:

find /usr/local /opt/local -name 'my*'

4. Пошук в поточному каталозі файлів 'my*' або (опція -o) файлів 'qu*' (якщо потрібно логічне І, тоді використовують опцію -a):

find . \(-name 'my*' -o -name 'qu*'\) -print

5. Вивести всі файли з поточного каталогу та нижче, змінені протягом останніх 10-ти хвилин (з розширенням '.xml'):

find . -mmin -10
find . -mmin -10 -name '*.xml'

6. Вивести всі файли з поточного каталогу та нижче, розмір яких перевищує 500 МБ:

find . -size +500M

7. Знайти файли жорсткого зв'язку з однаковим значенням inode:

find /usr -samefile /usr/bin/gcc
find /usr -inum 1196018

8. Знайти файли-архіви стиснуті gzip і вивести їх вміст в файл archives.txt, розташований в домашньому каталозі користувача:

find /usr -iname '*.tar.gz' -exec file '{}' \; -exec tar -tf '{}' >> $HOME/archives.txt \;

Команда grep

Утиліта командного рядка, що знаходить на введенні рядки, які відповідають заданому регулярному виразу. Використовується як без додаткових опцій, так і з ними. Приклади:

1. Виведення з файлу myscript всіх рядків, які містять слово 'echo':

grep 'echo' ./myscript

2. Виведення на екран рядків, які містять ім'я командного процесора Bash:

ps -ef | grep bash

3. З буфера повідомлень ядра, вивести тільки ті, які мають відношення до шини PCI з ігноруванням регістра символів:

dmesg | grep 'PCI' -i

4. Виведення на екран імен файлів з поточної директорії, які містять певне слово:

grep *-l* 'echo' *

5. Виведення на екран імен файлів з поточної директорії і нижче, які містять певне слово і не є бінарними:

grep *-r -I -l* '<HTML>' *

6. Виведення на екран тільки рядків з коментарями з заданого файлу:

grep -e ^[#] ./myscript

7. Виведення на екран всіх рядків без рядків з коментарями з заданого файлу:

grep -v -e ^[#] ./myscript

8. Пошук в поточному каталозі і нижче всіх текстових файлів, які містять підрядок 'STRING':

find . -type f -name '*.txt' -exec grep -i -H 'STRING' {} \;

Команда sed

Утиліта командного рядка, що представляє потоковий текстовий редактор, якій застосовує різні зумовлені текстові перетворення до послідовного потоку текстових даних. Приклади:

1. Виконати глобальну заміну символу ',' на '.' в певному файлі і вивести результат заміни на екран (або в новий файл):

```
sed  -e  's/,/./g'  ./geodata.dat
sed  -e  's/,/./g'  ./geodata.dat > ./geodata2.dat
```

2. Виконати глобальну заміну слова 'oldstuff' на 'newstuff' в певному файлі і вивести результат заміни в новий файл:

```
sed  -e  's/oldstuff/newstuff/g'  ./infotable.txt >  ./newinfotable.txt
```

Команда sort

Команда призначена для сортування рядків текстових файлів. Наприклад, нехай існує файл textfile.txt такого змісту:

```
1 Ivanov A.
3 Petrov V.
4 Sidorov M.
2 Ivanov K.
```

Тоді результат команди *sort ./textfile.txt* буде наступним:

```
1 Ivanov A.
2 Ivanov K.
3 Petrov V.
4 Sidorov M.
```

Використання ключа -b дозволяє ігнорувати пробіли на початку сортуємих полів або ключів. З ключем -r (reverse) сортування виконується в зворотному порядку. Використовуючи ключ -u отримують унікальне сортування (ігноруються повторювані рядки). За замовчуванням команда виводить результат сортування в потік стандартного виведення. При використанні ключа -o можна вказати файл, в який буде проведене виведення результату сортування. На сортування впливають установки локалі. Приклади:

```
sort ./textfile.txt
sort -r ./textfile.txt
sort -o ./sorttextfile.txt ./textfile.txt
du -h / etc | sort
```

Команди head і tail

Команди виводять вказану кількість рядків (за замовчуванням 10) з початку зазначеного файлу (head) або з кінця (tail). Вказати кількість виведених рядків можна за допомогою опції -n. Приклади:

```
cat /var/log/messages | tail -n 20
du -sc /usr/share/* | sort -nr | head -n 20
```

Команда tar

Архіватор. Одним з переваг формату tar при створенні архівів є те, що в архів записується інформація про структуру каталогів, про власника та групи окремих файлів. Сам tar не створює стислих архівів, а використовує для стиснення зовнішні утиліти, такі як: gzip, bzip2, lzma, lzoop, lzip, 7-zip. Приклади.

1. Архівування каталогу ./tmp в архів tmp.tar.gz і стиснення його архіватором gzip:

```
tar  -czvf  ./tmp.tar.gz  tmp
```

2. Розпакування архіву в поточний каталог або в заданий каталог:

```
tar  -xzvf  ./tmp.tar.gz
tar  -xzvf  ./tmp.tar.gz  -C /home/igor/packtmp
```

3. Виведення вмісту архіву на екран:

```
tar  -tf  ./tmp.tar.gz
```

4. Виведення вмісту архіву на екран з інформацією про права та атрибути:

```
tar  -tvf  ./tmp.tar.gz
```

5. Архівування з використанням 7-zip:

```
tar  -cvf  -  ./tmp | 7za  a  -si  ./tmp.tar.7z
```

6. Розпакування з використанням 7-zip:

```
7za  x  -so  ./tmp.tar.7z  |  tar  xf  -
```

7. Виведення вмісту tar.7z-архіву на екран:

```
7za x -so ./tmp.tar.7z | tar -tf -
```

Команди lscpu, lspci, lsusb, lsmod, lsblk, lsof

Команда lscpu показує інформацію про архітектуру процесора. lspci – показує список всіх пристроїв на шині PCI. lsusb – показує список всіх USB-пристроїв. lsmod – показує статус модулів ядра GNU/Linux. lsblk – виводить список блокових пристроїв. lsof – виводить список відкритих файлів. Приклади:

```
lscpu
lspci
lspci -tv
lsusb
lsusb -tv
lsmod
lsmod | more
lsblk
lsof
lsof | more
lsof | grep user | more
```

Команда modinfo

Команда виводить інформацію про зазначений модуль GNU/Linux-ядра. Наприклад:

```
modinfo video
```

Команда awk (іноді відсутній в дистрибутивах за замовчуванням)

Awk – утиліта, яка призначена для простих обчислювальних маніпуляцій над даними. Це повноцінна інтерпретуєма скриптова мова обробки текстової інформації з C-подібним синтаксисом. Awk має досить великі можливості. Далі будуть розглянуті тільки деякі з них.

Awk обробляє кожен рядок вхідного потоку і розбиває його на поля, які в тексті відокремлені один від одного пробільними символами. В якості роздільників можуть бути використані і інші символи. Awk аналізує та обробляє кожне поле окремо, що робить його ідеальним інструментом для роботи зі структурованими текстовими файлами, наприклад, які представляють інформацію в табличному вигляді.

Усередині сценаріїв командної оболонки, код awk екранують в одинарні лапки та фігурні дужки. Наприклад, нехай маємо файл textfile.txt з вмістом, розглянутим вище в команді *sort*. Тоді наступна команда виведе на консоль тільки прізвища людей (вміст другого поля), перерахованих в цьому файлі:

```
awk '{print $2}' ./textfile.txt
```

При формуванні виведення даних полів можна використовувати спецсимволи форматування мови С. Наприклад, використання табуляцій:

```
awk '{print $2 "\t\t" $3}' ./textfile.txt
```

Представлена команда виводить вміст другого поля, потім генерує два символи табуляції і потім виводить дані третього поля файлу textfile.txt.

Таким чином, $1 – перше поле, $2 – друге поле і т.д. Для посилання на весь рядок цілком використовується $0. Рядок може містити до 100 полів. Рядок може містити максимально до 256 символів.

Усередині awk можуть використовуватися змінні (як числові, так і рядкові). Наприклад, наступний фрагмент коду показує використання деякої змінної *i*, що є номером рядка, і виведенння спільно з даними другого і третього полів:

```
awk '{i ++; print i "\t\t" $2 "\t\t" $3}' ./textfile.txt
```

Конструкція у фігурних дужках виконується кожен раз для нового рядка вхідного потоку (або файлу), однак значення змінної *i* зберігається, і кожен раз збільшується. Початкове значення змінної дорівнює нулю.

Якщо потрібно вивести в потік виведення інформацію після обробки, то використовують так званий селектор END. У наступному прикладі значення змінної *i* буде виведено після виведення всіх даних поля 2 та 3 – значення буде відповідати кількості оброблених рядків файлу textfile.txt:

```
awk '{i ++; print $2 "\t\t" $3} END {print "COUNT=" i}' ./textfile.txt
```

Такий же результат дасть наступна команда:

```
awk '{print $2 "\t\t" $3} END {print "COUNT=" NR}' ./textfile.txt
```

NR – це передвизначена в awk змінна, яка дорівнює номеру рядка, який обробляється.

Часто зручно поєднати виведення з командою echo, щоб сформувати заголовок виведеної інформації. Наприклад:

```
echo -e "Number\t\tName"; awk '{print $1 "\t\t" $2 " " $3}' ./textfile.txt
```

Щоб виконати будь-які дії до початку аналізу рядків, використовується селектор BEGIN. Наприклад, перетворена попередня команда може бути записана наступним чином:

```
awk 'BEGIN {print "Number\t\tName"} {print $1 "\t\t" $2 " " $3}' ./textfile.txt
```

Наступна команда встановлює в якості роздільника полів символ ':'.

```
ps -e | awk '{print $3}' | awk 'BEGIN {FS = ":"} {print $3}'
```

Спочатку здійснюється виведення команди ps -e. Його результат (значення поля TIME – третє поле) подається на вхід утиліти awk. Результат обробки – тільки значення цього поля, яке подається для обробки знову на вхід awk і тепер як роздільник використовується символ ':'. Підсумком виконання буде витяг поля секунд (теж третє поле) з поля TIME команди ps -e.

Відзначимо, що для установки символу-роздільника полів можна використовувати ключ -F утиліти awk. Таким чином, попередня команда еквівалентна наступній:

```
ps -e | awk '{print $3}' | awk -F: '{print $3}'
```

При використанні утиліти можна застосовувати оператори керування, такі як: if, while, for та інші. Наприклад, необхідно вивести на консоль інформацію про перші 20-ь процесів, які споживають найбільшу кількість процесорного часу і працюють більше 3-х годин в системі:

```
ps -eo "%U:%p:%C:%t:%c" --sort=-%cpu | \
head -n 20 | \
awk -F: '{if(NR==1) print $0; if(NF==7 && $4>3) print $0}'
```

де внутрішня змінна NF показує кількість полів у рядку.

Приклад використання в awk for для форматованого виведення:

```
ps -eo user,pid,time,stime,etime,comm | \
head | \
awk '{ for(i=1;i<NF;i++) { s=s$i"|\t" } { s=s$NF } { print s; s="" } }'
```

Часто awk зручно використовувати спільно з командою *sort*. Наприклад:

```
du -sc /usr/share/* | sort -nr | head | awk '{ print $1/1024 " MB\t" $2 }'
```

Наведена команда дозволяє спочатку визначити обсяг дискового простору, займаного каталогами, які розташовані в каталозі /usr/share (обсяг виводиться в

кілобайтах). Потім список сортується в порядку спадання підрахованих значень, а потім виводяться тільки перші дев'ять найбільш великих каталогів і для першого рядку підсумкове значення. Ця інформація подається на вхід утиліти awk, яка перетворює значення в зручні для розуміння людиною (перераховує в мегабайти) з висновком абревіатури.

Більш детально про використання awk дивиться за наступними посиланнями:

https://uk.wikipedia.org/wiki/AWK

https://www.ibm.com/developerworks/ru/library/l-awk1/index.html

Навчальне видання

Гаркуша Ігор Миколайович

Конспект лекцій
з дисципліни “Операційні системи”
для студентів галузі знань 12 “Інформаційні технології”
спеціальності 126 “Інформаційні системи та технології”

Електронний ресурс

Видано
у Національному технічному університеті
«Дніпровська політехніка».
Свідоцтво про внесення до Державного реєстру ДК № 1842 від 11.06.2004.
49005, м. Дніпро, просп. Дмитра Яворницького, 19.