Лекція 2

1.3 Системні виклики

Функції ядра операційної системи можуть бути виконані внаслідок виконання в прикладних програмах спеціальних функцій – *системних викликів*.

Більшість ОС можуть працювати в двох режимах:

в режимі користувача (*user mode*), в якому не дозволені привілейовані операції, прямий доступ до пристроїв введення/виведення і деяким керуючим регістрам;

в системному режимі (*system mode*), в якому дозволені привілейовані операції, прямий доступ до пристроїв введення/виведення і до всіх керуючих регістрів. Системний режим також називається режимом ядра, або режимом супервізора.

Необхідність системних викликів виникає тому, що прикладні програми не в змозі самотужки визначити, за якими адресами знаходяться функції ядра.

Системний виклик в один з машинно-залежних способів реалізує механізм отримання адрес функцій ядра та передачу в ці функції необхідних параметрів системного виклику, а також отримання результату системного виклику. Найчастіше системні виклики забезпечуються через систему переривань, завдяки чому адреса функції ядра не тільки обраховується апаратно (в процесі обробки переривання), але й забезпечується захист інформаційних ресурсів ядра. Системні виклики найчастіше мають синтаксис функції мови програмування, якою написано ядро ОС.

1.4 Поняття операційного і програмного середовища

Образно можна сказати, що апаратура комп'ютера надає «сиру» обчислювальну потужність, а завдання операційної системи полягає в тому, щоб зробити використання цієї обчислювальної потужності доступним і, по можливості, зручним для користувача. Програміст може не знати деталі управління конкретними ресурсами (наприклад, диском) комп'ютера і повинен звертатися до операційної системи з відповідними викликами, щоб отримати від неї необхідні сервіси та функції. Цей набір сервісів і функцій і є операційним середовищем, в якому виконуються прикладні програми.

Таким чином, о***перацiйне середовище*** – це інтерфейси, що необхідні програмам і користувачам для отримання сервісів ОС. Операційне середовище включає в себе ***програмне середовище*** та інтерфейс користувача.

Програмне середовище, що безпосередньо утворене кодом ОС, називається основним. Окрім нього, в ОС можуть бути організовані додаткові програмні середовища шляхом емуляції іншої операційної системи. Таким чином можливо забезпечити виконання програм, що створені для інших операційних систем.

Звернення користувачів (програм) до функцій ОС відбувається за допомогою систем запитів або викликів.

Для кожної ОС характерним є свій набір системних викликів, який являє собою сукупність системних викликів і правила їх застосування (***API***, Application Programming Interface – ***Інтерфейс прикладного програмування***).

Програма, що створена для однієї ОС, скоріш за все не буде працювати в іншій, оскільки ці ОС мають різні API. Для подолання такого обмеження розробляються програмні середовища.

Програмне середовище – це системне програмне оточення, що дозволяє виконувати системні запити від прикладних програм.

1.5 Архітектура операційних систем

Під архітектурою операційної системи розуміють структурну і функціональну організацію ОС на основі деякої сукупності програмних модулів. До складу ОС входять виконувані і об'єктні модулі стандартних для

даної ОС форматів, програмні модулі спеціального формату (наприклад, завантажувач ОС, драйвери введення/виведення), конфігураційні файли, файли документації, модулі довідкової системи і т.д.

На архітектуру ранніх операційних систем зверталося мало уваги: по-перше, ні в кого не було досвіду в розробці великих програмних систем, а по-друге, проблема взаємозалежності та взаємодії модулів недооцінювалася.

монолітних ОС майже всі процедури могли викликати одна іншу. Така відсутність структури стала несумісною з розширенням операційних систем. Перша версія ОС OS/360 була створена колективом з 5000 робітників за 5 років і містила понад 1 млн рядків коду. Розроблена дещо пізніше операційна система Mastics містила до 1975 року вже 20 млн рядків. Стало зрозумілим, що розробка таких систем повинна вестися на основі модульного програмування.

Більшість сучасних ОС являють собою добре структуровані модульні системи, здатні до розвитку, розширення та переносу на нові платформи. Якої-небудь єдиної уніфікованої архітектури ОС не існує, але відомі універсальні підходи до структурування ОС. Принципово важливими універсальними підходами до розробки архітектури ОС є:

модульна організація;

функціональна надлишковість;

функціональна вибірковість;

параметрична універсальність;

концепція багаторівневої ієрархічної обчислювальної системи, за якою ОС є багатошаровою структурою;

поділ модулів на дві групи за функціями: ядро – модулі, що виконують основні функції ОС, і модулі, що виконують допоміжні функції ОС;

поділ модулів ОС на дві групи за розміщенням в пам'яті обчислювальної системи: *резидентні*, які постійно перебувають в оперативній пам'яті, і *транзитні*, що завантажуються в оперативну пам'ять тільки на час виконання своїх функцій;

реалізація двох режимів роботи обчислювальної системи: привілейованого режиму (режиму ядра – *Kernel mode*), або режиму супервізора (*supervisor mode*), і режиму користувача (*user mode*), або режиму завдання (*task mode*);

обмеження функцій ядра (а отже, і кількості модулів ядра) до мінімальної кількості необхідних найважливіших функцій.

Перші ОС розроблялися як монолітні системи без чітко вираженої структури (рисунок 1.3).

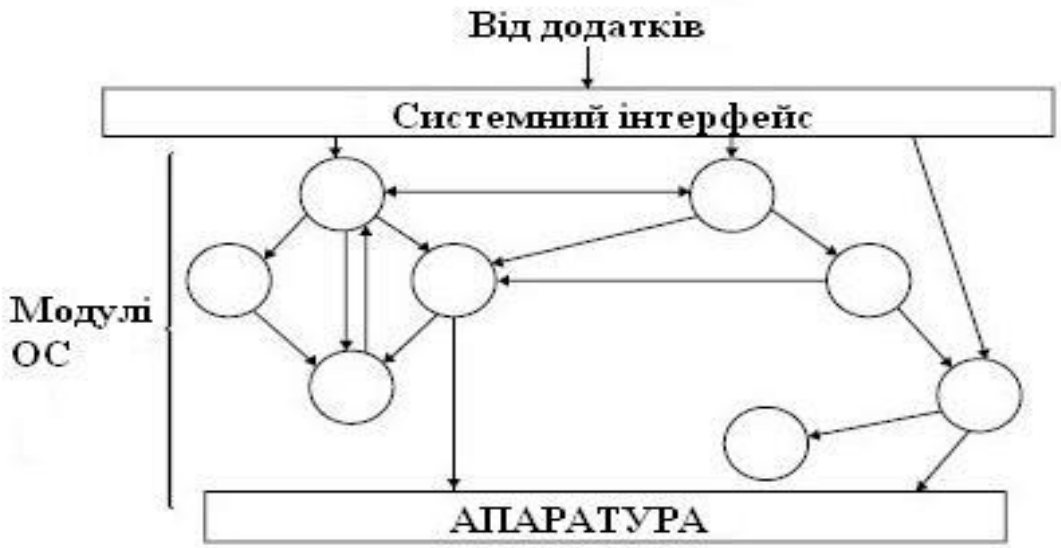

Рисунок 1.3 – Монолітна архітектура

Для побудови монолітної системи необхідно скомпілювати всі окремі процедури, а потім зв'язати їх разом в єдиний об'єктний файл за допомогою компонувальника (прикладами можуть служити ранні версії ядра UNIX або Novell NetWare). Кожна процедура бачить будь-яку іншу процедуру (на відміну від структури, що містить модулі, в якій більша частина інформації є локальною для модуля, і процедури модуля можна викликати тільки через спеціально визначені точки входу).

Але навіть такі монолітні системи можуть бути трохи структурованими. При зверненні до системних викликів, що підтримуються ОС, параметри поміщуються в строго визначені місця, такі як регістри або стек, а потім виконується спеціальна команда переривання, відома як виклик ядра чи виклик супервізора. Ця команда перемикає машину з режиму користувача в режим ядра, званий також режимом супервізора і передає управління ОС. Потім ОС перевіряє параметри виклику, для того щоб визначити, який системний виклик повинен бути виконаний. Після цього ОС індексує таблицю, яка містить посилання на процедури, і викликає відповідну процедуру. Схема структурованої архітектури подана на рисунку 1.4.

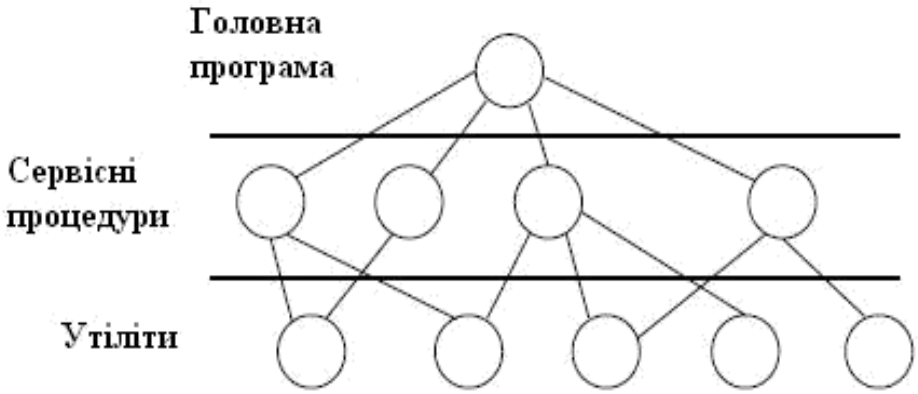

Рисунок 1.4 – Структурована архітектура

цій моделі для кожного системного виклику є одна сервісна процедура. Утиліти виконують функції, які потрібні декільком сервісним процедурам.

Класичною вважається архітектура ОС, заснована на концепції ієрархічної багаторівневої машини, привілейованому ядрі і режимі користувача роботи транзитних модулів. Модулі ядра виконують базові функції ОС: управління процесами, пам'яттю, пристроями введення/виведення і т.п. Ядро складає серцевину ОС, без якої вона є повністю непрацездатною і не може виконати ні одну зі своїх функцій. У ядрі вирішуються внутрішньосистемні задачі організації обчислювального процесу, недоступні для додатку.

Особливий клас функцій ядра служить для підтримки додатків, створюючи для них так зване прикладне програмне середовище. Додатки можуть звертатися до ядра з запитами - системними викликами - для виконання тих чи інших дій, наприклад, відкриття та читання файлу, отримання системного часу, виведення інформації на дисплей і т.д. Функції ядра, які можуть викликатися додатками, утворюють інтерфейс прикладного програмування – API (Application Programming Interface).

Для забезпечення високої швидкості роботи ОС модулі ядра (принаймні, більша їх частина) є резидентними і працюють в привілейованому режимі (Kernel mode). Цей режим, по-перше, повинен забезпечити безпечну роботу самої ОС від втручання додатків, і, по-друге, повинен забезпечити можливість роботи модулів ядра з повним набором машинних інструкцій, що дозволяють власне ядру виконувати управління ресурсами комп'ютера, зокрема, перемикання процесора з задачі на задачу, управління пристроями введення/виведення, розподілом і захистом пам'яті та ін.

Решта модулі ОС виконують не настільки важливі функції, як ядро, є транзитними. Наприклад, це можуть бути програми архівування даних, дефрагментації диска, стиснення дисків, очищення дисків і т.п.

Допоміжні модулі звичайно поділяються на групи:

утиліти – програми, які виконують окремі завдання управління і супроводу обчислювальної системи;

системні опрацьовуючі програми – текстові та графічні редактори (Paint, Imaging в Windows), компілятори та ін;

програми надання користувачу додаткових послуг (спеціальний варіант для користувача інтерфейсу, калькулятор, ігри, засоби мультимедіа Windows);

бібліотеки процедур різного призначення, що спрощують розробки додатків, наприклад, бібліотека функцій введення/виведення, бібліотека математичних функцій і т.п.

Ці модулі ОС оформляються як звичайні додатки, звертаються до функцій ядра за допомогою системних викликів і виконуються в режимі користувача (user mode). У цьому режимі забороняється виконання деяких команд, які пов'язані з функціями ядра ОС (управління ресурсами, розподіл і захист пам'яті і т.п.).

концепції багаторівневої (багатошарової) ієрархічної машини структура ОС також представляється рядом шарів. При такій організації кожен шар обслуговує вищерозміщений шар, виконуючи для нього деякий набір функцій, які утворюють міжшаровий інтерфейс. На основі цих функцій наступний верхній за ієрархією шар будує свої функції – більш складні і більш потужні і т.д. Така організація системи істотно спрощує її розробку, тому дозволяє спочатку «зверху вниз» визначити функції шарів і міжшарові інтерфейси, а при детальній реалізації, рухаючись «знизу вгору», – нарощувати потужність функцій шарів. Крім того, модулі кожного шару можна змінювати без необхідності змін в інших шарах (але не змінюючи міжшарових інтерфейсів!).

Багатошарова структура ядра ОС може бути представлена, наприклад, варіантом, показаним на рисунку 1.5.

Рисунок 1.5 – Багатошарова структура ОС

У даній схемі виділені наступні шари.

Засоби апаратної підтримки ОС. Як правило, в сучасних системах завжди є засоби апаратної підтримки ОС, які прямо беруть участь в організації обчислювальних процесів. До них відносяться: система переривань, засоби підтримки привілейованого режиму, засоби підтримки віртуальної пам'яті, системний таймер, засоби перемикання контекстів процесів (інформація про стан процесу в момент його припинення), засоби захисту пам'яті та ін.

Машинно-залежні модулі ОС. Відображається специфіка апаратної платформи комп'ютера. Призначення цього шару - "екранування" верхніх шарів ОС від особливостей апаратури (наприклад, у Windows - це шар HAL (Hardware Abstraction Layer), рівень апаратних абстракцій).

Базові механізми ядра. Цей шар модулів виконує найбільш примітивні операції ядра: програмне переключення контекстів процесів, диспетчеризацію переривань, переміщення сторінок між основною пам'яттю і диском і т.п. Модулі цього шару не приймають рішень про розподіл ресурсів, а тільки обробляють рішення, прийняті модулями вищерозміщених рівнів.

Менеджери ресурсів. Модулі цього шару виконують стратегічні завдання управління ресурсами обчислювальної системи. Це менеджери (диспетчери) процесів введення/виведення, оперативної пам'яті і файлової системи. Кожен менеджер веде облік вільних і використовуваних ресурсів і планує їх розподіл відповідно запитам додатків.

Інтерфейс системних викликів. Це верхній шар ядра ОС, який взаємодіє з додатками і системними утилітами, він утворює прикладний програмний інтерфейс ОС. Функції API, що обслуговують системні виклики, надають доступ до ресурсів системи в зручній компактній формі, без зазначення деталей їхнього фізичного розташування.

Для підвищення стійкості і надійності ОС в сучасних системах передбачений привілейований режим ядра, при якому відбувається деяке уповільнення виконання системних викликів. Системний виклик привілейованого ядра ініціює перемикання процесора з режиму користувача в привілейований, а при поверненні до додатка – зворотнє перемикання. Недоліком такого підходу є виникнення додаткової затримки в обробці системного виклику, але таке рішення стало класичним і використовується в багатьох ОС (UNIX, VAX, VMS, IBM OS/390, OS / 2 та ін.)

Недоліки багатошарової класичної багаторівневої архітектури ОС: значні зміни одного з рівнів можуть призвести до непередбачуваного впливу на суміжні рівні; 2) численні взаємодії між сусідніми рівнями ускладнюють забезпечення безпеки.

Тому, як альтернатива класичному варіанту архітектури ОС, часто використовується ***мікроядерна архітектура ОС***. При такій архітектурі ОС у привілейованому режимі залишається працювати тільки дуже невелика частина ОС, що називається *мікроядром.*

Мікроядро захищене від інших частин ОС і додатків. До його складу входять машинно-залежні модулі, а також модулі, що виконують базові механізми звичайного ядра. Всі інші більш високорівневі функції ядра оформляються як модулі, що працюють в режимі користувача. Так, менеджери ресурсів, які є невід'ємною частиною звичайного ядра, стають «периферійними» модулями, що працюють в режимі користувача. Таким чином, в архітектурі з мікроядром традиційне розташування рівнів по вертикалі замінюється горизонтальним (рисунок 1.6).

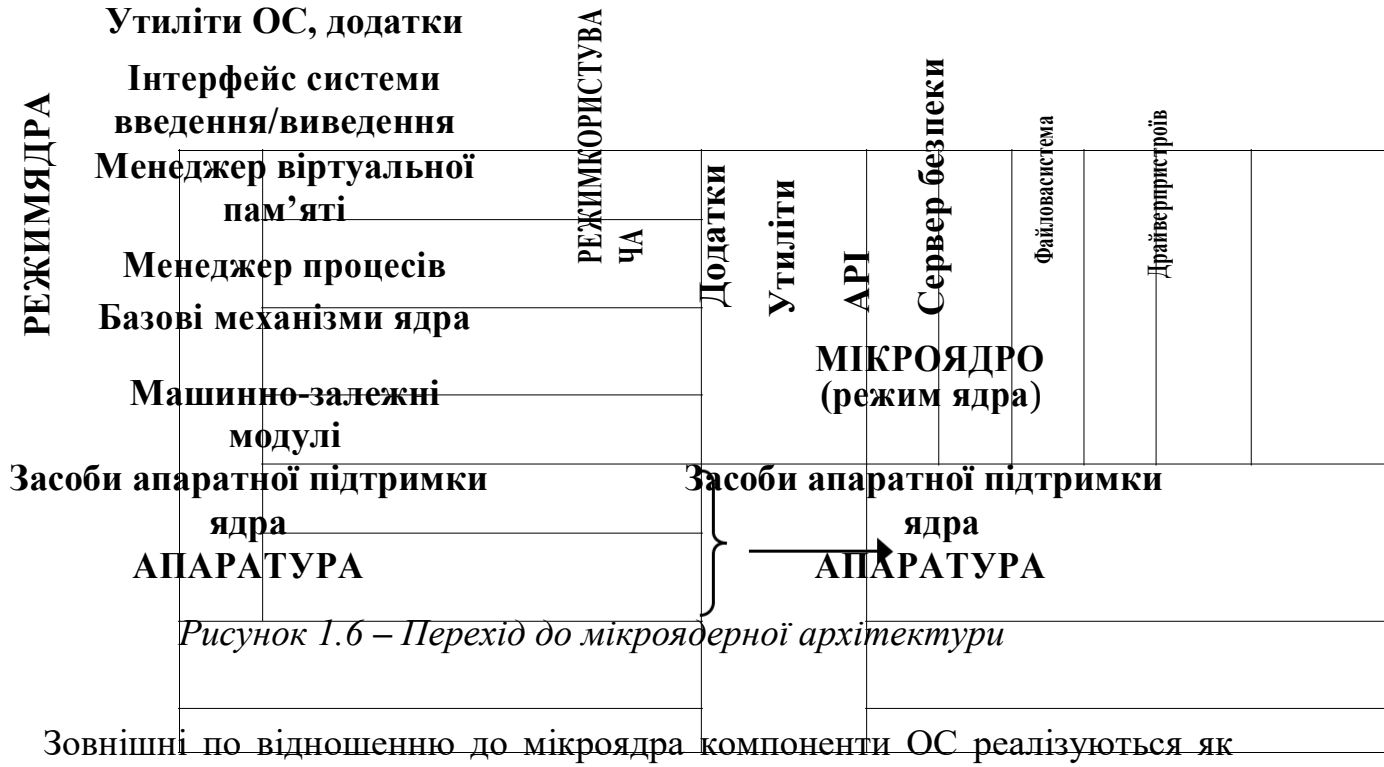

Рисунок 1.6 – Перехід до мікроядерної архітектури

Зовнішні по відношенню до мікроядра компоненти ОС реалізуються як обслуговуючі процеси. Між собою вони взаємодіють як рівноправні партнери за допомогою обміну повідомленнями, які передаються через мікроядро. Оскільки призначенням цих компонентів ОС є обслуговування запитів додатків користувачів, утиліт і системних обробляючих програм, менеджери ресурсів, винесені в користувацький режим, називаються серверами ОС, тобто модулями, основним призначенням яких є обслуговування запитів локальних додатків і інших модулів ОС.

Механізм звернення до функцій ОС у вигляді серверів формує клієнт-серверну архітектуру (рисунок 1.7).

Рисунок 1.7 – Клієнт-серверна архітектура

багатьох джерелах визнані переваги мікроядерної архітектури: однотипні інтерфейси, простота розширюваності, висока гнучкість, висока надійність, підтримка розподілених систем, підтримка об'єктно-орієнтованих ОС.

В сучасних операційних системах розрізняють наступні ***види ядер***.

Наноядро. Украй спрощене і мінімальне ядро, виконує лише одну задачу – обробку апаратних переривань, що генеруються пристроями комп'ютера. Після обробки посилає інформацію про результати обробки вищерозміщеному програмному забезпеченню. Наноядро використовується для віртуалізації апаратного забезпечення реальних комп'ютерів.

Мікроядро надає тільки елементарні функції управління процесами і мінімальний набір абстракцій для роботи з обладнанням. Велика частина роботи здійснюється за допомогою спеціальних користувацьких процесів – сервісів. У микроядерній операційній системі можна, не перериваючи її роботу, завантажувати і вивантажувати нові драйвери, файлові системи і т.д. Мікроядерними є ядра ОС Minix та GNU Hurd і ядро систем сімейства BSD. Класичним прикладом мікроядерної системи є Symbian OS. Це приклад поширеної і відпрацьованої микроядерної (починаючи с версії Symbian OS v8.1, наноядерної) операційної системи.

Екзоядро надає лише набір сервісів для взаємодії між додатками, а також необхідний мінімум функцій, пов'язаних із захистом: виділення і вивільнення ресурсів, контроль прав доступу і т.д. Екзоядро не займається наданням абстракцій для фізичних ресурсів – ці функції виносяться в бібліотеку користувацького рівня (так звану libOS). Мікроядра ОС, що базуються на екзоядрі, забезпечують більшу ефективність за рахунок відсутності необхідності перемиканні між процесами при кожному зверненні до обладнання.

Монолітне ядро надає широкий набір абстракцій обладнання. Всі частини ядра працюють в одному адресному просторі. Монолітне ядро вимагає перекомпіляції при зміні складу обладнання. Компоненти операційної системи не самостійними модулями, а складовими частинами однієї програми. Таке ядро більш продуктивне, ніж мікроядро, оскільки працює як один великий процес. Монолітним ядром є більшість Unix-систем і Linux. Монолітність ядер ускладнює налагодження, розуміння коду ядра, додавання нових функцій і можливостей, видалення непотрібного, успадкованого від попередніх версій

коду. «Розбухання» коду монолітних ядер також підвищує вимоги до обсягу оперативної пам'яті.

Модульне ядро – сучасна, вдосконалена модифікація архітектури мікроядра. На відміну від «класичних» ОС з архітектурою мікроядра, модульні ядра не вимагають повної перекомпіляції ядра при зміні складу апаратного забезпечення комп'ютера. Замість цього вони надають той чи інший механізм підвантаження модулів, що підтримують те чи інше апаратне забезпечення (наприклад, драйверів). Підвантаження модулів може бути як динамічним, так і статичним (при перезавантаженні ОС після переконфігурації системи). Модульне ядро зручніше для розробки, ніж традиційні монолітні ядра. Вони надають програмний інтерфейс (API) для зв'язування модулів з ядром, для забезпечення динамічного підвантаження і вивантаження модулів. Не всі частини ядра можуть бути зроблені модулями. Деякі частини ядра завжди зобов'язані бути присутніми в оперативній пам'яті і повинні бути жорстко «вшиті» в ядро.

Гібридне ядро - модифіковані мікроядра, що дозволяють для прискорення роботи запускати "несуттєві" частини в просторі ядра. Мають «гібридні» переваги і недоліки. Прикладом змішаного підходу може служити можливість запуску операційної системи з монолітним ядром під управлінням мікроядра. Мікроядро забезпечує управління віртуальною пам'яттю і роботу низькорівневих драйверів. Всі інші функції, в тому числі взаємодія з прикладними програмами, здійснюються монолітним ядром. Даний підхід сформувався в результаті спроб використати переваги мікроядерної архітектури, зберігаючи по можливості добре налагоджений код монолітного ядра.

Найбільш тісно елементи мікроядерної архітектури та елементи монолітного ядра переплетені в ядрі Windows NT. Хоча Windows NT часто називають микроядерною операційною системою, це не зовсім так. Мікроядро NT надто велике (більше 1 Мбайт), щоб носити приставку «мікро». Компоненти ядра Windows NT розташовуються в пам'яті і взаємодіють один з одним шляхом передачі повідомлень, як і належить в мікроядерних операційних системах. У той же час всі компоненти ядра працюють в одному адресному просторі і активно використовують загальні структури даних, що властиво операційним системам з монолітним ядром.

1.5.1 Архітектура ОС Windows

На рисунку 1.8 представлена загальна структура Windows.[2]

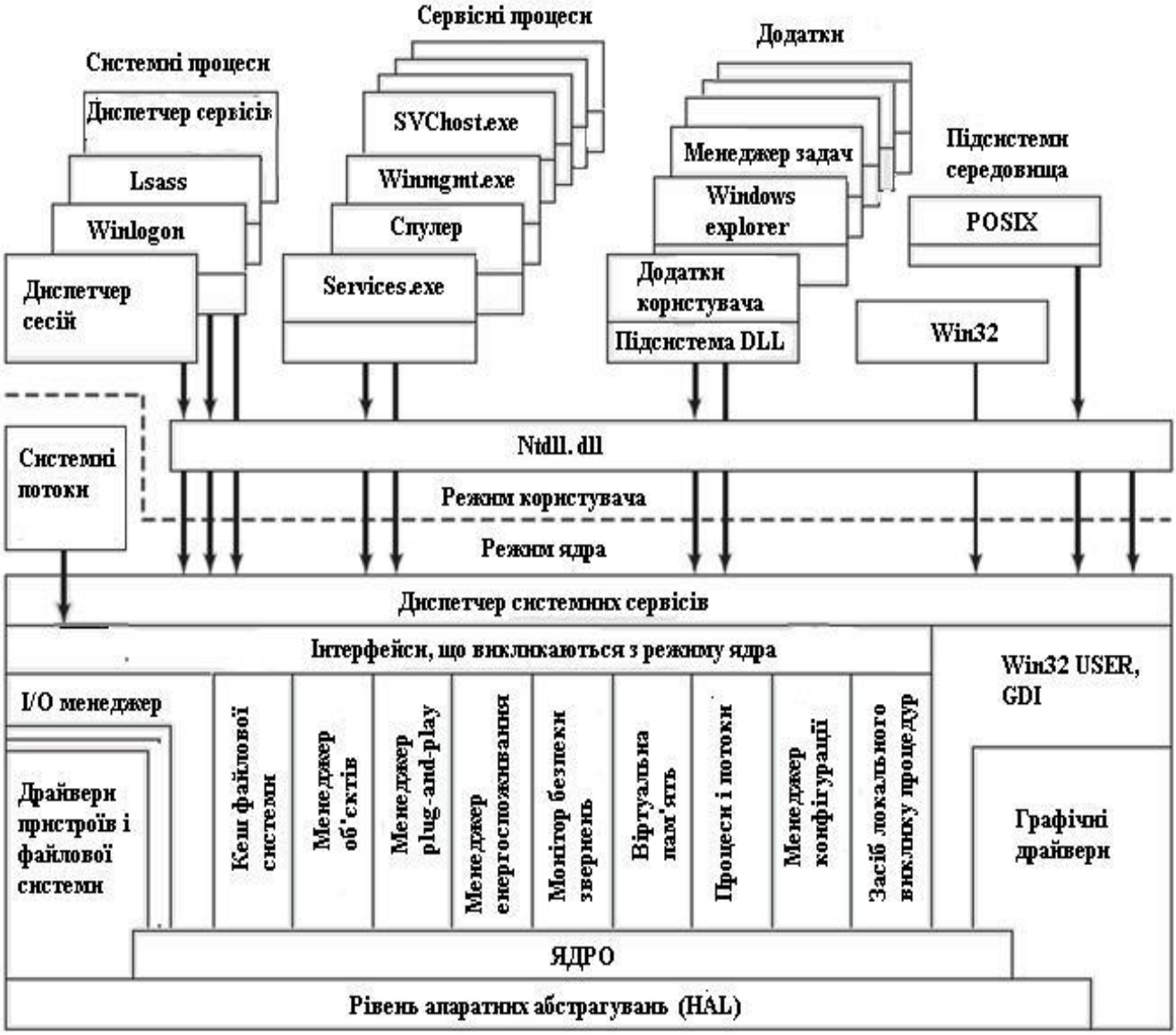

Рисунок 1.8 – Архітектура Windows

Всі версії Windows основані на NT версії і мають в головному схожу структуру на цьому рівні деталізації. У Windows відділено програмне забезпечення, що орієнтоване на користувача ОС від програмного забезпечення ядра ОС. Останнє, яке включає виконавчу систему (Windows Executive), ядро, драйвери пристроїв, рівень абстрагування від устаткування (HAL, hardware abstraction layer), запускається в режимі ядра. Програмне забезпечення режиму

Stallings, William. Operating systems: internals and design principles / William Stallings. – 7th ed. Prentice Hall, New Jersey, 2012. p.769.
ISBN-13:978-0-13-230998-1

ядра ОС має доступ до системних даних і до устаткування комп’ютера. Все інше програмне забезпечення, яке запускається в режимі користувача, має обмежений доступ до системних даних.

Організація ОС Windows має високорозвинену модульну архітектуру, якій кожна системна функція є керованою тільки одним компонентом ОС, а вся інша частина ОС і всі додатки отримують доступ до цієї функції лише через дублюючий компонент, який використовує стандартні інтерфейси. Доступ до ключових системних даних може бути отримано лише через відповідні функції.

принципі, будь-який модуль може бути вилучено, модифіковано чи переміщено без перезапису всієї системи, або її стандартних програмних інтерфейсів додатків (API, Application Programming Interface).

Як бачимо із схеми на рисунку 1.8 до компонентів режиму ядра відносяться:

виконавча система, що складається з шару сервісів ОС, таких як управління пам’яттю, управління процесами і потоками, безпеки, системи введення/виведення;

рівень апаратних абстракцій, який відокремлює ОС від особливостей апаратної платформи і тому системна шина, контролер прямого доступу до пам’яті, контролер переривань, системні таймери і пам’ять з точки зору ядра виглядають однаково;

ядро Windows відповідає за розподіл ресурсів між процесами, їх перемикання і синхронізацію; основним завданням ядра є якомога ефективніше завантаження процесорів системи. Ядро постійно перебуває в пам'яті, послідовність виконання його інструкцій може порушити тільки переривання (під час виконання коду ядра багатозадачність не підтримується);

драйвери пристроїв можуть бути драйвери не тільки апаратних пристроїв, але і застосування, якому потрібні засоби, доступні в режимі ядра, і які завжди варто оформляти як драйвер;

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

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

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

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

Засіб локального виклику процедур (ALPC, advanced local procedure call) – забезпечує механізм зв'язку між процесами і підсистемами на одному комп'ютері.

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

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

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

Менеджер кеша – керує кешуванням для системи введення/виведення. Часто використовувані блоки диска тимчасово зберігаються в пам'яті, наступні операції введення/виведення звертаються до цієї пам'яті, внаслідок чого підвищується продуктивність.

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

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

- *Менеджер plug-and-play* – менеджер підключення, відключення, налаштування пристроїв, які підтримують технологію Plug-and-Play.

1.5.2 Особливості сучасних архітектур ОС UNIX і Linux

Архітектура ОС UNIX. Сучасна ОС UNIX має порівняно просту архітектуру. В ядрі знаходиться найбільш вживані функції, що забезпечують роботу ОС і користувача. Ядро в свою чергу взаємодіє з прикладними програмами за допомогою системних викликів. Сучасна структура ядра UNIX може бути представлена, як зображено на рисунку 1.9. В архітектурі сучасних

UNIX-систем виділено окремий менеджер пам'яті, відповідальний за підтримку віртуальної пам'яті.

Рисунок 1.9 – Ядро сучасної системи UNIX

Стандартом для реалізації інтерфейсу файлової системи є віртуальна файлова система, що абстрагує цей інтерфейс і дає змогу організувати підтримку різних типів файлових систем. У цих системах підтримується багатопроцесорна обробка, а також багатопотоковість. Базові архітектурні рішення, такі як доступ до всіх пристроїв введення/виведення через інтерфейс файлової системи або організація системних викликів, залишаються незмінними в усіх реалізаціях UNIX.

Архітектура Linux. Зауважимо, що архітектура Linux організована за технологією монолітного ядра, тобто структура даних ядра і код знаходяться в єдиному адресному просторі.

На рисунку 1.10 показані головні компоненти ядра Linux, які впроваджені IA-64 архітектуру (тобто, Intel Itanium). Прямокутники з хвилястими лініями, що у верхній частині схеми - це окремі процеси з потоками в них.

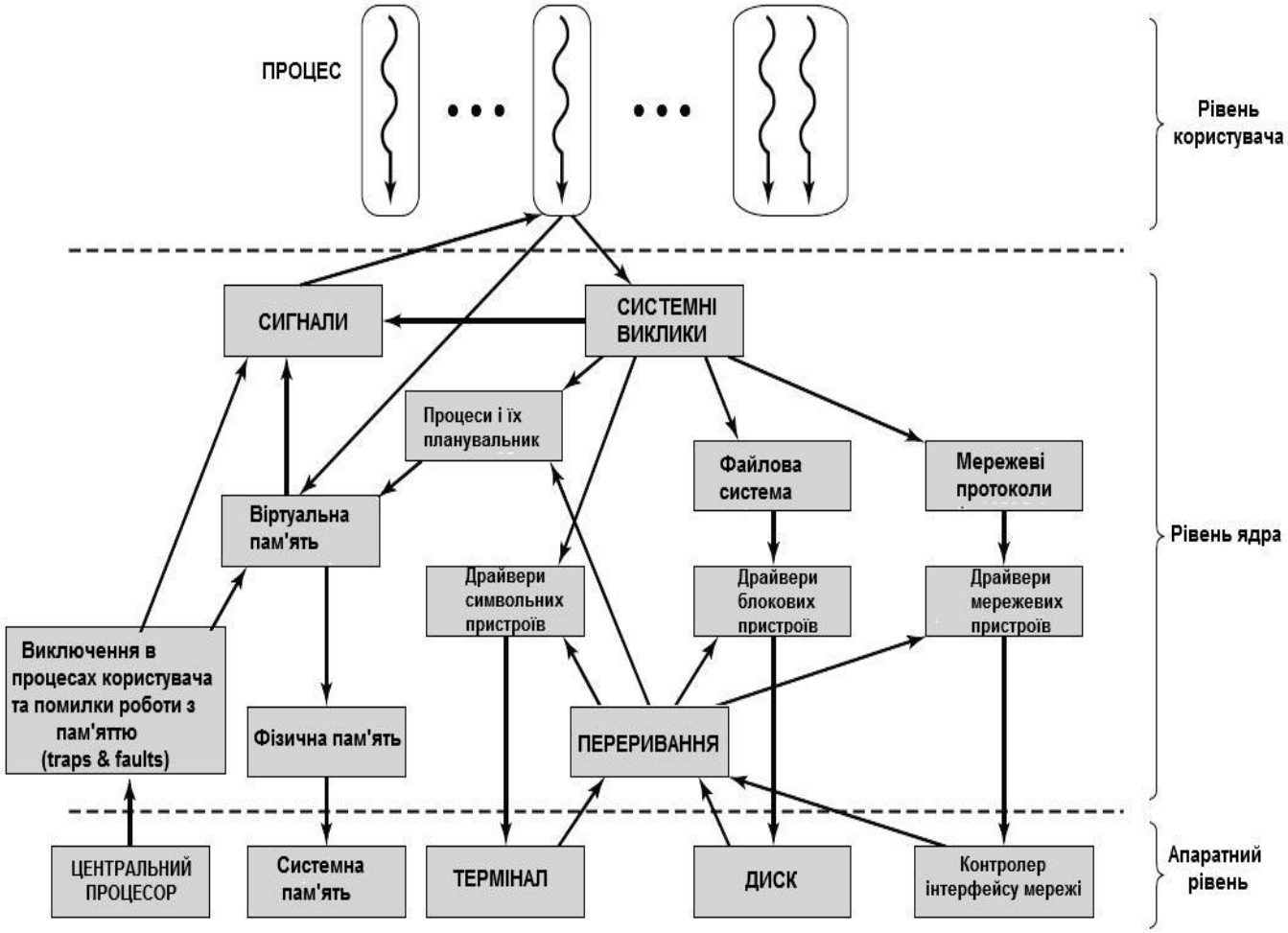

Рисунок 1.10 – Головні компоненти ядра Linux

Саме ядро складається з набору взаємодіючих компонентів між якими показано основні взаємодії. Нижче розміщена апаратна частина, що представлена як набір компонентів. Стрілками показано, які компоненти ядра використовують або керують компонентами апаратури. Всі компоненти ядра виконуються процесором, але це на схемі не показано.

Основні функціональні компоненти ядра:

Менеджер пам'яті – організація адресного простору і віртуальної пам'яті.

Планувальник процесів – реалізує підтримку багатозадачності в операційній системі: робота з таймером, створення процесів, завершення процесів і ін.

Міжпроцесова взаємодія – механізми, що забезпечують обмін даними між процесами.

Віртуальна файлова система – інтерфейс взаємодії з різними файловими системами та пристроями введення/виведення. Забезпечує доступ до д*райверів пристроїв, що* забезпечують безпосередню роботу з периферійними пристроями.

Мережний інтерфейс – забезпечує доступ до реалізації мережних протоколів і драйверів мережних пристроїв.

В Linux підтримується так звана концепція *модулів ядра*, тобто комплекс незалежних блоків. Ці блоки володіють, по-перше, *динамічним зв'язуванням*, при якому на вимогу завантажуються у пам'ять і вивантажуються з неї окремі секції коду. Такі секції (модулі) виконуються у привілейованому режимі. І, по-друге, с*тековою організацією,* при якій модулі організовані у вигляді певної ієрархічної структури. Окремі модулі виконують роль бібліотек при зверненні до них модулів більш високих рівнів в межах цієї структури, і вони самі можуть звертатися до модулів на більш низьких рівнях.

Для підтримки модулів у Linux використовують певні засоби: засоби, що завантажують модулі у пам'ять із здійсненням обміну даними між модулями та іншою частиною ядра; реєстрація драйверів для попередження іншої частини ядра про те, що новий драйвер став доступним; резервування апаратних ресурсів для захисту драйверів пристроїв від випадкового використання їх іншими драйверами.

1.6 Операційна система Android

Операційна система Android – це система на основі Linux, спочатку розроблена для мобільних телефонів. Це найпопулярніша мобільна ОС з великим відривом: телефони Android перевершують iPhone від Apple в світі приблизно в 4 рази [Morra, J. "Google Rolls Out New Version of Android Operating System."Electronic Design, August 24, 2016.]. Але це тільки один з елементів зростаючого домінування Android. Все частіше ця ОС стоїть практично за будь-яким пристроєм з комп'ютерним чіпом, крім серверів і ПК. Android – широко використовувана ОС для Інтернету речей.

Первісна розробка ОС Android була здійснена компанією Android, Inc., яка була куплена Google в 2005 році. Перша комерційна версія, Android 1.0,

була випущена в 2008 році. На момент написання цієї статті самою останньою версією є Android 7.0 (Nougat). Android має активне співтовариство розробників ентузіастів, які використовують вихідний код Android Open Source Project (AOSP) для розробки і поширення власних модифікованих версій операційної системи. Відкритий характер Android був ключем до його успіху.

1.6.1 Архітектура програмного забезпечення Android

Android визначається як програмний стек, який включає модифіковану версію ядра Linux, проміжне програмне забезпечення і ключові програми. На рисунку 1.11 подано архітектуру програмного забезпечення Android в деяких деталях. Таким чином, Android слід розглядати як повний програмний стек, а не просто як ОС.

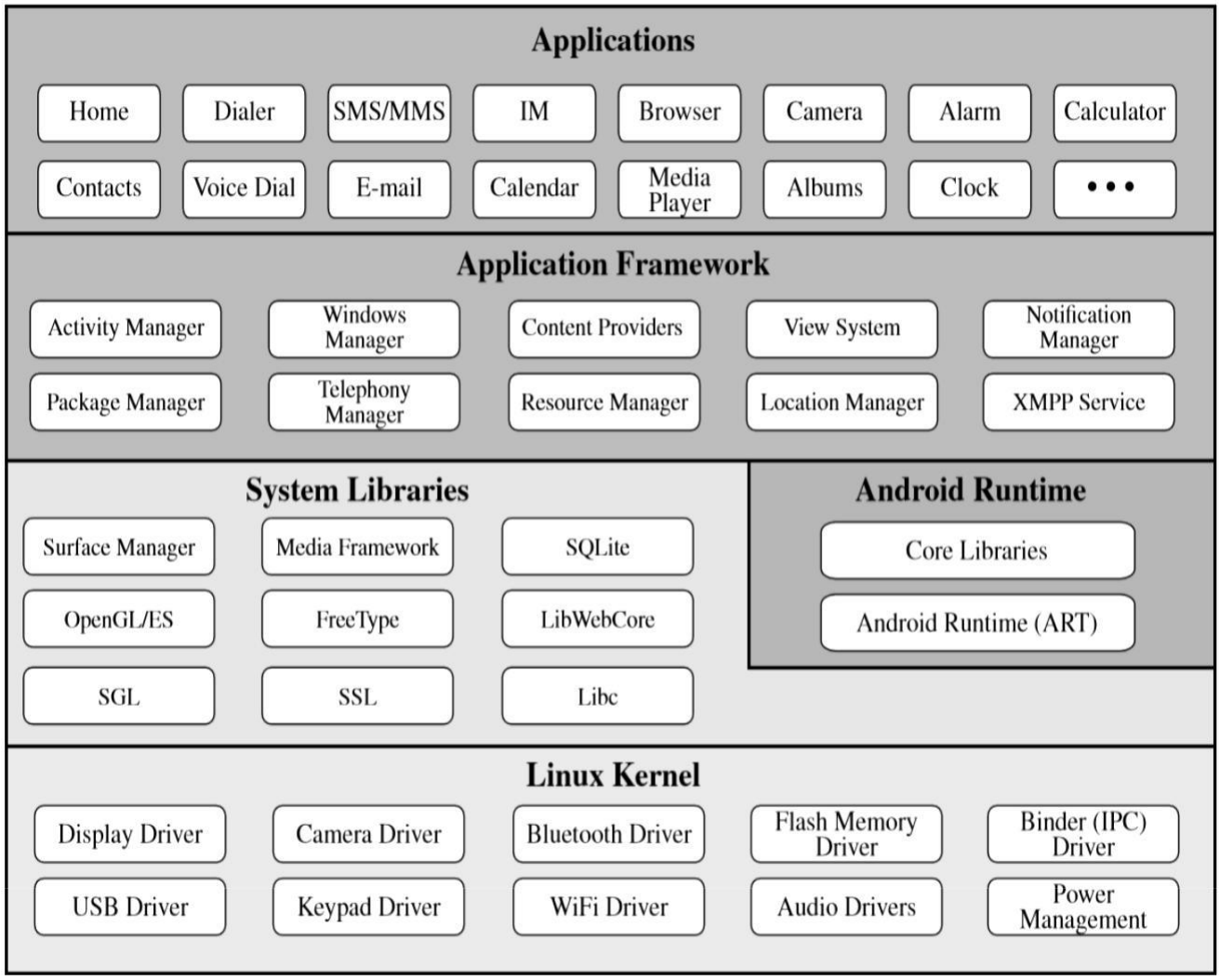

Рисунок 1.11 – Архітектура програмного забезпечення Android

Пояснення до рисунку 1.11:

Applications – рівень додатків;

Applications Framework – рівень спеціалізованих бібліотек для додатків; System Libraries – системні бібліотекі;

Android Runtime – середовище виконання Android-додатків з їх компіляцією під час установки;

Linux Kernel – ядро ОС Linux.

Додатки. Всі додатки, з якими користувач взаємодіє безпосередньо, частиною рівня додатків. Сюди входить основний набір додатків загального призначення, таких як поштовий клієнт, програма SMS, календар, карти, браузер, контакти та інші додатки, які зазвичай використовуються на будь-якому мобільному пристрої. Додатки зазвичай реалізуються на Java. Основна мета архітектури Android з відкритим вихідним кодом полягає в тому, щоб полегшити розробникам реалізацію нових додатків для конкретних пристроїв і конкретних вимог кінцевого користувача. Використання Java дозволяє розробникам позбутися специфічних для апаратного забезпечення міркувань і особливостей, а також використовувати більш широкі можливості мови Java, такі як зумовлені класи. Приклади типів базових додатків на платформі Android: Google Keep – нотатки та списки; Facebook, WhatsApp Messenger, Instagram; Google Новини; Microsoft Word: правка документів та загальний доступ; Google Документи та ін.

1.7 Основні етапи розвитку операційних систем

До середини 1950-х років перші ЕОМ ще не були забезпечені операційними системами.

На початку 1960-х вони лише комплектувались набором інструментів для розробки, планування та виконання завдань і до кінця 1960-х було розроблено цілий ряд операційних систем. Найбільш розвинуті ОС того часу, такі як «OS/360» (компанія «IBM»), «SCOPE» (компанія «CDC») та завершений вже в 1970-х роках «MULTICS» (MTI та компанія «Bell Labs»), передбачали можливість використання багатопроцесорних систем.

Спонтанний характер розробки ОС призвів до наростання кризових явищ, пов'язаних, перш за все, зі складністю та великими розмірами розроблюваних

систем. ОС погано масштабувались (простіші не використовували всіх можливостей потужних обчислювальних машин; складніші неоптимально виконувались або взагалі не виконувались на менш потужних системах) і були повністю несумісними між собою.

1969 році співробітники МТІ Кен Томпсон, Деніс Рітчі та Браян Керніган з колегами розробили та реалізували ОС «Юнікс» («Unix»; первинно «UNICS», на противагу «MULTICS»). Нова ОС увібрала в себе багато рис попередниць, але на противагу їм мала цілий ряд переваг:

- проста метафорика (два ключових поняття - процес та файл);
- компонентна архітектура (принцип «одна програма - одна функція», або інакше «кожна програма має робити лише одну роботу, але робити її добре» плюс потужні засоби об'єднання цих програм для розв'язання конкретних задач);
  мінімізація ядра та кількості системних викликів;
  незалежність від апаратної архітектури і реалізація на машино незалежній мові програмування (для цього була розроблена мова програмування «C»);
  уніфікація файлів (будь-що у системі є файлом, до якого можна отримати доступ за спільними для всіх правилами).

Завдяки зручності перш за все в якості інструментального середовища Unix дуже тепло зустріли в університетах, а потім і в галузі. В цілому і незабаром вона стала прототипом єдиної ОС, котру можна було використовувати у найрізноманітніших обчислювальних системах, і - більше того - швидко та з мінімумом зусиль перенести на іншу апаратну архітектуру.

кінці 1970-х років співробітники Каліфорнійського університету в Берклі внесли ряд суттєвих вдосконалень у джерельні коди Unix, включно з реалізацією стеку мережевих протоколів TCP/IP. Їх розробка стала відомою під іменем BSD (англ. *Berkeley Software Distribution*).

Через конфлікт з «Bell Labs» Річард Столмен поставив задачу реалізувати повністю незалежну від авторських прав ОС на основі Unix, заснувавши проект «GNU» (англ. *рекурсивсне скорочення «GNU's Not Unix»* – «ГНЮ Не Юнікс»).

Незабаром Unix стала стандартом де-факто, а потім і юридичним - ISO/IEC 9945. ОС, що дотримувались цього стандарту чи опираються на нього, називають «відкритими» або «стандартними». До них належать системи, що

базуються на останній версії Unix, випущеної «Bell Labs» («System V»), на розробках Університету Берклі («FreeBSD», «OpenBSD», «NetBSD»), а також ОС «Linux», розроблена спільнотою на чолі з Лінусом Торвальдсом та в межах проекту «GNU» (основні системні інструменти).

Нижче наведені ***основні етапи в розвитку операційних систем.***

Прості пакетні ОС. При роботі з такими ЕОМ завдання на машинних носіях збирали в пакети і поміщали в пристрій вводу даних. Після цього ОС, або монітор, зчитував з пристрою введення/виведення ці завдання по одному і передавала управління наступному зчитаному завданню. По завершенні завдання або при виникненні помилки управління поверталося монітору, який починав зчитувати наступне завдання. Приклади пакетних ОС: FMS (Fortran Monitor System), IBSYS (ОС, створена фірмою IBM для ЕОМ IBM 7094).

Багатозадачні пакетні ОС. Для підвищення ефективності використання процесора необхідно було реалізувати принцип багатозадачності (наприклад, поки одне завдання чекає завершення виконання операції введення/виведення, інше може використовувати процесор). Це можливо, якщо

ОП є достатньо місця і для самої ОС, і для двох (або більше) програм користувача. Для багатозадачних ОС обов'язковим є

наявність контролерів зовнішніх пристроїв, що працюють незалежно від процесора і дозволяють виконувати операції введення/виведення одночасно з командами процесора;

наявність системи переривань, за допомогою яких такі контролери взаємодіють з процесором.

При роботі багатозадачних ОС виникає необхідність в управлінню пам'яттю, в плануванні щодо виконання задач, в диспетчеризації процесів, в захисті завдань від впливу один одного. Приклад багатозадачної пакетної ОС: OS/360 фірми IBM.

ОС з розподілом часу. У цьому випадку декілька користувачів одночасно отримують доступ до системи за допомогою терміналів (термінал - робоче місце користувача: дисплей і клавіатура), а ОС забезпечує почергове виконання програм кожного користувача через малі проміжки часу. Таким чином, кожному користувачеві надається частина процесорного часу ЕОМ, не враховуючи витрат часу на роботу ОС. Приклади ОС з разподілом часу: CTSS (Compatible Time-Sharing System), MULTICS.

Контрольні питання

Що таке операційна система? Які основні функції ОС?

На якому рівні ієрархічної структури програмно-апаратних засобів комп'ютера реалізується виконання машинних команд?

Назвіть сервіси, що надає ОС?

Яким чином можна класифікувати ОС?

Як класифікуються ОС за призначенням? Відповідь деталізуйте.

Які основні складові ОС?

Що являє собою ядро ОС?

Що визначає структура ОС? Назвіть і опишіть основні типи структур ОС.

В яких режимах працюють сучасні ОС?

Що таке операційне і програмне середовище ОС?

Що таке *API*?

Що розуміють під архітектурою ОС? Назвіть універсальні підходи до розробки архітектури ОС?

Який основний недолік монолітної архітектури ОС?

На якій концепції заснована класична архітектура ОС?

Що являє собою багатошарова архітектура ОС?

Які особливості мікроядерної архітектури?

Назвіть компоненти режиму ядра ОС Windows?

Що називається виконавчою системою Windows (Windows Executive)?

Яку технологію ядра реалізує архітектура Linux?

Назвіть і опишіть основні етапи розвитку операційних систем?

Назвіть основні елементи архітектури програмного забезпечення Android.