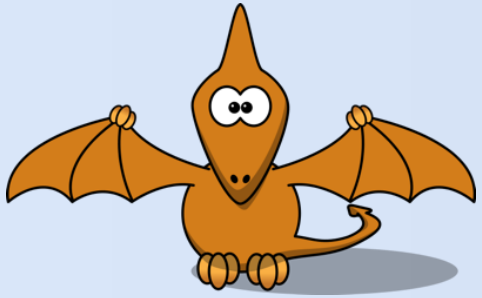
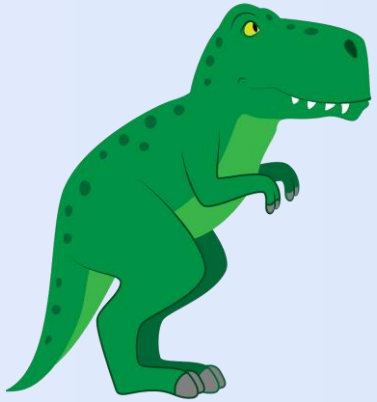


Chapter 1-2.

Introduction & O/S Structures



Operating System Concepts (10th Ed.)

01. 운영체제가 뭐길래?

운영체제가 뭔가요?

- An operating system is
  - a software that operates a ***computer*** system.

01. 운영체제가 뭐길래?

그럼, 컴퓨터는 뭔가요?

- A computer is
  - a machine that processes the ***information***.

01. 운영체제가 뭐길래?

그런데, 도대체 정보가 뭔가요?

- An information can be defined as
  - a quantitative representation that measures the uncertainty.

정보란
불확실성을 측정해서 수치화 한 것

$$I(x) = -\log_2 P(x)$$

예시 : 주사위를 1번 굴려 눈의 수 3이 나왔다는 정보
-> 정보량 = - log (1/6) = log 6

예시 : 동전를 1번 굴려 앞면이 나왔다는 정보
-> 정보량 = - log (1/2) = log 2 = 1 = 1 bit = 정보의 최소 단위

01. 운영체제가 뭐길래?

컴퓨터가 정보를 어떻게 처리하죠?

8 bit = 2^8 = 1 byte

- 정보의 최소 단위: bit(*binary digit*)
- 정보의 처리: 정보의 상태 변환 (0에서 1로, 1에서 0으로)
- 부울 대수(Boolean Algebra): NOT, AND, OR
- 논리 게이트: NOT, AND, OR, XOR, NAND, NOR
- 논리 회로: IC, LSI, VLSI, ULSI, SoC, ...
  - 무어의 법칙, 황의 법칙
- 정보의 저장과 전송: 플립-플롭, 데이터 버스

01. 운영체제가 뭐길래?

- 그래서, 컴퓨터가 정보를 어떻게 처리하죠?
  - 덧셈은? 반가산기, 전가산기
  - 뺄셈은? 2의 보수 표현법
  - 곱셈과 나눗셈은? 덧셈과 뺄셈의 반복
  - 실수 연산은? 부동 소수점 표현법
  - 함수는? GOTO
  - 삼각함수, 미분, 적분, 사진 촬영, 동영상 재생, ……

01. 운영체제가 뭐길래?

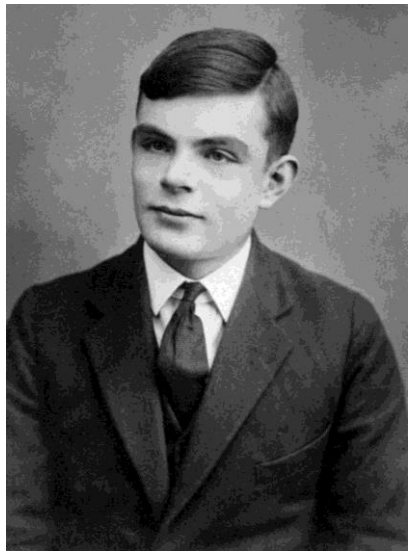
컴퓨터가 만능이라는 건가요?

- **범용성**: universality
  - NOT, AND, OR **게이트만으로 모든 계산을 할 수 있다.**
  - NAND **게이트만으로 모든 계산을 할 수 있다.**
  - **범용 컴퓨터**: general-purpose computer

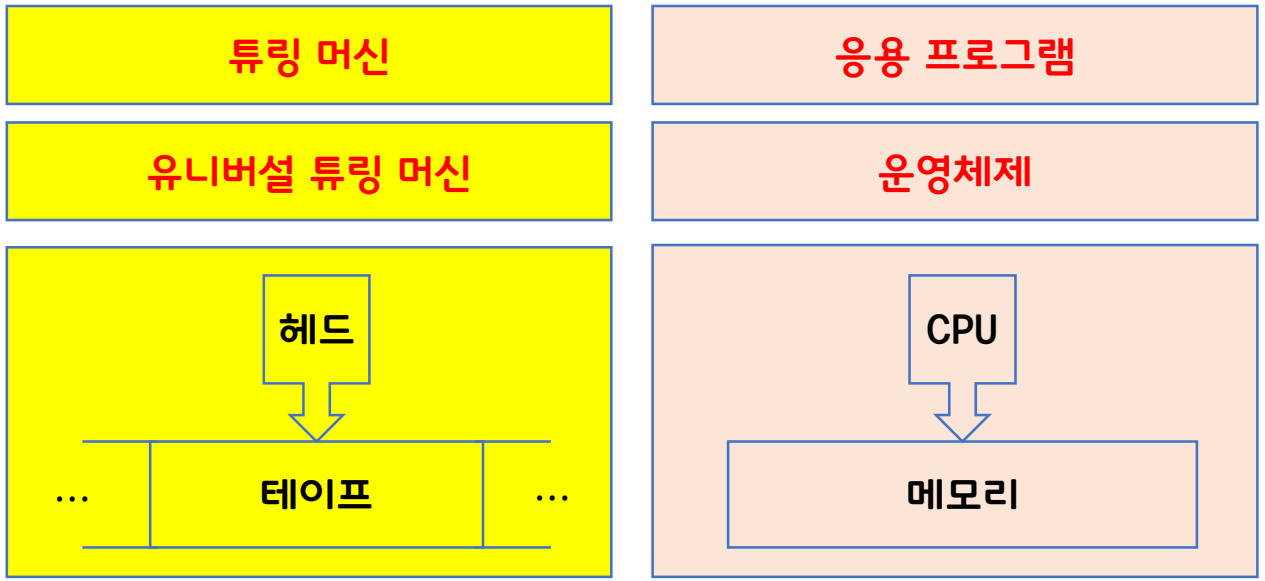
- **계산가능성**: computability
  - Turing-computable: **튜링 머신으로 계산가능한 것.**
  - **정지 문제**: Halting Problem: **튜링 머신으로 풀 수 없는 문제.**

01. 운영체제가 뭐길래?

- **컴퓨터는 누가 만들었어요?**
  - **컴퓨터의 할아버지**
    - Alan Turing – Turing Machine
  - **컴퓨터의 아버지**
    - John von Neumann – ISA: Instruction Set Architecture



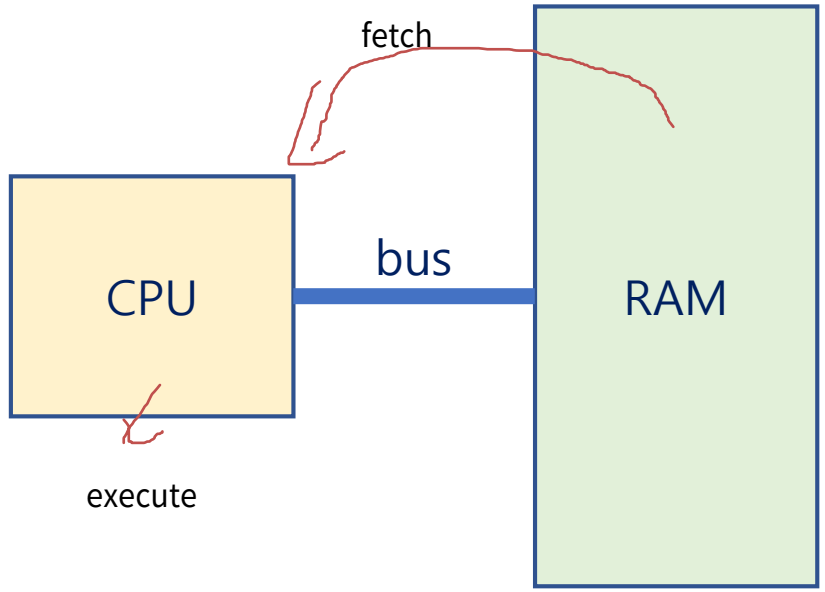
- **앨런 튜링이 왜 컴퓨터의 할아버지인가요?**
  - Head, Tape, Turing Machines, *Universal Turing Machine*
  - CPU, RAM, Application Programs, *Operating System*
  - Turing Machine: **주니온TV 〈TMItalk: 튜링 머신의 이해〉 참조.**



01. 운영체제가 뭐길래?

- **폰 노이만이 왜 컴퓨터의 아버지인가요?**
  - A **stored-program** computer is
    - a computer that stores programs in a memory.

메모리에 프로그램을 저장하는 컴퓨터



01. 운영체제가 뭐길래?

프로그램이 뭔데요?

- A program is a set of *instructions*

프로그램 = 명령어들의 집합

  - that tells a computer's hardware to perform a task.

기계어---------> 어셈블리어

```c
#include <stdio.h>

int main()
{
    int a = 3, b = 7;
    int c = a + b;
    printf("%d\n", c);
    return 0;
}
```

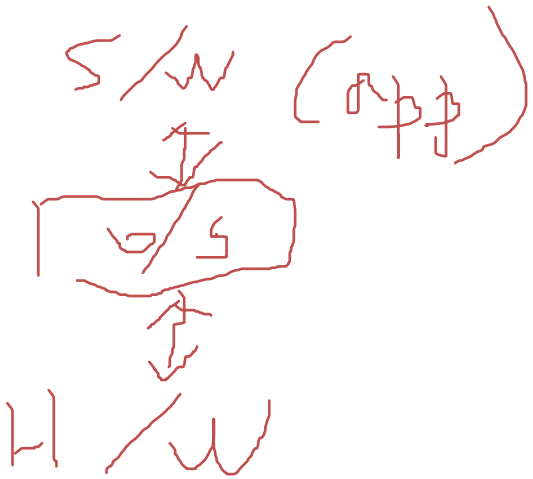
```
.....
movl    $3, -12(%rbp)
movl    $7, -8(%rbp)
movl    -12(%rbp), %edx
movl    -8(%rbp), %eax
addl    %edx, %eax
movl    %eax, -4(%rbp)
movl    -4(%rbp), %eax
movl    %eax, %esi
leaq    .LC0(%rip), %rdi
movl    $0, %eax
call    printf@PLT
movl    $0, %eax
.....
```

01. 운영체제가 뭐길래?

운영체제도 프로그램인가요?

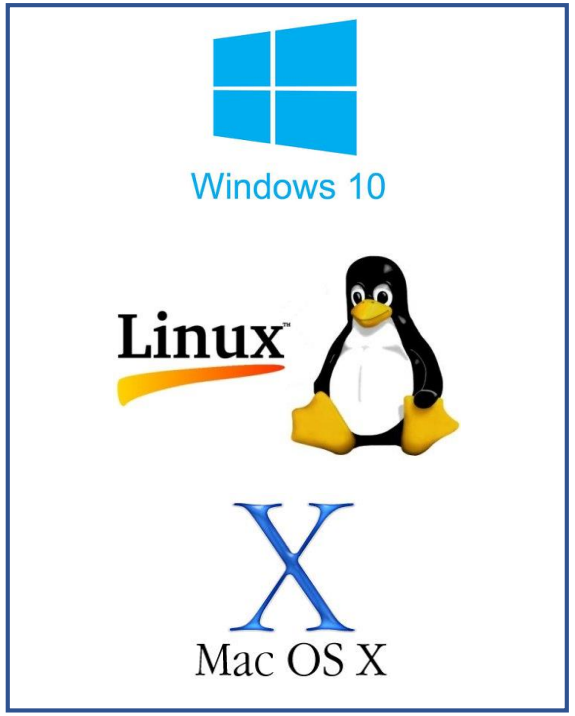
항상 실행 중인 컴퓨터 (유니버셜 튜링 머신)

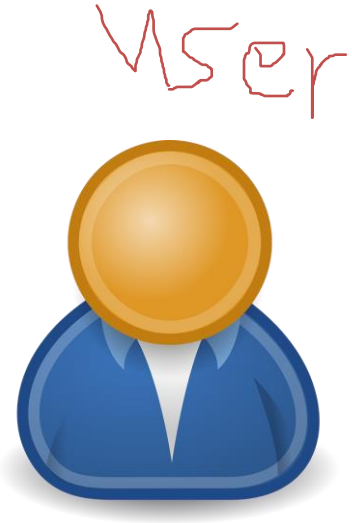
- Operating system
  - is a program *running at all times* on the computer
  - to provide *system services* to application programs
  - to manage *processes*, *resources*, *user interfaces*, and so on.



01. 운영체제가 뭐길래?

- **운영체제가 뭔가요?**
  - An operating system is
    - a software that operates a ***computer*** system.





operating system

01. 운영체제가 뭐길래?

- 다음 중 컴퓨터가 아닌 것은?
  - 휴대폰
  - 텔레비전
  - 자동차
  - 지갑

- 다음 중 운영체제가 필요없는 것은?
  - 휴대폰 → OS: Android, IOS
  - 텔레비전 → 스마트TV
  - ~~자동차~~
  - 지갑 → 정답!!

1.1 What Operating Systems Do

- An operating system is
  - a software that manages a computer's hardware.
  - It also provides a basis for application programs and
    - acts as an intermediary between
    - the computer user and the computer hardware.

1.1 What Operating Systems Do

- A computer system can be divided roughly into four components:
  - the hardware,
  - the operating system,
  - the application programs,
  - and a user

1.1 What Operating Systems Do

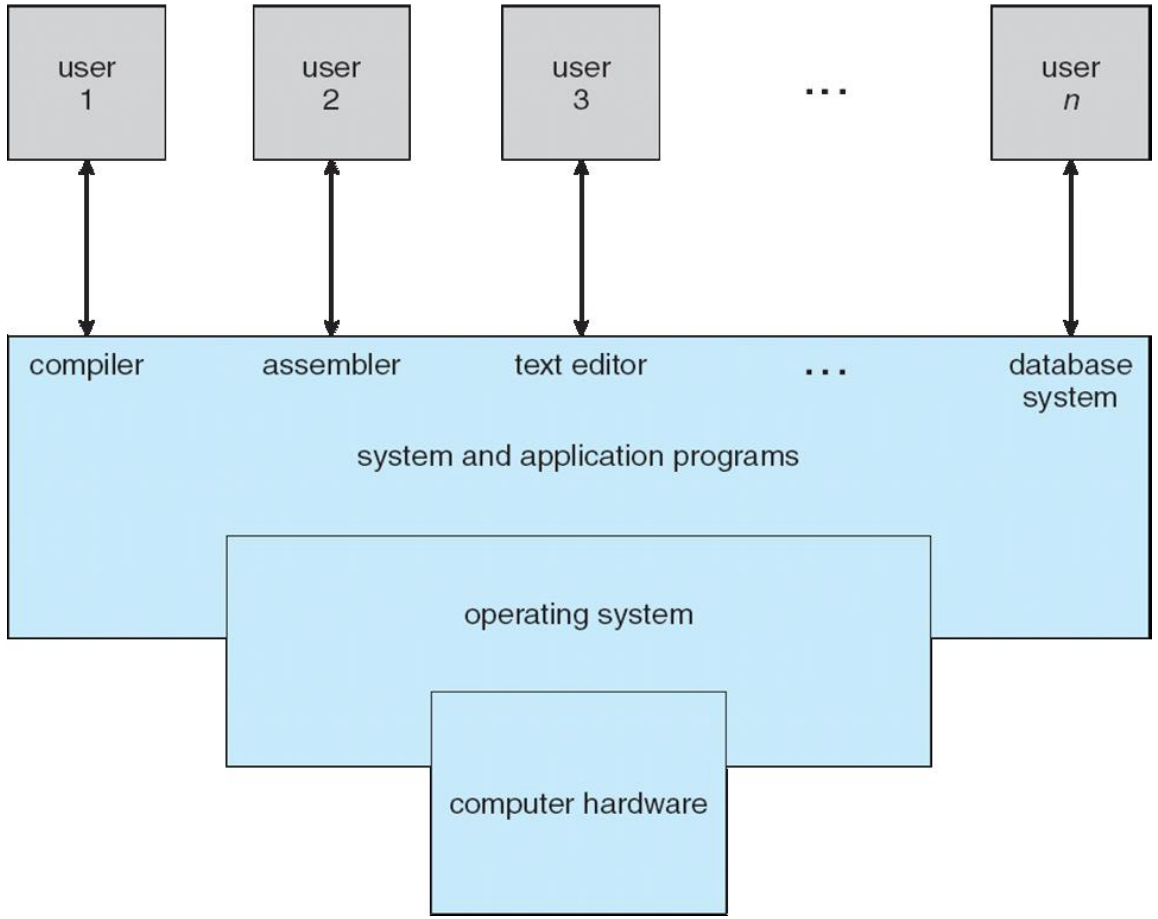


Figure 1.1 Abstract view of the components of a computer system.

1.1 What Operating Systems Do

- Defining Operating Systems
  - There are *NO* universally accepted definition of an operating system.
  - A more common definition is that
    - "*the one program running at all times on the computer*" 항상 작동중인 유일한 프로그램
    - usually called the *kernel*.
  - Along with the kernel, there are two other types of programs:
    - *system* programs
    - *application* programs

1.2 Computer-System Organization

- A *modern* computer system consist of:
  - one or more CPUs and
  - a number of device controllers connected through a common *bus*.

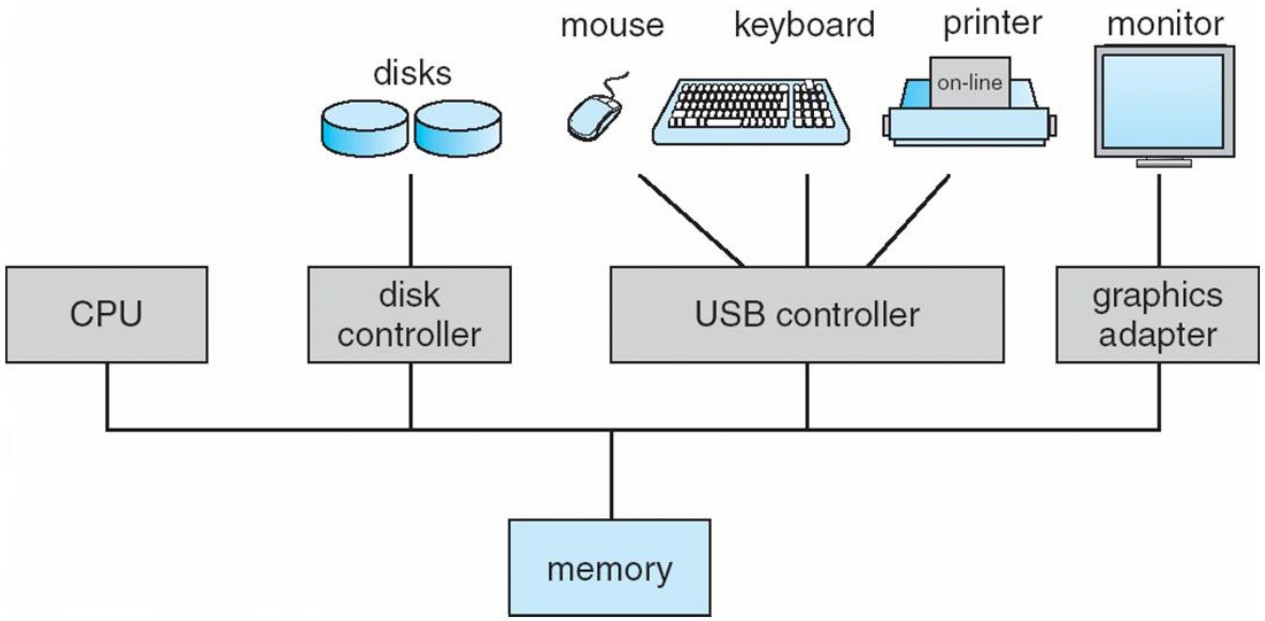


Figure 1.2 A typical PC computer system.

1.2 Computer-System Organization

- A *bootstrap* program is
  - the first program to run on computer power-on,
  - and then loads the operating system.

컴퓨터 시작 시 처음으로 실행되어 운영체제를 로드하는 프로그램

운영체제를 집어 넣어줌

1.2 Computer-System Organization

- Interrupts
  - Hardware may trigger an *interrupt* at any time
    - by sending a signal to the CPU, usually by way of the system bus.

하드웨어가 interrupt를 트리거하고
신호가 cpu에 보내짐

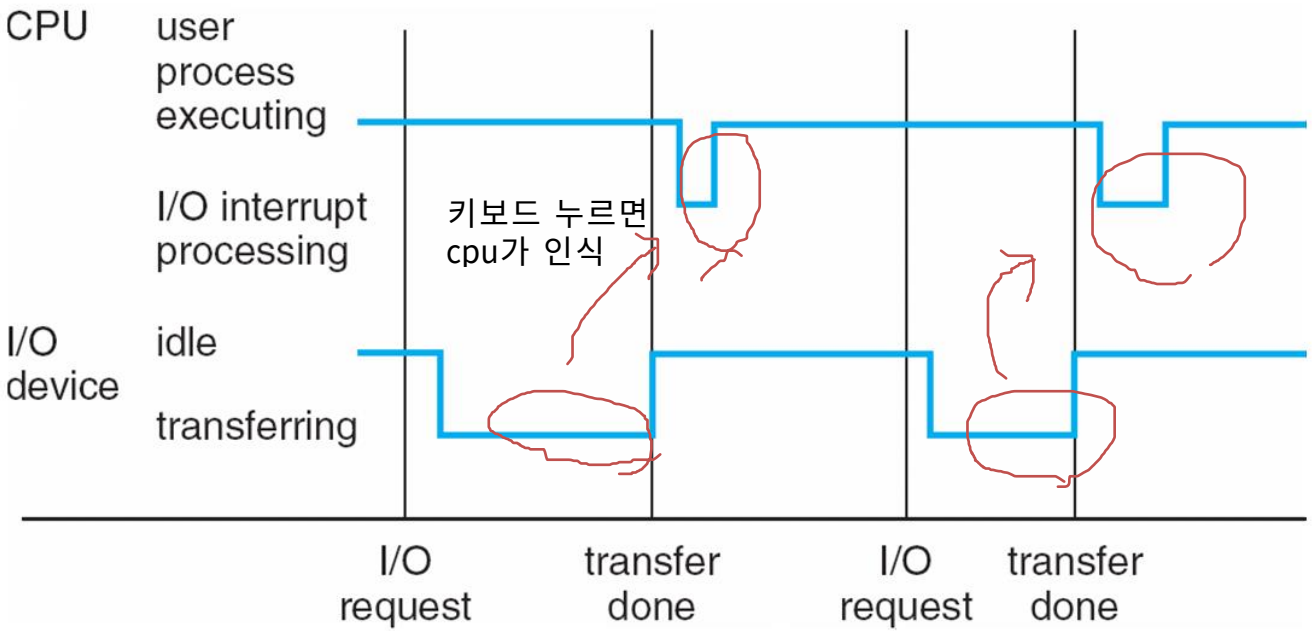
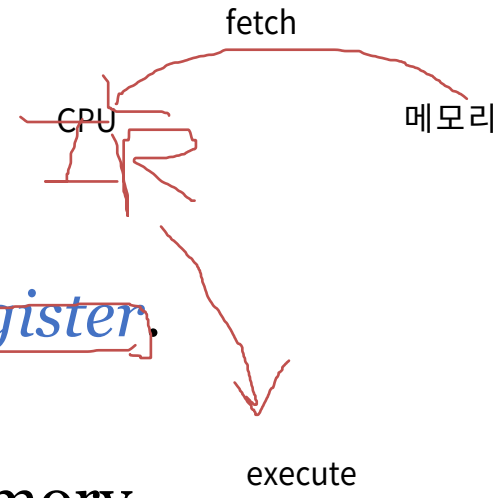


Figure 1.3 Interrupt timeline for a single program doing output.

1.2 Computer-System Organization

- *von Neumann architecture*
  - A typical instruction-execution cycle
    - first *fetches* an instruction from memory
    - and stores that instruction in the *instruction register*.
  - The instruction is then decoded
    - and may cause *operands* to be fetched from memory
    - and stored in some internal register.
  - After the instruction on the operands
    - has been executed,
    - the result may be stored back in memory.



1.2 Computer-System Organization

- The wide variety of storage systems can be organized in a hierarchy according to:
  - storage capacity,
  - and access time

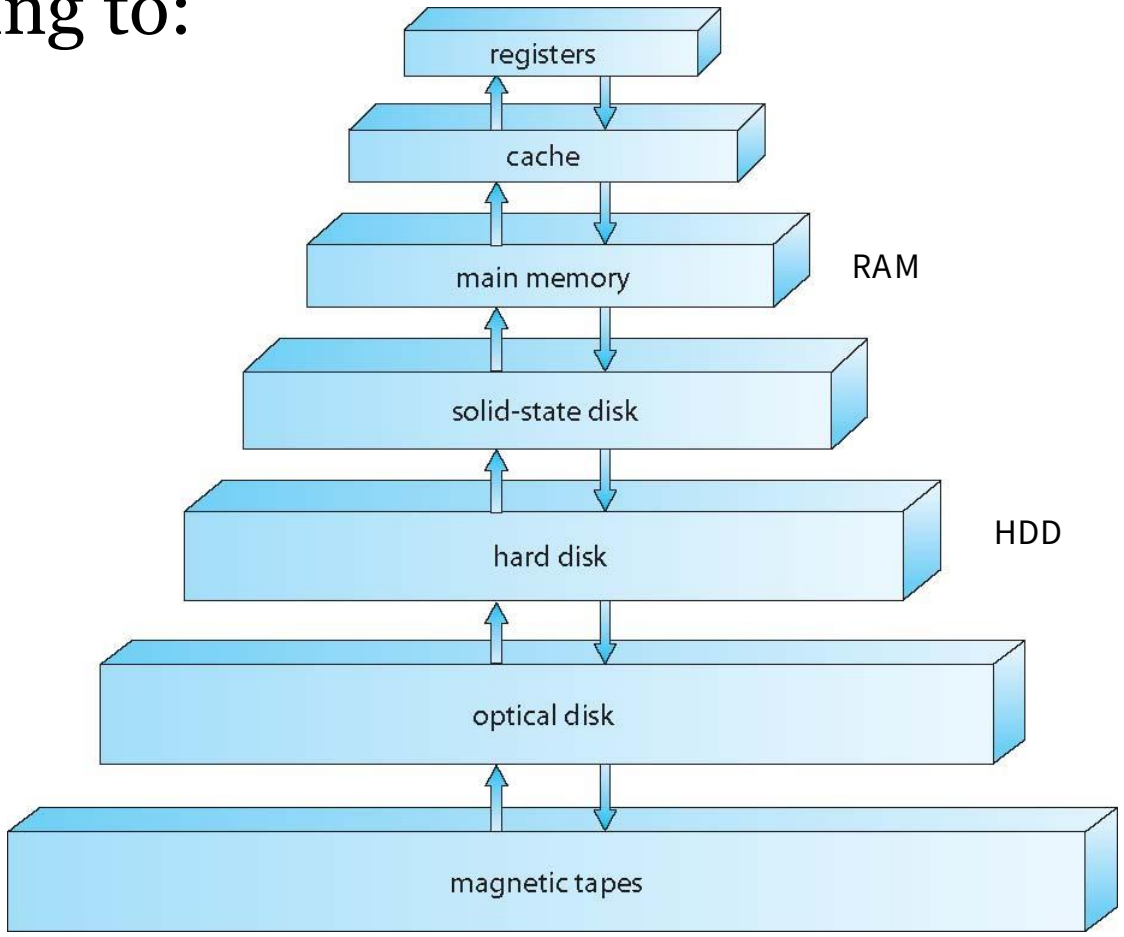
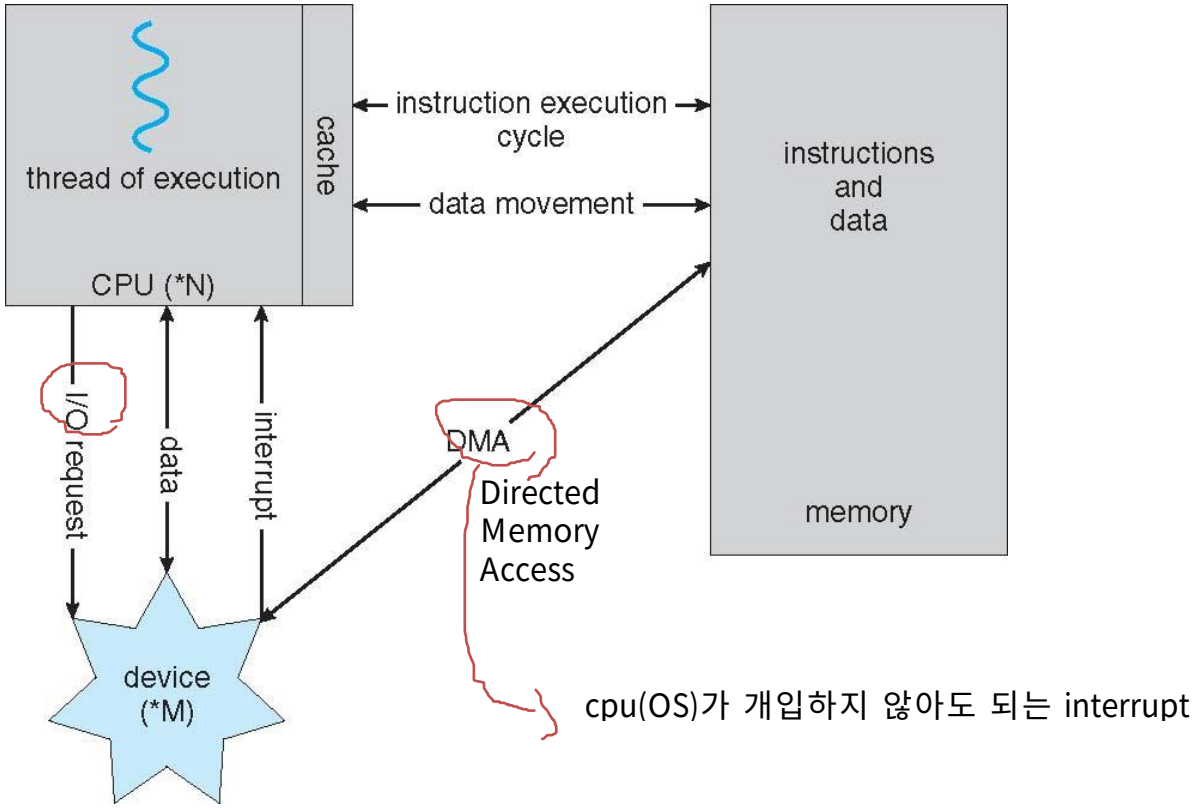


Figure 1.6 *Storage-device hierarchy.*

1.2 Computer-System Organization

- I/O Structure
  - A large portion of OS code is dedicated to managing I/O



입출력
input/output

OS의 기능은
주로 device의 input/output 관리에 집중되어 있음

cpu(OS)가 개입하지 않아도 되는 interrupt

Figure 1.7 How a modern computer system works.

1.3 Computer System Architecture

- Definitions of Computer System Components 컴퓨터 시스템 구조
  - CPU - The hardware that executes instructions.
  - Processor - A physical chip that contains one or more CPUs.
  - Core - The back computation unit of the CPU.
  - Multicore - Including multiple computing cores on the same CPU.
  - Multiprocessor - Including multiple processors.

1.3 Computer System Architecture

- *Symmetric multiprocessing* (SMP)
  - The most common multiprocessor systems,
    - in which each peer CPU processor performs all tasks.
  - *Asymmetric* multiprocessing:
    - each processor is assigned a specific task.

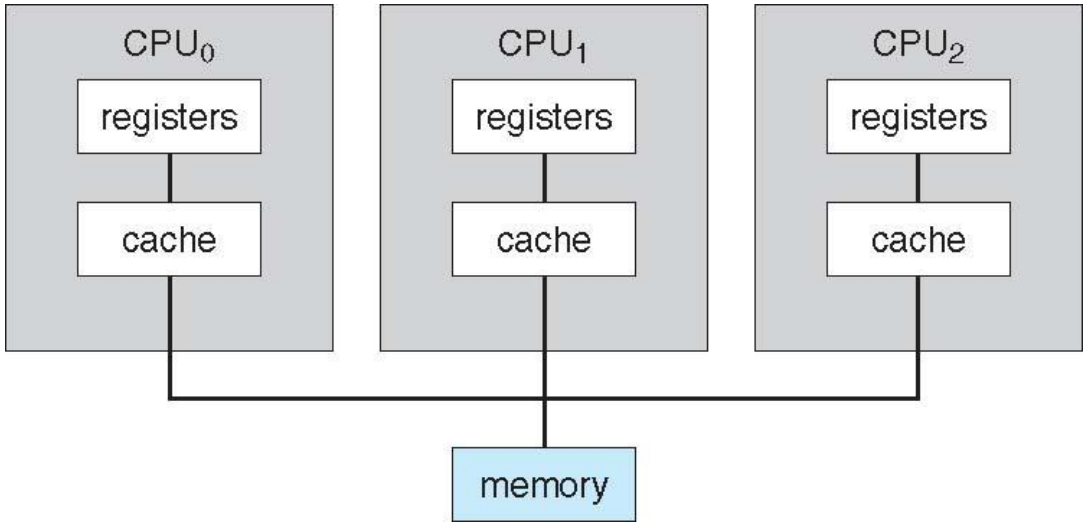


Figure 1.8 *Symmetric multiprocessing architecture.*

1.3 Computer System Architecture

- Multi-core design
  - with several cores on the same processor chip.

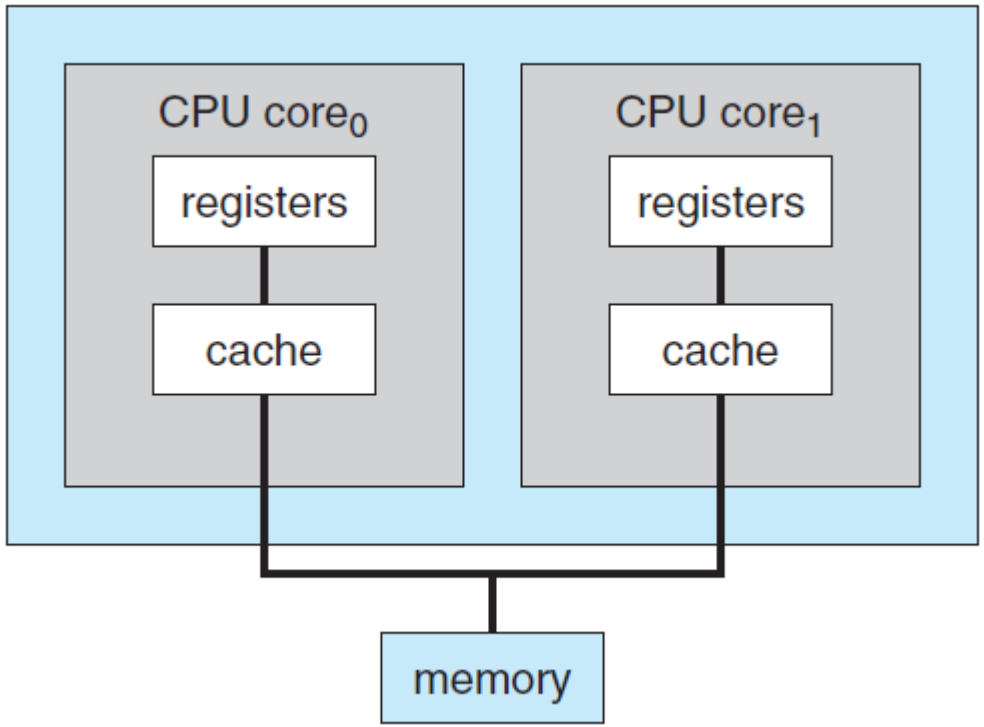


Figure 1.9 *A dual-core design with two cores on the same chip.* `

1.4 Operating System Operations

- Multiprogramming
  - runs more than one program at a time.
  - keeps several processes in memory simultaneously.
  - to increase CPU utilization.

cpu 사용률을 높일 수 있다.

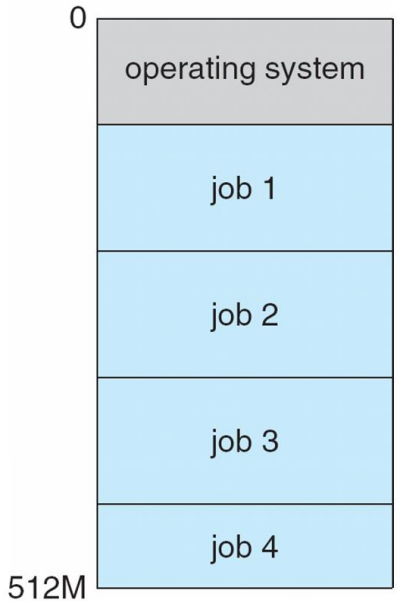


Figure 1.12 *Memory layout for a multiprogramming system.*

1.4 Operating System Operations

- Multitasking (=multiprocessing) = concurrency -> pararellism과 비교 (3장)
  - a logical extension of multiprogramming.
    - in which CPU switches jobs so frequently that
    - users can interact with each job while it is running.
  - CPU scheduling: 어떤 프로세스를 선택해야 cpu사용률을 가장 높일 수 있는가 (5장)
    - If several processes are ready to run at the same time,
    - the system must choose which process will run next

1.4 Operating System Operations

- Two separate mode of operations:
  - *user mode* and *kernel mode*
  - to ensure that an incorrect program
    - cannot cause other programs to execute incorrectly

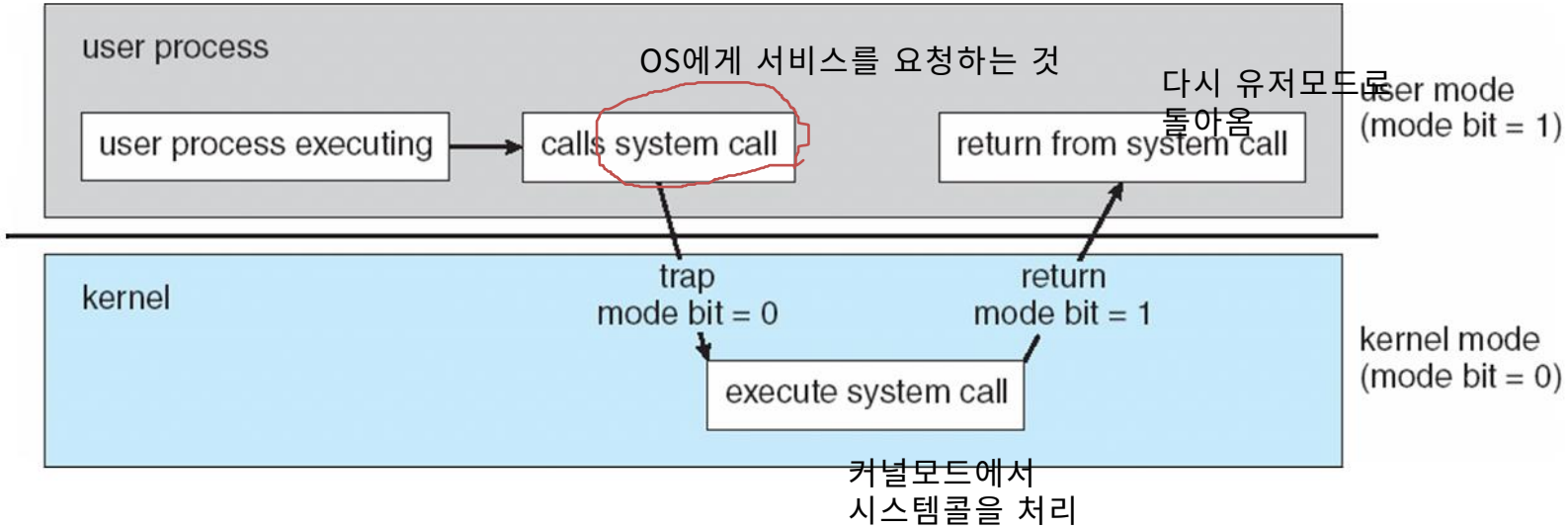


Figure 1.13 *Transition from user to kernel mode.*

1.7 Virtualization

- Virtualization is
  - a technology that allow us
    - to abstract the hardware of a single computer
    - into several different execution environments.
  - VMM: Virtual Machine Manager
    - VMware, XEN, WSL, and so on.

1.7 Virtualization

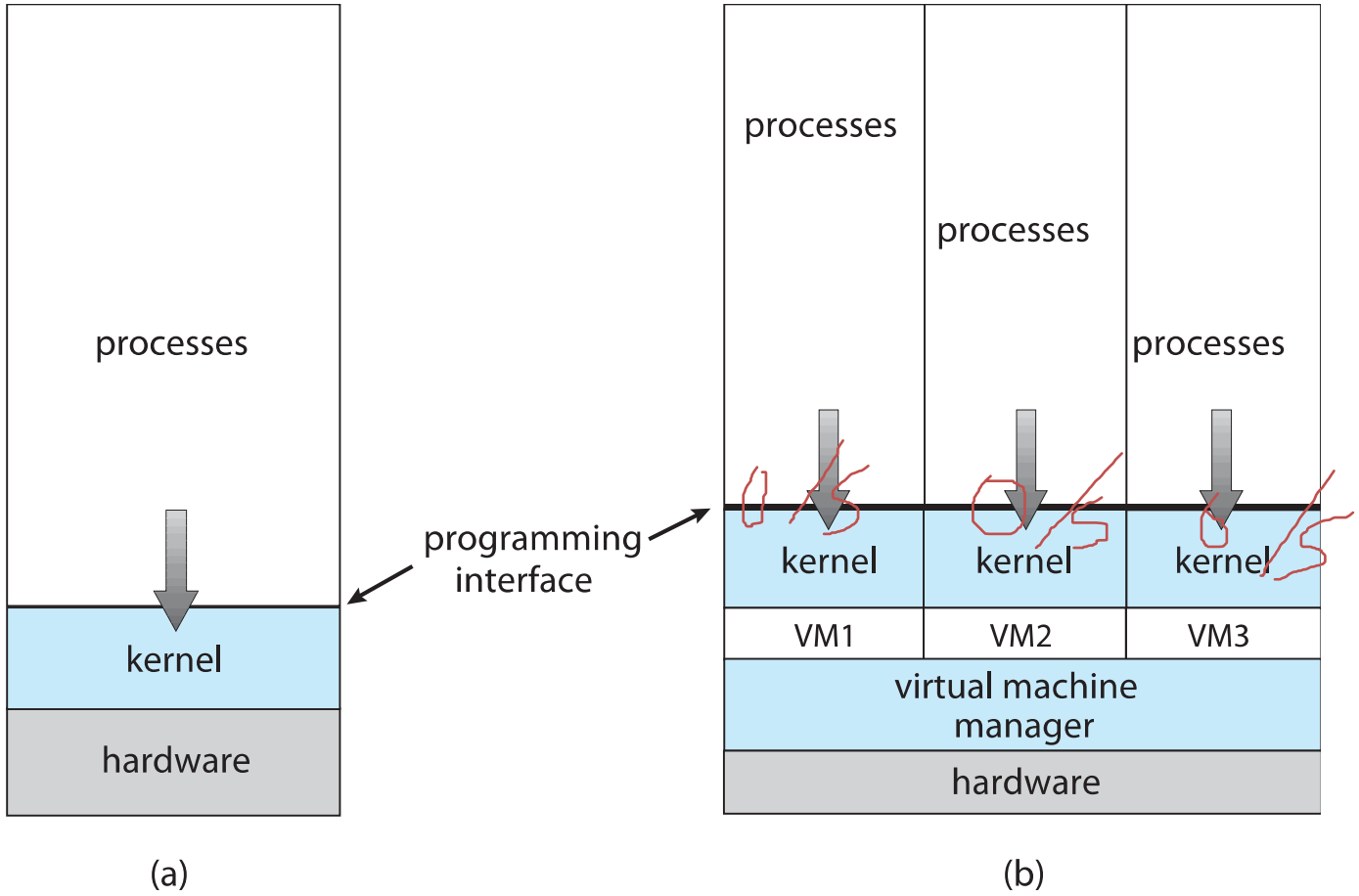


Figure 1.16 *A computer running (a) a single operating system and (b) three virtual machines.*

1.10 Computing Environments

- Operating Systems in the Variety of Computing Environments
  - Traditional Computing
  - *Mobile* Computing
  - *Client-Server* Computing
  - *Peer-to-Peer* Computing P to P
  - *Cloud* Computing AWS, Azure, Google Cloud Platform
  - *Real-Time* Embedded Systems

1.10 Computing Environments

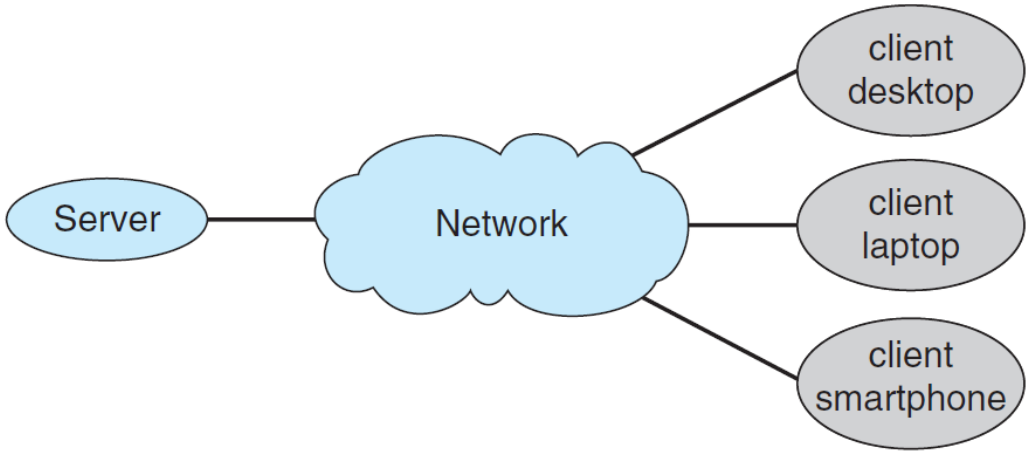


Figure 1.22
General structure of a client-server system.

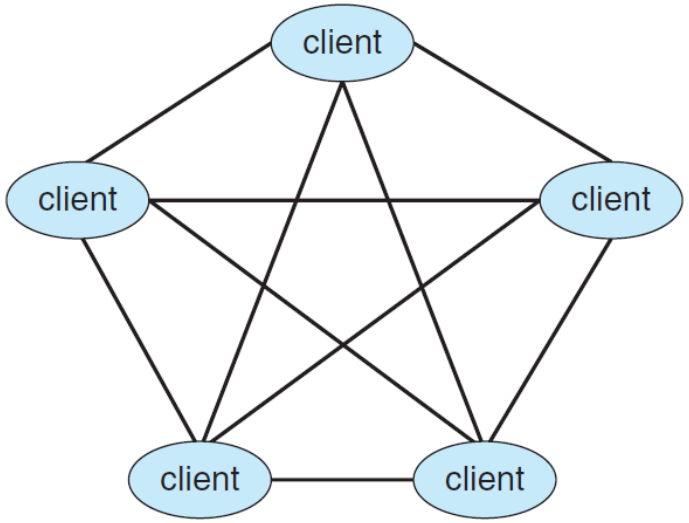


Figure 1.23
Peer-to-peer system with no centralized service.

1.10 Computing Environments

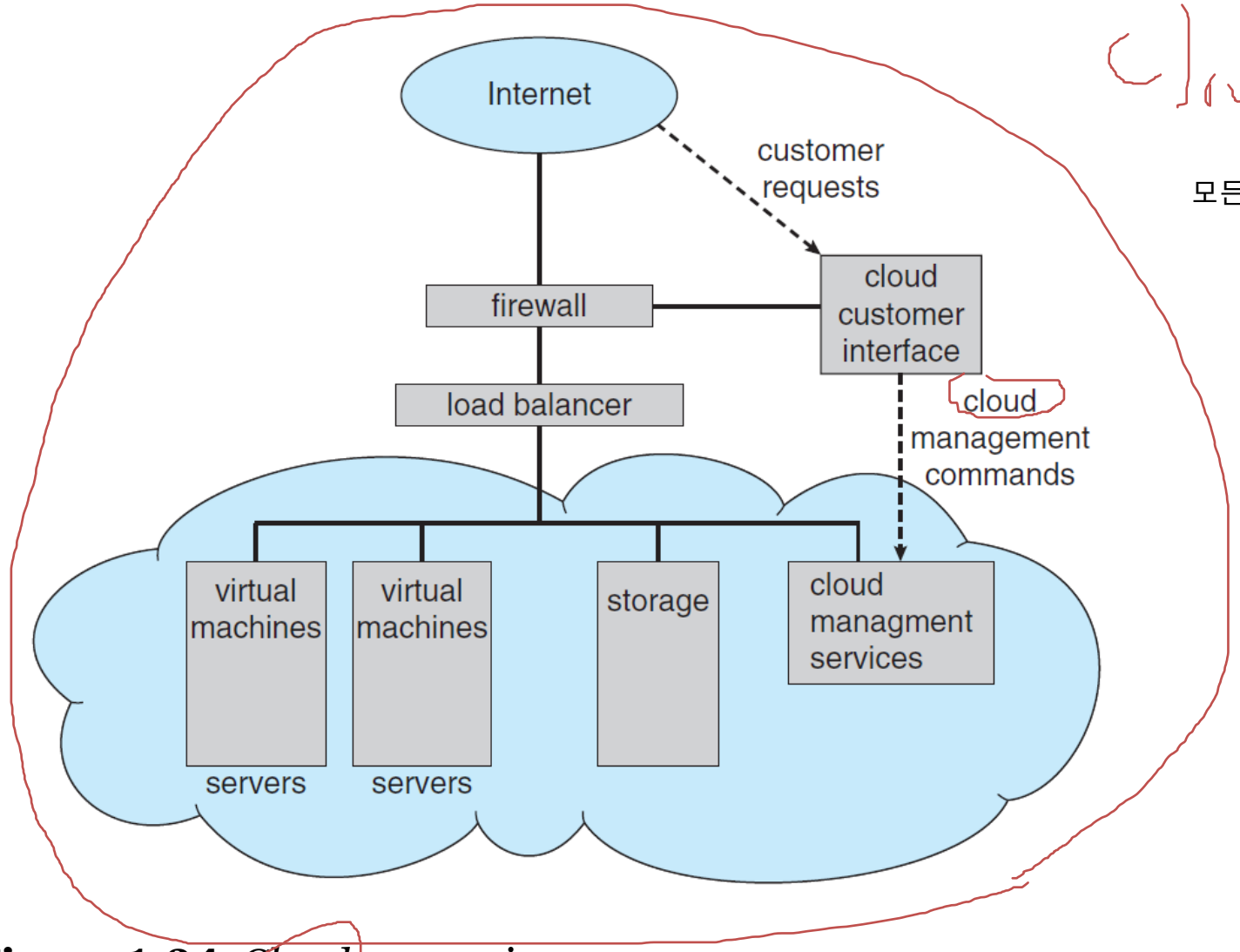


Figure 1.24 *Cloud computing.*

모든 자원이 cloud안에 가려진 구조

2.1 Operating System Services

- OS provides an *environment* for the execution of programs.

  - User interface
  - Program execution
  - I/O operation
  - File-system manipulation
  - Communications
  - Error detection
  - Resource allocation
  - Logging
  - Protection and security

Process와 Thread의 개념

+ multiprocessing + cpu processing

+ synchronization

+ deadlock

2.1 Operating System Services

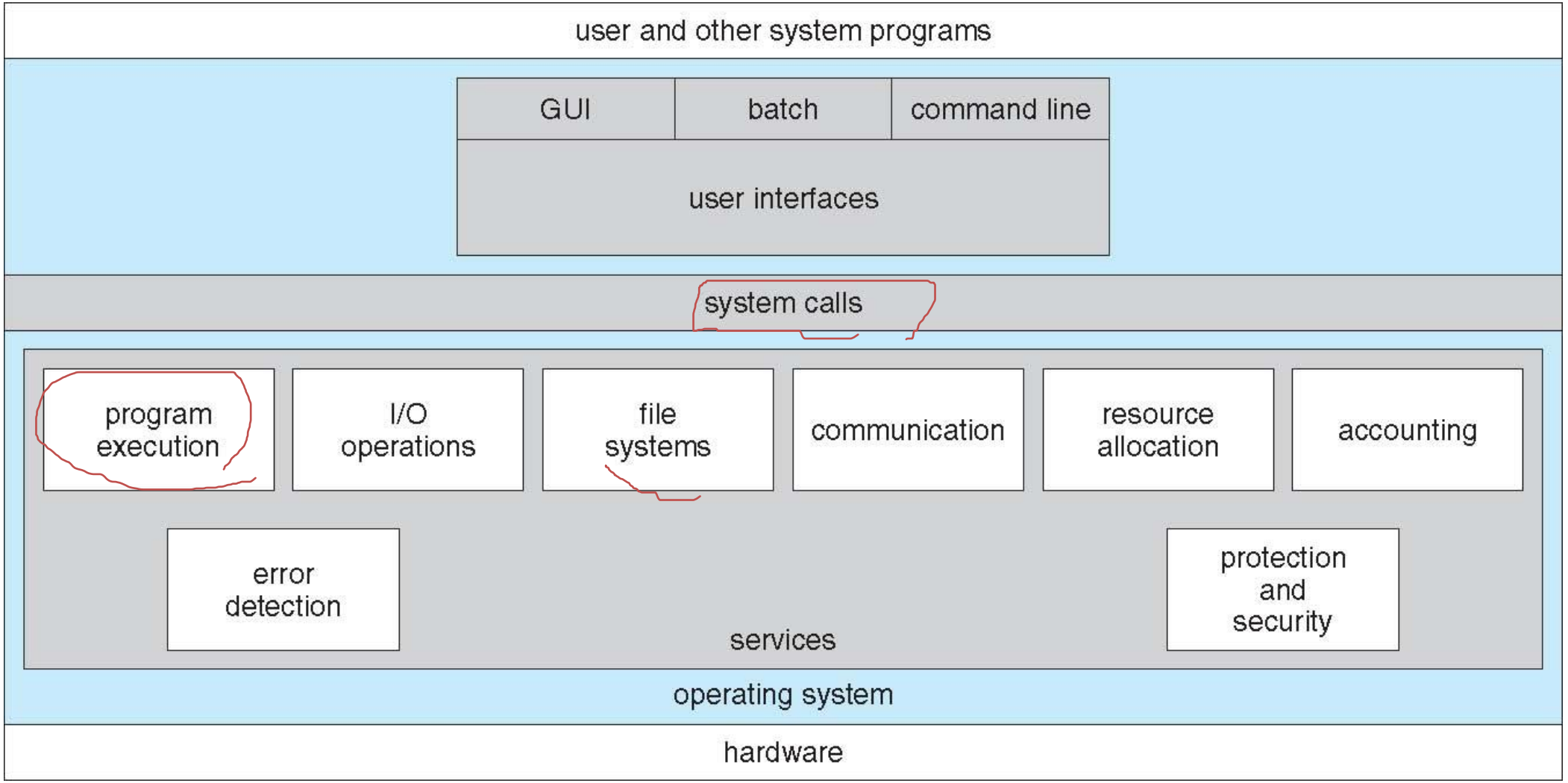


Figure 2.1 *A view of operating system services.*

2.2 User and Operating-System Interface

- Three fundamental ways for users to interface with the OS:
  - CLI: command line interface, or command interpreter
    - known as *shells*: sh, bash, csh, tcsh, zsh, etc.
  - GUI: graphical user interface
    - Windows, Aqua for MacOS, KDE/GNOME for Linux, etc.
  - Touch-Screen Interface
    - Android UI, iPhone UI, etc.

2.3 System Calls

- System calls
  - provide an interface to the services made available by the OS.
  - API: Application Programming Interface

```
#include <unistd.h>

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

return value — function name — parameters

2.3 System Calls

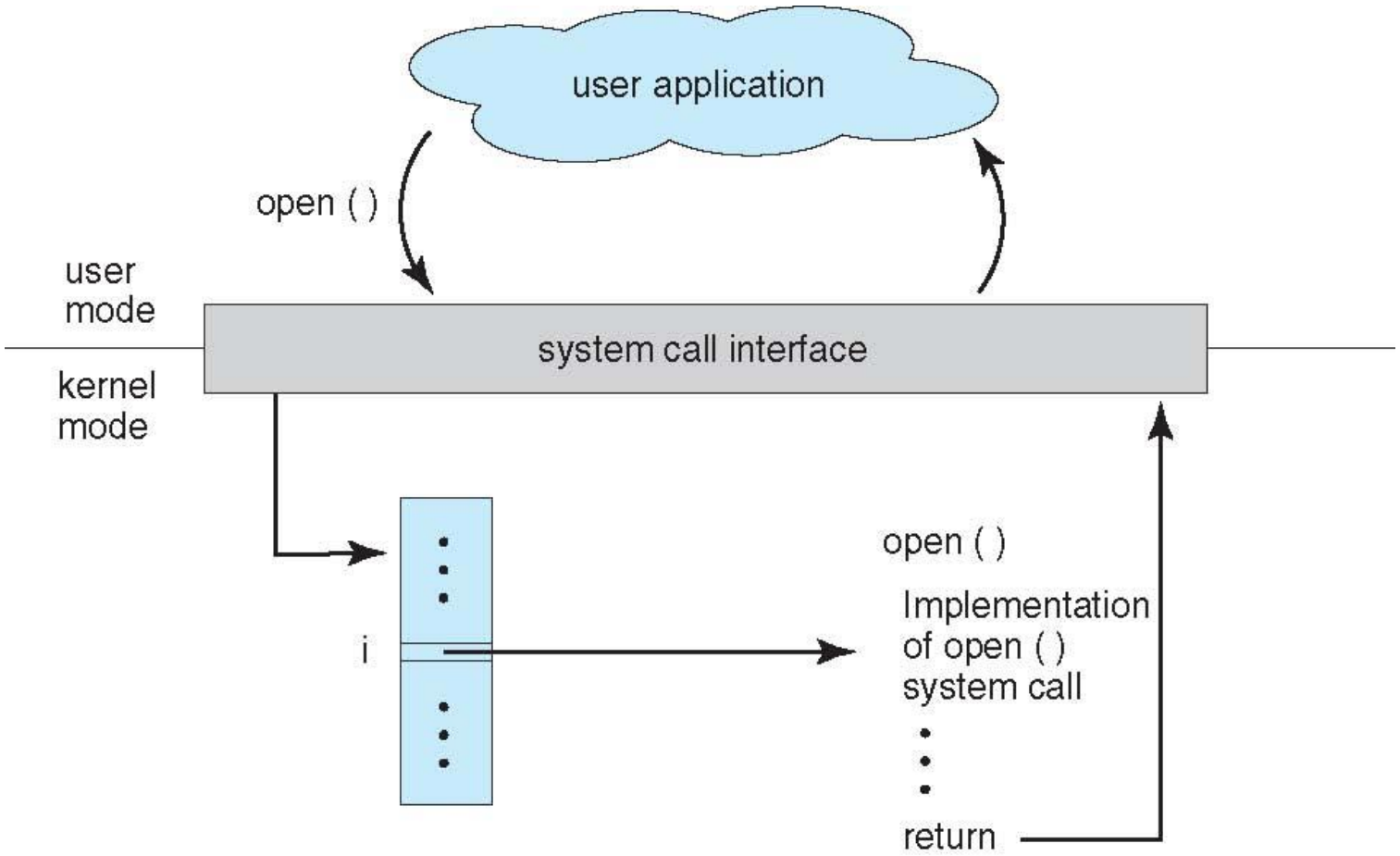


Figure 2.6 *The handling of a user application invoking the* `open()` *system call.*

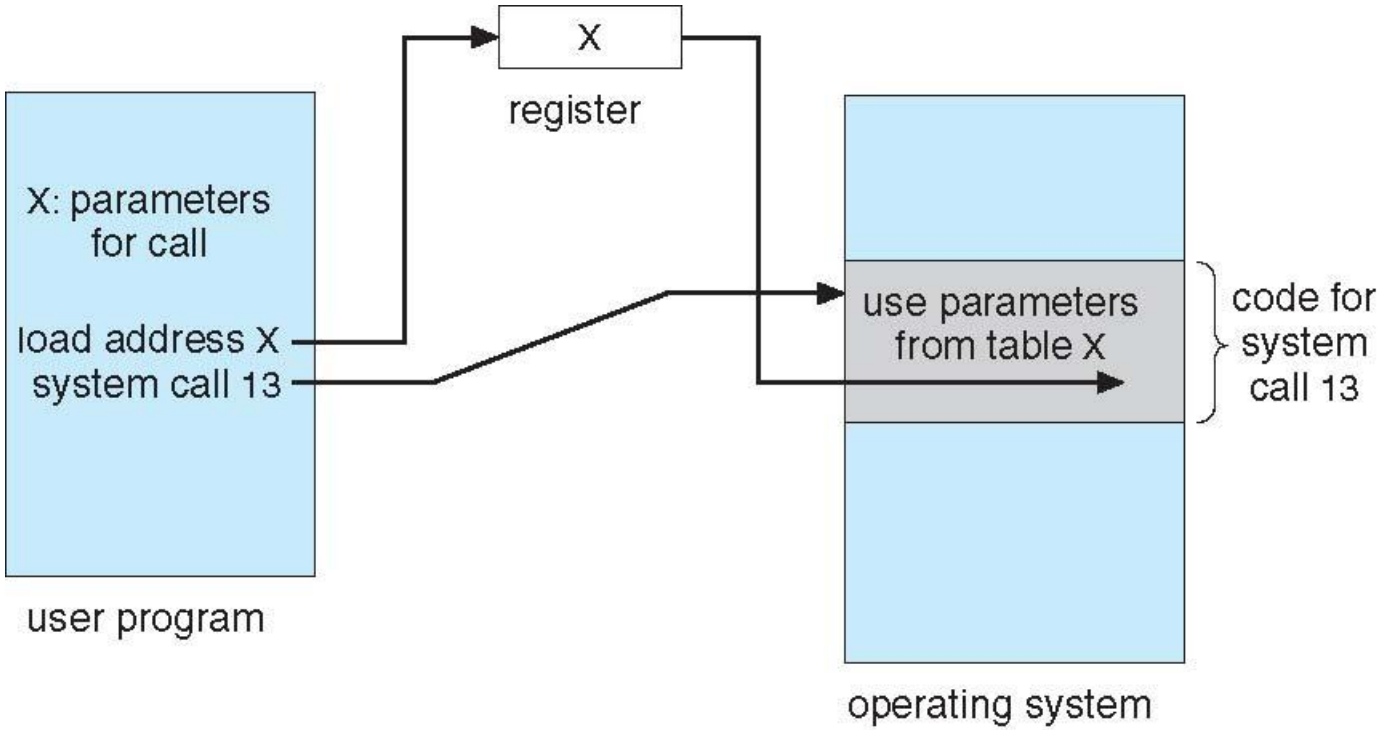


Figure 2.7 *Passing of parameters as a table.*

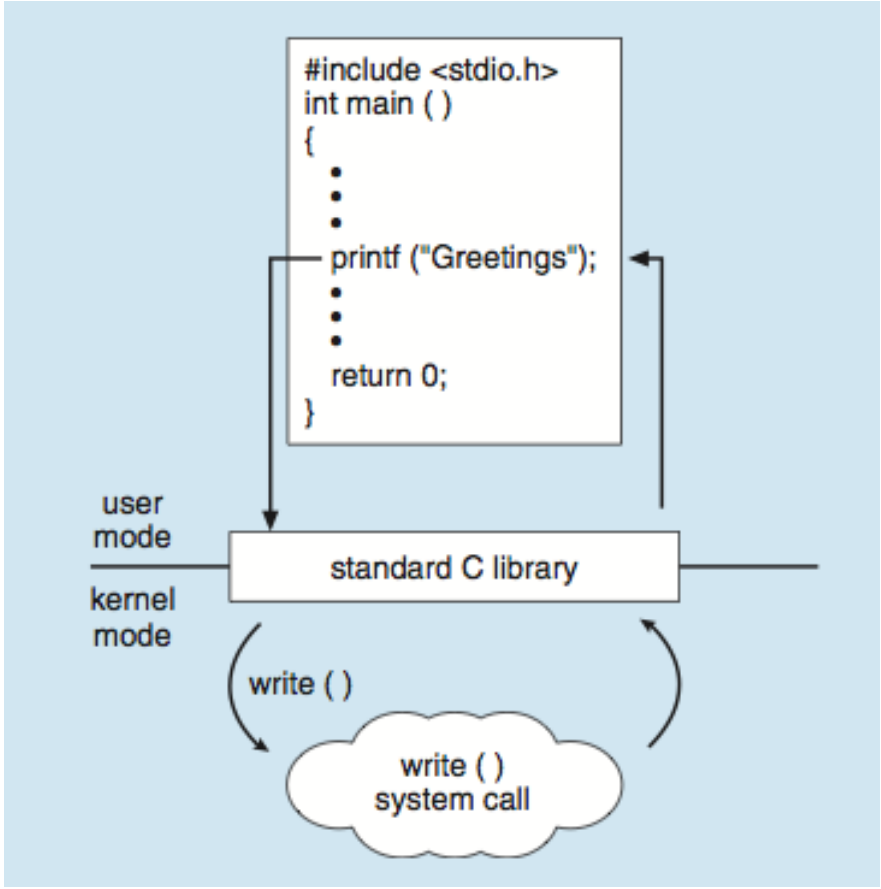
2.3 System Calls

- Examples of Windows and UNIX system calls:

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

2.3 System Calls

- The standard C library
  - provides a portion of the system-call interface.



라이브러리를 사용해서
시스템콜을 쉽게 할 수 있음.

Any Questions?