МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни

“Програмне забезпечення кіберфізичних систем”

ЗМІСТ

Тема 1. Архітектура та призначення операційних систем

Операційна система (ОС) - це програма, що забезпечує можливість раціонального використання устаткування комп'ютера зручним для користувача чином, тобто являють собою набір програмних модулів, які дозволяють користувачеві керувати машиною, а також забезпечують взаємодію програм з зовнішніми пристроями та один з одним. ОС — головна частина системного програмного забезпечення. Операційна система управляється командами.

Рис. 1.1. UNIVAC (UNIVersal Automatic Computer), 1960

1.1. КЛАСИФІКАЦІЯ ОПЕРАЦІЙНИХ СИСТЕМ:

- Локальні ОС;
- Мережні ОС;
- ОС розподілених систем(Cloud Computing);
- Спеціалізовані ОС;
- ОС реального часу;

Класифікація з точки зору можливостей ОС:

- Універсальні (для широкого використання), спеціальні (для розв'язання спеціальних задач) та спеціалізовані (виконуються на спеціальному обладнанні);
- Підтримка багатозадачності(одно/багато-задачні);
- Підтримка багатокористувацького режиму;
- За способом реалізації багатозадачності (з примусовим / без примусового переключення);
- Підтримка багатоядерності/багатопроцесорності;
- Вбудовані/невбудовані;
- По розширенню відкриті та закриті;
- По доступу до вхідного коду вільні та комерційні;

1.2. ОСНОВНІ ФУНКЦІЇ ОПЕРАЦІЙНИХ СИСТЕМ:

- ОС розглядається як розширення машини(апаратної частини), тобто оболонка Hardware яка в собі приховує роботу заліза;
- ОС як система управління ресурсами(планування та використання ресурсів, моніторинг біжучого стану використання ресурсу);

Основні функції операційних систем:

- Планування завдань та використання процесора(обробка переривань і забезпечення багатозадачної роботи);
- Забезпечення комунікації та синхронізації процесів;
- Забезпечення інтерфейсу між прикладними програмами та службовими сервісами ОС;
- Управління пам’яттю;
- Управління файловою системою;
- Управління вводом/виводом;
- Забезпечення безпеки;

1.3. АРХІТЕКТУРА ОПЕРАЦІЙНИХ СИСТЕМ

При функціональної декомпозиції ОС модулі поділяються на дві групи:

- ядро - модулі, які виконують основні функції ОС;
- модулі, які виконують допоміжні функції ОС.

Рис. 1.2. Основні рівні програмного забезпечення

Робота процесора можлива в 2 режимах:

- користувача (user mode);
- привілейований (kernel mode) в своїй віртуальній пам’яті.

Виникає ряд проблем з переключеннями режим/режим або контекст/контекст.

Перспективні архітектури:

- Архітектура монолітного ядра(ядро-одна програма. Висока швидкодія)
- Мікроядерна архітектура ОС(швидкодія низька)
- Комбіновані архітектури;

Архітектура ОС UNIX:

- апаратні засоби (hardware),
- ядро (kernel),
- системні служби та сервіси (system tasks and services),
- оболонка (shell),
- прикладні програми (application programs)

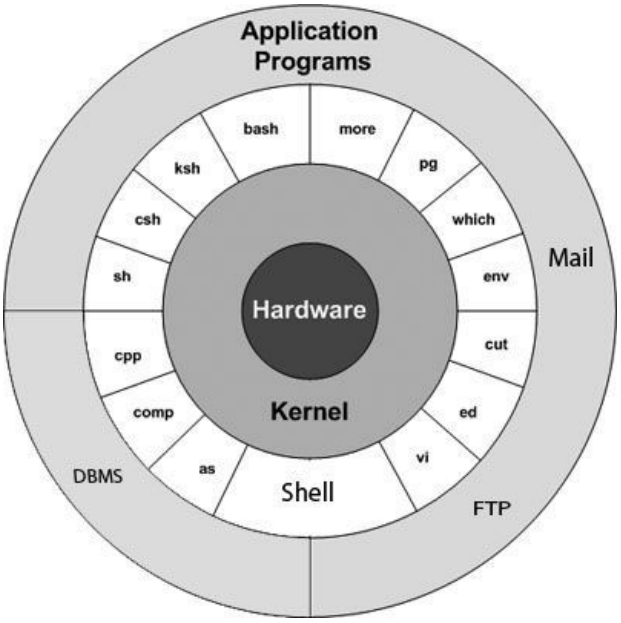


Рис. 1.3. Архітектура ОС UNIX

Ядро ОС (OS kernel) — частина ОС, яка постійно знаходиться в пам'яті, керує усіма іншими обчислювальними процесами та розподіляє між ними ресурси ЕОМ. Обчислювальний процес = програма під час виконання; паралельні процеси: процеси ядра, процеси системних служб, процеси прикладних програм. Приклади ОС:

1) UNIX, FreeBSD, Linux, Android + OS X, iOS

2) MS-DOS, Windows NT / XP / 7 / 8 / 10

1.4. ОСОБЛИВОСТІ МЕРЕЖНИХ ОПЕРАЦІЙНИХ СИСТЕМ

Мережна операційна система (ОС) – це пакет програм, що забезпечує реалізацію та управління мережею, дає змогу клієнтам користуватись мережним сервісом.

Важливою функцією мережної ОС є забезпечення системи захисту – конфіденційності зберігання даних, розмежування прав доступу до ресурсів, парольний захист, виявлення спроб несанкціонованого доступу, трасування дій користувачів, ведення журналів системних подій тощо.

Мережна ОС забезпечує підтримку різноманітних периферійних пристроїв, мережних адаптерів, протоколів та можливість їх конфігурування. Програмне забезпечення клієнтської частини перетворює запити прикладної програми на

використання мережних ресурсів у відповідні мережні формати, забезпечує їх пересилання через середовище передавання та здійснює зворотні перетворення.

1.5. СТРУКТУРА МЕРЕЖНОЇ ОПЕРАЦІЙНОЇ СИСТЕМИ

Мережна ОС має у своєму складі засоби передачі повідомлень між комп'ютерами по лініях зв'язку, що зовсім не потрібні в автономній ОС.
Серверна частина надає ресурси у загальне користування.
Клієнтська частина доступається до ресурсів.

Розділяють однорангові ОС(peer to peer) та серверні ОС(один за комп’ютерів має перевагу в серверній частині)

Однорангові мережі дають змогу кожному вузлу мережі одночасно виступати в ролі сервера та клієнта. Звичайно, клієнт може мати одночасний доступ до ресурсів різних мереж, що використовують спільне середовище передавання.

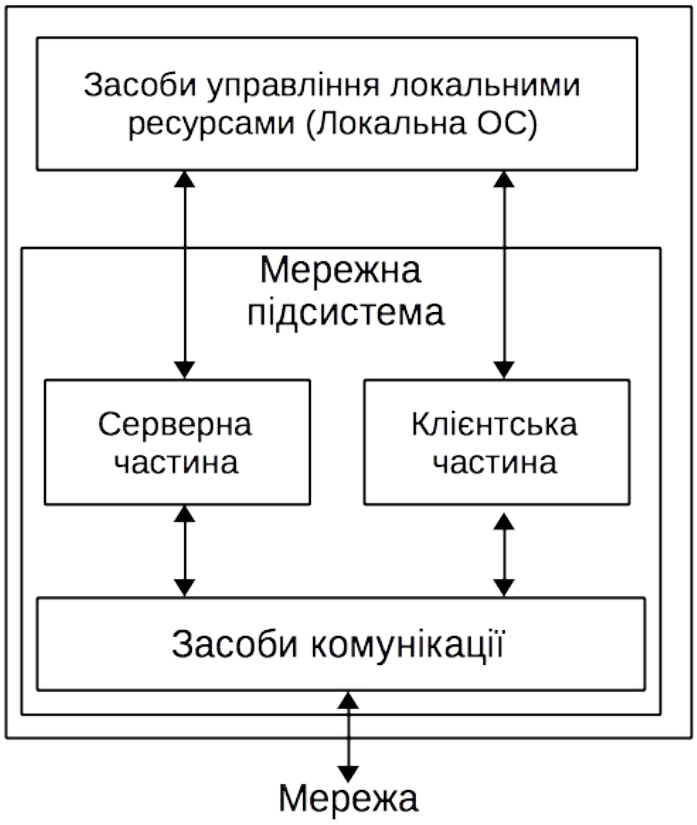


Рис. 1.4. Узагальнена схема мережної ОС

1.6. ОСНОВНІ ЗАВДАННЯ МЕРЕЖНОЇ ОПЕРАЦІЙНОЇ СИСТЕМИ:

- забезпечення сумісного використання та розподілу ресурсів мережі;
- прискорення обчислень;
- підвищення надійності за рахунок гарячого резервування;
- забезпечення взаємодії користувачів(надання клієнтам мережного сервісу та адміністрування мережі, взаємодії процесів у мережі та обміну повідомленнями між вузлами мережі);

Тема 2. Операційна система Linux

2.1. Особливості розробки ОС Linux

1. Основні етапи розробки ОС Linux: Linus Torvalds (1991р.), MINIX (Tanenbaum) , UNIX-POSIX + GNU Project (Richard Stallman, Free Software Foundation (FSF))
2. UNIX: 1) UNIX System V Release 4 (SVR4); 2) Berkeley Software Distribution (BSD)
3. Повна відкритість процесу розробки + повністю відкритий код системи
4. За оцінками експертів вартість розробки на даний час складає більше $1 млрд
5. Масштаб ядра системи: біля 30 тис. Файлів, біля 8 млн. рядків коду, репозитарій = 1Gb (версія 0.11 у вересні 1991 мала трохи більше 10 тис. рядків коду)
6. Широка популярність в академічному середовищі + у сфері комп'ютерних мереж (сервери і т.п.)
7. Остання версія ядра ОС Linux: 4.2 (30.09.2015)
8. Linux успадкувала у UNIX «ідеологію» побудови: ОС для групи досвідчених програмістів, які спільно працюють над складним проектом (на противагу «ідеології» Windows: «персональна» ОС для користувача-початківця)
9. Принципи, які характеризують «ідеологію» UNIX/Linux [Таненбаум]:
   - максимально можлива уніфікованість інтерфейсів користувача та програмних інтерфейсів -> принцип найменшої несподіваності (principle of least surprise);
   - модульність та гнучкість (як можливість конфігурувати різні набори модулів за потребою [досвідченого] користувача) -> принцип: одна програма = одна функція;
   - мінімалізм у назвах команд та способах взаємодії користувача з ОС.

2.2. Організація роботи ОС Linux

- supervisor/user-mode (підтримка на рівні процесорної архітектури: hierarchical protection domains = protection rings) -> У випадку системного виклику відбувається переключення з режиму користувача (user mode) у режим ядра (kernel mode), як правило, шляхом емульованого переривання. -> рівні інтерфейсів системи Linux

1. інтерфейс користувача: графічний (віконна система, наприклад X-Windows), командний рядок (символьний термінал tty) -> стандартні службові команди;
2. програмний інтерфейс -> бібліотеки функцій, glibc (GNU C Library - implementation of the C standard library) -> всього декілька тисяч;
3. System Call Interface (SCI) -> інтерфейс системних викликів (всього близько 380 системних викликів) -> стандартизація назв та «поведінки»: Portable Operating System Interface (POSIX);
4. апаратний інтерфейс -> породжує апаратно-залежний код.

Компоненти системи Linux: 1) Ядро; 2) Системні бібліотеки (SCI + відповідні бібліотечні фукнції); 3) Системні утиліти (в тому числі daemons).

Ядро ОС Linux - це монолітне ядро з підтримкою завантажених модулів (завантажуваний модуль ядра, LKM). Ядро – це єдиний, монолітний бінарний код. Основна причина цього - підвищення продуктивності. Оскільки всі коди ядра та структури даних зберігаються в одному адресному просторі, не потрібне перемикання контексту, коли процес викликає функцію операційної системи або коли обробляється апаратне переривання.

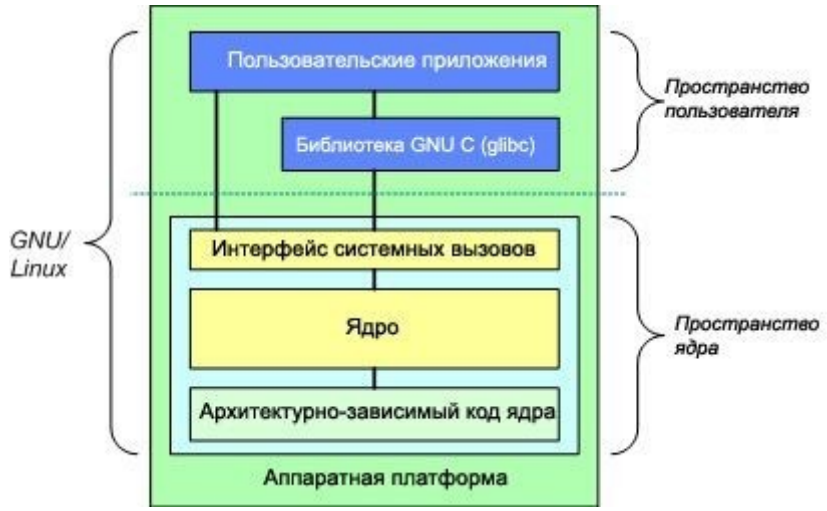


Рис. 2.1. Схема організації роботи ядра ОС Linux

- незалежна частина -> однакова для всіх
- апаратно-залежна частина

Основні апаранті компоненти, з якими працює ядро: 1) процесор, 2) пристрій управління пам'яттю (memory management unit (MMU)), 3) пристрої вводу/виводу.

На найнижчому рівні ядра знаходяться драйвери пристроїв, які забезпечують безпосереню взаємодію з апаратурою. Їх основне завдання: обробка переривань від апаратних пристроїв. У разі винекнення переривання запускається диспетчер переривань, який вибудовує наявні переривання у чергу обробки, після чого запускається обробник переривання з найбільшим пріоритетом. При цьому користувацький процес, який виконувався в цей час, призупиняється до моменту закінчення роботи обробника.

Драйвери та розширення ядра звичайно запускаються в 0-кільці (0-Ring) захисту з повним доступопм до апаратного забезпечення. На відміну від звичайних монолітних ядер, драйвери пристроїв легко збираються у вигляді модулей та завантажуються і вивантажуються під час роботи системи.

Підтримка модулів під Linux має три компоненти:

1. Управління модулем дозволяє завантажувати модулі в пам'ять і спілкуватися з рештою ядра.
2. Реєстрація драйверів дозволяє модулям повідомляти решту ядра про те, що новий драйвер став доступним.
3. Механізм вирішення конфліктів дозволяє різним драйверам пристроїв резервувати апаратні ресурси та захищати ці ресурси від випадкового використання іншим драйвером.

2.3. Основні підсистеми ядра ОС Linux

Рис. 2.2. Основні підсистеми ядра ОС Linux

1. Управління процесам (планування, виконання, взаємодія)
2. Управління пам’яттю (MM, сторінкова організація, віртуальна пам’ять)
3. Віртуальная файлова система (VFS) + драйвери FS
4. мережна підсистема
5. сервіси: час і т.п.
6. Драйвери пристроїв + обробка переривань
7. Архітектруно-залежні частини

0) Драйвери пристроїв та диспетчер переривань.

1) Компонент вводу/виводу містить всі ті частини ядра, які відповідають за взаємодію з пристроями, а також виконання мережевих операцій і операцій введення-виведення на зовнішні пристрої. На найвищому рівні всі операції введення-виведення інтегровані в рівень віртуальної файлової системи (virtual file system (VFS)) - пристрої вводу/виводу з символьних та блокових доступом.

2) До завдань управління пам'яттю входять:
- обслуговування відображення віртуальної пам'яті на фізичну;
- підтримка кешу сторінок, до яких нещодавно отримували доступ (і реалізація хорошою стратегії заміни сторінок);
- підвантаження сторінок з ПЗП у ВП (по вимозі корістувацькіх процесів).

3) Основна сфера відповідальності компонента управління процесами - це створення і завершення процесів. У ньому є також планувальник процесів, який вибирає, який процес (вірніше, потік) буде працювати наступним. Ядро Linux працює з процесами і потоками як з виконуваними модулями і планує їх виконання на основі глобальної стратегії планування. Код обробки сигналів також належить цьому компоненту.

Взаємодія підсистем ядра (приклади):
1) Файлові системи зазвичай звертаються до файлів через блокові пристрої. Однак для приховування великий латентності дискового доступу файли копіюються в кеш сторінок (що знаходиться в оперативній пам'яті). Деякі файли можуть навіть створюватися динамічно і мати уявлення тільки в оперативній пам'яті.
2) Система віртуальної пам'яті може використовувати дисковий розділ або область підкачки.

Крім статичних компонентів ядра Linux підтримує і динамічно завантажувані модулі. Ці модулі можуть використовуватися для додавання або заміни драйверів пристроїв за замовчуванням, файлових систем, роботи в мережі, а також інших кодів ядра.

Ядро ОС Linux (рис.2) - це фактично диспетчер ресурсов. Незалежно від типу ресурсу - процес, пам'ять або пристрій, - ядро організовує і впорядковує доступ до ресурсу множини конкуруючіх користувачів (як в режімі ядра, так и в режімі користувача).

Рис. 2.3. Спрощена схема основних підсистем ядра ОС Linux

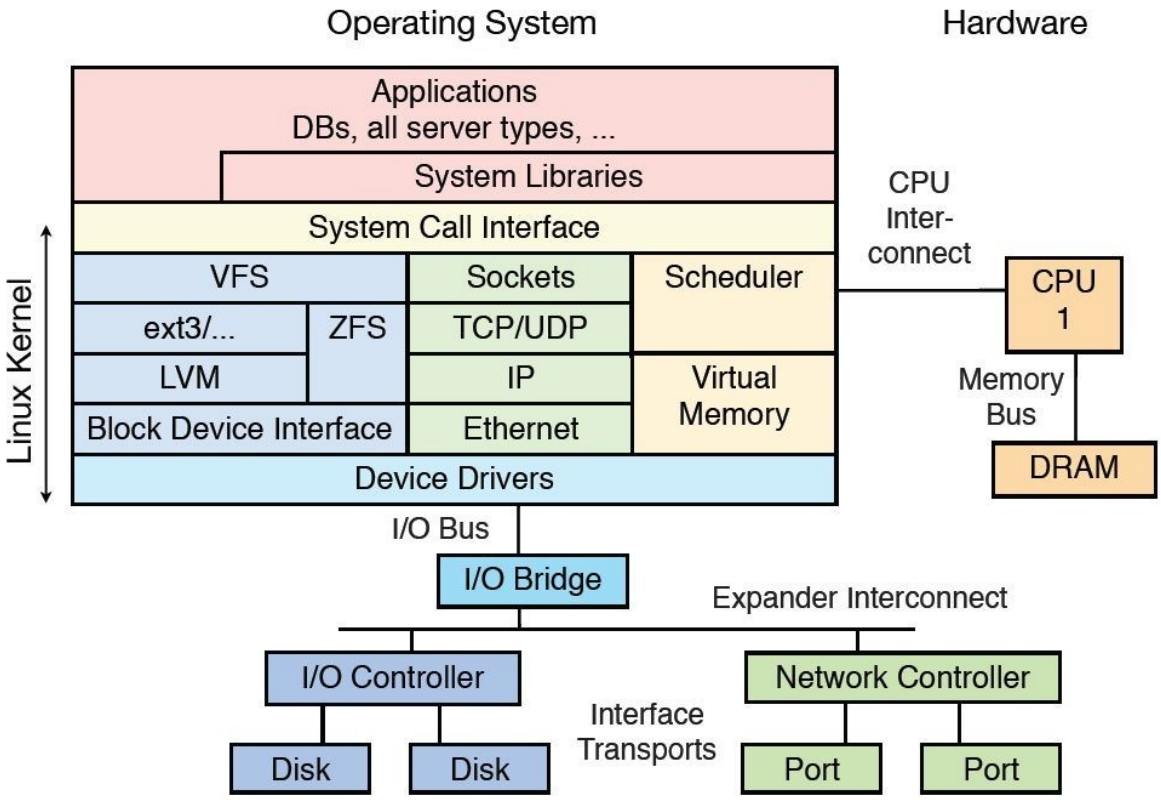


Рис. 2.4. Структура схема основних підсистем ядра ОС Linux

VFS - virtual file system (or virtual filesystem switch): уніфікований інтерфейс до конкретних файлових систем (ext3, ext4, JFS, ReiserFS, XFS)

LVM - Logical Volume Manager (manages disk drives and similar mass-storage devices)

2.4. Властивості ядра ОС Linux

- переносимість: Linux портований на більшість існуючих апаратних платформ
- мінімальні вимоги до апаратури: Linux можна запустити на «будь-якій» машині
- висока продуктивність: основна ОС для суперкомп’ютерів з top500
- архітектура: монолітне ядро з динамічно завантажуваними модулями

- використовується як «експериментальний майданчик» (відкритість для нових рішень)
- використовується в режимі супервізора віртуальних машин

Ще одне недавнє удосконалення Linux - можливість її використання в якості операційної системи для інших операційних систем (т.зв. гіпервізора). Нещодавно в ядро було внесено удосконалення, що отримало назву Kernel-based Virtual Machine (KVM, віртуальна машина на базі ядра). В результаті цієї модифікації в просторі користувача був реалізований новий інтерфейс, що дозволяє виконувати поверх ядра з підтримкою KVM інші операційні системи.
У такому режимі можна не тільки виконувати інші примірники Linux, але і віртуалізований Microsoft Windows. Єдине обмеження полягає в тому, що використовуваний процесор повинен підтримувати нові інструкції віртуалізації.

1) Стандартизація інтерфейсу системних викликів (SCI)
- UNIX System V Release 4 (SVR4);
- POSIX -> Portable Operating System Interface
2) Використання нових рішень в процесорній архітектурі
- розвиток архітекутри (системи команд) x86;
- управління обчисленнями -> переключення контексту процесів, потоки;
- багатоядерність;
- віртуалізація;
- енергозбереження.
3) Проблема підримки нових зовнішніх пристроїв -> драйвери пристроїв
4) Мобільні обчислювальні пристрої -> ОС для смартфонів та планшетів

3.1. УПРАВЛІННЯ ПРОЦЕСАМИ: ОСНОВНІ ПОНЯТТЯ

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

Процес — мінімальний програмний об'єкт, що володіє власними системними ресурсами (запущена програма).

По генеалогічній ознаці розрізняють процеси, що породжують і породжені.

По результативності розрізняють еквівалентні, тотожні і рівні процеси.

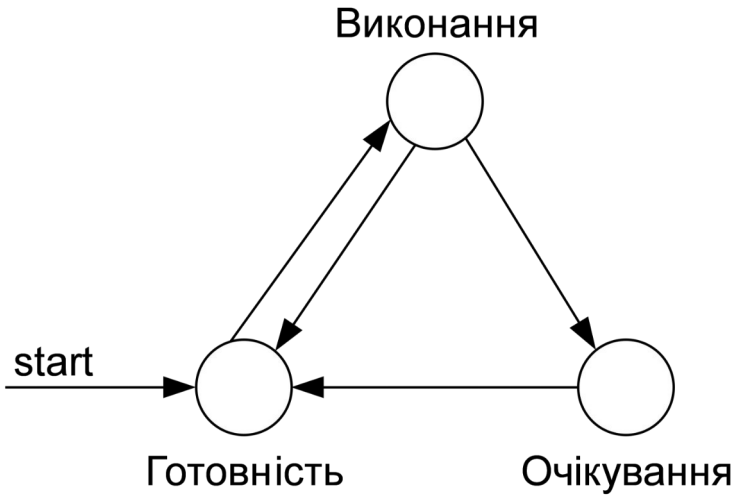


Рис. 3.1. Основні стани процесу

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

ВИКОНАННЯ - активний стан процесу, під час якого процес володіє всіма необхідними ресурсами і безпосередньо виконується процесором;

ОЧІКУВАННЯ - пасивний стан процесу, процес заблокований, він не може виконуватися по своїх внутрішніх причинах, він чекає виконання деякої події

ГОТОВНІСТЬ - також пасивний стан процесу, процес має всі необхідні для нього ресурси, він готовий виконуватися, однак процесор зайнятий виконанням іншого процесу.

3.2. УПРАВЛІННЯ ПРОЦЕСАМИ: КОНТЕКСТ ПРОЦЕСУ

Протягом існування процесу його виконання може бути багаторазово перерване і продовжене. Для того, щоб відновити виконання процесу, необхідно відновити стан його операційного середовища. Стан операційного середовища відображається станом регістрів і програмного лічильника, режимом роботи процесора, покажчиками на відкриті файли, інформацією про незавершені операції в/в, кодами помилок виконуваних даним процесом системних викликів і т.д. Ця інформація називається контекстом процесу.

Контекст процеса - значення лічильника команд, регістри, ресурси, що використовуються ОС, та інше.

Контекст поділяють на: Користувацький; Регістровий; Системного рівня;

3.3. ОРГАНІЗАЦІЯ ПЛАНУВАННЯ ПАРАЛЕЛЬНОГО ВИКОНАННЯ ПРОЦЕСІВ

- Визначення моменту часу зміни виконуваного процесу;
- Вибір процесу на виконання з черги готових до виконання процесів;
- Переключення контекстів старого і нового процесів;

Відповідно до алгоритмів, заснованих на квантуванні, зміна активного процесу відбувається, якщо:

· відбулася помилка,

· процес завершився і залишив систему,

· процес перейшов у стан ОЧІКУВАННЯ,

· вичерпано квант процесорного часу, відведений даному процесу.

Інша група алгоритмів використовує поняття "пріоритет" процесу:

- Статичний, динамічний;
- Відносний, абсолютний;

- Витісняючий, невитісняючий;
- Та інші;

3.4. ПОТОКИ УПРАВЛІННЯ

Потоки управління це основна одиниця розпаралелення в Windows.

Потік – це одиниця монопольного використання ресурсів

Потік виконання це незалежний потік управління обчисленнями який має свій лічильник команд та свій стан регістрів.

Розглядають дві концепції:

- Віртуальна пам'ять (звільнити програму від виділення пам’яті і їх накладання з іншими процесами)
- Потоків управління;

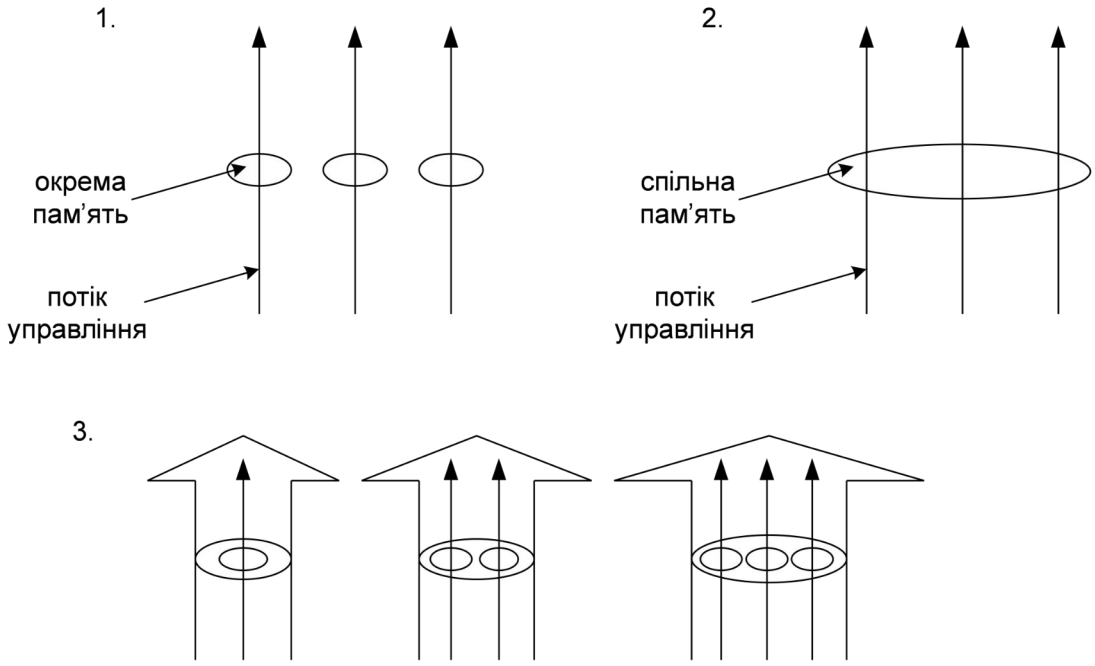


Рис. 3.2. Обчислювальні процеси та програмні потоки

3.5. Паралельне виконання обчислювальних процесів

В системах з розділенням в часі (time-sharing system) для організації паралельного виконання обчислювальних процесів використовуються:

1) черга готовності: черга процесів, готових до виконання (ready queue);

2) черга очікування: черга доступу до пристрою вводу/виводу (device queue); кожний пристрій вводу/виводу має свою окрему чергу очікування.

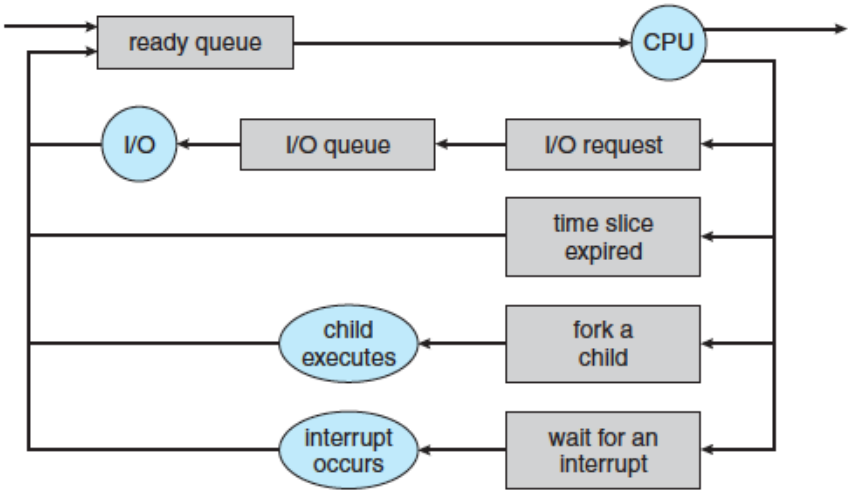


Рис. 3.3. Схема використання черг готовності та очікування

По часу планування розрізняють:

1) Коротко-термінове планування (short-term scheduler, CPU scheduler); диспетчеризація процесів завантажених в оперативну пам'ять; масштаб часу: менше 100 млс (квант часу (time quantum) процесора = 10 млс — 100 млс);

2) Довго-термінове планування (long-term scheduler, job scheduler), завантаження процесів з зовнішньої пам'яті в оперативну пам'ять (скільки паралельних процесів запустити на виконання? –> ступінь багатозадачності (degree of multiprogramming)); масштаб часу: більше 1 хв.

Процеси, які виконуються можна поділити на два класа:

1) I/O bound processes: процеси які більше часу витрачають на очікування та виконання операцій вводу/виводу ніж на обчислення;

2) CPU bound processes: процеси які які більше часу витрачають на обчислення, а операції вводу/виводу виконують рідко.

Логіка роботи диспетчера крім усього іншого зводиться до балансування кількістного співвідношення між цима двома класами процесів.

Завданння довготермінового планування: вибрати для завантаження в оперативну пам’ять збалансовану комбінацію процесів обох класів. Якщо кількість процесів 1-го класу в ОП буде набагато переважати кількість процесів 2-го класу, то черга готовності буде майже завжди пустою (в short-term scheduler не буде роботи). В протилежному випадку черга очікування буде майже завжди пустою, а в роботі з пристроями вводу/виводу будуть великі затримки.

Середньо-термінове планування (medium-term scheduler) — це спосіб забезпечити згаданий вище кількісний баланс за рахунок вивантаження деяких процесів з ОП. При цьому зберігається стан вивантажених процесів, що дає можливість їх подальшого заванатаження і продовження виконання. Ця схема роботи отримала назву свопінг (swapping).

3.6. Переключення контексту процесів (Context Switch)

Контекст процесу - це стан стан регістрів процесора та операційного середовища поточного процесу.

Зокрема в ОС UNIX розрізняють:

1) Користувацький контекст (вміст користувацького адресного простору: сегменти коду, даних, стеку + сегмент спільної пам’яті (shared memory) + сегменти файлів відображених у віртуальну пам’ять);

2) Регістровий контекст (вміст апаратних регістрів: лічильник команд, регістр стану процесора, вказівник стека, регістри загального призначення і т.д.);

3) Контекст системного рівня (структури даних ядра ОС, пов'язані з даним процесом: статична та динамічна частина (стеки, які використовуються процесом при його виконанні в режимі ядра), в тому числі дескриптор процесу).

Для забезпечення багатозадачності з розділенням в часі потрібно виконувати операцію заміни (переключення) контексту в ситуації, коли попередній процес залишає CPU, а наступний займає його місце.

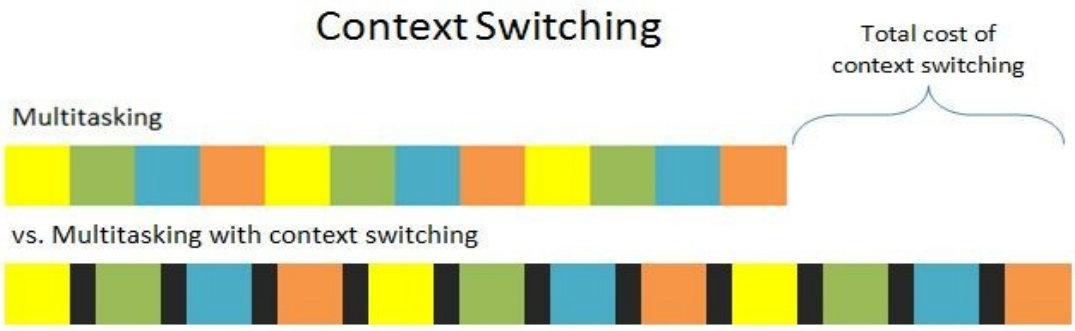


Рис. 3.4. Витрати на переключення контексту

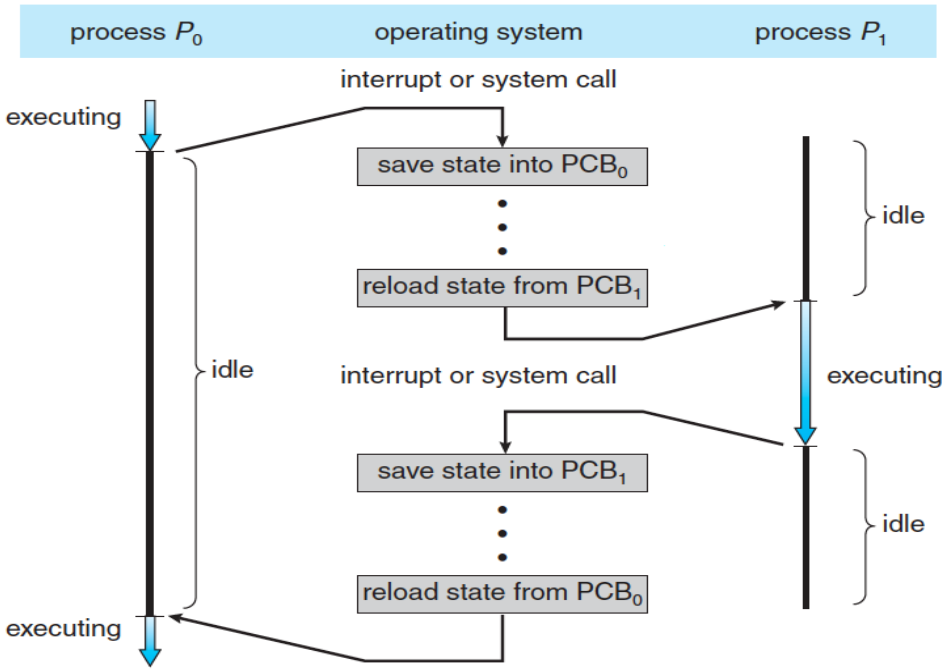


Рис. 3.5. Приклад переключення контексту

Переключення контексту складається з двох кроків (рис.3.5):

1) збереження контекста попереднього процеса;

2) завантаження контекста наступного процеса.

Збереження контексту включає:

1) збереження регістрового контекста у PCB процеса (в тому числі значення лічильника команд);

2) збереження динамічної частини системного контекста;

3) збереження службових даних по управлінню пам'яттю процеса для відновлення його адресного простору (в тому числі вміст translation lookaside buffer (TLB) = буфера асоціативної трансляції адрес віртуальної пам'яті в адреси фізичної пам'яті);

Переключення контексту може бути реалізовано: 1) апаратно; 2) програмно. В сучасних системах переключення контексту в своїй основі програмне з використанням апаратної підтримки (наприклад, в деяких процесорах реалізована можливість збереження всього регістрового контексту одною командою). Перевагою програмного переключення контекста перед апаратним є його селективність: зберігається лише вміст тих регістрів які потрібні, тоді як апаратно зберігається вміст всіх регістрів.

Втрати у продуктивності роботи («розплата» за багатозадачність):

1) витрати часу та обчислювальних ресурсів на процедуру переключення конекста;

2) при переключенні контекста відбувається очищення конвеєру команд та даних процесора;

3) вміст кеша процесора (особливо 1-го рівня), «оптимізований» під попередній процес не підходить для нового процеса;

4) при переключенні на процес, який довго не виконувався, може запуститись підкачка його сторінок пам’яті з ПЗП;

Оцінка швидкості переключення контексту:

1. «Типова швидкість складає декілько мілісекунд.» «A typical speed is a few milliseconds.» [Silberschatz et al., 2012]

2. За іншими оцінками ця швидкість складає від 1 до 200 мкс (в середньому менше 10 мкс).

3.7. Запуск та зупинка процесів на прикладі ОС Linux

1. Процеси утворюють ієрархію (дерево процесів), в якій в кожного процесу (окрім самого першого) є один батьківській процес і може бути декілько синівських процесів.

2. fork(): Створення нового процесу в UNIX-подібних ОС (в тому числі в ОС Linux) відбувається шляхом клонування батьківського процесу (створюється копія поточного процесу з новим pid).

3. exec(): В разі необхідності виконавчий код новоствореного процесу замінюється на виконавчий код вказаної програми.

4. wait(): Після створення батьківський та синівський процеси виконуються одночасно. При цьому батьківський процес перед своїм завершенням повинен дочекатися завершення синівського процесу. Завершений синівський процес, батько якого ще не викликав відповідний wait(), в UNIX-подібних ОС називається zombie.

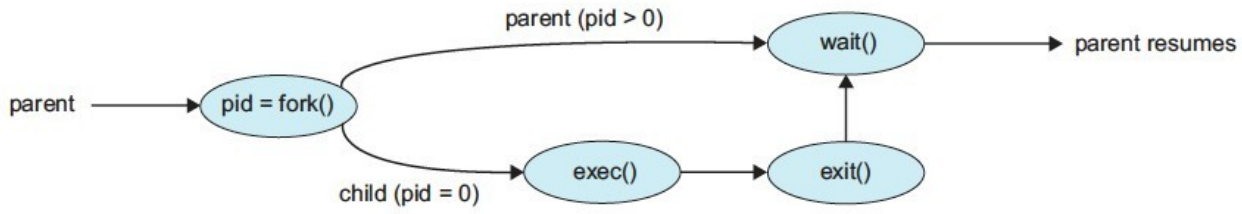


Рис. 3.6. Схема використання системних викликів fork, exec, wait, exit

Тема 4. Управління пам’яттю

4.1. ОРГАНІЗАЦІЯ ТА СПОСОБИ УПРАВЛІННЯ ПАМ’ЯТТЮ

Організація управління оперативною пам’яттю породжує специфічний клас оптимізаційних задач (основний критерій ефективності це час – час затримки t). Задача полягає в тому що потрібно передбачити яка сторінка буде використана надалі і саме її підвантажити в основну пам'ять. Використовують програмні або апаратні алгоритми передбачення. Алгоритм сам вирішує яку сторінку тре підвантажити. Існує певна локальність звертань до пам’яті. Інша задача це захист пам’яті (Memory protection). Паралельні задачі залазять в пам'ять один до одного і як результат виникнення задачі. Вирішення були наступні(пам’яті розбивають на куски):

- Кожна програма розбивається на сегменти(стеку, даних, програми). Ця ділянка має початок і зміщення в сегменті.
- Сторінкова організація пам’яті(Paging). Однакового розміру сторінки по 4К переважно.
- Сегментно-сторінкова(адреса початку сегмента, сторінки і зміщення в сторінці).

І як результат виникнення віртуальної пам’яті. Програміст працює з віртуальною пам’яттю як з реальною і не задумується про звертання процесів до одних адрес(небажане).

Три рівня організації управління пам’яті:

- Hardware MM(MMU)- Пристрій управління пам’яттю;
- Рівень операційної системи(operating system);
- Application Memory Menager. Garbage collection (GC).

4.2. РІЗНИЦЯ МІЖ СЕГМЕНТНОЮ ТА СТОРІНКОВОЮ ОРГАНІЗАЦІЄЮ ПАМ’ЯТІ

• paging реалізує розподіл пам’яті та підкачку більш легко;

• ніякої зовнішньої фрагментації;

• кожен процес поділяється на множину маленьких, фіксованого розміру частин, так званих сторінок (pages);
• фізична пам’ять поділяється на велику кількість маленьких, розміру частин, так званих фреймів (frames).
Сторінкова організація пам’яті(Paging). Однакового розміру сторінки по 16К переважно.
Сегментна організація пам’яті (стеку, даних, програми). Ця ділянка має початок і зміщення в сегменті.

4.3. УПРАВЛІННЯ ПАМ’ЯТТЮ: ПРИНЦИП ЛОКАЛЬНОСТІ ЗВЕРТАНЬ

Якщо програма читає дані або код програми то швидше за все наступна частина даних буде знаходитись недалеко від останньо-використаної. Це використовується як основа при прогнозуванні потрібних даних в основній пам’яті. Цей принцип локальності можна проаналізувати заздалегідь проаналізувавши код програми(цим займаються деякі модулі оптимізуючих компіляторів), можливе і апаратне рішення. Пам’ять розглядався як пасивний пристрій, але реально зараз розглядають його як активний, який може взяти на себе деякі функції(інтегрований контролер).

4.4. ОСНОВНІ ВИДИ ЛОКАЛЬНОСТІ ЗВЕРТАНЬ

1) Локальність звертань в часі (temporal locality),
2) Локальність звертань в просторі (spatial locality),
3) Еквідистантна локальність звертань (equidistant locality),
4) Локальність розгалуження (branch locality).

4.5. УПРАВЛІННЯ ПАМ’ЯТТЮ: ЛОКАЛЬНІСТЬ У ЧАСІ (TEMPORAL LOCALITY). Якщо відбулося звертання по деякій адресі, то наступне звертання по цій же адресі з великою імовірністю відбудеться найближчим часом (tempral proximity).

4.6. УПРАВЛІННЯ ПАМ’ЯТТЮ: ЛОКАЛЬНІСТЬ У ПРОСТОРІ (SPATIAL LOCALITY)

Якщо відбулося звертання по деякій адресі, то з високим ступенем імовірності найближчим часом відбудеться звертання до сусідніх адрес.

4.7. УПРАВЛІННЯ ПАМ’ЯТТЮ: ЕКВІДИСТАНТНА ЛОКАЛЬНІСТЬ

Еквідистантна локальність (Equidistant locality). Ділянки пам’яті до яких відбуваються звертання утворюють регулярну (передбачувану) структуру у просторово-часовому координатному просторі. За простою лінійною функцією можна прогнозувати до якої ділянки пам’яті буде звертання в наступний момент часу. Такі структури утворюються в циклах. Основна мета знаходження регулярності звертаннях, не обов’язково по лінійному закону але функції з повтором.

4.8. АВТОМАТИЧНЕ УПРАВЛІННЯ ВИДІЛЕННЯМ ТА ЗВІЛЬНЕННЯМ ПАМ’ЯТІ

Стратегія автоматичного управління пам'яттю (Garbage Collection, GC) складається з 2 кроків:

- Знаходження об’єктів даних які не будуть використовуватись в подальшому(сміття);
- Вивільнення ресурсів які споживають ці об’єкти;

При кожному оголошені змінної під неї виділяється область пам’яті. GC працює в нескінченному циклі збору даних про сміття, використовують різного роду програмування.

4.9. УПРАВЛІННЯ КІЛЬКІСТЮ СТОРІНОК ВИДІЛЕНИХ ПРОЦЕСУ

1. Управління заміщенням сторінок. Кількість сторінок що підвантажується в основну пам'ять постійна для процесу. Page fault.
2. Управління кількістю сторінок. Кількість сторінок змінна.

Задача на розподіл ресурсів:

Кожному процесу потрібно нарізати ресурсів з загально доступного.

Жодний з сучасних алгоритмів заміщення сторінок не гарантує захисту від trashing (попадання процесу в область де PF збільшена вище допустимої верхньої границі). При побудові цих алгоритмів визначають робочу множину (working set) W(t,T), де робоча множина це набір сторінок $p_1,p_2,...p_n$ які активно використовуються процесом. Робоча множина визначається для моменту часу t та вікна робочої множини T.

Рис. 4.1. Вікно робочої множини на осі часу

Якщо на осі часу поставити точку t то важлива глибина пам’яті, які сторінки попадають в множину а які ні. Вікно спостереження це проміжок часу на протязі якого ми спостерігаємо за сторінками. Тоді ми можемо визначити скільки сторінок може потрапити в робочу множину W(t,T) -> m^*.

Формування набору сторінок змінюється в часі, хоча в принципі є досить стабільним.

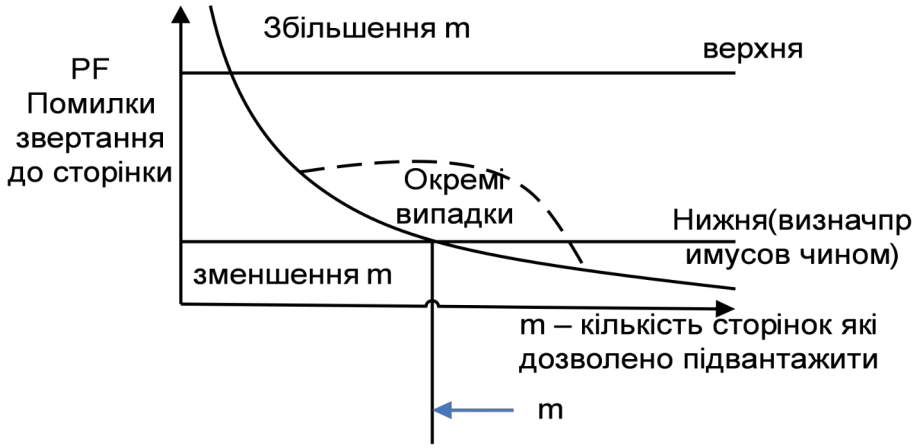


Рис. 4.2. Залежність кількості сторінок від частоти помилок звертання до сторінки

5.1. ФІЗИЧНА ОРГАНІЗАЦІЯ ПРИСТРОЇВ ВВОДУ/ВИВОДУ

Пристрою вводу-виводу поділяються на типи:

Блок-орієнтовані пристрої зберігають інформацію в блоках фіксованого розміру, кожний з який має свою власну адресу (диски).

Байт-орієнтовані пристрої не адресовані і не дозволяють здійснювати операцію пошуку. Вони генерують чи використовують послідовність байтів. Прикладами є термінали, рядкові принтери.

Мережні пристрої –карти, мережні адаптери.

Аудіо та відео пристрої.

Таймери

Зовнішній пристрій звичайно складається з механічного (власне пристрій) й електронного компонента (контролером пристрою чи адаптером). Деякі контролери можуть керувати декількома пристроями і мають програмну модель (регістри команд, статусу та даних).

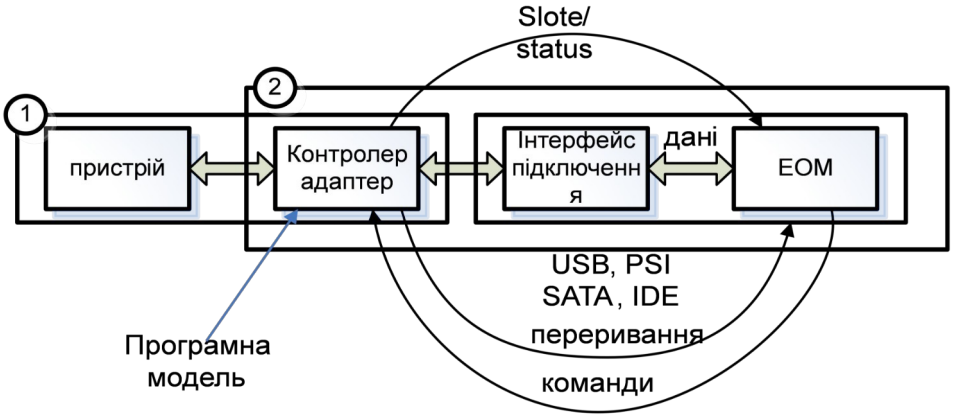


Рис. 5.1. Схема організації роботи пристроїв вводу-виводу

Варіанти фізичної організації в/в:

1- Пристрій разом зконтролларом знаходиться за межами ЕОМ

2- Контролер знаходиться в корпусі ЕОМ

3- Пристрій і вся система знаходиться в корпусі ЕОМ (HDD)

5.2. ОРГАНІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВВОДУ/ВИВОДУ

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

Ключовим принципом є незалежність від пристроїв. Вигляд програми не повинен залежати від того, чи читає вона дані з гнучкого чи з твердого диска. Проблема обробки помилок вирішується контролером.

Ще одне ключове питання – це використання блокувальних (синхронних) і неблокувальних (асинхронних) передач.

Мета:

- Забезпечити зручність програмування в рамках подолання семантичного розриву.
- Забезпечити взаємонезалежність програмного забезпечення та пристроїв різних типів.

5.3. СХЕМА БАГАТОРІВНЕВОЇ ОРГАНІЗАЦІЇ ПЗ ВВОДУ/ВИВОДУ

Абстракція незалежності (приховування)

1. Концепція потоків в/виводу.
2. Відображення пристроїв на файлову систему.

Завдання.

- Приховування при одночасному доступі до одного пристрою виникає проблема некоректного змагання паралельних процесів за доступом до пристрою.
- Spoding (розмотування) – процес моніторингу пристрою в/вив, який вирішує проблему одночасного доступу, формуючи через каталог звернень до пристроїв. Альтернативою може бути механізм монопольного захоплення.

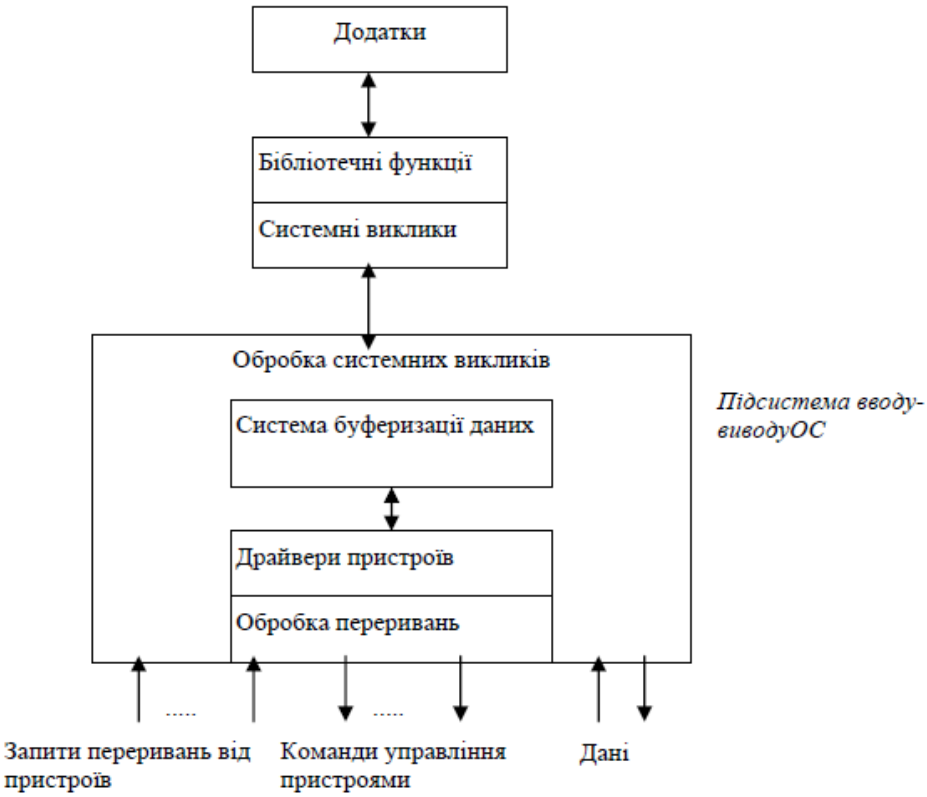


Рис. 5.2. Схема роботи програмного забезпечення вводу-виводу

5.3. УПРАВЛІННЯ ВВОДОМ/ВИВОДОМ: ОБРОБКА ПЕРЕРИВАНЬ

Існують 2 ідеології побудови систем:

1. Офісні, лабораторні системи – тотальна передбачуваність їх роботи. Для таких систем намагаються зменшити кількість виникнення переривань та локалізувати їх в надрах ОС.
2. Системи реального часу. Обробка переривань – центральний момент роботи системи. Характерна непередбачуваність. Управління процесами обробки переривань здійснює контролер переривань.

В сучасних контролерах переривань реалізована складна система пріоритетів яка використовується при побудові черги переривань на обробку.

5.4. УПРАВЛІННЯ ВВОДОМ/ВИВОДОМ: ДРАЙВЕРИ ПРИСТРОЇВ

Драйвер (англ. driver) - це комп'ютерна програма, за допомогою якої інша програма (зазвичай операційна система) Отримує доступ до апаратного

забезпечення деякого пристрою. У загальному випадку, для використання будь-якого пристрою (як зовнішнього, так і внутрішнього) необхідний драйвер. Операційна система керує деяким «віртуальним пристроєм», що розуміє стандартний набір команд. Драйвер переводить ці команди в команди, які розуміє безпосередньо пристрій.

Тема 6. Управління файловою системою

6.1. ПОНЯТТЯ ФАЙЛОВОЇ СИСТЕМИ

Файлова система - це частина ОС призначена для зручної організації та представлення даних, що зберігаються у зовнішніх пристроях пам'яті.

Файлова система може також означати:

- Сукупність всіх файлів на диску;
- Набір структур даних, які використовуються для управління даними;
- Набір системного ПЗ, який забезпечує управління файлами.

FAT(File Allocation Table): FAT16, FAT32

UFS(Unix File System)

NTFS(NT File System)

6.2. ІМЕНУВАННЯ ФАЙЛІВ (ІЄРАРХІЧНА СИСТЕМА ІМЕН)

Принцип отримати доступ до ресурсу, означає дізнатись його імя. З точки зору зручності для користувача файлам надаються символьні імена. Основна умова унікальність імені файлу.

Ідея ієрархії імен: згідно неї простір імен реалізується у вигляді направленого графу.

Паралельна інформаційна структура для системи(несимвольна).

Приклад: лінійний каталог номерів файлів. (в Unix i-node)

n2:alc n3:cde n4:four

6.3. ОБ'ЄКТИ ФАЙЛОВОЇ СИСТЕМИ

- Файл даних(звичайний файл);
- Спеціальні файли(файли пристрою). Файл асоційований з пристроєм вводу/виводу;
- Каталоги (вузли в ієрархічній системі імен);

Каталог – це файл, який містить системну інформацію про файли і каталоги, які входять в цей каталог.

Способи збереження файлу:

1. Вся інформація про файл і каталог міститься в цьому файлі;
2. Коли ця інформація міститься у вигляді посилань на відповідні рядки в службових таб.

Посилання (лінки) призначені для переконфігуровування дерева імен у зручну форму;

Типи лінків: Hard link та Simbolic(Soft) link.

6.4. ЛОГІЧНА ТА ФІЗИЧНА ОРГАНІЗАЦІЯ ФАЙЛУ

Логічна організація файлу призначена для програміста (включає початок, запис, кінець). Лінійна послідовність запису символів (потік символів) з вказівником початку файлу.

В результаті виникають певні ускладнення:

- Розбиття на рядки(однакової довжини, блоки), тому потрібний символ кінця рядка;
- Записи з ключами або індексами

Приклад: теги HTML/XML. При цьому ускладнюється семантичний розбір.

Фізична організація файлу – як файл розміщено у пристрої пам’яті (залежить від специфіки цього пристрою).

Основним принципом фізичної організації файлу, що файли розбиваються на блоки:

1. Лінійно-послідовне розміщення блоків;
2. Розміщення блоків зв’язаним списком;

Існують методи «боротьби» з фрагментацією дискового простору (дефрагментаційні утиліти)

Існують методи розміщення нових файлів, в такий спосіб, щоб зменшити фрагментацію диску.

6.5. АРХІТЕКТУРИ ФАЙЛОВИХ СИСТЕМ

1. Окремі логічні диски з незалежними файловими системами (використовують драйвери файлових систем, над якими розташований менеджер файлових систем, що ними керують).
2. Загальна файлова система до якої підключаються підпорядковано частинами файлові системи зовнішніх пристроїв пам’яті (Unix)

Інші архітектурні особливості:

1. Внесення змін в файлову систему. Файлова система з веденням журналу в якій реалізована можливість відкату.
2. Стиск даних
3. Безпека (шифрування/дешифрування даних)
4. Віртуальні файлові системи (у складі віртуальних машин)

Альтернативні способи організації даних:

1. Система лінійних каталогів(бібліотечний каталог). Підходять для збереження однорідних та однотипних даних.
2. Реляційні бази даних.
3. Гіпертекстові системи(симантичні мережі)

Тема 7. Мережна підсистема ОС

7.1. Призначення та структура мережної підсистеми ОС

Розглянемо призначення та структуру мережної підсистеми (Networking subsytem, Network Stack) на прикладі мережної підсистеми ОС Linux. Структура мережної підсистеми ОС Linux, як і переважна більшість мережних підсистем сучасних ОС, в цілому відображає багаторівневу архітектуру стеку мережних протоколів (рис.1). Призначення: 1) Мережна підсистема дозволяє системі Linux з'єднуватись з іншими системами в комп'ютерній мережі. 2) Існують різні мережні пристрої (мережні адаптери Ethernet, Wi-Fi та ін.), підримка яких реалізована в ОС, та різні мережні протоколи, які можуть використовуватись для обміну даними. Відтак мережна підсистема забезпечує абстрагування від низькорівневих деталей роботи як пристроїв, так і протоколів, внаслідок чого користувацькі процеси та інщі підсистеми ядра можуть користуватись мережою в стандартний уніфікований спосіб без обов'язкового уточнення який саме мережний пристрій та протокол застосовуються.

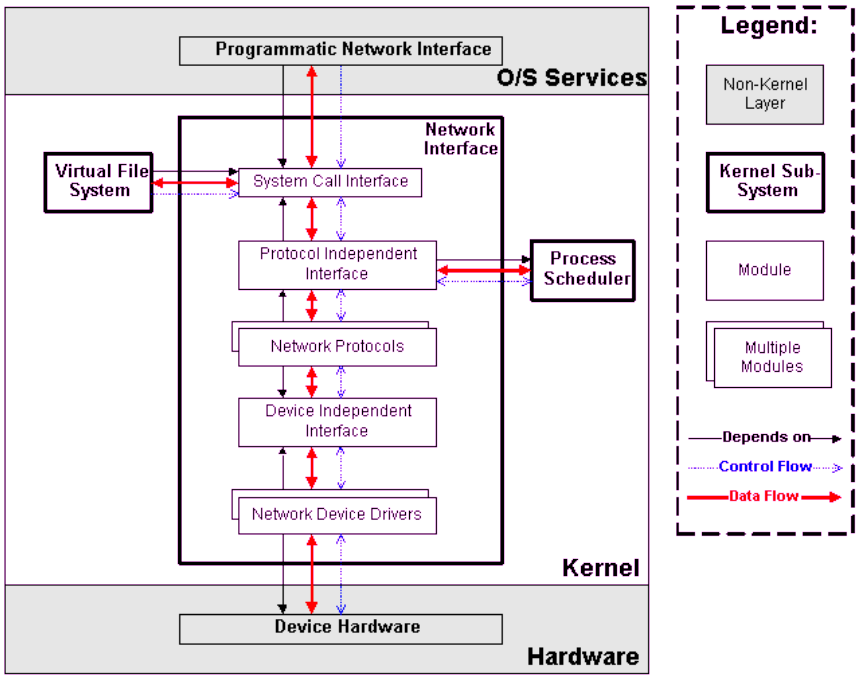


Рис.7.1. Структура мережної підсистеми ОС Linux

Основні складові (модулі) мережної підсистеми (рис.7.1):

1) Драйвери мережних пристроїв (Network device drivers) забезпечують взаємодію з відповідними апаратними пристроями. Кожним наявним мережним пристроєм керує свій окремий драйвер (часто використовується назва: NIC driver, де NIC - network interface controller/card).

2) Модуль уніфікованого інтерфейсу мережних пристроїв (Device independent interface) забезпечує стандартний інтерфейс доступу до апаратнго забезпечення всіх мережних пристроїв, приховуючи від інших модулів мережної підсистеми низькорівневі деталі їхньої роботи (наприклад, приховування різниці між роботою Ethernet та Wi-Fi).

3) Модулі мережних протоколів (Network protocols) забезпечують логіку роботи стеку протоколів (наприклад, стеку протоколів TCP/IP).

4) Модуль уніфікованого інтерфейсу до стеку протоколів (Protocol independent interface) забепечує стандартний інтерфейс доступу до стеку протоколів, незалежний від апраратних засобів та мережних протоколів. Цей модуль використовуються іншими підсистемами ядра для доступу до мережі без конкретної прив'язки до протоколів і апаратури.

5) Модуль інтерфейсу системних викликів (System calls interface) обмежує набір функцій мережної підсистеми, які можуть бути використані користувацькими процесами.

В ОС Linux та більшості інших UNIX-подібних ОС використовується реалізація модулів (4) та (5) під назвою сокети Берклі (Berkeley sockets). Сокети Берклі — це бібліотека функцій та відповідний програмний інтерфейс транспортного рівня (див. модель OSI). Сокети Берклі забезпечують уніфікований інтерфейс до широкого набору протоколів (IP, UDP, TCP та ін.) та надають старндартний механізм організації обміну даними за схемою «клієнт-сервер» (рис.7.2).

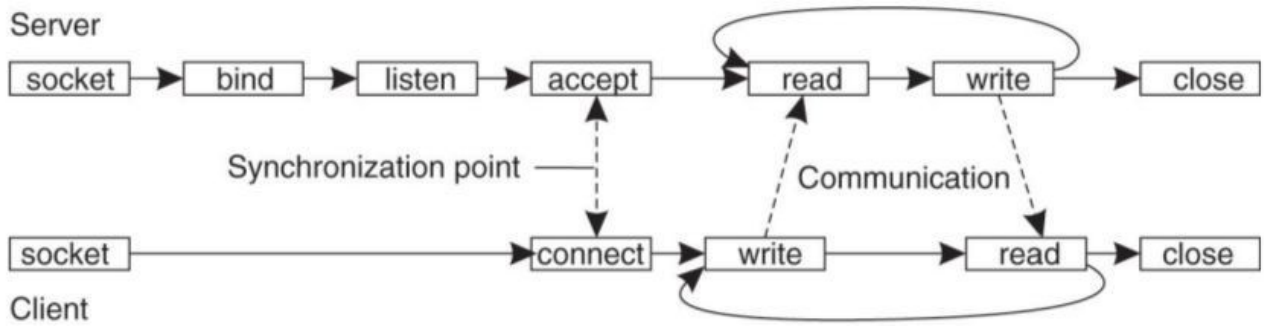


Рис.7.2. Схема роботи сокетів Берклі

Основні функції мережної підсистеми ОС Linux:

1) надання програмного інтерфейсу транспортного рівня (сокети Берклі);

2) підтримка стеку мережних протоколів (TCP/IP, UDP/IP, IPX/SPX, AppleTalk та ін.);

3) маршрутизація (routing) та пересилання (forwarding) пакетів даних;

4) фільтрація пакетів даних (модуль Netfilter);

5) управління трафіком на мережевому рівні (IP);

6) диспетчеризація пакетів даних (network scheduler);

7) надання абстракції мережних інтерфейсів (network interfaces).

Напрямки розробки мережної підсистеми ОС Linux:

1) Розробка вбудованих процедур обробки пакетів даних. Приклади: модуль Netfilter (фільтрація пакетів, NAT) та процедури управління мережним трафіком. Дані процедури передбачають наявність коду, роботу якого може конфігурувати користувач, у базовому процесі обробки пакетів даних.

2) Підвищення продуктивності програмних реалізацій логіки роботи протоколів (підвищення пропускної здатності, зменшення затримки). Приклад: вдосконалення алгоритмів управління навантаженням (congestion control) для протоколу TCP.

3) Підвищення ефективності обробки пакетів даних (збільшення максимальної кількості пакетів, які можуть бути оброблені за одиницю часу, шляхом зменшення відповідних обчислювальних витрат (циклів CPU) та обсягів пам'яті). Приклади: розпаралелення обробка пакетів на різних рівнях стеку

протоколів (stack parallel processing), прогнозування значень полів заголовку пакетів (header prediction), зменшення кількості копіювань вмісту внутрішніх буферів, розвантаження ядра за рахунок обчислення контрольних сум мережним адаптером (checksum offload).

7.2. Організація роботи мережної підсистеми ОС Linux

Робота мережної підсистеми ОС Linux базується на принципі реагування на події різними модулями на противагу єдиному потоку управління роботою всієї підсистеми (рис.7.3). Приклади подій: системні виклики від користувацьких процесів та інших модулів ядра, апаратні переривання від мережного адаптера і таймера, програмні переривання, тощо.

Мережна підсистема ядра, теоретично, майже уся може виконуватись в режимі користувача: для таких операцій, як формування пакетів TCP/IP ніякі привілеї режима ядра не потрібні. Однак у сучасних ОС (тим більше у ОС, які встановлюються на високонавантажених серверах) від мережного стеку вимагається максиальна продуктивність. Однак мережна підсистема, яка б працювала у режимі користувача, була б змушена постійно звертатись до ядра для «спілкування» з мережним обладнанням, що призвело б до суттєвих накладних витрат. Тому мережні підсистеми (як в Linux, так і в інших ОС) розміщують у складі ядра ОС.

Потоки управління роботою мережної підсистеми ОС Linux можна поділити на три основні групи:

1) Потоки управління в контексті користувацького процесу (зелені стрілки на рис.7.3). Зокрема потік управління (1) займається обробкою отриманого від користувацького процесу системного виклику, який не призводить до передачі пакету даних (отримання даних, контрольні функції налаштування TCP-

з'єднання та ін.). Потік урпавління (2) займається обробкою системного виклику, який призводить до передачі пакета даних. При цьому дані з користувацького процесу спочатку передаються модулю опрацювання TCP, в якому формуються відповідні пакети даних, які в свою чергу передаються програмним модулям нижчого рівня (IP/Ethernet). Після обробки на цих рівнях пакети даних потрапляють у чергу відправки пакетів (Transmit Queue). Контролер черги згідно заданого порядку (в термінології Linux: дисципліна черги (qdisc, queue discipline)) скеровує пакети до драйвера (NIC Driver) для перадчі в апаратну пам'ять мережного адаптера (після чого вони надсилаються по мережі отримувачу). Шляхом зміни дисципліни черги (qdisc) в ОС Linux здійснюється управління мережерним трафіком (по замовченню в якості qdisc використовуються проста схема FIFO).

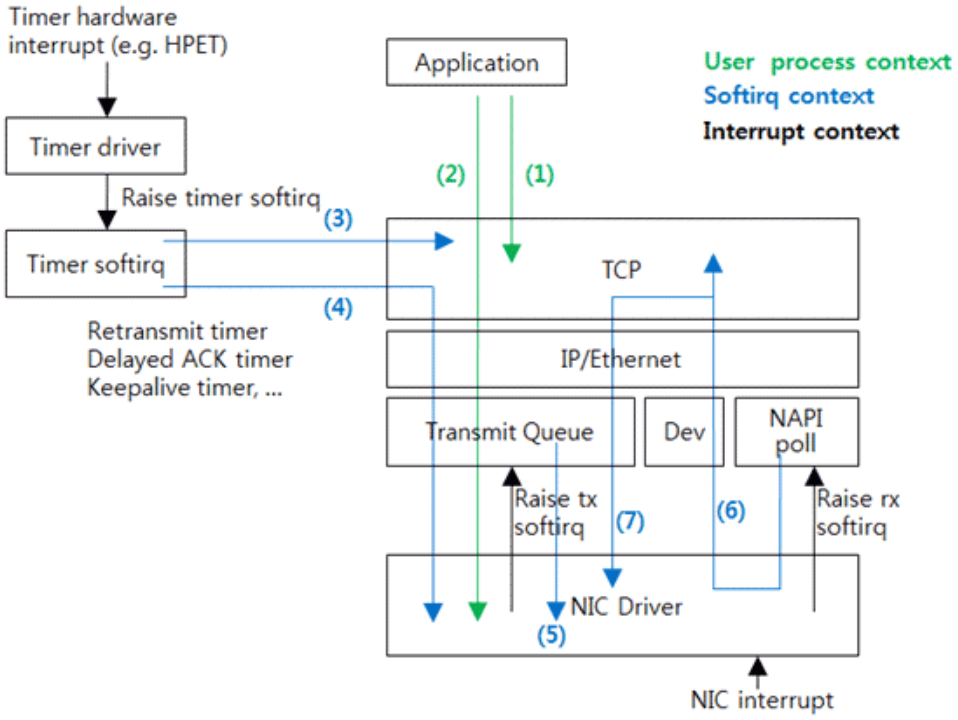


Рис. 7.3. Схема організації роботи мережної підсистеми ОС Linux.

2) Потоки управління в контексті програмних переривань (сині стрілки на рис.7.3). Програмні переривання (в термінології Linuxsoftirq — software interrupt request) — це спеціальний механізм ядра Linux, логіка роботи якого імітує обробку апаратних переривань. У випадку мережної підсистеми цей

механізм застосовується для виконання її внутрішніх завдань, які не пов'язані напряму із взаємодією з користувацькими процесами. Зокрема потік управління (3) займається обробкою спрацювання таймеру, встановленого модулем TCP, наприклад, для визначення часу завершення очікування (TIME-WAIT), після якого модуль TCP видалить відповідне з'єдання. Потік управління (4) так само займається обробкою спрацювань таймеру, встановленого модулем TCP, яка, на відміну від (3), призводить до відправки пакету даних (наприклад, повторна відправка пакету після закінчення часу очікування підтвердження (ACK) від отримувача пакету). Потік управління (5) займається скеруванням пакетів даних з черги відправки (Transmit Queue) до драйвера (відповідне програмне переривання генерується самим драйвером). Потік управління (6) займається отриманням пакетів даних з використанням механізму опитування (модуль NAPI poll) для розвантаження ядра від обробки великої кількості апаратних переривань по кожному отриманому пакету (механізм опитування (polling) вмикається тоді, коли кількість отримуваних пакетів перевищує задане порогове значення). Потік управління (7) займається випадками, які потребують додаткової відправки TCP-пакетів.

3) Потоки управління в контексті апартних переривань (чорні стрілки на рис.7.3). Мережна підсистема має справу з двома типами апаратних переривань: 1) переривання від таймера (ці переривання відповідно генерують програмні переривання потоків управління (3) та (4)); 2) переривання від мережного адаптера (NIC interrupt), які зокрема генерують програмні переривання потоку управління (5) (Raise tx softirq — обробка відправки пакетів) та потоку управління (6) (Raise rx softirq — обробка отримання пакетів).

7.3. Відправка та отримання даних мережною підсистемою ОС Linux

Відправка даних починається з виклику функції (наприклад, функції write) у користувацькому процесі. В якості «пункту призначення» вказується файловий

дескриптор (fd) сокета (який був створений раніше з вказанням адреси отримувача). Крім цього вказується які дані треба передати (buf) та їх розмір (len). Даний виклик функції запускає обробку відповідного системного виклику в режимі ядра. На рівні роботи з сокетами (Sockets) дані, які треба відправити по мережі, копіюються у буфер для відправки (send socket buffer), після чого керування передається на рівень транспортного протоколу (наприклад, TCP), далі на рівень IP і на рівень Ethernet. На кожному з цих рівнів відпрацьовує логіка відповідного протоколу та додаються відповідні заголовки до пакета даних. На останньому кроці дані передаються драйверу мережного адаптера (Driver), який скеровує їх до мережного адаптеру (NIC).

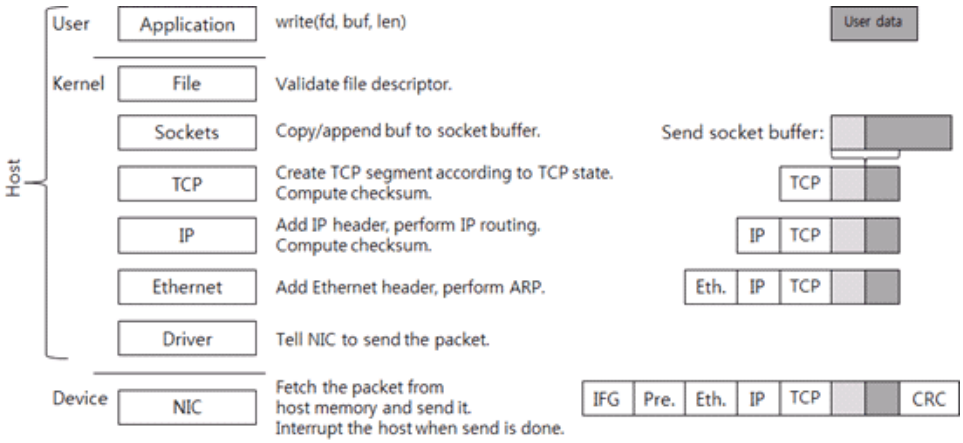


Рис. 7.4. Основні етапи відправки даних мережною підсистемою ОС Linux

Отримання даних (рис.7.5) починаєтья з надходження пакета даних до пам'яті мережного адаптера (NIC), який генерує відповідне апаратне переривання. Далі пакет даних скеровується на обробку драйвером (Driver). Після цього у відповідність до пакету ставиться примірник структури sk_buff, яка відображає структуру пакета (це потрібно для того, щоб наступні вищі рівні обробки знали, що з ним робити). Пакет передається послідовно рівням обробки: Ethernet, IP, TCP, кожний з яких виконує необхідні дії по обробці пакета згідно логіки роботи відповідного протоколу. Зрештою рівень транспортного протоколу (TCP на рис.7.5) розміщує отримані дані у буфер для отриманих даних (Receive socket buffer) рівня роботи с сокетами. Рівень роботи з сокетами, отримавши від

користувацького процесу запит на отримання даних (наприклад у вигляді функції read), копіює дані з буфера для отриманих даних (Receive socket buffer) у структуру даних користувацького процесу (buf).

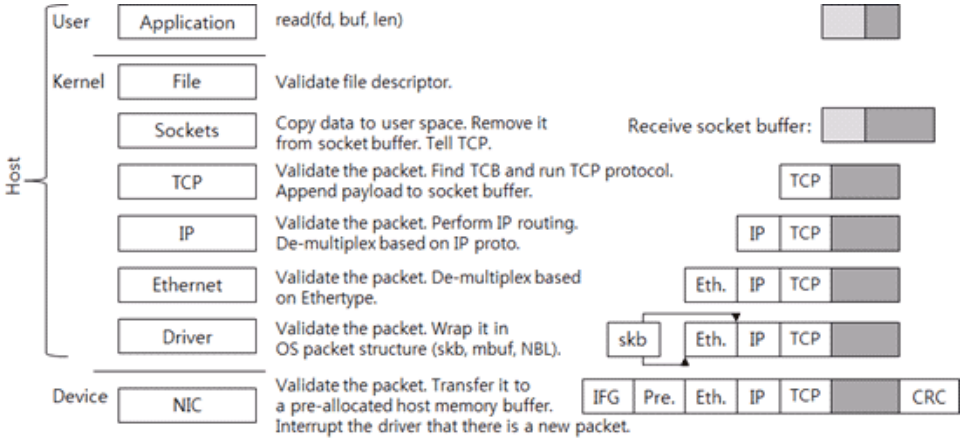


Рис. 7.5. Основні етапи отримання даних мережною підсистемою ОС Linux

Структури даних ядра, які використовуютсья для проміжного збереження даних, які відправляються (рис.7.6) мережною підсистемою: 1) буфер для відправки (Send socket buffer) рівня обробки сокетів, 2) черга відправки пакетів (Transmit Queue), 3) кільцевий буфер драйвера для відправки пакетів (TX ring). Структури даних ядра, які використовуютсья для проміжного збереження даних, які отримуються (рис.7.7) мережною підсистемою: 1) кільцевий буфер драйвера для отримання пакетів (RX ring), 2) буфер для отримання (Receive socket buffer) рівня обробки сокетів. Контроль за коректним використанням цих структур даних є одною з важливих функцій мережної підсистеми.

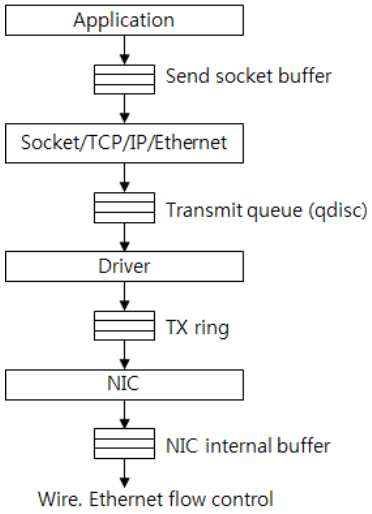


Рис. 7.6. Структури даних для відправки пакетів

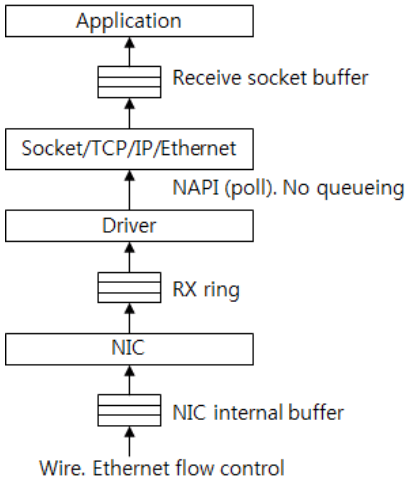


Рис. 7.7. Структури даних для отримання пакетів

7.4. Диспетчеризація пакетів даних в мережній підсистемі ОС Linux

На вузлах мережі з комутацією пакетів, як правило завжди виконується, так званий, диспетчер пакетів даних (network scheduler, packet scheduler), який визначає послідовність пакетів в черзі відправки (transmit queue) та черзі отримання (receive queue) (тобто визначає в якій послідовності відсилати пакети у мережу та «приймати» з мережі, тобто передавати прийняті пакети на вищі рівні обробки). Існує декілько різних програмних реалізацій такого диспетчера для ядер різних ОС, які основані на різних алгоритмах обробки черги пакетів даних.

В мережній підсистемі ОС Linux в якості базового диспетчера пакетів даних (в термінології Linux: дисципліни черги (qdisc, queue discipline), див. п.7.2) використовується дисципліна pfifo_fast (рис.7.8), яка реалізує простий принцип FIFO. Дана дисципліна використовує три черги з різними пріоритетами (FIFO 0, FIFO 1, FIFO 2). Пакети в черзі FIFO 0 мають найвищий пріоритет (тобто будуть відправлені чи отримані першими), пакети в черзі FIFO 2 мають

найменший пріоритет. Пакети розкладаються по цим чергам в залежності від значення флагу ToS (Type of Service) в кожному пакеті (значення ToS призначаються пакетам на етапі їх формування на рівні IP).

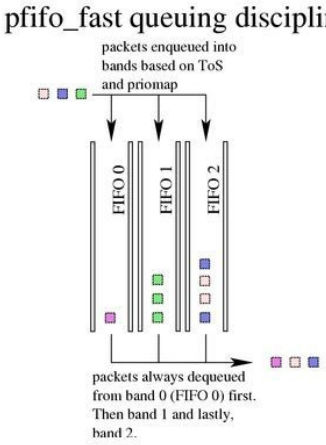


Рис. 7.8. Схема роботи дисципліни pfifo_fast.

Для кожного мережного інтерфейсу (див. п.7.5) можна призначити іншу відмінну від базової (pfifo_fast) дисципліну (схему диспетчеризації пакетів). За допомогою цього механізму в ОС Linux виконуєтья управління мережним трафіком (наприклад, обмеження швидкості вихідного трафіку за допомогою дисципліни TBF (Token Bucket Filter) та ін.).

7.5. Мережні інтерфейси в ОС Linux

Основним об'єктом мережної підсистеми OCLinux є мережний інтерфейс (network interface). Мережний інтерфейс в ОС Linux – це абстрактний іменований об'єкт, що використовується для передачі даних через деяку лінію зв'язку без прив'язкі до її реалізації (тобто до низькорівневих деталей її функціонування).

Наприклад, якщо в системі є інтерфейс eth0, то у більшості випадків на сучасних комп'ютерах він поставлений у відповідність Ethernet-адаптеру, який вбудований в материнську плату. Інтерфейс з іменем ppp0 відповідає за з'єднання «точка-точка» з іншим комп'ютером. Інтерфейс з іменем lo є

віртуальним і моделює замкнений сам на себе (вихід підключений безпосередньо до входу) мережний адаптер.

Основне завдання мережного інтерфейсу — асбстрагуватись від фізичної складової каналу. В такий спосіб програми і система будуть використовувати один і той самий метод «відправити пакет» для відправки даних через будь-який інтерфейс (lo, ethX, pppY, тощо), і так само використовувати один і той самий метод «отримати пакет». Таким чином забезпечується уніфікований API обміну даними, що не залежить від носія.

Для визначення того, які мережні інтерфейси присутні в системі, можна скористатись командою: ifconfig -a. В наведеному прикладі (рис.7.9) в системі два активних мережних інтерфейса (eth0 і lo), для кожного з яких виведена інформація про їх стан, налаштування та параметри (а також стан відповідних апаратних засобів для eth0). Зокрема, поле Link encap містить тип інтерфейсу, HWAddr – апаратну адресу пристрою (наприклад, MAC-адресу для Ethernet), MTU – максимальный розмір пакетів даних, що відправляються. Також в окремих полях міститься статистика того, який об'єм даних був відправлений та отриманий через відповідний мережний інтерфейс.

```
$ ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:11:2F:A8:DE:A4
          inet addr:172.23.2.114  Bcast:172.23.2.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
          Interrupt:11 Base address:0x4000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:60 errors:0 dropped:0 overruns:0 frame:0
          TX packets:60 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:4707 (4.5 KiB)  TX bytes:4707 (4.5 KiB)
```

Рис. 7.9. Приклад виклику команди ifconfig.

Найбільш часто зустрічаються наступні мережні інтерфейси:

1) eth (Ethernet) – звичайно відповідає октремому мережному адаптеру;

2) ppp (Point-To-Point) – з'єднання «точка-точка» з іншим комп'ютером;

3) wl або wlan (Wireless) – безпровідний інтерфейс;

4) lo (Loopback) – віртуальний «замкнений» мережний інтерфейс (використовується для взаємодії «по мережі» між користувацькими процесами одного комп'ютера).

Командою ifconfig також можна скористатись для зупинки (відключення) та активації (включення) мережного інтерфейсу, а також для зміни його параметрів. Про біжучий стан мережних інтерфейсів можна дізнатись командою: netstat -i. Також за допомогою команди netstat можна дізнатись про всі наявні мережні з'єднання (у вигляді списку всіх локальних мережних портів транспортного рівня): netstat -a. Відповідно список «активних» мережних з'єднань, по яким в даний момент часу відбувається обмін інформацією, можна отримати командою: netstat -a|grep ESTABLISHED. Список біжучих мережних з'єднань у вигляді списку IP-сокетів можна отримати командою: lsof -i. Також список усіх сокетів можна отримати командою: ss (socket statistics).

Розглянемо порядок передачі даних в термінах абстракції мережних інтерфейсів. Нехай користувацький процес намагається відправити якісь дані через мережу. Перед відправкою даних, за допомогою викликів функцій програмного інтерфесу сокетів Берклі (bind, connect та ін.) буде встановлено потрібне мережне з'єднання. Це з'єднання буде доступне для користувацього процесу у вигляді файлового дескриптора відповідного сокета. Тепер кожний байт даних записаний в цей файловий дескриптор буде відправлятись по мережі отримувачу. Розглянемо, що відбувається з цими даними в мережній підсистемі. Спочатку з даними має справу драйвер мережного протоколу. Він формує з даних пакети, визначає через який мережний інтерфейс ці пакети мають відправлятись, дописує до пакетів необхідні заголовки, після чого передає

пакети на обробку відповідному мережному інтерфейсу (точніше, ставить пакет в чергу відправки, пов'язану з цим інтерфейсом).

Далі, в залежності від типу мережного інтерфейсу, над пакетами можуть виконуватись різні дії. Наприклад, у випадку інтерфейсу lo (loopback) пакет даних забирається з черги на відправку і відразу розміщується в черзі отриманих пакетів, звідки він потрапить до драйвера протокола. У випадку інтерфейса eth0 до пакета буде дописано заголовок Ethernet і він буде переданий драйверу мережного адаптера, який проінструктує мережний адаптер, звідки взяти і як відправити цей пакет. Ще один показаний на схемі варіант – це обробка пакету мережним інтерфейсом PPP (Point-To-Point). Пакет розміщується в черзі відправки інтерфейса ppp0, звідки його забере демон (фонова служба в ОС Linux) pppd. Демон допише до пакету необхідні заголовки і через спеціальний символьний файл-пристрою dev/ttyS0 передасть пакет драйверу COM-порта, який забезпечить фізичну відправку пакета по послідовному порту. Відповідно, при отриманні даних пакети проходять розглянутий шлях у зворотньому порядку.

Тема 8. Надійність та безпека в ОС

8.1. Основні проблеми надійності та безпеки в МОС

Надійність ОС:

1) середній час роботи без збоїв,

2) частота виникнення помилок та збоїв у роботі,

3) наявність втрат важливих даних внаслідок збоїв,

4) швидкість відновлення роботи після збою,

5) стійкість системи до великих навантажень (обчислювальних та комунікаційних).

Базовим механізмом забезпечення надійності в сучасних ОС є захист (protection) компонентів системи від несанкціонованого (unauthorized) доступу.

Приклади:

1) режим ядра vs. режим користувача (права доступу до апаратних засобів),

2) механізм віртуальної пам’яті (заборона доступу одного процеса до пам’яті іншого),

3) система прав доступу до програм та даних користувачів в багатокористувацьких системах та ін.

В механізмах захисту використовується поняття області захисту (protection domain), яка визначає до яких ресурсів має доступ даний процес або користувач. Можливість виконати деяку операцію з ресуром називається право доступу (access right). Відтак область захисту можна представити як набір прав доступу: {<object-name, rights-set>}. Реалізація механізму захисту –> Основні завдання:

1. Визначення прав доступу та областей захисту (статичних, динамічних).
2. Забезпечення переключення областей захисту (domain switching).
3. Реалізація матриці прав доступу (Access Matrix).
4. Контроль доступу (Access Control).
5. Відкликання прав доступу (Revocation of Access Rights).

Основні проблеми безпеки з точки зору захисту від зовнішніх втручань:

1. Порушення конфіденційність (Breach of confidentiality).
2. Порушення цілісності даних (Breach of integrity).
3. Знищення або обмеження доступу до даних (Breach of availability).
4. Несанкціоноване використання ресурсів та сервісів (Theft of service).
5. Відмова в обслуговуванні (Denial of service).

Основні завдання забезпечення безпеки:

1. Організація захисту від програмних загроз (Trojan horse, Trap Door, Logic Bomb, Stack and Buffer Overflow, Viruses, Rootkits).
2. Організація захисту від мережних атак (Worms, Port Scanning, Distributed denial-of-service, Firewalling).
3. Захист даних та каналів зв'язку за допомогою шифрування (Cryptography).
4. Забезпечення аутентифікації користувача (User Authentication).
5. Виявлення вторгнень (Intrusion detection).
6. Запобігання вторгненням (Intrusion Prevention).
7. Забезпечення проактивного захисту (Honeypot).

Зв'язок між надійністю та безпекою: для зловмисного втручання в роботу ОС, як правило, використовуються помилки у її коді. Приклад: помилки переповнення буфера даних. Причини ненадійності сучасних ОС:

1. Великий «розмір» та складність системи.
2. Слабка «ізоляція» помилок.

За різними оцінками на 1000 рядків коду припадає від 2 до 75 помилок (bugs) в залежності від загального розміру програми. Приблизна оцінка: 2,5 млн. рядків коду ядра ОС х 6 помилок на кожні 1000 рядків = 15 тис. помилок. Драйвери

можуть складати до 70% коду ОС. Для драйверів показник кількості помилок на 1000 рядків коду вищий в 2-3 рази.

Compartmentalization (компартменталізація) від «compartment» (відсік) –> поділ інформації (структур даних) між модулями програми в такий спосіб, щоб вони могли доступитися тільки до того, що потрібно їм для роботи. В сучасних ОС відсутня ізоляція між окремими компонентами: ядро ОС це одна велика програма, яка складається з процедур/функцій, які мають повний доступ до усіх структур даних ядра.

Чотири підходи до забезпечення надійності та безпеки ОС [Таненбаум] (у порядку зростання «радикальності» підходу):
1. Використання проміжного рівня захисту від помилок у драйверах пристроїв.
2. Використання технології апаратної віртуалізації.
3. Використання мікроядерної архітектури ядра.
4. Використання безпечних мов програмування та елементів програмної віртуалізації.

8.2. Використання проміжного рівня захисту від помилок у драйверах пристроїв

Підхід застосовується до «звичайних» ОС з монолітним ядром (єдиний адресний простір, режим ядра (kernel mode)). Принцип: кожний драйвер ізолюється від ядра за допомогою спеціальної програмної «обгортки». Ізоляція: пошкодження драйвером вмістимого структур даних ядра виключається за допомогою механізму трансляції сторінок віртуальної пам’яті (virtual memory page map) –> коли запускається драйвер усі «зовнішні» по відношенню до драйвера сторінки адресного простору ядра переводяться в режим «тільки на читання» (read-only), в режими «читання та запис» драйвер може доступитись

тільки до «власної» пам’яті. Кожна спроба драйвера модифікувати структури даних ядра перехоплюється менеджером ізоляції (isolation manager).

Для кожного драйвера визачається 1) набір його функцій, які може викликати ядро; 2) набір функцій ядра, які потрібні драйверу для роботи (наприклад, виділення пам’яті для буфера даних). Для кожної з цих функцій створюється «обгортка», яка виконує «посередницьку» функцію: виклик функції йде спочатку до «обгортки» (wrapper), в якій перевіряються параметри на відповідність заданим обмеженням та виявляються спроби «атак». Якщо драйвер робить спробу модифікувати об’єкт даних ядра, то створюєтья копія цього об’єкта і розміщується у «власній» пам’яті драйвера. Після модифікації об’єкта він перевіряється «обгорткою» і в разі успішної перевірки модифікований об’єкт копіюється назад у пам’ять ядра. В такий спосіб помилка або відмова драйвера не впливає на пошкодження об’єктів даних ядра. «Обсяг робіт»: в сучасних ОС потрібно реалізувати до тисячі різних функцій-«обгорток» (це можна робити інкрементно).

Відновлення після збою: 1) звільнення ресурсів та перезапуск драйвера; 2) використання «тіньового» драйвера, який після перезапуску основного використовується для відновлення його поточного стану перед збоєм.

Обмеження: 1) не контролюються ситуації виконання драйверами окремих помилкових інструкцій в привілейованому режимі; 2) не виключається помилковий запис драйвером в некоректний порт вводу/виводу; 3) відсутній захист від переходу драйвера у нескінченний цикл.

8.3. Використання технології апаратної віртуалізації

Принцип (гаряче резервування): на найнижчому рівні замість ОС запускається спеціальна системна програма - монітор віртуальних машин (virtual machine

monitor), який «вміє» створювати віртуальні примірники реальної машини (на кожній з яких запускається відповідний примірник ОС). Базовий варіант: факт одночасного запуску декількох примірників машина-ОС (віртуальних машин, ВМ) приховується від користувача.

Відновлення після збою забезпечуєтья шляхом заміни «пошкодженого» примірника машина-ОС резервним примірником, в якому відновлюється поточний стан «пошкодженого» примірника перед збоєм. Додатковий спосіб захисту: різні віртуальні машини виконують різні «роботи», наприклад, на одній ВМ виконуються користувацькі процеси, а на іншій — драйвери пристроїв. Також можна розподілити драйвери між декількома ВМ. Якщо виникає збій у драйвері, то він «зачіпає» лише відповідну ВМ, тоді як інщі продовжують працювати.

Реалізація:
1) монітор віртуальних машин (МВМ);
2) проміжне програмне забезпечення, яке реалізує операції вводу/виводу драйверів за допомогою відповідних функцій МВМ;
3) проміжне програмне забезпечення, яке забезпечує зв’язок між драйверами та користувацькими процесами, що виконуються на різних ВМ.

Основний недолік цього підходу - це збільшення обчислювального навантаження і, відповідно, можливе зменшення швидкості роботи прикладного ПЗ.

8.4. Використання мікроядерної архітектури ядра

Принцип: код ОС розбивається на незалежні ізольовані один від одного модулі («сервери» та драйвери), які виконуються в режимі користувача, + мікроядро ОС (мінімальний набір базових функцій). В ОС MINIX 3 мікроядро виконує

обробку переривань, базове управління процесами, планування паралельного виконання (scheduling) та підтримку між-процесної взаємодії (IPC). Над мікроядром розташовується рівень драйверів, кожний з який виконується як окремий процес в режимі користувача. Драйвери не можуть 1) виконувати привілейовні машинні інструкції; 2) виконувати операції вводу/виводу напряму (тільки за посередництвом мікроядра). Над рівнем драйверів розташовується рівень «серверів», які виконують основні функції ОС (=аналоги модулів монолітного ядра). Приклади: управління процесами, управління файлами, мережна підсистема і т.д.

Серед «серверів» є спеціальний «сервер реінкарнації» (reincarnation server), який відповідає за запуск або перезапуск усіх інших «серверів» та драйверів (наприклад, після аварійної зупинки). Всі «сервери» та драйвери взаємодіють між собою за допомогою механізму між-процесної взаємодії (IPC) (прямий обмін повідомленнями по готовності передавача та отримувача з можливістю асинхронної нотифікації).

Надійність:

1) Значно менша кількість помилок за рахунок невеликого «розміру» мікроядра. Мікроядро MINIX 3 містить біля 4000 рядків коду х 6 помилок на кожні 1000 рядків = 24 помилки (в порівнянні з 15 тис. для монолітного ядра).

2) З'являється можливість верифікації коду мікроядра за допомогою одної з формальних технік.

3) Драйвери виконуються у режимі користувача як окремі процеси, які можна перезапустити в разі збою (без зупинки роботи всієї системи).

4) «Чужий» код драйверів відсутній у мікроядрі.

5) Присутня можливість захисту на основі встановлення прав доступу та виконання для кожного драйвера (в ОС з монолітним ядром драйвер має повні права).

6) Механізм IPC в MINIX 3 не потребує буферизації повідомлень (прямий обмін по готовності), що виключає відповідні помилки та ресурсні обмеження.
7) Присутня можливість захисту на основі обмеження процесів у використанні різних функцій IPC.

Відновлення після збою: ізольованість «серверів» та драйверів дозволяє виявляти помилки та збої у їх роботі (можливо «зловмисні») та реагувати відповідним чином (наприклад, перезапуск «сервером реінкарнації») –> забезпечеується властивість само-відновлення (self-healing property).

Основний недолік цього підходу — це зменшення продуктивності роботи внаслідок застосування мікроядерної архітектури (розвиток яких також стримують економічні фактори).

8.5. Використання безпечних мов програмування та елементів програмної віртуалізації

Принцип: вся система та прикладні програми створюються на безпечній мові програмування (safe language), проходять подвійну верифікацію (на рівні текста програми та специфікації між-програмної взаємодії + на рівні байткоду) та запускаються на виконання у окремих ізольованих примірниках середовища виконання (run-time environment).

Один з найбільш відомих прикладів цього підоходу: експериментальна ОС Singularity (Microsoft). Вся ОС Singularity написана на безпечній мові програмування Sing# (розширений діалект C#) (окрім невеликої апаратно-залежної частини мікроядра).

Мова Sing# реалізує ідеї контрактного програмування, в рамках якого програміст визначає формальні точні специфікації інтерфейсів для компонентів

системи, що дає можливість виконувати верифікацію її роботи. Специфікації інтерфейсів, які називаються контрактами (contracts), визначають сематнику роботи примітивів обміну повідомленнями (message passing primitives) за допомогою, так званих, передумов, постумов та інваріантів. Всі процеси в ОС Singularity виконуються в одному віртуальному адресному просторі (така можливість з'являється за рахунок реалізованої безпеки коду), внаслідок чого є непотрібним переключення контексту. Також в ОС Singularity весь верифікований код виконується в «режимі ядра» (0-ве кільце безпеки в x86), що значно прискорює роботу всієї системи.

Мікроядро ОС Singularity виконує базові функції ядра ОС (управління програмними потоками, виділення пам'яті, між-процесна взаємодія за допомогою каналів, управління вводом/виводом та доступом до апаратури). Кожна програма (в тому числі системні програми та драйвери) виконується у власному примірнику середовища виконання у вигляді "software-isolated processes" (SIPs), який повністю ізольований від інших примірників.

Взаємодія програм здійснюється двома основними способами:
1) Канали (channels) –> двонаправлений канал зв'язку, взаємодія по якому специфікується відповідним контрактом.
2) Область обміну даними (Exchange Heap) для великих за обсягом даних, які не вигідно передавати повідомленнями через канали –> область пам'яті, в якій кожний об'єкт даних належить лише одному процесу + процеси можуть передавати один одному «право власності» на дані через канал.

Приклад контракту:

```
contract C1 {
        in message Request(int x) requires x > 0;
        out message Reply(int y);
```

```
        out message Error();

        state Start:
                Request? -> Pending;
        state Pending: one {
                Reply! -> Start;
                Error! -> Stopped;}
        state Stopped: ;
}
```

Канал приймає повідомлення трьох типів: Request, Reply, Error. Протокол взаємоії по каналу задається схемою переходів відповідного автомата.

Надійність:

1) За рахунок ізоляції збій у роботі будь якої окремої компоненти системи (прикладної програми, системної служби, драйвера) не призводить до зупинки всієї системи. Відповідна компонента може бути додатково перевірена та перезапущена на виконання.

2) Кожна програма перед запуском проходить верифікацію на сумісність з вже інстальованими програмами (на основі аналізу специфікацій між-програмної взаємодії) та правильність роботи байт-коду.

Тема 9. Розподілені системи

9.1. Організація роботи розподілених систем

Розподілена система – це набір взаємодіючих незалежних обчислювальних машин які розглядаються користувачем як єдина система.

У схемі взаємодії клієнта та сервера в розподіленій системі (рис.1): $t_2 >> t_1$, де t_1 – затримка в середині машини-клієнта, t_2 - затримка обміну інформацією між клієнтом та сервером (як елементами розподіленої системи).

Основні проблеми: неможливо в підсистемі В дізнатися про зміну в підсистемі А в той самий момент, а лише через час t_2.

Основні аспекти організації роботи розподілених систем:

1. зв'язок – інформаційна взаємодія, обмін даними, обмін повідомленнями та виклик функцій.
2. організація – диспетчеризація обчислювальних процесів і розподіл обчислювального навантаження, розподіл мережних ресурсів.
3. іменування створення та підтримка єдиного простору імен.
4. синхронізація – забезпечення єдиного системного часу (фізичного та логічного) служби точного часу.
5. Реплікація та несуперечливість – підтримка достовірності (цілісності спільних даних).
6. надійність – стійкість для помилок опрацювання відмов та виключних ситуацій.
7. захист – гарантування безпеки при передачі даних і забезпечення санкціонованого доступу до програм і даних.

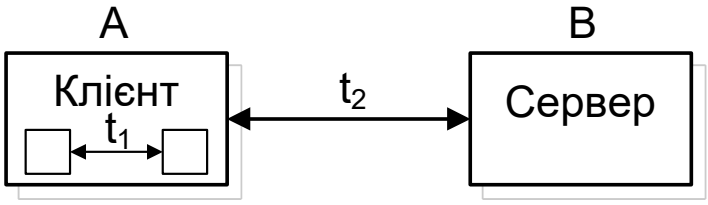


Рис.1. Схема взаємодії клієнта та сервера в розподіленій системі

9.2. Основні задачі СПЗ розподілених систем:

1. Поєднання процесів користувачів з ресурсами (власне виконання тої задачі для якої створена система).

   Віддалені ресурси – це обчислювальні машини інших користувачів пристрої збереження даних і інформаційні ресурси і так далі.

2. Прозорість – це спрощення механізмів взаємодії користувацьких процесів за рахунок приховування деталей їх функціонування. Від користувацьких процесів приховується той факт що всі обчислювальні процеси та ресурси в системі фізично розподілені між машинами в системі.

   Види прозорості:

   - прозорість доступу (представлення даних);
   - прозорість місця розташування;
   - прозорість реплікації (приховує той факт що працюємо з копією, а не самим ресурсом);

3. відкритість – надання послуг, сервісів, служб звертання до яких вимагає стандартного синтаксису та семантики. Стандартизовані інтерфейси взаємодії на всіх рівнях моделі OSI.
4. масштабованість – здатність до розширення розподіленої системи:
   a. масштабованість по розміру – легкість підключення нових користувачів та ресурсів;
   b. географічна масштабованість рознесення користувацьких процесів в просторі;
   c. адміністративна масштабованість – це легкість управління та збору інформації про розподілену систему;

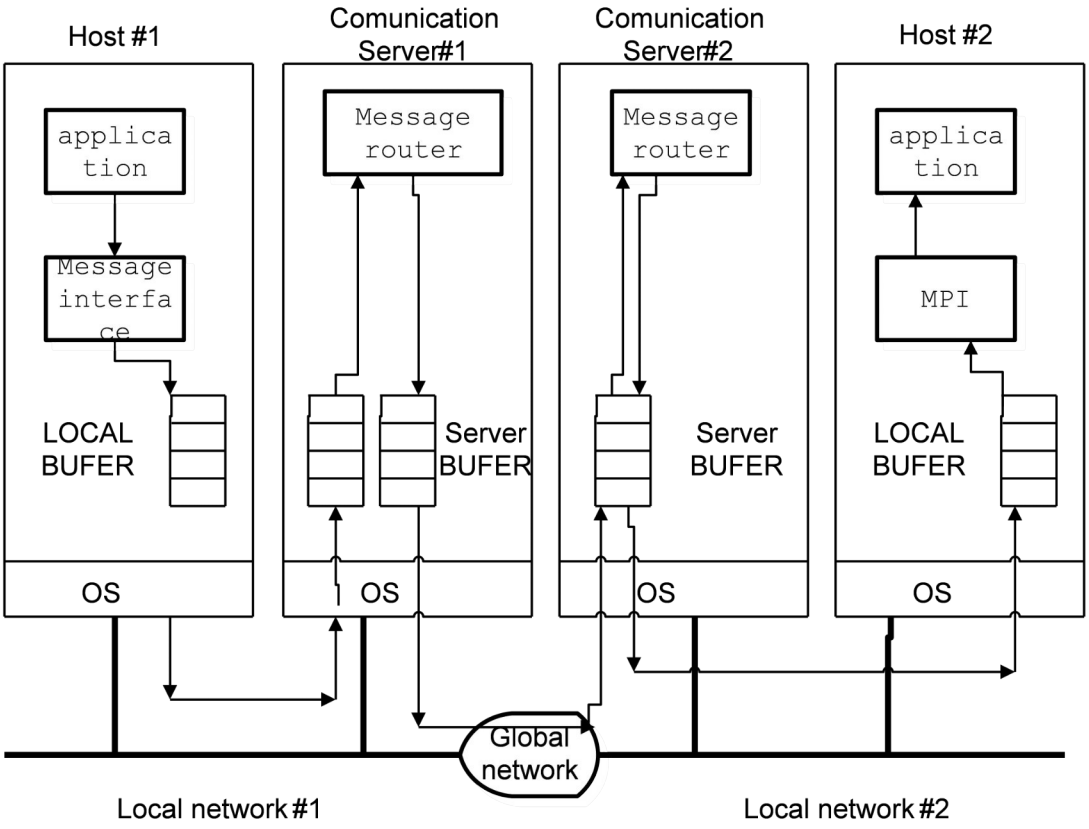
9.3. Обмін повідомленнями в розподілених системах

Зв'язок є ключовим питанням в розподілі системи

Message interface – створює свої структури даних які не залежить від application.

Message interface – повідомлення поставляється в buffer звідки в ОС.

Message router – завдання – якомога оптимальніше передати повідомлення.



Узагальнена схема система обміну повідомленнями

Класифікація способу обміну інформацією

I. За часом існування повідомлення в системі:

- стійкий зв'язок – повідомлення, що передається, зберігається в системі обміну доти поки воно не надійде утримувачу.
- нерезидентний зв'язок – повідомлення зберігається в системі доти, доки виконуються процеси, які відправляють та отримують ці повідомлення.

UDP протокол – є втіленням не резидентного зв’язку.

II. За способом організації відправки/прийому повідомлень:

- Асинхронний зв’язок – відправник продовжує роботу відразу після відправки
- Синхронний зв’язок - відправник блокується до того моменту, поки його повідомлення не дійде одержувачу.

T,S – де T=0 стійкий, T=1 нерезидентний, S=0 асинхронний, S=1 синхронний.

0,0 Стійкий асинхронний зв’язок (системи електронної пошти).

0,1 Стійкий синхронний зв’язок (вимагає більше ресурсів).

1,0 Нерезидентний асинхронний зв’язок (UDP, RPC).

1,1 Нерезидентний синхронний зв’язок з синхронізацією по отриманню.

1,1 Нерезидентний синхронний зв’язок з синхронізацією по обробці.

1,1 Нерезидентний синхронний зв’язок з синхронізацією по відповіді.

9.4. Стійкий асинхронний зв’язок в розподілених системах

- стійкий зв'язок – повідомлення, що передається, зберігається в системі обміну доти поки воно не надійде утримувачу.
- асинхронний зв’язок – відправник продовжує роботу відразу після відправки

Приклад: системи електронної пошти.

9.5. Стійкий синхронний зв’язок в розподілених системах

Стійкий зв'язок – повідомлення, що передається, зберігається в системі обміну доти поки воно не надійде отримувачу.

Синхронний зв’язок - відправник блокується до того моменту, поки його повідомлення не дійде одержувачу. СС зв'язок потребує найбільше ресурсів (зав. файлів на віддалений комп’ютер)

9.6. Нерезидентний асинхронний зв’язок в розподілених системах

Нерезидентний зв'язок – повідомлення зберігається в системі доти, доки виконуються процеси, які відправляють та отримують ці повідомлення.

Асинхронний зв’язок – відправник продовжує роботу відразу після відправки.НА звязок(UDP, RPC)

9.7. Нерезидентний синхронний зв’язок в розподілених системах (з синхронізацією по отримуванню повідомлення)

Нерезидентний зв'язок – повідомлення зберігається в системі доти, доки виконуються процеси, які відправляють та отримують ці повідомлення.

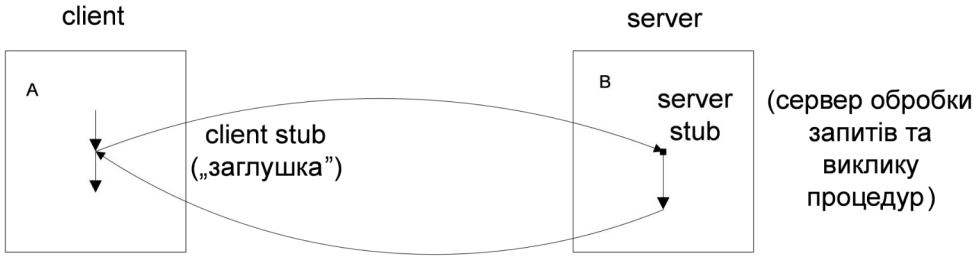
Синхронний зв’язок - відправник блокується до того моменту, поки його повідомлення не дійде одержувачу.

Нерезидентний синхронний зв’язок з синхронізацією по:

- отриманню.
- обробці.
- відповіді (коли відправник повідомлення блокується до отримання квитанції).

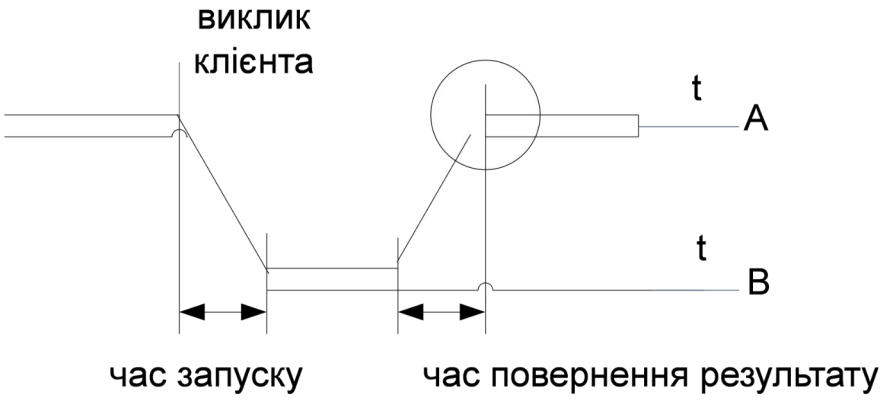
Тема 10. Віддалений виклик процедур (RPC) та віддалений виклик методів (RMI)

10.1. Віддалений виклик процедур (RPC – Remote Procedure Call)



Причини необхідності використання RPC:

1. Відсутність потрібної функціональності.
2. Оптимізація обчислюваного процесу.
3. Оптимізація комунікаційного навантаження.



Послідовність дій, що реалізує RPC:

1. Програма клієнта викликає клієнтську заглушку.
2. Клієнтська заглушка створює повідомлення та викликає локальну ОС (назва процедури, що пересилати? куди? які параметри?).
3. ОС клієнта відсилає повідомлення віддаленій ОС.
4. Віддалена ОС передає повідомлення так званій серверній заглушці.
5. Серверна заглушка розпаковує повідомлення і викликає процедуру на сервері.
6. Сервер виконує процедуру і повертає результат серверній заглушці.
7. Серверна заглушка запаковує результат повідомлення і викликає свою локальну ОС.
8. ОС сервера відсилає повідомлення ОС клієнта.

9. ОС клієнта отримує повідомлення і передає його клієнтській заглушці.
10. Клієнтська заглушка розпаковує результат повідомлення і пересилає їх програмі.

Класичним прикладом є SunRPC I DCE RPC.

10.2. Віддалений виклик методів (RMI)

Цей механізм з’явився разом з ООП.

На відмінну від RPC замість client stub було створено proxy, а замість server stub створено skeleton.

Використовується у DCOM, java RMI, CORBA.

Тема 11. Реплікація та несуперечливість в розподілених системах

11.1. Поняття реплікації та несуперечливості

Поняття реплікації та несуперечливості включає в себе кешування.

Репліка (replica) – це копія деяких даних. Як правило кількість копій (реплік) необмежена.

Реплікація – це створення потрібної кількості реплік з метою підвищення продуктивності та надійності роботи системи.

Надійність: в разі виникнення помилки, помилкову копію можна замінити на вірну.

Продуктивність: для великої кількості процесів, що звертаються до одних і тих же даних за рахунок створення персональних копій різко зменшується час доступу. Копія даних розміщується поблизу процесу, що їх використовує.

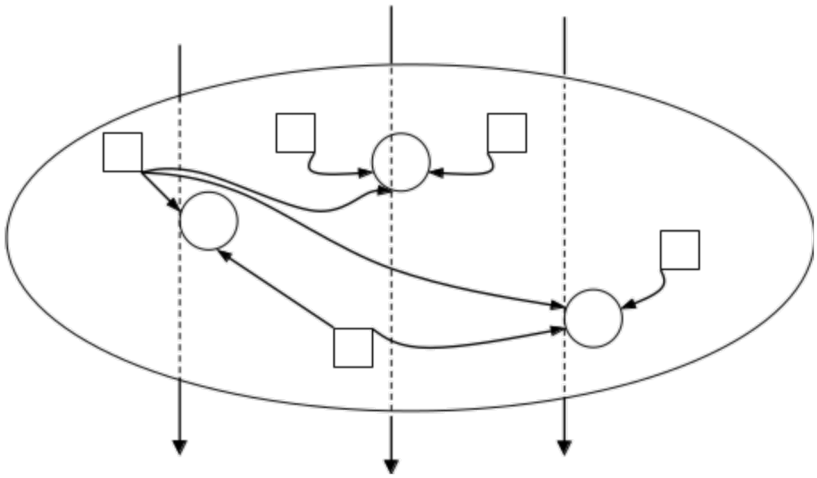


Рис.11.1. Схема розподіленої системи з серверами та клієнтами

Ціна, яку потрібно заплатити за підвищення надійності і продуктивності це розв’язання проблеми несуперечливості всіх реплік (копій даних).

Несуперечливість – це цілісність даних.

Кожний раз при змінах копії даних вона (копія) починає відрізнятися від усіх інших. Для забезпечення несуперечливості ці зміни треба перенести у всі інші копії. Як і коли треба переносити ці зміни визначає ціну реплікації.

Протиріччя: Реплікація підвищує продуктивність, а забезпечення несуперечливості зменшує.

Основна вимога до реалізації несуперечливості: операція читання повинна давати однаковий результат для кожної реплікації в часі та «просторі» розподіленої системи.

Ключова ідея підтримки несуперечливості: оновлення всіх реплік відбувається як атомарна операція.

11.2. Моделі несуперечливості, орієнтовані на дані

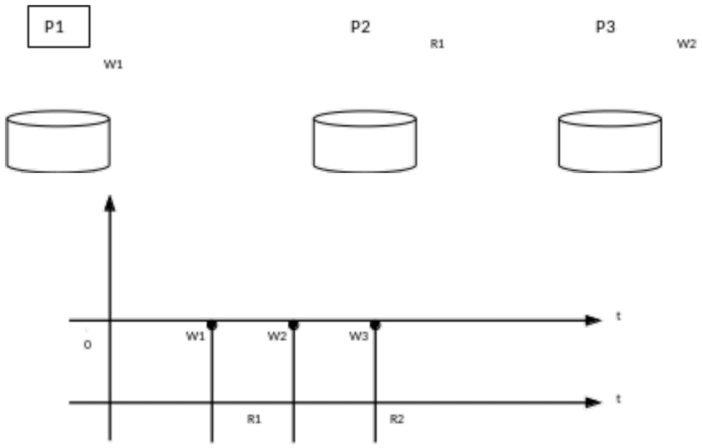


Рис.11.2. Схема роботи розпоідленого сховища даних (Distributed data store)

Модель несуперечливості (consistency model) – це домовленість між процесами та банком даних, яка гарантує несуперечливість даних. Класифікація МН:

1. Жорстка несуперечливість (strict). Абсолютна впорядкованість у часі всіх звертань до пам’яті.
2. Лінеаризація. Усі процеси спостерігають (бачать) всі звертання до банку даних в одному і тому самому порядку і ці звертання впорядковані згідно логічного часу.
3. Послідовна модель. Аналогічно до 2, але звертання не впорядковані у логічному часі.
4. FIFO. Всі процеси бачать операції запису усіх інших процесів у порядку їх виконання. При цьому різні процеси можуть бачити різні послідовності.
5. Причинно-наслідкова.Усі процеси бачать усі звертання пов’язані причинно-наслідковими зв’язками в одному і тому самому порядку.

Жорстка синхронізація найточніша.

Лінеаризація менш жорстка.

Найбільш широко використовується послідовна модель.

FIFO і причинно-наслідкова – ослаблені моделі в яких відсутній глобальний контроль, в якій послідовності які операції виконувати.

11.3. Моделі несуперечливості, орієнтовані на клієнта

Це специфічний випадок, коли система нечутлива до відносно високого ступеня порушення несуперечливості.

Наприклад: корпоративні СУБД, WWW.

Зазвичай в таких системах досить довго не відбувається зміна даних і всі репліки поступово стають несуперечливими. Ця модель називається потенційною несуперечливістю.

Основні проблеми в цьому випадку виникають з мобільними клієнтами, які час від часу змінюють точку підключення.

Тема 12. Організація обчислень в розподілених системах

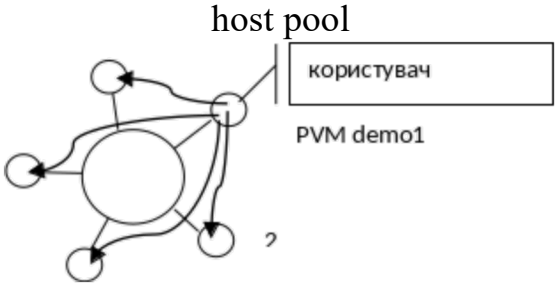
12.1. Способи організації обчислень в розподілених системах

1) модель клієнт-сервер
2) віддалені обчислення(аналог клієнт-серверної моделі з універсальним сервером)
3) GRID-computing

Однорідні(потужність ~ однакова)
Неоднорідні(різні потужності)

4) Модель віртуального обчислення
5) Cloud computing

12.2. Паралельна віртуальна машина (PVM)



12.3. Інтерфейс передачі повідомлень (MPI)

1 процес може бути одночасно у декількох групах.

Перша версія – 95
Остання версія MPI – 97р.

12.4. Концепція адаптивного паралелізму

Ідея: програмі передають повноваження

1) по створенню необхідної кількості паралельних процесів (з врахуванням ресурсів і задачі)
2) по вибору: де і коли запускати ці процеси

12.5. Концепція переносу коду

Концепція переносу коду – на противагу переносу даних, це переміщення виконавчого коду з 1 вузла на інший.

Принципи переносу коду:

1. Підвищення обчислювальної продуктивності.
2. Оптимізація мережного трафіка.
3. Забезпечення гнучкості.

В програмі можна розрізнити сегмент коду, сегмент ресурсів, сегмент виконання.

Моделі переносу коду:

- Модель слабкої мобільності (переноситься лише сегмент коду java applets)
- Модель сильної мобільності (одночасно переносяться сегмент коду та сегмент виконання)

1) Перенос коду ініційований відправником
SQL-запит
2) Перенос коду, ініційований отримувачем коду.

Тема 13. Синхронізація в розподілених системах

13.1. Синхронізація в розподілених системах: централізовані служби часу

Кожна машина має свій власний таймер, який час від часу синхронізується з показами глобального годинника. Існують міжнародні глобальні служби часу, які видають стандартний найбільш точний час.

GMT – Grinwich meridian time

UTC – Universal Coordinated Time

Є механізм корегування показів UTC до астрономічного часу.

International Atomac Time

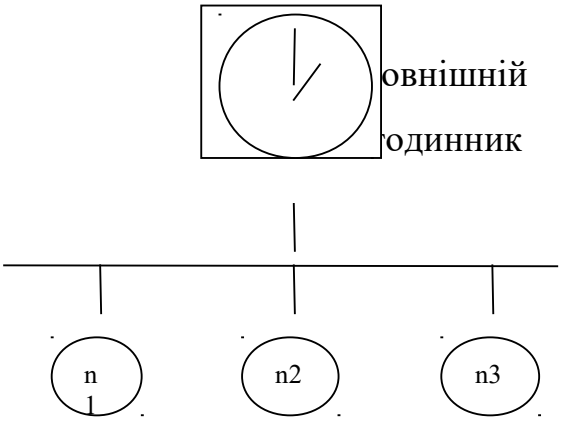


Рис.13.1. Узагальнена схама централізованої синхронізації

13.2. Синхронізація в розподілених системах: відмітки часу Лампорта

В логічному часі на відміну від фізичного важлива лише послідовність подій, а не точний час їх виникнення.

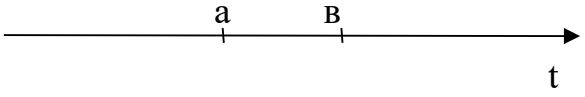


Рис.13.2. Події a та b в логічному часі

Предекат Лампорта ставить 1 подію перед іншою.

Всі процеси згідні з тим, що з початку відбулася подія а, а пізніше в. В розподіленій системі це означає:

1) а і в події, які відбулися в 1 процесі, то
а –>в істина
2) якщо а - це відправка повідомлення, в – отримання цього повідовлення, то
а –> в істина

Якщо для 2 подій не виконуються умови 1 і 2, то вони називаються паралельними.

Транзитивність:

а –>в в –>с, то а –>с .

Відмітки часу Лампорта – це цілі числа, які ставляться у відповідність події.

Алгоритм Лампорта (присвоєння відміток):

1. Будь яким двом подіям в 1 процесі призначаються відмітки, різняться хоча б на 1.
2. Кожне повідомлення супроводжується відміткою Лампорта відправника.
3. Якщо ця відмітка більша за показ логічного годинника отримувача, то він встановлюється в це значення.

Векторні відмітки часу додатково враховують причинно-наслідкові зв'язки.

13.3. Алгоритми голосування в розподілених системах.

В децентралізованих розподілених системах, в яких відсутній зовнішній годинник, виникає потреба у виборі головного годинника між рівними, цей вибір називається – голосування.

Необхідні умови для здійснення голосування:

1) кожний вузол знає кількість інших вузлів N;
2) кожний вузол має ідентифікатор (наприклад MAC-адресу);
3) всі вузли поєднанні між собою каналами зв'язку;

Приклади алгоритмів голосування в розподілених системах:

1) алгоритм забіяки;
2) алгоритм голосування на кільці;
3) рандомізований алгоритм голосування;

13.4. Голосування в розподілених системах: алгоритм забіяки.

Усі вузли мають ID і знають ID інших вузлів. Схема вузлів кожний з кожним.

Алгоритм:

1) Вузол, який претендує на лідерство розсилає усім іншим вузлам, з номерами більшими за його, повідомлення у голосуванні.
2) Якщо йому ніхто не відповів то він стає лідером і повідомляє про це усім іншим клієнтам. Інакше, якщо він отримав відповідь «ОК» то він не стає лідером.
3) Агент, який отримав повідомлення про голосування, відсилає агенту який прислав повідомлення, відповідь «ОК» і виконує п.п. 1,2

Характеристика: в такий спосіб буде вибрано вузол-лідер (вузол з найбільшим номером) за скінчений час.

13.5. Голосування в розподілених системах: алгоритм голосування на кільці.

Алгоритм:

1) Всі вузли мають ID і поєднанні в кільце з 2-ома сусідніми вузлами.
2) Якщо номер отриманий від свого сусіда більший за власний, тоді він пересилає його по кільцю. Якщо свій номер більший, тоді пересилає свій номер по кільцю (іншому з двох сусідніх вузлів).
3) Якщо вузол отримав свій ID то він лідер в даній групі вузлів і повідомляє інші вузли групи про те що він лідер.

13.6. Голосування в розподілених системах: рандомізований алгоритм голосування

Необхідні умови:

1) Вузли не мають ID.
2) Всі вузли взаємодіють по одно-направленій кільцевій схемі.

3) Кількість вузлів N відома усім вузлам.

Опис алгоритму:

1) Кожен вузол може знаходитись в станах S_0, S_R. Якщо вузол перейшов в стан S_0, тоді свій номер рівний 0. Якщо в стан S_R, тоді номер вибирається випадково в межах [1,N].

2) Алгоритм виконується циклами по N-тактів. В кожному такті вузол передає свій номер та ном. Отр. від сус. вузл, далі до іншого сусіднього вузла.

3) Після N-тактів роботи, кожен вузол отримав масив з N-номерів. Якщо в цьому масиві є один номер більший за всі інші номери, тоді вузол під цим номером стає лідером.

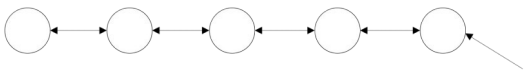
4) Якщо таких найбільших номерів є декілька, то всі вузли з цим номером залишаються в стані S_R, а інші вузли переходять в стан S_0. І цикл алгоритму продовжується для вузлів в стані S_R.

13.7. Децентралізована синхронізація в розподілених системах

Це варіант, коли зовнішній годинник відсутній і треба за скінченну кількість тактів встановити локальний таймер в одне локальне значення.

Як синхронізувати цілком однакові вузли, які не мають затримки:

Задача синхронізації ланцюжку стрільців(Джона Майхілла):



S_0 – початковий стан в якому знаходяться всі вузли; S – стан попередньої готовності.

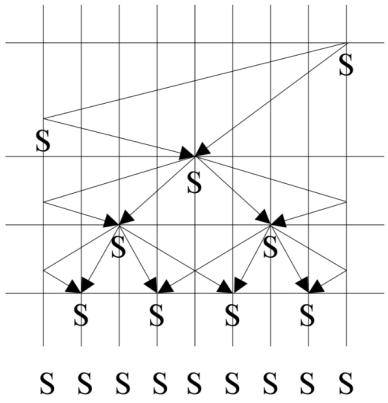
Агенти між собою обмінюються сигналами з різною «швидкістю»(з якою затр. вузол який передав сигнал передасть повідомлення іншому вузлу)

Крайній вузол відправляє в ланцюжок одночасно 2 сигнали ”a_1” та ”a_3” при цьому специфічно поводиться крайній вузол, він їх віддзеркалить (після цього крайній вузол при отриманні сигналу ”a” переходить в стан S_1).

Всі інші вузли виконують такий алгоритм:

- Якщо отриманий 1 сигнал, то передати його далі по ланцюжку;
- Якщо отримано 2-ва сигнали то віддзеркалити їх, і передати далі по ланцюжку, а також перейти в стан S;
- Якщо даний вузол і 2 його сусіди знаходяться в стані, S то зробити потрібну дію (зробити «постріл»).

Приклад процесу синхронізації для 9-ох вузлів:



Тема 14. Іменування ресурсів в розподілених системах

14.1. Іменування ресурсів в розподілених системах: розподілений простір імен

Розподілений простір імен(name spase) - принцип відокремлення адреси (точки доступу) від імені сутності(entity)

name resolution(резолюція імен) – процедура дозволяє по імені отримати точку доступу. Точку доступу можна змінити, а ім'я лишається тим самим(локальне незалежне ім'я).

Простір імен забезпечує:

1. Службу іменування (додавання, видалення, пошуку імен).
2. Службу резолюції імен (визначення адреси сутності виходячи з її імені).

Ці дві служби робить так зв. name server (сервер імен)

В розподілених системах внаслідок великої кількості сутностей реалізація простору імен розноситься по декількох серверах.

Як правило створюється ієрархія серверів імен, яку зручно поділити на 3-ри рівні:

а) global – глобальні

б) administrative – адміністративні

в) managerial – управлінські

Принцип ієрархії:

1. стабільність вмісту скеровуючи таблиць (directory tables) визначає рівень сервера імен.
2. Реалізація процесу резолюції імен (name resolution):
    a. Ітеративна резолюція;
    b. Рекурсивна резолюція.

14.2. Ітеративна резолюція імен

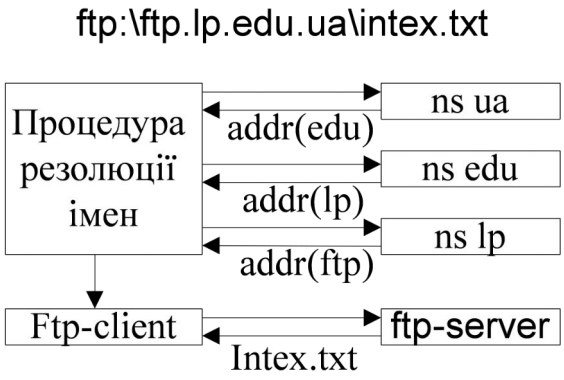


Рис. 14.1. Ітеративна резолюція імен, ns – name server (сервер імен)

Переваги:

- Висока надійність;
- Низьке навантаження на сервер імен.

Недоліки:

- Великі витрати на взаємодію клієнт-сервер

14.3. Рекурсивна резолюція імен

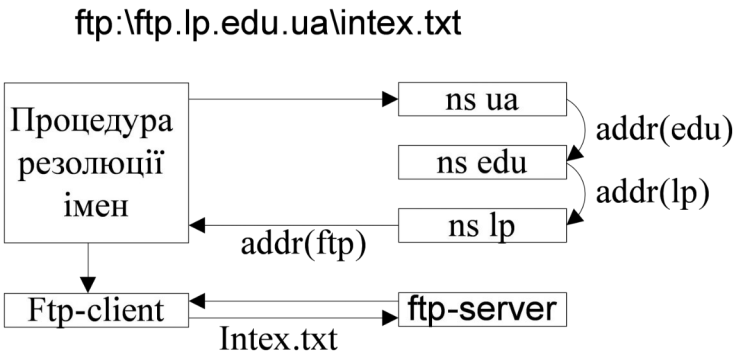


Рис. 14.2. Рекурсивна резолюція імен, ns – name server (сервер імен)

Переваги:

- Значно більш ефективне кешування результатів за рахунок того, що якась резолюція вже відбулася;
- Зниження витрат на взаємодію клієнта з серверами імен.

Недоліки:

- Велике навантаження на сервери імен (внаслідок цього на практиці на глобальному рівні застосовують тільки ітеративну резолюцію).

14.4. Простір імен DNS(Domain Name System)

DNS – найбільше централізована розподілена служба іменувань.

2 основних завдання:

- Пошук IP-адрес окремих хостів (вузлів);
- Пошук поштових серверів.

DNS-сервера підтримують розподілену базу відображень ім’я – адреса окрім таблиць відображення.

DNS сервер містить посилання на DNS сервери своїх відображень(під доменів)

Для обслуговування кореневого домена виділено декілька дублюючих один одного DNS серверів, адреса яких широко відома.

В DNS забезпечується повна незалежність іменування від розташування сутностей (хостів і т.д.).

14.5. Служби розподілених каталогів

Служба каталогів – це система організації великих даних про деякі сутності.

В розподіленому каталозі різні його частини зберігаються на різних серверах.

Приклад: служба каталогів X.500 (ISO стек протоколів OSI)

Ці служби розрізняють елемент каталогу:

Атрибути	Скорочення	Значення
A-attr	A	x, y, z
B-attr	B	b
C-attr	C	c

Множина всіх записів формує інформаційну базу каталогів. Унікальне ім’я кожного елементу формується як набір всіх його атрибутів та їх значення.

Основні операції: додавання/знищення елементів та пошук елементів.

В службі каталогів X.500 – реалізовані функції read (пошук по атрибутах), list (список синівських вузлів до даного вузла).

На основі перекриття атрибутів формують ієрархічний простір імен, який називається інформаційним деревом каталогів.

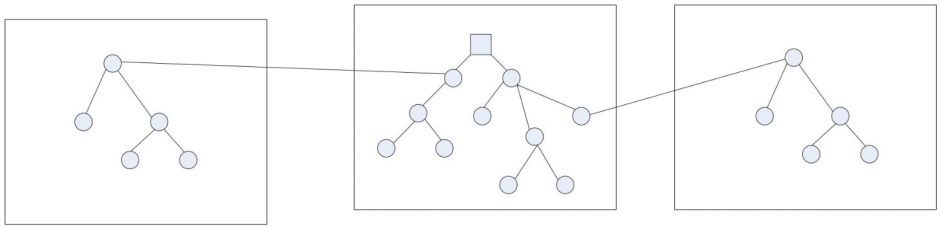


Рис. 14.3. Приклад роботи служби розподілених каталогів

14.6. Іменування та локалізація мобільних сутностей в розподілених системах

Мобільна сутність – це сутність, яка змінює час від часу свою точку доступу.

Як правило мобільна сутність належить управлінському рівню ієрархії серверів імен.

Вирішення задачі при зміні в мобільній сутності:

1. Змінити відображення пари ім’я – адреса.
2. До тієї пари, яка була додаємо нову пару (додавання символічного лінка).

Недоліки: просте переписання пари, зникає зв'язок в певний момент.

Використання проміжного рівня унікальних ідентифікаторів сутностей.

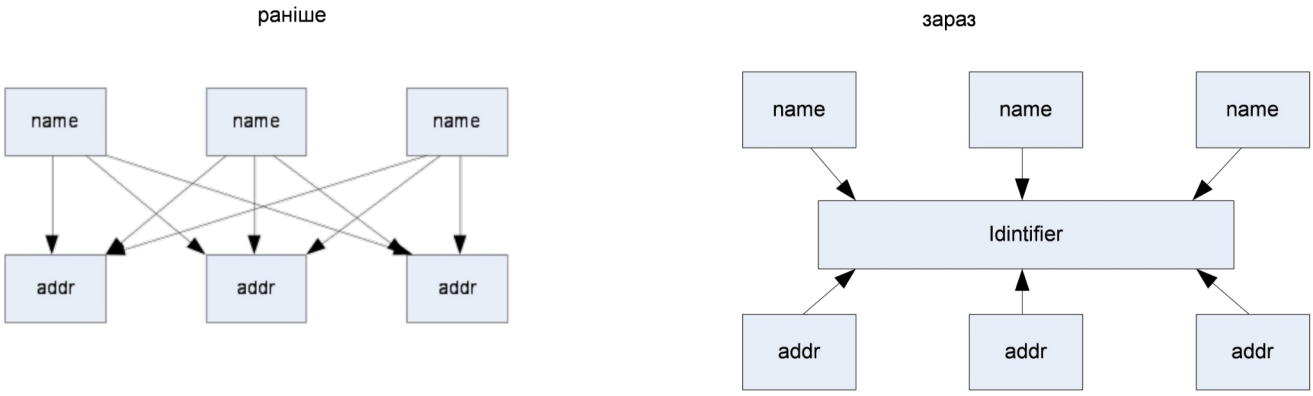


Рис. 14.4. Схеми іменування мобільних сутностей

Служба локалізації – визначає де розміщуються сутність з заданим ідентифікатором.

Прості рішення розсилки ідентифікаторі:

- широкомовна розсилка (в невеликих мережах)
- передача вказівників, це коли при переміщенні з А в В то в А залишається вказівник на В.

14.7. Локалізація мобільних сутностей на основі базової точки

Базова точка - фіксоване місце з якого відслідковуються переміщення мобільної сутності.

Як правило базова точка – місце створення або підключення мобільної сутності.

Схема мобільної IP адресації: В ролі ідентифікатора використовується фіксована адреса мобільного хоста. Точкою доступу виступає контрольна IP адреса.

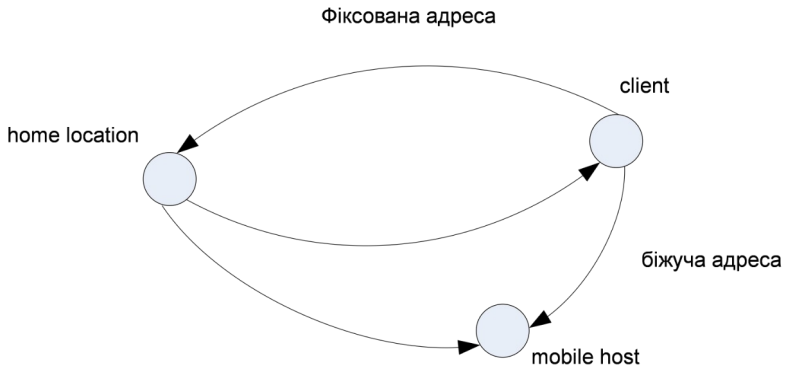


Рис. 14.5. Схема мобільної IP адресації

14.8. Ієрархічна служба локалізації мобільних сутностей в розподілених системах

Недолік: на основі БТ є зберігання інформації пари фіксована і біжуча адреса.

Замість одної базової точки використовується набір базових точок організованих в ієрархічну структуру по аналогії з доменною системою імен.

Основна ідея: Оптимізація пошуку за рахунок відображення локальності і мобільності сутностей в доменну структуру. Локалізація зводиться до пошуку по дереву.

Кешування вказівникі – багаторівневе кешування, яке дозволяє пришвидшити пошук мобільної сутності за рахунок використання “географічної” інформації.

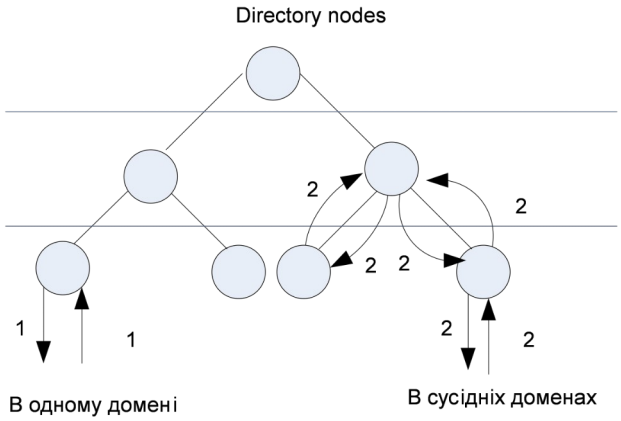


Рис. 14.6. Схема роботи ієрархічної служби локалізації мобільних сутностей

Список літератури

1. Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Operating System Concepts, 9th Edition, Wiley, 2012. – 944 p.
2. Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Operating System Concepts Essentials, 2nd Edition, Wiley, 2013. – 784 p.
3. William Stallings, Operating Systems: Internals and Design Principles, 8th Edition, Pearson, 2014. - 800 p.
4. Thomas Anderson, Michael Dahlin, Operating Systems: Principles and Practice, 2nd Edition, 2014. - 690 p.
5. Andrew S. Tanenbaum, Herbert Bos, Modern Operating Systems, 4th Edition, Pearson, 2014. - 1136 p.
6. Andrew S Tanenbaum, Albert S. Woodhull, Operating Systems Design and Implementation, 3rd Edition, Pearson, 2006. - 1080 p.
7. Andrew S. Tanenbaum, Jorrit N. Herder, Herbert Bos, Can We Make Operating Systems Reliable and Secure?, Computer, v.39., 2006. – pp.44-51.
8. Remzi Arpaci-Dusseau, Andrea Arpaci-Dusseau, Operating Systems: Three Easy Pieces [e-book], Arpaci-Dusseau Books, March, 2015 (Version 0.90)
9. Шеховцев В.А. Операційні системи. — К.: Видавнича група BHV, 2005.— 576 с.
10. Бондаренко М.Ф., Качко О.Г. Операційні системи : навч. посібник. — Х. : Компанія СМІТ, 2008. — 432 с.