Chapter 1: Introduction

Chien Chin Chen

Department of Information Management
National Taiwan University

What is an Operating System? (1/2)

- A ***program*** that acts as an intermediary between a ***user*** of a computer and the ***computer hardware***.
- ***Various*** operating system goals:
  - ***Mainframe*** operating systems: to optimize utilization of hardware.
  - ***PC*** operating systems: to support complex games, business applications …
  - ***Handheld computers***: to help users easily interface with the computer to execute programs.

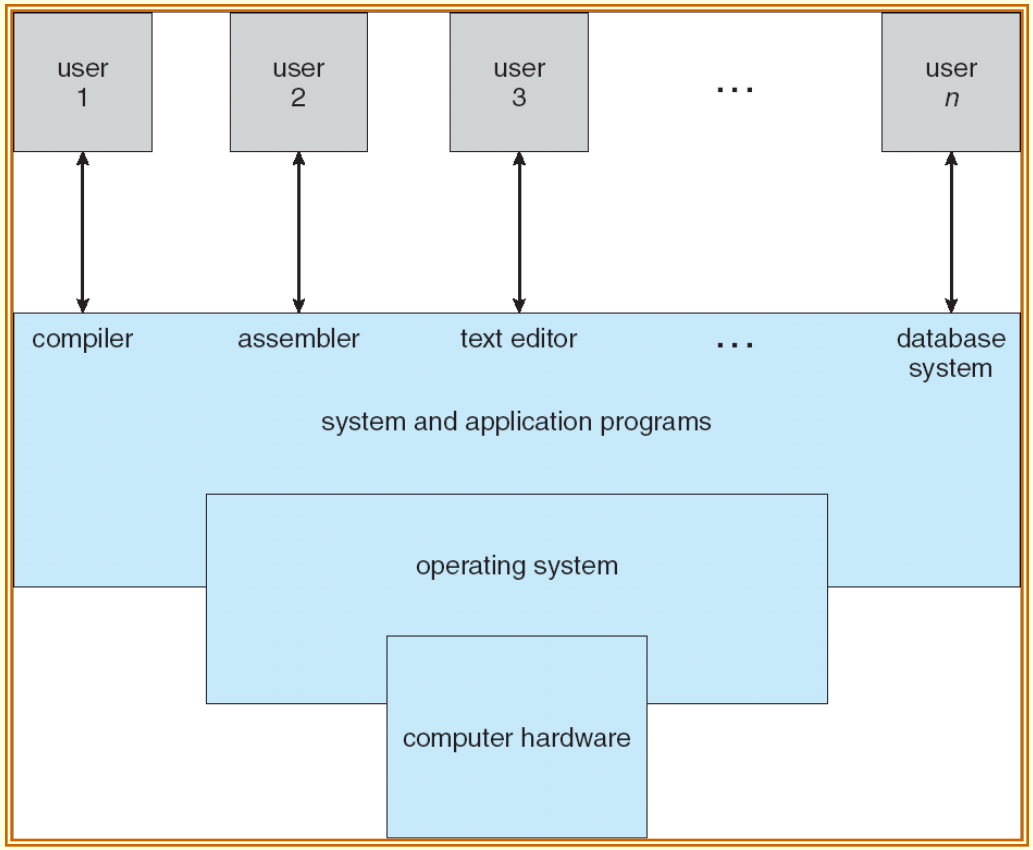
資源妥善利用
一個terminal，很多user

What is an Operating System? (2/2)

- Some operating systems are designed to be ***convenient***, others to be ***efficient***, and others some ***combination of the two***.

Four Components of a Computer System (1/2)

- Computer system can be divided into four components: ***hardware***, the ***operating system***, the ***application programs***, and the ***users***.



Four Components of a Computer System (2/2)

- **Hardware** – provides basic **computing resources**.
  - CPU, memory, I/O devices.
- **Operating system** – **controls** and **coordinates** use of hardware among various applications and users
- **Application programs** – define the ways in which the system resources are used **to solve the computing problems of the users**.
  - Word processors, compilers, web browsers, database systems, video games.
- **Users** – people, machines, other computers.

An operation system is similar to a ***government***. It provides an environment within which other (user) programs can do useful work.

Viewpoints From Users (1/2)

- **PC**: the OS is designed for ***one user only***.
  - Resources are monopolized.
  - The goal is to maximize the work of the user.
  - The OS is generally designed for ***ease of use***, with some attention paid to performance and non paid to resource utilization.
- **Mainframe**: the OS is designed for ***multiple users*** – accessing the same computer through terminals.
  - These users share resources.
  - The OS is designed to ***maximize resource utilization*** – to assure that all available CPU time, memory …

Viewpoints From Users (2/2)

- **Workstation**:
  - User**s** sit at workstation**s** connected to networks of other workstations and servers (file, compute, and print servers).
  - The OS is designed to compromise between individual usability and resource utilization.
- **Handheld computer**: are **standalone** units for individual users.
  - The OS is designed mostly for individual usability.
  - But performance per amount of battery life is important as well.
- **Computer with little (or no) user view**: embedded home devices.
  - The OS is designed to run without user intervention.

Viewpoints From Computers

- OS for computer is the program involved with the hardware.
- OS is a ***resource allocator***.
  - Manages all resources.
  - Decides between **conflicting requests** for efficient and **fair** resource use.
- OS is a ***control program***.
  - Controls execution of programs to prevent errors and improper use of the computer.

Operating System Definition (1/3)

- What is an operating system?
  - No universally accepted definition.
- Bare computer (hardware) alone is not easy to use, so application programs are developed.
  - These programs require certain ***common operations***, such as those controlling the I/O device.
  - These common functions of ***controlling*** and ***allocating*** resources are then brought together into one piece of software: the *operating system*.

Operating System Definition (2/3)

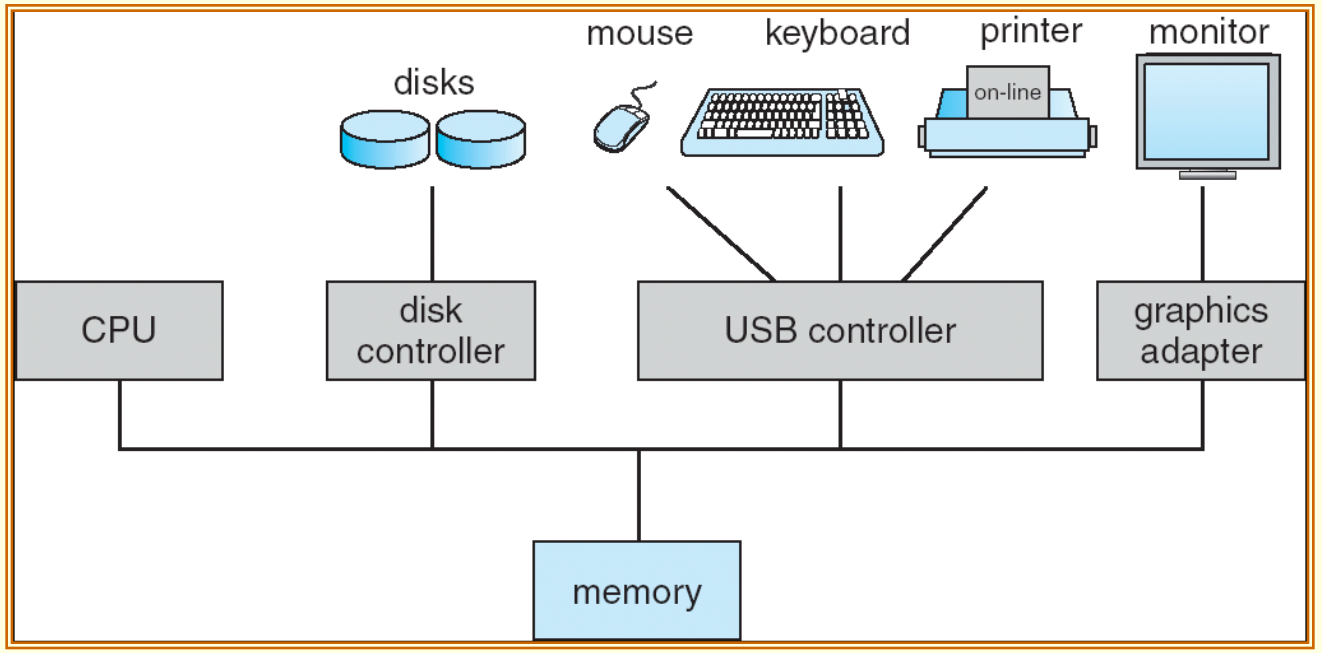
- OS is …
  - “Everything a vendor ships when you order an operating system”
    - But varies wildly, for example, text/graphic mode.
- A more common definition:
  - “The one program running at all times on the computer” is the ***kernel***. 關不掉的程式
  - Everything else is either a system program or an application program.

Operating System Definition (3/3)

- The matter of what constitutes an operating system has become increasingly important.
  - Antitrust of Microsoft windows.
  - Microsoft included too much functionality in its operating systems and thus prevented application vendors from competing.

Computer System Organization (1/2)

- Computer-system operation:
  - One or more ***CPUs***, ***device controllers*** connect through common ***bus*** that provides access to shared memory.
  - Each device controller is in charge of a specific type of ***device*** (disk drives, audio/video devices).



Computer System Organization (2/2)

- Each device controller has a **local buffer** and a set of special-purpose **registers**.
  - E.g., your SATA hard disk may contains 8M buffer.
- CPU moves data from/to main memory to/from local buffers.
- I/O devices and the CPU can execute **concurrently**.
- **Concurrent** I/O is from the device to local buffer of controller.

Computer System Operation (1/2)

- When a computer is **powered up** or **rebooted** …
  - A ***bootstrap program*** is loaded to Initialize all aspects of system, from CPU registers to device controllers to memory contents.
    - Typically stored in ROM (read-only memory) or EEPROM, (erasable programmable read-only memory) generally known as **firmware**.
    - **Load into memory the operating-system kernel.**
  - The operating system then starts executing the first process and waits for some **event** to occur.
  - **Events** are usually signaled by an ***interrupt*** from either the hardware or the software.
    - Hardware :I/O operations — disk drive access, keystroke, …
    - Software: system calls.

Computer System Operation (2/2)

- Why interrupt? — I/O operations ***without*** interrupt as an example.
  - The *handshaking* process of a host (a program) reads data through a port (device):
    1. The controller does the I/O to the device (hardware operations).
    2. The host ***repeatedly*** read the `busy` bit (of the device) until that bit becomes clear.
    3. The host reads data from the device controller.
    4. The I/O is finished.
  - In step 2, the host is ***busy-waiting*** or ***polling***.
    - It is in a *loop*, reading the `busy` bit over and over until the bit becomes clear.
    - The wait may be long, the host should probably switch to another task.
    - **Interrupt**: the *hardware mechanism* that enables a device to ***notify*** the CPU when it is ready for service.

Interrupt Mechanism (1/3)

- The CPU hardware has a wire called the ***interrupt-request line*** that the CPU senses after executing every instruction.
- When the CPU detects that a controller has asserted a signal on the line, the CPU performs a **state save** and **transfers control to a generic routine**.
- The generic routine examine the interrupt information and calls the **interrupt-specific handler**.
  - Pooling all the devices to see which one raised the interrupt.
- Ideally, interrupts must be handled quickly (data overflow on keyboard controller).
  - Fortunately, only a predefined number of interrupts is possible.
  - The modern interrupt mechanism accepts an ***address*** (index) in accordance with a ***interrupt vector*** (table) to provide fast interrupt service.

Interrupt Mechanism (2/3)

1.先interrupt
2.存程式狀態
3.進OS，到generic routine
4.確定是誰發出interrupt
5.跑那個地址的東東

- The ***interrupt vector*** contains the addresses of all the specific interrupt handle routines.
- The ***address*** (index) is an offset in the interrupt vector.
- This vectored interrupt mechanism can reduce the need for a single interrupt handler to search all possible sources of interrupts.
- Operating systems as different as Windows and Unix dispatch interrupt in this manner.

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

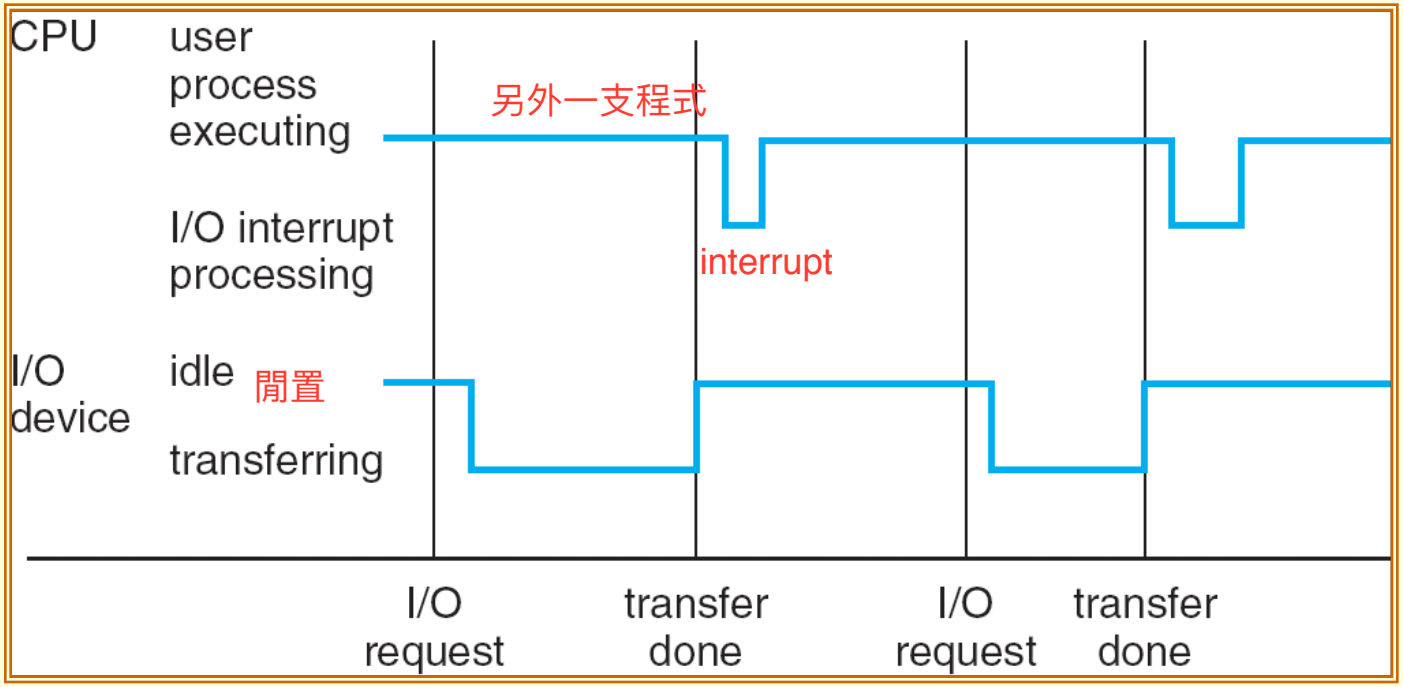
Used for device-generated interrupts

The design of the interrupt vector for the Intel Pentium processor.

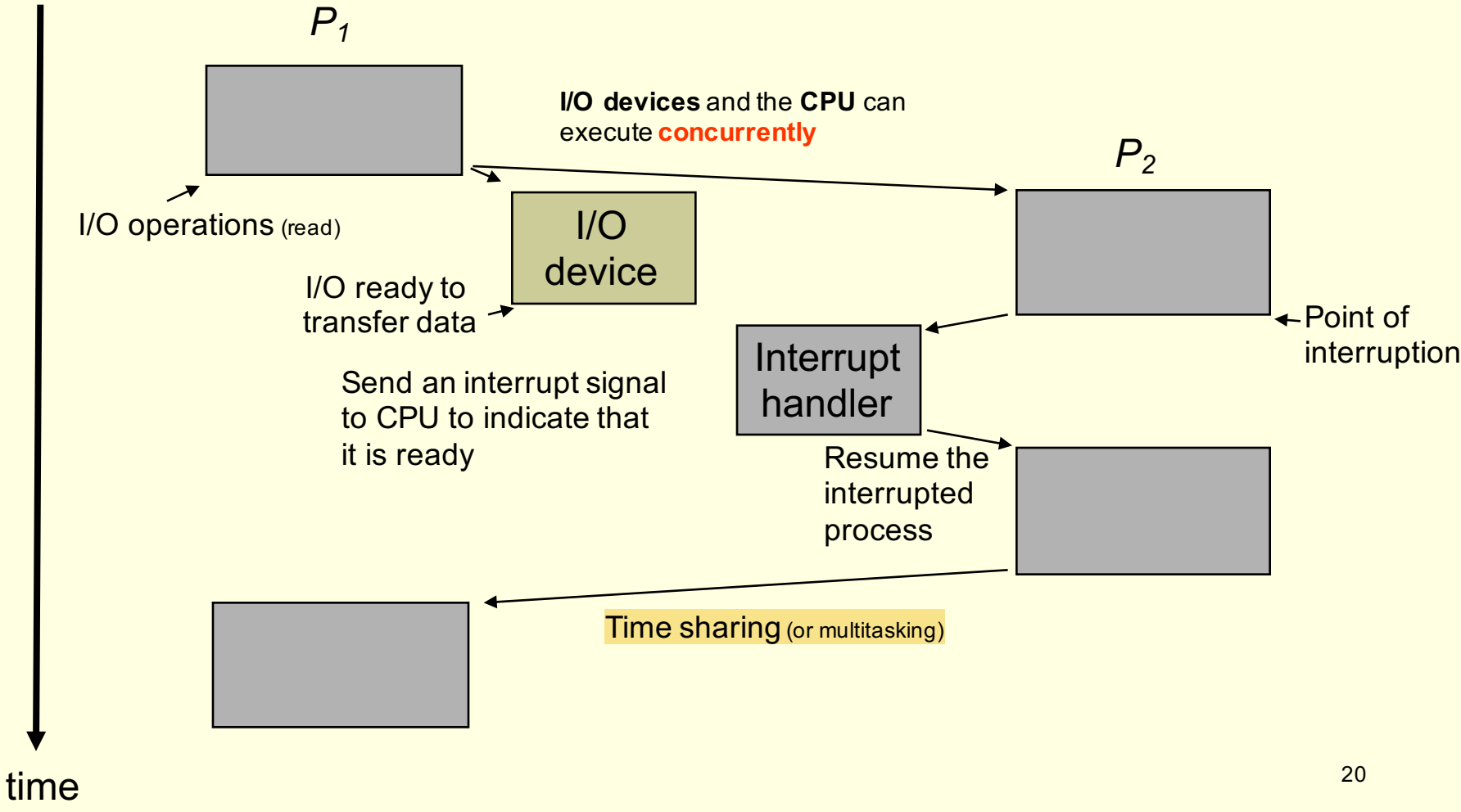
Interrupt Mechanism (3/3)

- The interrupt mechanism must also **save the address of the interrupted instruction**.
  - It may also save the **state** (processor register values).
- After the interrupt is serviced, the saved address (and saved state) is loaded, and the interrupted computation resumes as though the interrupt had not occurred.
- Note:
  - Incoming interrupts are *disabled* while another interrupt is being processed to prevent a *lost interrupt*.
  - A ***trap*** is a ***software-generated interrupt*** caused either by an error or a user request (system call).

Interrupt Timeline



Example of Interrupt



Storage Structure (1/4)

- ***Main memory*** and the ***registers*** are the only storage that the CPU can access directly.
- Registers:
  - Are built into the CPU.
  - CPU can decode instructions and perform simple operations on register contents.
- Main memory:
  - Computer programs must be in main memory to be executed.
  - Is the **only large storage area** that the CPU can access directly.

Storage Structure (2/4)

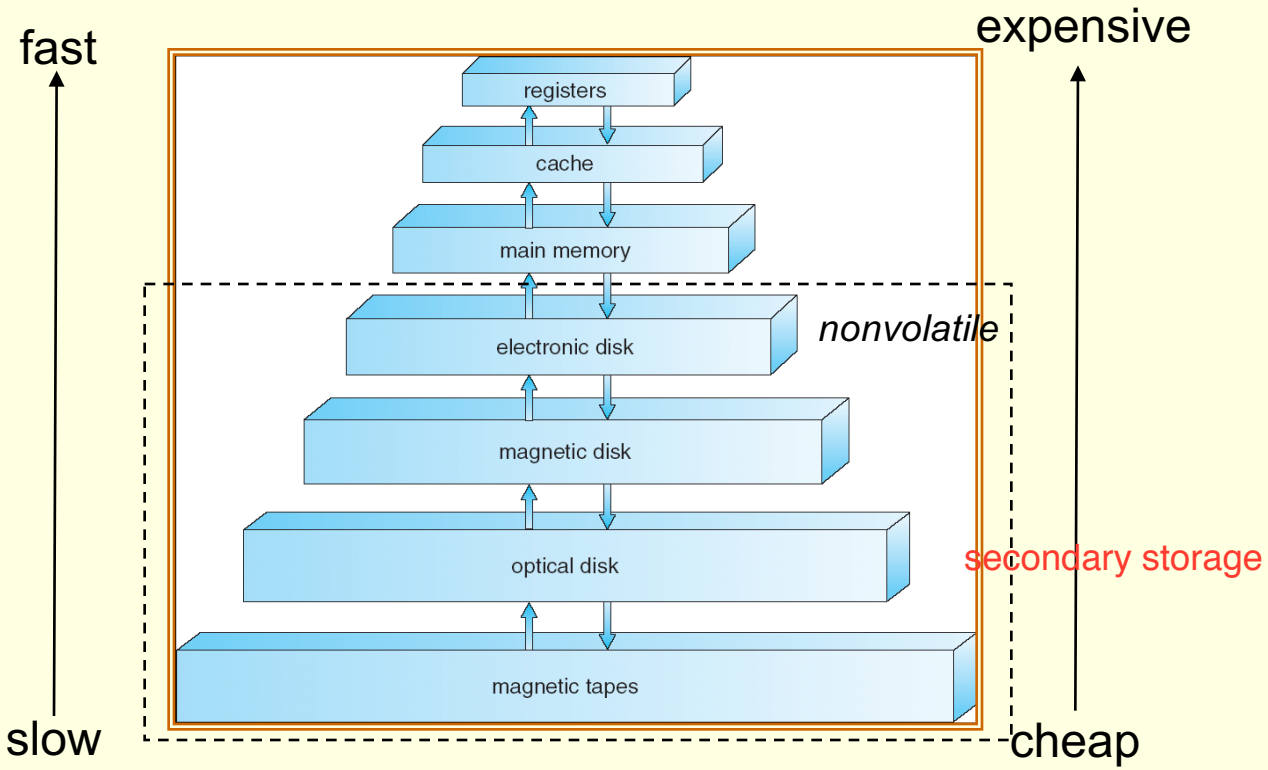
- A typical (instruction) execution cycle first ***fetches*** an instruction from memory.
  - The instruction is stored in the ***instruction register***.
- CPU then ***decodes*** the instruction.
  - May cause operands to be fetched from memory and stored in some ***internal register***.
- After the instruction on the operands has been executed, the result may be stored back in memory.
- Memory contains information about **data** and **programs**.

Storage Structure (3/4)

- Ideally, we want the **programs** and **data** to reside in main memory permanently.
  - **Impossible!!**
    - Main memory is too **small** to store all needed programs and data.
    - Main memory is a **volatile** storage device that loses its contents when power is turned off.
- ***Secondary storage*** – extension of main memory that provides large nonvolatile storage capacity.
  - Magnetic disks – The most common secondary storage.

Storage Structure (4/4)

- Other storage units: CD-ROM, tapes, and so on.
- Storage systems organized in ***hierarchy*** according to **speed** and **cost**.



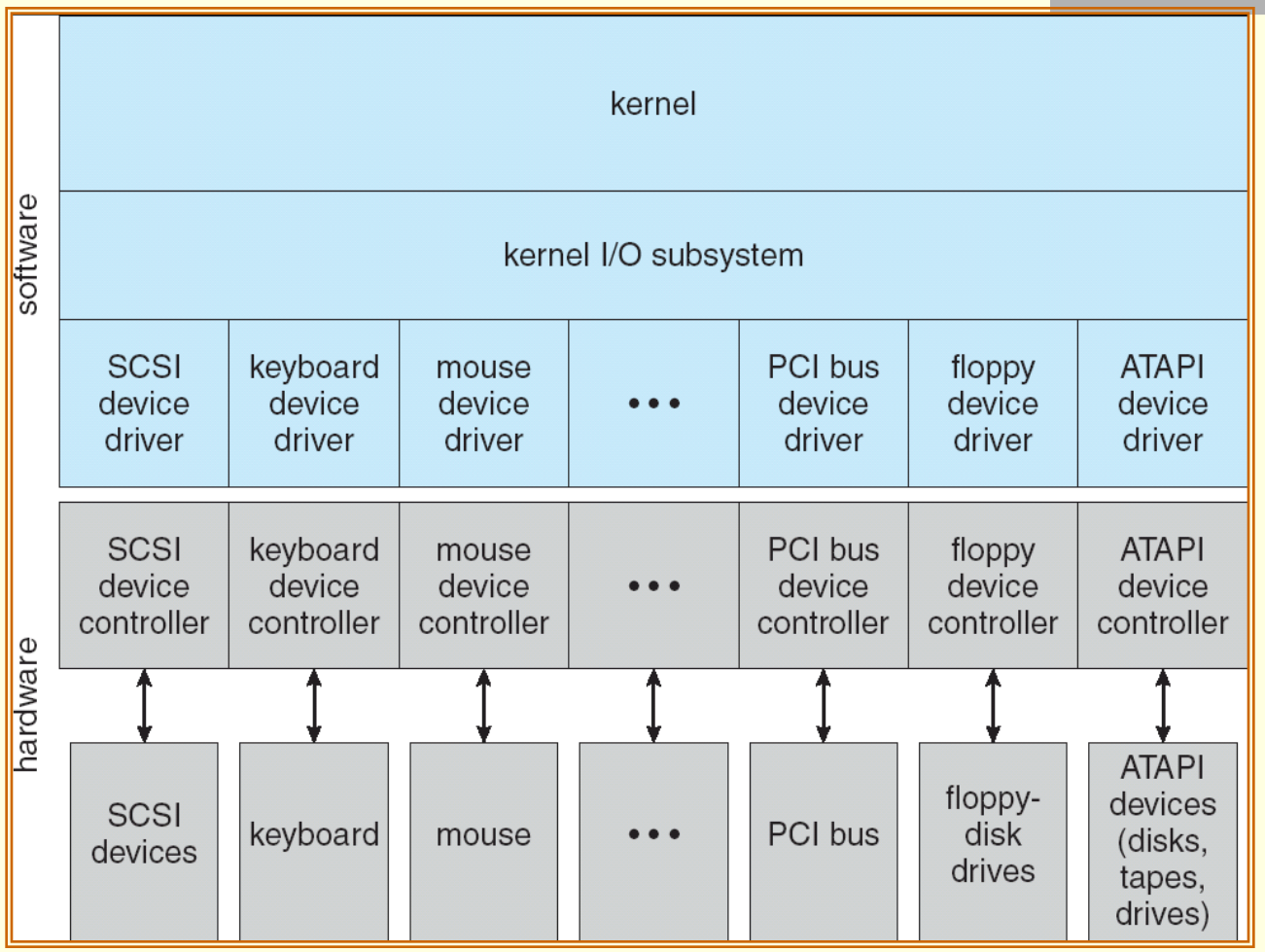
I/O Structure (1/6)

- A large portion of operating system code is dedicated to managing I/O.
  - Importance to the reliability and performance of a system.
  - The varying nature of the devices.
- A general-purpose computer system consists of ***CPUs*** and multiple ***device controllers*** that are connected through a common ***bus***.
  - Each device controller is in charge of a specific type of device (e.g., disk controller and disks).

I/O Structure (2/6)

- A device controller maintains some local ***buffer*** storage and a set of ***registers***.
  - Controller is responsible for moving the data between the devices that it controls and its local buffer storage.
- Operating systems have a ***device driver*** for each device controller.
  - The device driver understands the device controller and presents a **uniform interface** to the device to the operating system.

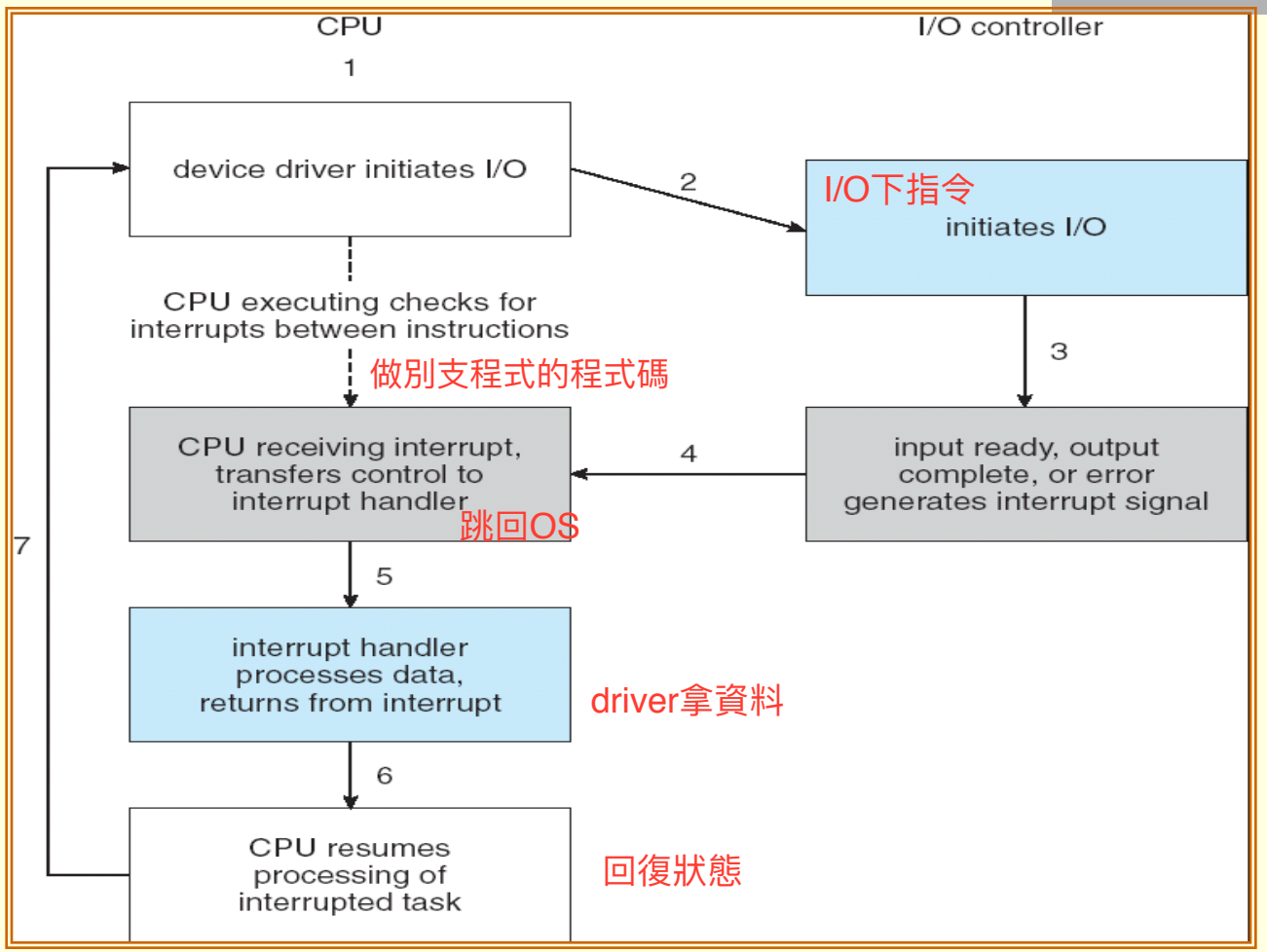
I/O Structure (3/6)



I/O Structure (4/6)

- To start an I/O operation (such as “read a character from a disk device”):
  - The **device driver** loads the appropriate registers with the device controller. 拉排線，下command
  - The **device controller**, in turn, examines the contents of these registers to determine what action to take. 接command，做事
  - The controller starts the transfer the data from the device to its local buffer. 這是specific routine
  - Once the transfer the data is complete, the device controller informs the device driver via an **interrupt**.
  - The device driver then returns control (the received data) to the operating systems.

I/O Structure (5/6)



I/O Structure (6/6)

- The purpose of device driver is to hide the differences among device controllers from the I/O subsystem of the kernel.
  - An I/O `read()` can read data from SATA, ATA, or SCSI hard disks by using specific device drivers.

客製化

- Device drivers are **internally custom-tailored** to each device but **export standard interfaces** to the I/O subsystem of the kernel. 遵循一定的STD interface
  - Benefit:
    - Simplify the job of the operating system developer.
    - New devices are easy attached to a computer system without waiting for the operating-system vendor to develop support code.

Computer System Architecture (1/5)

- Computer systems can be categorized according to the number of general-purpose processors used.
- ***Single-processor systems***:
  - There is *one main CPU* capable of executing a general-purpose instruction set.
  - Almost all systems have other *special-purpose processors* as well.
    - Device-specific processors (e.g., graphics controllers). eg. GPU
    - Do not run user processes and are managed by the operating system.
    - Relieve the overhead of the main CPU.

Computer System Architecture (2/5)

- ***Multiprocessor Systems***:
  - As known as ***parallel systems***.
  - Have **two or more processors** in close communication.
  - Share the same computation resources (memory, bus, …).
  - Main advantages:
    - **Increased throughput**:
      - We expect to get more work done in less time.
      - The speed-up ratio with *N* processors is not *N*, however; rather, it is less than *N*.
      - A certain amount of overhead is incurred in keeping all the parts working correctly.

Computer System Architecture (3/5)

- Main advantages:
  - **Economy of scale**:
    - Cost less than equivalent multiple single-processor systems, because they share computation resources.
      - Store data on the same disk vs. many copies of the data.
  - **Increased reliability**:
    - If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down.

Computer System Architecture (4/5)

- The multiple-processor systems in use today are of two types:
  - **Asymmetric multiprocessing** (master-slave relationship):
    - Each processor is assigned a specific task.
    - A **master** processor controls the system.
    - The master processor schedules and allocates work to the **slave** processors.
  - **Symmetric multiprocessing** (SMP):
    - All processors are **peers**, performs all tasks within the operating system.
    - Operating system must be written carefully to avoid unbalanced resource arrangement.
      - One processor may be sitting idle while another is overloaded.
    - Virtually all modern operating systems (windows, Mac OSX, Linux) now provide support for SMP.

Computer System Architecture (5/5)

- A recent trend in CPU design is to include multiple compute **core** on a single chip.
  - In essence, these are multiprocessor chips.
  - These multi-core CPUs look to the operating system just as *N* standard processors.

Operating System Structure (1/7)

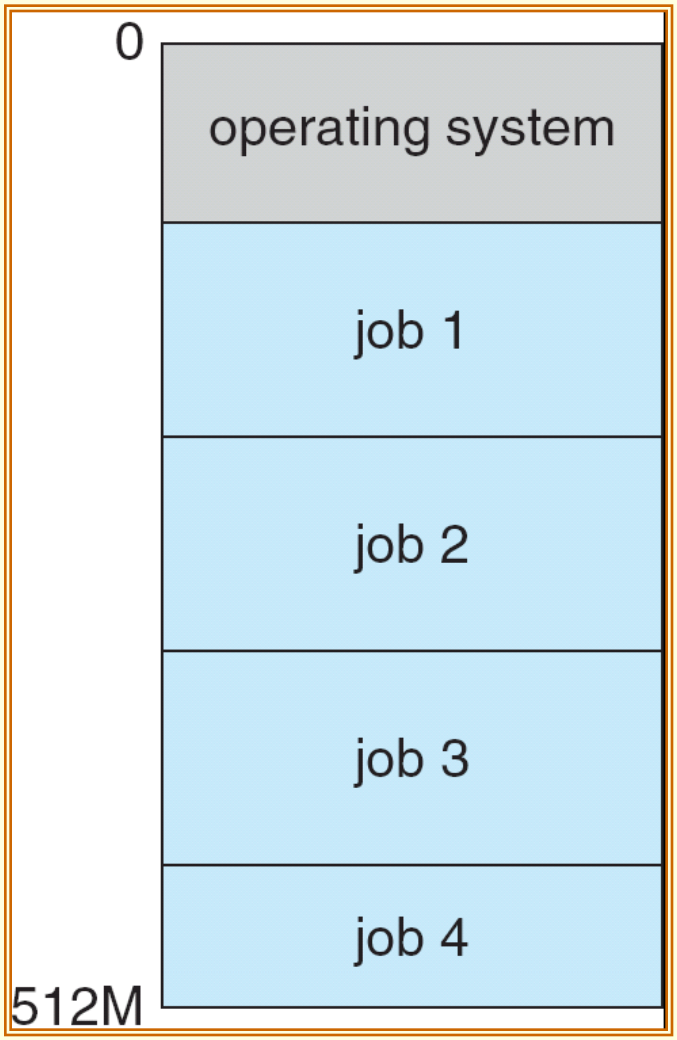
- Operating systems vary greatly in their makeup, since they are organized along many different lines.
  - However, there are many commonalities.
- One of the most important aspect of operating systems is the ability to ***multiprogramming***.
  - User cannot keep CPU and I/O devices busy at all times.
    - Jobs have to wait for some task, such as an I/O operation, to complete.
    - The CPU would sit idle.
  - Multiprogramming organizes jobs so **CPU always has one to execute**.

Operating System Structure (2/7)

- ***Multiprogramming***:
  - A subset of total jobs in system is kept in memory.
  - One job is selected and run.
  - When it has to wait (for I/O for example), OS switches to another job.
  - When that job needs to wait, the CPU is switched to another job, and so on.
  - Eventually, the first job finishes waiting and gets the CPU back.

  - As long as at least one job needs to execute, **the CPU is never idle**.

Operating System Structure (3/7)



Operating System Structure (4/7)

- ***Time sharing*** (multitasking):
  - Is logical **extension of multiprogramming**.
  - CPU switches jobs **frequently**.

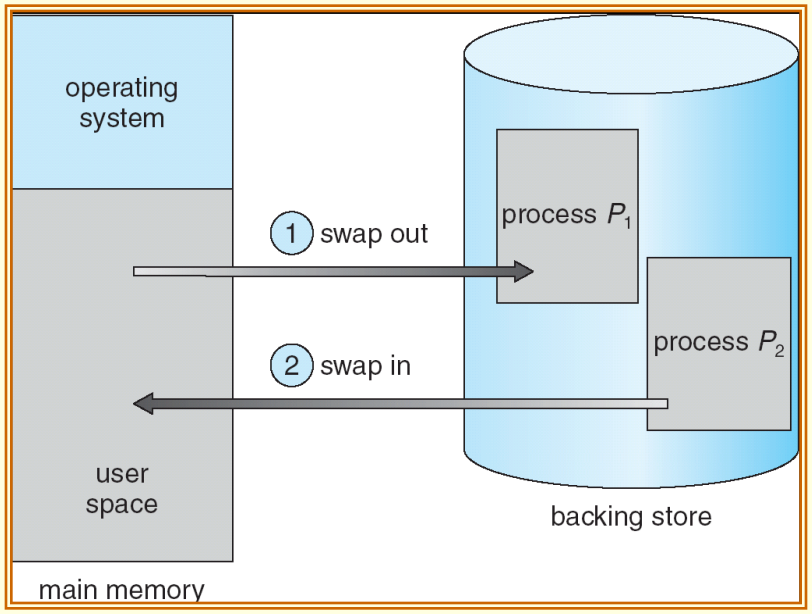
- Time sharing is frequently used in ***interactive computer system***.
  - For example, windows systems, which provide direct communication between the user and the system (using keyboard or mouse).
    - The response time (of each job) should be short!!
  - The operating system must switch rapidly from one job (for one user) to next, such that each user is given the impression that the entire computer system is dedicated to his use.

Operating System Structure (5/7)

- Time sharing and multiprogramming require several jobs to be kept simultaneously in memory.
  - A program loaded into memory and executing is called a ***process***.
- Since main memory is too small to accommodate all jobs, the jobs are kept initially on the disk in the ***job pool***.
  - The pool consists of all processes residing on disk awaiting allocation of main memory.
- ***Job scheduling*** (chapter 5) selects jobs from the pool and loads them into memory for execution.
- If several jobs ready to run at the same time ***CPU scheduling*** (chapter 5) chooses among them.

Operating System Structure (6/7)

- If processes don't fit in memory, ***swapping*** moves them in and out to run (chapter 8).
  - Swap to a backing store (e.g., hard disks).
  - For example a higher-priority process arrives and wants service, the memory manager can swap out the lower-priority process and then load and execute the higher-priority process.



Operating System Structure (7/7)

- ***Virtual memory*** allows execution of processes not completely in memory (chapter 9).

Operating-System Operations (1/2)

- Modern operating systems are ***event driven***.
  - Waiting for something (event) to happen.

hardware緊急狀況 或是 software發出（程式錯誤）

- Events are signaled by the occurrence of an ***interrupt*** or a ***trap*** (software interrupt).
  - **Interrupt** driven by **hardware**.
  - **Software error** or **request** creates **trap**.
    - E.g., division by zero, request for operating system service.
- The **operation structure** of interrupt (event) driven operating system:
  - For each interrupt, separate segments of code in the operating system determine what action should be taken.
  - An **interrupt service routine** (IRS) is provided that is responsible for dealing with the interrupt.

Operating-System Operations (2/2)

- Since there are multiple running processes, we (operating system) need to make sure that an error in a program can not affect other programs.
  - Infinite loop in one process could prevent the correct operation of many other processes.
  - One erroneous (malicious) program might modify another program/data/OS.
  - **Protection** and **security** are very important!!

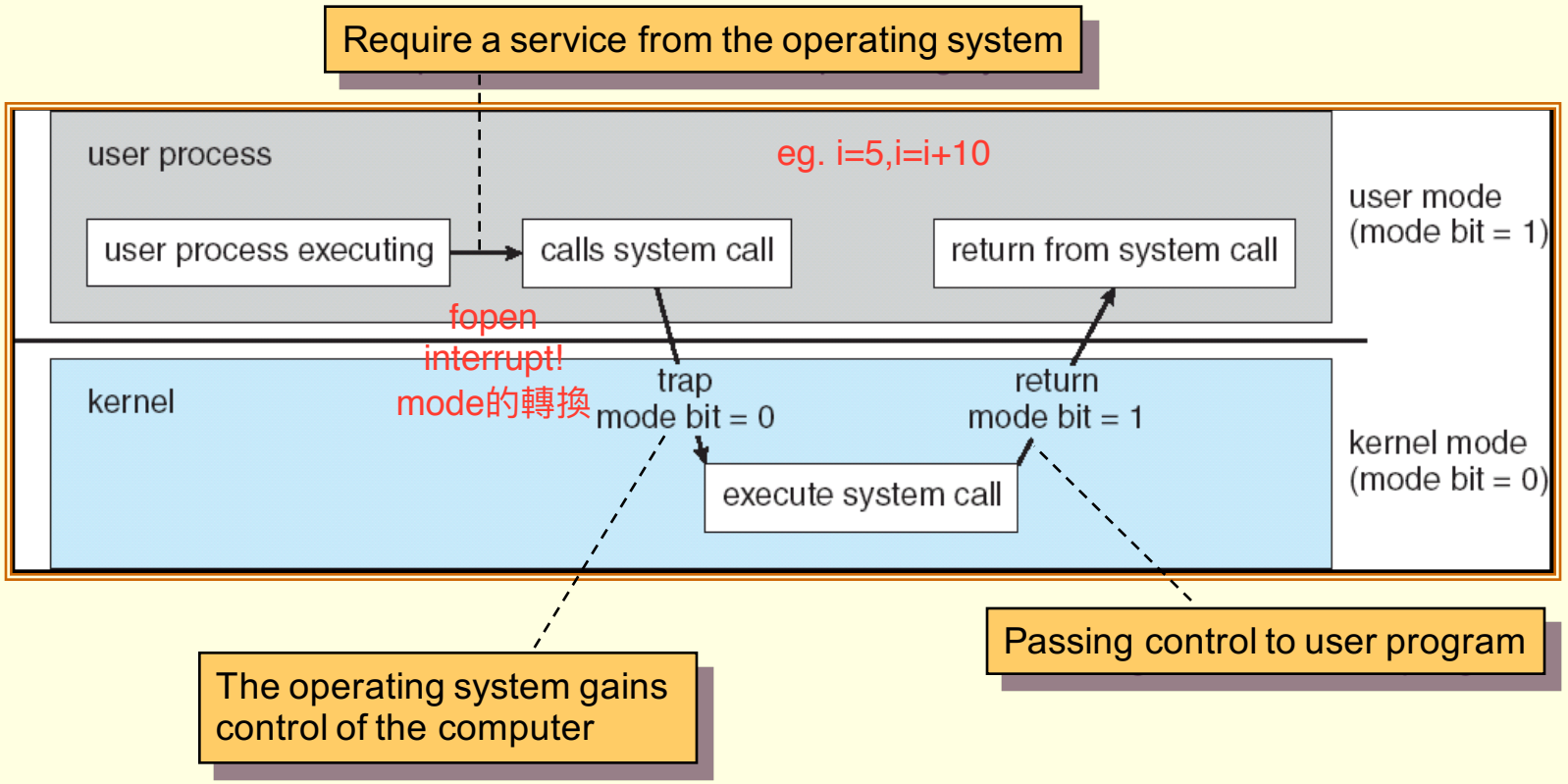
Dual-Mode Operation (1/4)

- To ensure the proper execution of the operating system, we must be able to **distinguish** between the execution of **operating-system code** and **user-defined code**. 需要hardware來幫忙
  - Computation resources **can only** be managed by operating-system code.
  - User-defined code can not cross the line.
  - Supported by hardware mechanism.

kernel mode:仲裁resource可不可以給user

- **Dual-mode: *user mode*** and ***kernel mode***. 針腳高低電位
  - **Mode bit** provided by hardware, kernel (0) or user (1).
    - Recent versions of the Intel CPU do provide dual-mode.
  - When a user application requests a service from operating system, it must transition from user to kernel mode to fulfill the request.
  - Request only through ***system call***. software interrupt

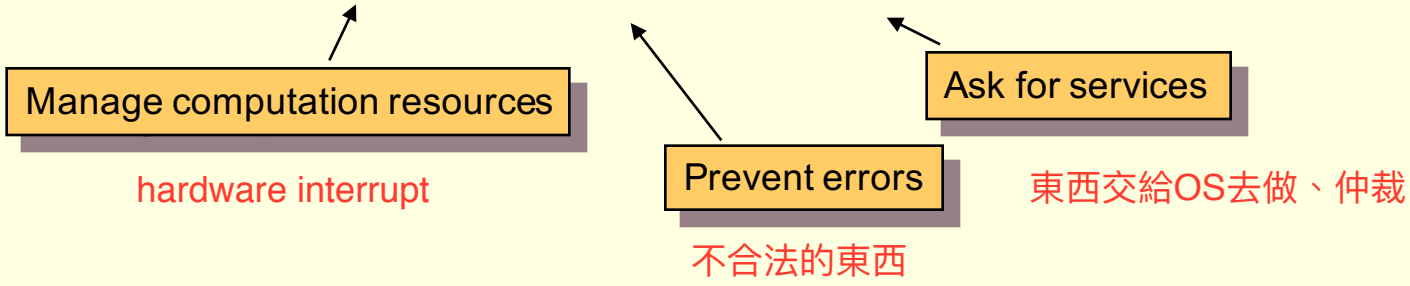
Dual-Mode Operation (2/4)



interrupt: user->generic routine->specific routine(eg.開檔案、記憶體夠不夠用，維持安全性)->轉換回user mode

Dual-Mode Operation (3/4)

- The dual mode protects the operating system from errant users. 避免OS被破壞
  - We designate some machine instructions that may cause **harm** as ***privileged instructions***, and only executable in kernel mode. user mode無法執行，只有kernel mode可執行
    - I/O control, interrupt management … are examples of privileged instruction.
  - Generally, control is switched to the operating system via an interrupt, a trap, or a system call.



Dual-Mode Operation (4/4)

- More description of system call:
  - When a system call is called, it is treated by the hardware as a software interrupt.
  - The mode bit is set to kernel mode.
  - Control passes through the interrupt vector to a service routine in the operating system. software interrupt
    - The kernel examines the parameter of the interrupt to determine what type of service the user program is requesting.
  - The kernel verifies and executes the request.
  - And returns control to the instruction following the system call (user mode).

Process Management (1/2)

- ***Process***:
  - A program in execution.
    - Program is a *passive entity*, process is an *active entity*.
  - A unit of work within the system.
- Process needs resources to accomplish its task.
- Process termination requires reclaim of any reusable resources.
  - CPU, memory, I/O, files.
- Single-threaded process has one ***program counter*** specifying **location of next instruction** to execute.
- Multi-threaded process has one program counter per thread.
  - Process executes instructions sequentially, one at a time, until completion.

Process Management (2/2)

- Typically system has many processes, some **user**, some **operating system** running **concurrently** on one or more CPUs.

- The operating system is responsible for the following activities in connection with process management (chapters 3 ~ 6):
  - Creating and deleting both user and system processes
  - Suspending and resuming processes.
  - Providing mechanisms for process synchronization.
  - Providing mechanisms for process communication.
  - Providing mechanisms for deadlock handling.

Memory Management (1/2)

- The ***main memory*** is generally the only large storage device that the CPU is able to address and access directly.
  - Main memory is a large array of **words** or **bytes**, ranging in size from hundreds of thousands to billions.
  - Each word or byte has its own (memory) address.
- **Data** must be in memory before and after processing.
  - For example, load data from disk into memory.
- All **instructions** must be in memory in order to execute.

Memory Management (2/2)

- General-purpose computers must keep **several** programs in memory to improve the computer performance.
  - Creating a need for memory management (chapters 8 and 9).
- Memory management activities:
  - Keeping track of which parts of memory are currently being used and by whom.
  - Deciding which processes (or parts thereof) and data to move into and out of memory.
  - Allocating and deallocating memory space as needed.

Storage Management

- The operating system provides uniform, **logical view** of **physical** storage media.
  - **Logical** storage unit — **file**.
  - **Physical** storage media — disk, tapes, …
- The operating system **maps** files onto physical media and access these files via the storage devices.

File-System Management

- A ***file*** is a collection of related information defined by its creator.
  - Represent programs and data.
    - Data: numeric, alphabetic, or binary. free-form or non-free form.
- Files usually organized into ***directories*** to make them easier to use.
- Multiple user can access to the same file.
  - **Access control** on most operating systems to determine who can access what.
- OS activities include (chapters 10 and 11):
  - Creating and deleting files and directories.
  - Primitives to manipulate files and directories.
  - **Mapping files onto secondary storage.**
  - Backup files onto stable (non-volatile) storage media.

Mass-Storage Management (1/2)

- Computer system must provide ***secondary storage*** to back up main memory.
  primary: memory
  - ***Disks*** are used as the principal storage medium for programs and data.
- Disks are frequently used as the source and destination of program processing.
  - Proper management is of central importance.
  - Speed of computer operation hinges on disk subsystem.
- OS activities:
  - Free-space management.
  - Storage allocation.
  - Disk scheduling.

Mass-Storage Management (2/2)

- ***Tertiary storage***: 第三位
  - Storage that is slower and lower in cost than secondary storage.
    - Optical storage, magnetic tape.
  - Backup disk data.
  - Still must be managed.

Caching (1/4)

- Important principle of computer systems.
- Information is normally kept in some storage system (large, slow, and cheap).
- As it is used, it is copied into a faster (small, and expensive) storage system — the ***cache***.
- When we need a particular piece of information, we **first** check faster storage (cache) to determine if information is there.
  - If it is, information used directly from the cache (fast).
  - If not, **data copied to cache** and used there.
- Because cache is smaller than storage being cached, **cache management** is an important design problem.
  - Careful selection of the **cache size** and **replacement policy** (chapter 9).

Caching (2/4)

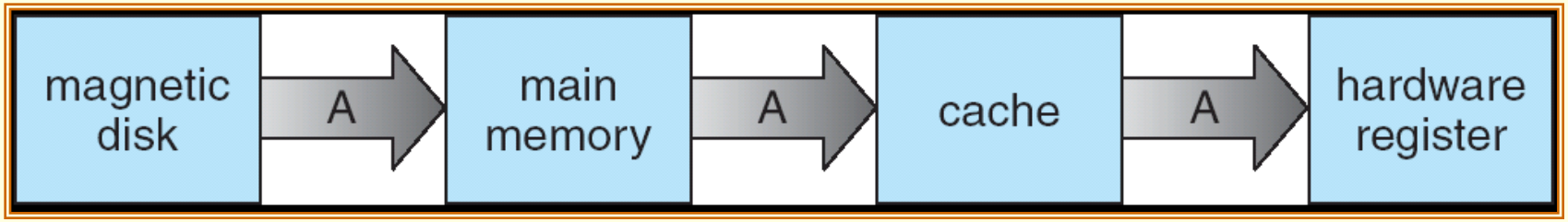
- Performance of Various Levels of Storage:

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

- Movement between levels of storage hierarchy can be **explicit** or **implicit**:
  - For example, transfer of data from disk to memory is usually controlled by the operating system (explicit).

Caching (3/4)

- In a hierarchical storage structure, **the same data may appear in different levels of the storage system**.
- An integer A that is to be incremented by 1 is located in file B on disk.



  - Once the increment takes place in the register, **the value of A differs in the various storage system!!**

- In an environment where only one process executes at a time.
  - An access to A will always to be the copy at the highest level of the storage hierarchy.

Caching (4/4)

- In multitasking environments:
  - Each process (has its memory space) must obtain the most recently updated value of A.
- In multiprocessor environment:
  - Each CPU also contains a local cache.
  - **A copy of A may exist simultaneously in several caches.**
  - Since CPUs can execute **concurrently**, an update of A in one cache must be reflected in all other caches where A resides — ***cache coherency***.
- Distributed environment situation even more complex.
  - Several copies of a datum can exist.
  - Various solutions covered in Chapter 17.

Protection and Security (1/2)

- A multi-user computer system allows the concurrent execution of multiple processes.
  - Computation resources must be operated in a **proper** and **authorized** manner.
    - For example, a process can execute only within its own address space.
- ***Protection*** – any mechanism for controlling access of processes or users to resources defined by the operating system.
- A system can have adequate protection but still be prone to failure and allow inappropriate access.
  - A user's authentication information is stolen.
- ***Security*** – defense of the system against internal and external attacks.
  - Huge range, including denial-of-service, viruses, …

Protection and Security (2/2)

- Protection and security require the operating system to be able to distinguish among all its users.
- Most systems maintain a list of user names and associated ***user identifiers*** (user IDs, security IDs).
  - User ID is then associated with all files, processes of that user to determine access control.
- ***Group identifier*** (group ID) allows set of users to be defined and managed, then also associated with each process, file.
- ***Privilege escalation*** allows user to change to effective ID with more rights for an activity.
  - For example, on UNIX, the `setuid` attribute on a program causes that program to run with the user ID of the owner of the file, rather than the current user's ID.

提升特權

End of Chapter 1

重點：
system call
dual mode