**Системне програмування і операційні системи.**
**Вступ до курсу.**

Зміст



## 3. Архітектура ядра операційної системи.

Сучасна ОС загального призначення виконує багато різних функцій при тому, що рівні вимог до їх виконання різні. Так забезпечення скоординованого доступу програм до процесора, пам'яті, зовнішніх пристроїв, підтримка взаємодії між окремими процесами та сервісів файлової системи являються критично важливими для функціонування обчислювальної системи як з точки зору надійності, так і максимальної продуктивності. Тоді як, скажімо, реалізація інтерфейсу користувача як правило не є критичним фактором продуктивності обчислень але повинна забезпечити максимальну ефективність роботи оператора. Виходячи з цих міркувань розвинені ОС розробляють у вигляді комплексу програм та програмних модулів одні з яких (обслуговуючі програми – ***утиліти***) виконуються незалежно, а інші — під управлінням ***ядра*** ОС.

Ядро — центральна частина ОС, являє собою набір функцій, структур даних і окремих програмних модулів, які завантажуються в пам'ять комп'ютера при завантаженні ОС і забезпечують, як правило:

- управління введенням-виведенням інформації;
- управління оперативною пам'яттю;
- управління процесами;
- підтримку багатозадачності.

В залежності від ***архітектури*** (базової організації програмної системи, втіленої в її компонентах і відношеннях їх між собою та з оточенням) ядра конкретної ОС набір його функцій може розширюватись, або звужуватись порівняно із вказаним. Але в будь-якому випадку ядро є найбільш низьким рівнем абстракції для доступу програми до ресурсів системи. Можна сказати, що ядро є ***контролером*** ОС.

Перш ніж звертатись до класифікації архітектур ОС розглянемо питання режимів роботи процесора та їх відношення до виконання коду ядра. В сучасних ОС процедури ядра як правило працюють в ***привілейованому режимі*** процесора (інші назви — ***режим ядра, режим супервізора***). В цьому режимі програми мають безпосередній доступ до всіх апаратних ресурсів таких як порти зовнішніх пристроїв, реальні адреси елементів оперативної пам'яті та ін..

На відміну від привілейованого режиму ***захищений режим*** процесора, або ***режим користувача*** ізолює оперативну пам'ять, виділену процесу, від інших процесів. Крім того, доступ до зовнішніх пристроїв в режимі користувача можливий лише через виклики ядра ОС. Всі прикладні програми і утиліти ОС працюють саме в режимі користувача. Однак час від часу прикладний процес вимушений звертатися до сервісів ОС (наприклад для введення чи виведення даних). При цьому виконується послідовність команд системного виклику,

процесор перемикається в привілейований режим і виконується код процедур ядра з подальшим зворотним перемиканням в режим користувача.

Якщо процедури ядра і дані необхідні для обслуговування процесу, що викликав ядро, знаходяться в тому самому адресному просторі що і код самого процесу, то говорять що код ядра виконується в контексті прикладного процесу. Важливо що захист пам'яті працює і ділянки пам'яті, виділені для ядра, недосяжні для прикладних процедур навіть в межах одного процесу.

В ряді ОС процедури ядра виконуються в окремому адресному просторі (***автономне ядро***). В такій схемі поняття процесу застосовується тільки до прикладних процесів, а ядро розглядається як окремий об'єкт що виконується в привілейованому режимі.

### 3.1. Типи архітектур ядра.

Поширена класифікація типів архітектур ядра виглядає так:

- монолітне ядро;
- модульне ядро;
- багаторівневе ядро;
- мікроядро;
- екзоядро;
- наноядро;
- гібридне ядро.

***Монолітне ядро*** — найстаріший спосіб організації ОС який, в дещо зміненому виді, використовується і в наш час. В цій схемі всі компоненти ядра є складовими частинами однієї програми, використовують загальні структури даних і взаємодіють один з одним шляхом безпосереднього виклику процедур. Всі процедури працюють в привілейованому режимі процесора, як правило це автономне ядро. Набір функцій ОС, реалізованих в монолітному ядрі, найбільш широкий з поміж всіх типів архітектур. Реально реалізація всіх постійно необхідних під час роботи сервісів ОС зосереджена в ядрі. На жаль, монолітність ядер ускладнює їх модернізацію та налагодження, неправильна робота окремих процедур ядра несе великий ризик виходу з ладу всієї ОС. Великі розміри монолітних ядер вимагають багато місця для розміщення їх в оперативній пам'яті. Крім того при зміні складу обладнання комп'ютера необхідно заново перекомпілювати ядро.

***Модульне ядро*** — сучасна модифікація архітектури монолітного ядра позбавлена двох останніх його недоліків. Модульні ядра, як правило, не вимагають повної перекомпіляції ядра при зміні складу апаратного забезпечення комп'ютера. Натомість модульні ядра надають той або інший механізм ***підвантаження*** окремих модулів ядра, що підтримують те або інше апаратне забезпечення (наприклад, драйверів). Підвантаження модулів може бути як динамічним без перезавантаження ОС, так і статичним при перезавантаженні ОС після переконфігурації системи на завантаження тих або інших модулів. Всі модулі ядра працюють в адресному просторі ядра і можуть користуватися всіма функціями, що надаються ядром.

Модульні ядра вимагають для своєї роботи менше оперативної пам'яті завдяки можливості мати в пам'яті тільки необхідну в даний момент конфігурацію коду ядра. Вони зручніші для розробки, ніж традиційні монолітні ядра, оскільки від розробника не вимагається багаторазова повна перекомпіляція ядра при роботі над окремою його підсистемою. Виявлення і усунення помилок при тестуванні також полегшуються.

Більшість сучасних UNIX-подібних ОС, таких як Linux, FreeBSD, Solaris мають модульну (частково) структуру ядер і дозволяють під час роботи за потреби динамічно підвантажувати і вивантажувати модулі, що виконують частини функцій ядра.

***Багаторівневе ядро*** є ще одним напрямком розвитку архітектури монолітного ядра. Основною ідеєю є організація ОС як ієрархії рівнів. Рівні утворюються групами функцій операційної системи - файлова система, управління процесами і пристроями і т.п. Кожен рівень може взаємодіяти тільки з своїм безпосереднім сусідом - вище- або нижчележачим рівнем. Прикладні програми або модулі самої ОС передають запити на обробку вгору і вниз по цих рівнях.

Такий структурний підхід дозволив певним чином структурувати ядро і потенційно забезпечити високий рівень захисту системи.

Однак у системах з багаторівневою структурою важко замінити одну реалізацію рівня іншою через множинність і розмитість інтерфейсів між сусідніми рівнями.

Найбільш відомим практичним втіленням архітектури багаторівневого ядра є ОС MULTICS, де ідея ієрархії рівнів системи розповсюджувалась і на прикладні програмні системи. Багаторівневий підхід також використовувався при реалізації ряду варіантів ОС UNIX. Пізніше на зміну йому прийшла модель клієнт-сервер і зв'язана з нею концепція мікроядра.

***Мікроядро*** — це модель ядра з мінімальною функціональністю. Класичні мікроядра надають лише невеликий набір системних викликів, що реалізовують такі базові сервіси ОС:

- управління пам'яттю;
- управління процесами;
- засоби комунікації між процесами.

Решта всіх сервісів ОС, які в класичних монолітних ядрах надаються безпосередньо ядром, в мікроядерній архітектурі реалізуються як процеси в адресному просторі користувача і називаються ***сервісами***. Прикладами таких сервісів, що виносяться в простір користувача в мікроядерній архітектурі, є мережеві сервіси, підтримка файлової системи, драйвери пристроїв.

Така конструкція потенційно дозволяє поліпшити загальну швидкодію системи за рахунок того, що компактне мікроядро може розміщатися в кеші процесора. За рахунок високого ступеня модульності істотно спрощується додавання в ОС нових компонентів. У мікроядерній ОС можна, не перериваючи її роботи, завантажувати і вивантажувати нові драйвери, файлові системи і т.д. Оскільки сервіси ОС нічим принципово не відрізняються від програм користувача, то можна застосовувати звичайні засоби розробки та істотно спростити процес розробки і налагодження компонентів ядра. Мікроядерна архітектура також підвищує надійність системи, оскільки помилка на рівні непривілейованої програми менш небезпечна, ніж відмова на рівні режиму ядра.

В той же час мікроядерна архітектура ОС вносить додаткові накладні витрати. Вони пов'язані з тим, що окремі сервіси, працюючи в захищеному режимі процесора, можуть взаємодіяти між собою лише шляхом передачі повідомлень, а це негативно впливає на продуктивність. Для підвищення швидкодії мікроядерної ОС необхідна ретельна проробка розбиття системи на компоненти з метою мінімізації трафіку повідомлень між ними.

Приклади ОС, що базуються на архітектурі мікроядра — QNX, Window CE, AIX, Minix3, Mac OS X, Symbian OS.

***Екзоядро*** — один з сучасних напрямків подальшого розвитку мікроядерної архітектури. Це ядро ОС, що надає лише функції для взаємодії між процесами і безпечного виділення і звільнення ресурсів. Надання прикладним програмам ***абстракцій*** для фізичних ресурсів не входить в обов'язки екзоядра. Ці функції виносяться в бібліотеку захищеного режиму — так звану libOS, яка може забезпечувати довільний набір абстракцій, сумісний з тією або іншою вже існуючою ОС, наприклад Linux або Windows.

В порівнянні з ОС на основі мікроядер, екзоядра забезпечують набагато більшу ефективність за рахунок відсутності перемикання процесів при кожному зверненні до апаратного устаткування.

***Наноядро*** — архітектура ядра ОС, в рамках якої украй спрощене ядро виконує лише

одне завдання — обробку апаратних переривань, що генеруються пристроями комп'ютера. Після обробки переривань від апаратури наноядро, у свою чергу, посилає інформацію про результати обробки вищерозміщеному програмному забезпеченню за допомогою того ж механізму переривань.

Найчастіше в сучасних обчислювальних системах наноядра використовуються для ***віртуалізації*** апаратного забезпечення з метою дозволити кільком різним ОС працювати одночасно і паралельно на одному і тому ж комп'ютері. Наноядра також використовуються для забезпечення переносимості ОС на різне апаратне забезпечення або для забезпечення можливості запуску ОС на новому, несумісному апаратному забезпеченні без її повного переписування і перекомпіляції.

Найбільш відомі приклади використання — сервер ***віртуальних машин*** Vmware ESX Server, наноядро для Mac OS Classic з процесором POWERPC яке емулювало для ОС апаратуру процесорів Motorola 680x0, наноядро Adeos, що працює як модуль ядра для Linux і дозволяє виконувати одночасно з Linux яку-небудь іншу ОС.

І нарешті ***гібридні ядра*** — практичний результат спроб поєднати переваги мікроядерної архітектури з ефективністю монолітного ядра. Частіше всього являють собою модифіковані мікроядра, що дозволяють для прискорення роботи запускати частину сервісів в просторі ядра як це робиться в ОС з монолітним ядром.

Найбільш показовим прикладом змішання елементів мікроядерної архітектури і елементів монолітного ядра є ОС сімейства Windows NT. Інші приклади — ОС Syllable, BEOS, NetWare, окремі реалізації BSD.