МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ ІМ. ІВАНА ПУЛЮЯ

КАФЕДРА КОМП’ЮТЕРНО-ІНТЕГРОВАНИХ ТЕХНОЛОГІЙ

**ОПЕРАЦІЙНІ СИСТЕМИ**

**КОНСПЕКТ ЛЕКЦІЙ**

для студентів напряму підготовки 6.050202 «Автоматизація та комп’ютерно-інтегровані технології» з орієнтацією на спеціальність 151 «Автоматизація та комп’ютерно-інтегровані технології»

Тернопіль

2016

УДК 681.3

ББК 32.973-01

Укладачі:

*Микитишин А.Г.*, канд. техн. наук, доцент,

*Чихіра І.В.*, канд. техн. наук, доцент.

Рецензент

*Коноваленко І.В.,* канд. техн. наук, доцент.

Схвалено та рекомендовано до друку на засіданні кафедри №1 від 29.08.2016р.

Засідання факультету №1 від 29.08.2016р.

Відповідальний за випуск *Микитишин А.Г.*, канд. техн. наук, доцент

Лекція 1

Історія розвитку мережевих операційних систем

**1. Поява перших операційних систем**

Цифрові обчислювальні машини з’явились після закінчення Другої світової війни. В середині 40-х були створені перші **лампові** обчислювальні пристрої. У той час одна і та ж група людей брала участь і в проектуванні, і в експлуатації, і в програмуванні обчислювальної машини. Це була швидше науково-дослідна робота в галузі обчислювальної техніки, а не використання комп’ютерів як інструменту вирішення будь-яких практичних завдань. Програмування здійснювалося виключно на машинній мові. Не було ніякого системного програмного забезпечення, окрім бібліотек математичних і службових підпрограм, які програміст міг використовувати для того, щоб не писати кожного разу коди, що обчислюють значення певної математичної функції або керують стандартним пристроєм вводу-виводу. Операційні системи ще не з’явилися, усі завдання організації обчислювального процесу вирішувалися вручну програмістом з пульта управління, який був примітивним пристроєм вводу-виводу, що складався з кнопок, перемикачів і індикаторів.

З середини 50-х років почався новий період в розвитку обчислювальної техніки, пов’язаний з появою нової технічної бази – **напівпровідникових елементів**. Виросла швидкодія процесорів, збільшилися об’єми оперативної і зовнішньої пам’яті. Помітний прогрес спостерігався також в області автоматизації програмування і організації обчислювальних робіт. У ці роки з’явилися перші алгоритмічні мови, і таким чином до бібліотек математичних і службових підпрограм додався новий тип системного програмного забезпечення – **транслятори**.

Виконання кожної програми стало включати велику кількість допоміжних робіт: завантаження потрібного транслятора (АЛГОЛ, ФОРТРАН, КОБОЛ і інші), запуск транслятора і отримання результуючої програми в машинних кодах, зв’язування програми з бібліотечними підпрограмами, завантаження програми в оперативну пам’ять, запуск програми, виведення результатів на периферійний пристрій. Для організації ефективного спільного використання трансляторів, бібліотечних програм і завантажувачів в штат багатьох обчислювальних центрів були введені посади операторів, що професійно виконували роботу по організації обчислювального процесу для усіх користувачів цього центру.

Але як би швидко і надійно не працювали оператори, велику частину часу процесор простоював в очікуванні, поки оператор запустить чергове завдання. Для вирішення цієї проблеми були розроблені перші **системи пакетної обробки**, які автоматизували усю послідовність дій оператора по організації обчислювального процесу. Початкові системи пакетної обробки стали прообразом сучасних операційних систем, вони стали першими системними програмами, призначеними не для обробки даних, а для управління обчислювальним процесом.

Оператор складав пакет завдань, які надалі без його участі послідовно запускалися на виконання керуючою програмою – **монітором**. Пакет зазвичай був набором перфокарт (з часом почали переносити на магнітну стрічку або магнітний диск). Сама програма-монітор в перших реалізаціях також зберігалася на перфокартах або перфострічці, а в пізніших - на магнітній стрічці і магнітних дисках.

Ранні системи пакетної обробки значно скоротили витрати часу на допоміжні дії з організації обчислювального процесу, що підвищило ефективність використання комп’ютерів. Проте, при цьому програмісти-користувачі втратили безпосередній доступ до комп’ютера, що знизило ефективність їх роботи - внесення будь-якого виправлення вимагало значно більше часу, ніж при інтерактивній роботі за пультом машини.

**2. Поява мультипрограмних операційних систем для мейнфреймів**

Наступний важливий період розвитку операційних систем відноситься до 1965-1975 років. В цей час в технічній базі обчислювальних машин стався перехід від окремих напівпровідникових елементів типу транзисторів до **інтегральних мікросхем**, що відкрило шлях до появи наступного покоління комп’ютерів.

У цей період були реалізовані практично усі основні механізми, властиві сучасним ОС: мультипрограмування, мультипроцесування, підтримка багатотермінального багатокористувацького режиму, віртуальна пам’ять, файлові системи, розмежування доступу і мережева робота. У ці роки починається розквіт системного програмування.

Революційною подією цього етапу стала промислова реалізація **мультипрограмування**. В умовах різко збільшених можливостей комп’ютера по обробці і зберіганню даних виконання тільки однієї програми в кожен момент часу виявилося вкрай неефективним. Рішенням стало мультипрограмування – спосіб організації обчислювального процесу, при якому в пам’яті комп’ютера знаходилося одночасно декілька програм, що поперемінно виконувались на одному процесорі.

На початку розвитку, мультипрограмування було реалізоване в двох варіантах – в системах пакетної обробки і розподілу часу.

В мультипрограмних **системах пакетної обробки** для одночасного виконання вибиралися програми, що пред’являли різні вимоги до ресурсів, так, щоб забезпечувалося збалансоване завантаження всіх пристроїв комп’ютера. Перемикання процесора між програмами відбувалося тільки у випадку, якщо активна програма сама відмовлялася від процесора, наприклад, через необхідність виконати операцію вводу-виводу. Оскільки, одна програма могла надовго зайняти процесор, це робило неможливим виконання інтерактивних програм. Очевидно, що у мультипрограмних системах пакетної обробки користувач як і раніше був позбавлений можливості інтерактивно взаємодіяти зі своїми програмами.

Мультипрограмні **системи** **розподілу часу** були розробленні для усунення основного недоліку систем пакетної обробки – ізоляцію користувача від процесу виконання його задач. Варіант мультипрограмування, який використовувався в системах розподілу часу, був направлений на створення для кожного окремого користувача ілюзії одноосібного володіння обчислювальною машиною за рахунок періодичного виділення кожній програмі своєї долі процесорного часу. Зрозуміло, що системи розподілу часу володіли меншою пропускною спроможністю, ніж системи пакетної обробки, оскільки на виконання приймалася кожна запущена користувачем задача, а не та, яка "вигідна" системі, і, крім того, існували втрати обчислювальної потужності на більш часте перемикання процесора із задачі на задачу. Критерієм ефективності систем розподілу часу є не максимальна пропускна спроможність, а зручність і ефективність роботи користувача.

На сьогодні, існує і третій варіант мультипрограмної системи - **системи** **реального часу.** Дані системи використовуються для управління різними технічними об’єктами (наприклад, верстатом, супутником, науковою експериментальною установкою і т. д.) або технологічними процесами (наприклад, гальванічною лінією, доменним процесом і тому подібне). У всіх цих випадках існує гранично допустимий час, протягом якого повинна бути виконана певна програма, що управляє об’єктом, інакше може відбутися аварія. Таким чином, критерієм ефективності для систем реального часу є їх здатність витримувати наперед задані інтервали часу між запуском програми і отриманням результату. Цей час називається **часом реакції системи**, а відповідна властивість системи - **реактивністю**. Для цих систем мультипрограмна суміш є фіксованим набором наперед розроблених програм, а вибір програми на виконання здійснюється виходячи з поточного стану об’єкту або відповідно до розкладу планових робіт.

**Багатотермінальний режим** використовувався в системах пакетної обробки та системах розподілу часу. При цьому не лише оператор, але і усі користувачі діставали можливість формувати свої завдання і керувати їх виконанням зі свого терміналу. Такі операційні системи дістали назву **систем віддаленого вводу завдань**. Термінальні комплекси могли розташовуватися на великій відстані від процесорних стійок, з’єднуючись з ними за допомогою різних глобальних зв’язків. Для підтримки віддаленої роботи терміналів в операційних системах з’явилися спеціальні програмні модулі, що реалізовували різні (в той час, як правило, нестандартні) протоколи зв’язку. Такі обчислювальні системи з віддаленими терміналами, зберігаючи централізований характер обробки даних, були прообразом сучасних мереж, а відповідне системне програмне забезпечення - прообразом мережевих операційних систем.

Реалізація мультипрограмування вимагала внесення важливих змін до апаратури комп’ютера. При розподілі ресурсів комп’ютера між програмами необхідно було забезпечити швидке перемикання процесора з однієї програми на іншу, а також надійно захистити коди і дані однієї програми від ненавмисного або навмисного пошкодження іншою програмою. У процесорах з’явився привілейований і користувацький режими роботи, спеціальні регістри для швидкого перемикання з однієї програми на іншу, засоби захисту областей пам’яті, а також розвинена система переривань.

У привілейованому режимі, призначеному для роботи програмних модулів операційної системи, процесор міг виконувати усі команди, у тому числі і ті з них, які дозволяли здійснювати розподіл і захист ресурсів комп’ютера. Програмам, що працювали в користувацькому режимі, деякі команди процесора були недоступні. Таким чином, тільки ОС могла керувати апаратними засобами і виконувати роль монітора і арбітра для програм користувача, які виконувалися в користувацькому режимі.

Система переривань дозволяла синхронізувати роботу різних пристроїв комп’ютера, що працюють паралельно і асинхронно, таких як канали вводу-виводу, диски, принтери і т. д.

Ще однією важливою тенденцією цього періоду було створення сімейств **програмно-сумісних машин** і операційних систем для них. Операційні системи, побудовані з можливістю роботи на великих і малих обчислювальних системах, в комерційній області і в області наукових досліджень, виявилися надзвичайно складними. Операційні системи цього покоління були дуже дорогими, вони складалися з багатьох мільйонів асемблерних рядків, написаних тисячами програмістів, і містили тисячі помилок, що викликали нескінченний потік виправлень. Так, розробка OS/360, об’єм коду для якої склав 8 Мбайт, коштувала компанії IBM 80 мільйонів доларів. Проте, незважаючи на величезні розміри і безліч проблем, OS/360 і інші їй подібні операційні системи цього покоління дійсно задовольняли більшості вимог споживачів.

**3. Поява мережевих операційних систем**

На початку 70-х років з’явилися перші **мережеві операційні системи**, які на відміну від багатотермінальних ОС дозволяли не лише розосередити користувачів, але і організувати розподілене зберігання і обробку даних між декількома комп’ютерами, з’єднаними електричними зв’язками. Будь-яка мережева операційна система, з одного боку, виконує усі функції локальної операційної системи, а з іншого боку, має деякі додаткові засоби, що дозволяють їй взаємодіяти по мережі з операційними системами інших комп’ютерів. Програмні модулі, що реалізовують мережеві функції, з’являлися в операційних системах поступово, у міру розвитку мережевих технологій, апаратної бази комп’ютерів і виникнення нових завдань, що вимагають мережевої обробки.

У 1969 році Міністерство оборони США ініціювало роботи по об’єднанню суперкомп’ютерів оборонних і науково-дослідних центрів в єдину мережу. Ця мережа дістала назву ARPANET і стала відправною точкою для створення найвідомішої нині глобальної мережі – Інтернету. Мережа ARPANET об’єднувала комп’ютери різних типів, що працювали під управлінням різних ОС з доданими модулями, що реалізовували комунікаційні протоколи, загальні для усіх комп’ютерів мережі.

У 1974 році компанія IBM оголосила про створення власної мережевої архітектури для своїх мейнфреймів, що дістала назву SNA (System Network Architecture). Ця багаторівнева архітектура, багато в чому подібна до стандартної моделі OSI, що з’явилася дещо пізніше, забезпечувала взаємодію типу "термінал-термінал", "термінал-комп’ютер" і "комп’ютер-комп’ютер" по глобальних зв’язках.

В цей же час в Європі велися активні роботи із створення і стандартизації мереж Х. 25. Ці мережі з комутацією пакетів не були прив’язані до якої-небудь конкретної операційної системи. Після отримання статусу міжнародного стандарту в 1974 році протоколи Х. 25 стали підтримуватися багатьма операційними системами. З 1980 року компанія IBM включила підтримку протоколів Х. 25 в архітектуру SNA і у свої операційні системи.

До середини 70-х років широкого поширення набули міні-комп’ютери, які першими використали переваги великих інтегральних схем, що дозволили реалізувати досить потужні функції при порівняно невисокій вартості комп’ютера.

Доступність міні-комп’ютерів і внаслідок цього їх поширеність на підприємствах послужили потужним стимулом для створення локальних мереж. Підприємство могло собі дозволити мати декілька міні-комп’ютерів, що знаходяться в одній будівлі або навіть в одній кімнаті. Природно, виникала потреба в обміні інформацією між ними і в спільному використанні дорогого периферійного обладнання.

Перші локальні мережі будувалися за допомогою нестандартного комунікаційного обладнання, в простому випадку – шляхом прямого з’єднання послідовних портів комп’ютерів. Програмне забезпечення також було нестандартним і реалізовувалося у вигляді користувацьких додатків.

Важливою віхою в історії мережевих операційних систем стало створення ОС UNIX. Спочатку ця ОС призначалася для підтримки режиму розподілу часу в міні-комп’ютері PDP-7. З середини 70-х років почалося масове використання ОС UNIX. До цього часу програмний код для UNIX був на 90% написаний на мові високого рівня С. Широке поширення ефективних С-компіляторів зробило UNIX унікальною для того часу ОС, з можливістю порівняно легкого перенесення на різні типи комп’ютерів. Оскільки ця ОС поставлялася разом з початковими кодами, то вона стала першою відкритою ОС, яку могли удосконалювати прості користувачі-програмісти.

Перший мережевий додаток для ОС UNIX – програма UUCP (UNIX-to-UNIX Copy program) з’явилася в 1976 році і почала поширюватися з версією 7 AT&T UNIX з 1978 року. Ця програма дозволяла копіювати файли з одного комп’ютера на іншій в межах локальної мережі через апаратні інтерфейси - RS-232, а також могла працювати через глобальні зв’язки, наприклад модемні.

Хоча UNIX була спочатку розроблена для міні-комп’ютерів, гнучкість, потужні функціональні можливості і відкритість дозволили їй зайняти міцні позиції в усіх класах комп’ютерів: суперкомп’ютерах, мейнфреймах, міні-комп’ютерах, серверах і робочих станціях на базі RISC-процесорів, персональних комп’ютерах.

**4. Поява операційних систем для персональних комп’ютерів**

До найбільш важливих подій цього десятиліття можна віднести розробку стеку TCP/IP, становлення Інтернету, стандартизацію технологій локальних мереж, появу персональних комп’ютерів і операційних систем для них.

Робочий варіант стеку протоколів TCP/IP був створений у кінці 70-х років. Цей стек був набором загальних протоколів для різнорідного обчислювального середовища і призначався для зв’язку експериментальної мережі ARPANET з іншими мережами "сателітів". У 1983 році стек протоколів TCP/IP був прийнятий Міністерством оборони США як військовий стандарт. Перехід комп’ютерів мережі ARPANET на стек TCP/IP прискорила його реалізація для операційної системи BSD UNIX. З того часу почалося спільне існування UNIX і протоколів TCP/IP.

У 1983 році мережа ARPANET була розділена на дві частини: MILNET, яку підтримували військові відомства США, і нову ARPANET. Для позначення складової мережі ARPANET почала використовуватися назва Internet. Інтернет став відмінним полігоном для випробувань багатьох мережевих операційних систем, що дозволило перевіряти в реальних умовах можливості їх взаємодії, міру масштабованості, здатність роботи при екстремальному навантаженні, що створюється сотнями і тисячами користувачів. Незалежність від виробників, гнучкість і ефективність, доведені успішною роботою в Інтернеті, а також відкритість і доступність стандартів зробили стек протоколів TCP/IP не лише головним транспортним механізмом Інтернету, але і основним стеком більшості мережевих операційних систем.

Усе десятиліття було відмічене постійною появою нових, усе більш досконалих версій ОС UNIX. Серед них були і фірмові версії UNIX: SunOS, HP-UX, Irix, AIX і багато інших, в яких виробники комп’ютерів адаптували код ядра і системних утиліт для своєї апаратури. Різноманітність версій породила проблему їх сумісності, яку періодично намагалися вирішити різні організації. В результаті були прийняті стандарти POSIX, які визначальні інтерфейси ОС для додатків, а спеціальний підрозділ компанії AT&T випустив декілька версій UNIX System III і UNIX System V, які були покликані консолідувати розробників на рівні коду ядра.

На початку 80-х років появились персональні комп’ютери, вартість яких була істотно нижча від вартості міні-комп’ютерів. Комп’ютери стали широко використовуватися неспеціалістами, що вимагало розробки "дружнього" програмного забезпечення, і надання цих "дружніх" функцій стало прямим обов’язком операційних систем. Персональні комп’ютери послужили також потужним каталізатором для бурхливого зростання локальних мереж, створивши для цього відмінну матеріальну основу у вигляді десятків і сотень комп’ютерів, що належали одному підприємству і розташованих в межах однієї будівлі. В результаті підтримка мережевих функцій стала для ОС персональних комп’ютерів необхідною умовою.

Перша версія найбільш популярної операційної системи раннього етапу розвитку персональних комп’ютерів – MS-DOS компанії Microsoft – була однопрограмною однокористувацькою ОС з інтерфейсом командного рядка, здатна стартувати з дискети. Основними завданнями для неї були управління файлами, розташованими на гнучких і жорстких дисках, а також почерговий запуск програм. Розробники перших персональних комп’ютерів вважали, що при індивідуальному використанні комп’ютера і обмежених можливостях апаратури немає сенсу в підтримці мультипрограмування, тому в процесорі не були передбачені привілейований режим і інші механізми підтримки мультипрограмних систем.

Відсутні функції для MS-DOS компенсувалися зовнішніми програмами, що надавали користувачеві зручний графічний інтерфейс (Norton Commander) або засоби тонкого управління дисками (PC Tools). Мережеві функції також реалізовувалися в основному мережевими оболонками, що працювали поверх ОС. Найбільший вплив на розвиток програмного забезпечення для персональних комп’ютерів зробило операційне середовище Windows, що було надбудовою над MS-DOS.

Історія мережевих засобів MS-DOS почалася з версії 3.1. Ця версія MS-DOS додала до файлової системи необхідні засоби блокування файлів і записів, які дозволили більш ніж одному користувачеві мати доступ до файлу. Користуючись цими функціями, мережеві оболонки могли забезпечити розподіл файлів між мережевими користувачами. Разом з випуском версії MS-DOS 3.1 в 1984 році компанія Microsoft також випустила продукт Microsoft Networks (MS-NET). Деякі концепції, закладені в MS-NET, такі як введення в структуру базових мережевих компонентів редиректора і мережевого сервера, успішно перейшли в пізніші мережеві продукти Microsoft.

Інший шлях вибрала компанія Novell. Вона з самого початку зробила ставку на розробку операційної системи із вбудованими мережевими функціями. Її мережеві операційні системи NetWare на довгий час стали еталоном продуктивності, надійності і захищеності для локальних мереж.

З найпершої версії (з 1983 року) ОС NetWare поширювалася як операційна система для центрального сервера локальної мережі, яка за рахунок спеціалізації на виконанні функцій файл-сервера забезпечує максимально можливу для цього класу комп’ютерів швидкість віддаленого доступу до файлів і підвищену безпеку даних. За високу продуктивність користувачі мереж Novell NetWare розплачувалися вартістю – виділений файл-сервер не міг використовуватися як робоча станція, а його спеціалізована ОС мала дуже специфічний прикладний програмний інтерфейс (API), що вимагав від розробників додатків особливих знань, спеціального досвіду і значних зусиль.

На відміну від Novell більшість інших компаній розвивали мережеві засоби для персональних комп’ютерів у рамках операційних систем з універсальним інтерфейсом API, тобто операційних систем загального призначення.

У 1987 році в результаті спільних зусиль Microsoft і IBM з’явилася перша багатозадачна операційна система для персональних комп’ютерів з процесором Intel 80286, що повною мірою використовувала можливості захищеного режиму – OS/2. Ця система підтримувала витісняючу багатозадачність, віртуальну пам’ять, графічний інтерфейс (не з першої версії) і віртуальну машину для виконання DOS-додатків. Фактично вона виходила за межі простої багатозадачності з її концепцією розпаралелювання окремих процесів, що дістала назву **багатопоточності**.

OS/2 з її розвиненими функціями багатозадачності і вбудованими засобами багато- користувацького захисту виявилася хорошою платформою для побудови локальних мереж персональних комп’ютерів. Найбільшого поширення набули мережеві оболонки LAN Manager компанії Microsoft і LAN Server компанії IBM, розроблені на основі одного базового коду. Ці оболонки поступалися по продуктивності файловому серверу NetWare і споживали більше апаратних ресурсів, але мали важливі переваги – вони дозволяли, по-перше, виконувати на сервері будь-які програми, розроблені для OS/2, MS-DOS і Windows, а по-друге, використовувати комп’ютер, на якому вони працювали, як робочу станцію.

У 80-і роки були прийняті основні стандарти на комунікаційні технології для локальних мереж: в 1980 році - Ethernet, в 1985 - Token Ring, у кінці 80-х - FDDI. Це дозволило забезпечити сумісність мережевих операційних систем на нижніх рівнях, а також стандартизувати інтерфейс ОС з драйверами мережевих адаптерів.

Для персональних комп’ютерів застосовувалися не лише спеціально розроблені для них операційні системи, подібні MS-DOS, NetWare і OS/2, але і адаптувалися вже існуючі ОС. Поява процесорів Intel 80286 і особливо 80386 з підтримкою мультипрограмування дозволило перенести на платформу персональних комп’ютерів ОС UNIX. Найбільш відомою системою цього типу була версія UNIX компанії Santa Cruz Operation (SCO UNIX).

**5. Особливості сучасного етапу розвитку мережевих ОС**

Починаючи з 90-их років, практично усі операційні системи, що займали помітне місце на ринку, стали мережевими. Мережеві функції сьогодні вбудовуються в ядро ОС. Операційні системи отримали засоби для роботи з усіма основними технологіями локальних (Ethernet, Token Ring, FDDI, ATM) і глобальних (Х. 25, frame relay, ISDN, ATM) мереж. У операційних системах використовуються засоби мультиплексування декількох стеків протоколів, за рахунок якого комп’ютери можуть підтримувати одночасну мережеву роботу з різнорідними клієнтами і серверами. З’явилися спеціалізовані ОС, які призначені виключно для виконання комунікаційних завдань (міжмережева ОС IOS компанії Cisco Systems).

У другій половині 90-х років усі виробники операційних систем різко посилили підтримку засобів роботи з Інтернетом (окрім виробників UNIX-систем, в яких ця підтримка завжди була суттєвою). Окрім самого стеку TCP/IP в комплект постачання почали включати утиліти, що реалізовують такі популярні сервіси Інтернету, як telnet, ftp, DNS і Web. Вплив Інтернету проявився і в тому, що комп’ютер перетворився з чисто обчислювального пристрою в засіб комунікацій з розвиненими обчислювальними можливостями.

Особлива увага протягом усього останнього десятиліття приділяється **корпоративним мережевим операційним системам**. Корпоративним мережам властива висока міра гетерогенності програмних і апаратних засобів, тому корпоративна ОС повинна безпроблемно взаємодіяти з ОС різних типів і працювати на різних апаратних платформах. Для корпоративної ОС дуже важлива наявність засобів централізованого адміністрування і управління, що дозволяють в єдиній базі даних зберігати облікові записи про десятки тисяч користувачів, комп’ютерів, комунікаційних пристроїв і модулів програмного забезпечення, наявних в корпоративній мережі.

У сучасних операційних системах засоби централізованого адміністрування зазвичай базуються на єдиній довідковій службі (службі каталогів). Першою успішною реалізацією довідкової служби каталогів корпоративного масштабу була система StreetTalk компанії Banyan. До тепер, найбільше визнання отримала довідкова служба NDS компанії Novell, випущена в 1993 році для першої корпоративної версії NetWare 4.0. Перша корпоративна ОС компанії Microsoft (Windows 2000) базувалась на службі каталогів Active Directory.

На сучасному етапі розвитку ОС на передній план вийшли засоби забезпечення безпеки. Це пов’язано із зростанням цінності інформації, що обробляється комп’ютерами, а також з підвищеним рівнем загроз, що існують при передачі даних по мережах. Багато ОС мають сьогодні розвинені засоби захисту інформації, засновані на шифруванні даних, автентифікації і авторизації.

Сучасним ОС властива багатоплатформеність, тобто здатність працювати на абсолютно різних типах комп’ютерів. Багато операційних систем мають спеціальні версії для підтримки кластерної архітектури, що забезпечує високу продуктивність і відмовостійкість.

Постійно підвищується зручність інтерактивної роботи з комп’ютером шляхом включення в ОС розвинених графічних інтерфейсів.

Рівень зручностей у використанні ресурсів, які сьогодні надають користувачам, адміністраторам і розробникам додатків операційні системи ізольованих комп’ютерів, для мережевих операційних систем є тільки привабливою перспективою. Операційні системи майбутнього повинні забезпечити високий рівень прозорості мережевих ресурсів, взявши на себе завдання організації розподілених обчислень, перетворивши мережу на віртуальний комп’ютер. Саме цей сенс вкладають в лаконічне гасло "Мережа - це комп’ютер" фахівці компанії Sun.

**6. Вимоги до сучасних мережевих ОС**

До сучасних мережевих ОС пред’являються наступні експлуатаційні вимоги:

**Розширюваність**. Якщо код ОС написаний таким чином, що доповнення і зміни можуть вноситися без порушення цілісності системи, то таку ОС називають розширюваною. Розширюваність досягається за рахунок модульної структури ОС, при якій програми створюються з набору окремих модулів, що взаємодіють тільки через функціональний інтерфейс.

**Переносимість**. У ідеалі код ОС повинен легко переноситися з процесора одного типу на процесор іншого типу і з апаратної платформи (які розрізняються не лише типом процесора, але і способом організації усієї апаратури комп’ютера) одного типу на апаратну платформу іншого типу. Переносимі ОС мають декілька варіантів реалізації для різних платформ, таку властивість ОС називають також багатоплатформенністю.

**Сумісність**. Якщо ОС має засоби для виконання прикладних програм, написаних для інших операційних систем, то про неї говорять, що вона має сумісність з цими ОС. Слід розрізняти сумісність на рівні двійкових кодів і сумісність на рівні початкових текстів. Поняття сумісності включає також підтримку інтерфейсів користувача інших ОС.

**Надійність і відмовостійкість**. Система має бути захищена як від внутрішніх, так і від зовнішніх помилок, збоїв і відмов. Її дії мають бути завжди передбачуваними, а додатки не повинні мати можливості завдавати шкоди ОС. Надійність і відмовостійкість ОС передусім визначаються архітектурними рішеннями, покладеними в її основу, а також якістю її реалізації (відлагодженою коду).

**Безпека**. Сучасна ОС повинна захищати дані і інші ресурси обчислювальної системи від несанкціонованого доступу. Щоб ОС мала властивість безпеки, вона повинна як мінімум мати у своєму складі засоби автентифікації - визначення легальності користувачів, авторизації - надання легальним користувачам диференційованих прав доступу до ресурсів, аудиту - фіксації усіх "підозрілих" для безпеки системи подій. У мережевих ОС до завдання контролю доступу додається завдання захисту даних, що передаються по мережі.

**Продуктивність**. Операційна система повинна мати настільки хорошу швидкодію і час реакції, наскільки це дозволяє апаратна платформа. На продуктивність ОС впливає багато чинників, серед яких основними є архітектура ОС, різноманіття функцій, якість програмування коду, можливість виконання ОС на високопродуктивній платформі.

Лекція 2

Функціональні компоненти операційної системи

1. Функціональні компоненти операційної системи автономного комп’ютера

**Управління процесами**

Найважливішою частиною операційної системи, що безпосередньо впливає на функціонування обчислювальної машини, є підсистема управління процесами.

Для кожного новостворюваного процесу ОС генерує системні інформаційні структури, які містять дані про потреби процесу в ресурсах обчислювальної системи, а також про фактично виділені йому ресурси. Таким чином, процес можна також визначити як деяку заявку на споживання системних ресурсів.

Щоб процес міг бути виконаний, операційна система повинна призначити йому область оперативної пам’яті, в якій будуть розміщені коди і дані процесу, а також надати йому необхідну кількість процесорного часу. Крім того, процесу може знадобитися доступ до таких ресурсів, як файли і пристрої вводу-виводу.

У мультипрограмній операційній системі одночасно може існувати декілька процесів. Частина процесів породжується за ініціативою користувачів і їх додатків, такі процеси називають **користувацькими**. Інші процеси, звані **системними**, ініціалізуються самою операційною системою для виконання своїх функцій.

Оскільки процеси часто одночасно претендують на одні і ті ж ресурси, то в обов’язки ОС входить підтримка черг заявок процесів на ресурси, наприклад черги до процесора, до принтера, до послідовного порту.

Важливим завданням операційної системи є захист ресурсів, виділених цьому процесу, від інших процесів. Одним з ресурсів процесу, що найретельніше захищається, є області оперативної пам’яті, в якій зберігаються коди і дані процесу. Сукупність усіх областей оперативної пам’яті, виділених операційною системою процесу, називається його **адресним простором**. Захищаються і інші типи ресурсів, такі як файли, зовнішні пристрої і т. д. Операційна система може не лише захищати ресурси, виділені одному процесу, але і організовувати їх спільне використання, наприклад дозволяти доступ до деякої області пам’яті декільком процесам.

Упродовж періоду існування процесу його виконання може бути багаторазово перерване і продовжене. Для того, щоб відновити виконання процесу, необхідно відновити стан його операційного середовища. Стан операційного середовища ідентифікується станом регістрів і програмного лічильника, режимом роботи процесора, покажчиками на відкриті файли, інформацією про незавершені операції вводу-виводу, кодами помилок виконуваних цим процесом системних викликів і т. д. Ця інформація називається **контекстом прогресу**. Говорять, що при зміні процесу відбувається перемикання контекстів.

Операційна система бере на себе також функції синхронізації процесів, що дозволяють процесу припиняти своє виконання до настання певної події в системі, наприклад завершення операції вводу-виводу.

У операційній системі немає однозначної відповідності між процесами і програмами. Один і той же програмний файл може породити декілька паралельно виконуваних процесів, а процес може в ході свого виконання змінити програмний файл і почати виконувати іншу програму.

Для реалізації складних програмних комплексів буває необхідно організувати їх роботу у вигляді декількох паралельних процесів, які періодично взаємодіють один з одним і обмінюються деякими даними. Оскільки операційна система захищає ресурси процесів і не дозволяє одному процесу писати або читати з пам’яті іншого процесу, то для оперативної взаємодії процесів ОС повинна надавати особливі засоби, які називають засобами міжпроцесної взаємодії.

Таким чином, підсистема управління процесами планує виконання процесів, тобто розподіляє процесорний час між декількома одночасно існуючими в системі процесами, займається створенням і знищенням процесів, забезпечує процеси необхідними системними ресурсами, підтримує синхронізацію процесів, а також забезпечує взаємодію між процесами.

**Управління пам’яттю**

Пам’ять для процесу є таким же важливим ресурсом, як і процесор, оскільки процес може виконуватися процесором тільки у тому випадку, якщо його коди і дані (не обов’язково усе) знаходяться в оперативній пам’яті.

Управління пам’яттю включає розподіл наявної фізичної пам’яті між усіма існуючими в системі в даний момент процесами, завантаження кодів і цих процесів у відведені їм області пам’яті, налаштування адресно-залежних частин кодів процесу на фізичні адреси виділеної області, а також захист областей пам’яті кожного процесу.

Існує велика різноманітність алгоритмів розподілу пам’яті. Вони можуть відрізнятися, наприклад, кількістю виділених процесу областей пам’яті (у одних випадках пам’ять виділяється процесу у вигляді однієї безперервної області, а в інших - у вигляді декількох несуміжних областей), ступенем свободи межі областей (вона може бути жорстко зафіксована на увесь час існування процесу або ж динамічно переміщатися при виділенні процесу додаткових об’ємів пам’яті). У деяких системах розподіл пам’яті виконується сторінками фіксованого розміру, а в інших – сегментами змінної довжини.

Одним з найбільш популярних способів управління пам’яттю в сучасних операційних системах є так звана віртуальна пам’ять. Наявність в ОС механізму віртуальної пам’яті дозволяє програмістові писати програму так, як ніби в його розпорядженні є однорідна оперативна пам’ять великого об’єму, що часто істотно перевищує об’єм наявної фізичної пам’яті. Насправді усі дані, що використовуються програмою, зберігаються на диску і при необхідності частинами (сегментами або сторінками) відображуються у фізичну пам’ять. При переміщенні кодів і даних між оперативною пам’яттю і диском підсистема віртуальної пам’яті виконує трансляцію віртуальних адрес, отриманих в результаті компіляції і компонування програми, у фізичні адреси елементів оперативної пам’яті. Дуже важливо, що усі операції по переміщенню кодів і даних між оперативною пам’яттю і дисками, а також трансляція адрес виконуються ОС прозоро для програміста.

Захист пам’яті - це здатність застерігати виконуваний процес від запису або читання пам’яті, призначеної іншому процесу. Засоби захисту пам’яті, що реалізовані в операційній системі, повинні присікати несанкціонований доступ процесів до чужих областей пам’яті.

Таким чином, функціями ОС по управлінню пам’яттю є відстежування вільної і зайнятої пам’яті; виділення пам’яті процесам і звільнення пам’яті при завершенні процесів; захист пам’яті; витіснення процесів з оперативної пам’яті на диск, коли розміри основної пам’яті недостатні для розміщення в ній усіх процесів, і повернення їх в оперативну пам’ять, коли в ній звільняється місце, а також налаштування адрес програми на конкретну область фізичної пам’яті.

**Управління файлами і зовнішніми пристроями**

Здатність ОС до "екранування" складнощів реальної апаратури дуже яскраво проявляється в одній з основних підсистем ОС – **файловій системі**. Операційна система віртуалізує окремий набір даних, що зберігаються на зовнішньому накопичувачі, у вигляді файлу – неструктурованої послідовності байтів, що має символьне ім’я. Для зручності роботи з даними файли групуються в каталоги, які, у свою чергу, утворюють групи – каталоги більш високого рівня. Користувач може за допомогою ОС виконувати над файлами і каталогами певні дії (пошук по імені, видалення, копіювання, зміну і збереження вмісту і ін.).

Файлова система ОС виконує перетворення символьних імен файлів, з якими працює користувач або прикладний програміст, у фізичні адреси даних на диску, організовує спільний доступ до файлів, захищає їх від несанкціонованого доступу.

При виконанні своїх функцій файлова система тісно взаємодіє з підсистемою управління зовнішніми пристроями, яка по запитах файлової системи здійснює передачу даних між дисками і оперативною пам’яттю.

Підсистема управління зовнішніми пристроями (підсистема вводу-виводу), виконує роль інтерфейсу для усіх зовнішніх пристроїв, підключених до комп’ютера. Спектр цих пристроїв дуже великий. Вони можуть істотно відрізнятися набором і послідовністю команд, за допомогою яких здійснюється обмін інформацією з процесором і пам’яттю комп’ютера, швидкістю роботи, кодуванням переданих даних, можливістю спільного використання і безліччю інших деталей.

Програма, що управляє конкретною моделлю зовнішнього пристрою і що враховує усі його особливості, зазвичай називається **драйвером** цього пристрою (від англійського drive - управляти, вести). Для користувача дуже важливо, щоб операційна система включала якомога більше різноманітних драйверів, оскільки це гарантує можливість підключення до комп’ютера великого числа зовнішніх пристроїв різних виробників. Від наявності відповідних драйверів багато в чому залежить успіх операційної системи на ринку (наприклад, відсутність багатьох необхідних драйверів зовнішніх пристроїв була однією з причин низької популярності OS/2). Створенням драйверів пристроїв займаються як розробники конкретної ОС, так і фахівці компаній, що випускають зовнішні пристрої.

Підтримка високорівневого уніфікованого інтерфейсу прикладного програмування до різнорідних пристроїв вводу-виводу є одним з найбільш важливих завдань ОС. З часу появи ОС UNIX такий уніфікований інтерфейс в більшості операційних систем будується на основі концепції файлового доступу. Ця концепція полягає в тому, що обмін з будь-яким зовнішнім пристроєм виглядає як обмін з файлом, що має ім’я і що є неструктурованою послідовністю байтів. Файлом може виступати як реальний файл на диску, так і алфавітно-цифровий термінал, друкуючий пристрій або мережевий адаптер.

**Захист даних і адміністрування**

Безпека даних обчислювальної системи забезпечується засобами відмовостійкості ОС, спрямованими на захист від збоїв і відмов апаратури та помилок програмного забезпечення, а також засобами захисту від несанкціонованого доступу.

Функції захисту ОС тісно пов’язані з функціями адміністрування, оскільки саме адміністратор визначає права користувачів при їх зверненні до різних ресурсів системи - файлів, каталогів, принтерів і тому подібне. Крім того, адміністратор обмежує можливості користувачів у виконанні тих або інших системних дій.

Важливими засобами захисту сучасних мережевих ОС є автентифікація, авторизація та аудит.

**Автентифікація** (authentication, лат. «встановлення достовірності») запобігає доступу до мережі небажаних осіб і дозволяє вхід для легальних користувачів. Автентифікацію слід відрізняти від ідентифікації. Ідентифікатори користувачів використовуються в системі з тими ж цілями, що і ідентифікатори будь-яких інших об’єктів, файлів, процесів, структур даних, але вони не пов’язані безпосередньо із забезпеченням безпеки. Ідентифікація полягає в повідомленні користувачем системі свого ідентифікатора, тоді як автентифікація - це процедура доказу користувачем того, що він є той, за кого себе видає, зокрема, доказ того, що саме йому належить введений ним ідентифікатор.

Засоби **авторизації** (authorization) контролюють доступ легальних користувачів до ресурсів системи (каталогів, файлів і периферійних пристроїв і ін.), надаючи кожному з них саме ті права, які йому були визначені адміністратором. Система авторизації також може контролювати можливість виконання користувачами різних системних функцій, таких як локальний доступ до сервера, установка системного часу, створення резервних копій даних, виключення сервера і тому подібне.

**Аудит** (auditing) – фіксація в системному журналі подій, пов’язаних з доступом до захищених системних ресурсів. Список подій, які необхідно відстежувати, визначає адміністратор ОС. Засоби обліку і спостереження забезпечують можливість виявити і зафіксувати важливі події, пов’язані з безпекою, або будь-які спроби створити, дістати доступ або видалити системні ресурси.

Підтримка відмовостійкості реалізується операційною системою, як правило, на основі **резервування**. Підтримка відмовостійкості також входить в обов’язки системного адміністратора. До складу ОС зазвичай входять утиліти, що дозволяють адміністраторові виконувати регулярні операції резервного копіювання для забезпечення швидкого відновлення важливих даних.

**Інтерфейс прикладного програмування**

Прикладні програмісти використовують у своїх додатках звернення до ОС, коли для виконання тих або інших дій їм вимагається особливий статус, який має тільки операційна система. Наприклад, в більшості сучасних ОС усі дії, пов’язані з управлінням апаратними засобами комп’ютера, може виконувати тільки ОС. Окрім цих функцій прикладний програміст може скористатися набором сервісних функцій ОС, які спрощують написання додатків.

Можливості операційної системи доступні прикладному програмістові у вигляді набору функцій, що називається **інтерфейсом прикладного програмування** (Application Programming Interface, **API**). Від кінцевого користувача ці функції приховані за оболонкою алфавітно-цифрового або графічного інтерфейсу користувача.

Для розробників додатків усі особливості конкретної операційної системи представлені особливостями її API. Тому операційні системи з різною внутрішньою організацією, але з однаковим набором функцій API здаються їм однією і тією ж ОС, що спрощує стандартизацію операційних систем і забезпечує переносимість додатків між ОС, що відповідають певному стандарту на API. Наприклад, наслідування загальних стандартів API UNIX, одним з яких є стандарт POSIX, дозволяє говорити про деяку узагальнену операційну систему UNIX, хоча численні версії цієї ОС від різних виробників іноді істотно відрізняються внутрішньою організацією.

Додатки виконують звернення до функцій API за допомогою системних викликів. Спосіб реалізації системних викликів залежить від структурної організації ОС, яка, у свою чергу, тісно пов’язана з особливостями апаратної платформи.

**Інтерфейс користувача**

Операційна система повинна забезпечувати зручний інтерфейс не лише для прикладних програм, але і для людини, що працює за терміналом. Ця людина може бути кінцевим користувачем, адміністратором ОС або програмістом.

У ранніх операційних системах пакетного режиму функції інтерфейсу користувача були зведені до мінімуму і не вимагали наявності терміналу. Команди мови управління завданнями набивалися на перфокарти, а результати виводилися на друкуючий пристрій.

Сучасні ОС підтримують розвинені функції інтерфейсу користувача для інтерактивної роботи за терміналами двох типів: алфавітно-цифровими і графічними.

При роботі за алфавітно-цифровим терміналом користувач має у своєму розпорядженні систему команд, потужність який відбиває функціональні можливості цієї ОС. Зазвичай командна мова ОС дозволяє запускати і зупиняти додатки, виконувати різні операції з файлами і каталогами, отримувати інформацію про стан ОС, адмініструвати систему. Команди можуть вводитися не лише в інтерактивному режимі з терміналу, але й виконуватись з **командного файлу**, що містить деяку послідовність команд.

Програмний модуль ОС, що відповідає за читання окремих команд або ж послідовності команд з командного файлу, іноді називають **командним інтерпретатором**.

Введення команди може бути спрощене, якщо операційна система підтримує **графічний інтерфейс користувача.**

2. Мережеві та розподілені ОС

**Мережева ОС** – це програмна система, яка керує та організовує використання ресурсів комп’ютерної мережі. Кожний комп’ютер в мережі в значній мірі автономний, тому під мережевою операційною системою в широкому значенні розуміється сукупність операційних систем окремих комп’ютерів, що взаємодіють з метою обміну повідомленнями і розділення ресурсів за єдиними правилами - протоколами.

Мережева ОС надає користувачеві деяку віртуальну обчислювальну систему, працювати з якою набагато простіше, ніж з реальним мережевим обладнанням. В той же час ця віртуальна система не повністю приховує розподілену природу свого реального прототипу, тобто є **віртуальною мережею**.

При використанні ресурсів комп’ютерів мережі користувач мережевої ОС завжди пам’ятає, що він має справу з мережевими ресурсами і що для доступу до них треба виконати деякі особливі операції, наприклад відобразити віддалений поділюючий каталог на певну локальну букву дисковода або поставити перед ім’ям каталогу ще й ім’я комп’ютера, на якому той розташований. Користувачі мережевої ОС зазвичай мають бути в курсі того, де зберігаються їх файли, і повинні використовувати явні команди передачі файлів для переміщення файлів з однієї машини на іншу.

Магістральним напрямом розвитку мережевих операційних систем є досягнення як можна вищої міри прозорості мережевих ресурсів. У ідеальному випадку мережева ОС повинна представити користувачеві мережеві ресурси у вигляді ресурсів єдиної централізованої віртуальної машини. Для такої операційної системи використовують спеціальну назву – **розподілена ОС**.

Розподілена ОС існує як єдина операційна система в масштабах обчислювальної системи. Кожен комп’ютер мережі, що працює під управлінням розподіленою ОС, виконує частину функцій цієї глобальної ОС. Розподілена ОС об’єднує усі комп’ютери мережі в тому сенсі, що вони працюють в тісній кооперації один з одним для ефективного використання усіх ресурсів комп’ютерної мережі.

3. Функціональні компоненти мережевої операційної системи

В мережевій операційній системі комп’ютера можна виділити декілька частин (рис.1):

**Засоби управління локальними ресурсами комп'ютера**

**(Локальна ОС)**

**Засоби надання власних ресурсів і послуг в загальне користування**

**Серверна частина**

**Засоби запиту доступу до віддалених ресурсів і послуг**

**(редиректор)**

**Клієнтська частина**

**Комунікаційні засоби ОС**

**в мережу**

Рис. 1 Структурна схема мережевої ОС

* **Засоби управління локальними ресурсами комп’ютера**: функції розподілу оперативної пам’яті між процесами, планування і диспетчеризація процесів, управління процесорами в мультипроцесорних комп’ютерах, управління периферійними пристроями і інші функції управління ресурсами локальних ОС.
* **Засоби надання власних ресурсів і послуг в загальне користування** - серверна частина ОС (сервер). Ці засоби забезпечують, наприклад, блокування файлів і записів, що необхідно для їх сумісного використовування; ведення довідників імен мережевих ресурсів; обробку запитів віддаленого доступу до власної файлової системи і бази даних; управління чергами запитів віддалених користувачів до своїх периферійних пристроїв.
* **Засоби запиту доступу до віддалених ресурсів і послуг** - клієнтська частина ОС (редиректор). Ця частина виконує розпізнавання і перенаправлення в мережу запитів до віддалених ресурсів від додатків і користувачів. При цьому запит поступає від додатку в локальній формі, а передається в мережу в іншій формі, відповідній вимогам серверу. Клієнтська частина також здійснює прийом відповідей від серверів і перетворення їх в локальний формат, так що для додатку виконання локальних і віддалених запитів однакове.
* **Комунікаційні засоби ОС**, за допомогою яких відбувається обмін повідомленнями в мережі. Ця частина забезпечує адресацію і буферизацію повідомлень, вибір маршруту передачі повідомлення по мережі, надійність передачі і т. п., тобто є засобом транспортування повідомлень.

Залежно від функцій, що покладаються на конкретний комп’ютер, в його операційній системі може бути відсутня клієнтська чи серверна частини.

На рис. 2 показана структурна схема взаємодії сервера та клієнта в мережі.

**Прикладна програма 1**

**Комунікаційні засоби ОС**

Рис. 2 Структурна схема взаємодії сервера та клієнта в мережі

**Локальна ОС**

**Клієнтська частина**

**Програма переспрямування**

**(редиректор)**

**Прикладна програма 2**

**Комунікаційні засоби ОС**

**Серверна частина**

**Локальна ОС**

В основі операційної системи клієнта є проста програма переспрямування (redirector). Вона резидентно міститься в пам’яті комп’ютера. Коли прикладна програма звертаються з запитом до операційної системи комп’ютера, ця програма перехоплює запит, аналізує, хто його може виконати, і спрямовує або в ОС того ж комп’ютера, або в мережу, до сервера, якому адресовано цей запит. Користувач не бачить, до яких ресурсів (свого комп’ютера чи мережі) він звертається. При цьому клієнтська частина перетворює запит з локальної форми в мережевий формат (\\server\dir1\file) і передає його транспортній підсистемі, яка відповідає за доставку повідомлень вказаному серверу. Серверна частина операційної системи сервера приймає запит, перетворює його в локальний формат (наприклад, с:, о:, z:, lpt1:) і передає для виконання своїй локальній ОС. Після того, як результат одержаний, сервер звертається до транспортної підсистеми і направляє відповідь клієнту, що видав запит. Клієнтська частина перетворює результат у відповідний формат і адресує його тому додатку, який видав запит.

4. Мережеві служби і мережеві сервіси

Сукупність серверної і клієнтської частин ОС, що надають доступ до конкретного типу ресурсу комп’ютера через мережу, називається мережевою службою. Мережева служба надає користувачам мережі деякий набір послуг. Ці послуги іноді називають також мережевим сервісом (service). Необхідно відмітити, що цей термін в технічній літературі перекладається і як «сервіс», і як «служба», і як «послуга». Хоча вказані терміни іноді використовуються як синоніми, слід мати на увазі, що в деяких випадках відмінність в значеннях цих термінів носить принциповий характер.

**Служба** – мережевий компонент, який реалізує деякий набір послуг.

**Сервіс** - опис того набору послуг, який надається цією службою. Таким чином, сервіс - це інтерфейс між споживачем послуг і постачальником послуг (службою).

Кожна служба пов’язана з певним типом мережевих ресурсів і/або певним способом доступу до цих ресурсів. Найбільш важливими для користувачів мережевих ОС є файлова служба і служба друку.

Серед мережевих служб можна виділити такі, які орієнтовані не на простого користувача, а на адміністратора. Такі служби використовуються для організації роботи мережі. Наприклад, централізовані довідкові служби (служби каталогів), які призначені для ведення бази даних про усіх користувачів мережі, а також програмні і апаратні компоненти мережі. Іншими прикладами мережевих служб, що надають сервіс адміністраторові, являються служба моніторингу мережі, що дозволяє захоплювати і аналізувати мережевий трафік, служба безпеки, у функції якої може входити, зокрема, виконання процедури логічного входу з перевіркою пароля, служба резервного копіювання і архівації.

Від того, наскільки багатий набір послуг пропонує ОС кінцевим користувачам, додаткам і адміністраторам мережі, залежить її позиція в загальному ряду мережевих ОС.

Мережеві служби за своєю природою є клієнт-серверними системами. Оскільки при реалізації будь-якого мережевого сервісу природно виникає джерело запитів (клієнт) і виконавець запитів (сервер), то і будь-яка мережева служба містить у своєму складі дві несиметричні частини - клієнтську і серверну. Мережева служба може бути представлена в операційній системі або обома (клієнтською і серверною) частинами, або тільки однією з них.

Принциповою різницею між клієнтом і сервером є те, що ініціатором виконання роботи мережевою службою завжди виступає клієнт, а сервер завжди знаходиться в режимі пасивного очікування запитів.

Звичайна взаємодія між клієнтською і серверною частинами стандартизується, так що один тип сервера може бути розрахований на роботу з клієнтами різного типу, реалізованими різними способами і, можливо, різними виробниками. Єдина умова для цього – клієнти і сервер повинні підтримувати загальний стандартний протокол взаємодії.

5. Варіанти впровадження мережевих служб в ОС

На практиці склалося декілька підходів до побудови мережевих ОС, що розрізняються глибиною впровадження мережевих служб в операційну систему (рис. 3), :

* мережеві служби об’єднані у вигляді деякого набору - **оболонки**;
* мережеві служби **вбудовані** в ОС;
* мережеві служби, що поставляються у вигляді **окремого продукту**.

Перші мережеві ОС були сукупністю вже існуючої локальної ОС і надбудованим над нею набором мережевих служб (**оболонки**). При цьому усі служби такого набору мали бути між собою узгоджені, тобто у своїй роботі вони могли звертатися один до одного, могли мати у своєму складі загальні компоненти, наприклад загальну підсистему автентифікації користувачів або єдиний інтерфейс користувача. Для роботи оболонки потрібна була наявність деякої локальної ОС, яка виконувала б звичайні функції, необхідні для управління апаратурою комп’ютера, і в середовищі якої виконувалися б мережеві служби, що становлять цю оболонку. Приклади поширених мережевих оболонок є – LAN Server і LAN Manager.

Надалі розробники мережевих ОС визнали ефективнішим підхід, при якому мережеві служби **вбудовувались** в основні модулі системи, що забезпечувало її логічну структуру, простоту експлуатації і модифікації, а також високу продуктивність. Важливо, що при такому підході відсутня надмірність. Якщо усі мережеві служби добре інтегровані, тобто розглядаються як невід’ємні частини ОС, то усі внутрішні механізми такої операційної системи можуть бути оптимізовані для виконання мережевих функцій.

ОС з вбудованими мережевими службами забезпечують вищі показники продуктивності і захищеності інформації в порівнянні з ОС, в яких мережеві служби являються надбудовою над локальною операційною системою.

Прикладами мережевих ОС зі вбудованими мережевими службами є усі сучасні версії UNIX, Windows, NetWare, OS/2.

Існує і третій спосіб реалізації мережевої служби – у вигляді **окремого продукту**. Наприклад, сервер віддаленого управління WinFrame (компанії Citrix), призначений для роботи в середовищі Windows NT. Він доповнює можливості вбудованого в Windows NT сервера віддаленого доступу Remote Access Server.

З часом мережева служба може отримати різні форми реалізації. Так, наприклад, компанія Novell планує поставляти довідкову службу NDS, яка вбудована в мережеву ОС NetWare, для інших ОС. Для цього служба NDS буде переписана у вигляді окремих продуктів, кожен з яких зважатиме на специфіку відповідної ОС. Вже є версії NDS для роботи в середовищах UNIX і Windows NT. А довідкова служба StreetTalk вже давно існує і у вигляді вбудованого модуля мережевої ОС Bayan Vines, і у складі оболонки ENS, і у вигляді окремого продукту для різних операційних систем.

Лекція 3

Архітектура мережевої операційної системи

1. Ядро і допоміжні модулі ОС

Будь-яка складна система повинна мати зрозумілу і раціональну структуру, тобто розділятися на частини – модулі, що мають цілком закінчене функціональне призначення з чітко обумовленими правилами взаємодії. Ясне розуміння ролі кожного окремого модуля істотно спрощує роботу по модифікації і розвитку системи.

Найбільш загальним підходом до структуризації операційної системи є розподіл усіх її модулів на дві групи:

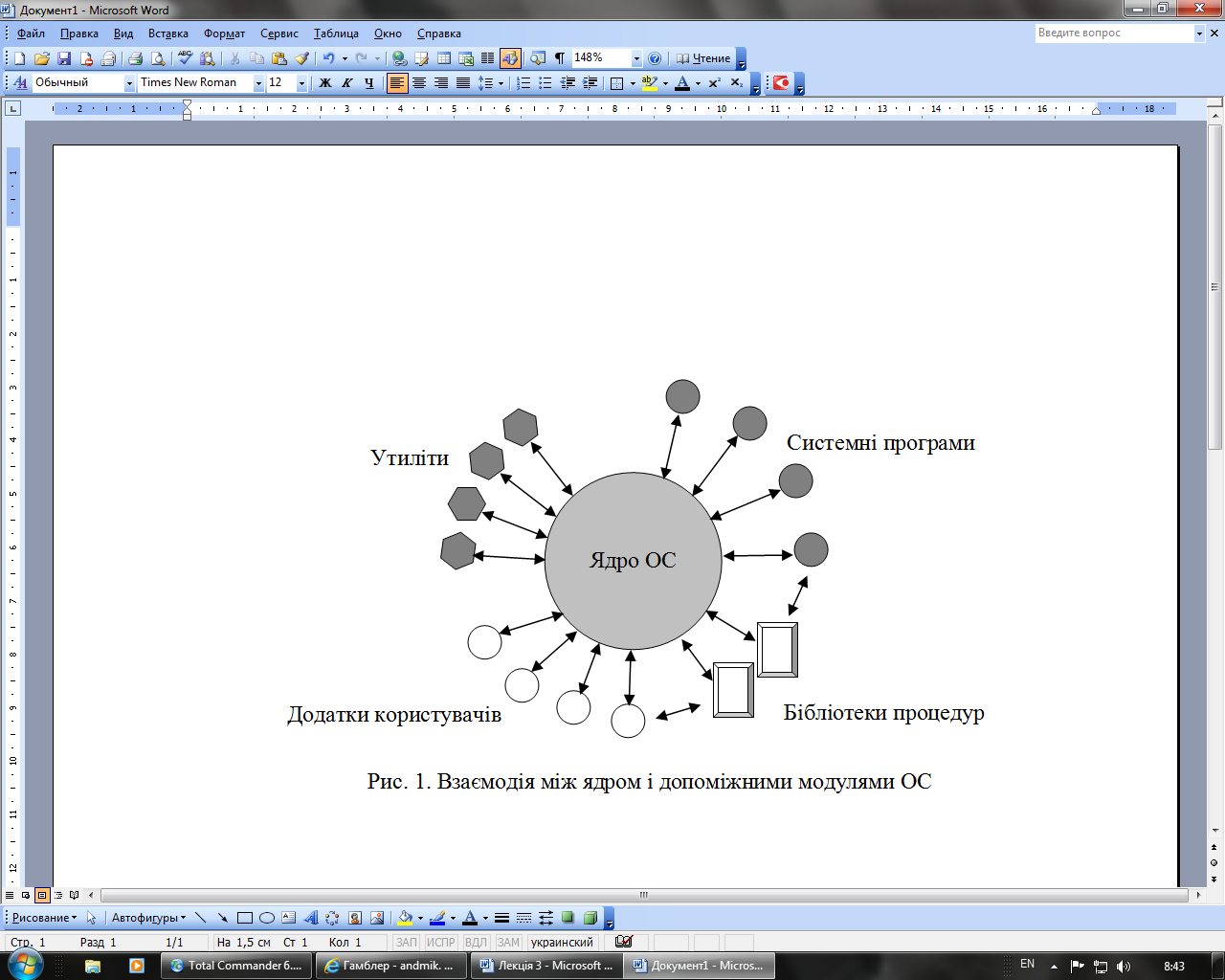
* ядро – модулі, що виконують основні функції ОС;
* модулі, що виконують допоміжні функції ОС.

**Модулі ядра** виконують такі базові функції ОС, як управління процесами, пам’яттю, пристроями вводу-виводу і тому подібне. Ядро складає серцевину операційної системи, без нього ОС є повністю непрацездатною і не зможе виконати жодну зі своїх функцій.

До складу ядра входять функції, що вирішують системні завдання організації обчислювального процесу, такі як перемикання контекстів, завантаження/вивантаження сторінок, обробка переривань. Ці функції недоступні для додатків. Інший клас функцій ядра служить для підтримки додатків, створюючи для них **прикладне програмне середовище**. Додатки можуть звертатися до ядра із запитами (системними викликами) для виконання тих або інших дій, наприклад для відкриття і читання файлу, отримання системного часу і т. д. Функції ядра, які можуть викликатися додатками, утворюють інтерфейс прикладного програмування - API.

Функції, що виконуються модулями ядра, є найбільш використовуваними функціями ОС, тому швидкість їх виконання визначає продуктивність усієї системи в цілому. Для забезпечення високої швидкодії ОС усі модулі ядра (або велика їх частина) постійно знаходяться в оперативній пам’яті, тобто є **резидентними**.

Допоміжні модулі ОС виконують корисні, але менш обов’язкові функції. Вони поділяються на наступні групи (рис. 1):

* **утиліти** – програми, що вирішують окремі завдання управління і супроводу комп’ютерної системи (наприклад, програми стиснення дисків, архівації даних);
* **системні програми** – текстові або графічні редактори, компілятори, компонувальники, відладчики;
* **бібліотеки процедур** різного призначення, що спрощують розробку додатків, наприклад бібліотека математичних функцій, функцій вводу-виводу і т. д.

Модулі ОС, що оформлені у вигляді утиліт, системних програм і бібліотек, звертаються до функцій ядра за допомогою системних викликів і, зазвичай, завантажуються в оперативну пам’ять тільки на час виконання своїх функцій, тобто є **транзитними**.

Рішення про те, чи являється деяка програма частиною ОС або ні, приймає виробник ОС. Серед багатьох чинників, здатних вплинути на це рішення, важливими є перспективи того, чи буде програма мати масовий попит у потенційних користувачів цієї ОС.

Для надійного управління ходом виконання додатків операційна система повинна мати по відношенню до додатків певні привілеї. Операційна система повинна мати виняткові повноваження також для того, щоб грати роль арбітра в суперечці додатків за ресурси комп’ютера в мультипрограмному режимі. Жоден додаток не повинен без відома ОС отримувати додаткову область пам’яті, займати процесор довше дозволеного ОС періоду часу, безпосередньо керувати спільно використовуваними зовнішніми пристроями.

Забезпечити привілеї операційній системі неможливо без спеціальних засобів апаратної підтримки. Апаратура комп’ютера повинна підтримувати як мінімум два режими роботи –**режим користувача** (user mode) і **привілейований режим**, який також називають **режимом ядра** (kernel mode), або **режимом супервізора** (supervisor mode).

Оскільки ядро виконує усі основні функції ОС, то найчастіше саме ядро стає тією частиною ОС, яка працює в привілейованому режимі.

Додатки ставляться в підлегле положення за рахунок заборони виконання в режимі користувача деяких критичних команд, пов’язаних з перемиканням процесора із завдання на завдання, управлінням пристроями вводу-виводу, доступом до механізмів розподілу і захисту пам’яті.

Повний контроль ОС над доступом до пам’яті досягається за рахунок того, що інструкції конфігурації механізмів захисту пам’яті дозволяється виконувати тільки в привілейованому режимі. Механізми захисту пам’яті використовуються ОС не лише для захисту своїх областей пам’яті від додатків, але і для захисту областей пам’яті, виділених ОС деякому додатку, від інших додатків. Кожен додаток повинен працювати у своєму адресному просторі.

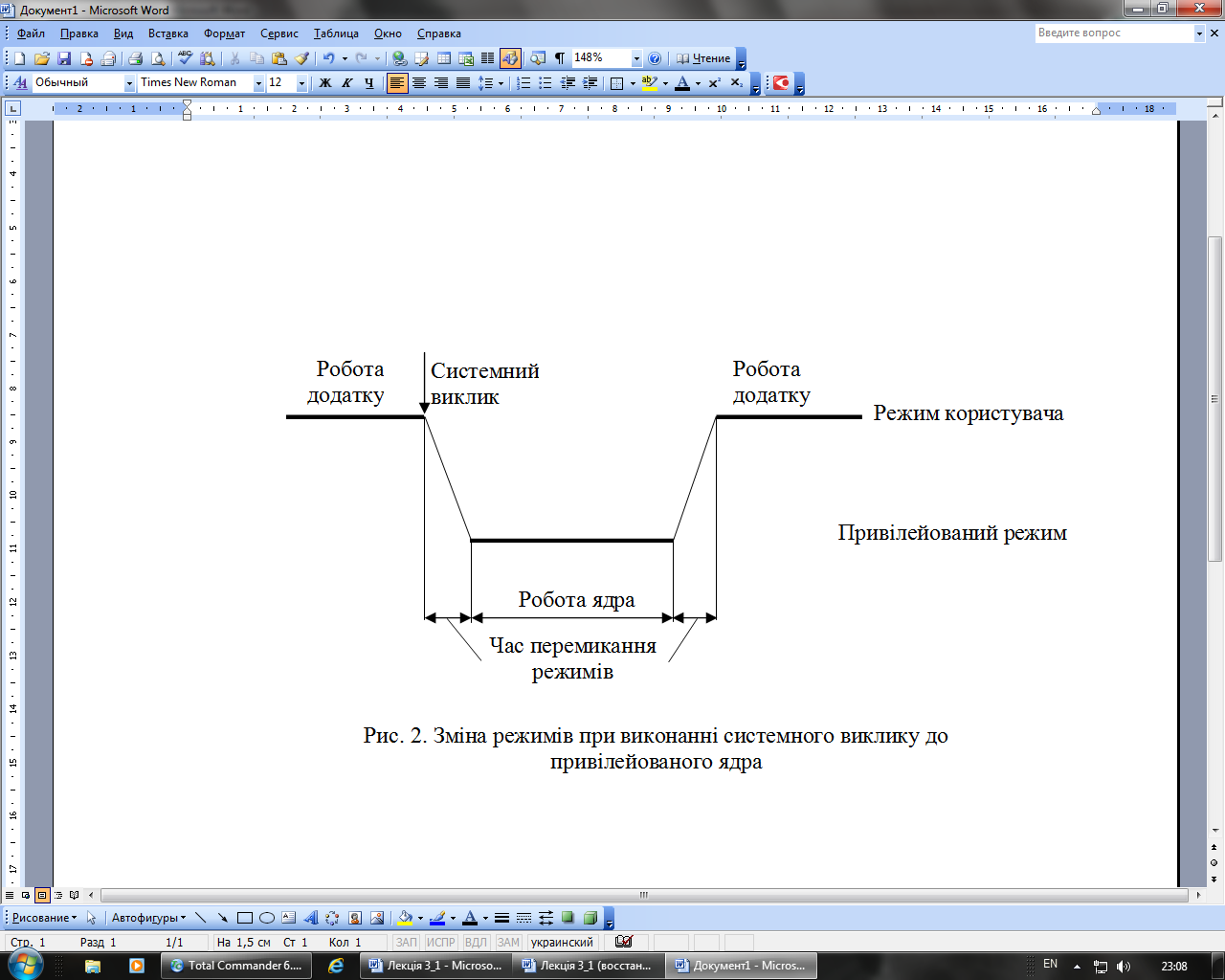
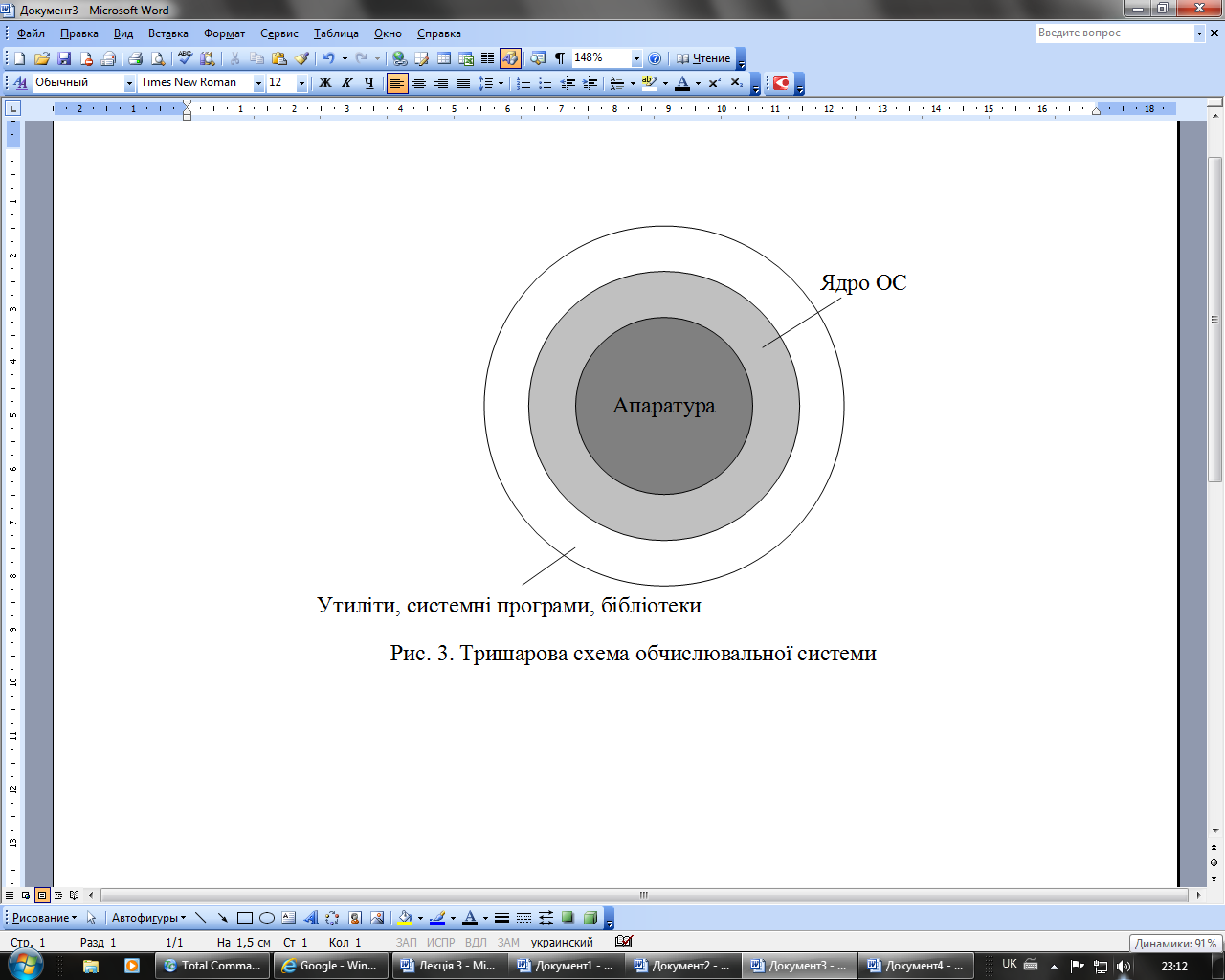
Підвищення стійкості операційної системи, що забезпечується переходом ядра в привілейований режим, досягається за рахунок деякого уповільнення виконання системних викликів. Системний виклик привілейованого ядра ініціює перемикання процесора з режиму користувача в привілейований режим, а при поверненні до додатка – навпаки (рис. 2).

Рис. 2. Зміна режимів при виконанні системного виклику до привілейованого ядра

В деяких випадках розробники ОС відступають від цього класичного варіанту архітектури, організовуючи роботу ядра і додатків в одному і тому ж режимі. Так, відома спеціалізована операційна система NetWare компанії Novell використовує привілейований режим процесорів Intel як для роботи ядра, так і для роботи своїх специфічних додатків – завантажуваних модулів NLM. При такій побудові ОС звернення додатків до ядра виконуються швидше, оскільки немає перемикання режимів, проте при цьому відсутній надійний апаратний захист пам’яті, яку займають модулі ОС, від некоректно працюючого додатку.

У одному режимі працюють також ядро і додатки тих операційних систем, які розроблені для процесорів, що взагалі не підтримують привілейованого режиму роботи. ОС MS-DOS, розроблена для цих процесорів, складалася з двох модулів msdos.sys і io.sys, що складали ядро системи, до яких з системними викликами зверталися командний інтерпретатор command.com, системні утиліти і додатки.

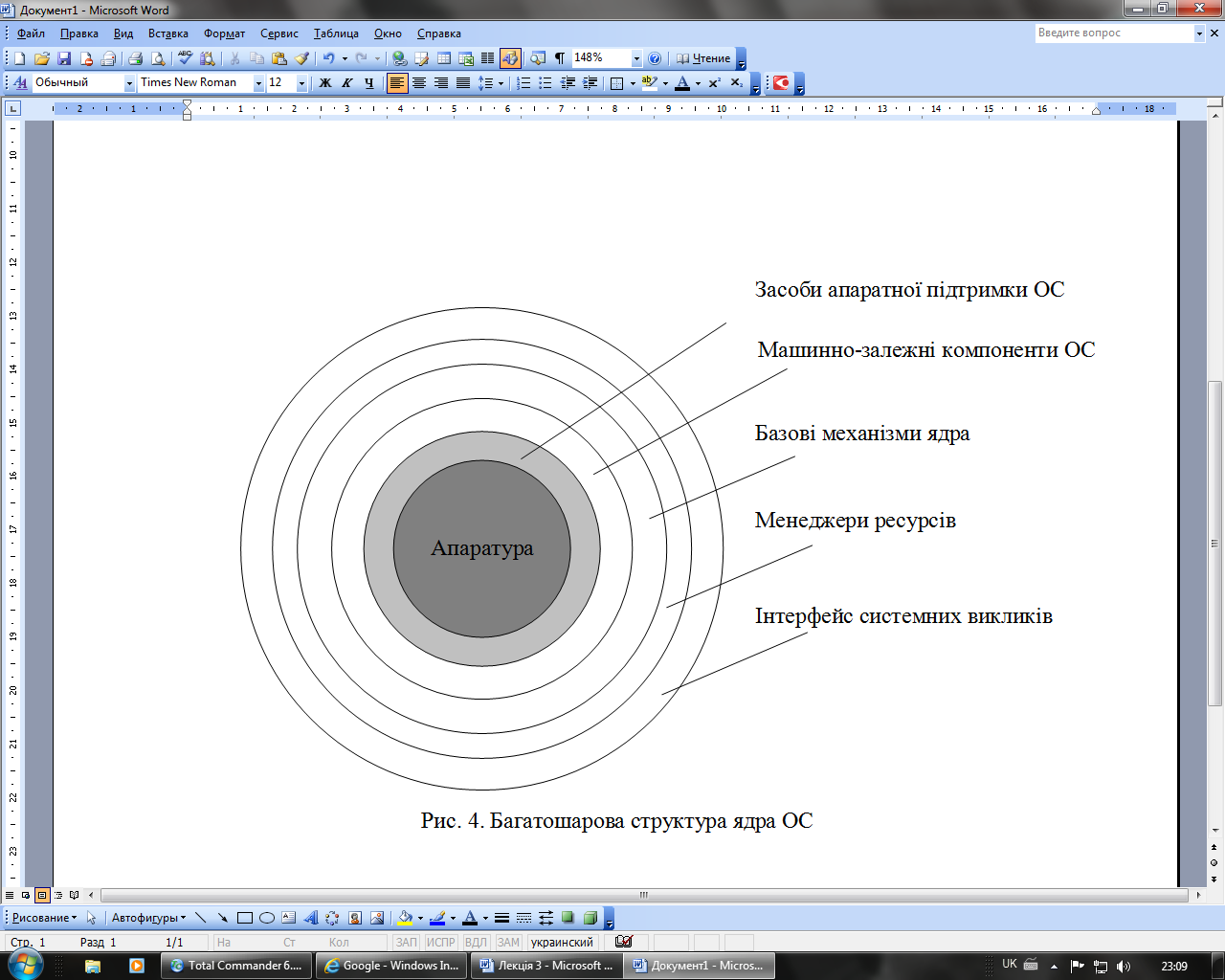
2. Архітектура на основі монолітного ядра

Обчислювальну систему, що працює під управлінням ОС на основі ядра, можна розглядати як систему, що складається з трьох ієрархічно розташованих шарів: нижній шар утворює апаратура, проміжний – ядро, а утиліти, оброблювальні програми і додатки, складають верхній шар системи (рис. 3).

Багатошарову структуру обчислювальної системи прийнято зображувати у вигляді системи концентричних кіл, ілюструючи той факт, що кожен шар може взаємодіяти тільки з суміжними шарами. При такій організації ОС додатки не можуть безпосередньо взаємодіяти з апаратурою, а тільки через шар ядра.

Оскільки ядро є складним багатофункціональним комплексом, то багатошаровий підхід зазвичай поширюється і на структуру ядра.

**Монолітне ядро** (monolithic kernel) – це така схема операційної системи, при якій усі компоненти її ядра є складовими частинами однієї програми, використовують загальні структури даних і взаємодіють один з одним шляхом безпосереднього виклику процедур. Усі частини монолітного ядра працюють в одному адресному просторі.

Монолітнеядро може складатися з наступних шарів (рис. 4):

**Засоби апаратної підтримки ОС**. Частина функцій ОС може виконуватися апаратними засобами. Тому іноді можна зустріти визначення операційної системи як сукупності програмних і апаратних засобів. Чіткої межі між програмною і апаратною реалізацією функцій ОС не існує - рішення про те, які функції ОС виконуватимуться програмно, а які апаратно, приймається розробниками апаратного і програмного забезпечення комп’ютера. Проте практично усі сучасні апаратні платформи мають деякий типовий набір засобів апаратної підтримки ОС, в який входять наступні компоненти:

* **засоби підтримки привілейованого режиму** – здійснюють перевірку допустимості виконання активною програмою інструкцій процесора при поточному рівні привілейованості;
* **засоби трансляції адрес** – виконують операції перетворення віртуальних адрес, які містяться в кодах процесу, в адреси фізичної пам’яті;
* **засоби перемикання процесів** – призначені для швидкого збереження контексту процесу, що призупиняється, і відновлення контексту процесу, який стає активним;
* **система переривань** – призначена для того, щоб оповістити процесор про виникнення в обчислювальній системі деякої непередбачуваної події або події, яка не синхронізована з циклом роботи процесора. Наприклад, завершення операції вводу-виводу зовнішнім пристроєм, некоректне завершення арифметичної операції, завершення інтервалу астрономічного часу;
* **системний таймер** – призначений для задання інтервалів часу, що використовуються для ініціалізації переривань, які обробляються процедурою ОС. Не слід плутати таймер ні з тактовим генератором, який виробляє сигнали, що синхронізують усі операції в комп’ютері, ні з системним годинником - електронною схемою, що працює на батареях, - який веде незалежний відлік часу і календарної дати.
* **засоби захисту областей пам’яті** – забезпечують на апаратному рівні перевірку можливості програмного коду здійснювати з даними певної області пам’яті такі операції, як читання, запис або виконання.

**Машинно-залежні компоненти ОС**. Цей шар утворюють програмні модулі, в яких відображається специфіка апаратної платформи комп’ютера. У ідеалі шар машинно-залежних компонентів ядра повністю екранує іншу частину ОС від конкретних деталей апаратної платформи (кеші, контролери переривань вводу-виводу і тому подібне), принаймні для того набору платформ, який підтримує ця ОС. В результаті відбувається підміна реальної апаратури деякою уніфікованою віртуальною машиною, однаковою для усіх варіантів апаратної платформи. Усі шари ОС, які лежать вище за шар машинно-залежних компонентів, можуть бути написані для управління саме цією віртуальною апаратурою. Таким чином, у розробників з’являється можливість створювати один варіант машинно-незалежної частини ОС (включаючи компоненти ядра, утиліти, системні оброблювальні програми) для усього набору підтримуваних платформ.

**Базові механізми ядра**. Цей шар виконує найбільш примітивні операції ядра, такі як програмне перемикання контекстів процесів, диспетчеризацію переривань, переміщення сторінок з пам’яті на диск і назад і тому подібне. Модулі цього шару не приймають рішень про розподіл ресурсів – вони тільки відпрацьовують прийняті вищими рівнями рішення. Наприклад, рішення про те, що в даний момент треба перервати виконання поточного процесу А і почати виконання процесу В, приймається менеджером процесів на вище розміщеному шарі, а шару базових механізмів передається тільки директива про те, що треба виконати перемикання з контексту поточного процесу на контекст процесу В.

**Менеджери ресурсів**. Цей шар складається з потужних функціональних модулів, що реалізовують стратегічні завдання по управлінню основними ресурсами обчислювальної системи. На цьому шарі працюють менеджери (диспетчери) процесів, вводу-виводу, файлової системи і оперативної пам’яті. Розбиття на менеджери може бути і декілька іншим, наприклад менеджер файлової системи іноді об’єднують з менеджером вводу-виводу, а функції управління доступом користувачів до системи в цілому і її окремих об’єктів доручають окремому менеджеру безпеки. Кожен з менеджерів веде облік вільних і використовуваних ресурсів певного типу і планує їх розподіл відповідно до запитів додатків. Для виконання прийнятих рішень менеджер звертається до нижчого шару – шару базових механізмів.

**Інтерфейс системних викликів**. Цей шар є самим верхнім шаром ядра і взаємодіє безпосередньо з додатками і системними утилітами, утворюючи прикладний програмний інтерфейс операційної системи. Функції API, які обслуговують системні виклики, надають доступ до ресурсів системи в зручній і компактній формі, без вказівки деталей їх фізичного розташування. Для здійснення комплексних дій системні виклики звертаються за допомогою до функцій шару менеджерів ресурсів.

Монолітні ядра мають довгу історію розвитку і удосконалення і, на даний момент, є найбільш архітектурно зрілими і придатними до експлуатації. В той же час, монолітність ядер ускладнює їх відладку, розуміння коду ядра, додавання нових функцій і можливостей, видалення "мертвого", непотрібного, успадкованого від попередніх версій коду. "Розбухання" коду монолітних ядер також підвищує вимоги до об’єму оперативної пам’яті, потрібного для функціонування ядра ОС. Це робить монолітну ядерну архітектуру малопридатною до експлуатації в системах, сильно обмежених за об’ємом оперативної пам’яті, наприклад, вбудовуваних системах, промислових мікроконтролерах і т. д.

Оскільки усе ядро працює в одному адресному просторі, збій в одному з компонентів може порушити працездатність усієї системи.

Прикладами поширених ОС з монолітним ядром – є ОС UNIX (BSD), Linux, MS-DOS.

Деякі старі монолітні ядра, особливо систем класу UNIX/Linux, вимагали перекомпіляції при будь-якій зміні складу обладнання. Більшість сучасних ядер дозволяють під час роботи підвантажувати модулі, що виконують частину функцій ядра. В цьому випадку компоненти операційної системи є не самостійними модулями, а складовими частинами однієї великої програми, званої монолітним ядром (monolithic kernel), яке є набором процедур, кожна з яких може викликати кожну. Усі процедури працюють в привілейованому режимі.

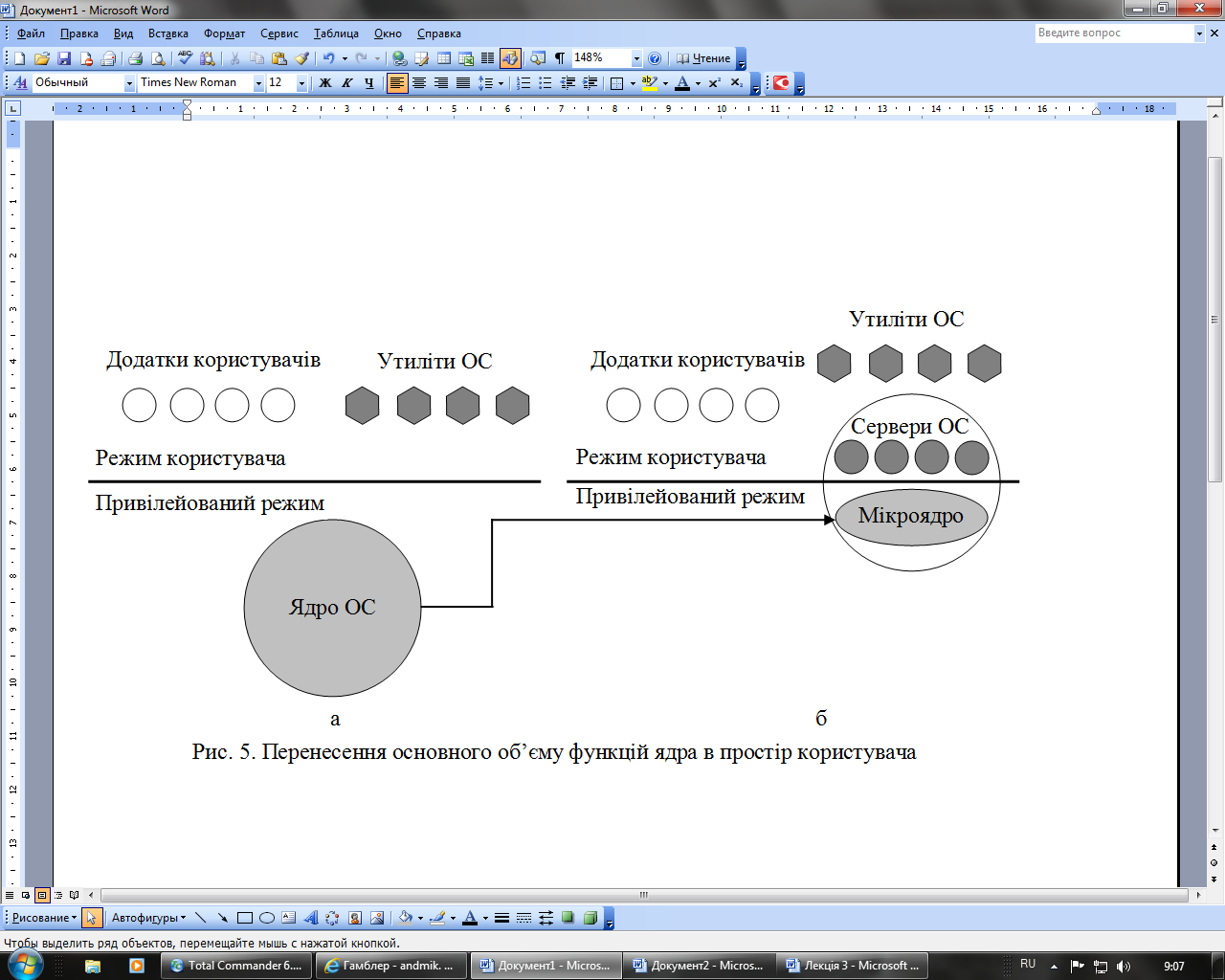
**Модульне ядро** – сучасна, вдосконалена модифікація архітектури монолітних ядер ОС.

На відміну від «класичних» монолітних ядер, що вважаються нині застарілими, модульні ядра, як правило, не вимагають повної перекомпіляції ядра при зміні складу апаратного забезпечення комп’ютера. Замість цього модульні ядра надають певний механізм підвантаження модулів ядра, що підтримують те або інше апаратне забезпечення (наприклад, драйверів). При цьому підвантаження модулів може бути як динамічним (виконується без перезавантаження ОС, в працюючій системі), так і статичним (виконується при перезавантаженні ОС після переконфігурації системи на завантаження тих або інших модулів).

Усі модулі ядра працюють в адресному просторі ядра і можуть користуватися усіма функціями, що надаються ядром. Тому модульні ядра продовжують залишатися монолітними. Модульність ядра здійснюється на рівні бінарного образу, а не на архітектурному рівні ядра, оскільки динамічно підвантажувані модулі завантажуються в адресний простір ядра і надалі працюють як інтегральна частина ядра. Модульні монолітні ядра не слід плутати з архітектурним рівнем модульності, що властивий мікроядрам і гібридним ядрам. Практично, динамічне завантаження модулів, це просто гнучкіший спосіб зміни образу ядра під час виконання - на відміну від перезавантаження з іншим ядром.

3. Мікроядерна архітектура

Мікроядерна архітектура є альтернативою архітектурі побудови операційної системи на основі монолітного ядра. В організації ОС на основі монолітного ядра усі основні функції операційної системи, що становлять багатошарове ядро, виконуються в привілейованому режимі. При цьому деякі допоміжні функції ОС оформляються у вигляді додатків і виконуються в режимі користувача разом із звичайними програмами користувача (стаючи системними утилітами або оброблювальними програмами). Кожен додаток режиму користувача працює у власному адресному просторі і захищений тим самим від будь-якого втручання інших додатків. Код ядра, що виконується в привілейованому режимі, має доступ до областей пам’яті усіх додатків, але сам повністю від них захищений. Додатки звертаються до ядра із запитами на виконання системних функцій.

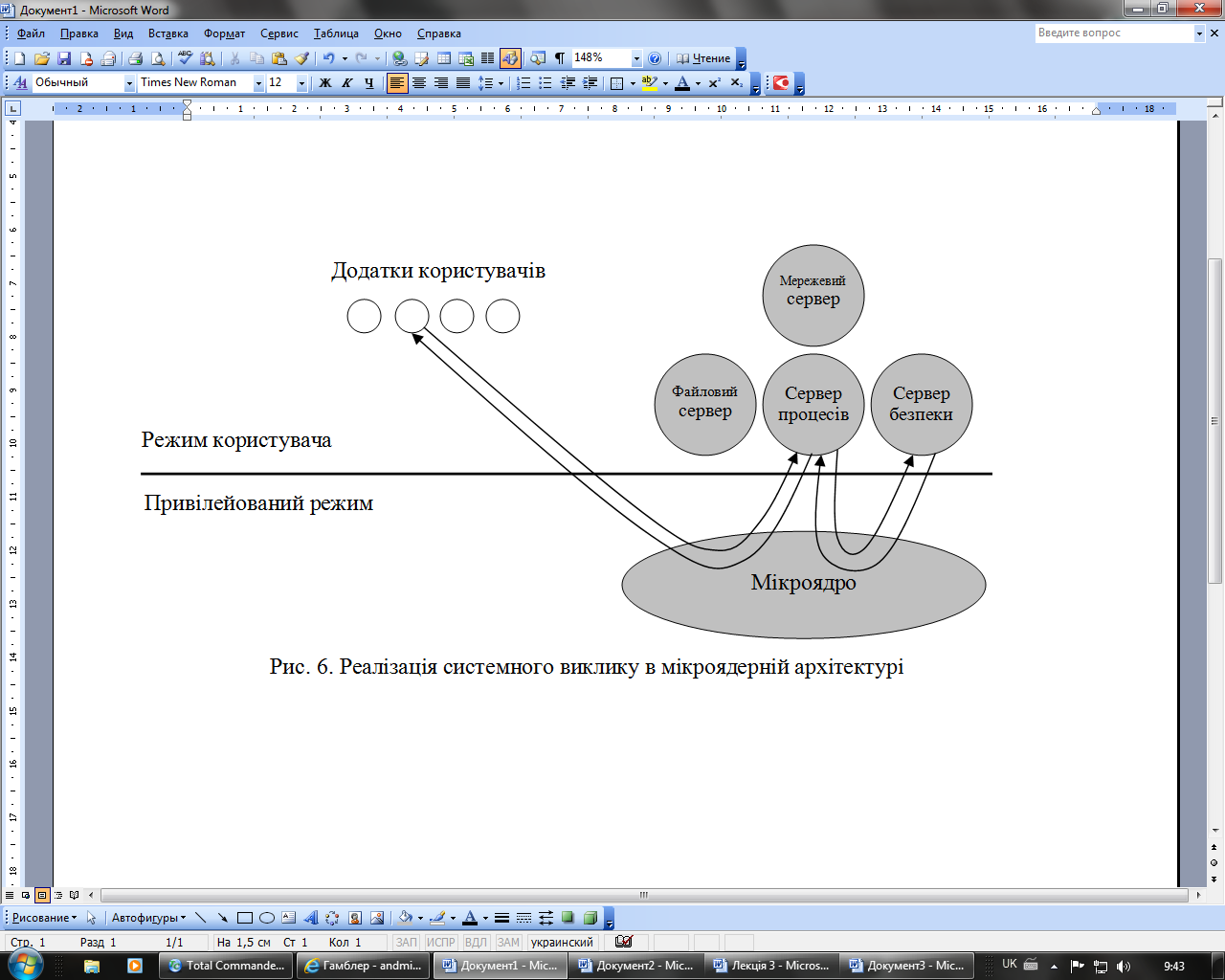
Суть мікроядерної архітектури полягає в наступному. У привілейованому режимі залишається працювати тільки дуже невелика частина ОС, звана мікроядром (рис. 5).

Мікроядро захищене від інших частин ОС і додатків. До складу мікроядра зазвичай входять машинно-залежні модулі, а також модулі, що виконують базові (але не усі) функції ядра по управлінню процесами, обробці переривань, управлінню віртуальною пам’яттю, пересилці повідомлень і управлінню пристроями вводу-виводу, пов’язані із завантаженням або читанням регістрів пристроїв. Набір функцій мікроядра зазвичай відповідає функціям шару базових механізмів звичайного ядра. Такі функції ОС не неможливо, виконати в просторі користувача.

Усі інші більш високорівневі функції ядра оформляються у вигляді додатків, що працюють в режимі користувача. Однозначного рішення про те, які з системних функцій треба залишити в привілейованому режимі, а які перенести в режим користувача, не існує. У загальному випадку багато менеджерів ресурсів, що є невід’ємними частинами звичайного ядра, - файлова система, підсистеми управління віртуальною пам’яттю і процесами, менеджер безпеки і ін., - стають периферійними модулями, що працюють в режимі користувача .

Менеджери ресурсів, що працюють в режимі користувача, мають принципові відмінності від звичайних утиліт і оброблювальних програм ОС, хоча при мікроядерній архітектурі усі ці програмні компоненти також оформлені у вигляді додатків. Утиліти і оброблювальні програми викликаються в основному користувачами. Ситуації, коли одному додатку необхідне виконання функції (процедури) іншого додатку, виникають вкрай рідко. Тому в ОС з монолітною архітектурою відсутній механізм, за допомогою якого однин додаток міг би викликати функції іншого.

Зовсім інша ситуація виникає, коли у формі додатка оформляється частина ОС. Основним призначенням такого додатку є обслуговування запитів інших додатків, наприклад створення процесу, виділення пам’яті, перевірка прав доступу до ресурсу і т. д. Саме тому менеджери ресурсів, які винесені в режим користувача, називаються **серверами ОС**, тобто модулями, основним призначенням яких є обслуговування запитів локальних додатків і інших модулів ОС. Таким чином, для реалізації мікроядерної архітектури необхідною умовою є наявність в ОС зручного і ефективного способу виклику процедур одного процесу з іншого.

Схемний механізм звернення до функцій ОС, оформлених у вигляді серверів, виглядає таким чином (рис. 6). Клієнт, яким може бути або прикладна програма, або інший компонент ОС, робить запит на виконання деякої функції у відповідного сервера, посилаючи йому повідомлення. Безпосередня передача повідомлень між додатками неможлива, оскільки їх адресні простори ізольовані один від одного. Мікроядро, що виконується в привілейованому режимі, має доступ до адресних просторів кожного з цих додатків і тому може працювати посередником. Мікроядро спочатку передає повідомлення, що містить ім’я і параметри викликаючої процедури потрібному серверу, потім сервер виконує замовлену операцію, після чого ядро повертає результати клієнтові за допомогою іншого повідомлення. Таким чином, робота мікроядерної ОС відповідає відомій моделі клієнт-сервер, в якій роль транспортних засобів виконує мікроядро.

ОС, засновані на концепції мікроядра, у високій мірі задовольняють більшості вимог, що висуваються до сучасних ОС, володіючи високою переносимістю, розширюваністю, надійністю і створюючи хороші передумови для підтримки розподілених додатків. За ці переваги доводиться платити зниженням продуктивності, і це є основним недоліком мікроядерної архітектури.

Високий рівень переносимості та розширюваності обумовлений тим, що увесь машинно-залежний код ізольований в мікроядрі, тому для перенесення системи на новий процесор вимагається менше змін і усі вони логічно згруповані разом.

Використання мікроядерної моделі підвищує надійність ОС. Кожен сервер виконується у вигляді окремого процесу у власній області пам’яті і таким чином захищений від інших серверів ОС, що не спостерігається в традиційній ОС, де усі модулі ядра можуть впливати один на одного. Іншим потенційним джерелом підвищення надійності ОС є зменшений об’єм коду мікроядра в порівнянні з традиційним ядром - це знижує ймовірність появи помилок програмування.

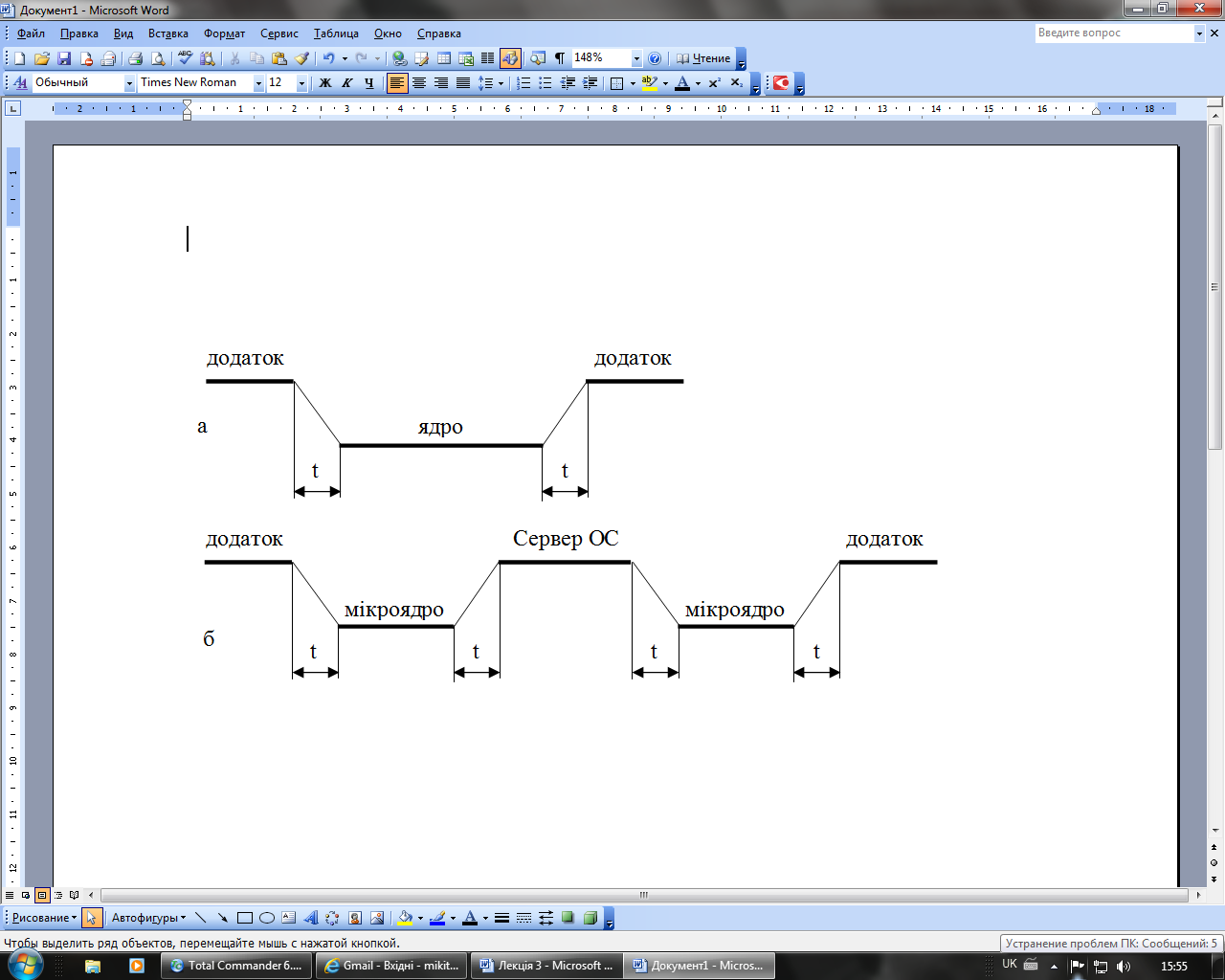
При класичній організації ОС (рис. 7, а) виконання системного виклику супроводжується двома перемиканнями режимів, а при мікроядерній організації (рис. 7, 6) - чотирма. Таким чином, операційна система на основі мікроядра за інших рівних умов завжди буде менш продуктивною, чим ОС з класичним ядром. Саме з цієї причини мікроядерний підхід не набув такого широкого поширення, яке йому передрікали.

Рис. 7. Зміна режимів при виконанні системного виклику

Для того, щоб мікроядерна операційна система за швидкістю не поступалася операційним системам на базі монолітного ядра, вимагається дуже акуратно проектувати розбиття системи на компоненти, прагнучи мінімізувати взаємодію між ними. Таким чином, основна складність при створенні мікроядерних ОС - необхідність дуже акуратного проектування.

Мікроядра типу ядра ОС і GNU Hurd розвиваються повільно, набагато повільніше, ніж Linux і ядро систем сімейства BSD. За словами творця Minix3, Таненбаума, він намагається "побудувати наднадійну (very highly reliable) систему. Вона може використовуватися у тому числі на серверах, яким потрібні роки безвідмовної роботи".

Прикладами поширених мікроядерних ОС є Symbian OS (починаючи c версії Symbian OS v8.1 вона є наноядерною), QNX, Minix3.

**Екзоядро** - ядро операційної системи комп’ютерів, що надає лише функції для взаємодії між процесами і безпечного виділення і звільнення ресурсів.

Екзо - приставка, що означає щось зовнішнє, що знаходиться зовні.

На відміну від ОС на основі мікроядра, ОС, що базуються на екзоядрі, забезпечують набагато більшу ефективність за рахунок відсутності необхідності в перемиканні між процесами при кожному зверненні до устаткування.

Архітектура на основі екзоядер є подальшим розвитком і удосконаленням мікроядерної архітектури і одночасно посилює вимоги до мінімалістичності і простоти коду ядра.

libOS може забезпечувати довільний набір абстракцій, сумісний з тією або іншою вже існуючою операційною системою, наприклад Linux або Windows.

**Наноядро** - архітектура ядра операційної системи комп’ютерів, у рамках якої вкрай спрощене і мінімалістичне ядро виконує лише одне завдання - обробку апаратних переривань, генерованих пристроями комп’ютера. Після обробки переривань від апаратури наноядро, у свою чергу, посилає інформацію про результати обробки (наприклад, отримані з клавіатури символи) вищерозміщеному програмному забезпеченню за допомогою того ж механізму переривань.

Найчастіше в сучасних комп’ютерах наноядра використовуються для віртуалізації апаратного забезпечення реальних комп’ютерів або для реалізації механізму гіпервізора, з метою дозволити декільком або багатьом різним операційним системам працювати одночасно і паралельно на одному і тому ж комп’ютері.

**Гіпервізор (чи Монітор віртуальних машин)** – програма або апаратна схема, що забезпечує або дозволяє одночасне, паралельне виконання декількох операційних систем на одному комп’ютері. Гіпервізор також забезпечує ізоляцію операційних систем один від одного, захист і безпеку, розподіл ресурсів між різними запущеними ОС і управління ресурсами.

Наноядра також можуть використовуватися для забезпечення переносимості операційних систем на різне апаратне забезпечення або для забезпечення можливості запуску «старої» операційної системи на новому, несумісному апаратному забезпеченні без її повного переписування.

Наноядро може бути настільки малим і примітивним, що навіть найважливіші пристрої, що знаходяться безпосередньо на материнській платі або на платі контролера вбудовуваного пристрою, такі, як таймер або програмований контролер переривань, обслуговуються спеціальними драйверами пристроїв, а не безпосередньо ядром. Такі надмінімалістичні наноядра називають іноді **пікоядрами**.

Термін «наноядро» іноді неформально використовується для опису дуже маленьких, спрощених і легких мікроядер.

4. Архітектура на основі гібридного ядра

**Гібридне ядро** (Hybrid kernel) - модифіковані мікроядра (мінімальна реалізація основних функцій ядра операційної системи комп’ютера), що дозволяють для прискорення роботи запускати "несуттєві" частини в просторі ядра.

Усі розглянуті підходи до побудови операційних систем мають свої переваги і недоліки. В більшості випадків сучасні операційні системи використовують різні комбінації цих підходів.

Прикладом змішаного підходу може служити можливість запуску операційної системи з монолітним ядром під управлінням мікроядра. Такі побудована ОС BSD 4.4, що заснована на мікроядрі Mach. Мікроядро забезпечує управління віртуальною пам’яттю і роботу низькорівневих драйверів. Усі інші функції, у тому числі взаємодія з прикладними програмами, здійснюється монолітним ядром. Цей підхід сформувався в результаті спроб використовувати переваги мікроядерної архітектури, зберігаючи по можливості добре відлагоджений код монолітного ядра.

Найтісніше елементи мікроядерної архітектури і елементи монолітного ядра переплетені в ядрі Windows NT. Хоча Windows NT часто називають мікроядерною операційною системою, це не зовсім так. Мікроядро NT занадто велике (більше 1 Мбайт), щоб носити приставку "мікро". Компоненти ядра Windows NT взаємодіють один з одним шляхом передачі повідомлень, як і належить в мікроядерних операційних системах. В той же час усі компоненти ядра працюють в одному адресному просторі і активно використовують загальні структури даних, що властиво операційним системам з монолітним ядром. Причина проста: чисто мікроядерний дизайн комерційно невигідний, оскільки неефективний.

Таким чином, ядра сімейства Windows NT (Windows NT, Windows 2000, Windows XP, Windows Vista, Windows 7) можна з повним правом назвати гібридними.

Лекція 4

Планування процесів

1. Стан процесів

Найважливішою частиною операційної системи є підсистема управління процесами. **Процес** (чи по-іншому, **задача**) – абстракція, що описує виконуючу програму. Для ОС процес є одиницею роботи, заявкою на використання системних ресурсів. Підсистема управління процесами планує виконання процесів, тобто розподіляє процесорний час між декількома одночасно існуючими в системі процесами, а також займається створенням і знищенням процесів, забезпечує процеси необхідними системними ресурсами, підтримує взаємодію між процесами.

У багатозадачній системі процес може знаходитися в одному з трьох основних станів:

* **виконання** – активний стан процесу, під час якого процес має усі необхідні ресурси і безпосередньо виконується процесором;
* **очікування** – пасивний стан процесу, процес заблокований, він не може виконуватися із власних причин, він чекає здійснення деякої події, наприклад, завершення операції вводу-виводу, отримання повідомлення від іншого процесу, звільнення необхідного йому ресурсу;
* **готовність** – також пасивний стан процесу, але в цьому випадку процес заблокований у зв’язку із зовнішніми по відношенню до нього обставинами: процес має усі потрібні для нього ресурси, він готовий виконуватися, проте процесор зайнятий виконанням іншого процесу.

В ході життєвого циклу кожен процес переходить з одного стану в інший відповідно до алгоритму планування процесів, що реалізовується в цій операційній системі. Типовий граф станів процесу показаний на рис.1.

В стані ВИКОНАННЯ в однопроцесорній системі може знаходитися тільки один процес, а в кожному із станів ОЧІКУВАННЯ і ГОТОВНІСТЬ - декілька процесів. Ці процеси утворюють черги відповідно очікуючих і готових процесів. Життєвий цикл процесу починається із стану ГОТОВНІСТЬ, коли процес готовий до виконання і чекає своєї черги. При активізації процес переходить в стан ВИКОНАННЯ і знаходиться в нім до тих пір, поки або він сам звільнить процесор, перейшовши в стан ОЧІКУВАННЯ якої-небудь події, або буде примусово "витіснений" з процесора, наприклад, внаслідок закінчення відведеного цьому процесу кванту процесорного часу. У останньому випадку процес повертається в стан ГОТОВНІСТЬ. У цей же стан процес переходить із стану ОЧІКУВАННЯ, після того, як очікувана подія станеться.

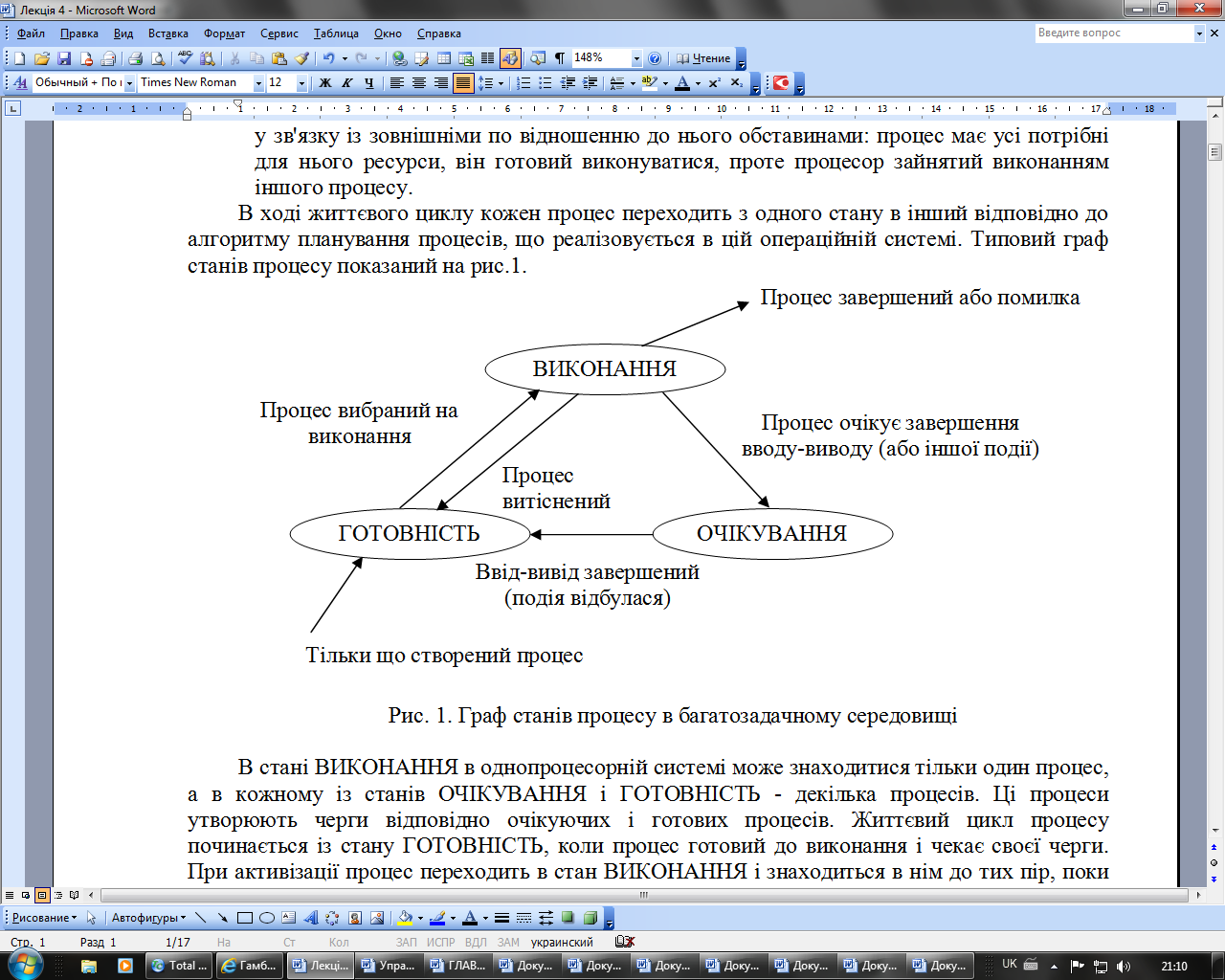
****

Рис. 1. Граф станів процесу в багатозадачному середовищі

2. Створення процесів

При управлінні процесами операційна система використовує два основні типи інформаційних структур: дескриптор процесу і контекст процесу.

**Дескриптор процесу** містить таку інформацію про процес, яка потрібна ядру протягом усього життєвого циклу процесу незалежно від того, знаходиться він в активному або пасивному стані, знаходиться образ процесу в оперативній пам’яті або вивантажений на диск. (Образом процесу називається сукупність його кодів і даних.)

Дескриптори окремих процесів об’єднані в список, що утворює таблицю процесів. Пам’ять для таблиці процесів відводиться динамічно в області ядра. На підставі інформації, що міститься в таблиці процесів, операційна система здійснює планування і синхронізацію процесів. У дескрипторі прямо або побічно (через покажчики, на пов’язані з процесом структури) знаходиться інформація про стан процесу, про розташування образу процесу в оперативній пам’яті і на диску, про значення окремих складових пріоритету, про ідентифікатор користувача, що створив процес, про споріднені процеси, про події, здійснення яких чекає цей процес, і деяка інша інформація.

**Контекст процесу** містить менш оперативну, але об’ємнішу частину інформації про процес, необхідну для відновлення виконання процесу з перерваного місця: вміст регістрів процесора, коди помилок виконуваних процесором системних викликів, інформація про усі відкриті цим процесом файли і незавершені операції вводу-виводу і інші дані, що характеризують стан обчислювального середовища у момент переривання. Контекст, так само як і дескриптор процесу, доступний тільки програмам ядра, тобто знаходиться у віртуальному адресному просторі операційної системи, проте він зберігається не в області ядра, а безпосередньо примикає до образу процесу і переміщається разом з ним, якщо це необхідно, з оперативної пам’яті на диск.

Дескриптор процесу в порівнянні з контекстом містить більш оперативну інформацію, яка має бути легко доступна підсистемі планування процесів. Контекст процесу містить менш актуальну інформацію і використовується операційною системою тільки після того, як прийнято рішення про відновлення перерваного процесу.

Черги процесів є дескрипторами окремих процесів, об’єднаними в списки. Таким чином, кожен дескриптор містить принаймні один покажчик на інший дескриптор, що є сусідом з ним у черзі. Така організація черг дозволяє легко їх перевпорядковувати, включати і виключати процеси, переводити процеси з одного стану в інший.

Програмний код тільки тоді почне виконуватися, коли для нього операційною системою буде створений процес. Створити процес – означає:

* створити інформаційні структури, що описують цей процес, тобто його дескриптор і контекст;
* включити дескриптор нового процесу в чергу готових процесів;
* завантажити кодовий сегмент процесу в оперативну пам’ять або в область свопінгу.

3. Планування і диспетчеризація процесів

Упродовж існування процесу виконання його може бути багаторазово перерване і продовжене. Перехід від виконання одного процесу до іншого здійснюється в результаті планування і диспетчеризації. Робота по визначенню того, в який момент необхідно перервати виконання поточного активного процесу і якому процесу надати можливість виконуватися, називається **плануванням**. Планування потоків здійснюється на основі інформації, що зберігається в описувачах процесів. Таким чином, планування процесів включає рішення двох завдань:

* визначення моменту часу для зміни поточного активного процесу;
* вибір процесу на виконання з черги готових процесів;

**Диспетчеризація** полягає в реалізації знайденого в результаті планування рішення, тобто в перемиканні процесора з одного процесу на іншій. Перш ніж перервати виконання процесу, ОС запам’ятовує його контекст для того, щоб згодом використовувати цю інформацію для наступного відновлення виконання цього процесу. Отже, диспетчеризація зводиться до наступного:

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

Оскільки операція перемикання контекстів істотно впливає на продуктивність обчислювальної системи, програмні модулі ОС виконують диспетчеризацію потоків спільно з апаратними засобами процесора.

4. Алгоритми планування процесів

Існує дві групи найбільш поширених алгоритмів планування процесів: алгоритми, засновані на квантуванні і алгоритми, засновані на пріоритетах.

В алгоритмах планування, що засновані на **квантуванні**, кожному процесу почергово для виконання надається обмежений неперервний період процесорного часу – **квант**. Зміна активного процесу відбувається, якщо:

* процес завершився і покинув систему,
* сталася помилка,
* процес перейшов в стан ОЧІКУВАННЯ,
* вичерпався квант процесорного часу, відведений цьому процесу.

Процес, який вичерпав свій квант, переводиться в стан ГОТОВНІСТЬ і чекає, коли йому буде наданий новий квант процесорного часу, а на виконання відповідно до певного правила вибирається новий процес з черги готових. Таким чином, жоден процес не займає процесор надовго, тому квантування широко використовується в системах розподілу часу. Граф станів процесу, зображений на рис. 2, відповідає алгоритму планування, заснованому на квантуванні.

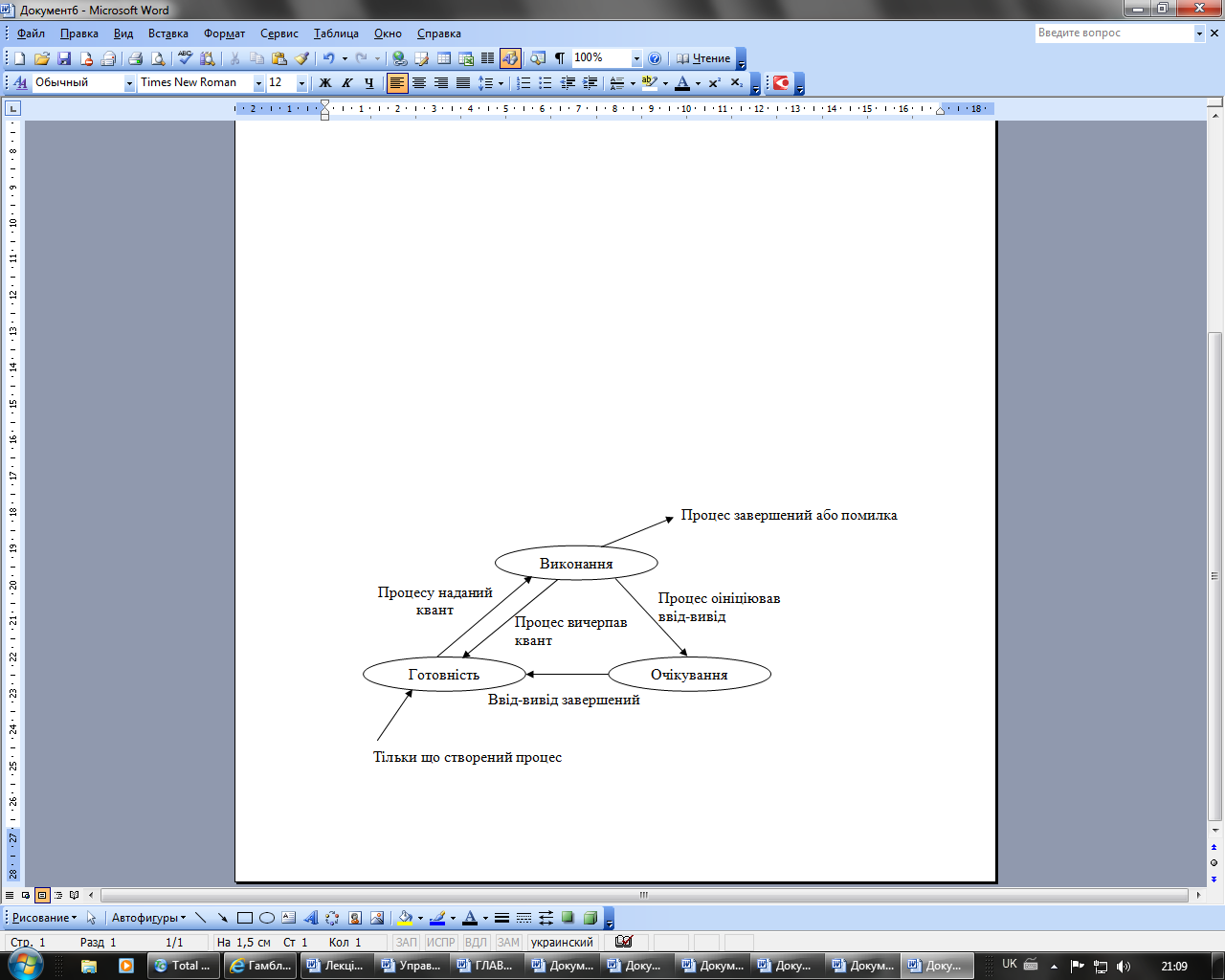
Кванти, що виділяються процесам, можуть бути однаковими для усіх процесів або різними. Кванти, що виділяються одному процесу, можуть бути фіксованої величини або змінюватися в різні періоди життя процесу. Процеси, які не повністю використовували виділений їм квант, можуть отримати або не отримати компенсацію у вигляді привілеїв при наступному обслуговуванні. По різному може бути організована черга готових процесів: циклічно, за правилом «перший прийшов - перший обслужився» (FIFO) або за правилом «останній прийшов - перший обслужився» (LIFO).

Рис. 2. Граф станів процесу в системі з квантуванням

В алгоритмах планування, що засновані на пріоритетах, припускається наявність у процесів деякої наперед відомої характеристики – пріоритету, на підставі якої визначається порядок їх виконання. **Пріоритет** – це число, що характеризує міру привілейованості процесу при використанні ресурсів обчислювальної машини, зокрема, процесорного часу: чим вищий пріоритет, тим вищі за привілеї, тим менше часу проводитиме процес у чергах.

Пріоритет може виражатися цілим або дробовим, позитивним або негативним значенням. Пріоритет може призначатися адміністратором системи залежно від важливості роботи або внесеної плати, або обчислюватися самою ОС за певними правилами, він може залишатися фіксованим упродовж усього життя процесу або змінюватися в часі відповідно до деякого закону. В останньому випадку пріоритети називаються **динамічними** на відміну від незмінних, **фіксованих** пріоритетів.

Існує два різновиди пріоритетних алгоритмів: алгоритми, що використовують відносні пріоритети, і алгоритми, що використовують абсолютні пріоритети.

У обох випадках вибір процесу на виконання з черги готових процесів здійснюється однаково: вибирається процес, що має найвищий пріоритет. По різному вирішується проблема визначення моменту зміни активного процесу. На рис. 3 показані графи станів процесу для алгоритмів з відносними (а) і абсолютними (б) пріоритетами.

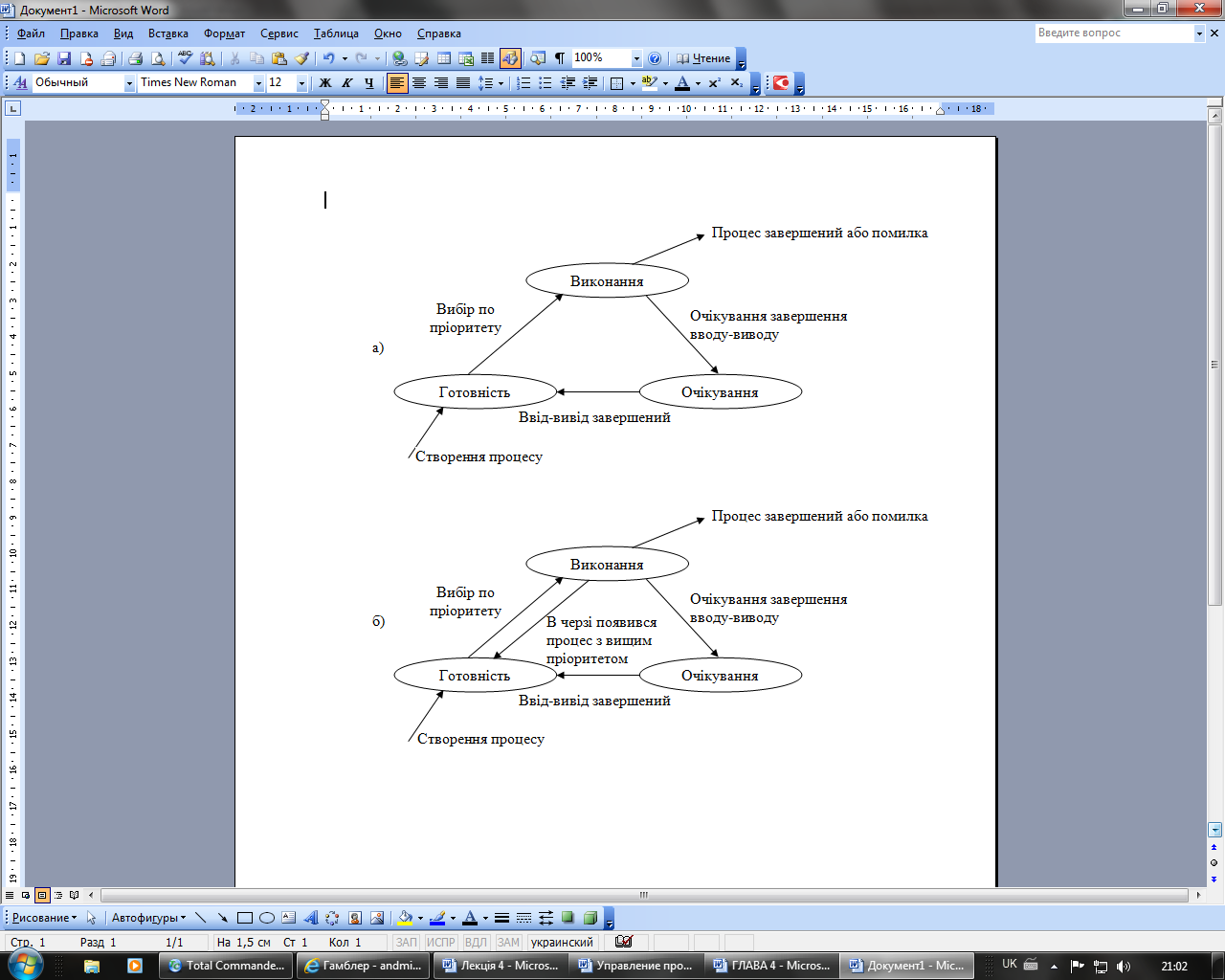
У системах з відносними пріоритетами активний процес виконується до тих пір, поки він сам не покине процесор, перейшовши в стан ОЧІКУВАННЯ (або ж станеться помилка, або процес завершиться). У системах з абсолютними пріоритетами виконання активного процесу переривається ще за однієї умови: якщо в черзі готових процесів з’явився процес, пріоритет якого вище пріоритету активного процесу. В цьому випадку перерваний процес переходить в стан готовності.

Рис. 3. Графи станів процесів в системах з відносними (а) та абсолютними (б) пріоритетами

У багатьох ОС алгоритми планування побудовані з використанням як квантування, так і пріоритетів. Наприклад, в основі планування лежить квантування, але величина кванта і/або порядок вибору процесу з черги готових процесів визначається пріоритетами процесів.

5. Витісняючі і невитісняючі алгоритми багатозадачності

Існує два основних класи алгоритмів багатозадачності: витісняючі і невитісняючі.

* **Невитісняюча багатозадачність (Non-preemptive multitasking)** – це спосіб планування процесів, при якому активний процес виконується до тих пір, поки він сам, за власною ініціативою, не віддасть управління ОС для того, щоб та вибрала з черги інший, готовий до виконання процес.
* **Витісняюча багатозадачність (Preemptive multitasking)** – це такий спосіб, при якому рішення про перемикання процесора з виконання одного процесу на виконання іншого процесу приймається ОС, а не активним процесом.

Основною відмінністю між витісняючими і невитісняючими варіантами багатозадачності є міра централізації механізму планування процесів. При витісняючій багатозадачності механізм планування процесів повністю зосереджений в ОС. При цьому ОС виконує наступні функції: визначає момент зняття з виконання активного процесу, запам’ятовує його контекст, вибирає з черги готових процесів наступний і запускає його на виконання, завантажуючи його контекст.

При невитісняючій багатозадачності механізм планування розподілений між системою і прикладними програмами. Прикладна програма, отримавши управління від ОС, сама визначає момент завершення чергового циклу свого виконання і передає управління ОС за допомогою якого-небудь системного виклику. ОС формує черги процесів і вибирає відповідно до деякого алгоритму (наприклад, з урахуванням пріоритетів) наступний процес на виконання. Такий механізм створює проблеми як для користувачів, так і для розробників.

Для користувачів це означає, що управління системою втрачається на довільний період часу, який визначається додатком (а не користувачем). Якщо додаток витрачає надто багато часу на виконання деякої роботи, наприклад, на форматування диска, користувач не може перемкнутися з цього завдання на інше, тоді як форматування тривало б у фоновому режимі.

Тому розробники додатків для операційного середовища з невитісняючою багатозадачністю, повинні створювати додатки таким чином, щоб вони виконували свої завдання невеликими частинами. Наприклад, програма форматування може відформатувати один сектор диску і повернути управління системі. Після виконання інших завдань система поверне управління програмі форматування, щоб та відформатувала наступний сектор. Подібний метод розподілу часу між завданнями працює, але він істотно ускладнює розробку програм і пред’являє підвищені вимоги до кваліфікації програміста.

Проте розподіл функцій планувальника між системою і додатками не завжди є недоліком, а за певних умов може бути і перевагою, тому що дає можливість розробнику додатків самому проектувати алгоритм планування, що найбільш відповідав би для цього фіксованого набору завдань. Оскільки розробник сам визначає в програмі момент часу віддачі управління, то при цьому виключаються нераціональні переривання програм в "незручні" для них моменти часу. Істотною перевагою невитісняючої багатозадачності є вища швидкість перемикання із завдання на завдання.

Прикладом ефективного використання невитісняючої багатозадачності є файл-сервер NetWare, в якому, значною мірою завдяки цьому, досягнута висока швидкість виконання файлових операцій. Менш вдалим виявилося використання невитісняючої багатозадачності в операційному середовищі Windows 3.х.

Проте майже в усіх сучасних ОС, орієнтованих на високопродуктивне виконання додатків (UNIX, Windows NT, OS/2, VAX/VMS), реалізована витісняюча багатозадачність.

6. Мультипрограмування на основі переривань

Переривання є основною рушійною силою будь-якої ОС. Періодичні переривання від таймера викликають зміну процесів в мультипрограмній ОС, а переривання від пристроїв вводу-виводу управляють потоками даних, якими обчислювальна система обмінюється із зовнішнім світом.

Система переривань переводить процесор на виконання потоку команд, відмінного від того, який виконувався досі, з наступним поверненням до початкового коду. Переривання виникає або залежно від зовнішніх по відношенню до процесу подій, або при появі непередбачених аварійних ситуацій в процесі виконання цієї програми.

Залежно від джерела переривання діляться на три великі класи:

* зовнішні;
* внутрішні;
* програмні.

**Зовнішні** переривання можуть виникати в результаті дій користувача або ж в результаті надходження сигналів від апаратних пристроїв – сигналів завершення операцій вводу-виводу, що виробляються контроллерами зовнішніх пристроїв комп’ютера, або ж сигналів від давачів, керованих комп’ютером технічних об’єктів. Зовнішні переривання називають також **апаратними**, вказуючи на те, що переривання виникає внаслідок подачі деякою апаратурою електричного сигналу, який передається на спеціальний вхід переривання процесора. Цей клас переривань є **асинхронним** по відношенню до потоку інструкцій перериваючої програми. Апаратура процесора працює так, що асинхронні переривання виникають між виконанням двох сусідніх інструкцій, при цьому система після обробки переривання продовжує виконання процесу, вже починаючи з наступної інструкції.

**Внутрішні** переривання, або ж **виключення (exeption)**, відбуваються **синхронно** виконанню програми при появі аварійної ситуації в ході виконання деякої інструкції програми. Прикладами виключень є ділення на нуль, помилки захисту пам’яті, звернення за неіснуючою адресою, спроба виконати привілейовану інструкцію в режимі користувача і тому подібне. Виключення виникають безпосередньо в ході виконання тактів команди.

**Програмні** переривання відрізняються від попередніх двох класів тим, що вони за своєю суттю не є "істинними" перериваннями. При виконанні команди програмного переривання процесор відпрацьовує ту ж послідовність дій, що і при виникненні зовнішнього або внутрішнього переривання, але тільки відбувається це в передбачуваній точці програми - там, де програміст помістив цю команду. Практично усі сучасні процесори мають в системі команд інструкції програмних переривань. Однією з причин появи інструкцій програмних переривань в системі команд процесорів є те, що їх використання часто призводить до компактнішого коду програм в порівнянні з використанням стандартних команд виконання процедур. Програмні переривання часто використовуються для виконання обмеженої кількості викликів функцій ядра операційної системи, тобто системних викликів.

**Системний виклик** дозволяє додатку звернутися до ОС з проханням виконати певну дію, оформлену як процедура (чи набір процедур) кодового сегменту ОС. Для прикладного програміста ОС виглядає як бібліотека, що надає деякий набір корисних функцій, за допомогою яких можна спростити прикладну програму або виконати дії, заборонені в режимі користувача, наприклад обмін даними з пристроєм вводу-виводу.

ОС може виконувати системні виклики в синхронному або асинхронному режимах. **Синхронний** **системний виклик** означає, що процес, який зробив такий виклик, припиняється (переводиться планувальником ОС в стан очікування) до тих пір, поки системний виклик не виконає усю роботу. Після цього планувальник переводить процес в стан готовності і при черговому виконанні процес гарантовано може скористатися результатами системного виклику, що завершився до цього часу. Синхронні виклики називаються також **блокуючими**, оскільки процес, що викликав системну дію, блокується до його завершення.

**Асинхронний системний виклик** не призводить до переведення процесу в режим очікування після виконання деяких початкових системних дій, наприклад запуску операції вводу-виводу, управління повертається прикладному процесу.

Перериванням приписується пріоритет, за допомогою якого вони ранжируються по мірі важливості і терміновості.

Переривання зазвичай обробляються модулями ОС, оскільки дії, що виконуються по перериванню, відносяться до управління поділяючими ресурсами обчислювальної системи (прінтером, диском, таймером, процесором і тому подібне). Процедури, що викликаються по перериваннях, називають **обробниками переривань**, або **процедурами обслуговування переривань** (Interrupt Service Routine, ISR). Апаратні переривання обробляються драйверами відповідних зовнішніх пристроїв, виключення - спеціальними модулями ядра, а програмні переривання - процедурами ОС, що обслуговують системні виклики. Окрім цих модулів в ОС може знаходитися диспетчер переривань, який координує роботу окремих обробників переривань.

Механізм переривань підтримується апаратними засобами комп’ютера і програмними засобами ОС. Апаратна підтримка переривань має свої особливості, яка залежать від типу процесора і інших апаратних компонентів, що передають сигнал запиту переривання від зовнішнього пристрою до процесора (контролер переривань, що є посередником між сигналами шини і сигналами процесора). Особливості апаратної peaлізації переривань чинять вплив на засоби програмної підтримки переривань, що працюють у складі ОС.

Існують два основні способи, за допомогою яких шини виконують переривання: **векторний (vectored)** і **опитуваний (polled)**. У обох способах процесору надається інформація про рівень пріоритету переривання на шині підключення зовнішніх пристроїв. У разі векторних переривань в процесор передається також інформація про початкову адресу програми обробки виниклого переривання – обробника переривань.

Пристроям, які використовують векторні переривання, призначається вектор переривань. Він є електричним сигналом, який виставляється на відповідні шини процесора і несе в собі інформацію про визначений, закріплений за цим пристроєм номер, що ідентифікує відповідний обробник переривань. Цей вектор може бути фіксованим, конфігурованим (наприклад, з використанням перемикачів) або програмованим. Операційна система може передбачати процедуру реєстрації вектору обробки переривань для певного пристрою, яка зв’язує деяку підпрограму обробки переривань з певним вектором. При отриманні сигналу запиту переривання процесор виконує спеціальний цикл підтвердження переривання, в якому пристрій повинен ідентифікувати себе. Протягом цього циклу пристрій відповідає, виставляючи на шину вектор переривань. Потім процесор використовує цей вектор для знаходження обробника цього переривання. Прикладом шини підключення зовнішніх пристроїв, яка підтримує векторні переривання, являється шина VMEbus.

При використанні опитуваних переривань процесор отримує від пристрою, що запитав переривання, тільки інформацію про рівень пріоритету переривання. З кожним рівнем переривань може бути пов’язані декілька пристроїв і відповідно декілька програм-обробників переривань. При виникненні переривання процесор повинен визначити, який пристрій з тих, які пов’язані з цим рівнем переривань, дійсно запитав переривання. Це досягається викликом усіх обробників переривань для цього рівня пріоритету, поки один з обробників не підтвердить, що переривання прийшло від обслуговуваного ним пристрою. Якщо ж з кожним рівнем переривань пов’язаний тільки один пристрій, то визначення потрібної програми обробки переривання відбувається негайно, як і при векторному перериванні. Опитувані переривання підтримують шини ISA, EISA, MCA, PCI і Sbus.

Узагальнено послідовність дій апаратних і програмних засобів по обробці переривання можна описати таким чином.

1. При виникненні сигналу (для апаратних переривань) або умови (для внутрішніх переривань) переривання відбувається первинне апаратне розпізнавання типу переривання. Якщо переривання цього типу в даний момент заборонені, то процесор продовжує підтримувати природний хід виконання команд. В іншому випадку, залежно від інформації, що поступила в процесор, відбувається автоматичний виклик процедури обробки переривання, адреса якої знаходиться в спеціальній таблиці ОС, що розміщується або в регістрах процесора, або у визначеному місці оперативній пам’яті.

2. Автоматично зберігається деяка частина контексту перерваного процесу, яка дозволить ядру відновити виконання процесу після обробки переривання. У цю підмножину зазвичай включаються значення лічильника команд, слова стану машини, що зберігає ознаки основних режимів роботи процесора, а також декількох регістрів загального призначення, які необхідні програмі обробки переривання.

3. Рішення про перепланування процесів може бути прийняте в ході обробки переривання, наприклад, якщо це переривання від таймера і після збільшення значення системного годинника з’ясовується, що процес вичерпав виділений йому квант часу. Проте це зовсім не обов’язково - переривання може виконуватися і без зміни процесу, наприклад прийом чергової порції даних від контролера зовнішнього пристрою найчастіше відбувається у рамках поточного процесу, хоча дані можуть бути призначені іншому процесу.

3. Одночасно із завантаженням адреси процедури обробки переривань в лічильник команд може автоматично виконуватися завантаження нового значення слова стану машини, який визначає режими роботи процесора при обробці переривання, у тому числі роботу в привілейованому режимі. У деяких моделях процесорів перехід в привілейований режим за рахунок зміни стану машини при обробці переривання є єдиним способом зміни режиму. Переривання практично в усіх мультипрограмних ОС обробляються в привілейованому режимі модулями ядра, оскільки при цьому зазвичай треба виконати ряд критичних операцій, від яких залежить життєздатність системи - управляти зовнішніми пристроями, перепланувати потоки і тому подібне.

4. Тимчасово забороняються переривання цього типу, щоб не утворилася черга вкладених один в одного потоків однієї і тієї ж процедури. Багато процесорів автоматично встановлюють ознаку заборони переривань на початку циклу обробки переривання, в іншому випадку це робить програма обробки переривань.

5. Після того, як переривання оброблене ядром операційної системи, перерваний контекст відновлюється і робота потоку поновлюється з перерваного місця. Частина контексту відновлюється апаратно по команді повернення з переривань (наприклад, адреса наступної команди і слово стану машини), а частина - програмним способом, за допомогою явних команд витягування даних із стека. При поверненні з переривання блокування повторних переривань цього типу знімається.

Лекція 5

Засоби синхронізації і взаємодії процесів

1. Необхідність синхронізації і перегони

Існує досить великий клас засобів ОС, за допомогою яких забезпечується взаємна синхронізація процесів. Потреба в синхронізації процесів виникає тільки в мультипрограмній операційній системі і пов’язана із спільним використанням апаратних і інформаційних ресурсів обчислювальної системи. Синхронізація потрібна для виключення перегонів і тупиків при обміні даними між процесами, розподілі даних, при доступі до процесора і пристроїв вводу-виводу.

У багатьох операційних системах ці засоби називаються засобами міжпроцесної взаємодії - Inter Process Communications (IPC). Зазвичай до засобів IPC відносять не лише засоби міжпроцесної синхронізації, але і засоби міжпроцесного обміну даними.

Будь-яка взаємодія процесів пов’язана з їх синхронізацією, яка полягає в узгодженні їх швидкостей шляхом призупинення процесу до настання деякої події і наступної його активізації при настанні цієї події. Синхронізація лежить в основі будь-якої взаємодії процесів, чи пов’язана ця взаємодія з розподілом ресурсів або з обміном даними.

Щомиті в системі відбуваються сотні подій, пов’язаних з розподілом і звільненням ресурсів, і ОС повинна мати надійні і продуктивні засоби, які б дозволяли їй синхронізувати процеси з подіями, що відбуваються в системі.

Для синхронізації процесів прикладних програм програміст може використовувати як власні засоби і прийоми синхронізації, так і засоби ОС. У багатьох випадках ефективнішими або навіть єдино можливими є засоби синхронізації, що надаються операційною системою у формі системних викликів. Різні процеси не мають можливості втручатися певним чином в роботу один одного. Без посередництва операційної системи вони не можуть припинити один одного або оповістити про подію, що сталася. Засоби синхронізації використовуються ОС не лише для синхронізації прикладних процесів, але і для її внутрішніх потреб. Зазвичай розробники ОС надають в розпорядження прикладних і системних програмістів широкий спектр засобів синхронізації.

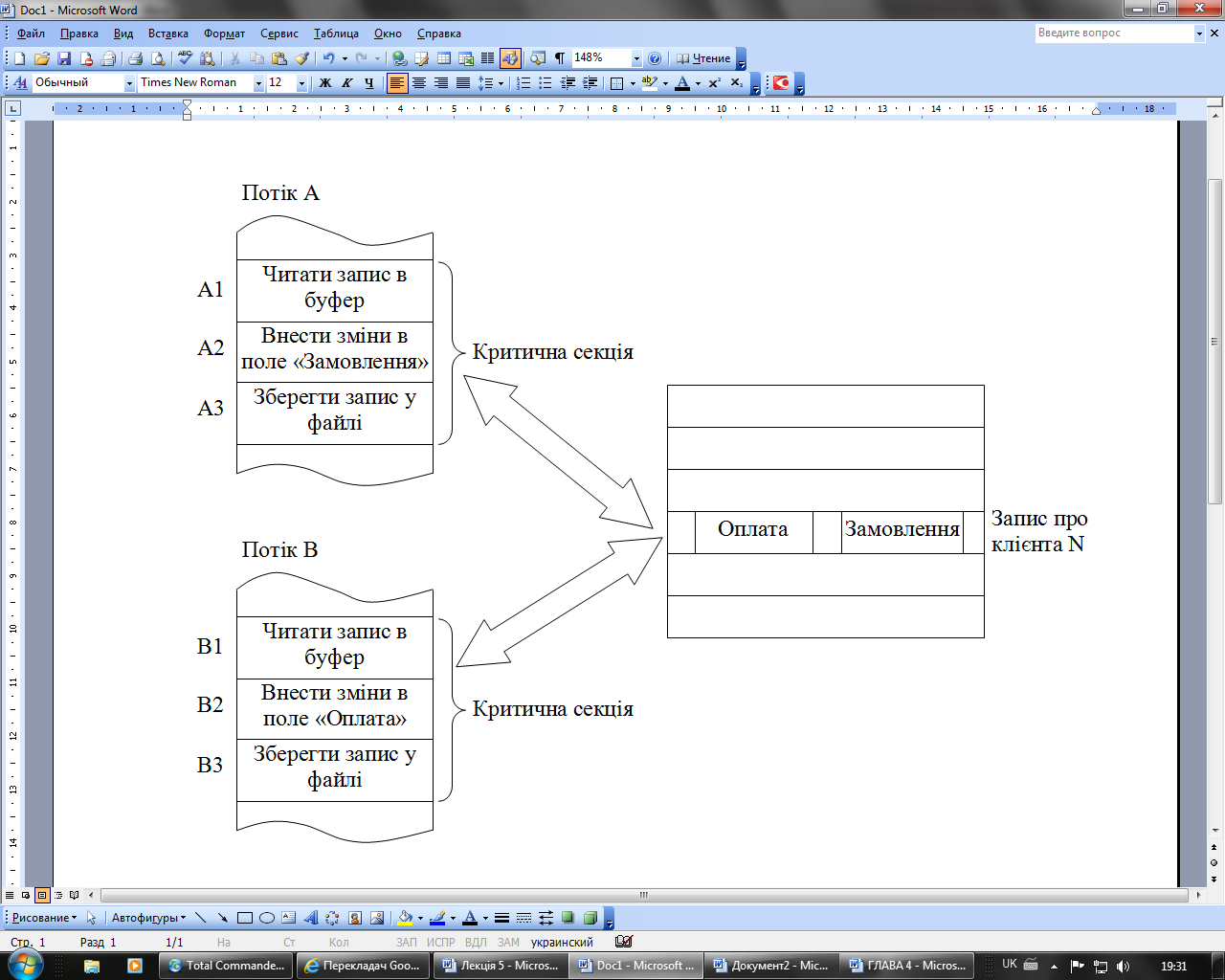
Розглянемо, наприклад, задачу ведення бази даних клієнтів підприємства (рис. 1). Кожному клієнтові відводиться окремий запис в базі даних, в якій серед інших полів є поля Замовлення і Оплата. Програма, що веде базу даних, оформлена як єдиний процес, що має декілька потоків, у тому числі потік А, який заносить в базу даних інформацію про замовлення, що поступили від клієнтів, і потік В, який фіксує в базі даних відомості про оплату клієнтами виставлених рахунків. Обидва ці потоки спільно працюють над загальним файлом бази даних, використовуючи однотипні алгоритми, які включають три кроки.

Рис. 1. Виникнення перегонів при доступі до поділюваних даних

1. Прочитати з файлу бази даних в буфер запис про клієнта із заданим ідентифікатором.

2. Внести нове значення в поле Замовлення (для потоку А) або Оплата (для потоку В).

3. Повернути модифікований запис у файл бази даних.

Позначимо відповідні кроки для потоку А як Al, A2 і A3, а для потоку В як Bl, B2 і ВЗ. Припустимо, що в деякий момент потік А оновлює поле Замовлення запису про клієнта N. Для цього він зчитує цей запис у свій буфер (крок А1), модифікує значення поля Замовлення (крок А2), але внести запис до бази даних (крок A3) не встигає, оскільки його виконання переривається, наприклад, внаслідок завершення кванту часу.

Припустимо також, що потоку В також потрібно було внести відомості про оплату відносно того ж клієнта N. Коли підходить черга потоку В, він встигає зчитати запис у свій буфер (крок В1) і виконати оновлення поля Оплата (крок В2), а потім переривається. У буфері потоку В знаходиться запис про клієнта N, в якому поле Замовлення має колишнє, не змінене значення.

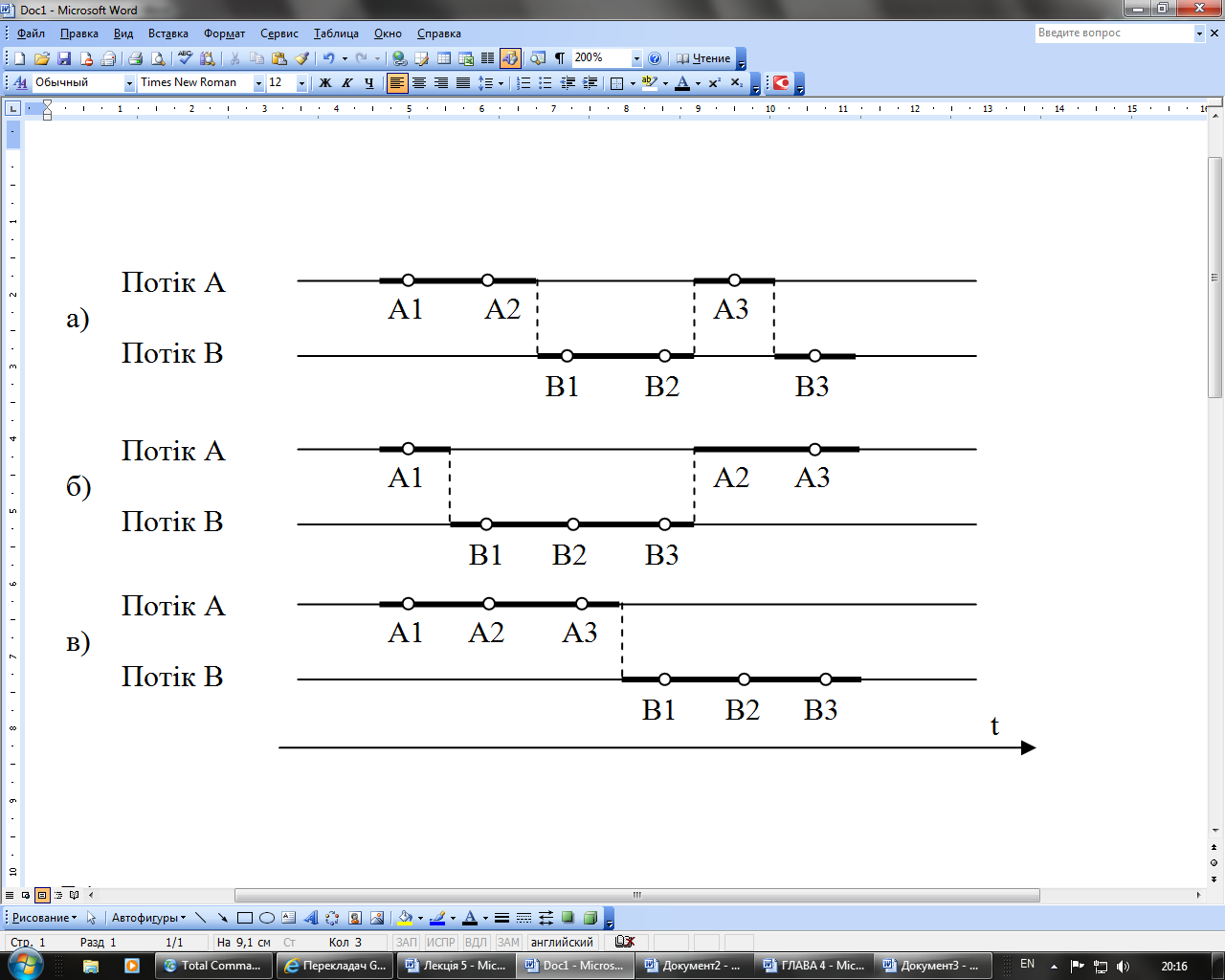
Коли у черговий раз управління буде передано потоку А, то він, продовжуючи свою роботу, запише запис про клієнта N з модифікованим полем Замовлення в базу даних (крок A3). Після переривання потоку А і активізації потоку В останній запише в базу даних поверх тільки що оновленого запису про клієнта N свій варіант запису, в якому оновлено значення поля Оплата. Таким чином, в базі даних будуть зафіксовані відомості про те, що клієнт N здійснив оплату, але інформація про його замовлення виявиться втраченою (рис. 2, а).

Рис. 2. Вплив відносних швидкостей потоків на результат рішення задачі

Складність проблеми синхронізації криється в нерегулярності виникаючих ситуацій. Так, в попередньому прикладі можна представити і інший розвиток подій: могла бути втрачена інформація не про замовлення, а про оплату (рис. 2, б) або, навпаки, усі виправлення були успішно внесені (рис. 2, в). Усе визначається взаємними швидкостями потоків і моментами їх переривання. Тому відладка взаємодіючих потоків є складним завданням. Ситуації, подібні до тієї, коли два або більше потоків обробляють поділювані дані і кінцевий результат залежить від співвідношення швидкостей потоків, називаються **перегонами**.

2. Критична секція

**Критична секція** – це частина програми, в якій здійснюється доступ до поділюваних даних. Щоб виключити ефект перегонів по відношенню до деякого ресурсу, необхідно забезпечити, щоб в кожен момент часу в критичній секції, пов’язаній з цим ресурсом, знаходився максимум один процес (потік). Цей прийом називають **взаємним виключенням**.

Простий спосіб забезпечити взаємне виключення - дозволити процесу, що знаходиться в критичній секції, забороняти усі переривання. Проте цей спосіб непридатний, оскільки небезпечно довіряти управління системою процесу користувача – він може надовго зайняти процесор, а при краху процесу в критичній області крах потерпить уся система, тому що переривання ніколи не будуть дозволені.

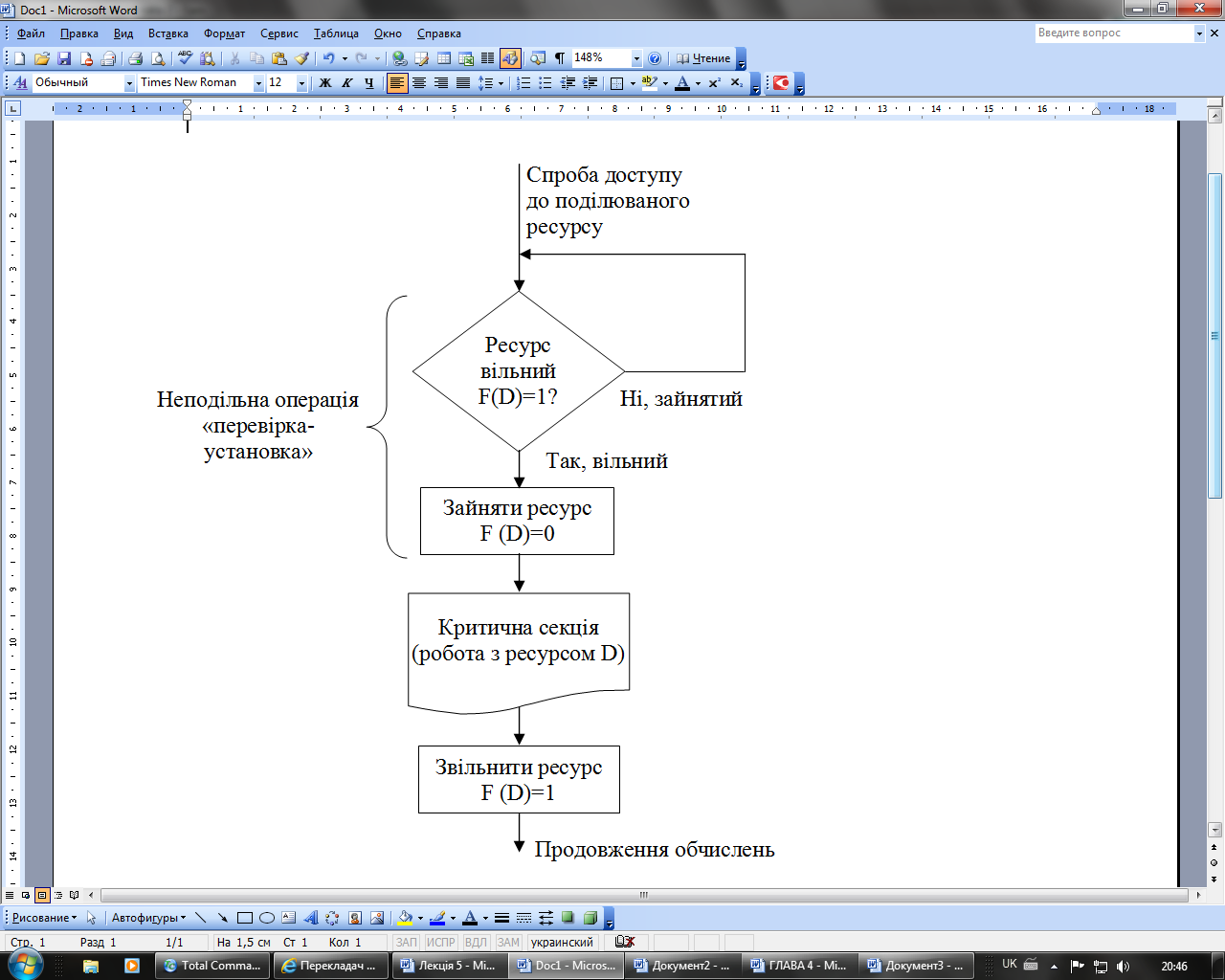
Іншим способом являється використання **блокуючих змінних** (рис. 3).

Рис. 3. Реалізація критичних секцій з використанням блокуючих змінних

З кожним поділюваним ресурсом зв’язується двійкова змінна, яка набуває значення 1, якщо ресурс вільний (тобто жоден процес не знаходиться в даний момент в критичній секції, пов’язаній з цим процесом), і значення 0, якщо ресурс зайнятий. На рис. 3 показаний фрагмент алгоритму процесу, що використовує для реалізації взаємного виключення доступу до поділюваного ресурсу D блокуючу змінну F(D). Перед входом в критичну секцію процес перевіряє, чи вільний ресурс D. Якщо він зайнятий, то перевірка циклічно повторюється, якщо вільний, то значення змінної F(D) встановлюється в 0 і процес входить в критичну секцію. Після того, як процес виконає усі дії з поділюваним ресурсом D, значення змінної F(D) знову встановлюється рівним 1.

Якщо усі процеси написані з використанням вищеописаних угод, то взаємне виключення гарантується. Слід зауважити, що операція перевірки і установки блокуючої змінної має бути неподільною. Пояснимо це. Нехай в результаті перевірки змінної процес визначив, що ресурс вільний, але відразу після цього, не встигнувши встановити змінну в 0, був перерваний. За час його призупинення інший процес зайняв ресурс, увійшов до своєї критичної секції, але також був перерваний, не завершивши роботи з поділюваним ресурсом. Коли управління було повернено першому процесу, він, вважаючи ресурс вільним, встановив ознаку зайнятості і почав виконувати свою критичну секцію. Таким чином був порушений принцип взаємного виключення, що потенційно може привести до небажаних наслідків. Щоб уникнути таких ситуацій в системі команд машини бажано мати єдину команду "перевірка-установка", або ж реалізовувати системними засобами відповідні програмні примітиви, які б забороняли переривання упродовж усієї операції перевірки і установки.

Реалізація критичних секцій з використанням блокуючих змінних має істотний недолік: протягом часу, коли один процес знаходиться в критичній секції, інший процес, якому потрібний цей же ресурс, виконуватиме рутинні дії з опитування блокуючої змінної, марно витрачаючи процесорний час. Для усунення таких ситуацій може бути використаний так званий **апарат подій**. За допомогою цього засобу можуть вирішуватися не лише проблеми взаємного виключення, але і загальніші завдання синхронізації процесів. У різних ОС апарат подій реалізується по своєму, але у будь-якому випадку використовуються системні функції аналогічного призначення, які умовно назвемо WAIT(x) і POST(x), де x – ідентифікатор деякої події. На рис. 4 показаний фрагмент алгоритму процесу, що використовує ці функції.

Якщо ресурс зайнятий, то процес не виконує циклічне опитування, як в попередньому випадку, а викликає системну функцію WAIT(D), тут D означає подію, що полягає в звільненні ресурсу D. Функція WAIT(D) переводить активний процес в стан ОЧІКУВАННЯ і робить відмітку в його дескрипторі про те, що процес очікує подію D. Процес, який в цей час використовує ресурс D, після виходу з критичної секції повинен виконати системну функцію POST(D), внаслідок чого ОС переглядає чергу процесів, що очікують подію D і переводить перший процес із черги в стан ГОТОВНІСТЬ.

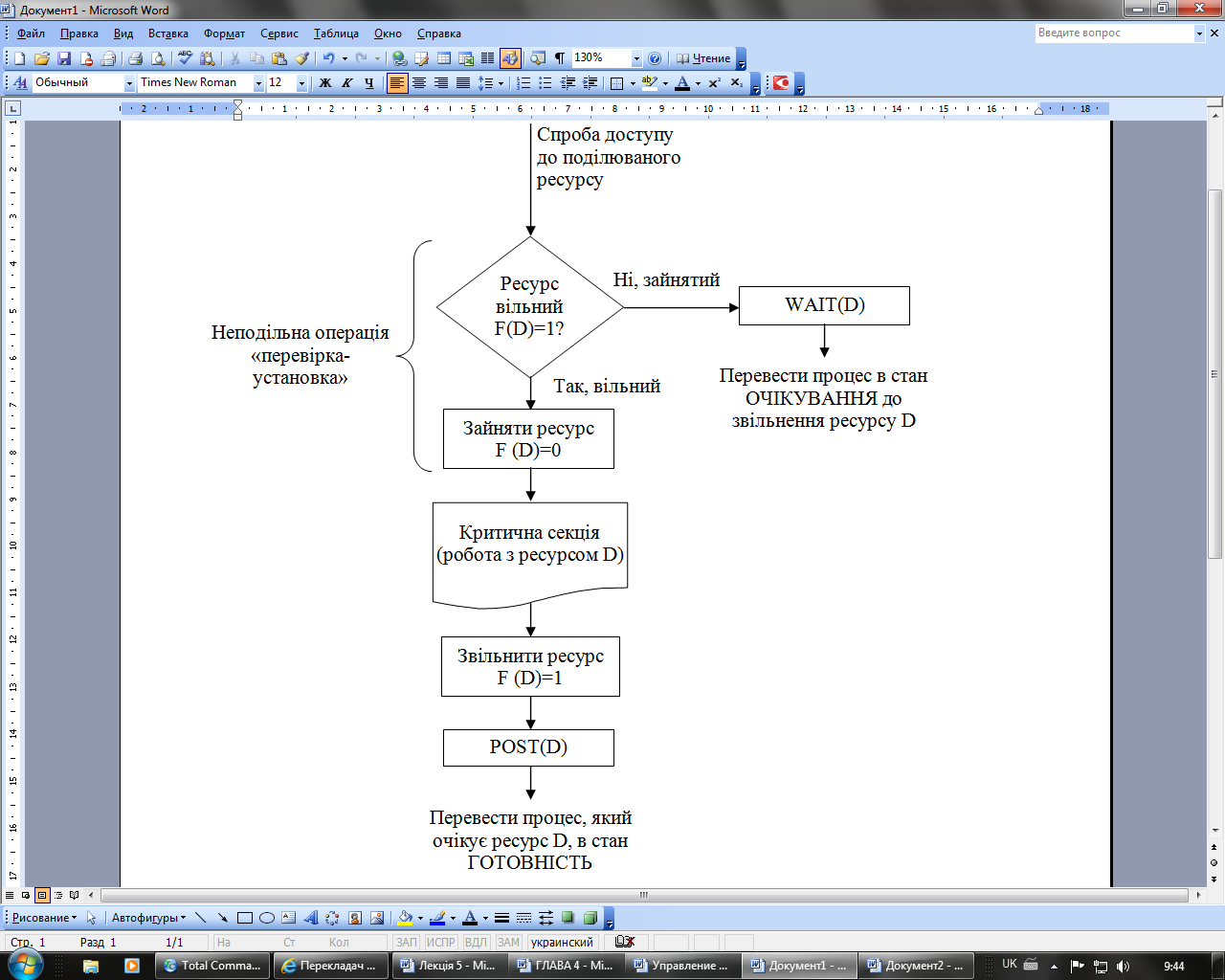


Рис. 4. Реалізація критичної секції з використанням системних

функцій WAIT(D) і POST(D)

3. Семафори

Узагальненням блокуючих змінних є **семафори Дійкстри**. Замість двійкових змінних Дійкстра запропонував використовувати змінні, які можуть набувати цілих не негативних значень. Для роботи з семафорами вводяться два примітиви, які позначають Р і V. Нехай змінна S є семафором. Тоді дії V(S) і P(S) визначаються наступним чином.

* V(S): змінна S збільшується на 1 єдиною дією. Вибірка, нарощування і запам’ятовування не можуть бути перервані. До змінної S немає доступу іншим процесам під час виконання цієї операції.
* P(S): зменшення S на 1, якщо це можливо. Якщо S=0 і неможливо зменшити S, залишаючись в області цілих не негативних значень, то в цьому випадку процес, що викликає операцію Р, очікує, поки це зменшення стане можливим. Успішна перевірка і зменшення також є неділимою операцією.

Ніякі переривання під час виконання примітивів V і Р недопустимі.

У окремому випадку, коли семафор S може набувати тільки значень 0 і 1, він перетворюється на блокуючу змінну, яку з цієї причини часто називають **двійковим семафором**. Операція Р містить в собі потенційну можливість переходу процесу, який її виконує, в стан очікування, тоді як операція V може при деяких обставинах активізувати інший процес, припинений операцією Р. Використання двійкової змінної не дозволяє організувати доступ до критичного ресурсу більш ніж одному потоку. Семафор же вирішує задачу синхронізації гнучкіше, допускаючи до пулу поділюваних ресурсів задану кількість процесів.

Розглянемо використання семафорів на класичному прикладі взаємодії двох процесів, що виконуються в режимі мультипрограмування, один з яких пише дані в буферний пул, а інший зчитує їх з буферного пулу. Нехай буферний пул складається з N буферів, кожен з яких може містити один запис. У загальному випадку процес-записувач і процес-читач можуть мати різні швидкості і звертатися до буферного пулу із зміною інтенсивністю. У один період швидкість запису може перевищувати швидкість читання, в іншій - навпаки. Для правильної спільної роботи процес-записувач повинен припинятися, коли усі буфери виявляються зайнятими, і активізуватися при звільненні хоча б одного буфера. Навпаки, процес-читач повинен зупинятися, коли усі буфери порожні, і активізуватися при появі хоча б одного запису.

Введемо два семафори: е – число порожніх буферів, і f – число заповнених буферів, причому в початковому стані е =N, a f =0. Тоді робота процесів із загальним буферним пулом може бути описана наступним чином (рис. 5).

Процес-записувач передусім виконує операцію Р(е), за допомогою якої він перевіряє, чи є в буферному пулі незаповнені буфери. Відповідно до семантики операції Р, якщо семафор е дорівнює 0 (тобто вільних буферів в даний момент немає), то процес-записувач переходить в стан очікування. Якщо ж значенням е є позитивне число, то він зменшує число вільних буферів, записує дані в черговий вільний буфер і після цього нарощує число зайнятих буферів операцією V(f). Процес-читач діє аналогічним чином, з тією різницею, що він починає роботу з перевірки наявності заповнених буферів, а після читання даних нарощує кількість вільних буферів.

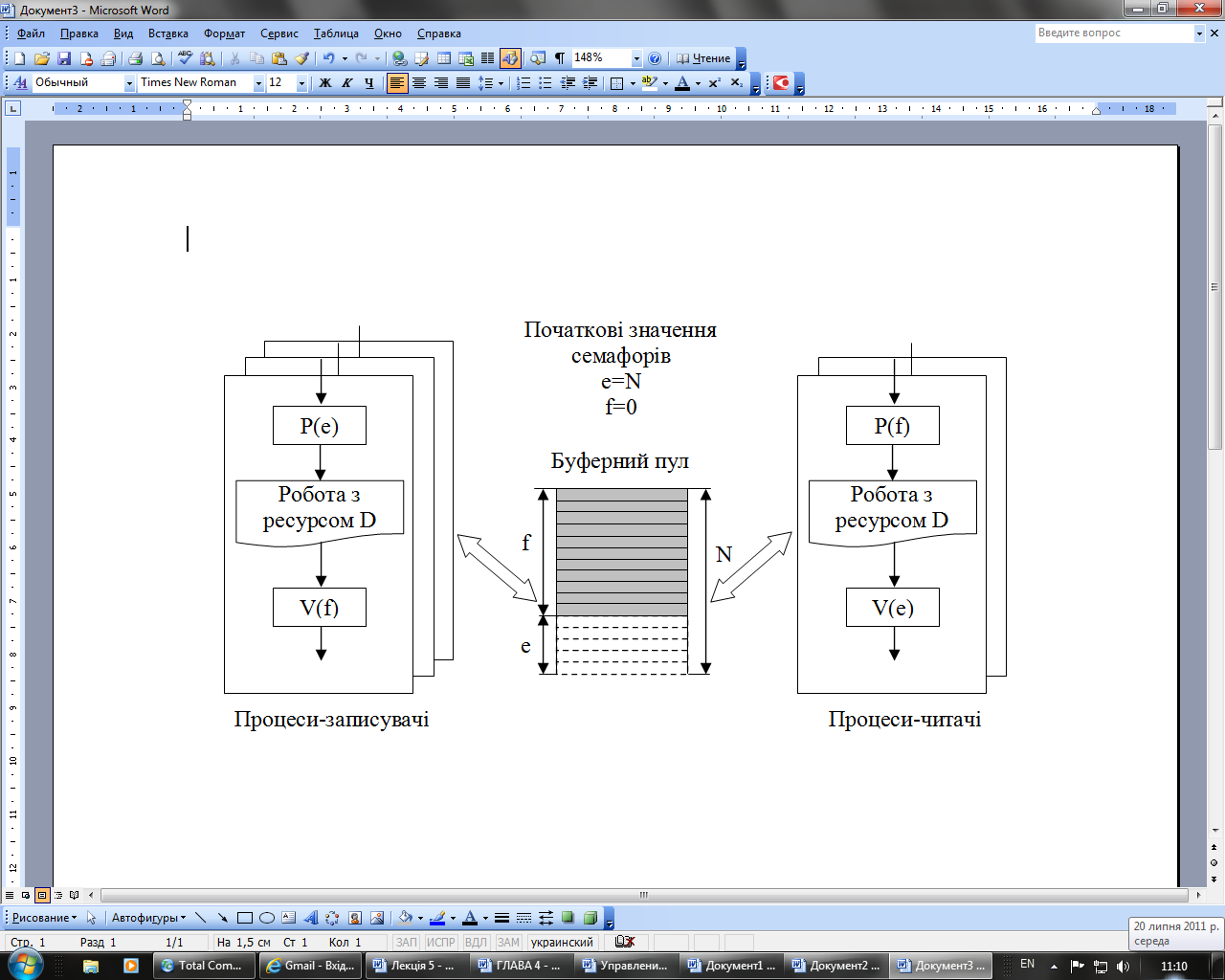


Рис. 5. Використання семафорів для синхронізації процесів

Таким чином, семафори дозволяють ефективно вирішувати задачу синхронізації доступу до ресурсних пулів, таких, наприклад, як набір ідентичних у функціональному призначенні зовнішніх пристроїв (модемів, принтерів, портів), або набір областей пам’яті однакової величини, або інформаційних структур. В усіх цих і подібних ним випадках за допомогою семафорів можна організувати доступ до поділюваних ресурсів відразу декільком процесам.

Семафор може використовуватися і як блокуюча змінна. У розглянутому вище прикладі, для того, щоб виключити колізії при роботі з областю поділюваної пам’яті, вважатимемо, що запис в буфер і зчитування з буфера є критичними секціями. Взаємне виключення забезпечуватиметься за допомогою двійкового семафора b (рис. 6). Обидва потоки після перевірки доступності буферів повинні виконати перевірку доступності критичної секції.

4. Взаємні блокування

Наведений вище приклад допоможе проілюструвати ще одну проблему синхронізації – **взаємні блокування**, звані також **дедлоками (deadlocks)**, **клінчами (clinch)** або **тупиками**. Якщо переставити місцями операції P(e) і P(b) в процесі-записувачі, то при деякому збігу обставин ці два процеси можуть взаємно заблокувати один одного. Дійсно, нехай процес-записувач першим увійде до критичної секції і виявить відсутність вільних буферів; він почне очікувати, коли процес-читач візьме черговий запис з буфера, але процес-читач не зможе цього зробити, оскільки для цього необхідно увійти до критичної секції, вхід в яку заблокований процесом-записувачем.

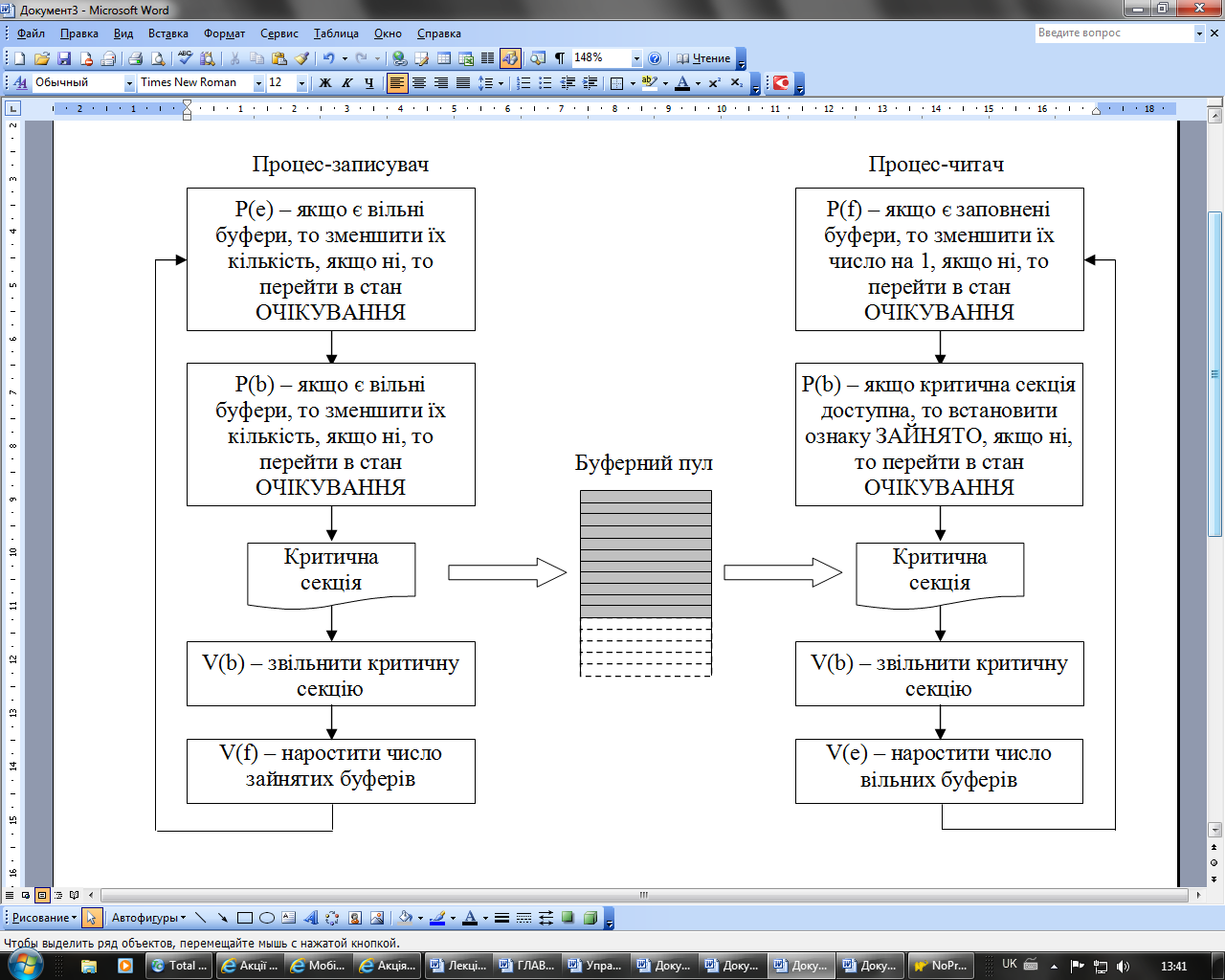


Рис. 6. Використання двійкового семафора

Розглянемо ще один приклад тупика. Нехай двом процесам, що виконуються в режимі мультипрограмування, для виконання їх роботи потрібно два ресурси, наприклад, принтер і диск. На рис. 7, а показані фрагменти відповідних програм. І нехай після того, як процес А зайняв принтер (встановив блокуючу змінну), він був перерваний. Управління отримав процес В, який спочатку зайняв диск, але при виконанні наступної команди був заблокований, оскільки принтер виявився вже зайнятим процесом А. Управління знову отримав процес А, який відповідно до своєї програми зробив спробу зайняти диск і був заблокований: диск вже виділений процесу В. В такому положенні процеси А і В можуть знаходитися скільки завгодно довго.

Залежно від співвідношення швидкостей процесів, вони можуть або абсолютно незалежно використовувати поділювані ресурси (г), або утворювати черги до поділюваних ресурсів (в), або взаємно блокувати один одного (б). Тупикові ситуації потрібно відрізняти від простих черг, хоча і ті і інші виникають при спільному використанні ресурсів і зовні виглядають схоже: процес припиняється і чекає звільнення ресурсу. Проте черга - це нормальне явище, невід’ємна ознака високого коефіцієнта використання ресурсів при випадковому поступленні запитів. Вона виникає тоді, коли ресурс недоступний в даний момент, але через деякий час він звільняється, і процес продовжує своє виконання. Тупик же, що видно з його назви, є в деякому роді нерозв’язною ситуацією.

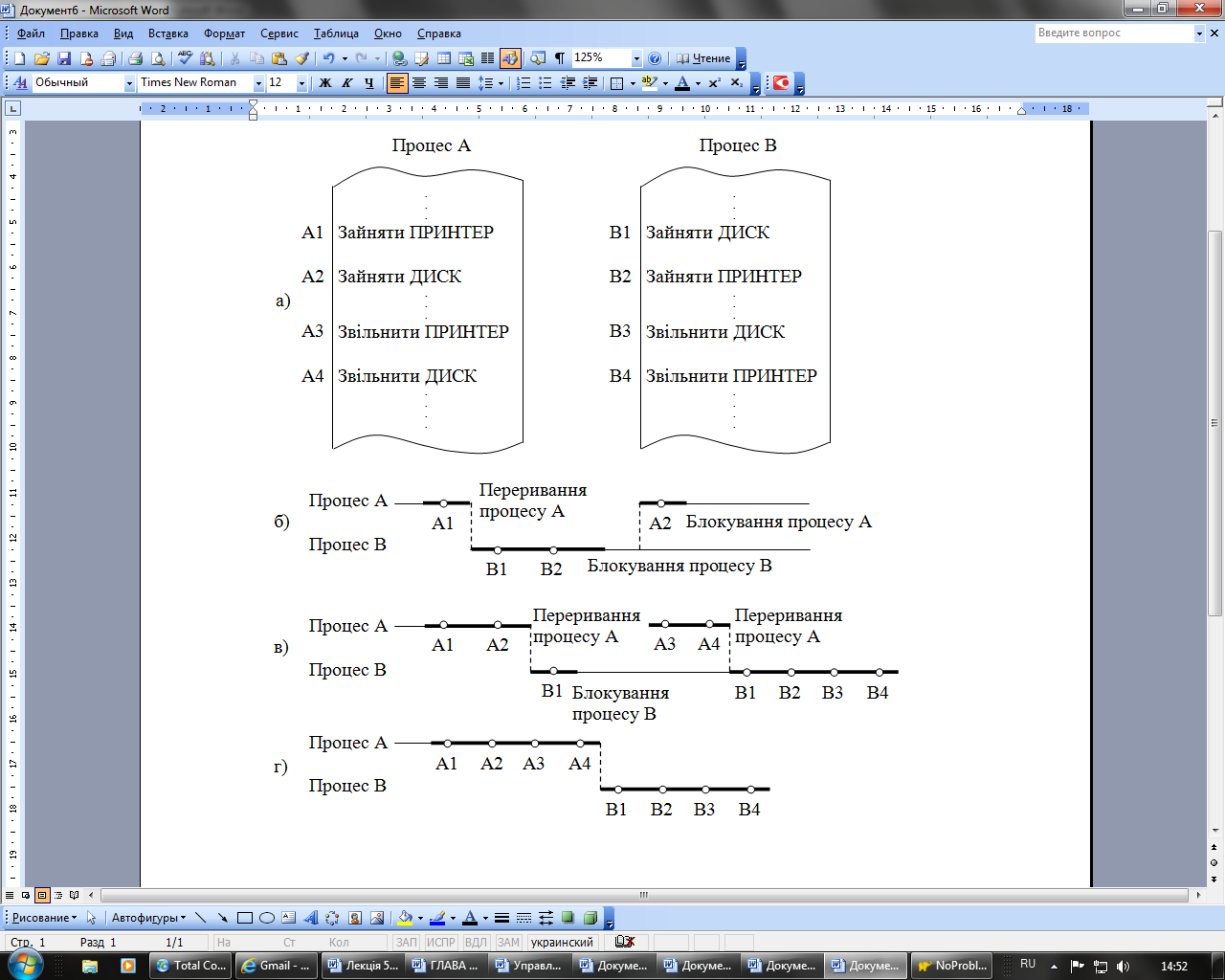


Рис. 7. (a) фрагменти програм А і В, що розділяють принтер і диск; (б) взаємне блокування (клінч); (в) черга до поділюваних ресурсів; (г) незалежне використання ресурсів

У розглянутих прикладах тупик був утворений двома процесами, але взаємно блокувати один одного можуть і більше число процесів.

Проблема тупиків включає наступні завдання:

* запобігання тупикам,
* розпізнавання тупиків,
* відновлення системи після тупиків.

Тупики можуть бути відвернені на стадії написання програм. Так, якби в попередньому прикладі процес А і процес В запитували ресурси в однаковій послідовності, то тупик був би неможливим. Другий підхід до запобігання тупикам називається динамічним і полягає у використанні певних правил при призначенні ресурсів процесам, наприклад, ресурси можуть виділятися в певній послідовності, загальній для усіх процесів.

В деяких випадках, коли тупикова ситуація утворена багатьма процесами, що використовують багато ресурсів, розпізнавання тупика є нетривіальним завданням. Існують формальні, програмно-реалізовані методи розпізнавання тупиків, що базуються на веденні таблиць розподілів ресурсів і таблиць запитів до зайнятих ресурсів. Аналіз цих таблиць дозволяє виявити взаємні блокування.

Якщо ж тупикова ситуація виникла, то не обов’язково знімати з виконання усі заблоковані процеси. Можна зняти тільки частину з них, при цьому звільняються ресурси, які очікують інші процеси, можна повернути деякі процеси в область свопінгу, можна зробити «відкат» деяких процесів до так званої контрольної точки, в якій запам’ятовується уся інформація, необхідна для відновлення виконання програми з цього місця. Контрольні точки розставляються в програмі в місцях, після яких можливе виникнення тупиків.

З усього вищесказаного ясно, що використовувати семафори треба дуже обережно, оскільки одна незначна помилка може привести до останову системи. Для того, щоб полегшити написання коректних програм, було запропоновано високорівневий засіб синхронізації, званий монітором. Монітор - це набір процедур, змінних і структур даних. Процеси можуть викликати процедури монітора, але не мають доступу до внутрішніх даних монітора. Монітори мають важливу властивість, яка робить їх корисними для досягнення взаємного виключення: тільки один процес може бути активним по відношенню до монітора. Компілятор обробляє виклики процедур монітора особливим чином. Зазвичай, коли процес викликає процедуру монітора, то перші декілька інструкцій цієї процедури перевіряють, чи не активний який-небудь інший процес по відношенню до цього монітора. Якщо так, то викликаючий процес призупиняється, поки інший процес не звільнить монітор. Таким чином, виключення входу декількох процесів в монітор реалізується не програмістом, а компілятором, що робить помилки менш ймовірними.

У розподілених системах, що складаються з декількох процесорів, кожен з яких має власну оперативну пам’ять, семафори і монітори виявляються непридатними. У таких системах синхронізація може бути реалізована тільки за допомогою обміну повідомленнями.

5. Нитки

Багатозадачність є важливою властивістю ОС. Для підтримки цієї властивості ОС визначає і оформляє для себе ті внутрішні одиниці роботи, між якими і розділятиметься процесор і інші ресурси комп’ютера. Ці внутрішні одиниці роботи в різних ОС носять різні назви - задача, завдання, процес, нитка.

ОС підтримує відособленість процесів: у кожного процесу є свій віртуальний адресний простір, кожному процесу призначаються свої ресурси - файли, вікна, семафори і так далі. Така відособленість потрібна для того, щоб захистити один процес від іншого, оскільки вони, спільно використовуючи усі ресурси машини, конкурують один з одним. У загальному випадку процеси належать різним користувачам, що розділяють один комп’ютер, і ОС бере на себе роль арбітра в спорах процесів за ресурси.

При мультипрограмуванні підвищується пропускна спроможність системи, але окремий процес ніколи не може бути виконаний швидше, ніж коли б він виконувався в однопрограмному режимі. Проте завдання, що вирішується у рамках одного процесу, може мати внутрішній паралелізм, який дозволяє прискорити її рішення. Наприклад, в ході виконання завдання відбувається звернення до зовнішнього пристрою, і на час цієї операції можна не блокувати повністю виконання процесу, а продовжити обчислення по іншій "гілці" процесу. Для цих цілей сучасні ОС пропонують використовувати механізм **багатониткової обробки (multithreading)**. При цьому вводиться нове поняття **«нитка» (thread)**.

Завдання, оформлене у вигляді декількох ниток у рамках одного процесу, може бути виконане швидше за рахунок псевдопаралельного (чи паралельного в мультипроцесорній системі) виконання її окремих частин.

Нитки, що відносяться до одного процесу, не настільки ізольовані один від одного, як процеси в традиційній багатозадачній системі, між ними легко організувати тісну взаємодію. На відміну від процесів, які належать різним, взагалі кажучи, конкуруючим додаткам, усі нитки одного процесу завжди належать одному додатку, тому програміст, що пише цей додаток, може заздалегідь продумати роботу безлічі ниток процесу так, щоб вони могли взаємодіяти, а не боротися за ресурси.

Нитки іноді називають полегшеними процесами або міні-процесами. Дійсно, нитки у багатьох відношеннях подібні до процесів. Кожна нитка виконується строго послідовно і має свій власний програмний лічильник і стек. Нитки, як і процеси, можуть, наприклад, породжувати нитки-нащадки, можуть переходити із стану в стан. Подібно до традиційних процесів (тобто процесів, що складаються з однієї нитки), нитки можуть знаходиться в одному з наступних станів: ВИКОНАННЯ, ОЧІКУВАННЯ і ГОТОВНІСТЬ. Поки одна нитка заблокована, інша нитка того ж процесу може виконуватися. Нитки розділяють процесор так, як це роблять процеси, відповідно до різних варіантів планування.

Лекція 6

Управління пам’яттю

1. Типи адрес

Пам’ять (мається на увазі оперативна пам’ять, memory) є найважливішим ресурсом системи, що вимагає ретельного управління з боку мультипрограмної ОС. Особлива роль пам’яті пояснюється тим, що процесор може виконувати інструкції програми тільки у тому випадку, якщо вони знаходяться в пам’яті. Пам’ять розподіляється як між модулями прикладних програм, так і між модулями самої операційної системи.

Функціями ОС по управлінню пам’яттю є: відстежування вільної і зайнятої пам’яті, виділення пам’яті процесам і звільнення пам’яті при завершенні процесів, витіснення процесів з оперативної пам’яті на диск, коли розміри основної пам’яті не достатні для розміщення в ній усіх процесів, і повернення їх в оперативну пам’ять, коли в ній звільняється місце, а також налаштування адрес програми на конкретну область фізичної пам’яті.

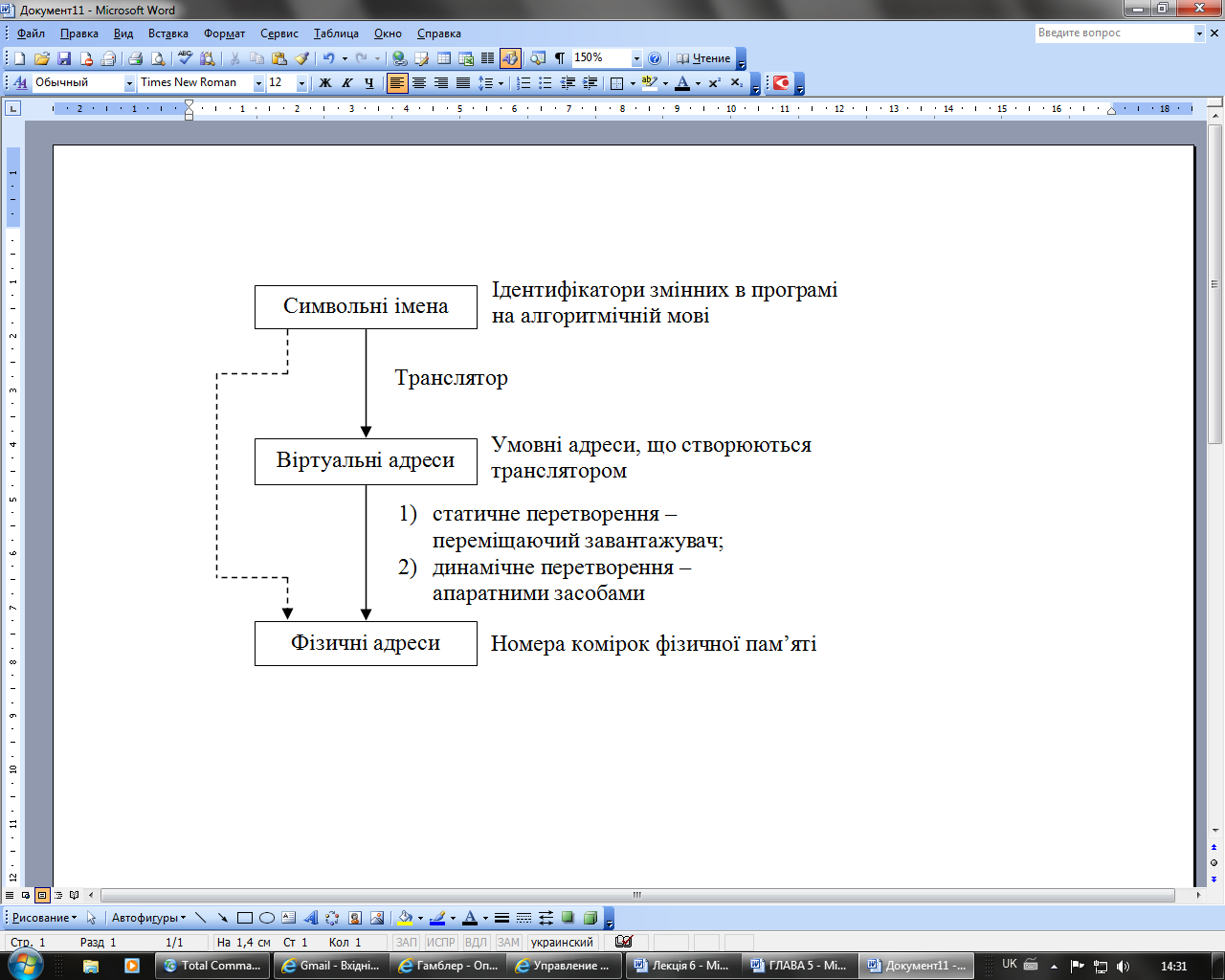
Для ідентифікації змінних і команд використовуються символьні імена (мітки), віртуальні адреси і фізичні адреси (рис. 1).

Рис. 1. Типи адрес

**Символьні імена** привласнює користувач при написанні програми на алгоритмічній мові або асемблері.

**Віртуальні адреси** (також **математичні**, або **логічними** адреси) виробляє транслятор, що переводить програму на машинну мову. Оскільки під час трансляції в загальному випадку не відомо, в яке місце оперативної пам’яті буде завантажена програма, то транслятор привласнює змінним і командам віртуальні (умовні) адреси. Сукупність віртуальних адрес процесу називається **віртуальним адресним простором**. Кожен процес має власний віртуальний адресний простір. Максимальний розмір віртуального адресного простору обмежується розрядністю адреси, властивій даній архітектурі комп’ютера, і, як правило, не співпадає з об’ємом фізичної пам’яті, наявної в комп’ютері.

**Фізичні адреси** відповідають номерам комірок оперативної пам’яті, де насправді розташовані або будуть розташовані змінні і команди. Перехід від віртуальних адрес до фізичних може здійснюватися двома способами. У першому випадку заміну віртуальних адрес на фізичних робить спеціальна системна програма – **переміщаючий завантажувач**. Переміщаючий завантажувач на підставі наявних у нього вихідних даних про початкову адресу фізичної пам’яті, в яку належить завантажувати програму, а такаж інформації, наданої транслятором про адресно-залежні елементи програми, виконує завантаження програми, поєднуючи її із заміною віртуальних адрес фізичними.

Другий спосіб полягає в тому, що програма завантажується в пам’ять в незміненому виді у віртуальних адресах, при цьому операційна система фіксує зміщення дійсного розташування програмного коду відносно віртуального адресного простору. Під час виконання програми при кожному зверненні до оперативної пам’яті виконується перетворення віртуальної адреси у фізичну. Другий спосіб є гнучкішим, він допускає переміщення програми під час її виконання, тоді як переміщаючий завантажувач жорстко прив’язує програму до наперед виділеної для неї ділянки пам’яті. В той же час використання переміщаючого завантажувача зменшує накладні витрати, оскільки перетворення кожної віртуальної адреси відбувається тільки один раз під час завантаження, а в другому випадку - кожного разу при зверненні за цією адресою.

В деяких випадках (зазвичай в спеціалізованих системах), коли заздалегідь точно відомо, в якій області оперативної пам’яті виконуватиметься програма, транслятор видає виконавчий код відразу у фізичних адресах.

2. Методи розподілу пам’яті без використання дискового простору

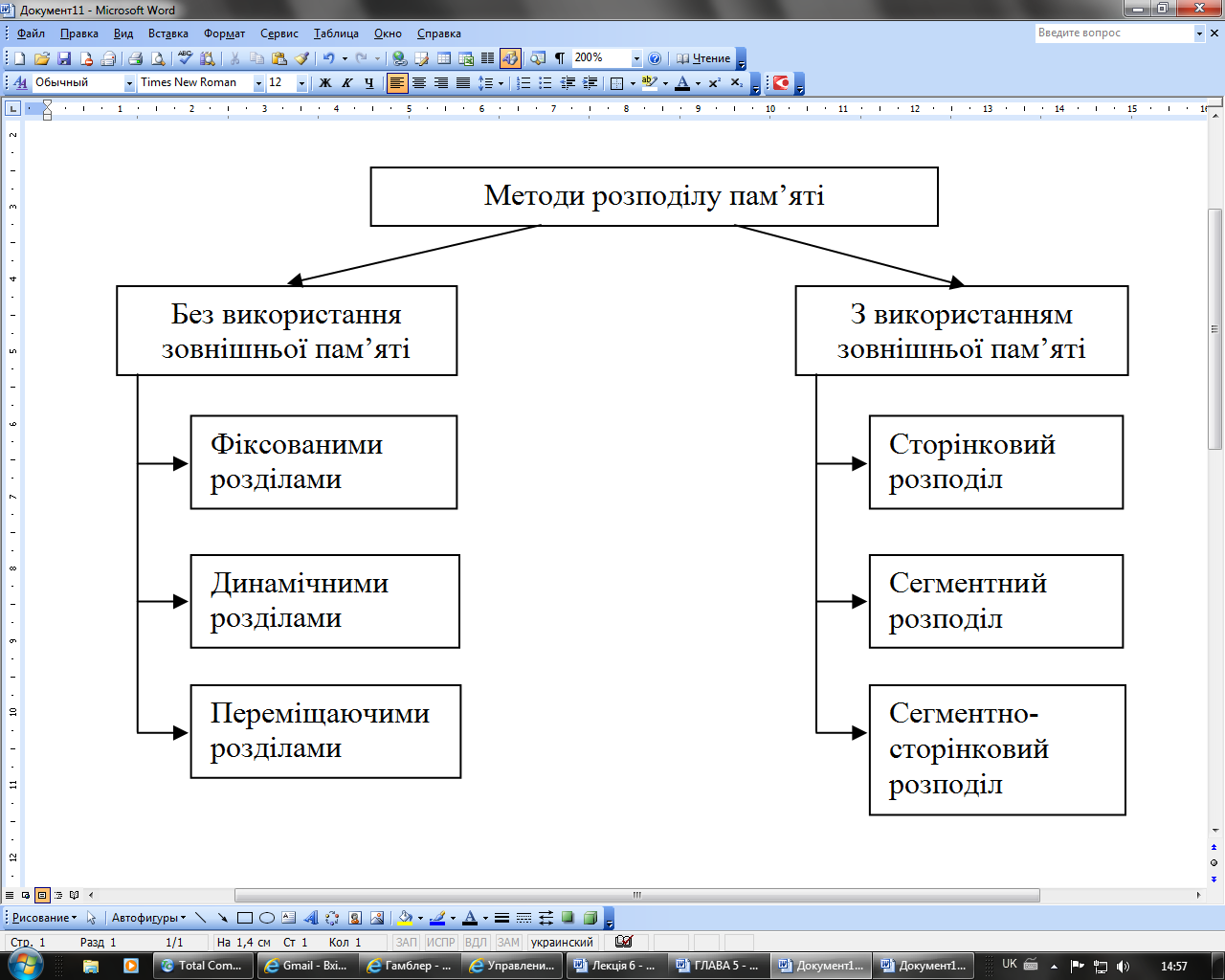
Усі методи управління пам’яттю можуть бути розділені на два класи: методи, які використовують переміщення процесів між оперативною пам’яттю і диском, і методи, які не роблять цього (рис. 2). Почнемо з останнього, простішого класу методів.

Рис. 2. Класифікація методів розподілу пам’яті

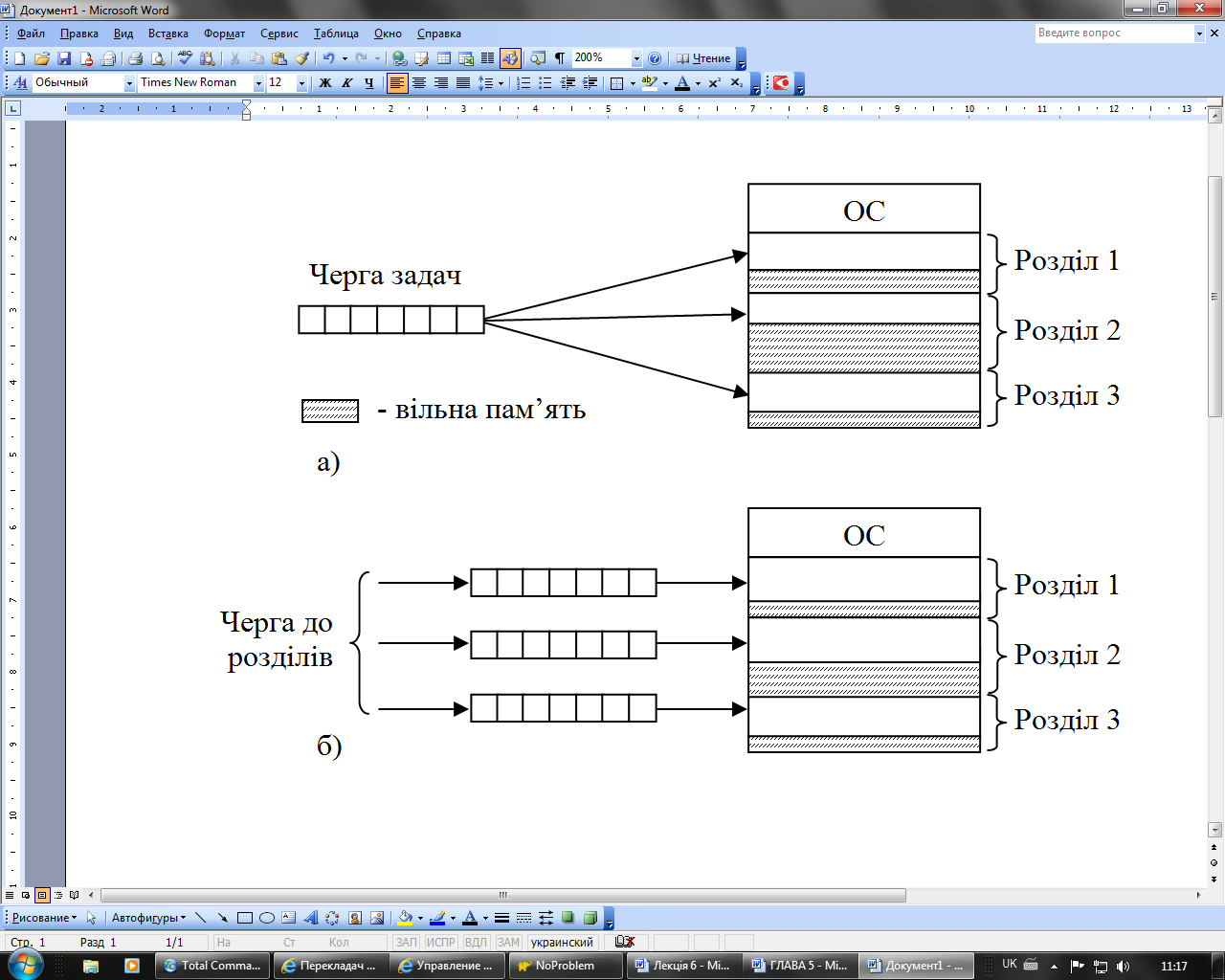
**Розподіл пам’яті фіксованими розділами**

Найпростішим способом управління оперативною пам’яттю є розподіл її на декілька розділів фіксованої величини. Це може бути виконано вручну оператором під час старту системи або під час її генерації. Чергове завдання, що поступило на виконання, поміщається або в загальну чергу (рис 3, а), або в чергу до деякого розділу (рис 3, б).

Підсистема управління пам’яттю в цьому випадку виконує наступні завдання:

* Порівнює об’єм пам’яті, необхідний для програми, що поступила на виконання, з розмірами вільних розділів і вибирає підходящий розділ.
* Здійснює завантаження програми в один із розділів і налаштування адрес.

При очевидній перевазі - простоті реалізації - цей метод має істотний недолік - жорсткість. Оскільки в кожному розділі може виконуватися тільки одна програма, то рівень мультипрограмування заздалегідь обмежений числом розділів не залежно від того, який розмір мають програми. Навіть якщо програма має невеликий об’єм, вона займатиме увесь розділ, що призводить до неефективного використання пам’яті. З іншого боку, розбиття пам’яті на розділи не дозволяє виконувати процеси, програми яких не поміщаються ні в один з розділів, але для яких було б достатньо пам’яті декількох розділів.

Рис. 3. Розподіл пам’яті фіксованими розділами:

а - із загальною чергою; б - з окремими чергами

Такий спосіб управління пам’яттю застосовувався в ранніх мультипрограмних ОС. Зараз метод розподілу пам’яті фіксованими розділами знаходить застосування в системах реального часу, в основному завдяки невеликим витратам на реалізацію.

**Розподіл пам’яті динамічними розділами**

В цьому випадку пам’ять машини не ділиться заздалегідь на розділи. Спочатку уся пам’ять вільна. Кожному новому завданню виділяється необхідна їй пам’ять. Якщо достатній об’єм пам’яті відсутній, то завдання не приймається на виконання і стоїть в черзі. Після завершення процесу пам’ять звільняється, і на це місце може бути завантажений інший процес. Таким чином, в довільний момент часу оперативна пам’ять є випадковою послідовністю зайнятих і вільних ділянок (розділів) довільного розміру. На рис. 4 показаний стан пам’яті в різні моменти часу при використанні динамічного розподілу. Так у момент t0 в пам’яті знаходиться тільки ОС, а до моменту t1 пам’ять розділена між 5 процесами, причому процес П4, завершуючись, покидає пам’ять. На звільнене від процесу П4 місце завантажується процес П6, що поступив у момент t3.

Завданнями операційної системи при реалізації цього методу управління пам’яттю є:

* ведення таблиць вільних і зайнятих областей, в яких вказуються початкові адреси і розміри ділянок пам’яті;
* при поступленні нового процесу - аналіз запиту, перегляд таблиці вільних областей і вибір розділу, розмір якого достатній для розміщення процесу, що поступив;
* завантаження процесу у виділений йому розділ і коригування таблиць вільних і зайнятих областей,
* після завершення процесу корегування таблиць вільних і зайнятих областей.

Програмний код не переміщається під час виконання, тобто може бути проведене одноразове налаштування адрес за допомогою використання переміщаючого завантажувача.

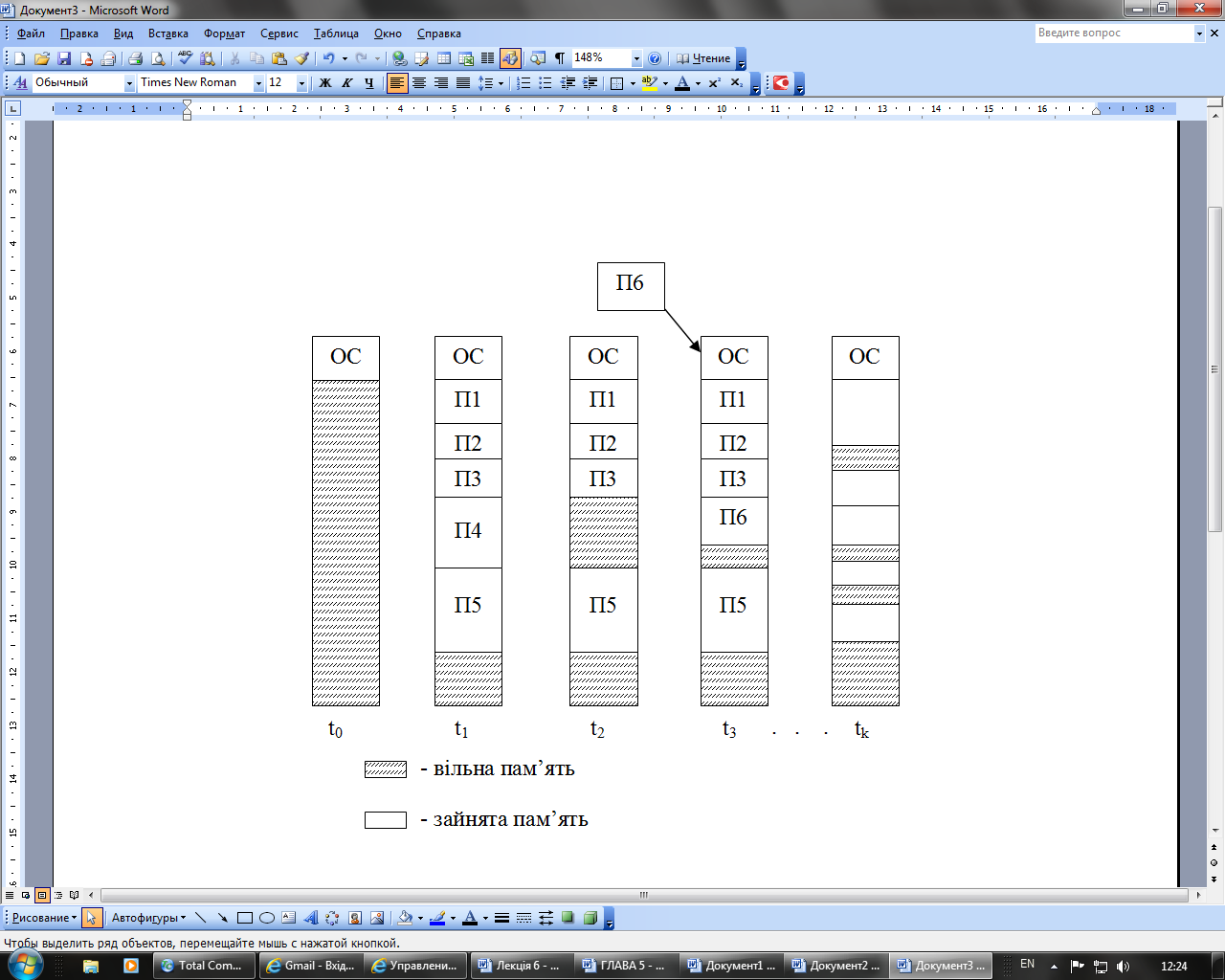
В порівнянні з методом розподілу пам’яті фіксованими розділами цей метод має набагато більшу гнучкість, але йому властивий дуже серйозний недолік – фрагментація пам’яті. **Фрагментація** – це наявність великого числа несуміжних ділянок вільної пам’яті дуже малого розміру (фрагментів). Настільки малого, що жоден з поступаючих процесів не може поміститися ні в одній з ділянок, хоча сумарний об’єм фрагментів може скласти значну величину, що набагато перевищує необхідний об’єм пам’яті.

Рис. 4. Розподіл пам’яті динамічними розділами

Розподіл пам’яті динамічними розділами лежить в основі підсистем управління пам’яттю багатьох мультипрограмних ОС 60-70-х років, зокрема популярної ОС OS/360.

**Розподіл пам’яті переміщаючими розділами**

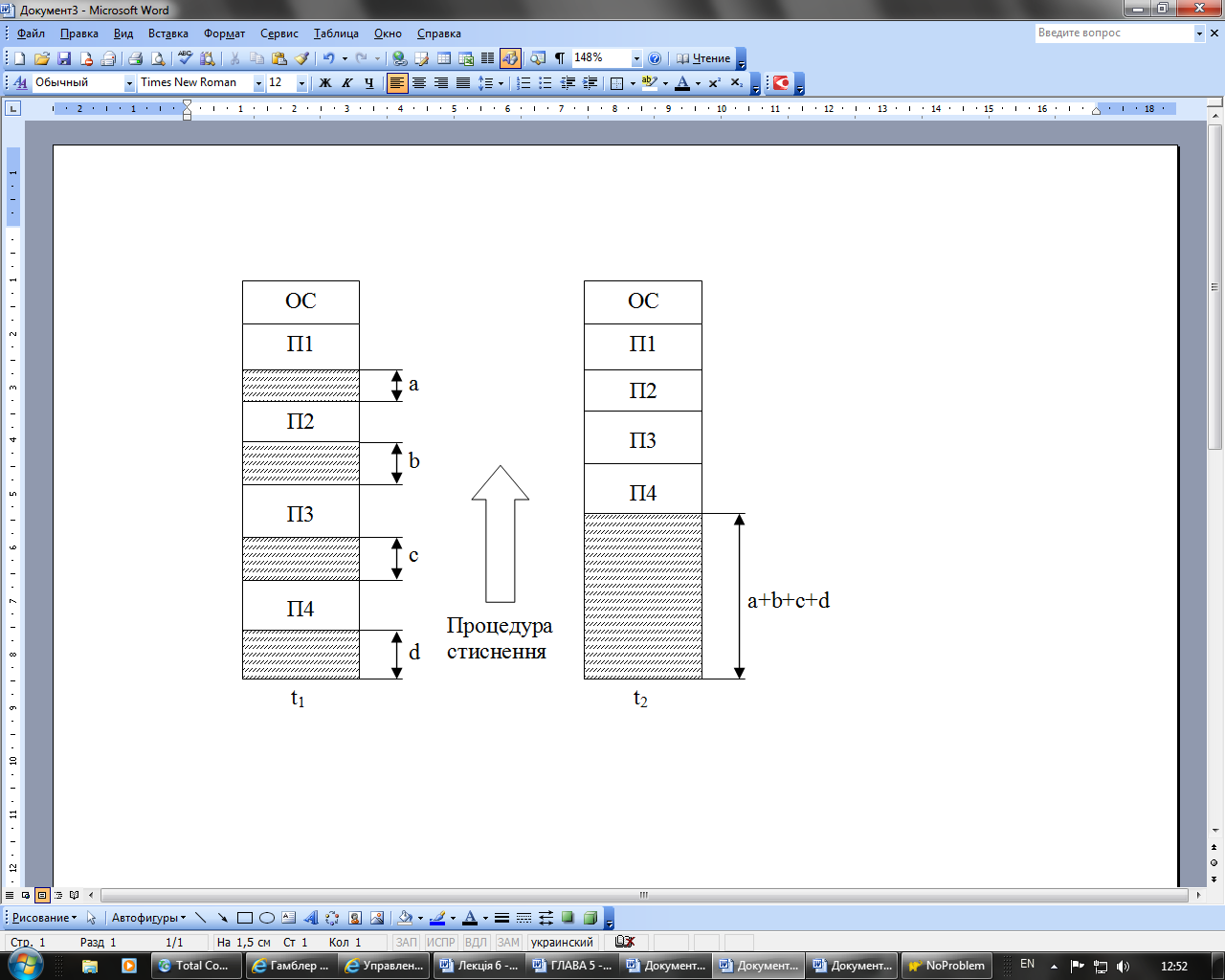
Одним з методів боротьби з фрагментацією є переміщення усіх зайнятих ділянок у бік старших або у бік молодших адрес, так, щоб уся вільна пам’ять утворювала єдину вільну область (рис. 5). На додаток до функцій, які виконує ОС при розподілі пам’яті змінними розділами, в даному випадку вона повинна ще й час від часу копіювати вміст розділів з одного місця пам’яті в інше, коригуючи таблиці вільних і зайнятих областей. Ця процедура називається **стисканням**. Стискання може виконуватися або при кожному завершенні процесу, або тільки тоді, коли для нового процесу немає вільного розділу достатнього розміру. У першому випадку вимагається менше обчислювальної роботи при коригуванні таблиць, а в другому – рідше виконується процедура стискання. Оскільки програми переміщаються по оперативній пам’яті в ході свого виконання, то перетворення адрес з віртуальної форми у фізичну повинно виконуватися динамічним методом.

Рис. 5. Розподіл пам’яті переміщуваними розділами

Хоча процедура стискання і призводить до ефективнішого використання пам’яті, вона може вимагати значного часу, що часто переважує переваги цього методу.

3. Методи розподілу пам’яті з використанням дискового простору

**Віртуалізація пам’яті**

Вже досить давно користувачі зіткнулися з проблемою розміщення в пам’яті програм, розмір яких перевищував наявну вільну пам’ять. Рішенням було розбиття програми на частини, звані **оверлеями**. Коли перший оверлей закінчував своє виконання, він викликав інший оверлей.Усі оверлеї зберігалися на диску і переміщалися між пам’яттю і диском засобами ОС. Проте розбиття програми на частини і планування їх завантаження в оперативну пам’ять повинен був здійснювати програміст.

Розвиток організації обчислювального процесу в цьому напрямі привів до появи методів, що базуються на основі підміни (**віртуалізації**) оперативної пам’яті дисковою пам’яттю, що дозволяє підвищити рівень мультипрограмування - об’єм оперативної пам’яті комп’ютера тепер не так жорстко обмежує кількість одночасно виконуваних процесів. **Віртуальним** називається ресурс, який користувачу або програмі користувача представляється володіючим властивостями, якими він насправді не володіє. Так, наприклад, користувачеві може бути надана віртуальна оперативна пам’ять, розмір якої перевершує усю наявну в системі реальну оперативну пам’ять. Користувач пише програми так, як ніби в його розпорядженні є однорідна оперативна пам’ять великого об’єму. Насправді ж усі дані, що використовуються програмою, зберігаються на дисках і лише при необхідності частинами завантажуються в реальну оперативну пам’ять.

Віртуалізація оперативної пам’яті здійснюється сукупністю програмних модулів ОС і апаратних схем процесора і включає вирішення наступних завдань:

* розміщення даних в запам’ятовуючих пристроях різного типу, наприклад, частину програми – в оперативній пам’яті, а частину – на диску;
* переміщення по мірі необхідності даних між пам’яттю і диском;
* перетворення віртуальних адрес у фізичні.

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

* **свопінг (swapping)** – образи процесів вивантажуються на диск і повертаються в оперативну пам’ять **цілком**;
* **віртуальна пам’ять (virtual memory)** – між оперативною пам’яттю і диском переміщаються **частини** (сегменти, сторінки і тому подібне) образів процесів.

Свопінг – це простіший в реалізації спосіб спільного використання оперативної пам’яті і диска. Цьому методу властива надмірність: коли ОС вирішує активізувати процес, для його виконання, як правило, не вимагається завантажувати в оперативну пам’ять усі його сегменти повністю. Аналогічно при звільненні пам’яті для завантаження нового процесу дуже часто не вимагається вивантажувати інший процес на диск повністю, досить витіснити частину його образу. Переміщення надлишкової інформації уповільнює роботу системи, а також призводить до неефективного використання пам’яті. Крім того, системи, що підтримують свопінг, мають ще один дуже істотний недолік: вони не здатні завантажити для виконання процес, віртуальний адресний простір якого перевищує наявну вільну пам’ять.

Саме із-за вказаних недоліків свопінг як механізм управління пам’яттю майже не використовується в сучасних ОС. На зміну йому прийшов досконаліший механізм – віртуальної пам’яті, який полягає в тому, що при нестачі місця в оперативній пам’яті на диск вивантажуються тільки частини образів процесів.

Ключовою проблемою віртуальної пам’яті, що виникає в результаті багатократної зміни місця розташування в оперативній пам’яті образів процесів або їх частин, є перетворення віртуальних адрес у фізичні. Вирішення цієї проблеми, у свою чергу, залежить від того, який спосіб структуризації віртуального адресного простору прийнятий в цій системі управління пам’яттю. Найбільш поширеними реалізаціями віртуальної пам’яті є сторінковий, сегментний і сторінково-сегментний розподіл пам’яті, а також свопінг.

**Сторінковий розподіл пам’яті**

На рис. 6 показана схема сторінкового розподілу пам’яті. Віртуальний адресний простір кожного процесу ділиться на частини однакового, фіксованого для цієї системи розміру, звані **віртуальними сторінками (virtual pages)**. У загальному випадку розмір віртуального адресного простору не є кратним розміру сторінки, тому остання сторінка кожного процесу доповнюється фіктивною областю.

Уся оперативна пам’ять машини також ділиться на частини такого ж розміру, звані **фізичними сторінками** (чи блоками).

Розмір сторінки зазвичай вибирається кратним степені двійки: 512, 1024, 4096 байт і так далі, це дозволяє спростити механізм перетворення адрес.

При завантаженні процесу частина його віртуальних сторінок поміщається в оперативну пам’ять, а інші – на диск. Суміжні віртуальні сторінки не обов’язково розташовуються в суміжних фізичних сторінках. При завантаженні операційна система створює для кожного процесу інформаційну структуру – **таблицю сторінок**, в якій встановлюється відповідність між номерами віртуальних і фізичних сторінок для сторінок, завантажених в оперативну пам’ять, або робиться відмітка про те, що віртуальна сторінка вивантажена на диск. Крім того, в таблиці сторінок знаходиться керуюча інформація, така як ознака модифікації сторінки, ознака невивантаженості (вивантаження деяких сторінок може бути заборонене), ознака звернення до сторінки (використовується для підрахунку числа звернень за певний період часу) і інші дані, що формуються і використовуються механізмом віртуальної пам’яті.



Рис. 6. Сторінковий розподіл пам’яті

При активізації чергового процесу в спеціальний регістр процесора завантажується адреса таблиці сторінок цього процесу.

При кожному зверненні до пам’яті відбувається читання з таблиці сторінок інформації про віртуальну сторінку, до якої сталося звернення. Якщо ця віртуальна сторінка знаходиться в оперативній пам’яті, то виконується перетворення віртуальної адреси у фізичну. Якщо ж потрібна віртуальна сторінка в даний момент вивантажена на диск, то відбувається так зване **сторінкове переривання**. Активний процес переводиться в стан очікування, і активізується інший процес з черги готових. Паралельно програма обробки сторінкового переривання знаходить на диску необхідну віртуальну сторінку і намагається завантажити її в оперативну пам’ять. Якщо в пам’яті є вільна фізична сторінка, то завантаження виконується негайно, якщо ж вільних сторінок немає, то вирішується питання, яку сторінку слід вивантажити з оперативної пам’яті.

При виборі сторінки на вивантаження можуть бути використані різні критерії, зміст яких зводиться до одного: на диск виштовхується сторінка, до якої в майбутньому, починаючи з цього моменту, найдовше не буде звернень. Найбільш популярним критерієм вибору сторінки на вивантаження є число звернень до неї за останній період часу.

У деяких системах використовується поняття **робочої множини сторінок**. Робоча множина визначається для кожного процесу і є переліком найбільш часто використовуваних сторінок, які повинні постійно знаходитися в оперативній пам’яті і тому не підлягають вивантаженню.

Після того, як вибрана сторінка, яка повинна покинути оперативну пам’ять, аналізується її ознака модифікації (з таблиці сторінок). Якщо виштовхувана сторінка з моменту завантаження була модифікована, то її нова версія має бути переписана на диск. Якщо ні, то вона може бути просто знищена, тобто відповідна фізична сторінка оголошується вільною.

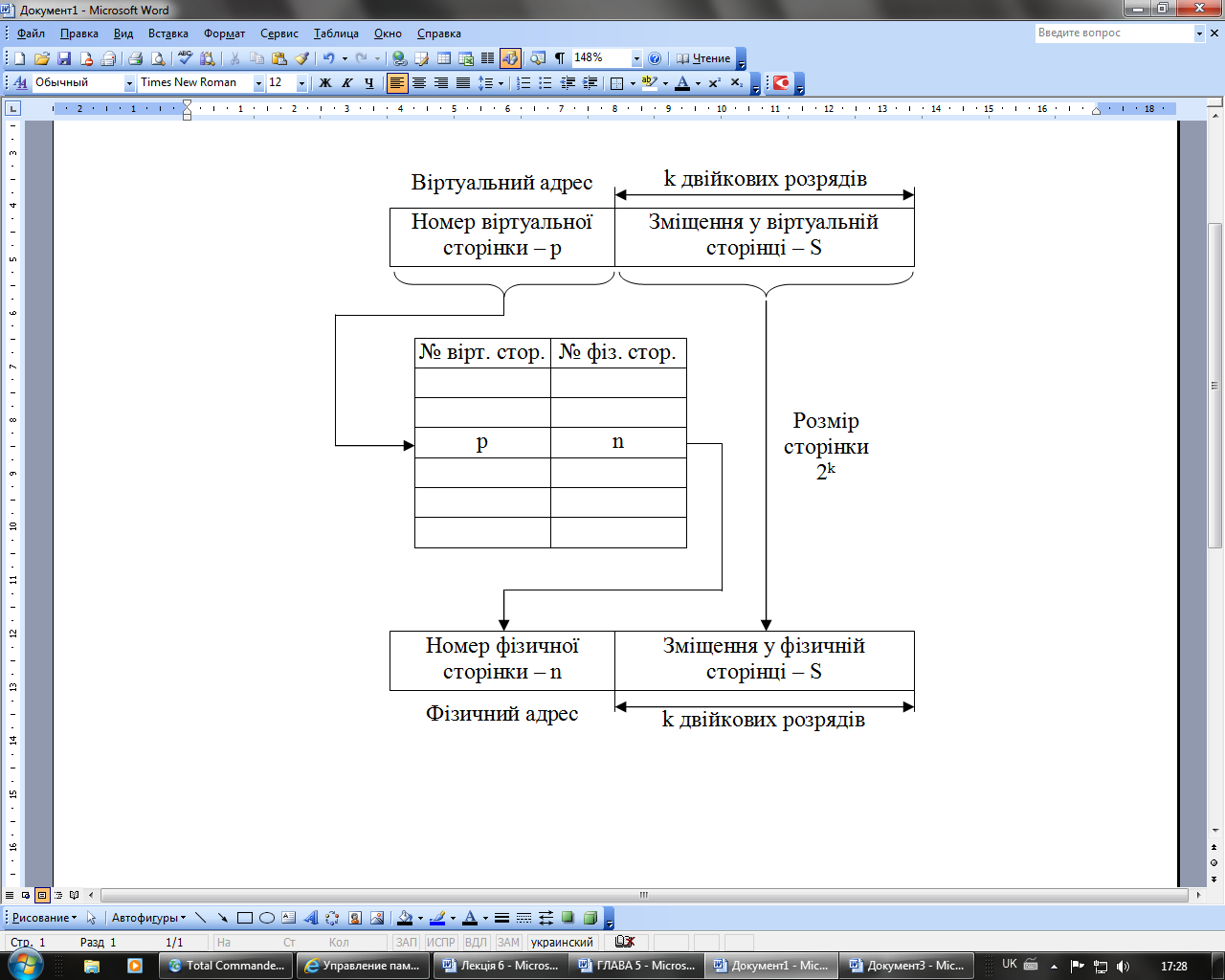
Розглянемо механізм перетворення віртуальної адреси у фізичну при сторінковій організації пам’яті (рис. 7).

Рис. 7. Механізм перетворення віртуальної адреси у фізичну

при сторінковій організації пам’яті

Віртуальна адреса при сторінковому розподілі може бути представлена у вигляді пари (p, s), де p - номер віртуальної сторінки процесу (нумерація сторінок починається з 0), а s - зміщення в межах віртуальної сторінки.

При кожному зверненні до оперативної пам’яті апаратними засобами виконуються наступні дії:

1. на підставі початкової адреси таблиці сторінок, номера віртуальної сторінки р (старші розряди віртуальної адреси) і довжини запису в таблиці сторінок (системна константа) визначається адреса потрібного запису в таблиці;
2. з цього запису вилучається номер фізичної сторінки – n;
3. до номера фізичної сторінки приєднується зміщення s (молодші розряди віртуальної адреси).

На продуктивність системи із сторінковою організацією пам’яті впливають тимчасові затримки, пов’язані з обробкою сторінкових переривань і перетворенням віртуальної адреси у фізичну. При частих сторінкових перериваннях система може витрачати велику частину часу на свопінг сторінок. З одного боку, щоб зменшити частоту сторінкових переривань, слід було б збільшувати розмір сторінки. З іншого боку, якщо сторінка велика, то велика і фіктивна область в останній віртуальній сторінці кожного процесу. Якщо врахувати, що в середньому в кожному процесі фіктивна область складає половину сторінки, то в сумі при великому об’ємі сторінки втрати можуть скласти істотну величину. На практиці розробники ОС і процесорів обмежуються деяким раціональним рішенням, придатним для широкого класу обчислювальних систем. Типовий розмір сторінки складає (4 Кбайти).

Сторінковий розподіл пам’яті може бути реалізований в спрощеному варіанті, без вивантаження сторінок на диск. В цьому випадку усі віртуальні сторінки усіх процесів повинні постійно знаходитись в оперативній пам’яті. Такий варіант сторінкової організації майже виключає фрагментацію. Такий режим роботи системи управління пам’яттю використовується в деяких спеціалізованих ОС, коли вимагається висока реактивність системи і здатність виконувати змінний набір додатків (Novell NetWare 4.x).

**Сегментний розподіл пам’яті**

При сторінковій організації віртуальний адресний простір процесу ділиться механічно на рівні частини. Це не дозволяє диференціювати способи доступу до різних частин програми (сегментів). Розбиття програми на "осмислені" частини робить принципово можливим розподіл одного сегменту декількома процесами. Наприклад, якщо два процеси використовують одну і ту ж математичну підпрограму, то в оперативну пам’ять може бути завантажена тільки одна копія цієї підпрограми.

На рис. 8 показано реалізацію сегментного розподілу пам’яті. Віртуальний адресний простір процесу ділиться на сегменти, розмір яких визначається програмістом з урахуванням змістовного значення інформації, що міститься в них. Окремий сегмент може бути підпрограмою, масивом даних і тому подібне.

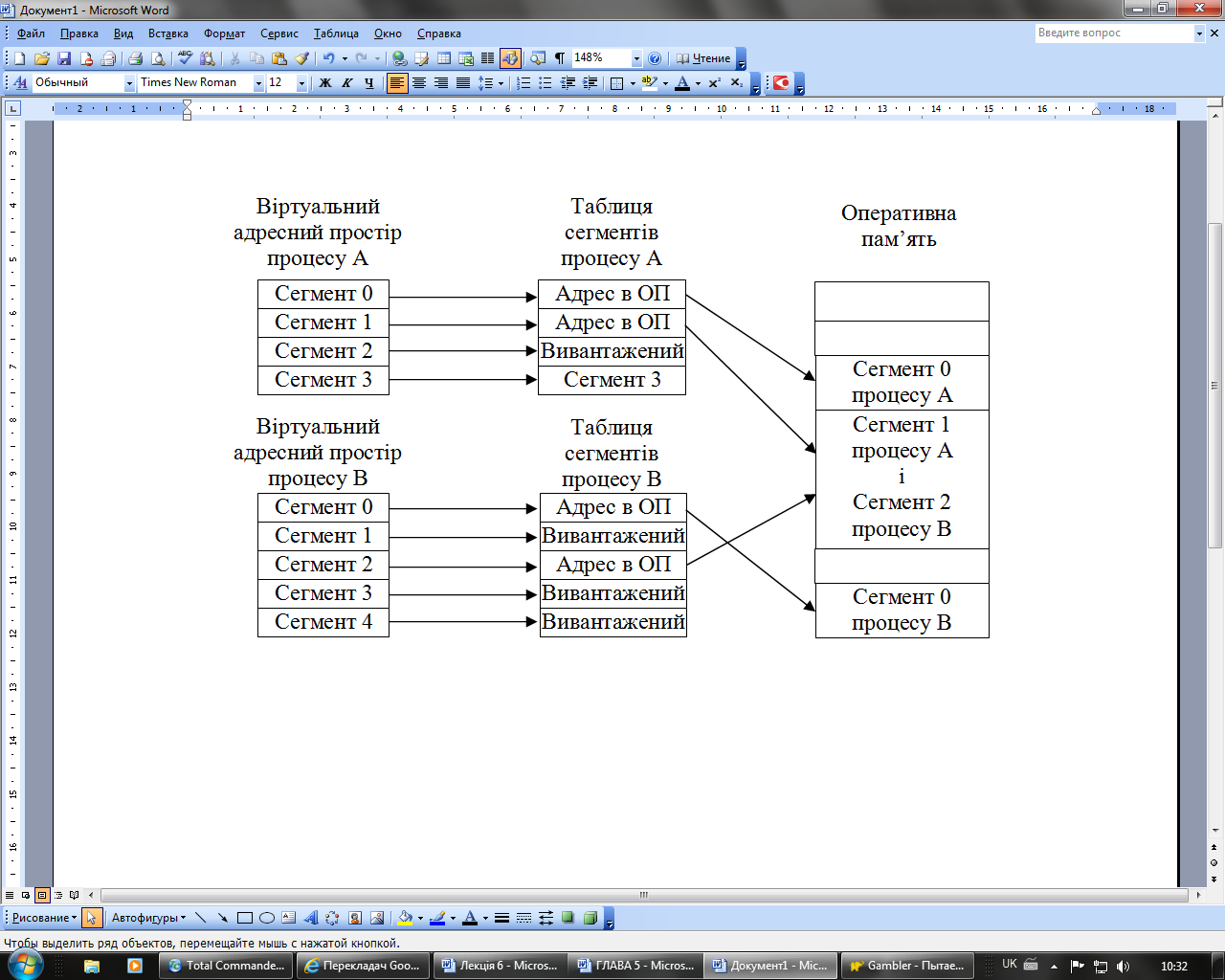


Рис. 8. Розподіл пам’яті сегментами

При завантаженні процесу частина сегментів поміщається в оперативну пам’ять (при цьому для кожного з цих сегментів ОС підшукує відповідну ділянку вільної пам’яті), а частина сегментів розміщується в дисковій пам’яті. Сегменти однієї програми можуть займати в оперативній пам’яті несуміжні ділянки. Під час завантаження система створює таблицю сегментів процесу (аналогічну таблиці сторінок), в якій для кожного сегменту вказується початкова фізична адреса сегменту в оперативній пам’яті, розмір сегменту, правила доступу, ознака модифікації, ознака звернення до цього сегменту за останній інтервал часу і інша інформація. Якщо віртуальні адресні простори декількох процесів включають один і той же сегмент, то в таблицях сегментів цих процесів робляться посилання на одну і ту ж ділянку оперативної пам’яті, в який цей сегмент завантажується в єдиному екземплярі.

Система з сегментною організацією функціонує аналогічно системі із сторінковою організацією: час від часу відбуваються переривання, пов’язані з відсутністю потрібних сегментів в пам’яті, при необхідності звільнення пам’яті деякі сегменти вивантажуються, при кожному зверненні до оперативної пам’яті виконується перетворення віртуальної адреси у фізичну.

Віртуальна адреса при сегментній організації пам’яті може бути представлена парою (g, s), де g - номер сегменту, а s - зміщення в сегменті. Фізична адреса отримується шляхом складання початкової фізичної адреси сегменту, знайденого в таблиці сегментів по номеру g, і зміщення s.

Недоліком цього методу розподілу пам’яті є фрагментація на рівні сегментів і повільніше, в порівнянні із сторінковою організацією, перетворення адреси.

Однією з істотних переваг сегментної організації пам’яті над сторінковою є можливість задання диференційованих прав доступу процесу до його сегментів. Наприклад, один сегмент даних, що містить початкову інформацію для додатка, може мати права доступу "тільки читання", а сегмент даних, що містить результати, - "читання і запис".

**Сегментно-сторінковий розподіл пам’яті**

Даний метод є комбінацією сторінкового і сегментного розподілу пам’яті і, внаслідок цього, поєднує в собі переваги обох підходів. Віртуальний простір процесу ділиться на сегменти, а кожен сегмент у свою чергу ділиться на віртуальні сторінки, які нумеруються в межах сегменту. Оперативна пам’ять ділиться на фізичні сторінки. Завантаження процесу виконується ОС посторінково, при цьому частина сторінок розміщується в оперативній пам’яті, а частина на диску. Для кожного сегменту створюється своя таблиця сторінок, структура якої повністю співпадає із структурою таблиці сторінок, що використовується при сторінковому розподілі. Для кожного процесу створюється таблиця сегментів, в якій вказуються адреси таблиць сторінок для усіх сегментів цього процесу. Адреса таблиці сегментів завантажується в спеціальний регістр процесора, коли активізується відповідний процес. На рис. 9 показана схема перетворення віртуальної адреси у фізичну для цього методу.

4. Кешування даних

Пам’ять комп’ютера є ієрархією запам’ятовуючих пристроїв (внутрішні регістри процесора, різні типи надоперативної і оперативної пам’яті, диски), що відрізняються середнім часом доступу і вартістю зберігання даних з розрахунку на один біт (рис. 10).

**Кеш-пам’ять**,або просто **кеш (cache)** – це спосіб організації спільного функціонування двох типів запам’ятовуючих пристроїв, що відрізняються часом доступу до даних, який дозволяє зменшити середній час доступу до даних за рахунок динамічного копіювання в «швидкий» ЗП найбільш часто використовуваної інформації з «повільного» ЗП. Кеш-пам’яттю часто називають не лише спосіб організації роботи двох типів запам’ятовуючих пристроїв, але й один з пристроїв – «швидкий» ЗП. Важливо, що механізм кеш-пам’яті є прозорим для користувача, який не повинен повідомляти ніякої інформації про інтенсивність використання даних і не повинен брати участь в переміщенні даних із ЗП одного типу в ЗП іншого типу, усе це робиться автоматично системними засобами.

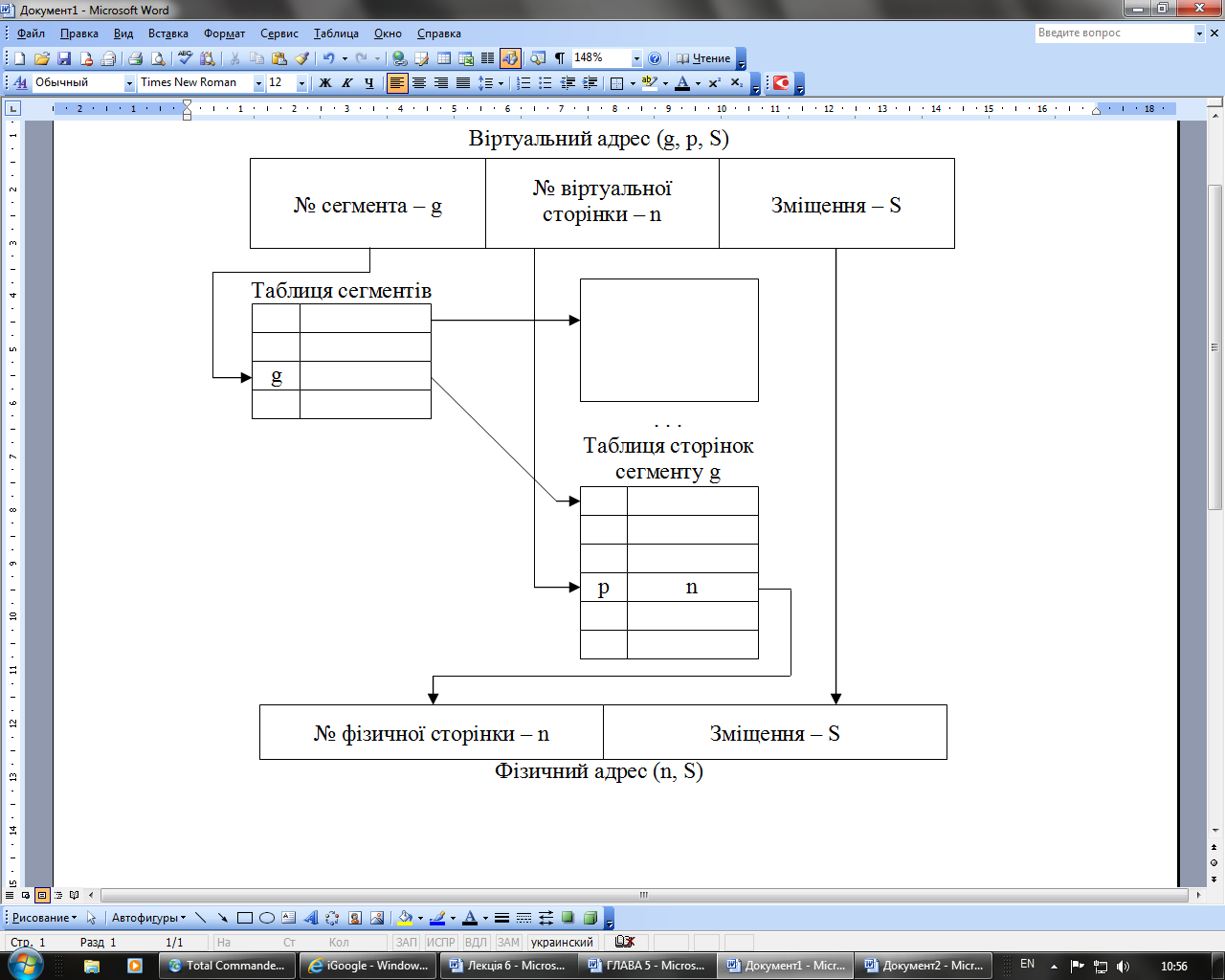


Рис. 9. Схема перетворення віртуальної адреси у фізичну для сегментно-сторінкової організації пам’яті

**Кешування** – це універсальний метод, придатний для прискорення доступу до оперативної пам’яті, до диска і до інших видів запам’ятовуючих пристроїв. Якщо кешування застосовується для зменшення середнього часу доступу до оперативної пам’яті, то в якості кешу використовують швидкодіючу статичну пам’ять. Якщо кешування використовується системою вводу-виводу для прискорення доступу до даних, що зберігаються на диску, то в цьому випадку роль кеш-пам’яті виконують буфери в оперативній пам’яті, в яких розміщуються часто використовувані дані.

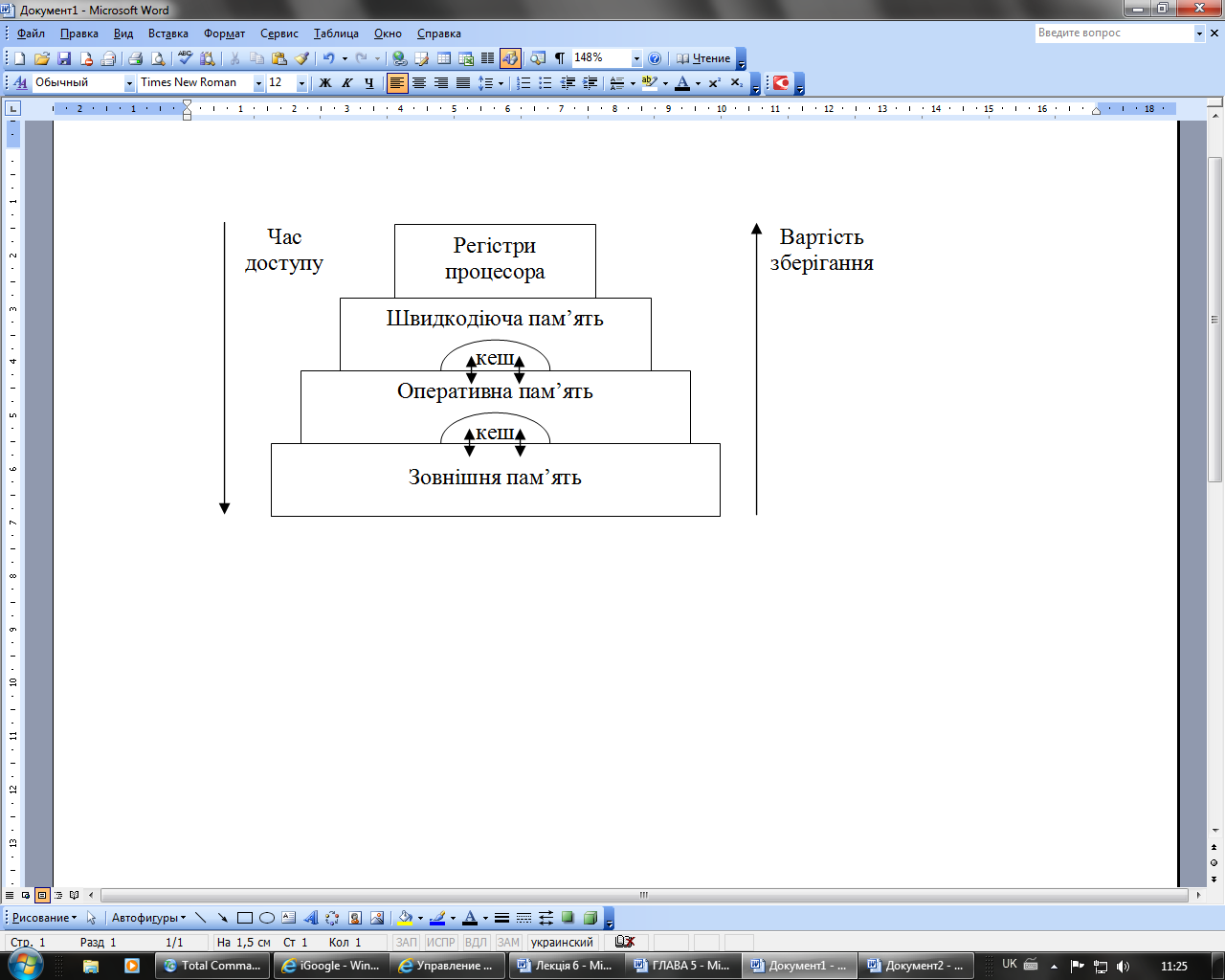


Рис.10. Ієрархія ЗП

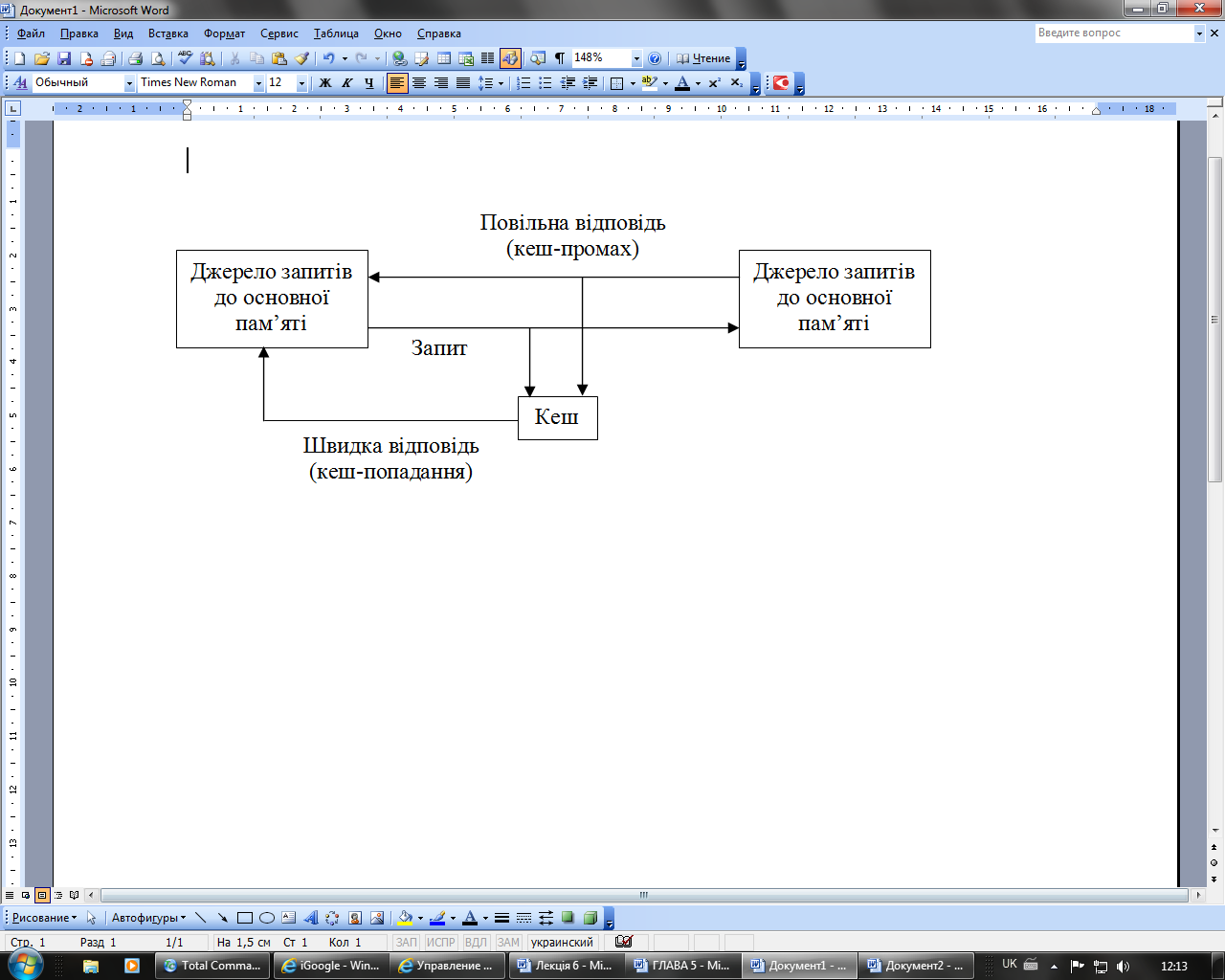
Розглянемо окремий випадок кешування (рис. 11). Вміст кеш-пам’яті являє собою сукупністю записів про усі завантажені в неї елементи даних з основної пам’яті. Кожен запис про елемент даних включає в себе: значення елемента даних; адресу, яку цей елемент даних має в основній пам’яті; керуючу інформацію (ознаку модифікації і ознаку звернення до даних за певний період часу).

Рис. 11. Схема функціонування кеш-пам’яті

При кожному зверненні до основної пам’яті за фізичною адресою переглядається вмістиме кеш-пам’яті з метою визначення, чи не знаходяться там потрібні дані. Далі можливий один з двох варіантів розвитку подій :

* якщо дані виявляються в кеш-пам’яті, тобто сталося **кеш-попадання (cache-hit)**, вони зчитуються з неї і результат передається джерелу запиту;
* якщо потрібні дані відсутні в кеш-пам’яті, тобто стався **кеш-промах (cache-miss)**, вони зчитуються з основної пам’яті, передаються джерелу запиту і одночасно з цим копіюються в кеш-пам’ять. При копіюванні даних може виявитися, що в кеш-пам’яті немає вільного місця, тоді вибираються дані, до яких в останній період було менше всього звернень, для витіснення з кеш-пам’яті.

На практиці в кеш-пам’ять зчитується не один елемент даних, до якого сталося звернення, а цілий блок даних, що збільшує ймовірність кеш-попадання.

У реальних системах ймовірність кеш-попадання складає приблизно 90%. Таке високе значення пояснюється наявністю у даних об’єктивних властивостей: просторової і часової локальності.

* **Просторова локальність.** Якщо сталося звернення за деякою адресою, то з високою ймовірністю найближчим часом станеться звернення до сусідніх адрес.
* **Часова локальність.** Якщо сталося звернення за деякою адресою, то наступне звернення за тією ж адресою з великою ймовірністю станеться найближчим часом.

Лекція 7

Управління вводом-виводом і файлова система

1. Фізична організація пристроїв вводу-виводу

Однією з головних функцій ОС є управління усіма пристроями вводу-виводу комп’ютера. ОС повинна передавати пристроям команди, перехоплювати переривання і обробляти помилки; вона також повинна забезпечувати інтерфейс між пристроями і іншою частиною системи.

Пристрої вводу-виводу діляться на два типи: блок-орієнтовані пристрої і байт-орієнтовані пристрої. Блок-орієнтовані пристрої зберігають інформацію в блоках фіксованого розміру, кожен з яких має свою власну адресу. Найпоширеніший блок-орієнтований пристрій – диск. Байт-орієнтовані пристрої не адресовані і не дозволяють проводити операцію пошуку, вони генерують або зчитують послідовність байтів. Прикладами є принтери, мережеві адаптери.

Зовнішній пристрій зазвичай складається з механічного і електронного компонента. Електронний компонент називається контролером пристрою або адаптером. Механічний компонент являє власне пристрій. Деякі контролери можуть управляти декількома пристроями. Якщо інтерфейс між контролером і пристроєм стандартизований, то незалежні виробники можуть випускати сумісні як контролери, так і пристрої.

Операційна система зазвичай має справу не з пристроєм, а з контролером. Контролер, як правило, виконує прості функції, наприклад, перетворює потік біт в блоки, що складаються з байт і здійснює контроль і виправлення помилок. Кожен контролер має декілька регістрів, які використовуються для взаємодії з центральним процесором. У деяких комп’ютерах ці регістри є частиною фізичного адресного простору. У таких комп’ютерах немає спеціальних операцій вводу-виводу. У інших комп’ютерах адреси регістрів вводу-виводу, званих часто портами, утворюють власний адресний простір за рахунок вступу спеціальних операцій вводу-виводу.

ОС виконує ввід-вивід, записуючи команди в регістри контролера. Наприклад, контролер зовнішнього диску приймає 15 команд, таких як READ, WRITE, FORMAT і так далі. Коли команда прийнята, процесор залишає контролер і займається іншою роботою. При завершенні команди контролер організовує переривання для того, щоб передати управління процесором операційній системі, яка повинна перевірити результати операції. Процесор отримує результати і статус пристрою, читаючи інформацію з регістрів контролера.

2. Організація програмного забезпечення вводу-виводу

Основна ідея організації програмного забезпечення вводу-виводу полягає в розбитті його на декілька рівнів, причому нижні рівні забезпечують екранування особливостей апаратури від верхніх, а ті, у свою чергу, забезпечують зручний інтерфейс для користувачів.

Ключовим принципом є незалежність від пристроїв. Вид програми не повинен залежати від того, чи читає вона дані з оптичного диска або з жорсткого диска.

Важливим питанням для програмного забезпечення вводу-виводу є обробка помилок. В загальному, помилки слід обробляти як можна ближче до апаратури. Якщо контролер виявляє помилку читання, то він повинен спробувати її скоректувати. Якщо ж це йому не вдається, то виправленням помилок повинен зайнятися драйвер пристрою. І тільки якщо нижній рівень не може впоратися з помилкою, він повідомляє про помилку верхній рівень.

Ще одне ключове питання – це використання блокуючих (синхронних) і неблокуючих (асинхронних) передач. Більшість операцій фізичного вводу-виводу виконуються асинхронно – процесор починає передачу і переходить на іншу роботу, поки не настає переривання. Програми користувача набагато легше писати, якщо операції вводу-виводу блокуючі – після команди READ програма автоматично припиняється до тих пір, поки дані не потраплять в буфер програми. ОС виконує операції вводу-виводу асинхронно, але представляє їх для програм користувача в синхронній формі.

Остання проблема полягає в тому, що одні пристрої є поділюваними, а інші – виділеними. Диски – це поділювані пристрої, оскільки одночасний доступ декількох користувачів до диску не є проблемою. Принтери – це виділені пристрої, тому що не можна змішувати дані, що друкуються різними користувачами. Наявність виділених пристроїв створює для операційної системи деякі проблеми.

Для вирішення поставлених проблем доцільно розділити програмне забезпечення вводу-виводу на чотири шари (рис. 1):

* Обробка переривань,
* Драйвери пристроїв,
* Незалежний від пристроїв шар операційної системи,
* Користувацький шар програмного забезпечення.

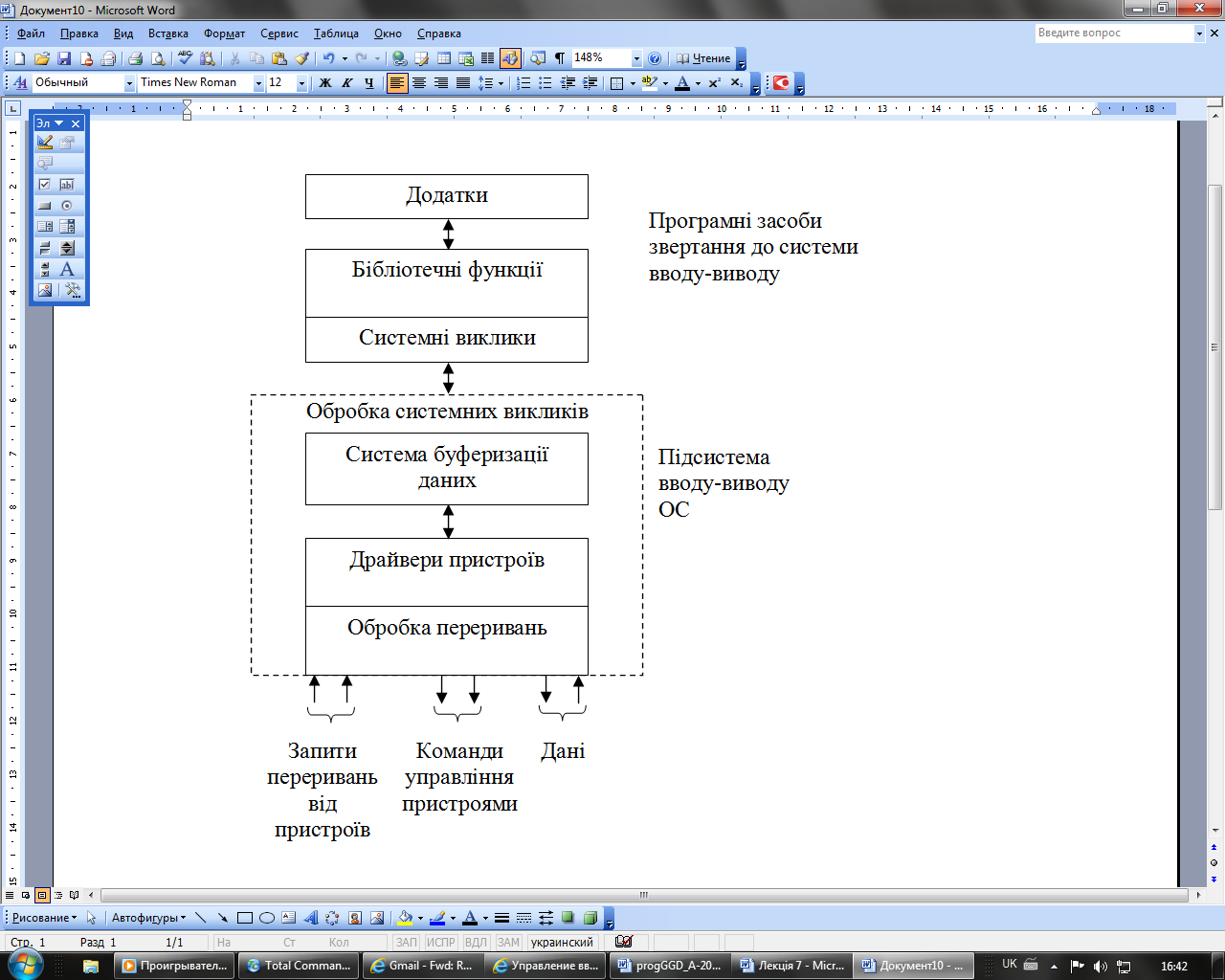


Рис. 1. Багаторівнева організація підсистеми вводу-виводу

**Обробка переривань**

Переривання мають бути приховані як можна глибше в надрах операційної системи, щоб як можна менша частина ОС мала з ними справу. Найкращий спосіб полягає в дозволі процесу, що ініціював операцію вводу-виводу, блокувати себе до завершення операції і настання переривання. Процес може блокувати себе, використовуючи, наприклад, виклик DOWN для семафора, або виклик WAIT для змінної умови, або виклик RECEIVE для очікування повідомлення. При настанні переривання процедура обробки переривання виконує розблокування процесу, що ініціював операцію вводу-виводу, використовуючи виклики UP, SIGNAL або посилаючи процесу повідомлення. У будь-якому випадку ефект від переривання полягатиме в тому, що раніше заблокований процес тепер продовжить своє виконання.

**Драйвери пристроїв**

Увесь залежний від пристрою код поміщається в драйвер пристрою. Кожен драйвер управляє пристроями одного типу або, можливо, одного класу.

У ОС тільки драйвер пристрою знає про конкретні особливості певного пристрою. Наприклад, тільки драйвер диска має справу з доріжками, секторами, циліндрами і іншими чинниками, що забезпечують правильну роботу диска.

Драйвер пристрою приймає запит від пристроїв програмного шару і вирішує, як його виконати. Типовим запитом є читання n блоків даних. Якщо драйвер був вільний під час надходження запиту, то він починає виконувати запит негайно. Якщо ж він був зайнятий обслуговуванням іншого запиту, то запит, що поступив, приєднується до черги вже наявних запитів, і він буде виконаний, коли наступить його черга.

Перший крок в реалізації запиту вводу-виводу полягає в перетворенні його з абстрактної форми в конкретну. Наприклад, для дискового драйвера це означає перетворення номерів блоків в номери циліндрів, головок, секторів, перевірку, чи працює мотор, чи знаходиться головка над потрібним циліндром. Отже, він повинен вирішити, які операції контролера треба виконати і в якій послідовності.

Після передачі команди контролеру драйвер повинен вирішити, чи блокувати себе до закінчення заданої операції або ні. Якщо операція займає значний час, як при друці деякого блоку даних, то драйвер блокується до тих пір, поки операція не завершиться, і обробник переривання не розблоковує його. Якщо команда вводу-виводу виконується швидко (наприклад, прокрутка екрану), то драйвер чекає її завершення без блокування.

**Незалежний від пристроїв шар операційної системи**

Велика частина програмного забезпечення вводу-виводу є незалежною від пристроїв. Точна межа між драйверами і незалежними від пристроїв програмами визначається системою, оскільки деякі функції, які могли б бути реалізовані незалежним способом, насправді виконані у вигляді драйверів для підвищення ефективності або з інших причин.

Типовими функціями для незалежного від пристроїв шару є:

* забезпечення загального інтерфейсу до драйверів пристроїв,
* іменування пристроїв,
* захист пристроїв,
* забезпечення незалежного розміру блоку,
* буферизація,
* розподіл пам’яті на блок-орієнтованих пристроях,
* розподіл і звільнення виділених пристроїв,
* повідомлення про помилки.

Верхнім шарам програмного забезпечення не зручно працювати з блоками різної величини, тому цей шар забезпечує єдиний розмір блоку, наприклад, за рахунок об’єднання декількох різних блоків в єдиний логічний блок. У зв’язку з цим верхні рівні мають справу з абстрактними пристроями, які використовують єдиний розмір логічного блоку незалежно від розміру фізичного сектора.

При створенні файлу або заповненні його новими даними необхідно виділити йому нові блоки. Для цього ОС повинна вести список або бітову карту вільних блоків диска. На підставі інформації про наявність вільного місця на диску може бути розроблений алгоритм пошуку вільного блоку, реалізований програмним шаром, що знаходиться вище шару драйверів.

**Користувацький шар програмного забезпечення**

Хоча велика частина програмного забезпечення вводу-виводу знаходиться усередині ОС, деяка його частина міститься в бібліотеках, що зв’язуються з програмами користувача. Системні виклики, що включають виклики вводу-виводу, зазвичай робляться бібліотечними процедурами. Стандартна бібліотека вводу-виводу містить велике число процедур, які виконують ввід-вивід і працюють як частина програми користувача.

Іншою категорією програмного забезпечення вводу-виводу є підсистема спулінгу (spooling). **Спулінг** – це спосіб роботи з виділеними пристроями в мультипрограмній системі. Розглянемо типовий пристрій, що вимагає спулінгу – принтер. Хоча технічно легко дозволити кожному процесу користувача відкрити спеціальний файл, пов’язаний з прінтером, такий спосіб небезпечний через те, що процес користувача може монополізувати прінтер на довільний час. Замість цього створюється спеціальний процес – монітор, який отримує виняткові права на використання цього пристрою. Також створюється спеціальний каталог (каталогом спулінгу). Для того, щоб надрукувати файл, процес користувача поміщає інформацію, що виводиться, в цей файл і поміщає його в каталог спулінгу. Процес-монітор по черзі роздруковує усі файли, що містяться в каталозі спулінгу.

3. Логічна організація файлу

**Файлова система** – це частина операційної системи, призначення якої полягає в тому, щоб забезпечити користувачеві зручний інтерфейс при роботі з даними, що зберігаються на диску, і забезпечити спільне використання файлів декількома користувачами і процесами.

Таким чином, файлова система включає:

* сукупність усіх файлів на диску,
* набори структур даних, що використовуються для управління файлами (каталоги файлів, дескриптори файлів, таблиці розподілу вільного і зайнятого простору на диску),
* комплекс системних програмних засобів, що реалізовують управління файлами, зокрема: створення, знищення, читання, запис, іменування, пошук і інші операції над файлами.

**Файл** – це іменована область пам’яті, в яку можна записувати і з якої можна зчитувати дані. Файли зберігаються в енергонезалежній пам’яті. Дані, що зберігаються у файлі, мають певну логічну структуру. Ця структура є базою при розробці програми, призначеної для обробки цих даних. Наприклад, щоб текст міг бути правильно виведений на екран, програма повинна мати можливість виділити окремі слова, рядки, абзаци і т. д. Підтримка структури даних може бути цілком покладено на додаток, або цю роботу може узяти на себе файлова система.

У першому випадку, коли усі дії, пов’язані із структуризацією і інтерпретацією вмістимого файлу цілком відносяться до ведення додатка, файл представляється ФС неструктурованою послідовністю даних. Додаток формулює запити до файлової системи на ввід-вивід, використовуючи загальні для усіх додатків системні засоби, наприклад, вказуючи зміщення від початку файлу і кількість байт, які необхідно зчитати або записати. Потік байт, що поступив до додатка, інтерпретується відповідно до закладеної в програмі логіки. Наприклад, компілятор генерує, а редактор зв’язків сприймає цілком певний формат об’єктного модуля програми. При цьому формат файлу, в якому зберігається об’єктний модуль, відомий тільки цим програмам. Підкреслимо, що інтерпретація даних ніяк не пов’язана з дійсним способом їх зберігання у файловій системі.

Модель файлу, відповідно до якої вміст файлу представляється неструктурованою послідовністю (потоком) байт, стала популярною разом з ОС UNIX, а тепер вона широко використовується в більшості сучасних ОС (Windows, NetWare). Неструктурована модель файлу дозволяє легко організувати розподіл файлу між декількома додатками: різні додатки можуть по-своєму структурувати і інтерпретувати дані, що містяться у файлі.

Інша модель файлу, яка застосовувалася в ОС OS/360, DEC RSX і VMS, а нині використовується досить рідко – це структурований файл. В цьому випадку підтримка структури файлу доручається файловій системі. Файлова система бачить файл як впорядковану послідовність логічних записів. Додаток може звертатися до ФС із запитами на ввід-вивід на рівні записів, наприклад "прочитати запис 25 з файлу FILE.DOC". ФС повинна мати інформацію про структуру файлу, достатню для того, щоб виділити будь-який запис. ФС надає додатку доступ до запису, а уся подальша обробка даних, що містяться в цьому записі, виконується додатком. Розвитком цього підходу стали системи управління базами даних (СУБД), які підтримують не лише складну структуру даних, але і взаємозв’язки між ними.

4. Фізична організація файлу

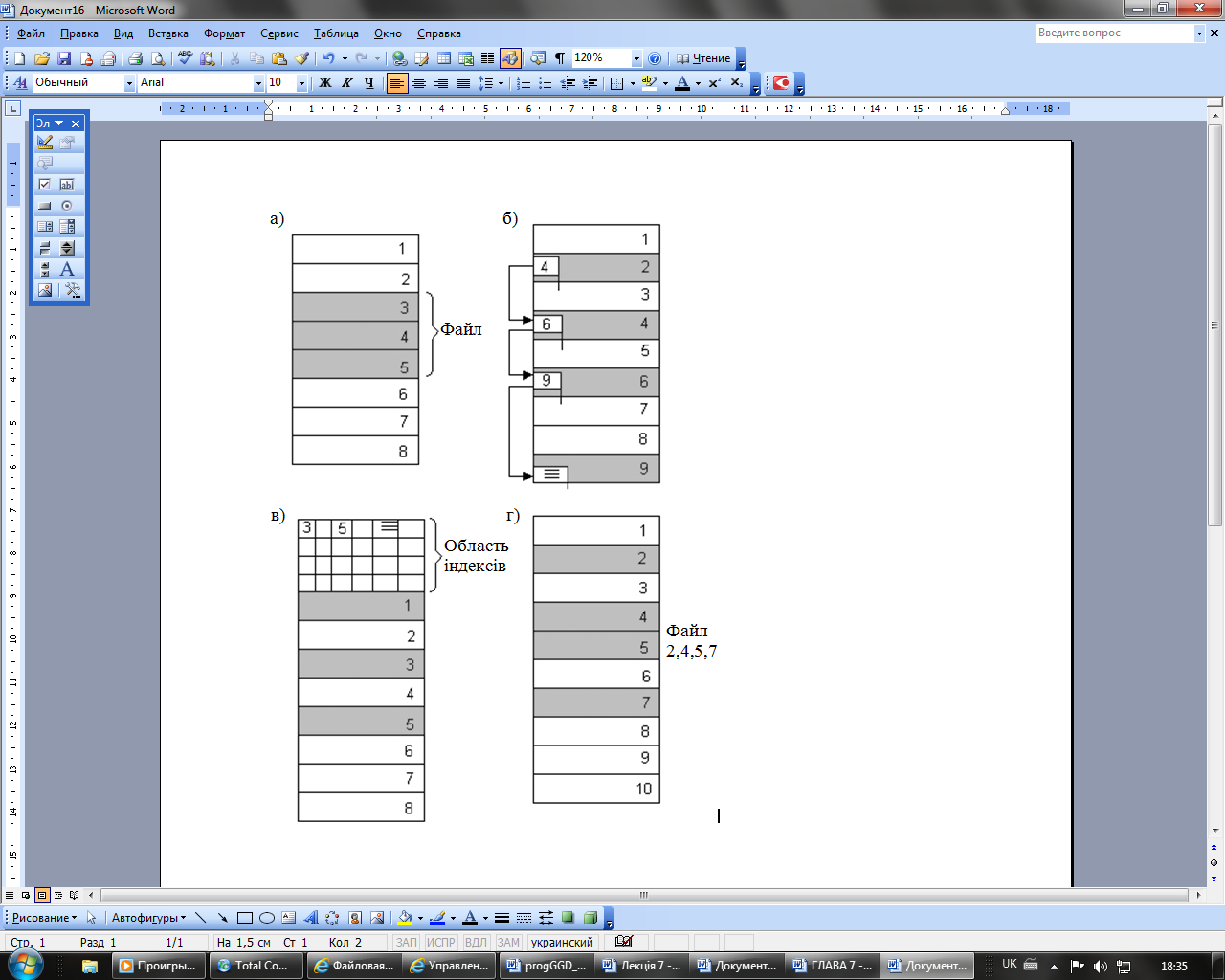
Фізична організація файлу описує правила розташування файлу на пристрої зовнішньої пам’яті, зокрема на диску. Файл складається з фізичних записів - блоків. Блок - найменша одиниця даних, якою зовнішній пристрій обмінюється з оперативною пам’яттю. Безперервне розміщення - простий варіант фізичної організації (рис. 2, а), при якому файлу надається послідовність блоків диска, що утворюють єдину суцільну ділянку дискової пам’яті. Для завдання адреси файлу в цьому випадку досить вказати тільки номер початкового блоку. Інша перевага цього методу - простота. Але є і два істотні недоліки.

Рис. 2. Фізична організація файлу: а - безперервне розміщення; б - зв’язаний список блоків; в - зв’язаний список індексів; г - перелік номерів блоків

По-перше, під час створення файлу заздалегідь не відома його довжина, а значить не відомо, скільки пам’яті потрібно зарезервувати для цього файлу, по-друге, при такому порядку розміщення неминуче виникає фрагментація, і простір на диску використовується не ефективно, оскільки окремі ділянки малого розміру (мінімально 1 блок) можуть залишитися не використаними.

Наступний спосіб фізичної організації – розміщення у вигляді зв’язаного списку блоків дискової пам’яті (рис. 2, б). При такому способі на початку кожного блоку знаходиться покажчик на наступний блок. В цьому випадку адреса файлу також може бути задана одним числом – номером першого блоку. На відміну від попереднього способу, кожен блок може бути приєднаний в ланцюжок блоків якого-небудь файлу, отже фрагментація відсутня. Файл може змінюватися під час свого існування, нарощуючи число блоків. Недоліком є складність реалізації доступу до довільно заданого місця файлу: для того, щоб прочитати п’ятий по порядку блок файлу, необхідно послідовно прочитати чотири перші блоки, простежуючи ланцюжок номерів блоків.

Популярним способом, що використовується, наприклад, у файловій системі FAT, являється використання зв’язаного списку індексів (рис. 2, в). З кожним блоком зв’язується деякий елемент – індекс. Індекси розташовуються в окремій області диска (таблиця FAT). Якщо деякий блок призначений деякому файлу, то індекс цього блоку містить номер наступного блоку цього файлу. При такій фізичній організації зберігаються усі переваги попереднього способу: мінімальність адресної інформації, відсутність фрагментації, відсутність проблем при зміні розміру. Крім того, цей спосіб має перевагу – для доступу до довільного місця файлу досить прочитати тільки блок індексів, відлічити потрібну кількість блоків файлу по ланцюжку і визначити номер потрібного блоку.

Ще один спосіб фізичного розташування файлу полягає в простому перерахуванні номерів блоків, що займає цей файл (рис. 1, г). Цей перелік і служить адресою файлу. Недолік цього способу очевидний: довжина адреси залежить від розміру файлу і для великого файлу може скласти значну величину. Перевагою ж є висока швидкість доступу до довільного блоку файлу, оскільки тут застосовується пряма адресація, яка виключає перегляд ланцюжка покажчиків при пошуку адреси довільного блоку файлу. Фрагментація на рівні блоків в цьому способі також відсутня. ОС UNIX використовує варіант цього способу, що дозволяє забезпечити фіксовану довжину адреси, незалежно від розміру файлу.

5. Загальна модель файлової системи

Функціонування будь-якої файлової системи можна представити багаторівневою моделлю (рис. 3), в якій кожен рівень надає деякий інтерфейс (набір функцій) вище розміщеному рівню, а сам, у свою чергу, для виконання своєї роботи використовує інтерфейс (поводиться з набором запитів) нижчого рівня.

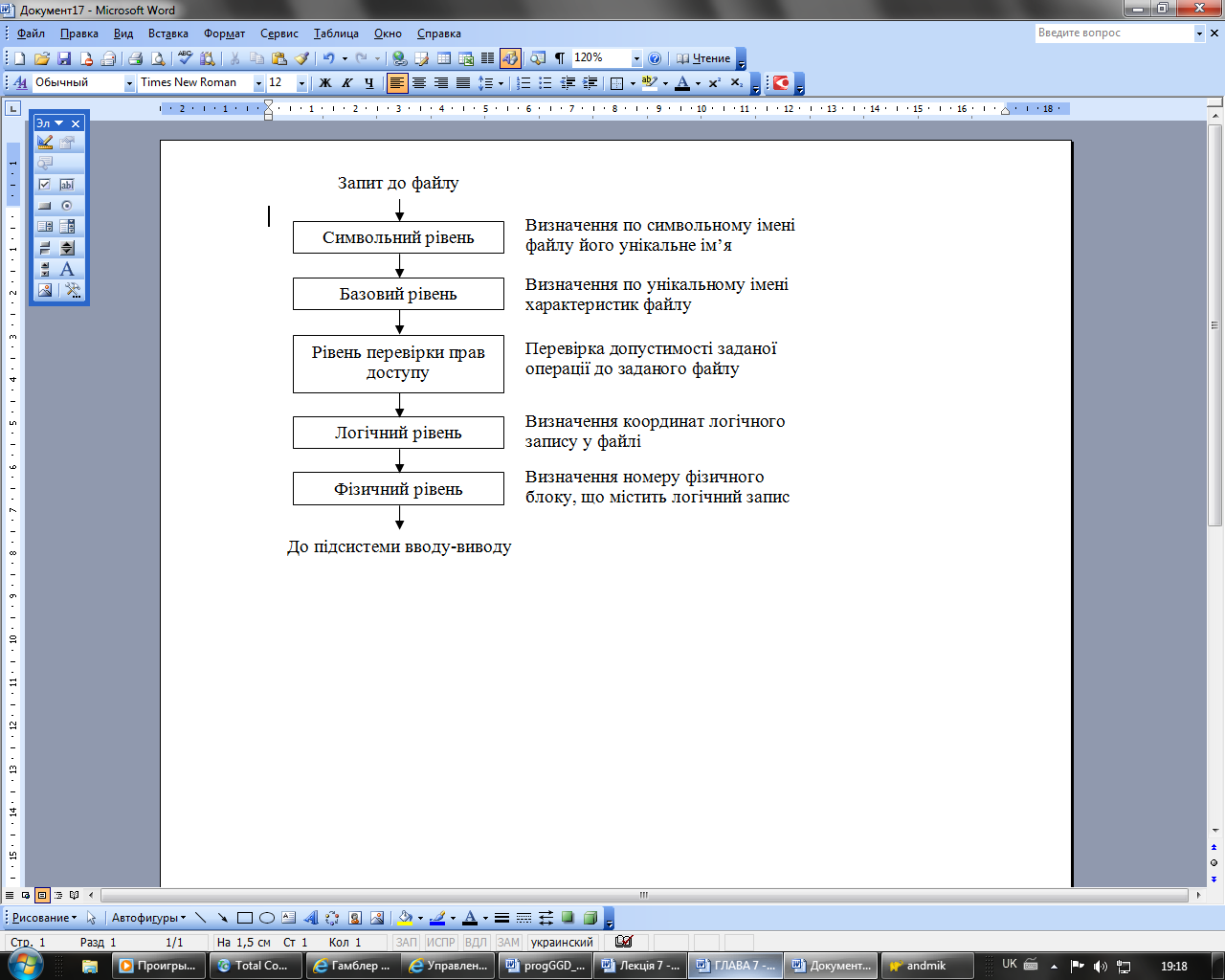


Рис. 3. Загальна модель файлової системи

Завданням символьного рівня є визначення по символьному імені файлу його унікального імені. У файлових системах, в яких кожен файл може мати тільки одне символьне ім’я, цей рівень відсутній, оскільки символьне ім’я, присвоєне файлу користувачем, є одночасно унікальним і може бути використано операційною системою. У інших файлових системах, в яких один і той же файл може мати декілька символьних імен, на цьому рівні проглядається ланцюжок каталогів для визначення унікального імені файлу. У файловій системі UNIX, наприклад, унікальним ім’ям є номер індексного дескриптора файлу.

На наступному, базовому рівні по унікальному імені файлу визначаються його характеристики: права доступу, адреса, розмір і інші. Характеристики файлу можуть входити до складу каталогу або зберігатися в окремих таблицях. При відкритті файлу його характеристики переміщаються з диска в оперативну пам’ять, щоб зменшити середній час доступу до файлу. У деяких файлових системах (наприклад, HPFS) при відкритті файлу разом з його характеристиками в оперативну пам’ять переміщаються декілька перших блоків файлу, що містять дані.

У різних файлових системах для характеристики файлів можуть використовуватися різні набори атрибутів. **Атрибути** – це інформація, що описує властивості файлу. Приклади можливих атрибутів файлу :

* тип файлу (звичайний файл, каталог, спеціальний файл);
* власник файлу;
* творець файлу;
* пароль для доступу до файлу;
* інформація про дозволені операції доступу до файлу;
* часи створення, останнього доступу і останньої зміни;
* поточний розмір файлу;
* максимальний розмір файлу;
* ознака "тільки для читання";
* ознака "прихований файл";
* ознака "системний файл";
* ознака "архівний файл";
* ознака "двійкова/символьна";
* ознака "тимчасова" (видалити після завершення процесу);
* ознака блокування;
* довжина запису у файлі;
* покажчик на ключове поле в записі;
* довжина ключа.

Наступним етапом реалізації запиту до файлу є перевірка прав доступу до нього. Для цього порівнюються повноваження користувача або процесу, що видали запит, із списком дозволених видів доступу до цього файлу. Якщо запрошуваний вид доступу дозволений, то виконання запиту триває, якщо ні, то видається повідомлення про порушення прав доступу.

У різних файлових системах може бути визначений свій список операцій доступу. Цей список може включати наступні операції:

* створення файлу,
* знищення файлу,
* відкриття файлу,
* закриття файлу,
* читання файлу,
* запис у файл,
* доповнення файлу,
* пошук у файлі,
* отримання атрибутів файлу,
* встановлення нових значень атрибутів,
* перейменування,
* виконання файлу,
* читання каталогу,
* і інші операції з файлами і каталогами.

На логічному рівні визначаються координати запрошуваного логічного запису у файлі, тобто вимагається визначити, на якій відстані (у байтах) від початку файлу знаходиться необхідний логічний запис. При цьому абстрагуються від фізичного розташування файлу, він представляється у вигляді безперервної послідовності байт. Алгоритм роботи цього рівня залежить від логічної організації файлу. Напр., якщо файл організований як послідовність логічних записів фіксованої довжини l, то n-й логічний запис має зміщення l(n-1) байт. Для визначення координат логічного запису у файлі з індексно-послідовною організацією виконується читання таблиці індексів (ключів), в якій безпосередньо вказується адреса логічного запису.

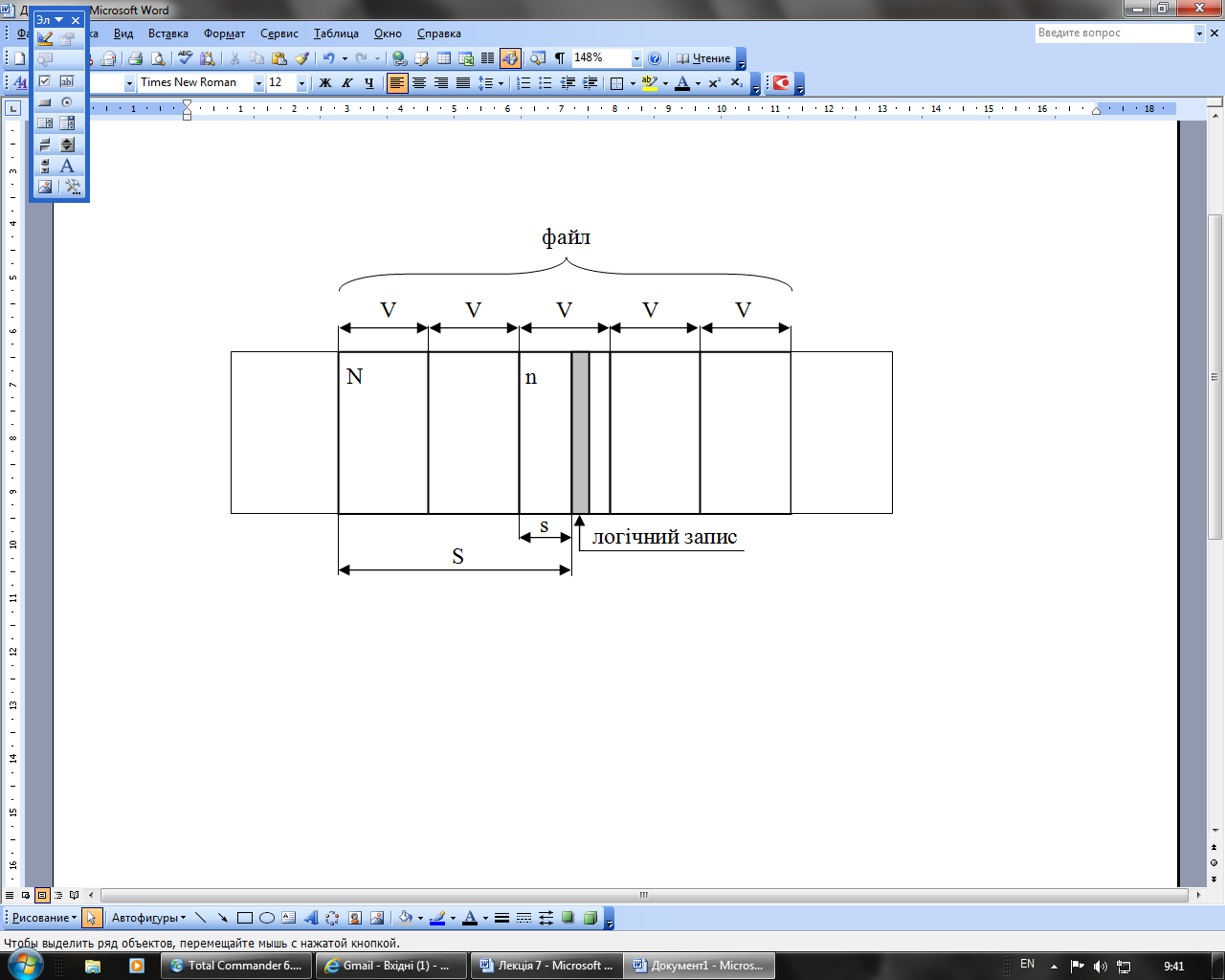
На фізичному рівні файлова система визначає номер фізичного блоку (n), який містить необхідний логічний запис, і зміщення логічного запису (s) у фізичному блоці. Для вирішення цього завдання використовуються результати роботи логічного рівня - зміщення логічного запису у файлі, адреса файлу на зовнішньому пристрої, а також відомості про фізичну організацію файлу, включаючи розмір блоку. Рис. 4 ілюструє роботу фізичного рівня для простої фізичної організації файлу у вигляді безперервної послідовності блоків. Завдання фізичного рівня вирішується незалежно від того, як був логічно організований файл.

Рис. 4. Функції фізичного рівня файлової системи

*Початкові дані:* V - розмір блоку;

N - номер першого блоку файлу;

S - зміщення логічного запису у файлі

*Вимагається визначити на фізичному рівні:*

n - номер блоку, що містить необхідний логічний запис

s - зміщення логічного запису в межах блоку

n = N + [S/V], де [S/V] - ціла частина числа S/V

s = R [S/V] - дробова частина числа S/V

Після визначення номера фізичного блоку, файлова система звертається до системи вводу-виводу для виконання операції обміну із зовнішнім пристроєм. У відповідь на цей запит в буфер файлової системи буде переданий потрібний блок, в якому на підставі отриманого при роботі фізичного рівня зміщення вибирається необхідний логічний запис.

6. Модель мережевої файлової системи

Мережева файлова система в загальному випадку включає наступні елементи (рис. 5):

* локальна файлова система;
* інтерфейс локальної файлової системи;
* сервер мережевої файлової системи;
* клієнт мережевої файлової системи;
* інтерфейс мережевої файлової системи;
* протокол клієнт-сервер мережевої файлової системи.

Клієнти мережевої ФС – це програми, які працюють на численних комп’ютерах, підключених до мережі. Ці програми обслуговують запити додатків на доступ до файлів, що зберігаються на видаленому комп’ютері.

Клієнт мережевий ФС передає по мережі запити іншому програмному компоненту - серверу мережевої ФС, що працює на віддаленому комп’ютері. Сервер, отримавши запит, може виконати його або самостійно, або, що є поширенішим варіантом, передати запит локальній файловій системі для обробки. Після отримання відповіді від локальної файлової системи сервер передає його по мережі клієнтові, а той, у свою чергу, - додатку, що звернувся із запитом.

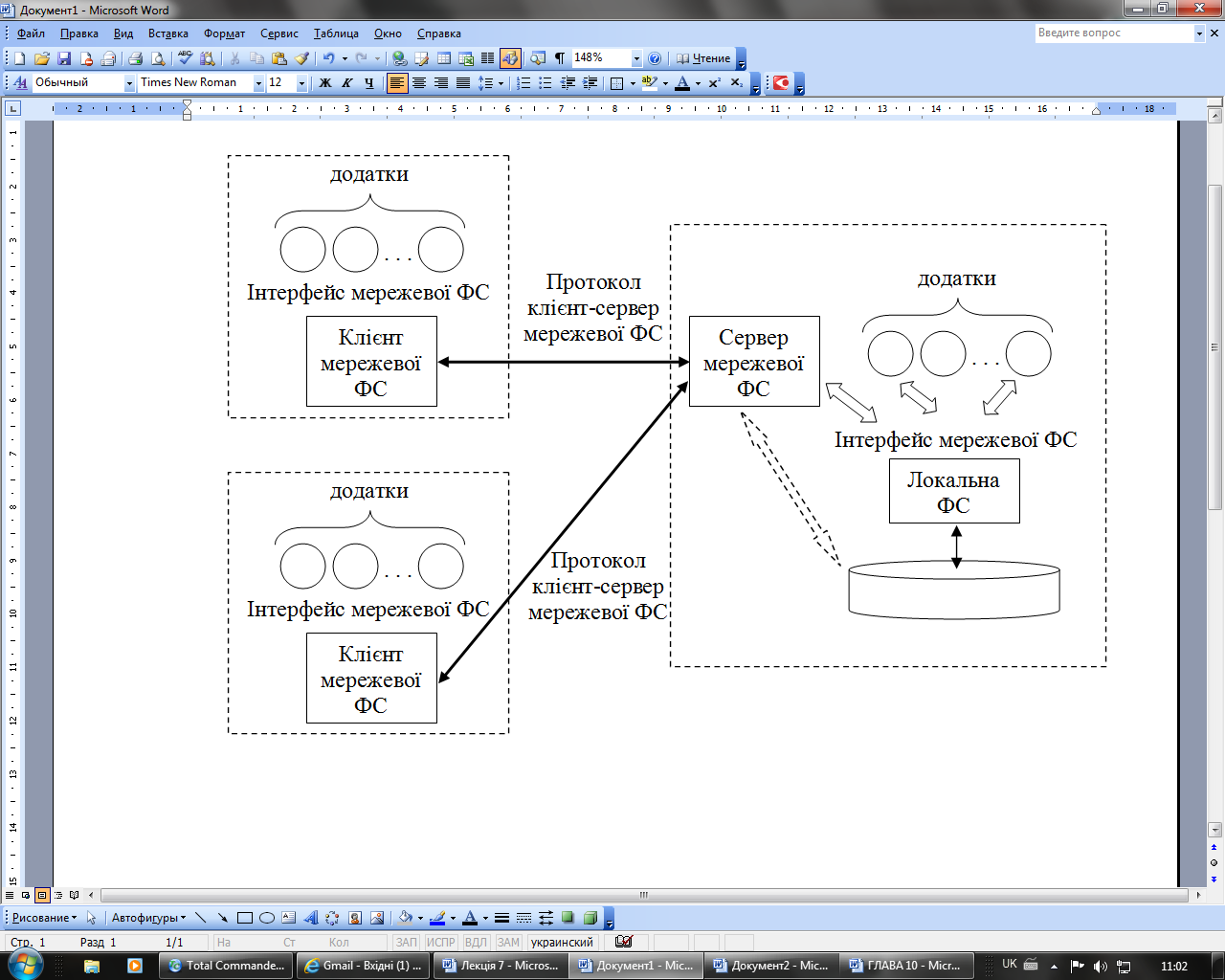
Додатки звертаються до клієнта мережевої ФС, використовуючи певний програмний інтерфейс, який в даному випадку є інтерфейсом мережевої файлової системи. Цей інтерфейс прагнуть зробити більш схожим на інтерфейс локальної файлової системи, щоб дотримати принцип прозорості. При повному збігу інтерфейсів додаток може звертатися до локальних і віддалених файлів і каталогів за допомогою одних і тих же системних викликів, абсолютно не зважаючи на місця зберігання даних.

Рис. 5. Модель мережевої файлової системи

Клієнт і сервер мережевої файлової системи взаємодіють один з одним по мережі по певному протоколу. У разі збігу інтерфейсів локальної і мережевої файлових систем цей протокол може бути досить простим - в його функції входитиме ретрансляція серверу запитів, прийнятих клієнтом від додатків, з якими той потім звертатиметься до локальної файлової системи. Одним з механізмів, що може бути використаний для цієї мети, може бути механізм **виклику віддалених процедур RPC (Remote Procedure Call).**

За досить довгий термін розвитку мереж в них затвердилося декілька мережевих файлових систем. У локальних мережах упродовж багатьох років домінувала мережева ОС NetWare, яка використовувала на файлових серверах локальну ФС NetWare і протокол **NCP (NetWare Control Protocol)**.

Іншим популярним типом мережевих файлових систем стали системи компаній Microsoft і IBM, які були спочатку розроблені для локальних мереж на основі ПК під управлінням MS-DOS і Windows 3.x. Загальним для цих мережевих файлових систем стало використання локальної ФС FAT та протоколу **SMB (Server Message Block)**. Пізніше протокол SMB був застосований і для доступу до локальних файлових систем HPFS і NTFS.

У середовищі операційної системи UNIX найбільшого поширення набула мережева файлова система **NFS (Network File System)**. Вона розроблялася для доступу до локальної файлової системи s5/ufs, що є основною для більшості ОС сімейства UNIX. Файлові системи s5 (від System V, імені декількох версій ОС UNIX, розроблених AT&T) і ufs (UNIX File System) використовують дуже близьку фізичну модель. ФС ufs є розвитком системи s5 і розширює її можливості по підтримці великих дисків і файлів, а також підвищує її надійність.

З часом у великих мережах стали одночасно застосовуватися декілька мережевих файлових систем різних типів, наприклад NetWare, SMB і NFS. Це часто відбувалося при об’єднанні декількох мереж в одну. В результаті в мережі з’явилися різні файлові сервери, що підтримували різні локальні файлові системи і протоколи клієнт-сервер, а також клієнти, що забезпечували додаткам і користувачам різні інтерфейси. Виникла проблема - як забезпечити доступ клієнта будь-якого типу до файлового сервера будь-якого типу? Існує декілька варіантів вирішення цієї проблеми, заснованих на комбінуванні локальних файлових систем, протоколів клієнт-сервер і інтерфейсів, що підтримуються клієнтами файлової системи.

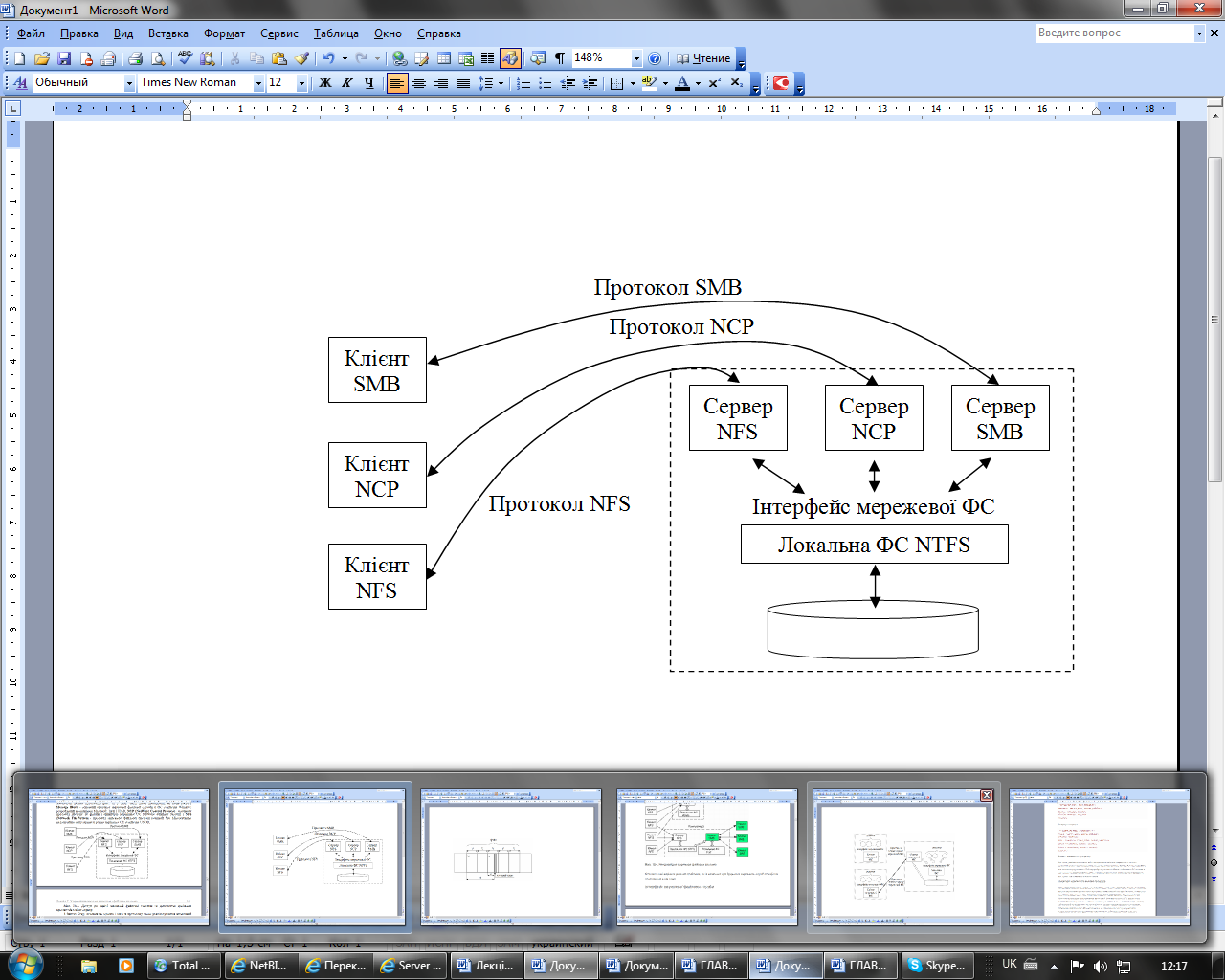
Для однієї і тієї ж локальної файлової системи можуть існувати різні протоколи мережевої файлової системи. Так, до файлової системи NTFS можна дістати доступ за допомогою різних протоколів (рис. 6), у тому числі таких поширених, як SMB, NCP і NFS.

Рис. 6. Доступ до однієї локальної ФС за допомогою декількох протоколів клієнт-сервер

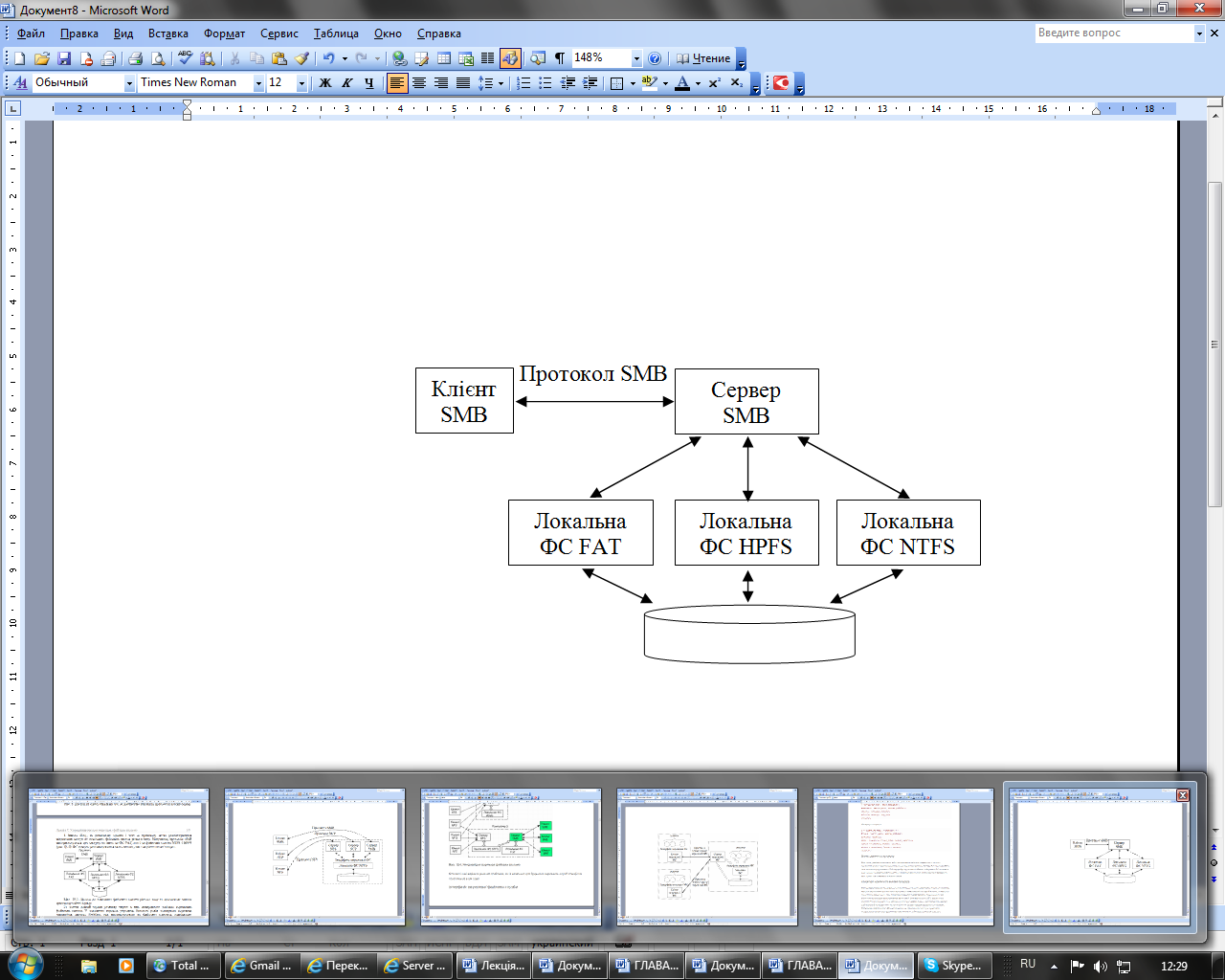
З іншого боку, за допомогою одного і того ж протоколу може реалізовуватися віддалений доступ до локальних файлових систем різного типу. Наприклад, протокол SMB використовується для доступу не лише до ФС FAT, але і до файлових систем NTFS і HPFS (рис. 7). Ці ФС можуть розташовуватися як на одному, так і на різних комп’ютерах.

Рис. 7. Доступ до локальних ФС різного типу за допомогою одного протоколу клієнт-сервер

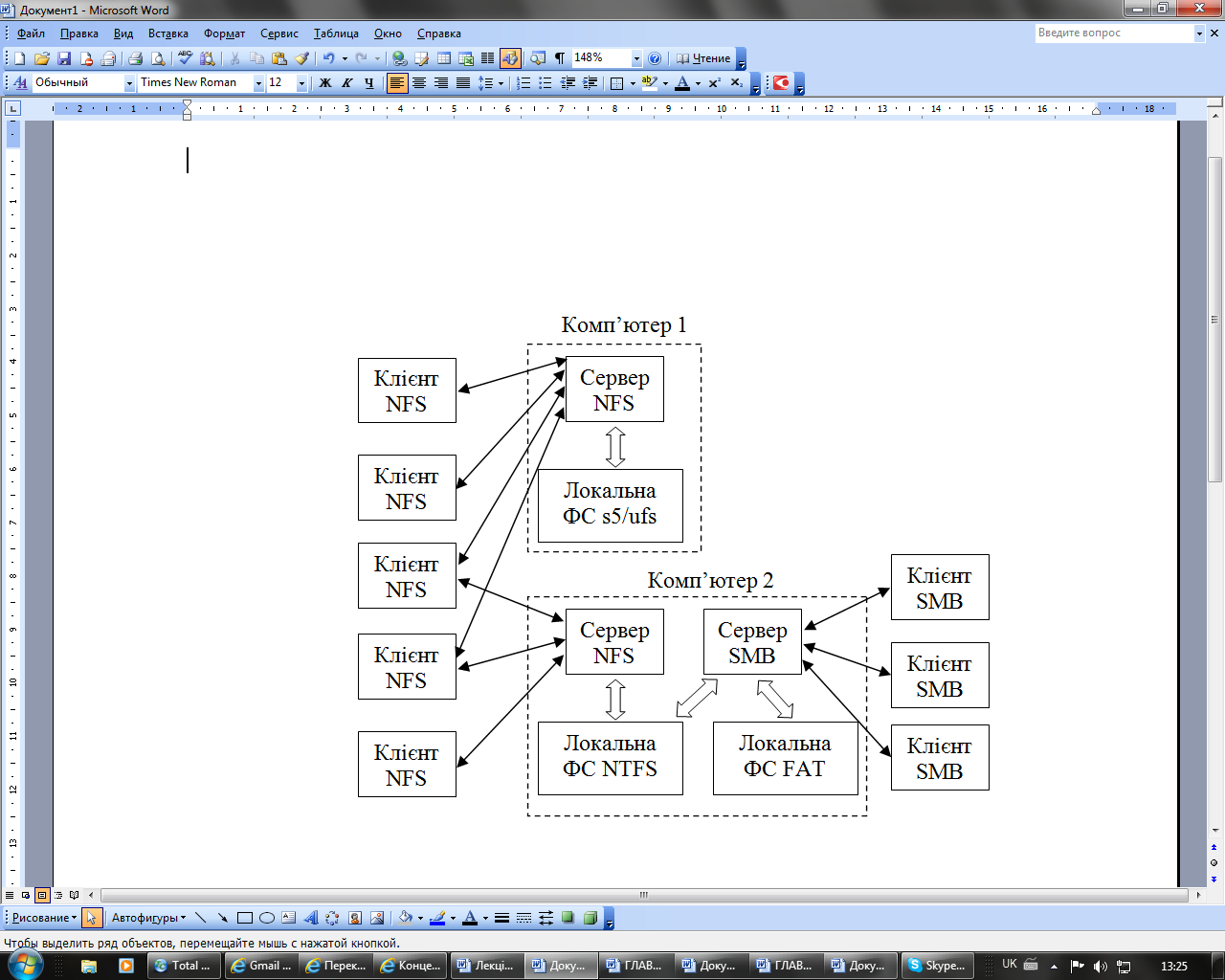
На рис. 8 показаний варіант організації неоднорідної мережевої файлової системи.

Рис. 8. Неоднорідна мережева файлова система

Лекція 8

Взаємодія операційних систем в гетерогенних мережах

1. Основні підходи до реалізації взаємодії мереж

Тільки невелика кількість мереж володіє **однорідністю (гомогенністю)** програмного і апаратного забезпечення. Однорідними частіше являються мережі, які складаються з невеликої кількості компонентів від одного виробника. Нормою сьогоднішнього дня є **неоднорідні (гетерогенні)** мережі, що складаються з різнотипних робочих станцій, операційних систем і додатків, в яких для реалізації взаємодії між комп’ютерами використовується різне апаратне та програмне забезпечення.

Засоби взаємодії комп’ютерів в мережі організовані у вигляді багаторівневої структури – стека протоколів. У однорідній мережі усі комп’ютери використовують один і той же стек. Проблема виникає тоді, коли необхідно організувати взаємодію комп’ютерів, на яких підтримуються різні стеки комунікаційних протоколів.

Проблема міжмережевої взаємодії може мати різні зовнішні прояви, але суть її одна – неспівпадання використовуваних комунікаційних протоколів. Наприклад, ця проблема може виникнути в мережі, в якій є лише одна мережева ОС, проте транспортна підсистема неоднорідна через те, що мережа включає фрагменти Ethernet, що об’єднані кільцем FDDI. Тут взаємодіючими мережами виступають групи комп’ютерів, що працюють на різних протоколах канального і фізичного рівня, наприклад мережа Ethernet, мережа FDDI.

Так само проблема міжмережевої взаємодії може виникнути в мережі, побудованій виключно на основі технології Ethernet, але в якій встановлено декілька різних мережевих ОС. Наприклад, частина комп’ютерів мережі може працювати під управлінням ОС Window, які використовують для доступу до файлів протокол SMB, а частина – під управлінням ОС NetWare, які використовують протокол доступу до файлів NCP. В цьому випадку взаємодіючими мережами виступають групи комп’ютерів, що працюють на різних протоколах прикладного рівня.

Завдання усунення неоднорідності мають деяку специфіку і навіть різні назви залежно від того, до якого рівня OSI моделі вони відносяться. Завдання об’єднання транспортних підсистем, що відповідають тільки за передачу повідомлень, зазвичай називається **internetworking** – утворення інтермереж. Класичним підходом для її вирішення є використання єдиного мережевого протоколу, такого, наприклад, як IP або IPX.

Інше завдання, зване **interoperability** – узгодження мережевих служб ОС, виникає при об’єднанні мереж, що використовують різні протоколи вищих рівнів, в основному прикладного і відображення.

Кардинальним рішенням проблеми міжмережевої взаємодії могло б стати використання єдиного стека протоколів в усіх мережах. І така спроба введення єдиного стека комунікаційних протоколів була зроблена в 1990 році урядом США, який оприлюднив програму GOSIP (Government OSI Profile), відповідно до якої стек протоколів OSI передбачалося зробити загальним для усіх мереж, що встановлюються в урядових організаціях США. Проте масового переходу на стек OSI не відбулося. Тим часом у зв’язку з швидким зростанням популярності Інтернету стандартом де-факто став стек протоколів TCP/IP. Але як і раніше, багато мережевих вузлів підтримують інші стеки протоколів, такі як IPX/SPX, DECnet, IBM SNA, NetBEUI, так що до єдиного стека протоколів ще далеко.

Для організації взаємодії різних мереж використовується два підходи: трансляція і мультиплексування (рис. 1).

Перший підхід пов’язаний з використанням шлюзів, які забезпечують узгодження двох стеків протоколів шляхом перетворення (трансляції) протоколів (рис. 1, а). Шлюз розміщується між взаємодіючими мережами і служить посередником, що переводить повідомлення, які поступають від однієї мережі, у формат іншої мережі.

Другий підхід полягає в тому, що в операційні системи серверів і робочих станцій вбудовуються декілька найбільш популярних стеків протоколів. За рахунок використання даної технології або клієнтські запити використовують стек протоколів тієї мережі, до якої відносяться потрібні сервери, або сервери підключають відповідний стек протоколів, згідно клієнтського запиту (рис. 1, б).

Взаємодія комп’ютерів, що належать різним мережам, нагадує спілкування людей, що говорять на різних мовах. Для досягнення взаєморозуміння вони також можуть використовувати два підходи: запросити перекладача (аналог шлюзу), або перейти на мову співрозмовника, якщо вони нею володіють (аналог мультиплексування стеків протоколів).

2. Трансляція

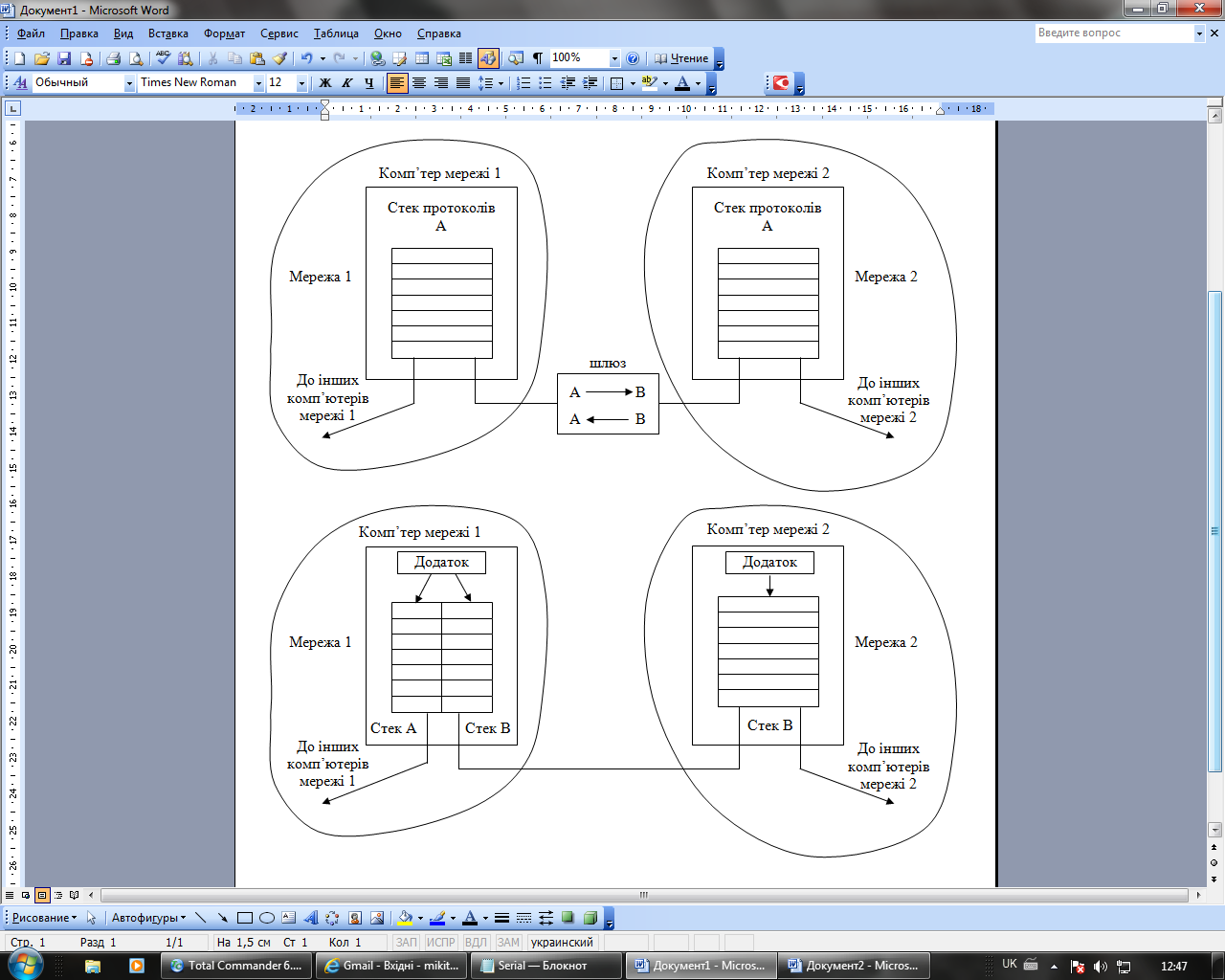
Трансляція забезпечує узгодження стеків протоколів шляхом перетворення повідомлень, що поступають від однієї мережі у формат повідомлень іншої мережі. Транслюючий елемент, в якості якого можуть виступати, наприклад, програмний або апаратний шлюз, міст, комутатор або маршрутизатор, розміщується між взаємодіючими мережами і служить посередником в їх "діалозі". Отже, шлюз погоджує комунікаційні протоколи одного стека з комунікаційними протоколами іншого стека.

Рис. 1 Варіанти узгодження протоколів: а - трансляція протоколів;

б - мультиплексування стеків протоколів

На рис. 2 показаний шлюз, розміщений на комп’ютері 2, який погоджує протоколи клієнтського комп’ютера 1 в мережі А з протоколами комп’ютера 3 в мережі В. У шлюзі встановлено обидва стеки протоколів.

Запит від прикладного процесу клієнтського комп’ютера мережі А поступає на прикладний рівень його стека протоколів. Відповідно до цього протоколу на прикладному рівні формується пакет (чи декілька пакетів), в якому передається запит на виконання послуг деякому серверу мережі В. Пакет прикладного рівня переміщається вниз по стеку протоколів комп’ютера 1, доповнюючись заголовками протоколів нижчих рівнів, а потім передається по лініях зв’язку на комп’ютер 2, тобто на шлюз.

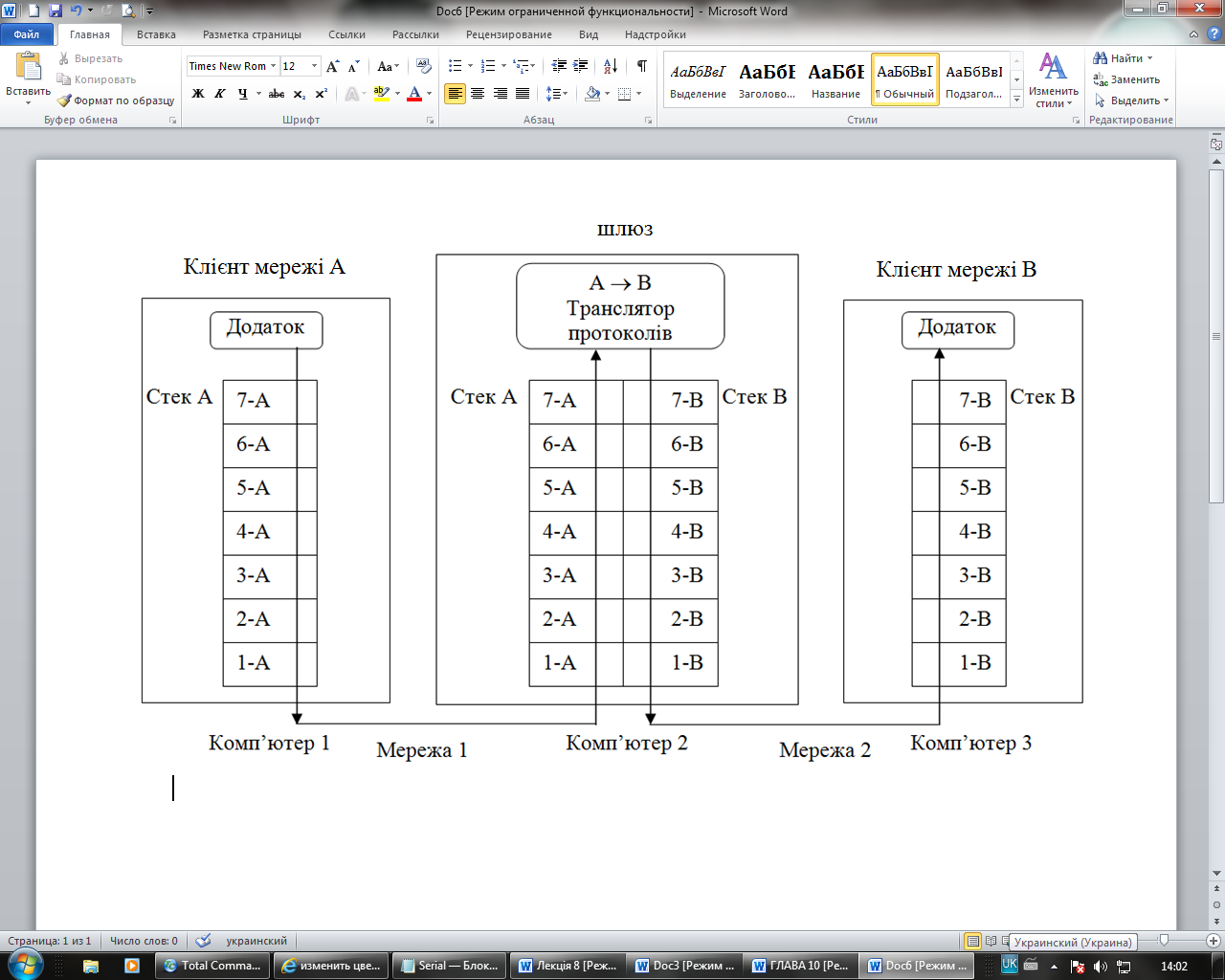
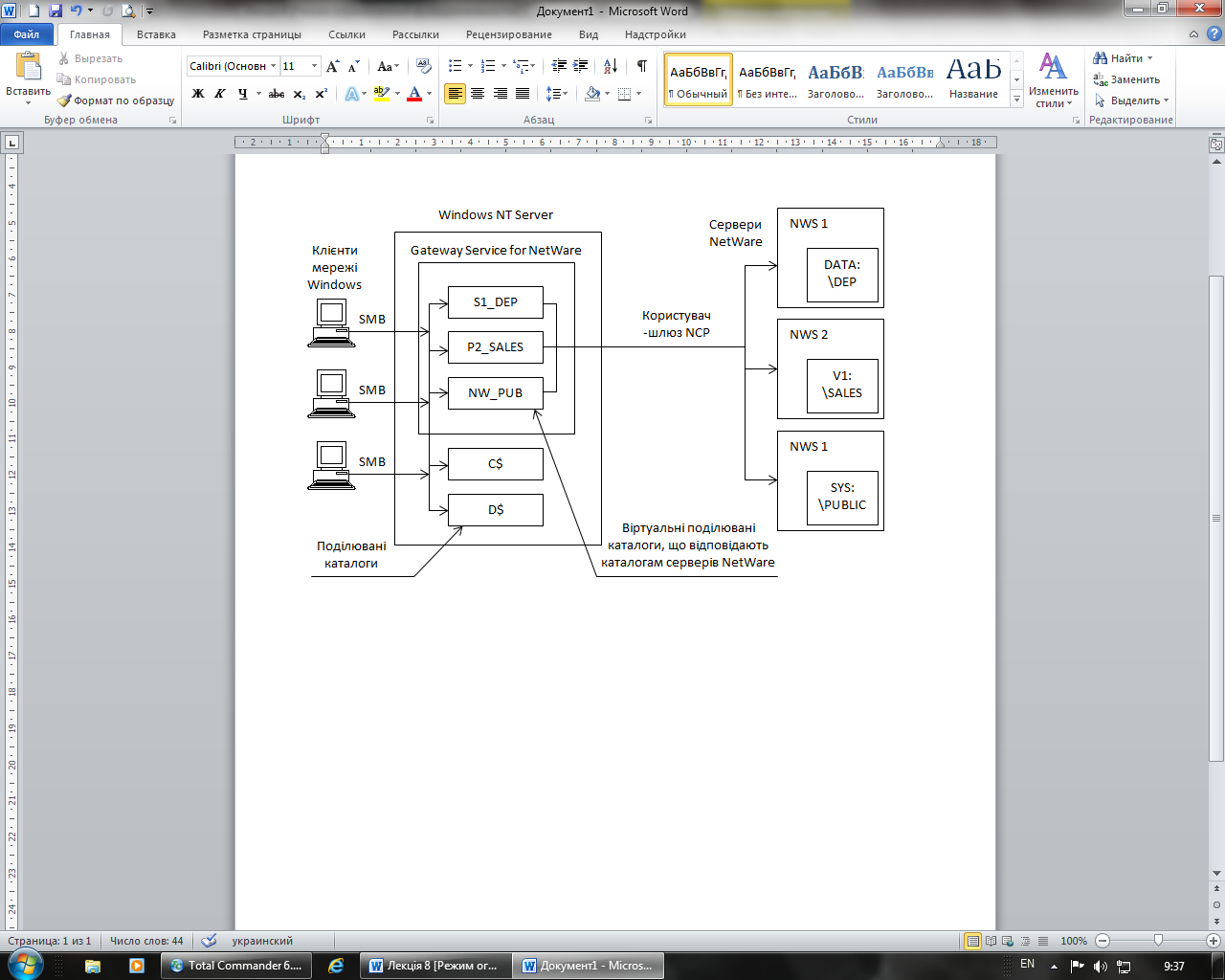


Рис. 2. Принцип функціонування шлюзу

На шлюзі обробка вхідних даних йде в зворотному порядку, від протоколу самого нижнього до протоколу самого верхнього рівня стеку А. Потім пакет прикладного рівня стеку мережі А транслюється в пакет прикладного рівня стеку мережі В. Алгоритм перетворення пакетів залежить від конкретних протоколів і може бути досить складним. Перетворений пакет від верхнього рівня стеку мережі В передається до нижніх рівнів відповідно до правил цього стека, а потім по фізичних лініях зв’язку відповідно до протоколів фізичного і канального рівнів мережі В поступає в іншу мережу до потрібного сервера. Відповідь сервера перетвориться шлюзом аналогічно.

Прикладом шлюзу, що транслює протоколи прикладного рівня, являється компонент Windows NT Gateway Services for NetWare (GSNW), який забезпечує клієнтам Windows NT прозорий доступ до томів і каталогів серверів NetWare (рис. 3).

Рис. 3. Шлюз Gateway Services for NetWare

Шлюз встановлюється на тому ж комп’ютері, на якому встановлений сервер Windows NT. Між шлюзом і сервером NetWare встановлюється зв’язок по протоколу NCP. Для доступу до файлів NetWare клієнти Windows NT, використовуючи свій "рідний" протокол SMB, звертаються до сервера Windows NT, на якому працює шлюз GSNW. Шлюз віртуалізує поділювані каталоги серверів NetWare: вони виглядають для клієнтів SMB точно так, як і поділювані каталоги сервера Windows NT. Якщо запит, що поступив на сервер Windows NT, відноситься до ресурсів серверів NetWare, то він переадресується шлюзу, який транслює його в зрозумілий для сервера NetWare вид і передає по протоколу NCP відповідному серверу. При цьому шлюз виступає по відношенню до серверів NetWare як клієнт-користувач.

Перевага шлюзів полягає в тому, що вони зберігають в незмінному вигляді програмне забезпечення на клієнтських комп’ютерах. Користувачі працюють в звичному середовищі і можуть навіть не помітити, що вони дістають доступ до ресурсів іншої мережі. Проте, як і всякий централізований ресурс, шлюз знижує надійність мережі. Крім того, при обробці запитів в шлюзі можливі відносно великі тимчасові затримки, по-перше, через витрати часу на процедуру трансляції, а по-друге, через затримки запитів в черзі до шлюзу, що розділяється усіма клієнтами. Це робить шлюз погано масштабованим рішенням. Правда, ніщо не заважає встановити в мережі декілька паралельно працюючих шлюзів.

3. Мультиплексування стеків протоколів

При мультиплексуванні стеків протоколів в мережеве обладнання або в операційні системи серверів і робочих станцій вбудовуються декілька стеків протоколів. Це дозволяє клієнтам і серверам вибирати для взаємодії той протокол, який є для них загальним.

На рис. 4 наведений приклад взаємодії клієнтського комп’ютера мережі В з сервером у своїй мережі і сервером мережі А, який працює із стеком протоколів, що повністю відрізняється від стека мережі В. В клієнтському комп’ютері реалізовано обидва стеки. Для того, щоб запит від прикладного процесу був правильно оброблений і спрямований через відповідний стек, потрібна наявність спеціального програмного елементу – **мультиплексора протоколів**, або **менеджера протоколів**. Менеджер повинен уміти визначати, до якої мережі спрямований запит клієнта. Для цього може використовуватися служба імен мережі, в якій відзначається приналежність того або іншого ресурсу певної мережі з відповідним стеком протоколів.

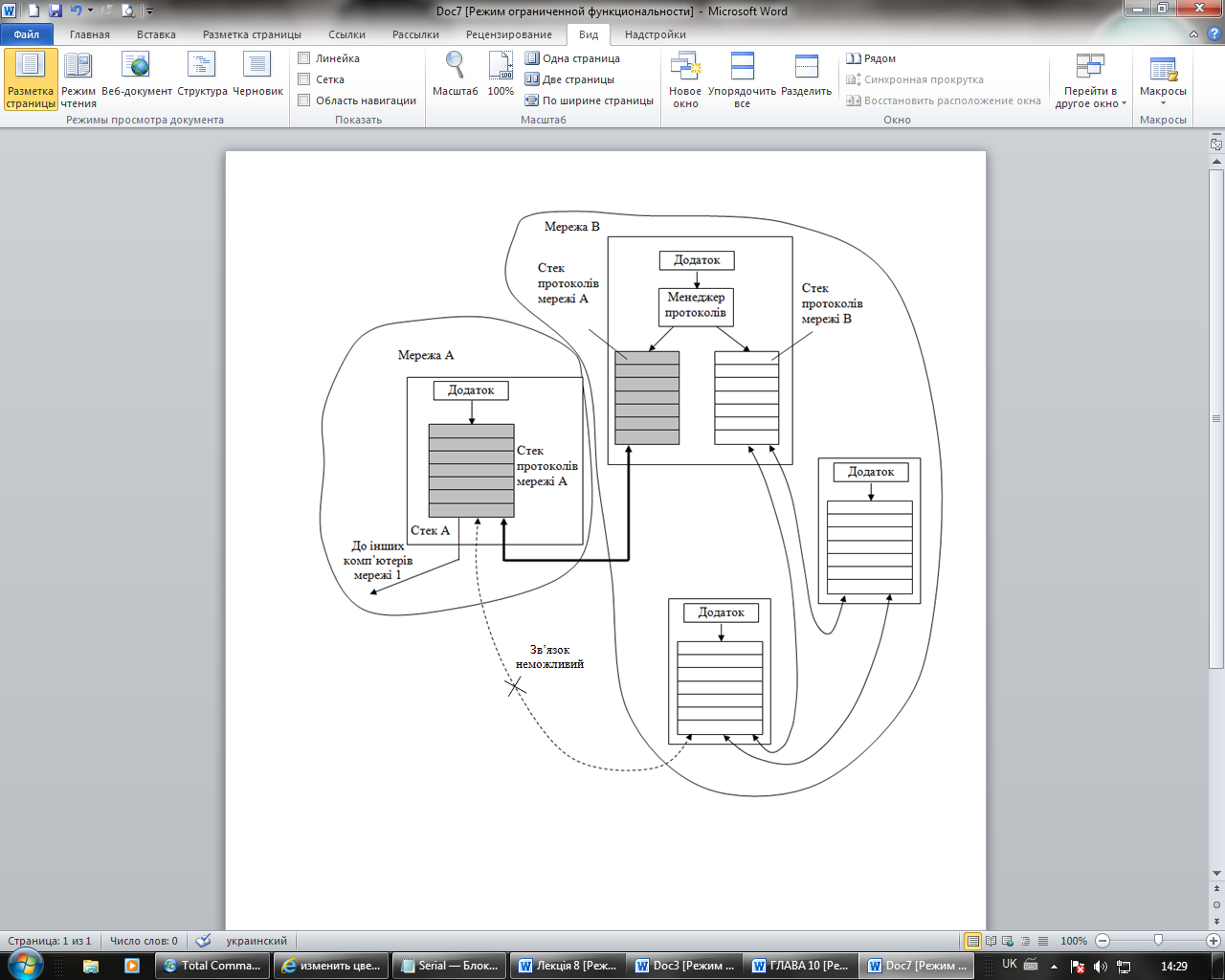


Рис. 4. Мультиплексування стеків

При використанні технології мультиплексування структура комунікаційних засобів операційної системи може бути і складнішою. У загальному випадку на кожному рівні замість одного протоколу з’являється цілий набір протоколів і може існувати декілька мультиплексорів, що виконують комутацію між протоколами різних рівнів.

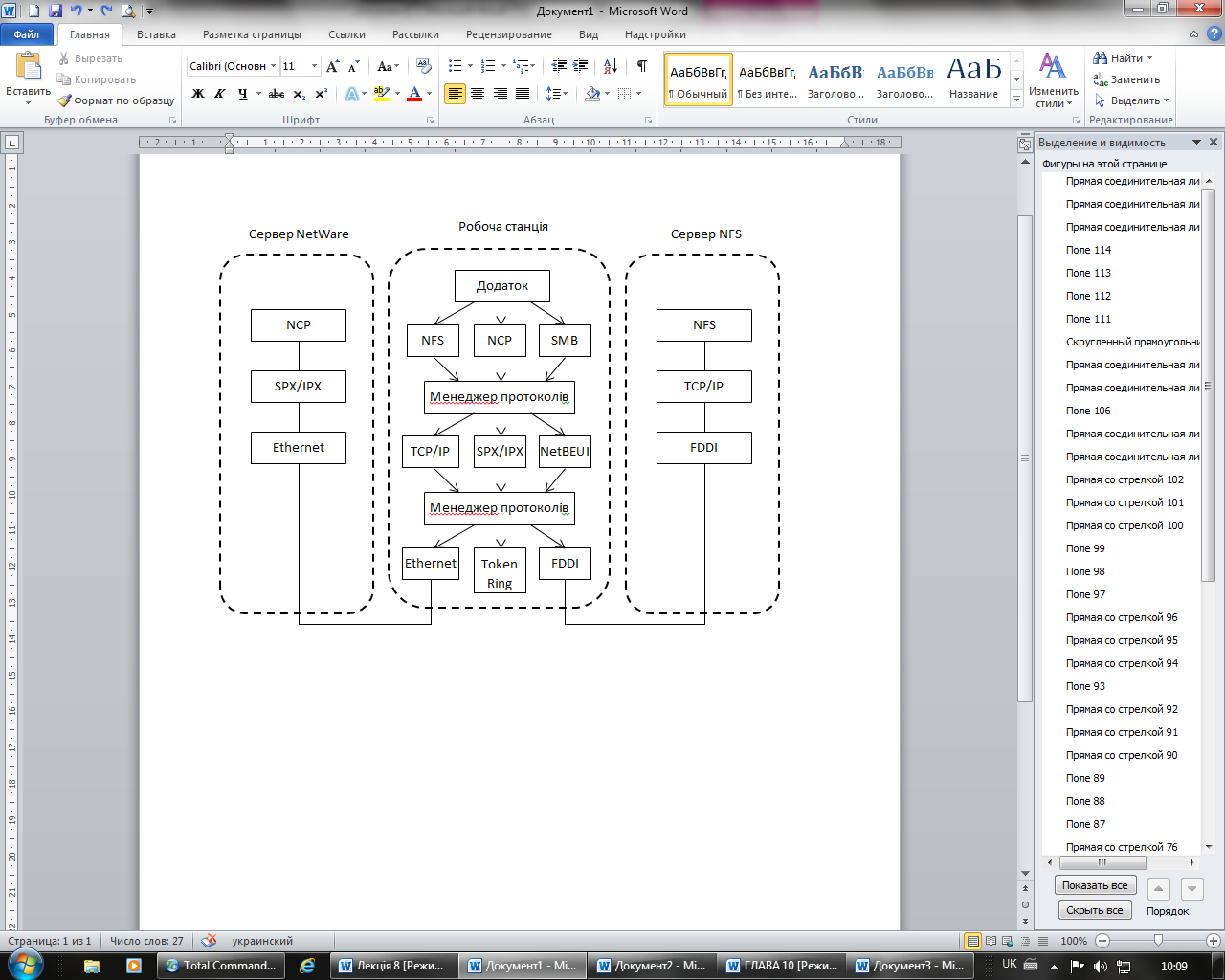
При використанні технології мультиплексування структура комунікаційних засобів операційної системи може бути і складнішою. У загальному випадку на кожному рівні замість одного протоколу з’являється цілий набір протоколів і може існувати декілька мультиплексорів, що виконують комутацію між протоколами різних рівнів. Наприклад, робоча станція, стек протоколів якої показаний на рис. 5, може через один мережевий адаптер дістати доступ до мереж, що працюють по протоколах NetBIOS, IP, IPX. Ця робоча станція може бути клієнтом відразу декількох файлових серверів: NetWare (NCP), Windows NT (SMB) і Sun (NFS).

Рис. 5. Мультиплексування протоколів

Передумовою для розвитку технології мультиплексування стеків протоколів стала поява чітких відкритих описів протоколів різних рівнів і міжрівневих інтерфейсів, так що виробник програмного чи апаратного забезпечення при реалізації "чужого" протоколу може бути впевнений, що її продукт коректно взаємодіятиме з продуктами інших фірм по цьому протоколу і з ним нормально взаємодіятимуть протоколи сусідніх рівнів.

Важливою перевагою мультиплексування є менший час виконання запиту, ніж при використанні шлюзу. Це пов’язано, по-перше, з відсутністю тимчасових витрат на процедуру трансляції, а по-друге, з тим, що при мультиплексуванні на кожен запит вимагається тільки одна мережева передача, тоді як при трансляції – дві: запит спочатку передається на шлюз, а потім з шлюзу на ресурсний сервер.

У таблиці 1 приведені порівняльні характеристики двох підходів до реалізації міжмережевої взаємодії.

Таблиця 1. Порівняння методів трансляції і мультиплексування протоколів

|  |  |  |
| --- | --- | --- |
| **Метод** | **Переваги** | **Недоліки** |
| Трансляція протоколів | Збереження звичного середовища користувачів; відсутність необхідності в додатковому програмному забезпеченні на робочих станціях; локалізація усіх проблем міжмережевої взаємодії; забезпечення можливості доступу до "чужих" ресурсів відразу для декількох клієнтів | Уповільнення роботи; зниження надійності; погана масштабованість; необхідність в двох мережевих передачах для виконання одного запиту |
| Мультиплексування протоколів | Швидший доступ; підвищення надійності взаємодії за рахунок установки стека на декількох вузлах мережі; добре масштабований засіб | Ускладнення адміністрування і контролю доступу; висока надмірність, що вимагає додаткових ресурсів від робочих станцій; менш зручно для користувача, ніж шлюзи |

**Список використаної літератури**

1. Бэкон Дж., Харрис Т. Операционные системы. Параллельные и распределенные системы. - СПб.: Питер, 2004. - 800 с.: ил.
2. Єфименко В.В., Оніщенко С.М., Франчук В.М. Операційні системи. Лабораторний практикум: Навчальний посібник. – К.: НПУ імені М.П. Драгоманова, 2008. – 124 с.
3. Олифер В. Г., Олифер Н. А. Сетевые операционные системы. — СПб.: Питер, 2008. – 528 с.
4. Буров Є. Комп’ютерні мережі. 2-ге оновлене і доповн. вид. Львів: БаК, 2003. – 584 с.
5. Головина О.С., Кондратьев В.К. Операционные системы и оболочки. М: МЭСИ. 2002. – 434 с.
6. Таненбаум. Э. Современные операционные системы. СПб.: Питер.2002.
7. Ричард Петерсен. LINUX: руководство по операционной системе. Второе издание, переработанное и дополненное в двух томах, перевод. – Киев: BHV, 1998, 1000 стр.
8. Андрей Лобачевский. Операционная система UNIX. BHV-Санкт-Петербург. 1997, 500 стр.
9. Стахнов А.А. Linux. – СПб.: БХВ-Петербург, 2004. – 912 с.
10. Шеховцов В.А. Операційні системи. — К.: Видавнича група ВНУ, 2005. — 576 c.
11. Таненбаум Е. Сучасні операційні системи (видання 2). — СПб.: Пітер, 2002.