3.2 СТРУКТУРА ПОБУДОВИ ОС. АРХІТЕКТУРА ЯДРА

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 з модифікації і розвитку системи. Навпаки, складну систему без хорошої структури частіше простіше розробити наново, чим модернізувати.

Функціональна складність операційної системи неминуче призводить до складності її архітектури, під якою розуміють структурну організацію ОС на основі різних програмних модулів. До складу ОС входять виконувані і об'єктні модулі стандартних для цієї ОС форматів, бібліотеки різних типів, програмні модулі спеціального формату (наприклад, завантажувач ОС, драйвери введення- виведення), файли документації, модулі довідкової системи тощо.

Більшість сучасних операційних систем є добре структурованими модульними системами, здатними до розвитку, розширення і перенесення на нові платформи. Якої-небудь єдиної архітектури ОС не існує, але існують універсальні підходи до структуризації ОС.

3.2.1 Ядро і допоміжні модулі ОС

Найзагальнішим підходом до структуризації операційної системи є розділення 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 введення-виведення тощо. Ядро складає серцевину операційної системи, без нього ОС є повністю непрацездатною і не зможе виконати жодної зі своїх функцій.

До складу ядра входять функції, які виконують внутрісистемні задачі організації обчислювального процесу, такі як перемикання контекстів процесів, завантаження/вивантаження сторінок, обробка переривань. Ці функції недоступні для додатків.

Інший клас функцій ядра служить для підтримки додатків, створюючи для них так зване прикладне програмне середовище. Додатки можуть звертатися до ядра із запитами (системними викликами) для виконання тих або інших дій, наприклад для відкриття і читання файлу, виведення графічної інформації на дисплей, отримання системного часу тощо. Функції ядра, які можуть викликатися додатками, утворюють інтерфейс прикладного програмування – API.

Функції, що виконуються модулями ядра, є найчастіше використовуваними функціями операційної системи, тому швидкість їх виконання визначає продуктивність усієї системи в цілому. Для забезпечення високої швидкості роботи ОС усі модулі ядра або велика їх частина постійно знаходяться в оперативній пам'яті, тобто є резидентними.

Ядро є рушійною силою усіх обчислювальних процесів в комп'ютерній системі, і крах ядра рівносильний краху усієї системи. Тому розробники ОС приділяють особливу увагу надійності кодів ядра, в результаті процес їх відладки може розтягуватися на багато місяців. Ядро оформляється у вигляді програмного модуля деякого спеціального формату, що відрізняється від формату додатків користувача. Термін «ядро» в різних ОС трактують по-різному. Однією з визначальних властивостей ядра є робота в привілейованому режимі.

Інші модулі ОС виконують дуже корисні, але менш обов'язкові функції. Наприклад, до таких допоміжних модулів можуть бути віднесені програми архівації даних, дефрагментації диска і тому подібні. Допоміжні модулі ОС оформляються або у вигляді додатків, або у вигляді бібліотек процедур.

Рішення про те, чи є яка-небудь програма частиною ОС чи 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.

Окрема програма може існувати певний час як додаток користувача, а потім стати частиною ОС, або навпаки.

Допоміжні модулі ОС підрозділяються на такі групи (рис. 3.1):

– утиліти – програми, які виконують окремі задачі управління і супроводу комп'ютерної системи, такі, наприклад, як програми стискування дисків, архівації даних;
– системні обробляючі програми – текстові або графічні редактори, компілятори, компонувальники, відладчики (дебагер, англ. debugger);
– програми надання користувачеві додаткових послуг – спеціальний варіант призначеного для користувача інтерфейсу, калькулятор і навіть ігри;
– бібліотеки процедур різного призначення, що спрощують розробку додатків, наприклад бібліотека математичних функцій, функцій введення-виведення тощо.

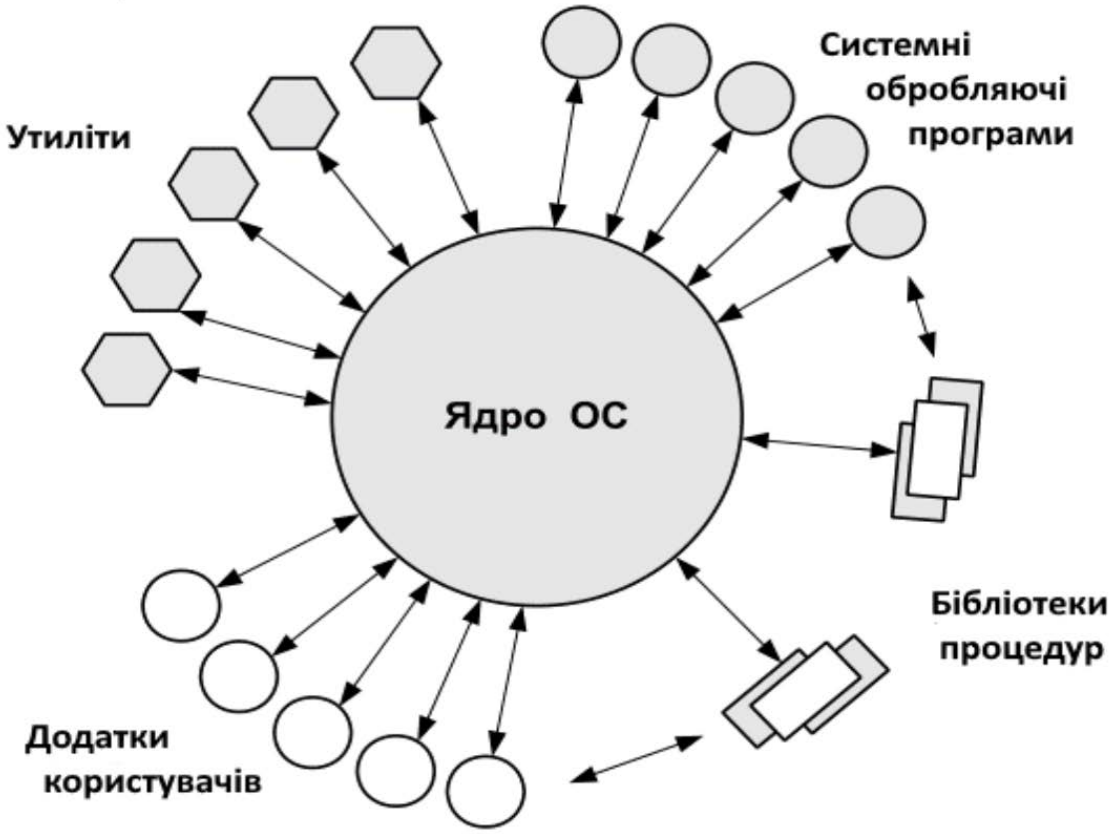

Рисунок 3.1 – Взаємодія між ядром і допоміжними модулями ОС

Для виконання своїх задач як звичайні додатки, так і утиліти, оброблювальні програми і бібліотеки ОС, звертаються до функцій ядра за допомогою системних викликів.

Модулі ОС, оформлені у вигляді утиліт, системних оброблювальних програм і бібліотек, завантажуються в оперативну пам'ять тільки на час виконання своїх функцій, тобто є транзитними. Постійно в оперативній пам'яті розташовуються тільки

найнеобхідніші коди ОС, що становлять її ядро. Така організація ОС економить оперативну пам'ять комп'ютера.

Розділення операційної системи на ядро і модулі-додатки забезпечує легку розширюваність ОС. Щоб додати нову високорівневу функцію, досить розробити новий додаток, і при цьому не вимагається модифікувати основні функції, що утворюють ядро системи.

3.2.2 Ядро в привілейованому режимі

Для надійного управління ходом виконання додатків операційна система повинна мати певні привілеї стосовно додатків. Інакше некоректно працюючий додаток може втрутитися в роботу ОС і, наприклад, зруйнувати частину її коду. Усі зусилля розробників операційної системи виявляться марними, якщо їх рішення втілені в незахищені від додатків модулі системи, якими б елегантними і ефективними ці рішення не були.

Операційна система повинна мати виняткові повноваження також для того, щоб грати роль арбітра в суперечці додатків за ресурси комп'ютера в мультипрограмному режимі. Жодний додаток не повинен мати можливості без відома ОС отримувати додаткову область пам'яті, займати процесор довше дозволеного операційною системою періоду часу, безпосередньо управляти спільно використовуваними зовнішніми пристроями.

Забезпечити привілеї ОС 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). Мається на увазі, що операційна система або деякі її частини працюють в привілейованому режимі, а додатки – в режимі користувача.

Оскільки ядро виконує всі основні функції ОС, то найчастіше саме ядро стає тією частиною ОС, яка працює в привілейованому режимі. Іноді ця властивість – робота в привілейованому режимі – служить основним визначенням поняття «ядро».

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, а при поверненні до додатку – перемикання з привілейованого режиму в режим користувача (рис. 3.2). В усіх типах процесорів із-за додаткової двократної затримки перемикання перехід на процедуру зі зміною режиму виконується повільніше, ніж виклик процедури без зміни режиму.

Рисунок 3.2 – Перемикання процесора з режиму користувача в привілейований режим

Додатки ставляться в підпорядковане положення за рахунок заборони виконання в режимі користувача деяких критичних команд, пов'язаних з перемиканням процесора із завдання на завдання, управлінням пристроями введення-виведення, доступом до механізмів розподілу і захисту пам'яті. Виконання деяких інструкцій в режимі користувача забороняється безумовно, тоді як інші забороняється виконувати тільки за певних умов. Очевидно, що до таких інструкцій належить інструкція переходу в привілейований режим.

Наприклад, інструкції введення-виведення можуть бути заборонені додаткам при доступі до контролера жорсткого диска, який зберігає дані, загальні для ОС і усіх додатків, але дозволені при доступі до послідовного порту, який виділений в монопольне користування для певного додатку. Важливо, що умови дозволу виконання критичних інструкцій знаходяться під повним контролем ОС і цей контроль забезпечується за рахунок набору інструкцій, безумовно заборонених для режиму користувача.

Аналогічним чином забезпечуються привілеї ОС при доступі до пам'яті. Наприклад, виконання інструкції доступу до пам'яті для додатка дозволяється, якщо інструкція звертається до області пам'яті, відведеної цьому додатку операційною системою, і забороняється при зверненні до областей пам'яті, займаних ОС або іншими додатками. Повний контроль ОС над доступом до пам'яті досягається за рахунок того, що інструкція або інструкції конфігурації механізмів захисту пам'яті дозволяється виконувати тільки в привілейованому режимі. Механізми захисту пам'яті використовуються операційною системою не лише для захисту своїх областей пам'яті від додатків, але і для захисту областей пам'яті, виділених ОС якому-небудь додатку, від інших додатків. Говорять, що кожний додаток працює у своєму адресному просторі. Ця властивість дозволяє локалізувати некоректно працюючий додаток у власній області пам'яті, так що його помилки не роблять впливу на інші додатки і операційну систему.

Архітектура ОС, заснована на привілейованому ядрі і додатках, які виконуються в режимі користувача, стала, по суті, класичною. Її використовують багато популярних операційних систем, у тому числі численні версії UNIX, IBM OS/390, OS/2 і з певними модифікаціями – Windows.

У 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 x86/Pentium 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, займаної модулями ОС, від некоректно працюючого додатку. Розробники NetWare пішли на таке потенційне зниження надійності своєї операційної системи, оскільки обмежений набір її спеціалізованих додатків дозволяє компенсувати цей архітектурний недолік за рахунок ретельної відладки кожного додатку.

В одному режимі працюють також ядро і додатки тих операційних систем, які розроблені для процесорів, що взагалі не підтримують привілейованого режиму роботи. Найпопулярнішим процесором такого типу був процесор Intel 8088/86, що

послужив основою для персональних комп'ютерів компанії IBM. Операційна система MS-DOS, розроблена компанією Microsoft для цих комп'ютерів, складалася з двох модулів msdos.sys і io.sys, що складали ядро системи. До цих модулів з системними викликами зверталися командний інтерпретатор command.com, системні утиліти і додатки.

Поява в пізніших версіях процесорів Intel (починаючи з 80286) можливості роботи в привілейованому режимі не були використані розробниками MS-DOS. Ця ОС завжди працювала на процесорах цього типу в так званому реальному режимі, в якому емулюється процесор 8086/88. Не слід вважати, що реальний режим є синонімом режиму користувача, а привілейований режим – його альтернативою. Реальний режим був реалізований тільки для сумісності пізніх моделей процесорів з ранньою моделлю 8086/88 і альтернативою йому є захищений режим роботи процесора з доступними всіма особливостями процесорів пізніх моделей.

3.2.3 Монолітні системи

Відмітною особливістю більшості сучасних ОС є велике монолітне ядро. Ядро ОС забезпечує більшість її можливостей, включаючи планування, роботу з файловою системою, мережеві функції, роботу драйверів різних пристроїв, управління пам'яттю і багато що інше. Монолітне ядро реалізується як єдиний процес, усі елементи якого використовують один і той же адресний простір. Для побудови монолітної системи необхідно скомпілювати усі окремі процедури, а потім зв'язати їх в єдиний об'єктний файл. Тут повністю відсутнє приховування деталей реалізації – кожна процедура бачить будь-яку іншу процедуру.

У загальному випадку «структура» монолітної системи є відсутністю структури (рис. 3.3). ОС написана як набір процедур, кожна з яких може викликати інші, коли їй це треба. При використанні цієї техніки кожна процедура системи має певний інтерфейс у термінах параметрів і результатів, і кожна може викликати будь-яку іншу для виконання потрібної для неї корисної роботи.

Така відсутність структури була несумісна з розширенням ОС, до того ж такі ОС були дуже великими і складними. Так, перша версія OS/2 була створена колективом з 5000 програмістів за 5 років і містила близько 1 млн рядків коду. Розроблена пізніше ОС Multics містила вже 20 млн. рядків коду. Прикладами ОС з

монолітним ядром можуть служити ранні версії ядра UNIX. Розмір програмного коду ядра Linux в 1995 році складав 250 тисяч рядків, а в 2010 році їх число збільшилось вже до 14 мільйонів.

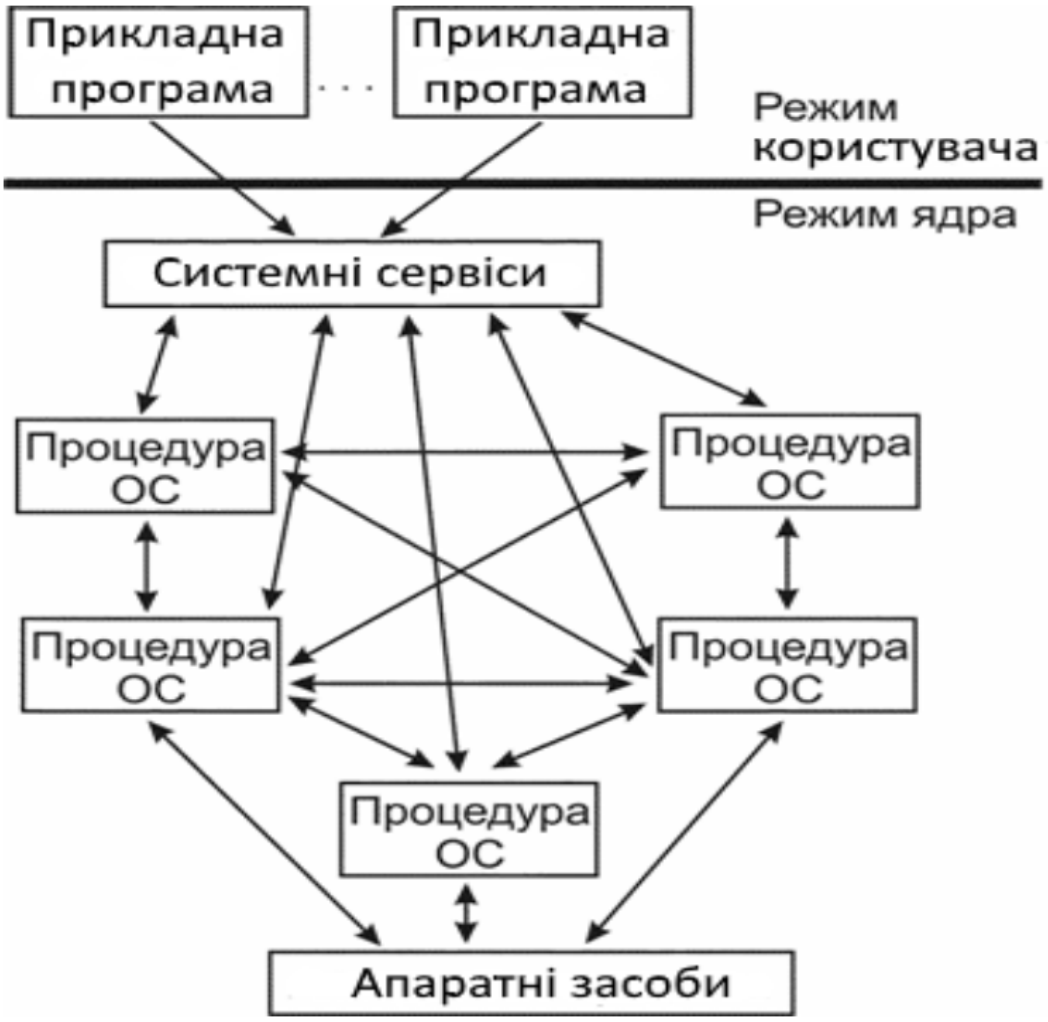

Рисунок 3.3 – Монолітна структура ОС

Проте навіть такі монолітні системи можуть бути трохи структурованими. При зверненні до системних викликів, підтримуваних ОС, параметри поміщаються в строго певні місця, такі, як регістри або стек, а потім виконується спеціальна команда переривання, відома як виклик ядра або виклик супервізора. Ця команда перемикає машину з режиму користувача в режим ядра, що називається також режимом супервізора, і передає управління ОС. Потім ОС перевіряє параметри виклику для того, щоб визначити, який системний виклик має бути виконаний. Після цього ОС індексує таблицю, що містить посилання на процедури, і викликає відповідну процедуру. Така організація ОС припускає таку структуру:

1. Головна програма, яка викликає необхідні сервісні процедури.
2. Набір сервісних процедур, що реалізовують системні виклики.
3. Набір утиліт, обслуговуючих сервісні процедури.

У цій моделі для кожного системного виклику є одна сервісна процедура. Утиліти виконують функції, які потрібні декільком сервісним процедурам. Це ділення процедур на три шари показане на рис. 3.4.

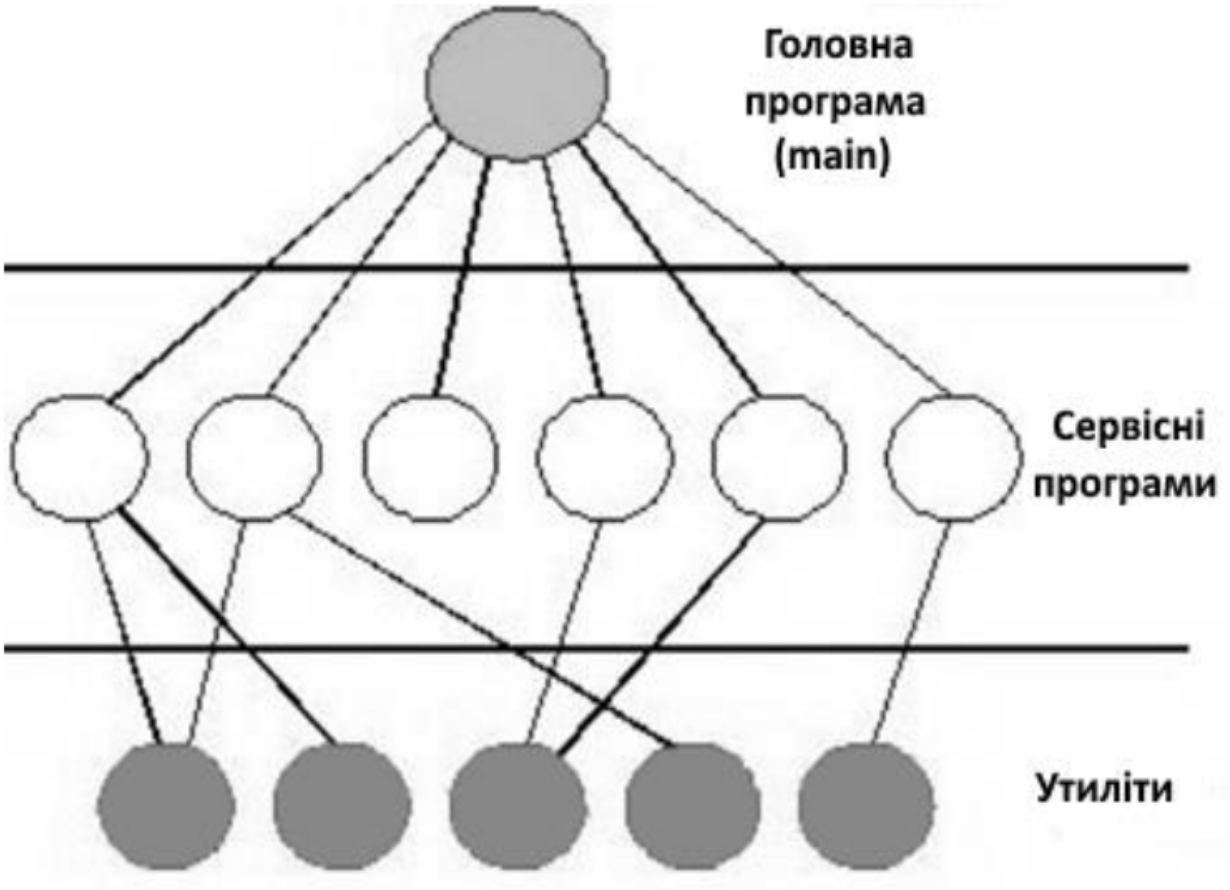

Рисунок 3.4 – Проста структуризація монолітної ОС

3.2.4 Багаторівневі (багатошарові) системи

Узагальненням попереднього підходу є організація ОС як ієрархії рівнів. Рівні утворюються групами функцій ОС – файлова система, управління процесами і пристроями і тому подібне. Кожен рівень може взаємодіяти тільки зі своїм безпосереднім сусідом, який знаходиться вище або нижче.

Обчислювальну систему, працюючу під управлінням ОС на основі ядра, можна розглядати як систему, що складається з трьох основних ієрархічно розташованих шарів: нижній шар утворює апаратуру, проміжний – ядро, а утиліти, оброблювальні

програми і додатки, складають верхній шар системи (рис. 3.5). Шарову структуру обчислювальної системи зручно зображати 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.

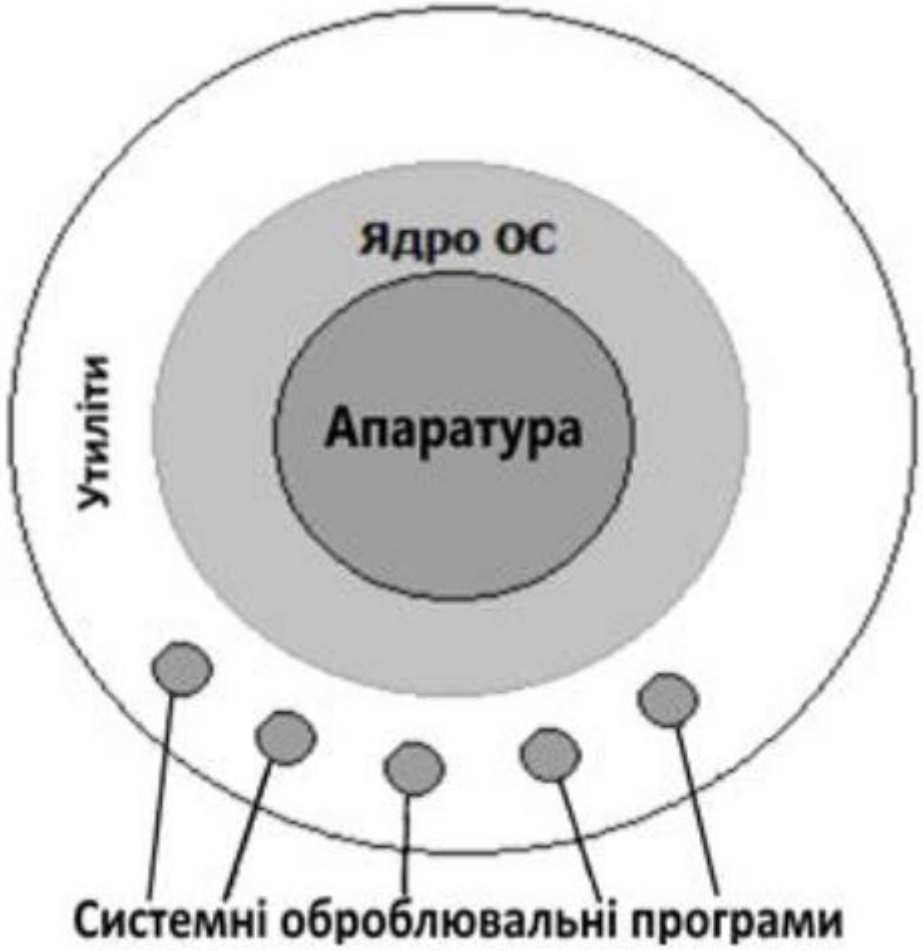

Рисунок 3.5 – Тришарова схема обчислювальної системи

Багатошаровий підхід є універсальним і ефективним способом декомпозиції складних систем будь-якого типу, у тому числі і програмних. Відповідно до цього підходу система складається з ієрархії шарів. Кожен шар обслуговує вищерозміщений шар, виконуючи для нього деякий набір функцій, які утворюють міжшаровий інтерфейс. На основі функцій шару, що пролягає нижче, наступний (вгору за ієрархією) шар будує свої функції – складніші і потужніші, які, у свою чергу, виявляються примітивами для створення ще потужніших функцій вищерозміщеного шару. Суворі правила стосуються тільки взаємодії між шарами системи, а між

модулями усередині шару зв'язки можуть бути довільними. Окремий модуль може виконати свою роботу або самостійно, або звернутися до іншого модуля свого шару, або звернутися за допомогою до шару, що пролягає нижче, через міжшаровий інтерфейс.

Першою системою, побудованою таким чином, була пакетна система THE (Technishe Hogeschool Eindhoven) Multiprogramming System, яку створив Е. Дейкстра і його студенти в 1968 році (рис. 3.6). Система мала 6 рівнів (шарів).

Рівень 0 займався розподілом часу процесора, перемикаючи процеси за перериванням або після закінчення часу.

Рівень 1 управляв пам'яттю – розподіляв оперативну пам'ять і простір на магнітному барабані для тих частин процесів (сторінок), для яких не було місця в ОП, тобто шар 1 виконував функції віртуальної пам'яті.

Рисунок 3.6 – Багатошарова операційна система THE

Рівень 2 управляв зв'язком між консоллю оператора і процесами. За допомогою цього рівня кожен процес мав свою власну консоль оператора.

Рівень 3 управляв пристроями введення-виведення і буферизував потоки інформації до них і від них. За допомогою рівня 3 кожен процес замість того, щоб працювати з конкретними пристроями, з їх різноманітними особливостями, звертався до абстрактних пристроїв введення-виведення.

На рівні 4 працювали призначені для користувача програми, яким не потрібно було піклуватися ні про процеси, ні про пам'ять, ні про управління пристроями введення-виведення.

На рівні 5 розміщувався процес системного оператора.

Оскільки ядро є складним багатофункціональним комплексом, то багатошаровий підхід поширюється і на структуру ядра. Ядро може складатися з таких шарів (рис. 3.7).

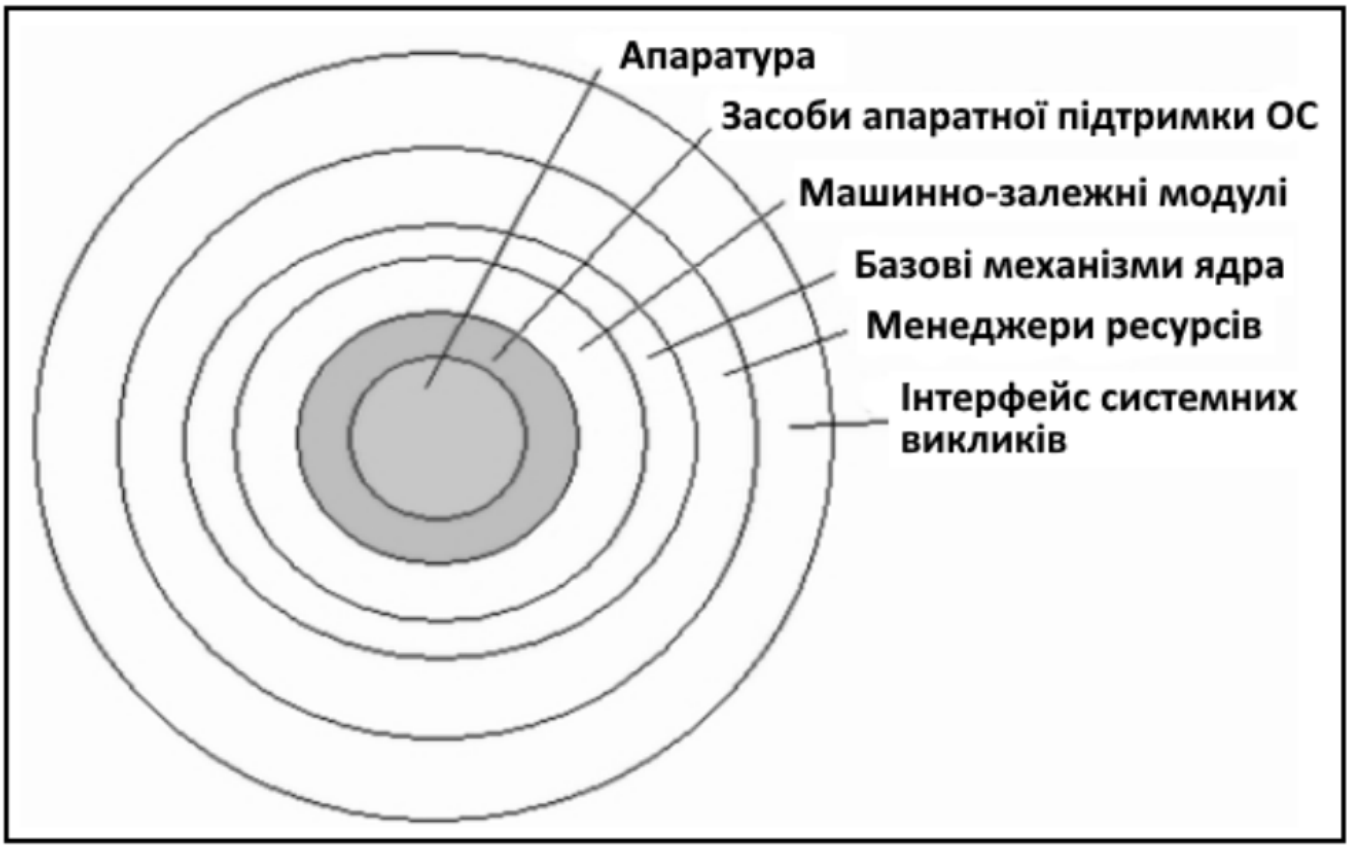

Рисунок 3.7 – Багатошарова структура ядра ОС

Засоби апаратної підтримки ОС. Досі про операційну систему говорилося як про комплекс програм, але, взагалі кажучи, частина функцій ОС може виконуватися і апаратними засобами. Тому іноді можна зустріти визначення операційної системи як сукупності програмних і апаратних засобів, що й показано на рис. 3.7. До операційної системи відносять не всі апаратні пристрої комп'ютера, а тільки засоби апаратної підтримки ОС, тобто ті, які прямо беруть участь в організації обчислювальних процесів: засоби підтримки привілейованого режиму, систему переривань, засоби перемикання контекстів процесів, засоби захисту областей пам'яті тощо.

Машинно-залежні компоненти ОС. Цей шар утворюють програмні модулі, в яких відбивається специфіка апаратної платформи комп'ютера. В ідеалі цей шар повністю екранує вищерозміщені шари ядра від особливостей апаратури. Це дозволяє розробляти вищерозміщені шари на основі машинно- незалежних модулів, існуючих в єдиному екземплярі для усіх типів апаратних платформ, підтримуваних цією ОС. Прикладом екрануючого шару може служити шар HAL операційної системи Windows NT.

Базові механізми ядра. Цей шар виконує найпримітивніші операції ядра, такі як програмне перемикання контекстів процесів, диспетчеризацію переривань, переміщення сторінок з пам'яті на диск і назад тощо. Модулі цього шару не приймають рішень про розподіл ресурсів – вони тільки відпрацьовують прийняті «вгорі» рішення, що і дає привід називати їх виконавчими механізмами для модулів верхніх шарів. Наприклад, рішення про те, що в даний момент треба перервати виконання поточного процесу А і почати виконання процесу В, приймається менеджером процесів на вищерозміщеному 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 з управління основними ресурсами обчислювальної системи. На цьому шарі працюють менеджери (які називаються також диспетчерами) процесів, введення-виведення, файлової системи і оперативної пам'яті.

Розбиття на менеджери може бути і дещо іншим. Наприклад, менеджер файлової системи іноді об'єднують з менеджером введення-виведення, а функції управління доступом користувачів до системи в цілому і її окремим об'єктам доручають окремому менеджерові безпеки. Кожен з менеджерів веде облік вільних і використовуваних ресурсів певного типу і планує їх розподіл відповідно до запитів додатків.

Менеджер віртуальної пам'яті управляє переміщенням сторінок з оперативної пам'яті на диск і назад. Менеджер повинен відстежувати інтенсивність звернень до сторінок, час перебування їх в пам'яті, стану процесів, що використовують дані, і

багато інших параметрів, на підставі яких він час від часу приймає рішення про те, які сторінки необхідно вивантажити і які – завантажити.

Для виконання прийнятих рішень менеджер звертається до шару базових механізмів, що пролягає нижче, із запитами про завантаження (вивантаженню) конкретних сторінок. Усередині шару менеджерів існують тісні взаємні зв'язки, що відбивають той факт, що для виконання процесу потрібний доступ одночасно до декількох ресурсів – процесора, області пам'яті, можливо, до певного файлу або пристроїв введення-виведення. Наприклад, при створенні процесу менеджер процесів звертається до менеджера пам'яті, який повинен виділити процесу певну область пам'яті для його кодів і даних.

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, обслуговуючі системні виклики, надають доступ до ресурсів системи в зручній і компактній формі, без вказівки деталей їх фізичного розташування. Для здійснення таких комплексних дій системні виклики звертаються за допомогою до функцій шару менеджерів ресурсів, причому для виконання одного системного виклику може знадобитися декілька таких звернень.

Наведене розбиття ядра ОС на шари є досить умовним. У реальній системі кількість шарів і розподіл функцій між ними може бути і іншим. У системах, призначених для апаратних платформ одного типу, наприклад ОС NetWare, шар машинно-залежних модулів не виділяється, зливаючись з шаром базових механізмів і, частково, з шаром менеджерів ресурсів. Не завжди оформляються в окремий шар базові механізми. У цьому випадку менеджери ресурсів не лише планують використання ресурсів, але і самостійно реалізують свої плани.

Можлива і протилежна картина, коли ядро складається з більшої кількості шарів. Наприклад, менеджери ресурсів, складаючи певний шар ядра, у свою чергу, можуть мати багатошарову структуру. Передусім це стосується до менеджера введення-виведення, нижній шар якого складають драйвери пристроїв, наприклад драйвер жорсткого диска або драйвер мережевого адаптера, а верхні шари – драйвери

файлових систем або протоколів мережевих служб, що мають справу з логічною організацією інформації.

Спосіб взаємодії шарів в реальній ОС також може відхилятися від описаної вище схеми. Для прискорення роботи ядра в деяких випадках відбувається безпосереднє звернення з верхнього шару до функцій нижніх шарів, оминаючи проміжні. На багатьох апаратних платформах для реалізації системного виклику використовується інструкція програмного переривання. Цим додаток фактично викликає модуль первинної обробки переривань, який знаходиться в шарі базових механізмів, а вже цей модуль викликає потрібну функцію з шару системних викликів. Самі функції системних викликів також іноді порушують субординацію ієрархічних шарів, звертаючись прямо до базових механізмів ядра. Вибір кількості шарів ядра є відповідальною і складною справою.

Збільшення числа шарів веде до деякого уповільнення роботи ядра за рахунок додаткових накладних витрат на міжшарову взаємодію, а зменшення числа шарів погіршує розширюваність і логічність системи. ОС, що пройшли довгий шлях еволюційного розвитку, наприклад, багато версій UNIX, мають невпорядковане ядро з невеликим числом чітко виділених шарів (рис. 3.8).

У порівняно «молодих» операційних систем, таких як Windows NT, ядро розділене на більше число шарів і їх взаємодія формалізована набагато краще (рис. 3.9).

Хоча такий структурний підхід на практиці працював непогано (висока продуктивність), сьогодні він все більше сприймається монолітним. У системах, що мають багаторівневу структуру, було нелегко видалити один шар і замінити його іншим в силу множинності і розмитості інтерфейсів між шарами. Додавання нових функцій і зміна існуючих вимагали хорошого знання операційної системи і багато часу (великий код ядра, і як наслідок великий вміст помилок).

Коли стало ясно, що операційні системи живуть довго і повинні мати можливості розвитку і розширення, монолітний і багатошаровий підходи стали давати тріщину, і на зміну їм прийшла модель клієнт-сервер і тісно пов'язана з нею концепція мікроядра.

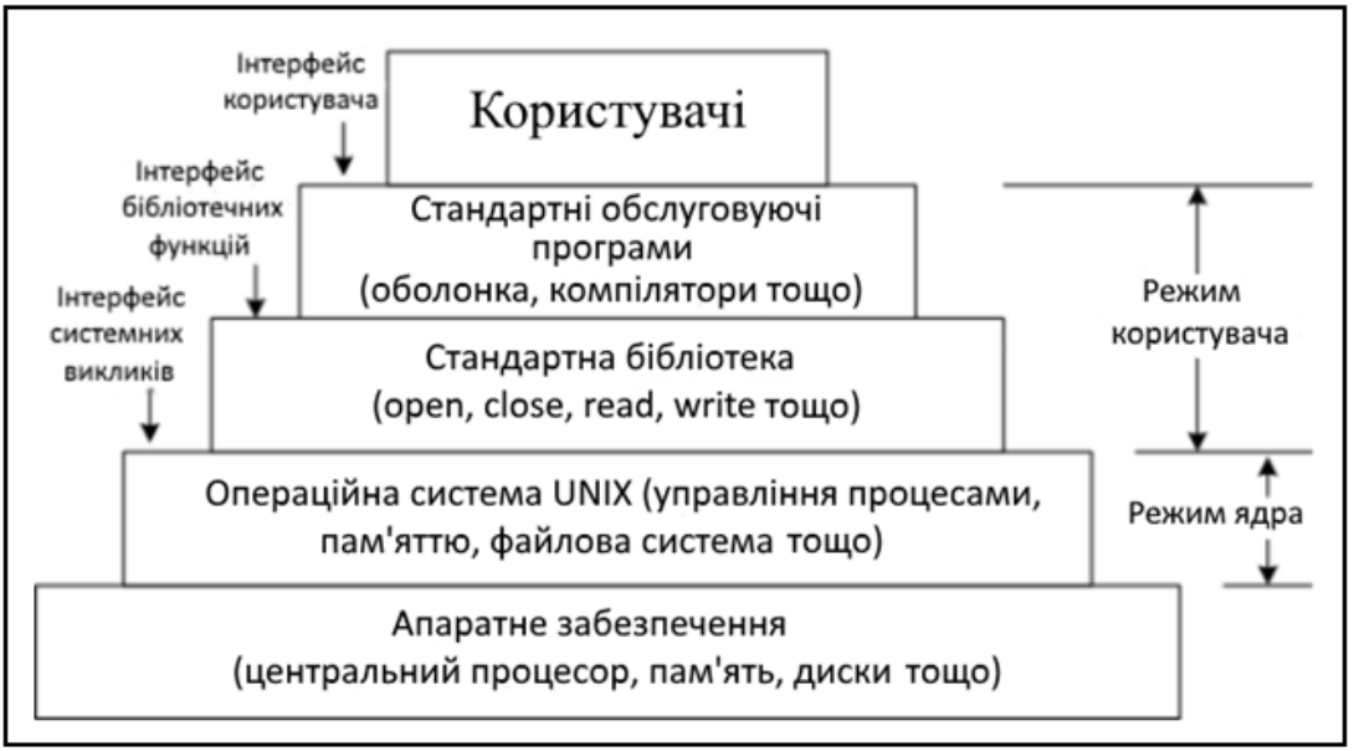

Рисунок 3.8 – Структура ОС UNIX

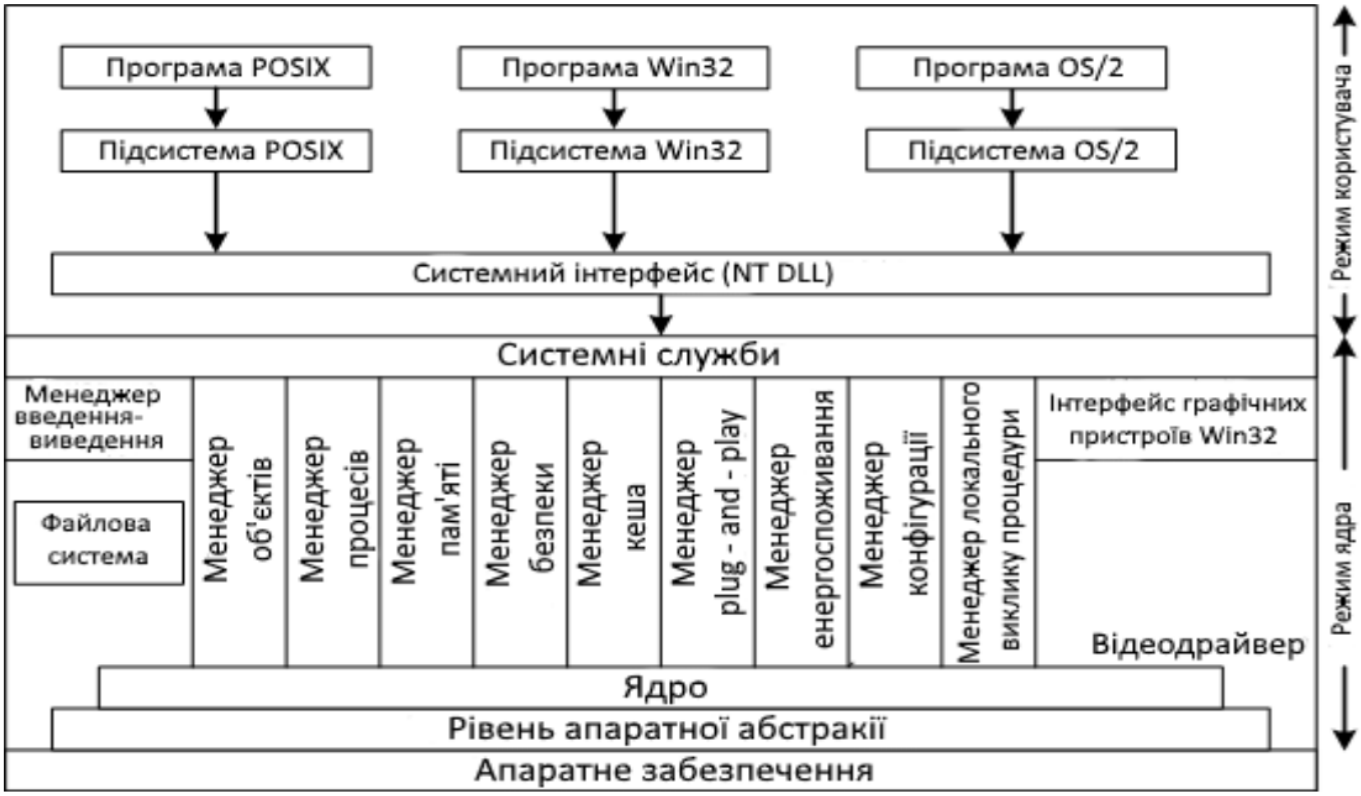

Рисунок 3.9 – Структура Windows 2000

3.2.5 Модель клієнт-сервер і мікроядро

Модель клієнт-сервер – це ще один підхід до структуризації ОС. У широкому сенсі модель клієнт-сервер припускає наявність програмного компонента-споживача якого-небудь сервісу – клієнта, і програмного компонента-постачальника цього сервісу – сервера. Взаємодія між клієнтом і сервером стандартизується, так що сервер може обслуговувати клієнтів, реалізованих різними способами і, можливо, різними виробниками. При цьому головною вимогою є те, щоб вони просили послуги сервера зрозумілим йому способом. Ініціатором обміну є клієнт, який посилає запит на обслуговування серверу, що знаходиться в стані очікування запиту.

Ще в деяких ранніх ОС частина функцій реалізувалася в режимі користувача, оскільки системний виклик не був потрібний. Системний виклик виконується повільніше, ніж виклик функції, реалізованої в режимі користувача, оскільки процесор двічі перемикається між режимами.

До складу мікроядра входять машинно-залежні модулі, а також модулі, що виконують базові (але не всі) функції ядра з управління процесами, обробки переривань, управління віртуальною пам'яттю, пересилання повідомлень і управління пристроями введення-виведення, пов'язані із завантаженням або читанням регістрів пристроїв. Набір функцій мікроядра відповідає функціям шару базових механізмів звичайного ядра. Такі функції операційної системи важко, якщо не неможливо, виконати в просторі користувача.

Усі інші більш високорівневі функції ядра оформляються у вигляді додатків (процеси-сервери), працюючих в режимі користувача. Такий підхід дозволяє розділити задачу розробки ОС на розробку ядра і розробку серверів. Сервери можна Налагоджувати для вимог конкретних додатків або середовища. Виділення в структурі системи мікроядра спрощує реалізацію системи, забезпечує її гнучкість (рис. 3.10).

Стосовно структуризації ОС ідея полягає в розбитті її на декілька процесів-серверів, кожен з яких виконує окремий набір сервісних функцій – наприклад, управління пам'яттю, створення або планування процесів. Кожен сервер виконується в призначеному для користувача режимі.

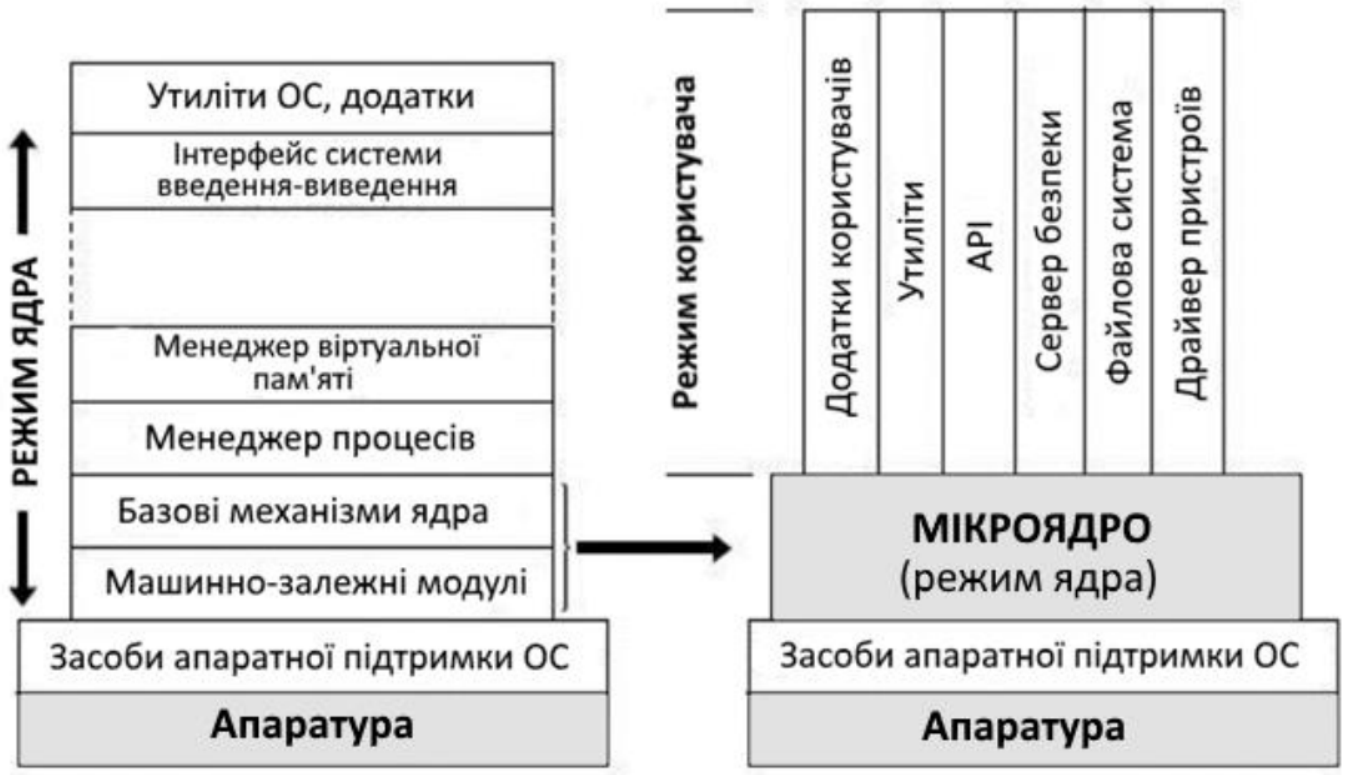

Рисунок 3.10 – Перехід до мікроядерної архітектури

Клієнт, яким може бути або інший компонент ОС, або прикладна програма, просить сервіс, посилаючи повідомлення на сервер. Ядро ОС (що називається тут мікроядром), працюючи в привілейованому режимі, доставляє повідомлення потрібного сервера, сервер виконує операцію, після чого ядро повертає результати клієнтові за допомогою іншого повідомлення (рис. 3.11).

Мікроядерна архітектура є альтернативою класичному способу побудови операційної системи. Під класичною архітектурою в даному випадку розуміється розглянута вище структурна організація ОС, відповідно до якої всі основні функції операційної системи, що становлять багатошарове ядро, виконуються в привілейованому режимі.

Рисунок 3.11 – Структура ОС клієнт-сервер

Суть мікроядерної архітектури полягає в такому. У привілейованому режимі залишається працювати тільки дуже невелика частина ОС, що називається мікроядром. Мікроядро захищене від інших частин ОС і додатків.

Підхід з використанням мікроядра замінив вертикальний розподіл функцій операційної системи на горизонтальний. Компоненти, мікроядра, що лежать вище, хоча і використовують повідомлення, що пересилаються через мікроядро, взаємодіють один з одним безпосередньо. Мікроядро грає роль регулювальника. Воно перевіряє повідомлення, пересилає їх між серверами і клієнтами, і надає доступ до апаратури.

Схема зміни режимів при виконанні системного виклику в ОС з мікроядерною архітектурою виглядає, як показана на рис. 3.12. З рисунка видно, що при класичній організації ОС (рис. 3.12, а) виконання системного виклику супроводжується двома перемиканнями режимів (2 t). При мікроядерній організації (рис. 3.12, б) – чотирма (4 t). Отже, ОС з мікроядерною архітектурою за інших рівних умов завжди буде менш продуктивною, чим ОС з класичним ядром.

Мікроядро реалізує важливі функції, що лежать в основі операційної системи. Це базис для менш істотних системних служб і додатків. Саме питання про те, які з системних функцій вважати несуттєвими, і, відповідно, не включати їх до складу ядра, є предметом суперечки серед прибічників ідеї мікроядра. У загальному випадку,

підсистеми, що були традиційно невід'ємними частинами ОС, – файлові системи, управління вікнами і забезпечення безпеки – стають периферійними модулями, що взаємодіють з ядром і один з одним, і працюють в режимі користувача.

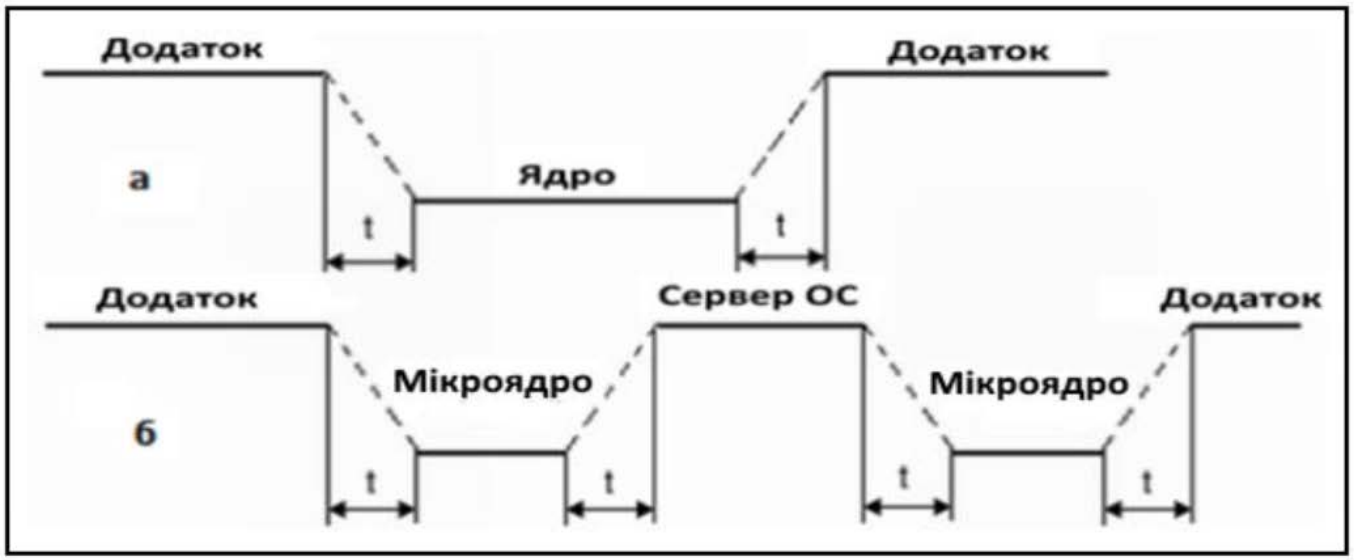

Рисунок 3.12 – Обробка системного виклику в мікроядерній архітектурі

Головний принцип розділення роботи між мікроядром і модулями, що оточують його, – включати в мікроядро тільки ті функції, яким абсолютно необхідно виконуватися в режимі супервізора і в привілейованому просторі. Під цим маються на увазі машинно-залежні програми, деякі функції управління процесами, обробка переривань, підтримка пересилки повідомлень, деякі функції управління пристроями введення-виведення, пов'язані із завантаженням команд в регістри пристроїв. Ці функції операційної системи важко, якщо не неможливо, виконати програмам, працюючим в просторі користувача.

Тут важливо зробити відмінність. Запуск процесу або потоку вимагає доступу до апаратури, так що за логікою – це функція ядра. Але ядру все одно, який з потоків запускати, тому рішення про пріоритети потоків і дисципліну постановки в чергу може приймати працюючий поза ядром планувальник.

Окрім вже представлених міркувань, переміщення планувальника на призначений для користувача рівень може знадобитися для чисто комерційних цілей. Деякі виробники ОС планують ліцензувати своє мікроядро іншим постачальникам, яким може потрібно замінити початковий планувальник на інший, такий, що

підтримує, наприклад, планування в завданнях реального часу або що реалізовує якийсь спеціальний алгоритм планування.

Як і управління процесами, управління пам'яттю може розподілятися між мікроядром і сервером, працюючим в режимі користувача.

Драйвери пристроїв також можуть розташовуватися як усередині ядра, так і поза ним. При розміщенні драйверів пристроїв поза мікроядром для забезпечення можливості дозволу і заборони переривань, частина програми драйвера повинна виконуватися в просторі ядра. Відділення драйверів пристроїв від ядра робить можливою динамічну конфігурацію ОС.

Окрім динамічної конфігурації, є і інші причини розглядати драйвери пристроїв в якості процесів режиму користувача. СУБД, наприклад, може мати свій драйвер, оптимізований під конкретний вид доступу до диска, але його не можна буде підключити, якщо драйвери будуть розташовані в ядрі. Цей підхід також сприяє переносимості системи, оскільки функції драйверів пристроїв можуть бути в багатьох випадках абстраговані від апаратної частини.

Нині саме операційні системи, побудовані з використанням моделі клієнт-сервер і концепції мікроядра, найбільшою мірою задовольняють вимогам, що пред'являються до сучасних ОС.

Однаковий інтерфейс. Використання мікроядра припускає однаковий інтерфейс запитів, генерованих процесами. Процесам не треба відрізняти служби, що виконуються на рівні ядра і на призначеному для користувача рівні, оскільки доступ до усіх цих служб здійснюється тільки за допомогою передачі повідомлень.

Можливість розширення ОС. Архітектура мікроядра сприяє розширюваності ОС, дозволяючи додавати в них нові сервісні, а також забезпечувати нові множинні сервіси в одній і тій же функціональній області. Складність монолітних операційних систем, що збільшується, зробила важким, якщо взагалі можливим, внесення змін до ОС з гарантією надійності її подальшої роботи. Обмежений набір чітко певних інтерфейсів мікроядра відкриває шлях до впорядкованого зростання і еволюції ОС.

Переносимість. В архітектурі з мікроядром спеціально призначений для конкретного процесора код, або велика його частина, знаходиться в мікроядрі. Тому

зміни, необхідні для перенесення системи на новий процесор, зводяться до мінімуму і мають тенденцію до розміщення в окремих логічних групах.

Надійність. Хоча модульність допомагає підвищити надійність роботи системи, архітектура з мікроядром дає ще вагоміші переваги. Маленьке мікроядро можна ретельно протестувати. Використання в ньому невеликої кількості інтерфейсів прикладного програмування підвищує шанси на те, що сервіси ОС, які працюють поза ядром, будуть реалізовані за допомогою якісного коду. Системний програміст має у своєму розпорядженні обмежену кількість інтерфейсів і обмежені способи організації взаємодії, тому він не зможе негативно вплинути на інші системні компоненти.

Кожен сервер виконується у вигляді окремого процесу у своїй власній області пам'яті, і таким чином захищений від інших процесів і від інших серверів операційної системи. Більше того, оскільки сервери виконуються в просторі користувача, вони не мають безпосереднього доступу до апаратури і не можуть модифікувати пам'ять, в якій зберігається управляюча програма. І якщо окремий сервер може потерпіти крах, то він може бути перезапущений без зупинки або ушкодження іншої частини ОС.

Обробка апаратних переривань. В архітектурі мікроядра є можливість обробляти апаратні переривання аналогічно повідомленням, а також включати в адресний простір порти введення-виведення. Таке мікроядро може розпізнавати переривання, але не обробляти їх. Замість цього воно генерує повідомлення процесу (сервісу), який виконується на рівні користувача і пов'язаному з цим перериванням. Таким чином, коли переривання дозволене, з ним зіставляється процес на рівні користувача, і таке відображення підтримується ядром.

Розподілення обчислень. Ця модель також добре підходить для розподілених обчислень, оскільки окремі сервери можуть працювати на різних процесорах мультипроцесорного комп'ютера або навіть на різних комп'ютерах, оскільки використовує механізми, аналогічні мережевим: взаємодія клієнтів і серверів шляхом обміну повідомленнями. При отриманні повідомлення від процесу мікроядро може обробити його самостійно або переслати іншому процесу, оскільки мікроядру все одно, чи прийшло повідомлення від локального або віддаленого процесу.

Прикладне середовище. Відповідно до мікроядерної архітектури всі функції ОС реалізуються мікроядром і серверами в режиме користувача. Важливо, що кожне

прикладне середовище оформляється у вигляді окремого сервера і не включає базових механізмів (рис. 3. 13).

Додатки, використовуючи API, поводяться з системними викликами до відповідного прикладного середовища через мікроядро. Прикладне середовище обробляє запит, виконує його (можливо, за допомогою базових функцій мікроядра) і посилає додатку результат. У ході виконання запиту прикладному середовищу доводиться, у свою чергу, звертатися до базових механізмів ОС, мікроядра та до інших серверів ОС.

Рисунок 3. 13 – Мікроядерний підхід до реалізації багатьох прикладних середовищ

Головним недоліком мікроядерного підходу є зниження продуктивності. Системний виклик виконується повільніше, ніж виклик функції, реалізованої в режимі користувача, оскільки замість двох перемикань режиму процесора в разі системного виклику при класичній організації ОС здійснюється чотири (два – під час обміну між клієнтом і мікроядром, два – між сервером і ядром).

Але все таки цей недолік є більше теоретичним, оскільки на практиці продуктивність і надійність ядра залежить безпосередньо від якості його реалізації.

Серйозність цього недоліку добре ілюструє історія розвитку Windows NT. У версіях 3.1 і 3.5 диспетчер вікон, графічна бібліотека і високорівневі драйвери графічних пристроїв входили до складу сервера призначеного для користувача

режиму, і виклик функцій цих модулів здійснювався відповідно до мікроядерної схеми. Проте дуже скоро розробники Windows NT зрозуміли, що такий механізм звернень до часто використовуваних функцій графічного інтерфейсу істотно уповільнює роботу додатків і робить цю операційну систему уразливою в умовах гострої конкуренції. В результаті у версію Windows NT 4.0 були внесені істотні зміни – усі перелічені вище модулі були перенесені в ядро, що віддалило цю ОС від ідеальної мікроядерної архітектури, та зате різко підвищило її продуктивність.

Існує ще одна проблема, з якою стикаються розробники операційної системи, які вирішили застосувати мікроядерний підхід, – що включати в мікроядро, а що виносити в призначений для користувача простір. В ідеальному випадку мікроядро може складатися тільки із засобів передачі повідомлень, засобів взаємодії з апаратурою, у тому числі засобів доступу до механізмів привілейованого захисту. Проте багато розробників не завжди жорстко дотримуються принципу мінімізації функцій ядра, часто жертвуючи цим заради підвищення продуктивності.

Для можливості уявлення про розміри мікроядер операційних систем у ряді джерел наводяться такі дані:

- типове мікроядро першого покоління – 300 Кб коду і 140 інтерфейсів системних викликів;
- мікроядро ОС L4 (друге покоління) – 12 Кб коду і 7 інтерфейсів системних викликів;
- мікроядро UNIX-подібної POSIX-сумісної, 32-бітової, багатозадачної, розрахованої на багато користувачів, високопродуктивної операційної системи реального часу QNX фірми Quantum Software systems (Канада) займає декілька кілобайт пам'яті і забезпечує мінімальний набір функцій. У сучасних операційних системах відрізняють такі види ядер.

Наноядро. Украй спрощене і мінімальне ядро, виконує лише одне завдання – обробку апаратних переривань, генерованих облаштуваннями комп'ютера. Після обробки посилає інформацію про результати обробки вищерозміщеному програмному забезпеченню.

Мікроядро надає тільки елементарні функції управління процесами і мінімальний набір абстракцій для роботи з устаткуванням. Велика частина роботи

здійснюється за допомогою спеціальних призначених для користувача процесів, що називаються сервісами.

Екзоядро надає лише набір сервісів для взаємодії між додатками, а також необхідний мінімум функцій, пов'язаних із захистом, виділенням і вивільненням ресурсів, контролем прав доступу тощо.

Монолітне ядро надає широкий набір абстракцій устаткування. Усі частини ядра працюють в одному адресному просторі. Монолітне ядро вимагає перекомпіляції при зміні складу устаткування. Компоненти операційної системи є не самостійними модулями, а складовими частинами однієї програми. Монолітне ядро продуктивніше, ніж мікроядро, оскільки працює як один великий процес.

Модульне ядро – сучасна, вдосконалена модифікація архітектури мікроядра. На відміну від «класичного» монолітного ядра, модульні ядра не вимагають повної перекомпіляції ядра при зміні складу апаратного забезпечення комп'ютера. Замість цього вони надають той або інший механізм підвантаження модулів, що підтримують те або інше апаратне забезпечення (наприклад, драйверів).

Гібридне ядро – це модифіковані мікроядра, що дозволяють для прискорення роботи запускати «несуттєві» частини в просторі ядра. Прикладом гібридного підходу може служити можливість запуску операційної системи з монолітним ядром під управлінням мікроядра. Так влаштовані 4.4BSD і MkLinux, засновані на мікроядрі Mach.

Ядро Windows NT. 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 Мб), щоб носити приставку «мікро». Компоненти ядра Windows NT розташовуються в пам'яті, що витісняється, і взаємодіють 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.

3.2.6 Об'єктно-орієнтований підхід

Хоча технологія мікроядер і заклала основи модульних систем, вона не змогла в повній мірі забезпечити можливості розширення систем. Нині цій меті найбільше відповідає об'єктно-орієнтований підхід, при якому кожен програмний компонент є функціонально ізольованим від інших.

Основним поняттям цього підходу є «об'єкт». Об'єкт – це одиниця програм і даних, що взаємодіє з іншими об'єктам за допомогою прийому і передачі повідомлень. Об'єкт може бути представленням як прикладної програми або документу, так і деяких абстракцій – процесу, події.

Програми (функції) об'єкта визначають перелік дій, які можуть бути виконані над даними цього об'єкта. Об'єкт-клієнт може звернутися до іншого об'єкта, послави сполучення із запитом на виконання якої-небудь функції об'єкта-сервера.

Об'єкти можуть описувати сутності, які вони представляють, з різною мірою деталізації. Для забезпечення спадкоємності при переході до детальнішого опису розробникам пропонується механізм спадкоємства властивостей вже існуючих об'єктів, тобто механізм, що дозволяє породжувати конкретніші об'єкти із загальних. Наприклад, за наявності об'єкту «текстовий документ» розробник може легко створити об'єкт «текстовий документ у форматі Word», додавши відповідну властивість до базового об'єкта. Механізм спадкоємства дозволяє створити ієрархію об'єктів, в якій кожен об'єкт нижчого рівня набуває всіх властивостей свого предка (прабатька).

Внутрішня структура даних об'єкту прихована від спостереження. Не можна довільно змінювати дані об'єкта. Для того щоб отримати дані з об'єкта або помістити дані в об'єкт, необхідно викликати відповідні об'єктні функції. Це ізолює об'єкт від того коду, який використовує його. Розробник може звертатися до функцій інших об'єктів, або будувати нові об'єкти шляхом наслідування властивостей інших об'єктів, нічого не знаючи про те, як вони сконструйовані. Ця властивість називається інкапсуляцією.

Таким чином, об'єкт з'являється для зовнішнього світу у вигляді «чорного ящика» з певним інтерфейсом. З точки зору розробника, що використовує об'єкт, поки зовнішня реакція об'єкта залишається без змін, не мають значення ніякі зміни у

внутрішній реалізації. Це дає можливість легко замінювати одну реалізацію об'єкта іншою. наприклад, у разі зміни апаратних засобів. З іншого боку, здатність об'єктів з'являтися у вигляді «чорного ящика» дозволяє упаковувати в них і представляти у вигляді об'єктів уже існуючі додатки, нічого в них не змінюючи.

Повністю об'єктно-орієнтовані операційні системи дуже привабливі для системних програмістів, оскільки, використовуючи об'єкти системного рівня, програмісти зможуть залізати вглиб операційних систем для пристосування їх до своїх потреб, не порушуючи цілісність системи. Об'єктно-орієнтований підхід є однією з найперспективніших тенденцій в конструюванні програмного забезпечення. Він був прийнятий на озброєння багатьма відомими фірмами, такими як Microsoft, Apple, IBM, Novell/USL (UNIX Systems Laboratories) і Sun Microsystems – усі вони розгорнули свої операційні системи в цьому напрямі.