# Основи операційної системи Linux

# Лекція 2. Основи операційної системи Linux

## План

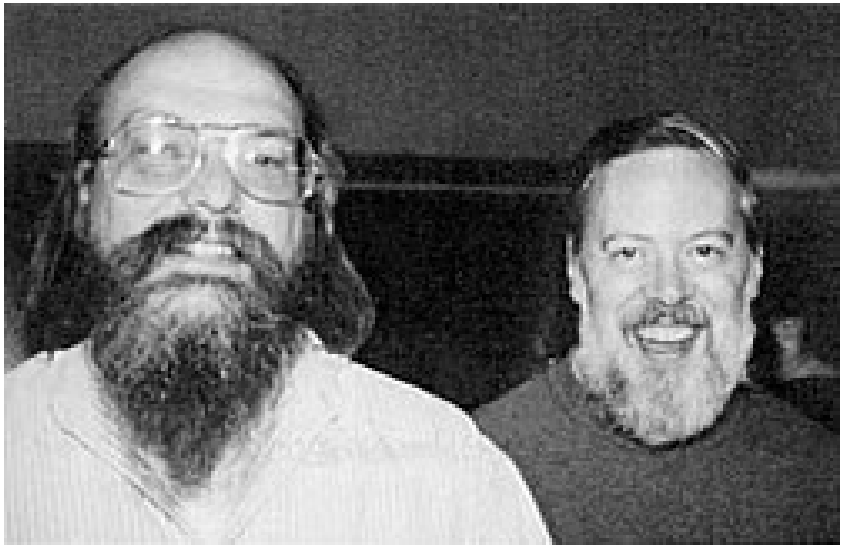
1. Історичний аспект
2. API та концепції Linux
3. Файлова система
4. Процеси
5. Користувачі та групи
6. Права доступу
7. Сигнали та взаємодія між процесами
8. Обробка помилок

# 1. Історичний аспект

UNIX

Операційна система **UNIX** була розроблена у **1969** році співробітниками **Bell Labs** (науково-дослідним підрозділом телекомунікаційної компанії **AT&T**) **Кеном Томпсоном** (Kenneth Lane Thompson) і **Деннісом Рітчі** (Dennis MacAlistair Ritchie). Початково вона розроблялася для комп'ютера **DEC PDP-7**.

DEC PDP-7



К. Томпсон і Д. Рітчі

# 1. Історичний аспект

Мова програмування **C** була розроблена в **1973** р. **Д. Рітчі,** як розвиток мови **B**, створеної раніше **К. Томпсоном**. Мова **C** спеціально створювалася для написання операційної системи **UNIX**, ядро якої було повністю переписано у 1973 році на **C**, що дало змогу переносити **UNIX** на інші комп'ютери.

Президент США Білл Клінтон вручає К. Томпсону і Д. Рітчі Національну медаль в галузі технологій за «**винахід операційної системи UNIX та мови програмування C**», 1999 рік

Компілятор мови C було вбудовано у UNIX

# 1. Історичний аспект

**Проєкт GNU** (GNU's Not Unix) був започаткований у 1983 р. **Річардом Столменом** (Richard Stallman) з метою створення вільної операційної системи, яку назвали **система GNU**.

У 1991 р. **Лінус Торвальдс** (Linus Torvalds) – фінський студент Університету Гельсінкі розпочав розробку іншого ядра на основі **MINIX** (спрощеної версії Unix, розробленої **Ендрю Таненбаумом** (Andrew Stuart Tanenbaum) для навчання).

GNU

MINIX

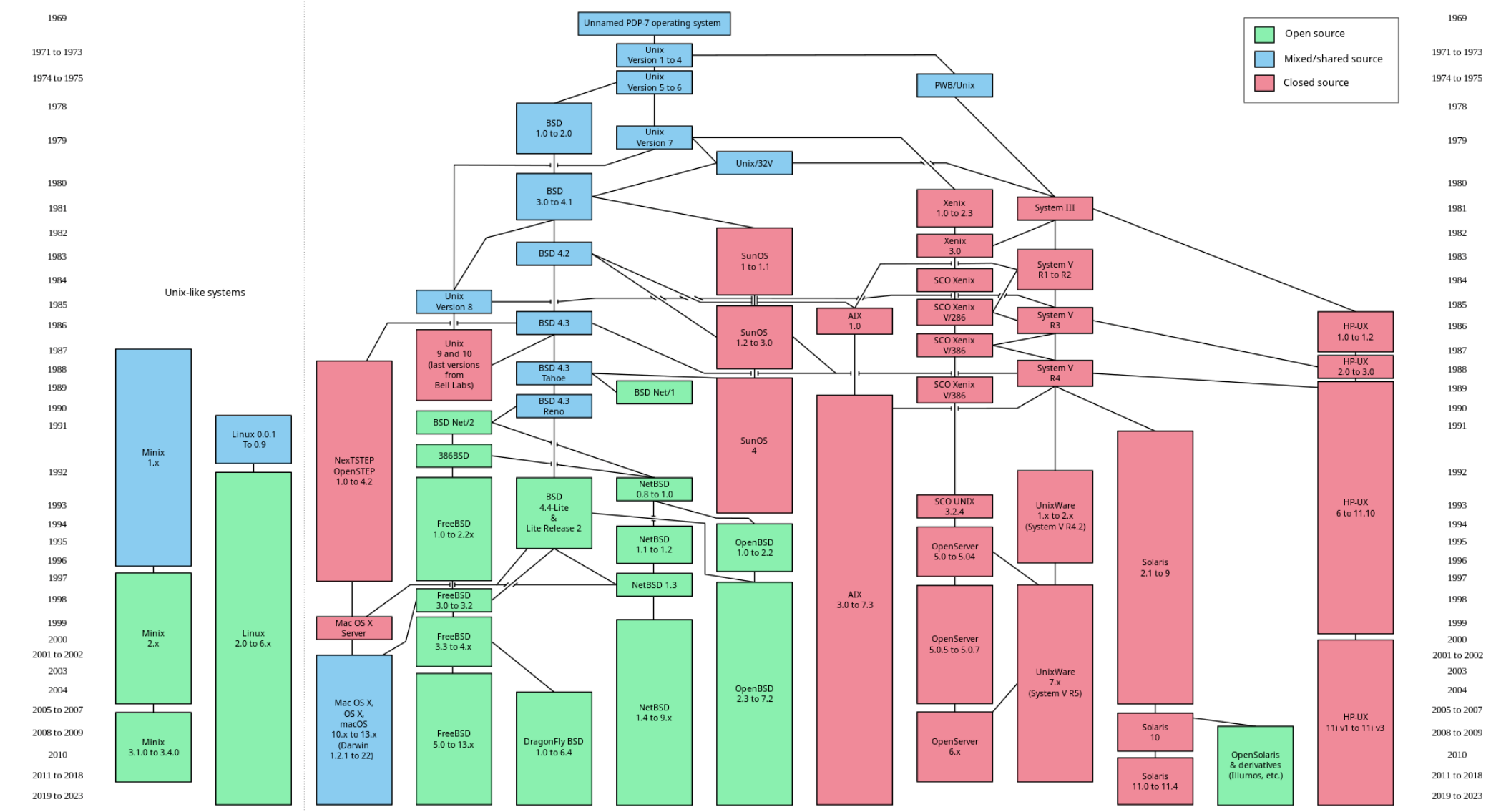
Linux

Р. Столмен

Е. Таненбаум

Л. Торвальдс

# 1. Історичний аспект




Генеалогічне дерево Unix-подібних операційних систем

# 2. API та концепції Linux

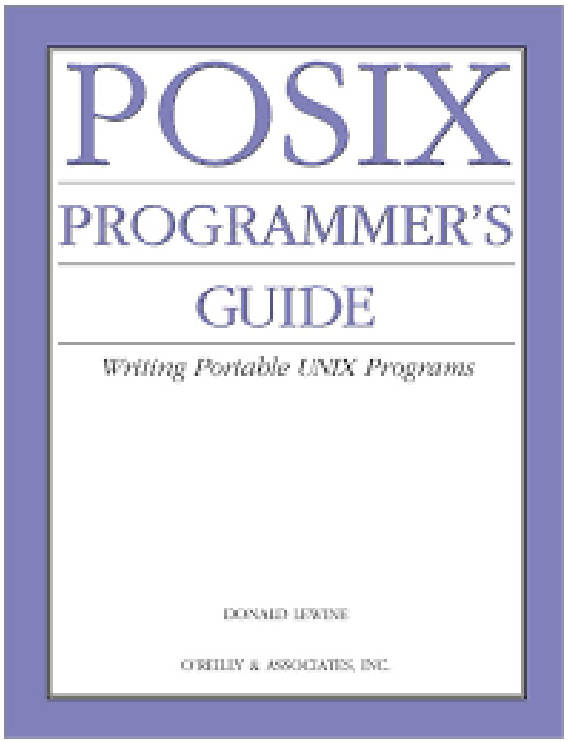
Для реалізації можливості перенесення програм між комп’ютерними платформами використовують спеціальні програмні інтерфейси, які називаються API.

**API** (Application Programming Interface) – це інтерфейс прикладного програмування, що реалізується у вигляді формального опису способів, за допомогою яких одна комп’ютерна програма може взаємодіяти з іншою. Цей опис містить набір констант, типів даних, функцій, класів тощо. Взаємодія з програмними або апаратними системами на практиці зазвичай реалізується за допомогою спеціального API (наприклад, Android API, Windows API, MS Office API і таке інше).

Розробники Unix-подібних систем дотримуються двох базових API:
- **POSIX** (Portable Operating System Interface);
- **SUS** (Single UNIX Specification).

# 2. API та концепції Linux

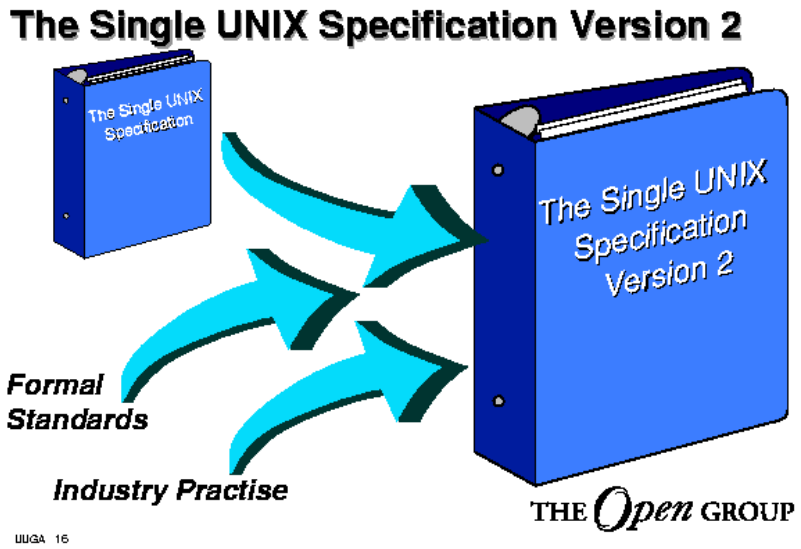
POSIX




**POSIX** (Portable Operating System Interface) – переносний інтерфейс операційних систем був розроблений у 1980-х роках **Інститутом інженерів з електротехніки та електроніки** (IEEE – Institute of Electrical and Electronics Engineers) для стандартизації інтерфейсу взаємодії з Unix-подібними операційними системами.

Остання на сьогодні версія була випущена у **2017** році. Вона має назву – **POSIX 2017**.

# 2. API та концепції Linux

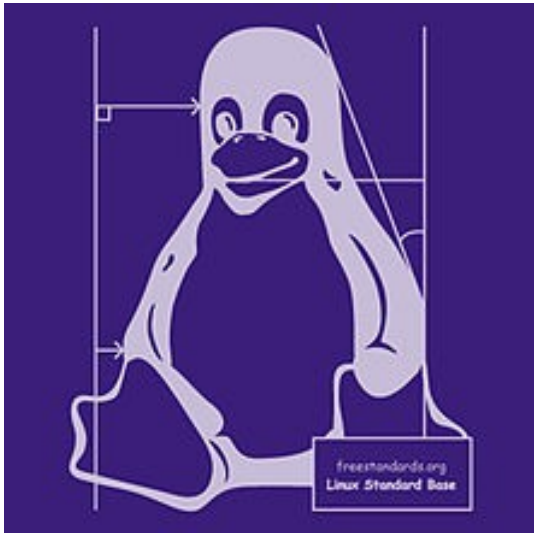



**SUS** (Single UNIX Specification) – альтернативний стандарт від **The Open Group** – промислового консорціуму, що сформувався внаслідок злиття **Open Software Foundation** (OSF) та X/Open.

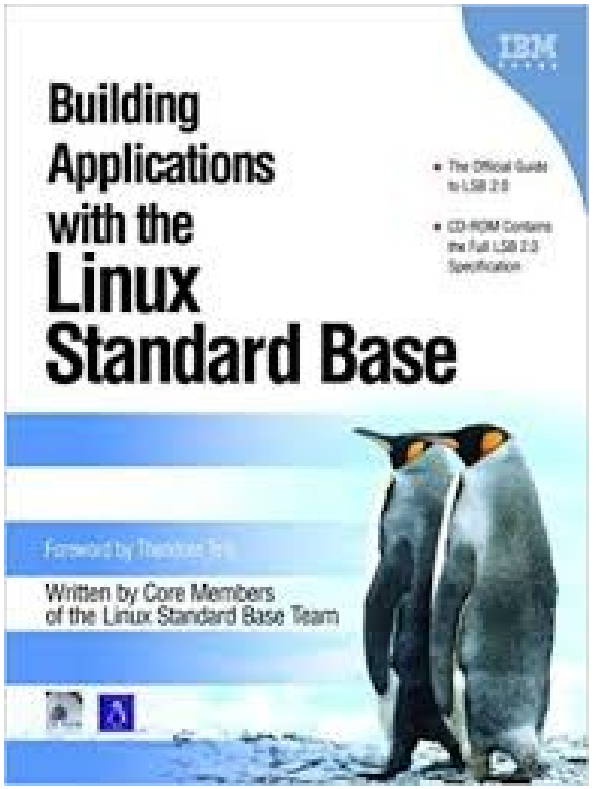
Останній варіант єдиної специфікації Unix – **SUSv4**, було опубліковано у **2008** році. В даний час SUS входить до складу POSIX. Фактично ці стандарти на сьогодні визначають системне програмування в UNIX-подібних ОС.

**ВАЖЛИВО! POSIX і SUS регламентують використання мови С для взаємодії з UNIX-подібними ОС.**

# 2. API та концепції Linux




Розробники Linux намагаються дотримуватися стандартів POSIX та SUS, однак вони створили свій власний стандарт **LSB** (Linux Standard Base), який доповнює та розширює POSIX та SUS.



# 2. API та концепції Linux

Операційна система Linux базується на використанні наступних концепцій:

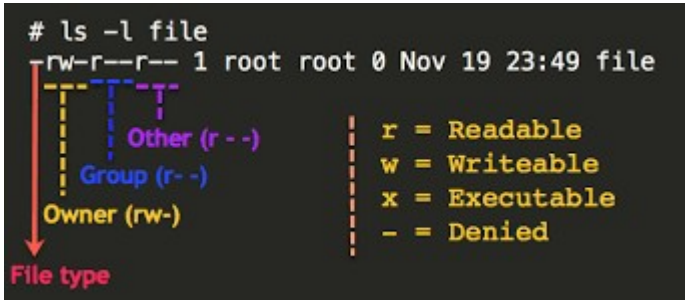
1) файли та файлова система;

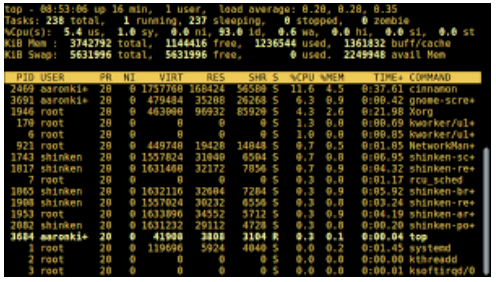
2) процеси;

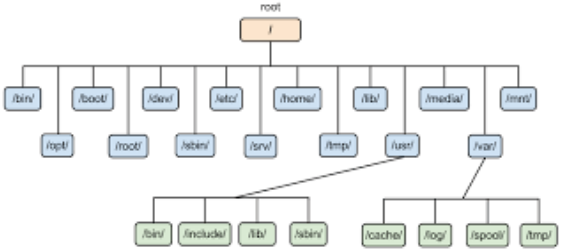
3) користувачі та групи;

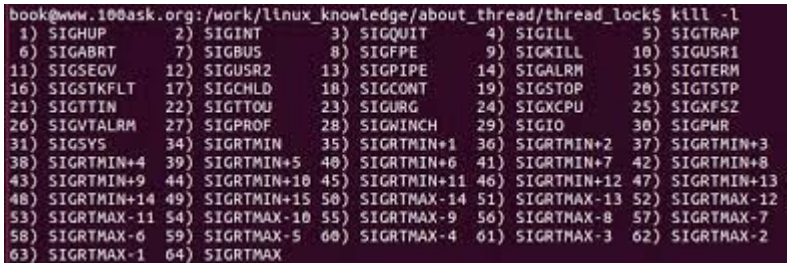
4) права доступу;

5) сигнали та взаємодія між процесами.



# 3. Файлова система

Всі Unix-подібні операційні системи підтримують єдиний набір абстракцій та інтерфейсів. Поняття файлу в них є базовою абстракцією. ОС Linux (як і Unix) дотримується концепції «**все є файлом**».

**Файл** – це іменована область даних на електронному носії інформації (у пам’яті комп’ютера, на жорсткому диску, CD, DVD тощо).

Взаємодія програм між собою або програм із апаратними пристроями у Linux здійснюється шляхом читання-запису даних у файлах. З будь-яким файлом можна виконувати низку стандартних дій:

- відкрити файл;
- виконувати читання-запис даних;
- здійснювати навігацію у файлі;
- закрити файл.

# 3. Файлова система

**Файл в Linux можна відкрити для:**
1) читання;
2) запису;
3) читання або запису одночасно.

Відкритому файлу ядро операційної системи присвоює унікальний **дескриптор** (описувач), який з'єднує файл із його **метаданими**. У Linux дескриптор – це ціле число, яке називається **файловим дескриптором** (file descriptor).

**Всі файли в Linux можна розділити на дві великі групи:**
- стандартні файли;
- спеціальні файли (файли символьних пристроїв, файли блокових пристроїв, іменовані канали та сокети).

# 3. Файлова система

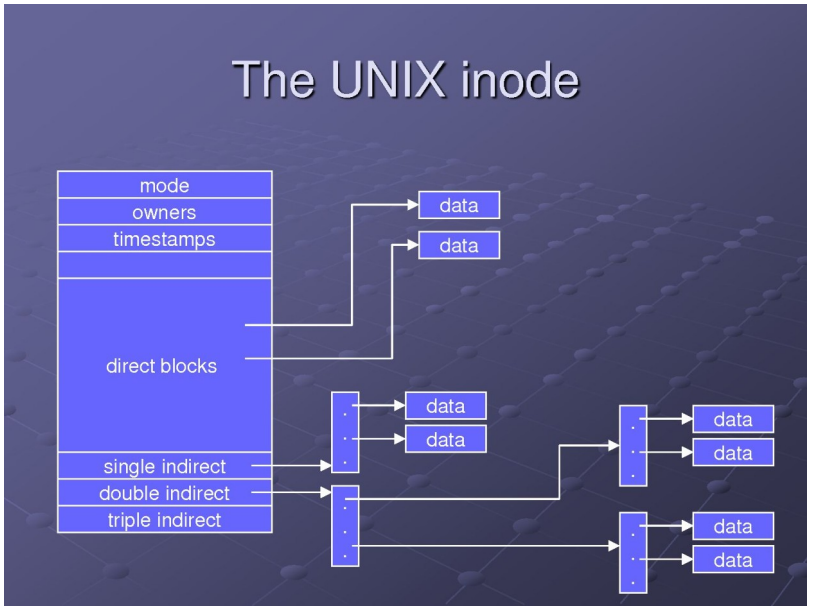
**Стандартні файли** в Linux часто називають **звичайними файлами** (regular file). Вони містять байти даних, організовані у вигляді лінійного масиву, який називається потоком байтів.

Читання/запис даних у файлі завжди починається з певного байта, який називається **файловою позицією** (зміщенням) у файлі. Зміщення є одним із головних елементів метаданих, який ядро асоціює з кожним відкритим файлом. Коли файл відкривається вперше, його файлова позиція дорівнює нулю. Файлову позицію можна встановлювати в задане значення (вона може бути за межами файлу, у цьому випадку проміжні байти будуть заповнені нулями).

При записі даних у середину файлу значення, які раніше знаходилося за цим зміщенням, замінюються на нове (розширити файл, записуючи інформацію в його середину, так просто не вийде).

Файл можна одночасно відкрити декілька разів як в одному й тому ж, так і у різних процесах. Ядро Linux не накладає жодних обмежень на паралельний доступ до файлів.

# 3. Файлова система




Доступ до файлів здійснюється за їх іменами, однак, безпосередній зв'язок файлу з його назвою в Linux відсутній: посилання на файл виконується за цілим **номером індексного дескриптора**, який є унікальним для файлової системи.

**Індексний дескриптор** (inode) – це об'єкт файлової системи (структура даних), що містить метадані, асоційовані з файлом, такі, наприклад, як час його останньої зміни, власник, тип і тому подібне (ім'я файлу там не зберігається!). Звичайні програми не можуть отримати доступ до файлу за його індексним дескриптором! Для цього використовуються каталоги.

**Каталог** – це спеціальний файл, що містить відображення імен файлів у номери індексних дескрипторів. Пара, що складається з імені та індексного дескриптора, називається **посиланням**.

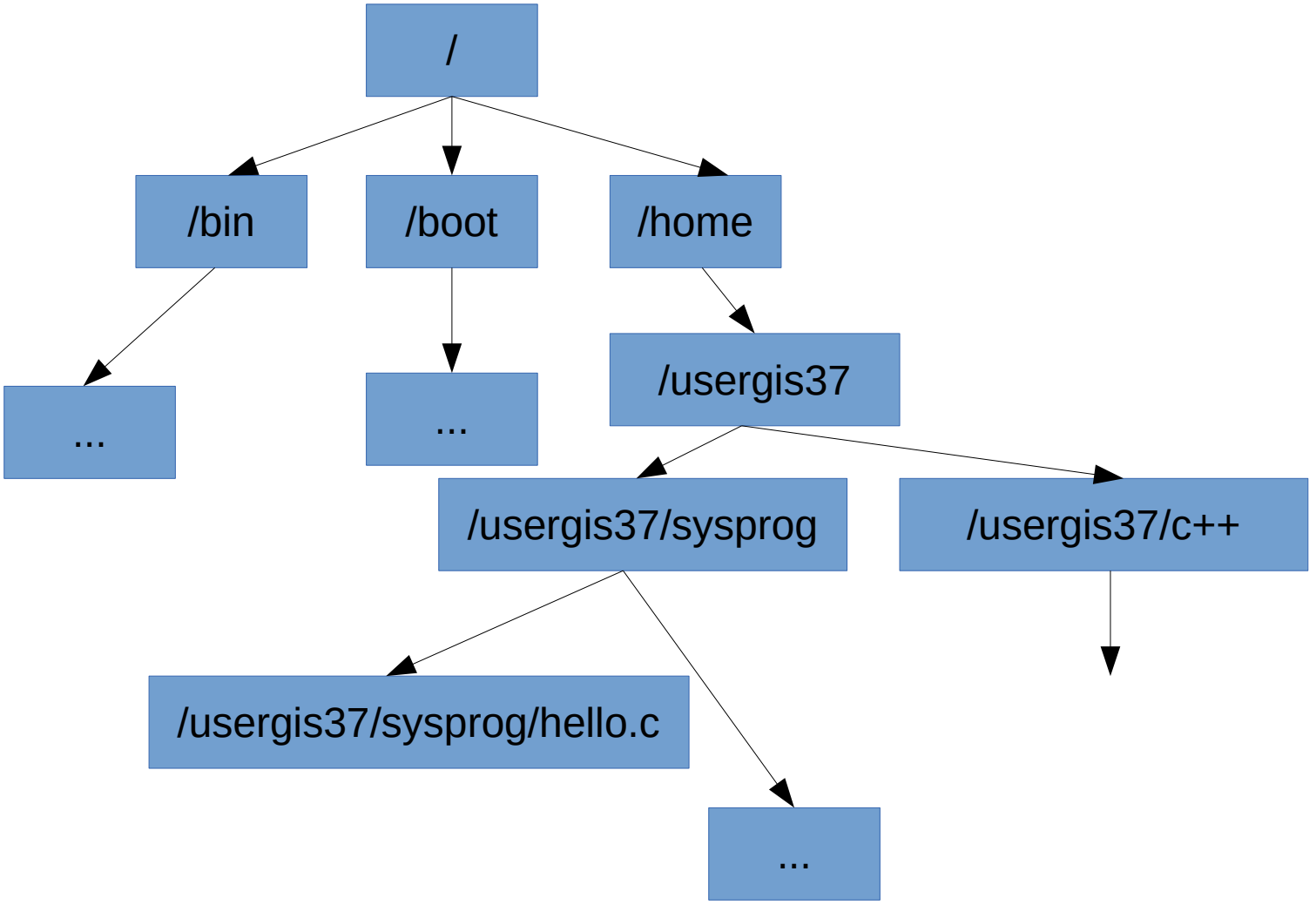
# 3. Файлова система

**Відкриття файлу за його ім’ям здійснюється таким чином:**
1) ядро відкриває каталог, який містить ім’я файлу;
2) у відкритому каталозі здійснюється пошук імені файлу;
3) за ім’ям файлу ядро отримує номер його індексного дескриптора;
4) за цим номером знаходиться сам індексний дескриптор.

В індексному дескрипторі міститься вся потрібна ядру інформація, зокрема, де безпосередньо на диску записаний файл (наприклад, у яких секторах HDD). Спочатку, при створенні файлової системи на диску є єдиний **кореневий каталог** «/». Каталоги, починаючи від кореневого, утворюють деревоподібну ієрархічну структуру.

Для однозначного пошуку файлу необхідно вказати повний шлях до нього. Наприклад, ім’я /home/user37/stud/6.0803/hello.c задає файл hello.c, який розташований у каталозі 6.0803, який у свою чергу лежить у stud і т. д. Ядро, відкриваючи такий файл, шукає в кореневому каталозі індексний дескриптор каталогу home, потім у home шукає user37 і т. д., поки не знайде дескриптор файлу hello.c.

# 3. Файлова система



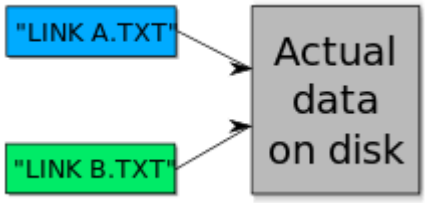

Деревоподібна ієрархічна структура каталогів та файлів

# 3. Файлова система

На фізичному рівні каталог – це звичайний файл, але ядро Linux не дозволяє програмам, які працюють на рівні користувача (тобто не мають спеціальних привілеїв ядра), отримувати повний доступ до цих файлів, так як будь-яка помилка (або зловмисна дія) може призвести до краху всієї файлової системи.

У Linux допускається відображення множини імен на один і той самий індексний дескриптор. І тут говорять про наявність так званих **жорстких посилань** (hardlink) на файл. Фактично, один і той самий файл може фігурувати на диску одночасно в різних каталогах або під різними іменами в одному каталозі.

Якщо редагувати файл із застосуванням одного з посилань на нього, то його вміст зміниться і за всіма іншими посиланнях. Щоб видалити файл із кількома жорсткими посиланнями, потрібно видалити всі ці посилання.



# 3. Файлова система

**Спеціальні файли** – це об’єкти ядра ОС, які у представляються у вигляді файлів. Спеціальні файли забезпечують можливості вбудовування певних абстракцій у файлову систему і таким чином підтримують парадигму «все є файл». Для створення таких файлів у Linux застосовується спеціальний системний виклик (функція).

Доступ до пристроїв у Linux здійснюється через спеціальні файли пристроїв, які можна відкривати та закривати, зчитувати інформацію та її записувати її у них.

З програми можна отримати доступ до файлів пристроїв і маніпулювати пристроями в системі (як фізичними, так і віртуальними). Всі пристрої в Linux можна розділити на дві групи:
1) символьні;
2) блокові пристрої.

Кожному типу пристроїв відповідає свій спеціальний файл пристрою.

# 3. Файлова система

Доступ до **символьного пристрою** здійснюється через файли символьних пристроїв, які організовані у вигляді лінійної послідовності байтів. Драйвер пристрою ставить байти в чергу, один за одним, а програма користувача зчитує їх в тому порядку, в якому вони були поміщені в чергу. Наприклад, клавіатура є символьним пристроєм.

Доступ до **блокового пристрою** здійснюється через файли блокових пристроїв як до масиву байтів. Драйвер пристрою відображає цей масив на пристрій з можливістю позиціонування, тобто програма користувача може читати його в довільному порядку. Блокові пристрої, це, наприклад, жорсткі диски, флеш-накопичувачі і т. п.

# 3. Файлова система

**Іменовані канали** (FIFO – «першим прийшов, першим вийшов») – це механізм взаємодії між процесами, який є каналом зв'язку між ними, коли здійснюється передача інформації з однієї програми до іншої. Такі файли створюються в пам'яті за допомогою спеціального системного виклику і не існують у файловій системі. Іменовані канали діють як і стандартні, але звернення до них відбувається через файл, який називається **особливим файлом FIFO**. Процеси можуть звертатися до цього файлу та обмінюватися інформацією.

**Сокети** – основна концепція мережевого програмування. Вони є вдосконаленим різновидом іменованих каналів для взаємодії між двома процесами, розташованими, зазвичай, на різних комп'ютерах у мережі. **Сокет домену Unix** (IPC-сокет) або **локальний сокет** використовується для взаємодії між процесами на локальній машині. Для цього застосовується спеціальний файл, розташований у файловій системі.

# 3. Файлова система


У Linux за замовчуванням використовується файлова система **ext4** (Fourth extended file system).

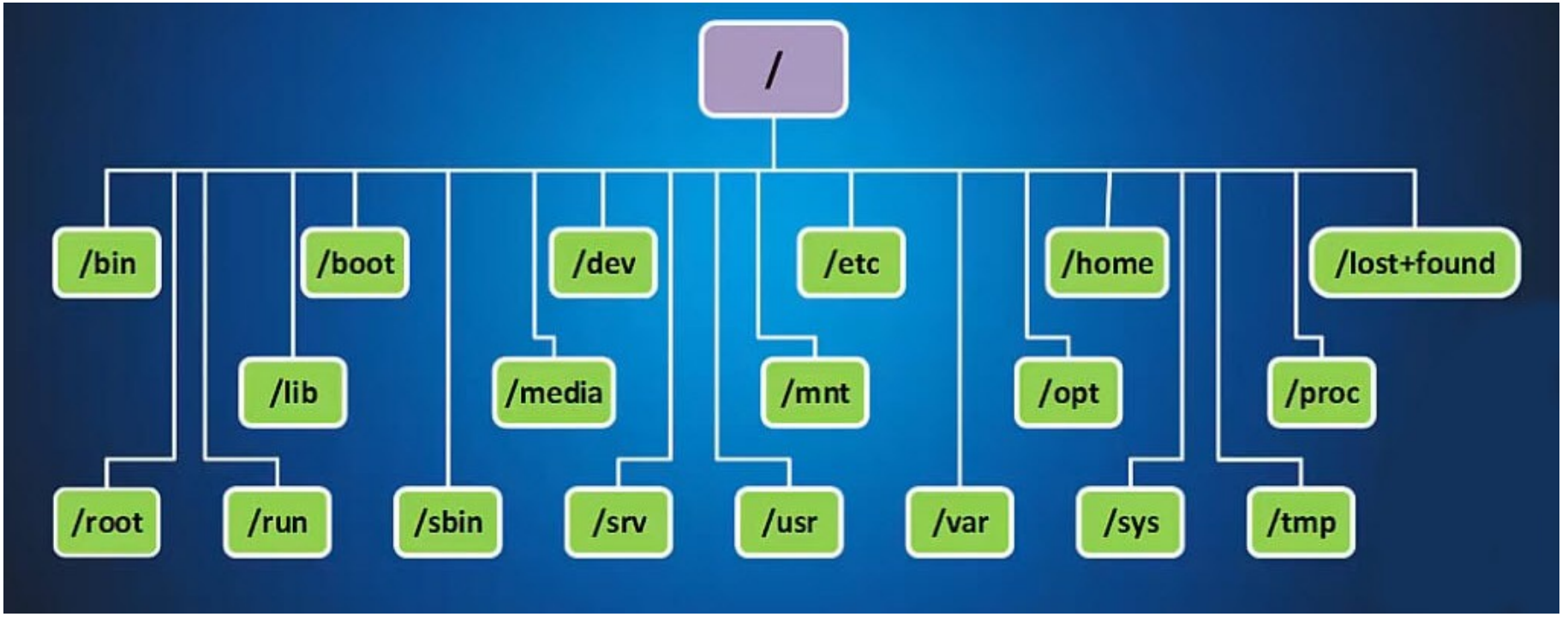
**Файлова система** – це ієрархічно структурований набір файлів та каталогів. Наприклад, FAT16, FAT32, NTFS, vFAT тощо.

**Журнальованою** називається файлова система, у якій підтримується ведення журналу змін, що дозволяє зберегти її цілісність при виникненні збоїв.

Linux надає **глобальний** та **єдиний простір імен** для файлів і каталогів (немає дрібнення на логічні диски, як, наприклад, у Windows – c:, d: і т. д.). Файлові системи різноманітних пристроїв (наприклад, флеш-накопичувачів) можна окремо додавати (або вилучати) до глобального простору файлів та каталогів. Це називаються **монтуванням** та **розмонтуванням**. Кожна файлова система може бути примонтована до конкретного каталогу, який називається точкою монтування (наприклад, CD може бути примонтований у точці /media/cdrom).

# 3. Файлова система

Типова структура файлової системи ОС Linux



# 4. Процеси

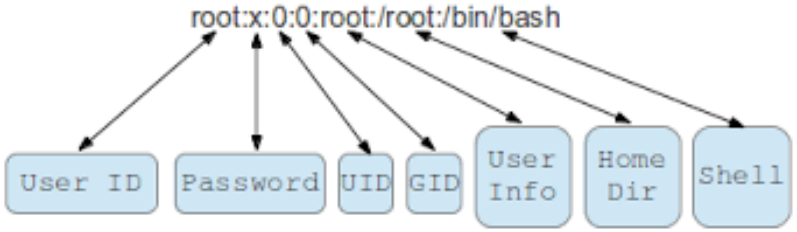

**Процес** – це об’єктний код програми, завантажений ОС з зовнішнього накопичувача у пам’ять комп'ютера і запущений на виконання (іншими словами, процес – це програма, що виконується).

**Об’єктний код** зберігається на диску у спеціальному **форматі виконання та зв’язування** (ELF – Executable and Linkable Format), що містить метадані і декілька розділів з програмним кодом і даними.

Кожен процес складається з одного або декількох **потоків** (threads) – мінімальних одиниць активності у процесі. Процеси бувають **однопотоковими** і **багатопотоковими**.

Для ідентифікації процесу застосовується унікальне ціле беззнакове число – **ідентифікатор процесу** (pid). У Linux процеси утворюють строгу ієрархію, яка називається **деревом процесів**. Корінь дерева перебуває у першому процесі – **процесі ініціалізації**. Процеси можуть запускати інші процеси. Вихідний процес називається **предком**, а породжений – **нащадком**.

# 5. Користувачі та групи

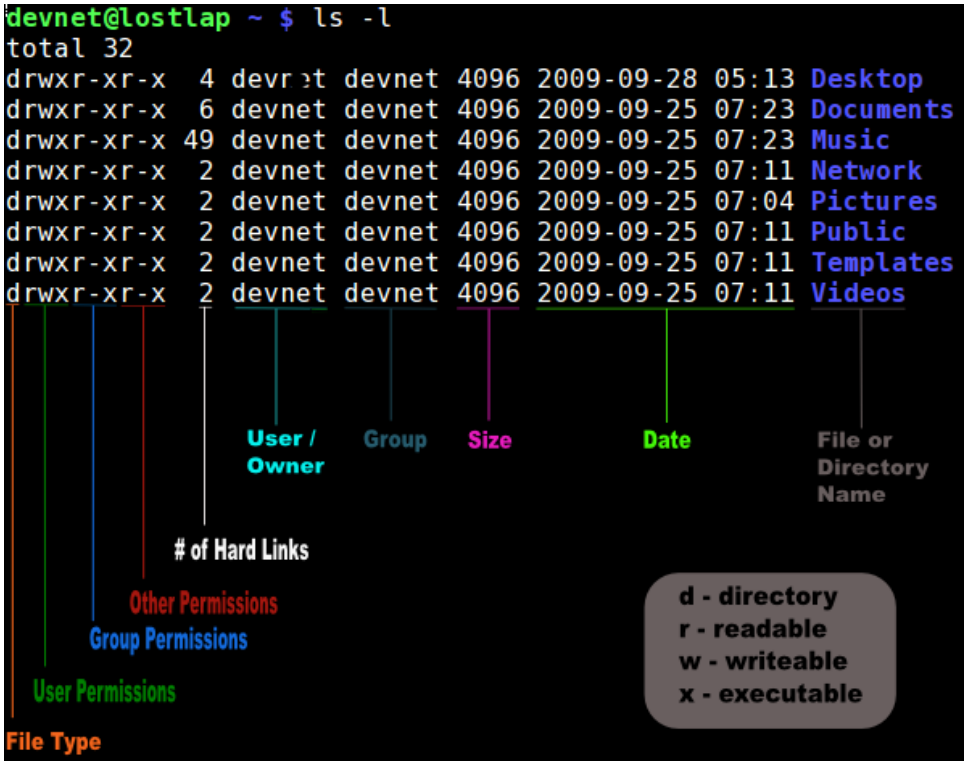



**Авторизація** в Linux забезпечується за допомогою системи **користувачів** та **груп**.

Кожному користувачу у Linux присвоюється унікальне ціле беззнакове число – **ідентифікатор користувача** (uid). Кожен процес асоціюється з одним uid, який ідентифікує користувача, що запустив процес. Цей ідентифікатор називається **реальним процесом uid**. Зв'язок між іменами користувачів (username) та відповідними uid зберігається у файлі /etc/passwd.

При вході в систему користувач вводить свій логін і пароль у спеціальну програму входу, яка в разі успіху запускає **оболонку користувача** (вона також визначається в /etc/passwd), і робить uid оболонки рівним uid користувача, що увійшов. Процеси-нащадки успадковують uid своїх предків. Ідентифікатор uid, який дорівнює 0, відповідає особливому користувачу (**суперкористувачу**), який називається **root**. Він має особливі привілеї і може робити в системі практично все.

# 5. Користувачі та групи




Для впорядкування прав у Linux, крім користувачів, існують так звані **групи**. Так само як і користувач, група має права доступу до тих чи інших каталогів, файлів і т. п. Для кожного файлу визначений не тільки користувач, але й група. Групи об’єднують користувачів для надання однакових повноважень на будь-які дії. Кожній групі відповідає унікальний **ідентифікатор групи** (gid).

Належність користувача до групи встановлюється адміністратором (root). Прикладами груп є adm, audio, disk, docker і тому подібне.

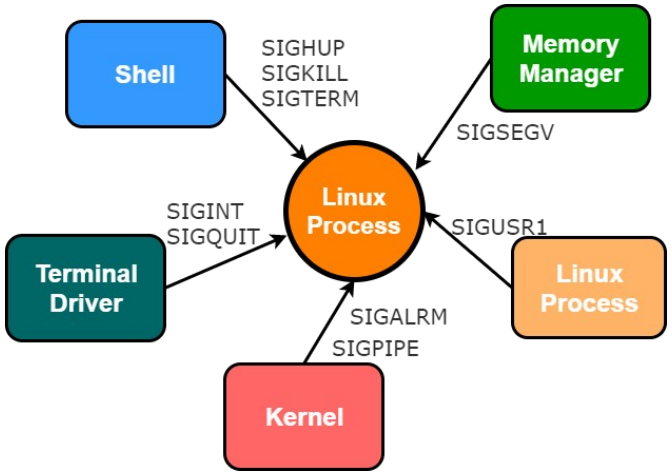
# 6. Права доступу

Будь-який файл в Linux асоціюється зі своїм власником, групою та трьома наборами бітів доступу, що описують права користувача-власника, відповідної групи та всіх інших. Ці біти визначають можливість читання файлу, запису в нього та його виконання. На кожен із цих трьох класів дій припадає по три біти (всього – дев'ять біт). Інформація про власника файлу та права зберігається в індексному дескрипторі файлу.

**Біти доступу та їх значення**

| Біт | Вісімкове значення | Текстове значення | Відповідні права |
|---|---|---|---|
| 8 | 400 | r-------- | Власник може читати |
| 7 | 200 | -w------- | Власник може записувати |
| 6 | 100 | --x------ | Власник може виконувати |
| 5 | 040 | ---r----- | Члени групи можуть читати |
| 4 | 020 | ----w---- | Члени групи можуть записувати |
| 3 | 010 | -----x--- | Члени групи можуть виконувати |
| 2 | 004 | ------r-- | Будь-хто може читати |
| 1 | 002 | -------w- | Будь-хто може записувати |
| 0 | 001 | --------x | Будь-хто може виконувати |

# 7. Сигнали та взаємодія між процесами




**Сигнали** – це механізм, що забезпечує односторонні асинхронні сповіщення. Сигнал може бути відправлений від ядра до процесу, від процесу до іншого процесу або від процесу до самого себе і повідомляє, що сталася якась подія (наприклад, користувач натиснув кнопку на клавіатурі).

Кожен сигнал задається числовою константою та текстовою назвою. Наприклад, сигнал SIGHUP використовується, щоб повідомити про зависання терміналу та має значення 1. Отримання сигналу призупиняє роботу поточного процесу для негайного виконання заздалегідь визначеної дії з обробки сигналу. Всі сигнали, за винятком SIGKILL (завершити процес) та SIGSTOP (зупинити процес), залишають процесам можливість вибору того, що має статися після отримання конкретного сигналу: процес може обробити отриманий сигнал або проігнорувати його. Сигнали є стандартним способом **взаємодії між процесами** в Unix-подібних системах.

# 8. Обробка помилок

Якщо при виконанні системних викликів виникла помилка, то проаналізувати її можна за допомогою глобальної змінної **errno**, оголошеної в спеціальному заголовному файлі **errno.h** наступним чином:

**extern int errno;**

При виникненні помилки, змінній errno присвоюється спеціальний код, значення якого пояснює причину помилки, наприклад, **EDOM**, якщо аргумент лежить поза області значень математичної функції.

Бібліотека мови програмування С надає набір функцій, які дозволяють розшифрувати конкретне значення errno. Прикладом є функція, визначена у заголовку stdio.h:

**void perror(const char *str);**

Ця функція направляє у стандартний потік виводу текстове повідомлення про помилку, що виникла, додаючи в якості префікса рядок, на який вказує параметр str.

# 8. Обробка помилок

Наприклад, спроба закрити файл із некоректним файловим дескриптором виведе на екран повідомлення «close: Bad file descriptor»:

```
#include <stdio.h>

int main(void)
{
    int fd = 1000; /* Некоректний файловий дескриптор */

    /* Спроба роботи з некоректним файловим дескриптором */
    if (close(fd) == -1)
    {
        perror("close");
        return 1;
    }
    return 0;
}
```