Операційні системи

Лекція 14

Концепція розподіленого оброблення інформації

План лекції

- Обмін повідомленнями як єдиний спосіб керування розподіленими ресурсами
- Базові примітиви обміну повідомленнями

Особливості керування розподіленими ресурсами

- Найважливіша відмінність від централізованих систем – організація взаємодії між процесами
- Концептуально можливі лише два способи організації взаємодії між процесами:
  - За допомогою спільного використання одних і тих самих даних (спільна пам’ять)
  - За допомогою обміну повідомленнями
- У централізованих системах практично завжди застосовують спільну (поділювану) пам’ять
  - Семафор – окремий випадок спільної пам’яті (усі потоки звертаються до однієї змінної)
- У розподілених системах фізична спільна пам’ять відсутня
  - Тому єдина можливість – обмін повідомленнями
  - Повідомлення – це блок інформації, сформований процесом-відправником таким чином, щоби він був зрозумілий процесу-одержувачу

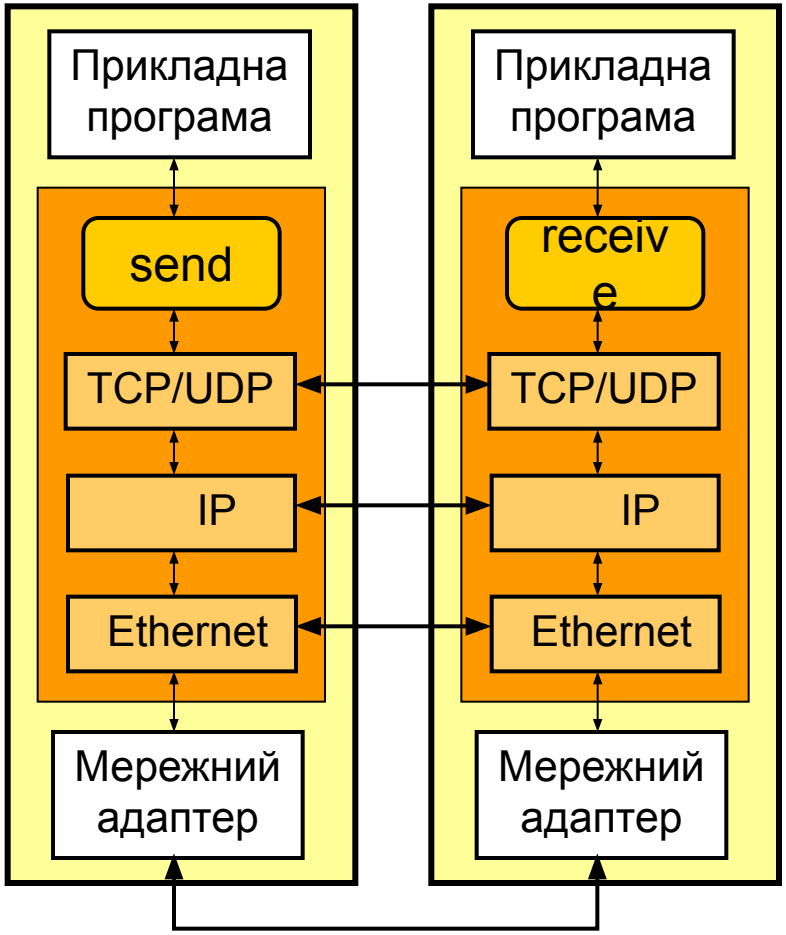
Формати повідомлень

- Повідомлення як правило має:
  - Заголовок
  - Тіло
- Заголовок часто (але не завжди) має фіксовану довжину і містить дані чітко визначених типів і чітко визначеного обсягу
- Заголовок як правило містить такі елементи:
  - Адреса – набір символів, що однозначно ідентифікує процес-відправник і процес-одержувач
  - Номер, який є ідентифікатором повідомлення
  - Ідентифікатор типу даних, що містяться у повідомленні
  - Поле, яке визначає обсяг даних, що містяться у повідомленні
  - Інформація, яка дозволяє перевірити цілісність даних (контрольна сума)
- Повідомлення може не мати форматного заголовка. Тоді воно містить структуровану інформацію, яка у загальному випадку складається з таких полів:
  - Тип даних
  - Довжина даних
  - Значення даних

Базові примітиви обміну повідомленнями

- Будь-яка мережна ОС має підсистему обміну повідомленнями, яку також називають транспортною підсистемою
  - Її завдання, як і інших підсистем ОС – приховати від прикладних програм деталі складних мережних протоколів і надати прикладним програмам абстракції у вигляді простих примітивів обміну повідомленнями
- У найпростішому випадку системні засоби обміну повідомленнями можуть бути зведені до двох базових примітивів:
  - send (відправити)
  - receive (отримати)
- На основі примітивів будують більш потужні засоби мережних комунікацій, наприклад, такі як:
  - Розподілена файлова система
  - Служба виклику віддалених процедур

Примітиви обміну повідомленнями і мережні протоколи



- Виконання примітивів send і receive спирається на виконання усіх протоколів стеку, які лежать нижче, відповідно до моделі взаємодії відкритих систем (OSI)
- Наприклад, для стеку TCP/IP
  - Примітив send звертається до засобів транспортного рівня (протокол TCP або UDP)
  - Той у свою чергу звертається до засобів мережного рівня (протокол IP)
  - Той звертається до засобів нижнього рівня (наприклад, драйвера адаптера Ethernet)
  - Драйвер керує апаратним засобом (мережним адаптером)
  - Примітив receive на іншому боці приймає повідомлення від засобів транспортного рівня після того, як воно було послідовно оброблено усіма нижчими рівнями у зворотному порядку

Варіанти реалізації базових примітивів

- Від реалізації базових примітивів залежить ефективність роботи мережі
- Основні питання, на які треба дати відповіді:
  - Як задають адресу одержувача?
  - Одержувач конкретного повідомлення завжди один, чи їх може бути кілька?
  - Чи необхідні гарантії доставки повідомлень?
  - Чи повинен відправник дочекатись відповіді на повідомлення перед тим, як продовжити роботу?
  - Як відправник, одержувач і комунікаційна підсистема повинні реагувати на відмови вузлів і каналів мережі?
  - Що робити, якщо приймач не готовий прийняти повідомлення? Чи відкинути його, чи зберегти у буфері?
  - Що робити, коли буфер переповнений?
  - Чи дозволено приймачу змінювати порядок оброблення повідомлень відповідно до їх важливості?
- Відповіді на ці запитання складають семантику конкретного протоколу передавання повідомлень

Синхронізація

- Спосіб синхронізації процесів у мережі цілком залежить від реалізації базових примітивів обміну повідомленнями, які бувають:
  - Блокуючі (синхронні)
    - Процес, що викликав примітив send, призупиняється до отримання з мережі підтвердження, що адресат отримав повідомлення
    - Процес, що викликав примітив receive, призупиняється до моменту отримання повідомлення
    - Якщо в процесі взаємодії процесів обидва примітиви є блокуючими, кажуть, процеси взаємодіють синхронно, в іншому випадку взаємодію вважають асинхронною
  - Неблокуючі (асинхронні)
    - Після виклику примітиву, керування повертається процесу миттєво
    - У процесі виклику ядру передається інформація про буфер, з якого треба взяти повідомлення для відправлення в мережу, або у який треба покласти повідомлення, що надійшло з мережі

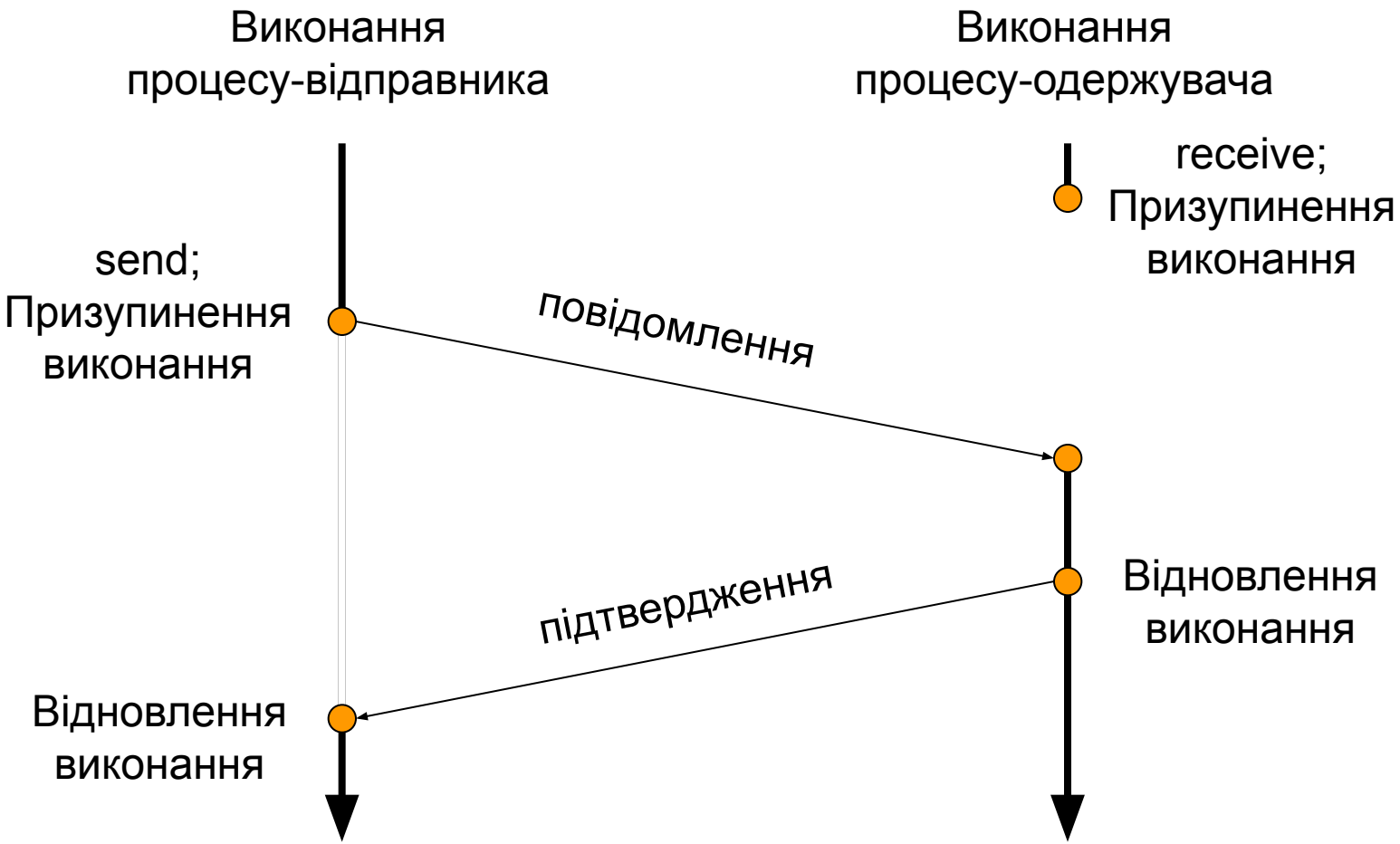
Тайм-аути у блокуючих примітивах

- Проблема блокуючих примітивів: процес може бути заблокованим назавжди
  - Це рівною мірою стосується і процесу, що здійснив виклик send, і процесу, що здійснив виклик receive
  - Блокування може статись, якщо
    - Повідомлення було втрачено в мережі внаслідок помилки
    - Інший процес-учасник обміну потерпів крах
- Для запобігання таким ситуаціям впроваджують механізм тайм-аутів
  - Задають інтервал часу, протягом якого триває очікування повідомлення (або підтвердження)
  - Якщо повідомлення (або підтвердження) не отримують протягом цього інтервалу, виклик завершується зі статусом “помилка”

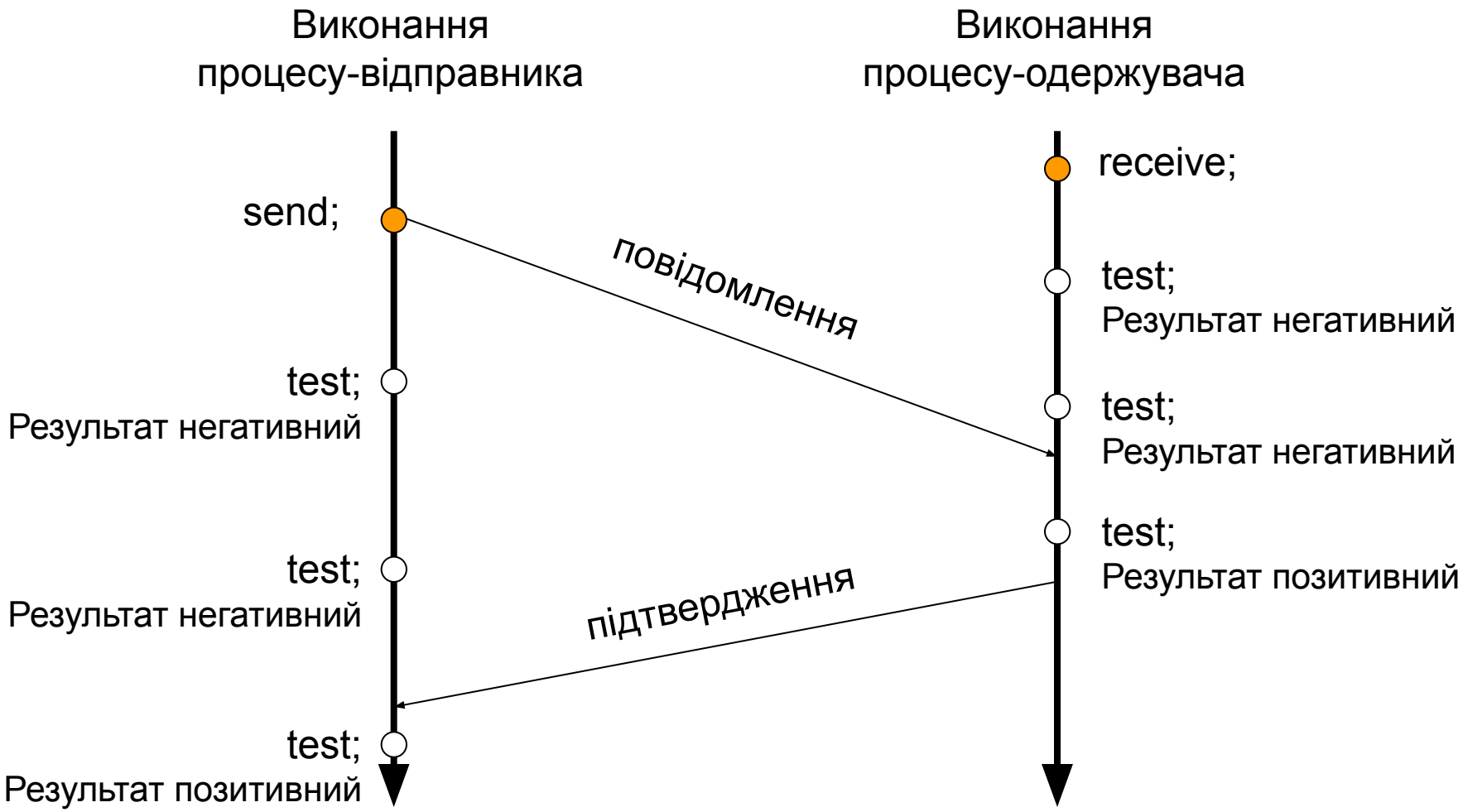
Особливості неблокуючих примітивів

- Великою перевагою неблокуючих примітивів є паралельне виконання процесу, що викликав примітив, і процедур, які цей примітив реалізують
  - Ці процедури не обов'язково працюють у контексті процесу, що їх викликав
- Проблемою реалізації неблокуючих примітивів є робота з буфером повідомлень
  - Наприклад, необхідно реалізувати повідомлення процесу-одержувача про те, що повідомлення надійшло і знаходиться у буфері
  - Застосовують один із двох способів:
    - Опитування (polling). Для цього вводять додатковий базовий примітив test (перевірити)
    - Переривання (interrupt). Застосовують програмне переривання процесу-отримувача повідомлення. Перевагою є підвищена ефективність, недоліком – ускладнене програмування

Синхронна взаємодія



Асинхронна взаємодія



Буферизація у примітивах обміну повідомленнями

- Примітиви бувають із застосуванням буферизації або без неї
  - (рос. – «буферизуемые» і «небуферизуемые»)
- Асинхронні примітиви завжди вимагають буферизацію
  - Для виключення втрат даних необхідним є буфер необмеженої довжини
  - Оскільки на практиці буфер завжди має обмежену довжину, можливі ситуації з переповненням буфера
    - Вихід – у керуванні потоком повідомлень (тобто, певне обмеження асинхронності)
- Синхронні примітиви можуть обходитись без буферизації взагалі
  - Повідомлення потрапляє у мережу безпосередньо з пам’яті процесу-відправника, а після одержання його з мережі – у пам’ять процесу-одержувача
  - На практиці для використання синхронних примітивів все ж передбачають буферизацію, обираючи буфер розміром в одне повідомлення

Примітив створення буфера

- Зазвичай ОС надає прикладним програмам спеціальний примітив для створення буферів create_buffer
- Процес повинен використовувати такий примітив перед відправленням і одержанням повідомлень
  - Тобто, перед send і receive
  - Часто буфер, що створений таким примітивом, називають порт (port) або поштова скринька (mailbox)
- Необхідно вирішити питання, що робити з одержаними повідомленнями, для яких буфер створено не було
  - Наприклад, якщо send на одному комп'ютері виконали раніше, ніж create_buffer на іншому
  - Один варіант – просто відмовитись від повідомлення
    - Перевага – простота
  - Інший варіант – помістити "неочікуване" повідомлення у спеціальний системний буфер на визначений час, очікуючи на виконання create_buffer
    - Недолік – проблема підтримання системного буфера

Надійні і ненадійні примітиви

- Повідомлення у мережі можуть втрачатись
- Три підходи до того, що з цим робити:
  1. Система не бере на себе зобов'язань щодо доставки повідомлень (дейтаграмний режим)
     - Реалізація надійної доставки – турбота прикладного програміста
  2. Ядро ОС одержувача повідомлень надсилає квитанцію-підтвердження на кожне повідомлення або групу повідомлень
     - У разі синхронних примітивів ОС розблоковує процес-відправник лише після одержання квитанції
     - Обробленням квитанцій займається підсистема обміну повідомленнями, прикладним процесам вони взагалі не видимі
  3. В якості підтвердження використовується відповідь
     - Застосовується у тих системах, де передбачається відповідь на кожний запит (характерно для архітектури клієнт-сервер)
- Надійність передавання повідомлень може також передбачати гарантії упорядкованості повідомлень

Способи адресації

- Адреси у вигляді констант
  - Можна застосовувати у дуже простій мережі
- Апаратні адреси мережних адаптерів
  - Адресують адаптер, але не процес
  - Важко знайти адресата у складній структурованій мережі
- Адресація машина-процес
  - Можна адресувати процеси за їх унікальними ідентифікаторами
  - Частіше адресують не процеси, а служби – за добре відомими номерами
  - Приклад – IP адреса і № порту
- Символьні адреси замість числових
  - Значно підвищують прозорість адресації
  - Приклад – система доменних імен
  - Перетворення символьних адрес на числові здійснюється
    - Або із застосуванням широкомовних запитів
    - Або із застосуванням централізованих служб

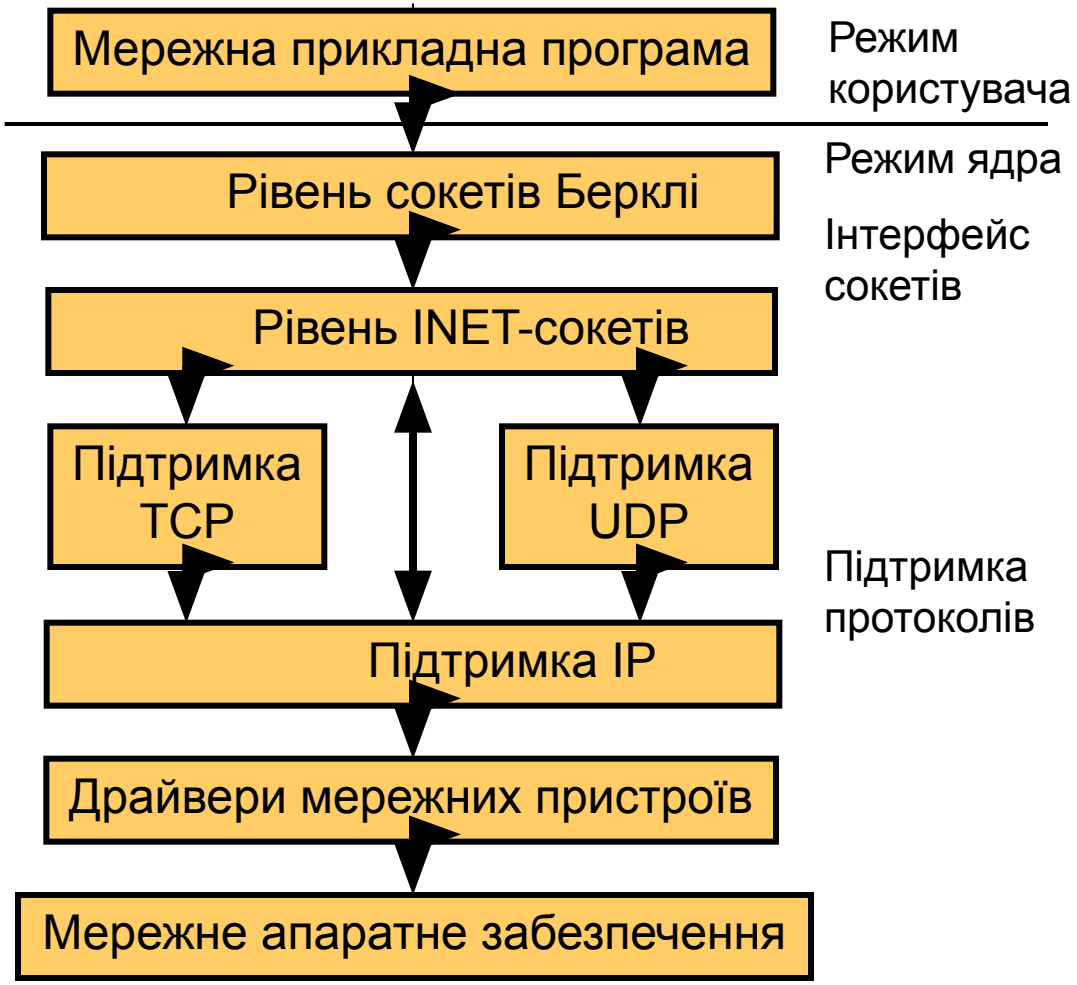
Механізм сокетів (sockets)

- Механізм сокетів є найпоширенішою системою обміну повідомленнями у мережі
  - Вперше був розроблений для версії 4.3 BSD UNIX (дотепер називають Berkeley Sockets – сокети Берклі)
  - Реалізацію для Windows називають Windows Sockets (WinSock)
- Програмний інтерфейс сокетів Берклі – це засіб зв'язку між прикладним рівнем мережної архітектури TCP/IP і транспортним рівнем
  - Фактично, це засіб зв'язку між кодом прикладної програми (що реалізує прикладний рівень) і реалізацією стека TCP/IP в ядрі ОС
  - Таким чином, інтерфейс сокетів – це набір системних викликів ОС

Основні концепції механізму сокетів

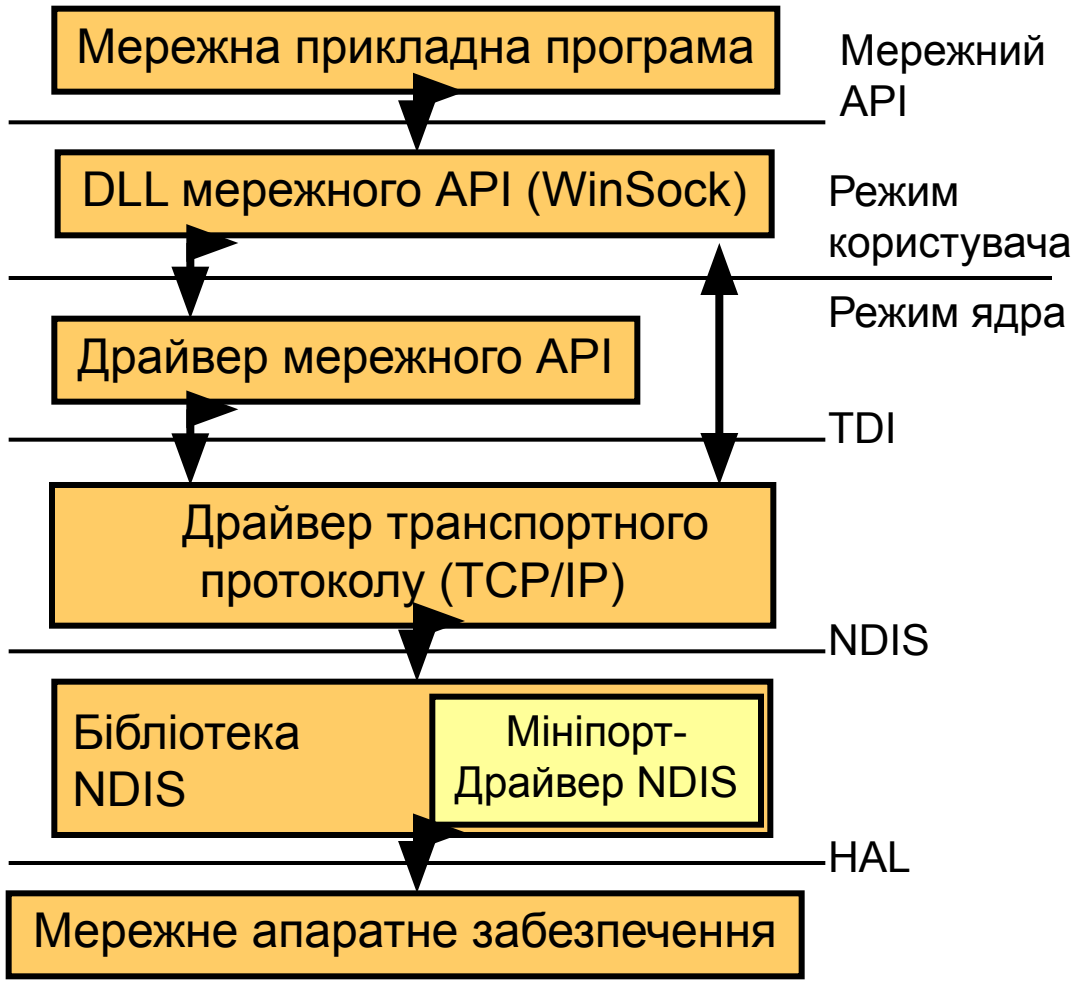
- Застосування абстрактної кінцевої точки з'єднання
  - Саме ця точка має назву *сокет* (*socket* – гніздо)
  - З'єднання між двома процесами здійснюється через пару сокетів
- Незалежність від мережних протоколів і технологій
  - Застосовується поняття *комунікаційний домен* (*communication domain*), який характеризується певним набором властивостей
  - Приклади – домен Інтернету (протоколи стеку TCP/IP), домен UNIX, або локальний домен (комп'ютер з його файловою системою)
- Сокет може мати як високорівневе символьне ім'я (адресу), так і низькорівневе
  - У домені Інтернету високорівневе ім'я – URL, низькорівневе – IP-адреса + порт
  - У домені UNIX ім'я сокету – це ім'я файлу
- Для кожного комунікаційного домену можуть існувати сокети різних типів
  - Наприклад, дейтаграмні (datagram) сокети і потокові (stream) сокети
  - Потокові з'єднання гарантують надійну упорядковану доставку

Архітектура мережної підтримки Linux



- Ядро розрізняє два рівня підтримки інтерфейсу сокетів
- На верхньому – інтерфейс сокетів Берклі, реалізований за допомогою універсальних *об'єктів сокетів Берклі*
  - Об'єкти реалізовані як файли, і містять:
    - тип сокета,
    - стан сокета,
    - універсальні операції сокетів,
    - покажчик на відповідний об' єкт INET-сокета
- На нижньому – реалізації інтерфейсу сокетів, що залежать від конкретної мережної архітектури
  - Наприклад, INET-сокети

Архітектура мережної підтримки Windows



- Драйвери транспортних протоколів надають драйверам мережних API універсальний інтерфейс транспортного драйвера (Transport Driver Interface, TDI)
- Мініпорт-драйвери NDIS відповідають за взаємодію драйверів транспортних протоколів і мережного апаратного забезпечення
  - Наприклад, драйвери конкретних мережних адаптерів
  - NDIS – Network Driver Interface Specification (специфікація інтерфейсу мережного драйвера)