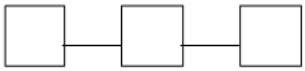
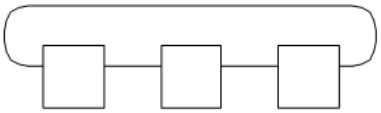
2.3. Топології багатопроцесорних систем

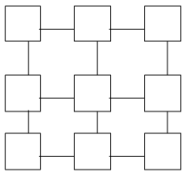
Під топологією багатопроцесорної системи розуміємо спосіб з'єднання процесорних вузлів між собою каналами передачі даних. Зручно подати топологію системи у вигляді графу, вершини якого відповідають процесорним вузлам, а ребра – каналам зв'язку відповідно. Умовно топології можна поділити на фіксовані, реконфігуровані, з одного боку, на регулярні й нерегулярні – з другого. Серед регулярних широко використовують топології таких типів:



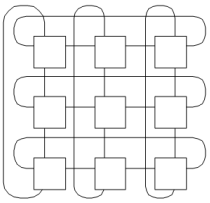
Топологія «лінійка»



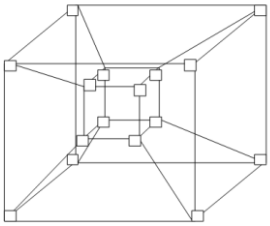
Топологія «кільце»



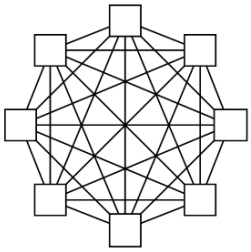
Топологія «решітка3х3»



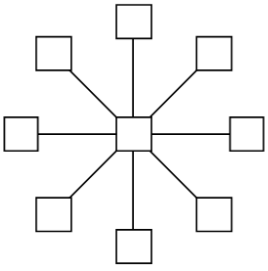
Топологія «тор 3х3»



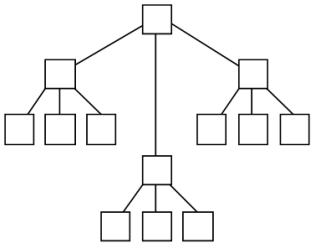
Топологія «гіперкуб ступеня 4»



Топологія «клику»



Топологія «зірка»



Топологія «трійкове дерево»

Властивості використовуваної топології визначають лише ефективність виконання паралельної програми, але й можливість масштабування самої обчислювальної системи. Будь-яку кількість процесорів можна об'єднати в топології типу «лінійка», «кільце», «кліку». Однак для побудови топологій типу «решітка» або «тор» потрібно $n1 \cdot n2$ процесорів, а тому збільшення кількості процесорів можливе тільки квантами розміру $n1$ або $n2$. Для побудови топології «гіперкуб» потрібно $n2$ процесорів, це означає, що кожна наступна система має містити вдвічі більше процесорів, ніж попередня; така топологія вимагає для своєї реалізації наявності n каналів зв'язків на кожному процесорному вузлі, що також обмежує можливості збільшення кількості вузлів у системі.

Щоб підвищити ефективність виконання програм на обчислювальних системах, необхідно узгоджувати фізичну топологію системи й топологію задачі. Велика частина задач математичної фізики успішно розв'язується на системах, процесори яких об'єднані за топологією «решітка». Прямокутні просторові сітки, використовувані для чисельного інтегрування систем диференціальних рівнянь, які описують такі задачі, зручно поділяти на прямокутні частини, які безпосередньо відбиваються на решітці процесорів. Важливо, що від фізичної топології може суттєво залежати ефективність виконання конкретної програми. Перші багатопроцесорні системи, які набули поширення, мали обмежені можливості з погляду реконфігурування. Системи на основі трансп'ютерів дозволяли будувати топології процесорів «двовимірна решітка», «кільце», «циліндр» і «тор». Однак системами з топологією «тор» те, що можна було побудувати на основі трансп'ютерів, можна назвати умовно: для реалізації топології «тор» потрібно саме чотири *link*, при цьому не залишається *link* для приєднання системи до обчислювальної машини, яка керує обчислювальним процесом (рис. 2.14).

У разі такого поєднання процесорів порушується однорідність процесів передачі даних між ними під час обчислень. Побудувати таким чином топологію «гіперкуб» розміром більше 16 ($2^4 = 16$) процесорів, або топологію «тривимірний куб», важко, не згрупувавши процесори у блоки, наприклад,

поєднавши на основі спільної пам’яті у блоки по два процесори, отримаємо обчислювальні вузли з вісьмома *link* (рис. 2.15).

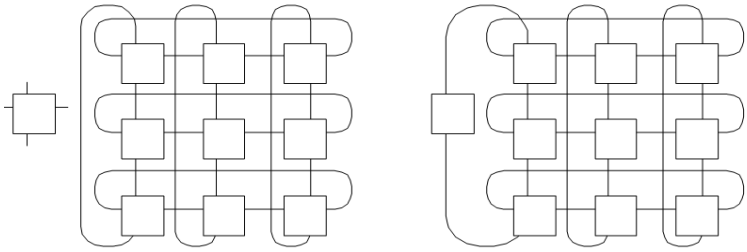


Рис. 2.14. Під’єднання процесорів, з’єднаних за топологією «тор», до керувальної машини

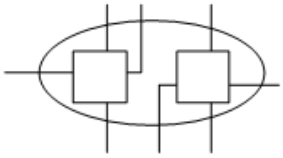


Рис. 2.15. Об’єднання процесорів у обчислювальні вузли

Гіперкуб і тривимірний куб містять топологію «решітка». У зв’язку з чим виникає запитання, якою має бути фізична топологія, щоб за наявності досить жорстких обмежень щодо кількості каналів отримати мінімальну відстань між найбільш віддаленими процесорами? Інакше кажучи, як можна мінімізувати діаметр графу процесорів, зберігаючи низку зв’язків, необхідних для ефективного виконання програми?

Можливе часткове розв’язання цієї задачі – побудова топологій типу «піраміда» (рис. 2.16, 2.17), які дають найкращий з можливих результат, але не є масштабованими.

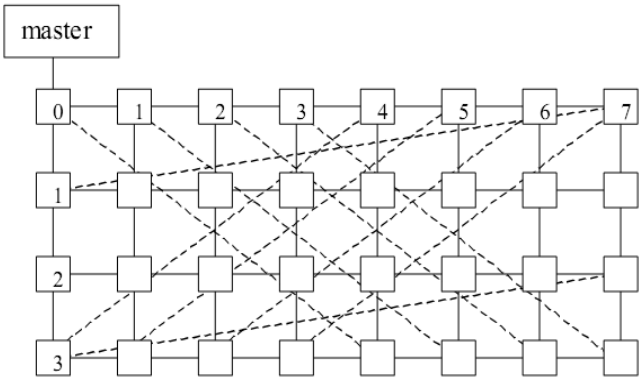


Рис. 2.16. Приклад графу із 32 процесорами з діаметром і радіусом, що дорівнюють чотирьом

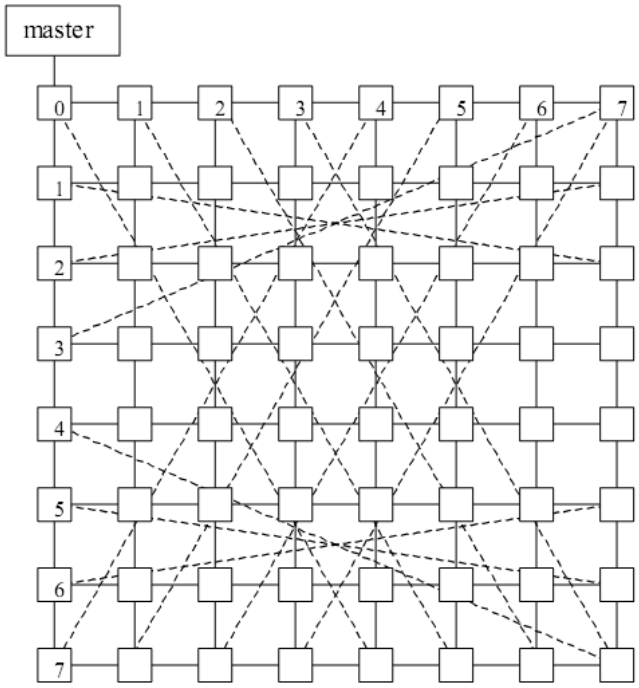


Рис. 2.17. Приклад графу «піраміда» із 64 процесорами з діаметром і радіусом, що дорівнюють шести

2.4. Концепції програмних рішень

Найважливішою властивістю розподіленої системи є можливість здійснення ефективного обміну даними під час взаємодії програмних компонентів, які містяться на одному або на різних комп'ютерах.

Розподілені системи подібні до традиційних операційних систем. Вони, по-перше, працюють як менеджери ресурсів (resource managers) наявного апаратного забезпечення, чим допомагають багатьом користувачам і прикладним програмам спільно використовувати такі ресурси, як процесори, пам'ять, периферійні пристрої, мережу й дані всіх видів. По-друге, розподілена система приховує складність і гетерогенну природу апаратного забезпечення, на основі якого її створено, надаючи віртуальну машину для виконання прикладних завдань.

Першорядними програмними компонентами розподілених систем є операційні системи й системи проміжного рівня. Основні дані щодо розподілених і мережних операційних систем та засобів проміжного рівня подано у табл. 2.1.

Таблиця 2.1. Короткий опис розподілених і мережних операційних систем та засобів проміжного рівня

Система	Опис	Основне призначення
Розподілені операційні системи	Сильнозв'язні операційні системи для мультипроцесорів і гомогенних мультикомп'ютерних систем	Приховання й керування апаратним забезпеченням
Мережні операційні системи	Слабкозв'язні операційні системи для гетерогенних мульти-комп'ютерних систем (локальних або глобальних мереж)	Надання локальних служб віддаленим клієнтам
Засоби проміжного рівня	Додатковий рівень понад мережною операційною системою, що реалізує служби загального призначення	Забезпечення прозорості розподілу

2.4.1. Операційні системи й розподіленість

Операційні системи (ОС) для розподілених комп'ютерів можна поділити на дві категорії: сильнозв'язні та слабкозв'язні системи. У сильнозв'язних системах операційна система переважно працює з одним, глобальним уявленням ресурсів, якими вона керує. Слабкозв'язні системи можуть являти собою набір операційних систем, кожна з яких працює на власному комп'ютері, але вони функціонують спільно, роблячи власні служби доступними для інших.

Сильнозв'язні операційні системи зазвичай називають розподіленими операційними системами (Distributed Operating System, DOS) і використовують для керування мультипроцесорними й гомогенними мультикомп'ютерними системами. Як і в традиційних однопроцесорних операційних системах, основна мета розподіленої операційної системи полягає у прихованні тонкощів керування апаратним забезпеченням, яке одночасно використовує низку обчислювальних процесів.

Слабкозв'язні мережні операційні системи (Network Operating Systems, NOS) використовують для керування гетерогенними мульти-

комп’ютерними системами. Хоча керування апаратним забезпеченням і є основним завданням мережних операційних систем, ці системи відрізняються від традиційних тим, що локальні служби мають бути доступними для віддалених клієнтів. Мережні ОС надають локальні служби в розпорядження віддаленим клієнтам, тому ці ОС мають сервіс віддаленого доступу – RAS (Remote Access Service).

Розрізняють два типи розподілених операційних систем: мультипроцесорну операційну систему (multiprocessor operating system), яка керує ресурсами мультипроцесора, та мультикомп’ютерну операційну систему (multicomputer operating system), яку розроблено для гомогенних мультикомп’ютерів.

Функціональність розподілених операційних систем зазвичай не відрізняється від функціональності традиційних операційних систем, призначених для комп’ютерів з одним процесором, за винятком того, що вона підтримує роботу декількох процесорів.

Операційні системи традиційно будували для керування комп’ютерами з одним процесором, тому їх називано однопроцесорними. Основним завданням цих систем була організація легкого доступу користувачів і прикладних програм до поділюваних ними пристроїв, таких як процесор, пам’ять, диски й периферійні пристрої. Поділ ресурсів означає можливість використання одного й того ж апаратного забезпечення різними прикладними програмами ізольовано одна від одної. Для прикладної програми це виглядає так, немов ці ресурси перебувають у його повному розпорядженні, при цьому в одній системі може виконуватися одночасно декілька прикладних програм, кожна зі своїм набором ресурсів. У такому разі операційна система реалізує віртуальну машину (virtual machine), надаючи прикладним програмам засоби мультизадачності. За умови спільного використання ресурсів у такій віртуальній машині прикладні програми відділено одну від другої, тому неможливі ситуації, коли під час одночасного виконання двох прикладних програм, наприклад *A* та *B*, програма *A* може змінити дані програми *B* через те, що вона працює з тією самою частиною загальної пам’яті, де ці дані зберігаються. Крім того, прикладні програми мають використовувати надані засоби лише так, як запропоновано операційною системою, натомість операційна система надає

первинні операції зв’язку, які можна використовувати для пересилання повідомлень між прикладними програмами на різних машинах.

Операційна система має повністю контролювати використання й розподілення апаратних ресурсів, тому більшість процесорів підтримують як мінімум два режими роботи: у режимі ядра (kernel mode) та у режимі користувача (user mode). У режимі ядра виконуються всі дозволені інструкції, при цьому в ході виконання доступні вся наявна пам’ять і будь-які регістри. У режимі користувача доступ до регістрів й пам’яті обмежений, тобто прикладні програми не можуть працювати з пам’яттю за межами набору адрес, установлених для них операційною системою, або звертатися прямо до регістрів пристроїв. На час виконання коду операційної системи процесор перемикається в режим ядра. Єдиний спосіб перейти з режиму користувача в режим ядра – це зробити системний виклик, реалізований через операційну систему. Оскільки системні виклики здійснюють лише базові служби, які надає операційна система, а обмеження доступу до пам’яті й регістрів нерідко реалізується апаратно, то операційна система може повністю їх контролювати.

Наявність двох режимів роботи зумовило таку організацію операційних систем, за якої майже весь їх код виконується в режимі ядра, в результаті чого часто створюють гігантські монолітні програми, які працюють у єдиному адресному просторі. Такий підхід ускладнює процес переналаштування системи, оскільки замінити або адаптувати компоненти операційної системи без повного перезавантаження, а можливо, й повної перекомпіляції й нової установки дуже важко. З погляду вимог до проектування програм, таких як відкритість, надійність або легкість обслуговування, монолітні операційні системи неефективні.

Більш зручна організація операційної системи у вигляді двох частин. Одна частина містить набір модулів для керування апаратним забезпеченням, який легко може виконуватися в режимі користувача. Наприклад, керування пам’яттю полягає передусім у відстеженні, які блоки пам’яті виділені під процеси, а які вільні, причому робота в режимі ядра необхідна під час установлення регістрів блока керування пам’яттю.

Друга частина операційної системи має невелике мікроядро (microkernel), яке містить винятково код, що виконується в режимі ядра. На практиці

мікроядро має містити лише код для установлення регістрів пристроїв, перемикання процесора з одного процесу на другий, роботи з блоком керування пам’яттю й перехоплення апаратних переривань. Крім того, у ньому зазвичай міститься код, який перетворює виклики відповідних модулів рівня користувача операційної системи в системні виклики й повертає результати. Такий підхід зумовлює організаціїю ОС, показану на рис. 2.18.

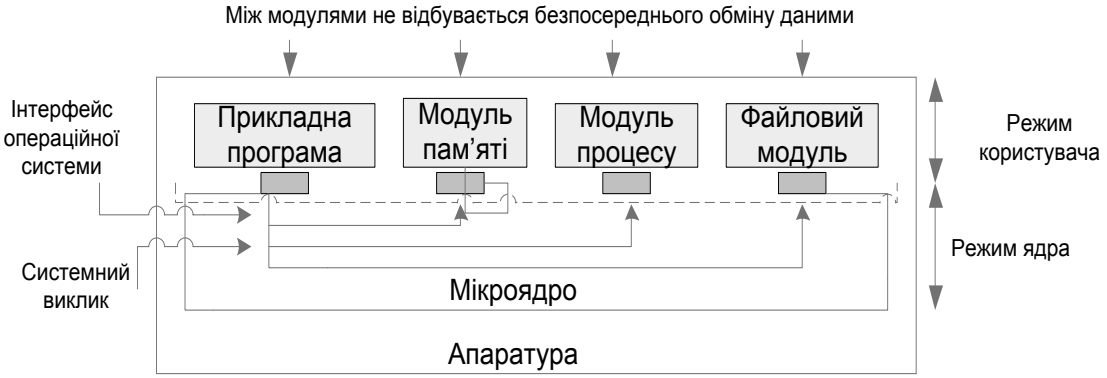


Рис. 2.18. Організація операційної системи з використанням мікроядра

Використання мікроядра має низку переваг, найбільш важливою з яких є гнучкість системи, оскільки більша частина операційної системи виконується в режимі користувача, то порівняно просто замінити один з модулів без повторних компіляції або установлення всієї системи. Інша суттєва перевага полягає в тому, що модулі рівня користувача можуть розміщуватися на різних машинах. Підхід з використанням мікроядра зручний для роботи однопроцесорних операційних систем на розподілених комп'ютерах.

У мікроядер є два істотні недоліки: по-перше, вони працюють інакше, ніж наявні операційні системи; по-друге, мікроядро вимагає додаткового обміну даними, що знижує продуктивність.

Важливим подальшим кроком розвитку однопроцесорних операційних систем є можливість підтримки декількох процесорів, що мають доступ до спільно використовуваної пам’яті. Такі операційні системи називають мультипроцесорними. Всі структури даних, необхідні операційній системі для підтримки апаратури, зокрема декількох процесорів, розміщуються в пам’яті, що доступна декільком процесорам, тому мають бути захищені від паралельного доступу для забезпечення їх цілісності.

Багатопроцесорні операційні системи потрібні для підтримання високої продуктивності конфігурацій з декількома процесорами. Основне їх завдання – забезпечити прозорість кількості процесорів для прикладних програм. Повідомлення, якими обмінюються різні прикладні програми або їх частини, вимагає тих самих примітивів, що застосовують багатозадачні однопроцесорні операційні системи, тобто всі повідомлення працюють з даними у спеціальній, області пам'яті, яка спільно використовується, тому необхідно захистити дані від одночасного доступу до них за допомогою примітивів синхронізації, зокрема двох найбільш важливих (і еквівалентних) примітивів –семафорів і моніторів.

Семафор (semaphore) можна подати у вигляді цілого числа, що підтримує дві операції: *up* (збільшити) і *down* (зменшити). У разі зменшення числа спочатку перевіряється, чи перевищує значення семафора нульове значення. Якщо це так, його значення зменшується й виконання процесу триває; якщо ж значення семафора нульове, то процес, який виконує виклик, блокується. Оператор збільшення виконує протилежну дію: спочатку він перевіряє всі заблоковані в цей час процеси, які не завершилися під час виконання попередньої операції зменшення. Якщо такі наявні, то він розблоковує один з них і продовжує роботу, інакше він просто збільшує лічильник семафора. Процес, який розблоковано, виконується до виклику операції зменшення. Важливою властивістю операцій з використанням семафорів є те, що вони атомарні (atomic), тобто коли запущено операцію зменшення або збільшення до моменту її завершення (або до моменту блокування процесу), жодний інший, процес не може отримати доступ до семафора.

Використання семафорів для синхронізації процесу спричиняє багато похибок, за винятком простого захисту поділюваних даних. Альтернативою семафорам у багатьох сучасних системах, що підтримують паралельне програмування й надають бібліотеки для його реалізації, є використання моніторів.

Формально **монітор** (monitor) є конструкцією мови програмування, такою самою, як об'єкт в об'єктно-орієнтованому програмуванні. Монітор можна розглядати як модуль, що містить змінні та процедури, а доступ до змінної можна отримати,лише, викликавши одну з процедур монітора. Таким чином,

монітор дуже схожий на об’єкт, бо також має свої захищені дані, доступ до яких можна одержати лише через методи, реалізовані в цьому об’єкті. Відмінність між моніторами й об’єктами полягає в тому, що монітор дозволяє виконання процедури тільки одному процесу в кожний момент часу. Якщо процедура, яка міститься в моніторі, виконується процесом *A* (*A* ввійшов у монітор), а процес *B* також викликає одну з процедур монітора, то процес *B* буде блоковано до завершення виконання процесу *A* (доки *A* не залишить монітор).

Мультикомп’ютерні операційні системи мають набагато різноманітнішу структуру і складніші, ніж мультипроцесорні. Для мультикомп’ютерних операційних систем структури даних, необхідні для керування системними ресурсами, не мають задовольняти умову їх спільного використання, оскільки їх не потрібно розміщувати у загальній пам’яті. Єдиним можливим видом зв’язку є передача повідомлень (message passing). Мультикомп’ютерні операційні системи здебільшого організовані так, як показано на рис. 2.19.

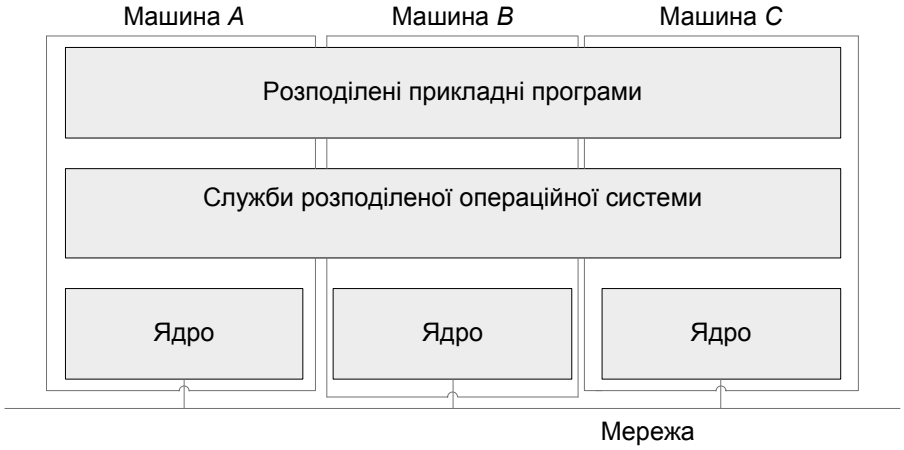


Рис. 2.19. Загальна структура мультикомп’ютерних операційних систем

Кожний вузол системи має своє ядро, яке містить модулі для керування локальними ресурсами (пам’яттю, локальним процесором, локальними дисками та ін.), а також окремий модуль для міжпроцесорної взаємодії, тобто відсилання повідомлень на інші вузли і прийому повідомлень від них. Над кожним локальним ядром міститься рівень програмного забезпечення загального призначення, що реалізує операційну систему у вигляді віртуальної машини, яка підтримує паралельну роботу із різними завданнями. Цей рівень може навіть надавати абстракцію мультипроцесорної машини, тобто повну про-

грамну реалізацію пам’яті, яка спільно використовується. Додаткові засоби необхідні, наприклад, для обрання завдань процесорам, маскування збоїв апаратури, забезпечення прозорості збереження даних і загального обміну між процесами.

Системи з розподіленою поділюваною пам’яттю – це системи, які використовують віртуальну пам’ять кожного окремого вузла для підтримання загального віртуального адресного простору, що зумовлює використання розподіленої поділюваної пам’яті (Distributed Shared Memory, DSM) зі сторінковою організацією. Принцип роботи цієї пам’яті такий: у системі з DSM адресний простір розділено на сторінки (зазвичай по 4 або 8 Кбайт), які розподілено по всіх процесорах системи. Коли процесор адресується до пам’яті, що не є локальною, відбувається внутрішнє переривання. Операційна система зчитує в локальну пам’ять сторінку, що містить зазначену адресу, і перезапускає виконання інструкції, яка спричинила переривання та яка після цього успішно виконується. Як тимчасове сховище інформації використовується не диск, а віддалена оперативна пам’ять.

Одним з покращень базової системи, що підвищує її продуктивність, є реплікація сторінок, які оголошуються закритими для запису, наприклад, таких, що містять текст програми, константи «тільки для читання» або інші закриті на запис структури.

Ще одним покращенням є можливість реплікації також не закритих на запис сторінок, оскільки виконується лише читання, то ніякої різниці між реплікацією закритих і не закритих на запис сторінок немає. Однак, якщо реплікована сторінка змінюється, то необхідно вживати спеціальних заходів для запобігання появі низки несумісних копій. Зазвичай усі копії, крім однієї, перед записуванням вважаються хибними.

Додаткового підвищення продуктивності можна досягти відходом від строгої відповідності між сторінками, які реплікуються, щоб окрема копія тимчасово відрізнялася від інших. Практика показує, що цей підхід насправді може допомогти, але слід обов’язково відслідковувати можливу несумісність. Оскільки основною передумовою розробки DSM була простота програму-

вання, то послаблення відповідності між сторінками не набуває реального застосування.

Ще однією проблемою під час розробки ефективних систем DSM є розмір сторінок. Витрати на передачу сторінки мережею передусім визначаються витратами на підготовку до передачі, а не обсягом переданих даних. Відповідно, великий розмір сторінок може зменшити загальну кількість сеансів передачі у разі необхідності доступу до великої кількості послідовних елементів даних. Натомість, якщо сторінка містить дані двох незалежних процесів, що виконуються на різних процесорах, операційна система буде змушена постійно пересилати цю сторінку від одного процесора до другого, як показано на рис. 2.20. Розміщення даних двох незалежних процесів на одній сторінці називають помилковим поділом (false sharing).

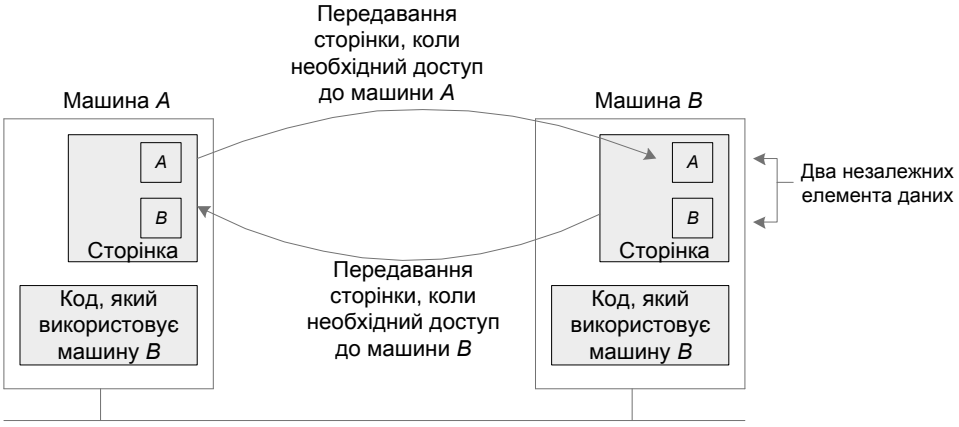


Рис. 2.20. Помилковий поділ сторінки двома незалежними процесами

Для досягнення високої продуктивності великомасштабних мультикомп'ютерних систем здійснюється пересилання повідомлень, незважаючи на високу складність його реалізації порівняно з програмуванням систем з віртуальною пам'ятью спільного використання. Це дозволяє зробити висновок про те, що DSM не є ефективною для високопродуктивного паралельного програмування.

На відміну від розподілених операційних систем, мережні не мають потреби в тому, щоб апаратне забезпечення, на якому вони функціонують, було гомогенним і керованим як єдина система. Мережні системи будуються з набору однопроцесорних систем, кожна з яких має власну операційну сис-

тему, як показано на рис. 2.21. Машини та їх операційні системи можуть бути різними, але їх з'єднано в мережу. Крім того, мережна операційна система дозволяє користувачам використовувати служби, розміщені на конкретній машині, зокрема:

- віддалене з'єднання користувача з другою машиною;
- віддалене копіювання файлів з однієї машини на другу.

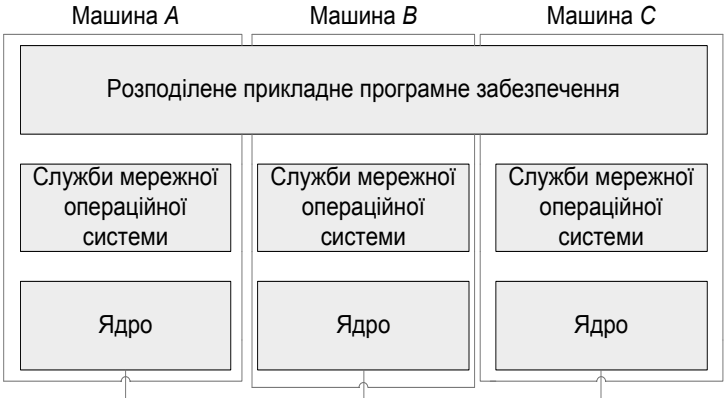


Рис. 2.21. Загальна структура мережної операційної системи

Для віддаленого з'єднання користувача з другою машиною використовується команда *rlogin machin*, у результаті виконання якої відбувається перемикання робочої станції користувача в режим віддаленого терміналу, під'єднаного до віддаленої машини. Команди передаються на віддалену машину, а результати з неї відображаються у вікні на екрані користувача. Щоб переключитися на другу віддалену машину, необхідно відкрити нове вікно і скористатися командою *rlogin* для з'єднання. Вибір виконується вручну.

Приклад. Команда віддаленого копіювання файлів копіює файл *file 1* з машини *machine 1* на *machine 2* і присвоює йому ім'я *file 2*, при цьому переміщення файлів задається явно, й користувачеві необхідно точно знати, де перебувають файли і як виконуються команди.

Така форма зв'язку вкрай примітивна. Більш зручним варіантом зв'язку і спільного використання інформації є один з підходів, що передбачає створення глобальної загальної файлової системи, доступної з усіх робочих станцій. Файлова система підтримується однією або декількома машинами, які називають **файловими серверами** (file servers).

Файлові сервери отримують запити від програм користувачів, що запускаються на інших машинах (не на серверах), які називають **клієнтами** (clients),

для читання й записування файлів. Кожний запит, що надійшов, перевіряється й виконується, а результат пересилається назад, як показано на рис. 2.22.

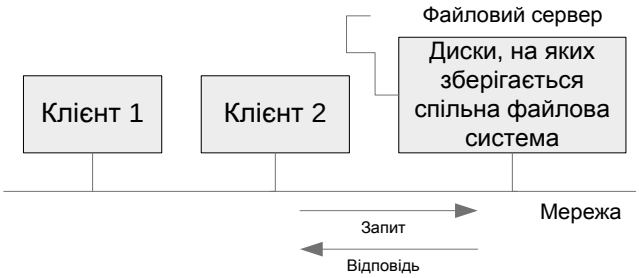


Рис. 2.22. Два клієнти й сервер у мережній операційній системі

Файлові сервери зазвичай підтримують ієрархічні файлові системи, кожна з яких має кореневий каталог, який містить вкладені каталоги й файли. Робочі станції можуть імпортувати або монтувати ці файлові системи, збільшуючи свою локальну файлову систему за рахунок файлової системи сервера.

Мережні операційні системи виглядають значно простіше, ніж розподілені. Основна відмінність між мережними й розподіленими типами операційних систем полягає у тому, що в розподілених операційних системах повністю виконується вимога прозорості системи, тобто подання всього інформаційного простору у вигляді єдиної системи.

Неповне виконання вимоги прозорості в мережних операційних системах має деякі очевидні недоліки. Наприклад, з ними часто складно працювати, оскільки користувач змушений вручну приєднуватися до віддалених машин або копіювати файли з однієї машини на другу, що спричиняє проблеми з керуванням ресурсами всієї системи. Оскільки всі машини, якими керує мережна операційна система, незалежні, то часто й керувати ними можна тільки незалежно, у результаті чого користувач може виконати віддалене з'єднання з машиною *X*, лише зареєструвавшись на ній. Якщо користувач хоче використовувати один пароль для роботи з декількома машинами, то за потреби змінити пароль він змушений буде змінити його на кожній машині. Очевидно, що на кожній машині наявні свої права доступу. Такий децентралізований підхід до безпеки ускладнює захист мережної операційної системи.

Мережні операційні системи порівняно з розподіленими мають також і переваги. Оскільки вузли мережних операційних систем значною мірою не

залежать один від одного, додати або видалити машину дуже легко. Зокрема, щоб додати вузол, треба приєднати відповідну машину до загальної мережі й повідомити про її наявність іншим машинам мережі. В Internet, наприклад, додавання нового сервера відбувається в такий спосіб: щоб відомості про машину потрапили в Internet, необхідно надати їй мережну адресу, а краще символічне ім'я, яке потім буде внесено в доменну систему іменування DNS разом з її мережною адресою.

Domain Name System (DNS) – доменна система іменування (імен), або служба доменних імен, Internet-служба є розподіленою по всій земній кулі базою даних для ієрархічної системи імен мереж і комп'ютерів, під'єднаних до глобальної мережі, а також є способом або протоколом прикладного рівня перетворення рядкових адрес серверів Internet у числові IP-адреси. Протокол DNS працює над протоколом UDP (протоколом дейтаграм користувача, протоколом транспортного рівня з набору протоколів TCP/IP) і йому призначено порт із номером *53*; DNS також часто використовують для розподілу навантаження між дублюючими серверами (дзеркалами) популярних сайтів і поштових серверів.

Система імен у DNS – ієрархічно організована розподілена система з дублюванням усіх функцій між двома й більше серверами (рис. 2.23).

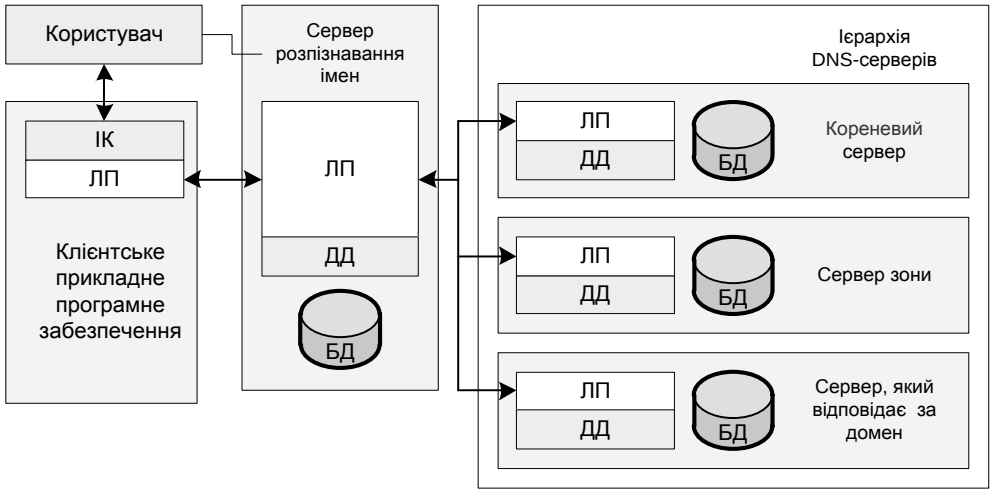


Рис. 2.23. Система DNS: ІК – інтерфейс користувача; ЛП – логіка прикладних програм; ДД – доступ до даних

Запит користувача на перетворення імені (наприклад, www.microsoft.com) у мережну адресу (65.225.182.56) передається серверу розпізнавання імен постачальника послуг Internet, сервер почергово опитує сервери з ієрархії служби імен, починаючи з кореневих серверів, які повертають адреси серверів, що відповідають за зону домена, у яких містяться проміжні шляхи.

Потім опитується сервер, який відповідає за зону (у цьому разі – .com), що повертає адреси серверів, які відповідають за домен других рівнів, і так далі. Сервери імен кешують інформацію про відповідність імен і адрес для зменшення навантаження на систему. Програмне забезпечення на комп'ютері користувача зазвичай може з'єднатися як мінімум з двома різними серверами розпізнавання імен, проте і в системі розпізнавання імен не всі вимоги до розподілених систем виконано, зокрема така система не містить явних механізмів забезпечення безпеки. Це призводить до регулярних атак на сервери імен, щоб вивести їх з ладу, наприклад значною кількістю запитів.

2.4.2. Проміжне середовище

Розподілені операційні системи не призначені для керування набором незалежних комп'ютерів, а мережні операційні системи не створюють уявлення однієї узгодженої системи. Бажаною властивістю розподіленої системи є можливість організації ефективного обміну даними під час взаємодії програмних компонентів, що перебувають на одному або на різних комп'ютерах. Щоб насправді побудувати розподілену систему, служб мережної операційної системи недостатньо, необхідно долучити до них прикладні програмні компоненти для покращення прозорості розподілу. Задовольнити вимоги масштабованості й відкритості можна за допомогою додаткового рівня програмного забезпечення, який у мережних операційних системах дозволяє якось сховати від користувача різнорідність набору апаратних платформ, чим і підвищити прозорість розподілу. Багато сучасних розподілених систем побудовано з використанням цього додаткового рівня, який названо програмним забезпеченням проміжного рівня, платформою розподілу, системою проміжного рівня або проміжним середовищем (middleware).

Велика кількість розподілених прикладних програм передбачає безпосереднє використання програмного інтерфейсу, наявного у мережних операційних систем та інтерфейсів локальних файлових систем. Проблема такого підходу полягає в тому, що наявність розподілу є занадто конкретною. Для забезпечення додаткового абстрагування між прикладною програмою й мережною операційною системою розміщено рівень програмної підтримки, як показано на рис. 2.24, який і називають **проміжним рівнем**.

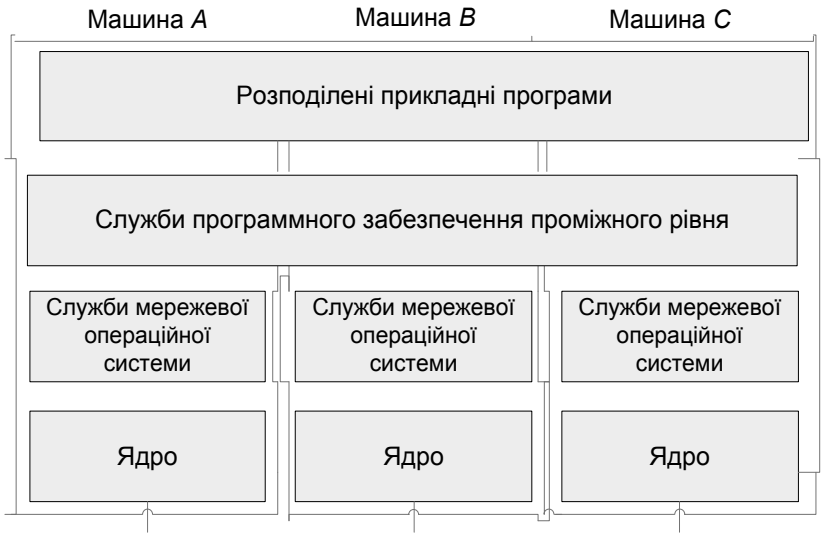


Рис. 2.24. Загальна структура розподілених систем із проміжним рівнем

Система проміжного рівня забезпечує скоординовану роботу мереж і ОС, надаючи можливість використання їх програмного інтерфейсу. Ефективне проміжне середовище має організовувати взаємодію групи комп'ютерів мережі, не порушуючи стек протоколів TCP/IP. Для цього можуть використовуватися системні сокети (unix sockets) у POSIX-системах або іменовані канали (named pipes).

Платформа розподілу підтримує взаємодію між потенційно гетерогенними системами, які на верхньому рівні працюють з прикладними компонентами, може додаватися до локальної операційної системи або отримувати завдання від операційної системи. У такий спосіб підтримується прозорість розподілу. Прикладні програми ізолюються від деталізації структури функцій операційної системи і взаємодії її внутрішніх процесів. Платформи розподілу можуть бути

покладені в основу багатьох конкретних прикладних систем. На платформі розподілу може функціонувати одночасно декілька прикладних програм.

Основне завдання програмного забезпечення проміжного рівня – приховати відмінності базових стандартів ОС від прикладного програмного забезпечення, розробляючи яке, використовують низку моделей або параметрів, що характеризують розподілені системи. Основними моделями є такі, що визначають розподіл і зв'язок.

Крім платформи розподілу, ще однією моделлю організації взаємодії між прикладними програмами або системами в розподіленому гетерогенному середовищі є модель віддаленого виклику процедур RPC (Remote Procedure Call), яка забезпечує виклик процедур з віддалених машин. Під час виклику процедури параметри RPC прозоро передаються на віддалену машину, а результат повертається назад. У разі використання об'єктного підходу прикладні програми створюються за допомогою засобів поєднання розподілених об'єктів, що являють собою певну програмну реалізацію, причому кожний із цих об'єктів реалізує свій інтерфейс, який приховує всі внутрішні деталі реалізації процесу взаємодії від кінцевого користувача. Інтерфейс передає вихідні дані для реалізації методів. Внутрішній та зовнішній процеси, які взаємодіють між собою, бачать лише свій інтерфейс. Часто такі розподілені об'єкти розміщуються на одній машині, а програмне забезпечення проміжного рівня надає дозвіл на доступ до інтерфейсів об'єктів і здійснює цей доступ до них з інших машин. Під час виклику процесором методу інтерфейс перетворює дані, що містяться в інтерфейсі, в повідомлення, яке відсилається об'єкту, який виконує метод і повертає результат.

Прикладну програму, що реалізує кожну із системних функцій платформи розподілу, називають **службою**. Розрізняють такі основні служби платформи розподілу: засоби прозорого доступу до віддалених даних (FS, www); служби віддаленого доступу (для виклику процедур і звертання до розподілених об'єктів); служби іменування (url); засоби збереження даних або засоби живучості (persistence), наприклад розподілені FS, інтегровані бази даних, засоби зв'язування прикладних програм з базами даних; засоби розподілених

транзакцій, які здійснюють низку операцій зчитування й записування у межах однієї атомарної операції.

Програмне забезпечення проміжного рівня дозволяє поєднувати масштабованість і відкритість мережних операційних систем із прозорістю і простотою використання розподілених систем. Таке поєднання дозволяє створювати розподілену інформаційну систему із загальною структурою, показаною на рис. 2.24.

2.5. Поняття розподіленого середовища

Розподілене середовище – віртуальний обчислювальний простір, який може обмежуватися однією розподіленою системою або містити кілька розподілених систем, які взаємодіють між собою. Такий віртуальний обчислювальний простір надається користувачеві у вигляді систематизованого сховища інформаційних та програмних ресурсів, має певну структуру, зрозумілу систему адресації ресурсів та певні моделі обчислювальних процесів або бізнес-процесів цього користувача, які є проблемно-орієнтованими, відповідають певним видам робіт. Користувач звертається до бізнес-процесів, які, у свою чергу, отримують необхідні ресурси для своєї роботи.

Учасниками взаємодії в розподіленому середовищі є окремі сутності, якими можуть бути користувачі, прикладні програми та інші обчислювальні ресурси. Як основу опису взаємодії двох сутностей розглянемо загальну модель взаємодії *клієнт – сервер*, у якій одна зі сторін (клієнт) ініціює обмін даними, надсилаючи запит другій стороні (серверу). Сервер обробляє запит й у разі потреби надсилає відповідь клієнтові (рис. 2.25).

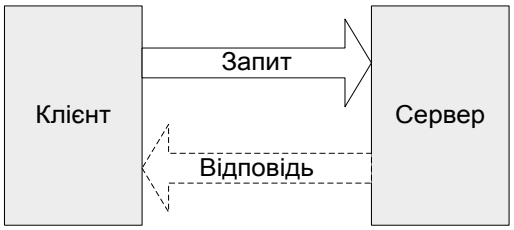


Рис. 2.25. Модель взаємодії *клієнт – сервер*

У базовій моделі *клієнт – сервер* усі процеси в розподілених системах поділяють на дві групи: процеси, які реалізують певну службу, наприклад