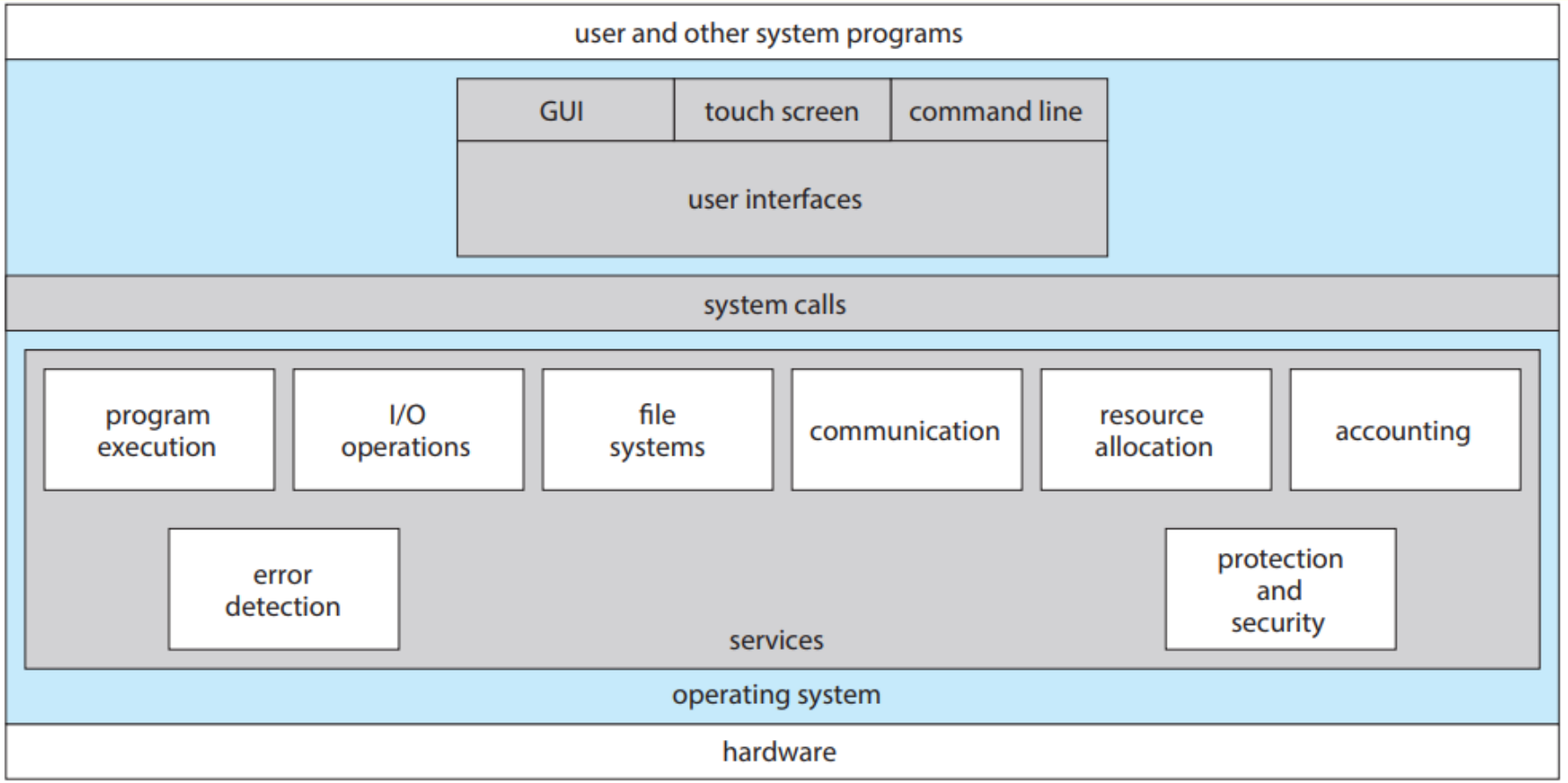
ОПЕРАЦІЙНІ СИСТЕМИ ТА СИСТЕМНІ ВИКЛИКИ Й СЛУЖБИ

Питання 1.4

Служби операційної системи

- ОС постачає середовище для виконання програм.
  - Спеціальні служби доступні програмам та їх користувачам



Поширені системні служби

- ***Користувацький інтерфейс***: віконний графічний інтерфейс, сенсорний інтерфейс, інтерфейс командного рядка.
- ***Виконання програм***: ОС повинна вміти завантажувати програму в пам'ять та запускати.
- ***Операції вводу-виводу***: використання файлів або пристроїв вводу-виводу. Для специфічних пристроїв можуть знадобитись спеціальні функції (запис у файлову систему, зчитування даних від інтерфейсу мережі тощо). Пряме керування пристроями вводу-виводу користувачем часто неможливе – це робить ОС.
- ***Операції з файловою системою***: ОС повинна вміти читати та записувати файли й папки, створювати та видаляти їх, шукати файли та виводити про них інформацію. Деякі ОС включають управління дозволами доступу до файлів та папок на основі належності власникові.

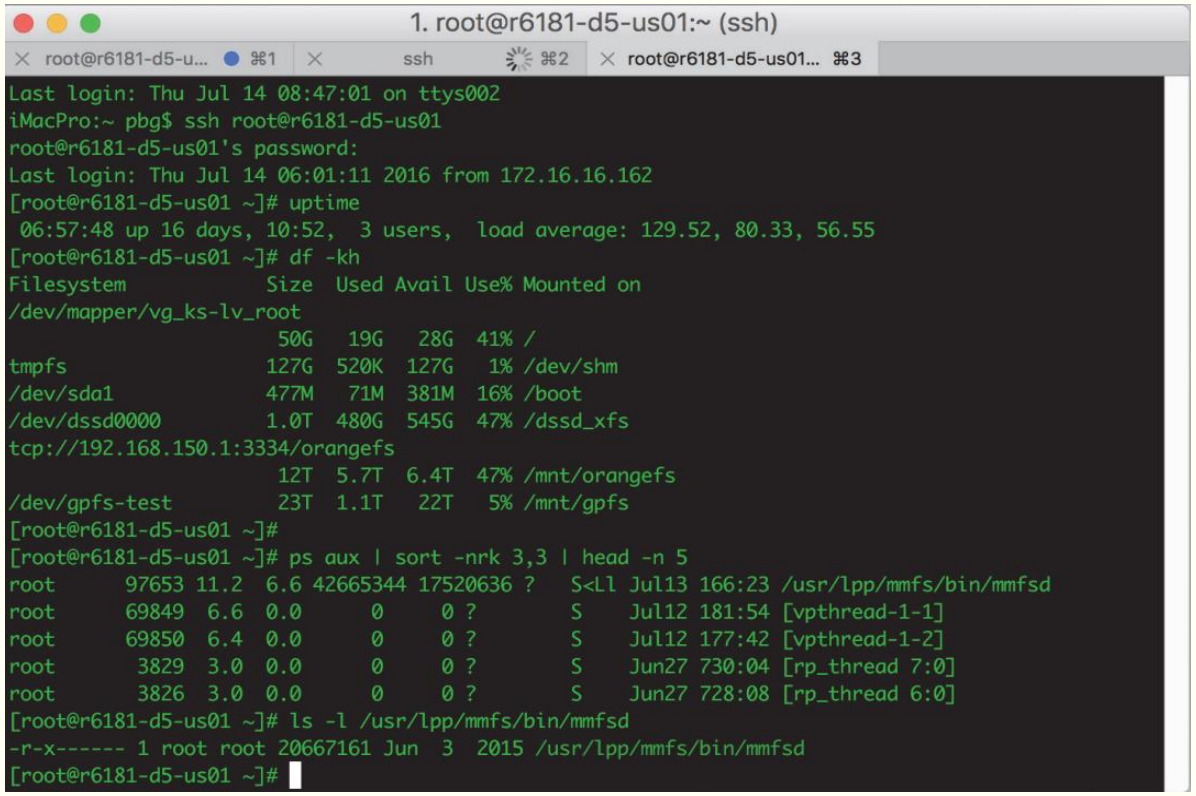
Поширені системні служби

- ***Комунікації***: в багатьох випадках процеси мають обмінюватись інформацією.
  - Реалізувати обмін можна за допомогою ***спільної пам'яті*** (shared memory – спільна для процесів область пам'яті) або передачі повідомлень (message passing – пакування інформації в пакети та переміщення від процесу до процесу за допомогою ОС).
- ***Виявлення помилок***: ОС постійно знаходить та виправляє помилки в роботі
  - ЦП, пам'яті (зникнення електропостачання, помилка пам'яті тощо),
  - пристроїв вводу-виводу (parity error на диску, відсутність підключення до мережі, нестача паперу в принтері та ін.),
  - користувацьких додатків (звернення до недоступного блоку пам'яті, переповнення тощо).
- Для кожного виду помилки ОС підбирає доречні перевірки коректності та цілісності обчислень:
  - припинення роботи ОС,
  - переривання роботи процесу,
  - повернення коду помилки для можливого подальшого виправлення тощо.

Управління ресурсами

- ***Алокація ресурсів.*** Коли одночасно працює багато процесів, кожному з них потрібно виділити ресурси.
  - Деякі ресурси (як ЦП-цикли, оперативна пам'ять та файлові сховища) можуть мати спеціальні коди (allocation code), а інші (як пристрої вводу-виводу) – більш узагальнені коди запиту та вивільнення.
  - Наприклад, ОС містить підпрограми планування роботи ЦП, які враховують його швидкодію, необхідний для виконання процес, кількість процесорних ядер та інші чинники.
  - Також можуть бути підпрограми для виділення ресурсів принтерам, USB- та іншим периферійним пристроям.
- ***Логування (Logging)***. Потрібно відстежувати, якими програмами та скільки ресурсів використовується.
  - Статистика користування може бути корисною системним адміністраторам для реконфігурації системи.
- ***Захист та безпека.*** Власники інформації, збереженої в багатокористувацьких чи мережевих комп'ютерних системах, можуть потребувати контролю за використанням цієї інформації.
  - Окремі конкурентні процеси не повинні заважати один одному та системі.
  - Захист передбачає управління доступом до системних ресурсів.
  - Безпека системи від зовнішніх загроз також важлива: ОС вимагає аутентифікації для отримання доступу до ресурсів, попереджає несанкціонований доступ через пристроїв вводу-виводу, зокрема, мережеві адаптери, веде систему дозволів та засторог.

Інтерфейси користувача та ОС: командні інтерпретатори



- Спеціальні програми, які працюють при ініціалізації процесу або при першому вході в ОС (в інтерактивних системах).
  - У системах з кількома доступними командними інтерпретаторами їх називають ***командними оболонками (shell)***.
  - Для UNIX / Linux доступні C shell, Bourne-Again shell, Korn shell та ін.
- Основна функція командного рядка – отримати та виконати введені користувачем команди.
  - Велика частина таких команд оперують файлами: створюють, видаляють, перелічують, копіюють, виконують тощо.

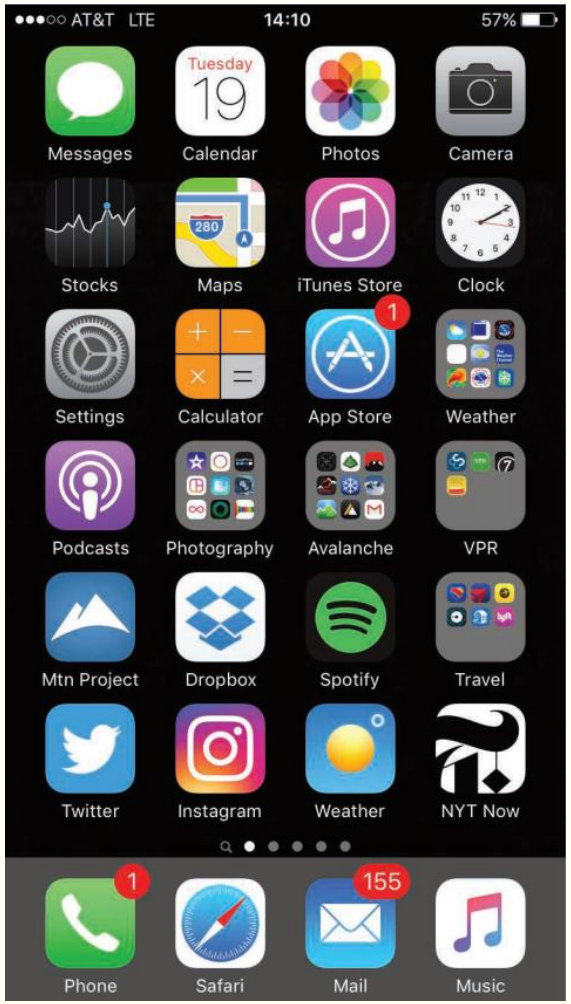
Інтерфейси користувача та ОС: командні інтерпретатори

- Команди можуть реалізовуватись двома загальними способами:
  - ***1) Командний рядок має код для виконання команди.***
    - Наприклад, команда видалити файл може змусити його перейти до частини коду, яка задає параметри та виконує відповідні системні виклики.
    - Кількість подібних команд визначає розмір командного рядка.
  - ***2) Командний рядок реалізує більшість команд через системні програми.***
    - Часто зустрічається в UNIX-подібних системах.
    - Командний рядок не розуміє команду, а використовує її для знаходження файлу і його подальшого завантаження й виконання.
    - UNIX-команда «rm file.txt» шукатиме файл з назвою «rm», завантажить його в пам'ять та виконає з параметром file.txt.
    - Логіка роботи команди rm повністю описана в коді з файлу rm.
    - Програмісти можуть просто додавати нові команди в систему, а командний рядок може бути невеликим та не змінюватись для додавання кількох команд.

Графічний інтерфейс користувача

- Застосовує метафори на основі вікон та меню з управлінням мишею.
  - Був запропонований на початку 1970-х років у дослідницькій лабораторії Xerox PARC та з'явився на комп'ютері Xerox Alto у 1973р.
  - Поширився завдяки комп'ютерам Apple Macintosh у 1980-х роках (Aqua).
  - GUI від Microsoft з'явився у Windows 1.0—надбудові над ОС MS-DOS.
- У UNIX-системах традиційно домінують інтерфейси командного рядка.
  - GUI-інтерфейси доступні від різних open-source проектів, таких як K Desktop Environment (KDE) та GNOME від проекту GNU.
  - Їх первинний код знаходиться у відкритому доступі зі спеціальними умовами ліцензування на його читання та модифікацію.

Сенсорний інтерфейс та вибір інтерфейсу

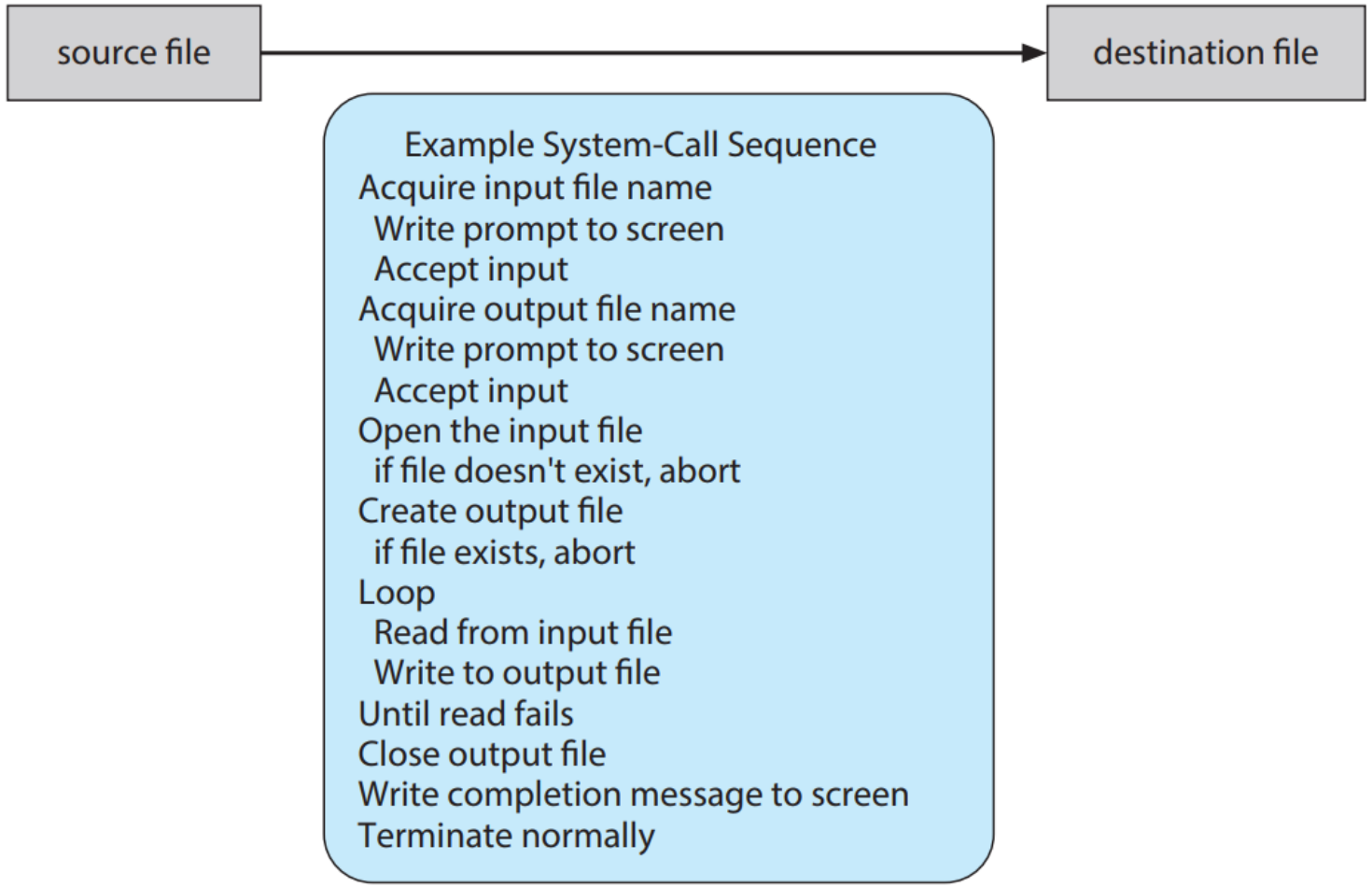


- Сенсорні екрани поширені серед смартфонів та планшетів.
  - iPad та iPhone використовують сенсорний інтерфейс Springboard.
- Вибір інтерфейсу зазвичай відповідає особистим вподобанням.
  - Системні адміністратори та просунуті користувачі часто використовують командний рядок, оскільки можуть швидше отримати доступ до потрібних їм дій.
  - У деяких системах GUI забезпечує лише частину доступних системних функцій, а менш поширені задачі вирішуються в командному рядку.
  - Інтерфейс командного рядка зазвичай спрощує виконання повторюваних задач, оскільки допускає власне програмування (скрипти).
  - Shell-скрипти дуже поширені на системах, орієнтованих на роботу з командним рядком, таких як UNIX та Linux.

Вибір інтерфейсу взаємодії

- Користувачі Windows зазвичай використовують GUI-середовище та майже не користуються shell-інтерфейсом.
  - Останні версії Windows адаптовані як для стандартного GUI, так і для сенсорних екранів.
  - Mac OS історично не постачала інтерфейс командного рядка, проте з релізом macOS на UNIX-ядрі стали доступні як Aqua GUI, так і командний рядок.
- Для мобільних ОС iOS та Android доступні командні інтерпретатори, проте вони ідко використовуються.
  - Інтерфейс користувача може змінюватись від системи до системи та навіть залежно від користувача; проте зазвичай від суттєво відокремлений від структури ОС.
  - Проєктування корисного та інтуїтивного інтерфейсу користувача не є прямою функцією ОС.

Системні виклики (System calls)



- Забезпечують інтерфейс для служб операційної системи.
  - Зазвичай доступні у вигляді функцій, написаних мовами C та C++, проте деякі низькорівневі задачі (наприклад, прямий доступ до заліза) може реалізовуватись за допомогою ассемблерних інструкцій.
- Приклад: проста програма для зчитування даних з одного файлу та їх копіювання в інший.
  - 1) cp in.txt out.txt
  - 2) Користувач може застосувати мишу та віконний інтерфейс.

Прикладний інтерфейс програмування (API)

- API задає набір функцій, доступних розробнику додатка, включаючи параметри, що передаються в кожну функцію, та значення, повернення яких очікується.
- Найбільш поширені API для розробки:
  - Windows API для Windows-систем;
  - POSIX API для POSIX-базованих систем (майже всі версії UNIX, Linux та macOS);
  - Java API для програмістів, які запускають додатки на Java virtual machine.
- Програміст отримує доступ до API через бібліотеку коду, яку постачає ОС.
  - Для програм під UNIX та Linux – написана мовою C бібліотека libc.
  - Кожна ОС має власні назви для системних викликів.
  - За лаштунками API-функції зазвичай задіюють системні виклики.
  - Наприклад, Windows-функція function CreateProcess() насправді викликає системний виклик NTCreateProcess() з ядра Windows.

Приклад стандартного API – функція read() в ОС UNIX / Linux

```
#include <unistd.h>

ssize_t     read(int fd, void *buf, size_t count)
```

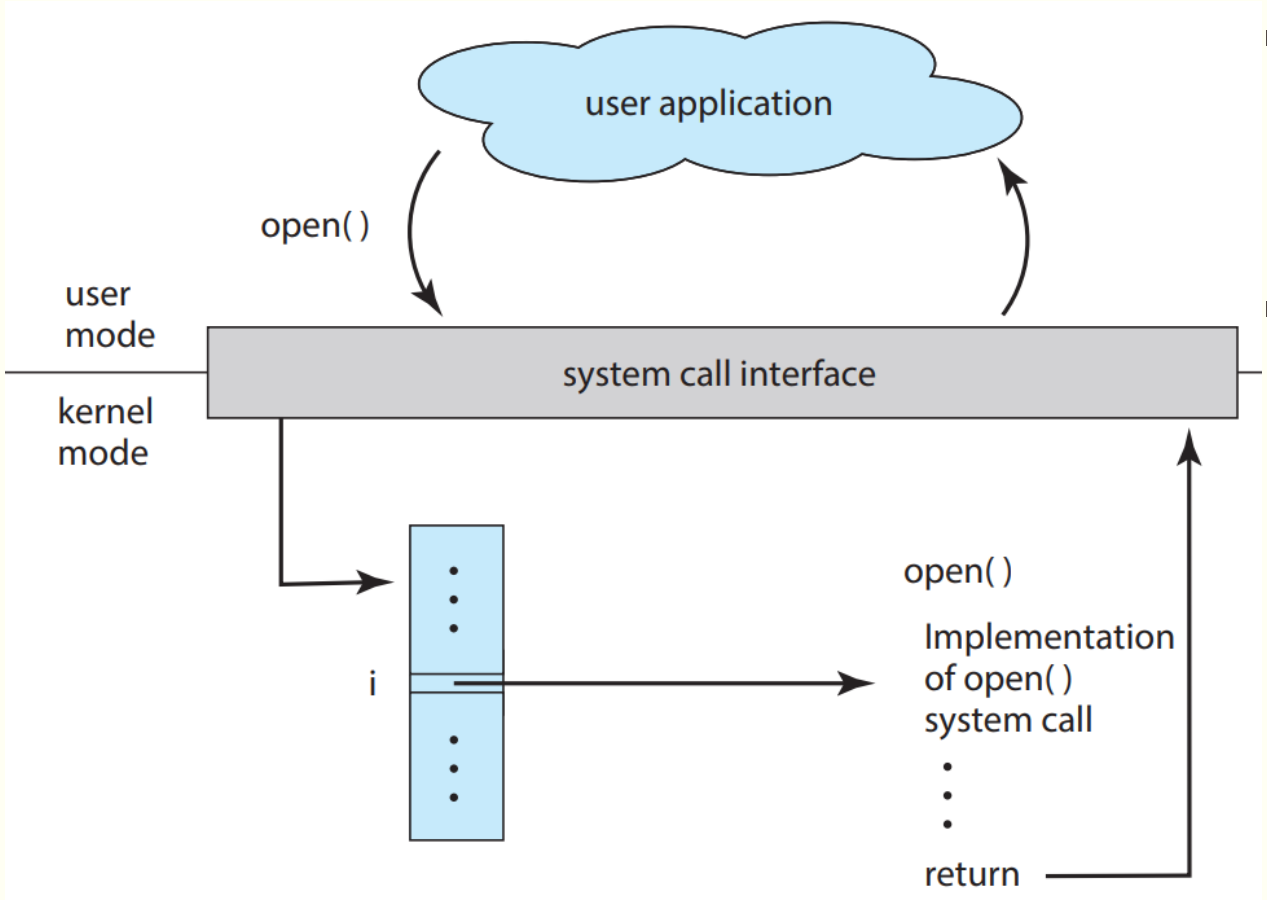
return value — function name — parameters

- Програма, яка використовує функцію read(), повинна включати заголовковий файл unistd.h, оскільки він визначає типи даних ssize_t tasize_t та багато іншого.
- Параметри, які передаються в read() :
  - int fd—дескриптор зчитуваного файлу;
  - void *buf—буфер, у який будуть зчитуватись дані;
  - size_t count—максимальна кількість байтів для зчитування в буфер.
- При успішному читанні повертається кількість зчитаних байтів.
  - Повернене значення 0 вказує на кінець файлу, а −1 – на помилку виконання.

Чому програміст працює з API, а не системними викликами?

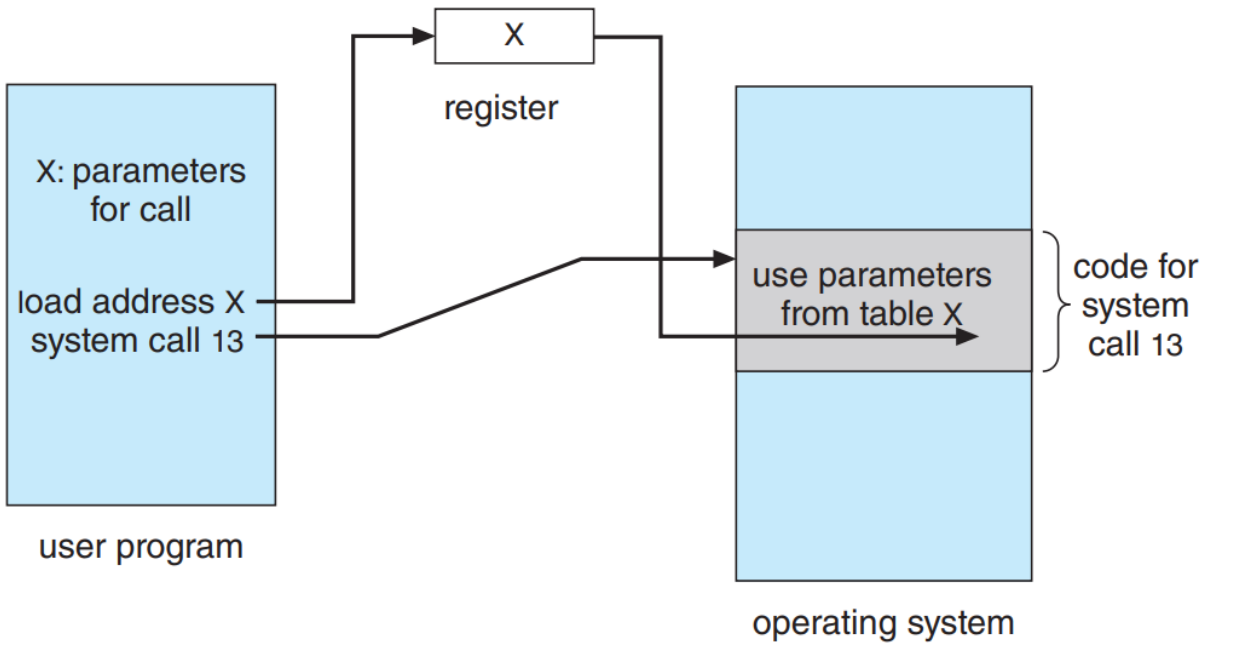
- Спроектований на базі API додаток може скомпілюватись та запуститись на різних ОС з підтримкою цього API (хоч насправді архітектурні відмінності ускладнюють таку можливість).
  - Крім того, системні виклики можуть бути більш деталізованими та складними в роботі, ніж API.
  - Все ж, існує значна кореляція між API-функціями та відповідними їм системними викликами з ядра.
- Інший важливий чинник обробки системних викликів – ***середовище виконання (run-time environment, RTE)—***повний набір ПЗ, необхідного для виконання додатків, написаних заданою мовою програмування (компілятори/інтерпретатори, бібліотеки, завантажувачі тощо).
  - RTE забезпечує інтерфейс для системних викликів, який слугує лінком на доступні від ОС системні виклики, перехоплює виклики API-функцій та звертається до відповідних системних викликів ОС.
  - Зазвичай, інтерфейс системних викликів підтримує індексовану таблицю (номер виклику – виклик).
  - Він звертається до передбаченого системного виклику в ядрі ОС та повертає статус цього виклику.

Принцип роботи з системними викликами



- Викликач (caller) не потребує знань про реалізацію системного виклику чи його дії.
  - Він слідує API та працює з результатом системного виклику.
- Системні виклики трапляються в різних випадках, залежно від використання комп’ютера.
  - Тип та об’єм інформації для передачі залежить від ОС та конкретного системного виклику.
  - Наприклад, для отримання введеної інформації необхідно вказати джерело (файл або пристрій) даних, адресу та величину буферу пам’яті, в який буде здійснено зчитування.
  - Ця інформація може передаватись неявно.

Методи передачі параметрів операційній системі



Найпростіший підхід – передавати параметри в регістри.

- Проте інколи параметрів більше, ніж регістрів.
- Тоді параметри зберігають у блоці або таблиці в пам'яті, а адреса блоку передається в якості параметра в регістр.

Linux комбінує ці підходи.

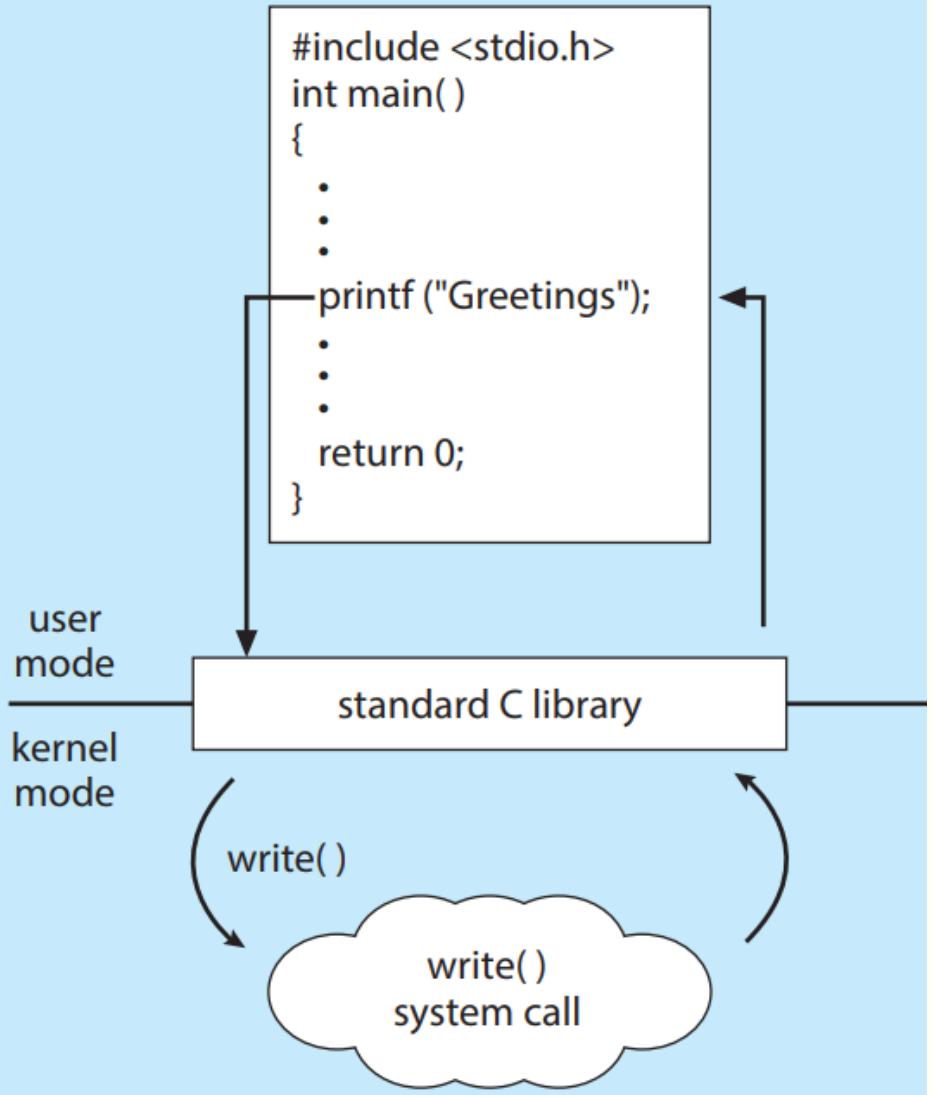
- Якщо параметрів 5 або менше, використовуються регістри. Інакше – блочний підхід.
- Параметри також можуть пушитись програмою в стек та вийматись з нього операційною системою.
- Деякі ОС віддають перевагу блочному або стековому підходу, оскільки вони не обмежують кількість або розмір переданих параметрів.

Типи системних викликів (6 категорій)

- ***Управління процесами (Process control)***
  - Створити, перервати процес
  - Завантажити, виконати
  - Отримати атрибути процесу, задати їх
  - Очікувати подію, сповістити про подію
  - Виділяти та очищати пам’ять
- ***Файловий менеджмент***
  - Створювати та видаляти файли
  - Читати, записувати, переміщати (reposition)
  - Отримувати атрибути файлів та задавати їх
- ***Управління пристроями***
  - Надсилати запит до/звільняти пристрій
  - Читати, записувати, переміщати (reposition)
  - Отримувати атрибути пристрою та задавати їх
  - Логічно монтувати та відмонтувати ристрої
- ***Інформаційна підтримка (Information maintenance)***
  - Отримати дату/час, задати дату/час
  - Отримати/задати системну дату
  - Отримати атрибути процесу, файлу або пристрою
  - Задати атрибути процесу, файлу або пристрою
- ***Взаємодія (Communications)***
  - Створити, видали зв’язок для взаємодії
  - Надіслати, отримати повідомлення
  - Передати інформацію про статус
  - Монтувати/відмонтувати віддалений пристрій
- ***Захист***
  - Отримати дозволи для файлу
  - Задати дозволи для файлу

Системні виклики Windows та UNIX

	Windows	Unix
Process control	`CreateProcess()`	`fork()`
	`ExitProcess()`	`exit()`
	`WaitForSingleObject()`	`wait()`
File management	`CreateFile()`	`open()`
	`ReadFile()`	`read()`
	`WriteFile()`	`write()`
	`CloseHandle()`	`close()`
Device management	`SetConsoleMode()`	`ioctl()`
	`ReadConsole()`	`read()`
	`WriteConsole()`	`write()`
Information maintenance	`GetCurrentProcessID()`	`getpid()`
	`SetTimer()`	`alarm()`
	`Sleep()`	`sleep()`
Communications	`CreatePipe()`	`pipe()`
	`CreateFileMapping()`	`shm_open()`
	`MapViewOfFile()`	`mmap()`
Protection	`SetFileSecurity()`	`chmod()`
	`InitlializeSecurityDescriptor()`	`umask()`
	`SetSecurityDescriptorGroup()`	`chown()`



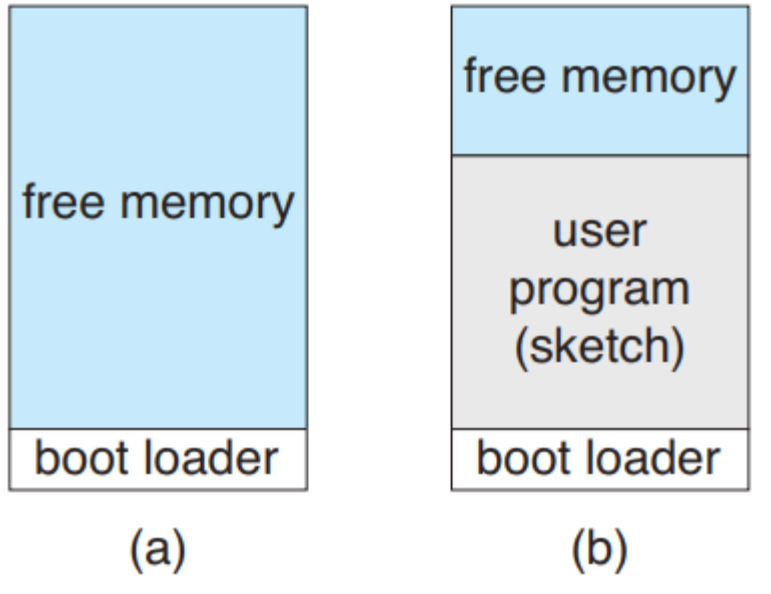
Управління процесами

- Працююча програма повинна мати змогу зупинити свою роботу нормально (end()) або абнормально (abort()).
  - При абнормальному завершенні виконання програми (переривання або виняток (error trap)) інколи генерується дамп пам'яті та повідомлення про помилку.
  - Дамп записується в спеціальний log-файл на диску та може переглядатись налагоджувальником (debugger), щоб визначити причину проблеми.
  - У будь-якому випадку ОС повинна передати керування викликаючому командному рядку, який далі зчитає наступну команду.
  - В інтерактивній системі командний рядок просто переходить до наступної команди; передбачається, що користувач підбере доречну команду для відповіді на помилку.
  - У GUI-системі може з'явитись спливаюче вікно та запитати користувача про подальші дії.
- Процес, який виконує одну програму, може захотіти load() та execute() іншу.
  - Кому повертати управління, коли завантажена програма перерве виконання?
  - Залежить від того, чи поточна програма була втрачена, збережена або їй дозволено виконуватись далі паралельно з новою програмою.
  - Якщо управління повертається поточній програмі після переривання нової (викликаної), необхідно зберегти memory image поточної програми.
  - Якщо обидві програми продовжують конкурентно виконуватись, створюється новий процес. Часто для цього використовують спеціальний системний виклик create_process().

Управління процесами

- При створенні одного або групи процесів слід мати змогу керувати їх виконанням.
  - Для цього потрібно мати можливість визначати та змінювати атрибути процесу (пріоритет, максимальний час виконання тощо) за допомогою функцій на зразок get_process_attributes() і set_process_attributes()).
  - Краще мати можливість перервати виконання створеного процесу функціями на зразок terminate_process(), якщо процес працює некоректно або більше не потрібний.
- Створивши нові процеси, можемо очікувати завершення їх виконання.
  - Протягом певного періоду часу – wait_time(), з настанням події - wait_event() чи надходженням сигналу про її настання (signal_event()).
- Часто кілька процесів мають доступ до спільних даних.
  - Для підтримки цілісності даних ОС часто забезпечують механізм блокування (**lock**) процесом таких даних.
  - Зазвичай, використовуються системні виклики на зразок acquire_lock() та release_lock().

Приклад управління процесом для Arduino

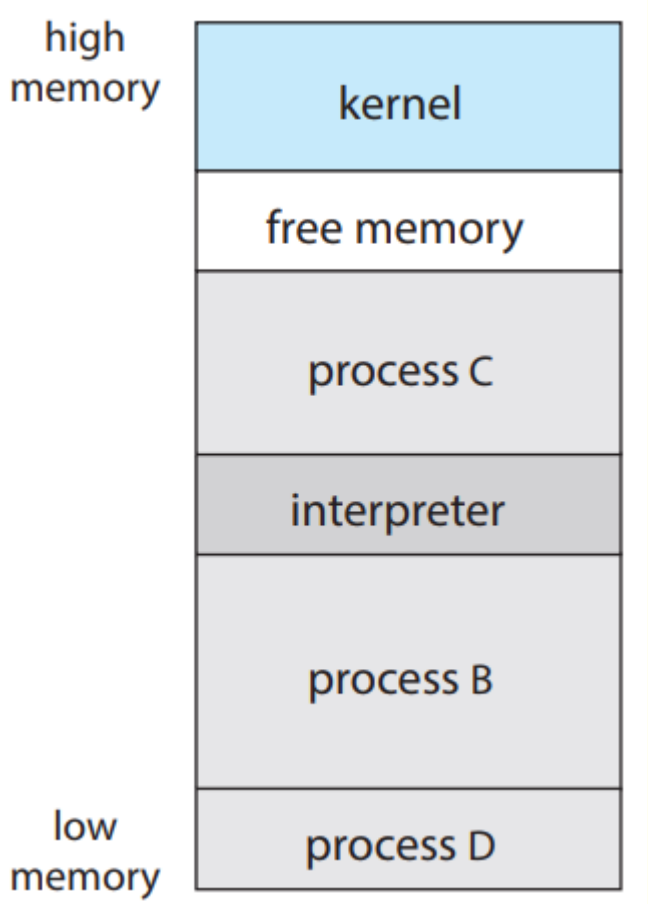


(a) At system startup.

(b) Running a sketch.

- Arduino складається з мікроконтролера та датчиків, які реагують на деякі події (зміну освітленості, температуру, атмосферний тиск тощо).
  - Спочатку на ПК пишеться програма, скомпільована версія якої (скетч) буде завантажена у флеш-пам’ять Arduino по USB.
  - Стандартна платформа Arduino не постачає ОС; невеликі шматки коду, що називають завантажувачем (boot loader), завантажують скетч у спеціальну область пам’яті Arduino.
  - Після завантаження скетч запускається та очікує на події, на які він запрограмований реагувати.

Приклад багатозадачної ОС – FreeBSD



- Коли користувач авторизується в системі, за його вибором запускається shell, який очікуватиме на команди та запускатиме програми за запитом користувача.
  - Оскільки FreeBSD – багатозадачна система, командний рядок продовжуватиме роботу разом з іншими програмами.
- Для створення нового процесу командний рядок виконує системний виклик ***fork()***.
  - Обрана програма завантажується в пам'ять за допомогою системного виклику ***exec()*** і починає виконання.
  - Залежно від введеної команди, shell або очікує завершення процесу, або виконує його «фоново» ("in the background").
  - В останньому випадку shell негайно очікує вводу нової команди.
  - Фоновий процес не може отримувати ввід з клавіатури напряму, оскільки цей ресурс занятий командним рядком – ввід вивід виконується через файли або GUI-інтерфейс.
  - Коли процес закінчив роботу, він виконує системний виклик exit(), повертаючи управління викликаючому процесу з числовим кодом статусу.

Управління файлами – кілька системних викликів

- Системні виклики ***create()*** та ***delete()*** вимагають назви файлу та, можливо, деякі його атрибути.
  - Для використання створеного файлу його відкривають – ***open()***. Доступні системні виклики ***read()***, ***write()***, ***reposition()*** (зміна місця в файлі, на яке вказує курсор), ***close()***.
- Аналогічний набір операцій потрібен для директорій (якщо є в файловій системі).
  - Атрибути файлів включають назву, тип, protection codes, accounting information тощо. Потрібні, принаймні, системні виклики ***get_file_attributes()*** and ***set_file_attributes().***
  - Деякі ОС постачають набагато більше викликів, зокрема ***move()***, ***copy()*** та ін.
  - Інші забезпечують API, який виконує ці операції в коді та за допомогою інших системних викликів.
  - Решта можуть постачати системні програми для виконання цих задач. Якщо системні програми доступні для виклику іншими програмами, кожна з них є API щодо них.

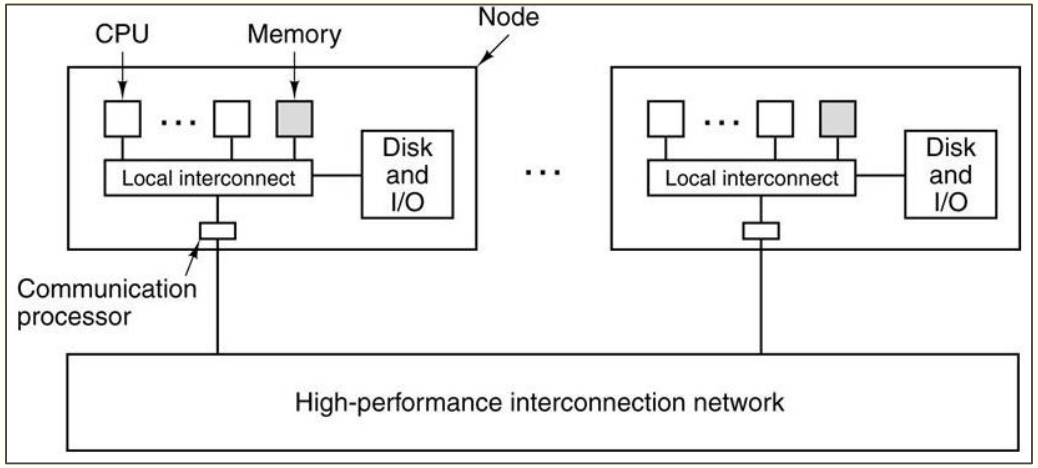
Управління пристроями

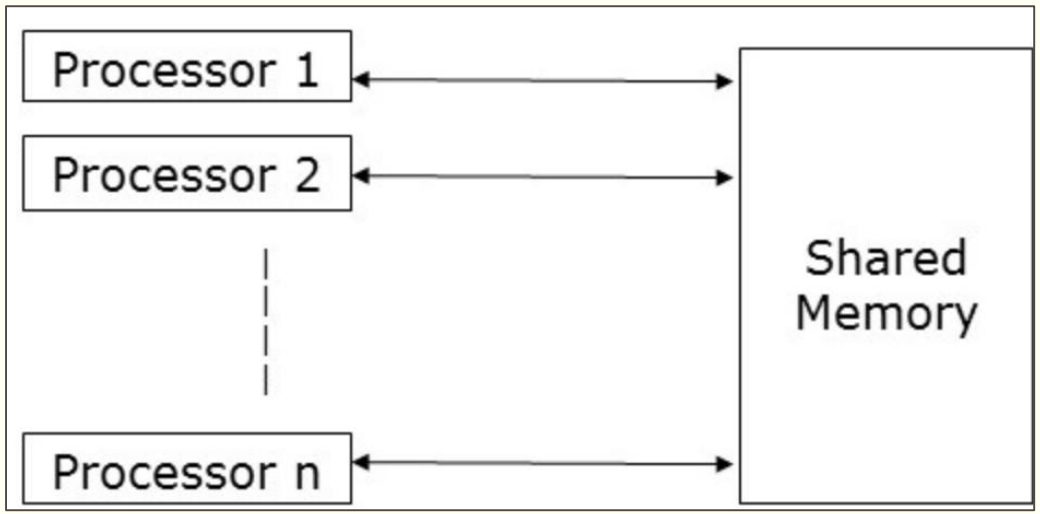
- Процесу для виконання можуть бути потрібними кілька ресурсів: оперативна пам’ять, дискові накопичувачі, доступ до файлів та ін.
  - Доступними ресурсами дозволяється управляти та повертати керування до user process. Інакше процес очікуватиме достатніх ресурсів.
- Різні ресурси, якими керує ОС, можуть розглядатись як пристрої.
  - Деякі з них фізичні (наприклад, жорсткий диск), інші – абстрактні або віртуальні (наприклад, файли).
  - Багатокористувацька система може вимагати спочатку надіслати запит (***request()***) до пристрою, щоб забезпечити ексклюзивність його використання.
  - Після завершення роботи з пристроєм він вивільняється – ***release()***.
  - Інші ОС дозволяють некерований доступ до пристроїв. Це загрожує конфліктом пристроїв та дедлоками.
- Після запиту пристрою (та передачі доступу до нього) можна виконувати системні виклики ***read()***, ***write()***, та можливо, ***reposition()***.
  - Подібність пристроїв вводу-виводу та файлів призводить для багатьох ОС (включаючи UNIX) до комбінованої, file–device структури – єдиних системних викликів для файлів та пристроїв.
  - Інколи пристроїв вводу-виводу ідентифікуються спеціальними назвами файлів, розташуванням директорій чи атрибутами файлів.

Інформаційна підтримка

- Багато системних викликів існують просто для передачі інформації між користувацькою програмою та ОС.
  - Наприклад, більшість ОС мають системний виклик для повернення поточних ***time()*** і ***date()***, інформації про систему (номер версії, доступний дисковий простір або кількість вільної оперативної пам'яті тощо).
- Інші системні виклики корисні для налагоджування програми.
  - Поширеним є ***dump()*** пам'яті. Програма ***strace***, доступна на Linux-системах, перелічує кожен виконаний системний виклик.
  - Навіть мікропроцесори визначають CPU-режим «single step», в якому ЦП виконує виняток (trap) після кожної інструкції. Зазвичай виняток перехоплюється дебагером.
- Багато ОС забезпечують часовий профіль програми, щоб вказувати тривалість її виконання в деякій області (кількох областях) пам'яті.
  - Часовий профіль потребує або трасувальника, або регулярних переривань таймера (при перериванні записується значення program counter, достатньо часті переривання дадуть статистичну картину часу, проведеного програмою в різних областях пам'яті).
  - Також ОС тримає інформацію про всі свої процеси, використані для доступу до інформації системні виклики.
  - Загалом, використовуються системні виклики для отримання чи задання інформації про процес – ***get_process_attributes()*** та ***set_process_attributes()***.

Взаємодія (Communication)





Існують 2 поширені моделі міжпроцесної взаємодії: ***модель з передачею повідомлень*** (message-passing model) та ***модель зі спільною пам'яттю*** (shared-memory model).

- У **message-passing model**, взаємодіючі процеси обмінюються повідомленнями напряму або опосередковано through a common mailbox.
- До початку взаємодії необхідно відкрити зв'язок: назва іншого комунікатора повинна бути відомою.
- Кожен комп'ютер у мережі має **host name**, за яким його всі бачать, а також мережевий ідентифікатор на зразок IP-адреси.
- Аналогічно, кожен процес має **process name**, що транслюється в ідентифікатор, за яким ОС може звернутись до процесу (системні виклики get_hostid() та get_processid()).
- The identifiers are then passed to the generalpurpose open() and close() calls provided by the file system or to specific open connection() and close connection() system calls, depending on the system's model of communication.
- The recipient process usually must give its permission for communication to take place with an accept connection() call.
- Більшість процесів, that will be receiving connections are special-purpose **daemons**, which are system programs provided for that purpose.
- Вони виконують «очікування» для виклику connection() and are awakened when a connection is made.
- The source of the communication, known as the **client**, and the receiving daemon, known as a **server**, then exchange messages by using read message() and write message() system calls.
- The close connection() call terminates the communication.

Модель зі спільною пам’яттю

- Процеси використовують системні виклики до спільної пам’яті create() та attach(), щоб створити та отримати доступ до областей пам’яті, власниками яких є інші процеси.
  - Згадайте, що зазвичай ОС намагається усунути можливість такого доступу.
  - Процеси повинні обмінюватись інформацією через зчитування/запис даних у спільній пам’яті.
  - Форма даних визначається процесами та не управляється ОС.
  - Також процеси відповідальні за координацію між собою одночасного запису в одне місце пам’яті – схему роботи з потоками (threads).
- Більшість ОС імплементують обидві моделі.
  - Передача повідомлень корисна для обміну невеликими об’ємами даних, оскільки уникаються конфлікти. Також цю модель простіше реалізувати.
  - Спільна пам’ять дозволяє забезпечити максимальну швидкість та зручність комунікації, оскільки communication, since it can be done at memory transfer speeds when it takes place within a computer. Problems exist, however, in the areas of protection and synchronization between the processes sharing memory.

Захист (Protection)

- Protection provides a mechanism for controlling access to the resources provided by a computer system.
  - Historically, protection was a concern only on multiprogrammed computer systems with several users.
  - However, with the advent of networking and the Internet, all computer systems, from servers to mobile handheld devices, must be concerned with protection.
  - Typically, system calls providing protection include set permission() and get permission(), which manipulate the permission settings of resources such as files and disks.
  - The allow user() and deny user() system calls specify whether particular users can—or cannot—be allowed access to certain resources.

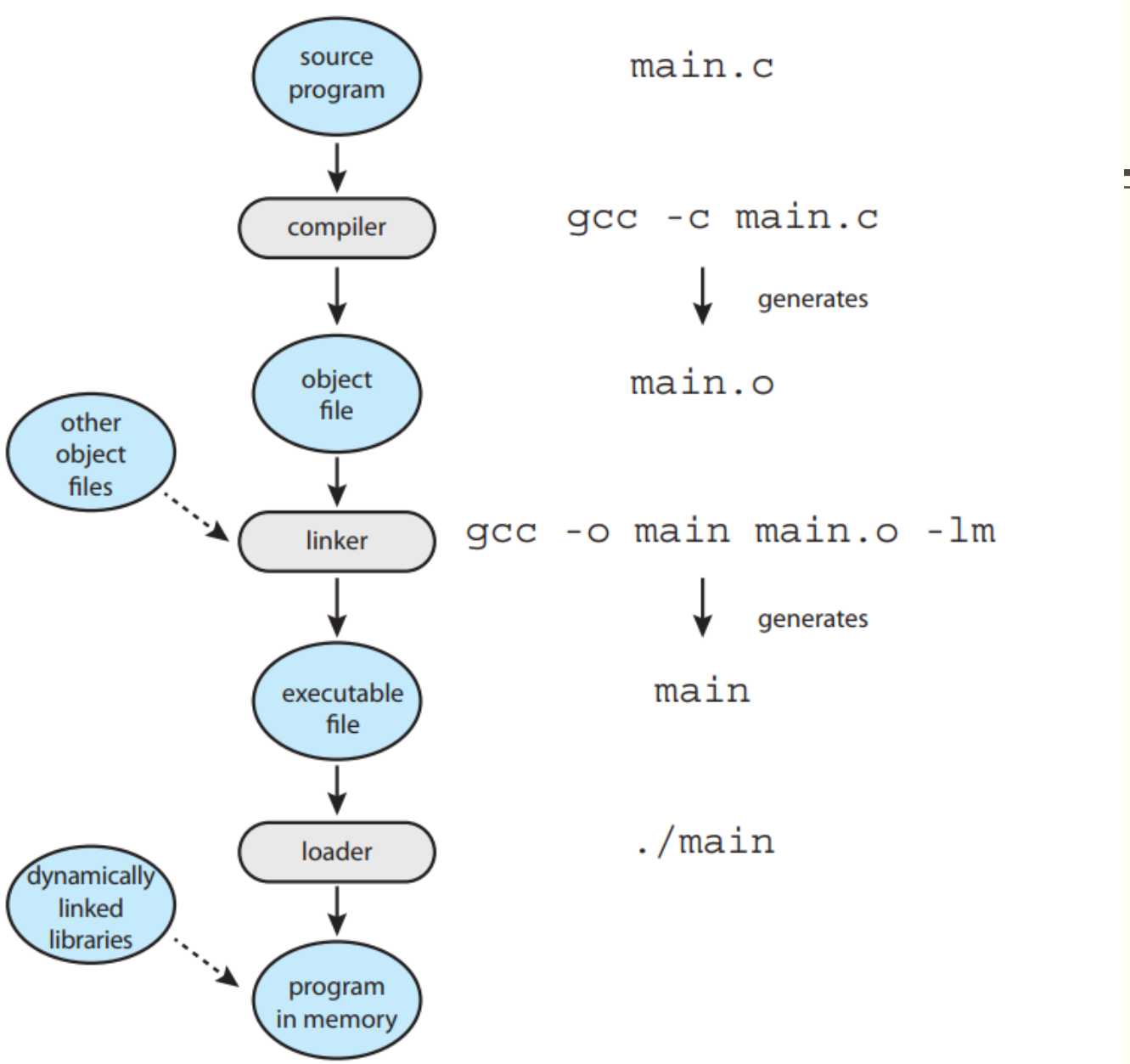
Системні служби (системні утиліти)

- Постачають зручне середовище для розробки та виконання програм.
- Деякі з них – просто користувацькі інтерфейси для системних викликів, інші – значно складніші. Можна розділити їх на категорії:
  - **File management**. These programs create, delete, copy, rename, print, list, and generally access and manipulate files and directories.
  - **Status information**. Some programs simply ask the system for the date, time, amount of available memory or disk space, number of users, or similar status information. Others are more complex, providing detailed performance, logging, and debugging information. Typically, these programs format and print the output to the terminal or other output devices or files or display it in a window of the GUI. Some systems also support a **registry**, which is used to store and retrieve configuration information.
  - **File modificatio** . Several text editors may be available to create and modify the content of files stored on disk or other storage devices. There may also be special commands to search contents of files or perform transformations of the text.
  - **Programming-language support**. Compilers, assemblers, debuggers, and interpreters for common programming languages (such as C, C++, Java, and Python) are often provided with the operating system or available as a separate download.
  - **Program loading and execution**. Once a program is assembled or compiled, it must be loaded into memory to be executed. The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders. Debugging systems for either higher-level languages or machine language are needed as well.

- **Communications**. These programs provide the mechanism for creating virtual connections among processes, users, and computer systems. They allow users to send messages to one another's screens, to browse web pages, to send e-mail messages, to log in remotely, or to transfer files from one machine to another.

- **Background services**. All general-purpose systems have methods for launching certain system-program processes at boot time. Some of these processes terminate after completing their tasks, while others continue to run until the system is halted. Constantly running system-program processes are known as **services**, **subsystems**, or daemons. One example is the network daemon discussed in Section 2.3.3.5. In that example, a system needed a service to listen for network connections in order to connect those requests to the correct processes. Other examples include process schedulers that start processes according to a specified schedule, system error monitoring services, and print servers. Typical systems have dozens of daemons. In addition, operating systems that run important activities in user context rather than in kernel context may use daemons to run these activities.

- Along with system programs, most operating systems are supplied with programs that are useful in solving common problems or performing common operations. Such **application programs** include web browsers, word processors and text formatters, spreadsheets, database systems, compilers, plotting and statistical-analysis packages, and games.
  The view of the operating system seen by most users is defined by the application and system programs, rather than by the actual system calls. Consider a user's PC. When a user's computer is running the macOS operating system, the user might see the GUI, featuring a mouse-and-windows interface. Alternatively, or even in one of the windows, the user might have a commandline UNIX shell. Both use the same set of system calls, but the system calls look different and act in different ways. Further confusing the user view, consider the user dual-booting from macOS into Windows. Now the same user on the same hardware has two entirely different interfaces and two sets of applications using the same physical resources. On the same hardware, then, a user can be exposed to multiple user interfaces sequentially or concurrently.



Linkers and Loaders

- Usually, a program resides on disk as a binary executable file—for example, a.out or prog.exe.
  - To run on a CPU, the program must be brought into memory and placed in the context of a process.
- Source files are compiled into object files that are designed to be loaded into any physical memory location, a format known as an relocatable object file.
  - Next, the linker combines these relocatable object files into a single binary executable file.
  - During the linking phase, other object files or libraries may be included as well, such as the standard C or math library (specified with the flag -lm).
  - A **loader** is used to load the binary executable file into memory, where it is eligible to run on a CPU core.
  - An activity associated with linking and loading is **relocation**, which assigns final addresses to the program parts and adjusts code and data in the program to match those addresses so that, for example, the code can call library functions and access its variables as it executes.

- При введенні назви програми в командному рядку UNIX-систем shell спочатку створює новий процес для запуску програми за допомогою системного виклику ***fork()***.
  - Потім shell викликає завантажувач за допомогою системного виклику exec(), передаючи йому назву виконуваного файлу.
  - Потім завантажувач завантажує задану програму в пам’ять using the address space of the newly created process. (When a GUI interface is used, double-clicking on the icon associated with the executable file invokes the loader using a similar mechanism.)
- The process described thus far assumes that all libraries are linked into the executable file and loaded into memory.
  - In reality, most systems allow a program to dynamically link libraries as the program is loaded.
  - Windows, for instance, supports dynamically linked libraries (DLLs).
  - The benefit of this approach is that it avoids linking and loading libraries that may end up not being used into an executable file.
  - Instead, the library is conditionally linked and is loaded if it is required during program run time.
  - For example, in Figure 2.11, the math library is not linked into the executable file main.
  - Rather, the linker inserts relocation information that allows it to be dynamically linked and loaded as the program is loaded.

- Object files and executable files typically have standard formats that include the compiled machine code and a symbol table containing metadata about functions and variables that are referenced in the program.
  - For UNIX and Linux systems, this standard format is known as ELF (for Executable and Linkable Format).
  - There are separate ELF formats for relocatable and executable files.
  - One piece of information in the ELF file for executable files is the program's entry point, which contains the address of the first instruction to be executed when the program runs.
  - Windows systems use the Portable Executable (PE) format, and macOS uses the Mach-O format.
- Linux provides various commands to identify and evaluate ELF files.
  - For example, the file command determines a file type.
  - If main.o is an object file, and main is an executable file, the command
    file main.o
    will report that main.o is an ELF relocatable file, while the command
    file main
    will report that main is an ELF executable.
  - ELF files are divided into a number of sections and can be evaluated using the readelf command.