COMMONWEALTH OF AUSTRALIA
Copyright Regulation 1969

WARNING

This material has been copied and communicated to you by or on behalf of **Curtin University of Technology** pursuant to Part VB of the *Copyright Act 1968* (**the Act**)

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice

Operating Systems
COMP2006

Overview
Lecture 1

COMP2006 Operating Systems

Prerequisites:

- Data Structures and Algorithms
- Unix and C Programming

Knowledge on data structures and C language is essential to complete programming assignment.

Textbooks

Required Text:

Silberschatz, Galvin, and Gagne, *Operating System Concepts*, 9th edition, John-Wiley & Sons, 2013

Overview

Reference

Chapter 1-2, and Section 13.1-13.2

Topics:

- Overview of the operating system, its development, purposes, components and services.
- Overview of basic hardware structure and operations
- OS structures.

What is an OS?

- An Operating System (OS) is a PROGRAM that acts as an interface between a USER of a computer system and the COMPUTER HARDWARE
  - OS manages the computer hardware, and provides a basis for running application programs
- There is no universally accepted definition of OS and what is part of OS and what is not
  - Is a web browser a part of OS?
- Kernel – a program that is running all the time on the computer
  - Other programs are considered as system/application programs

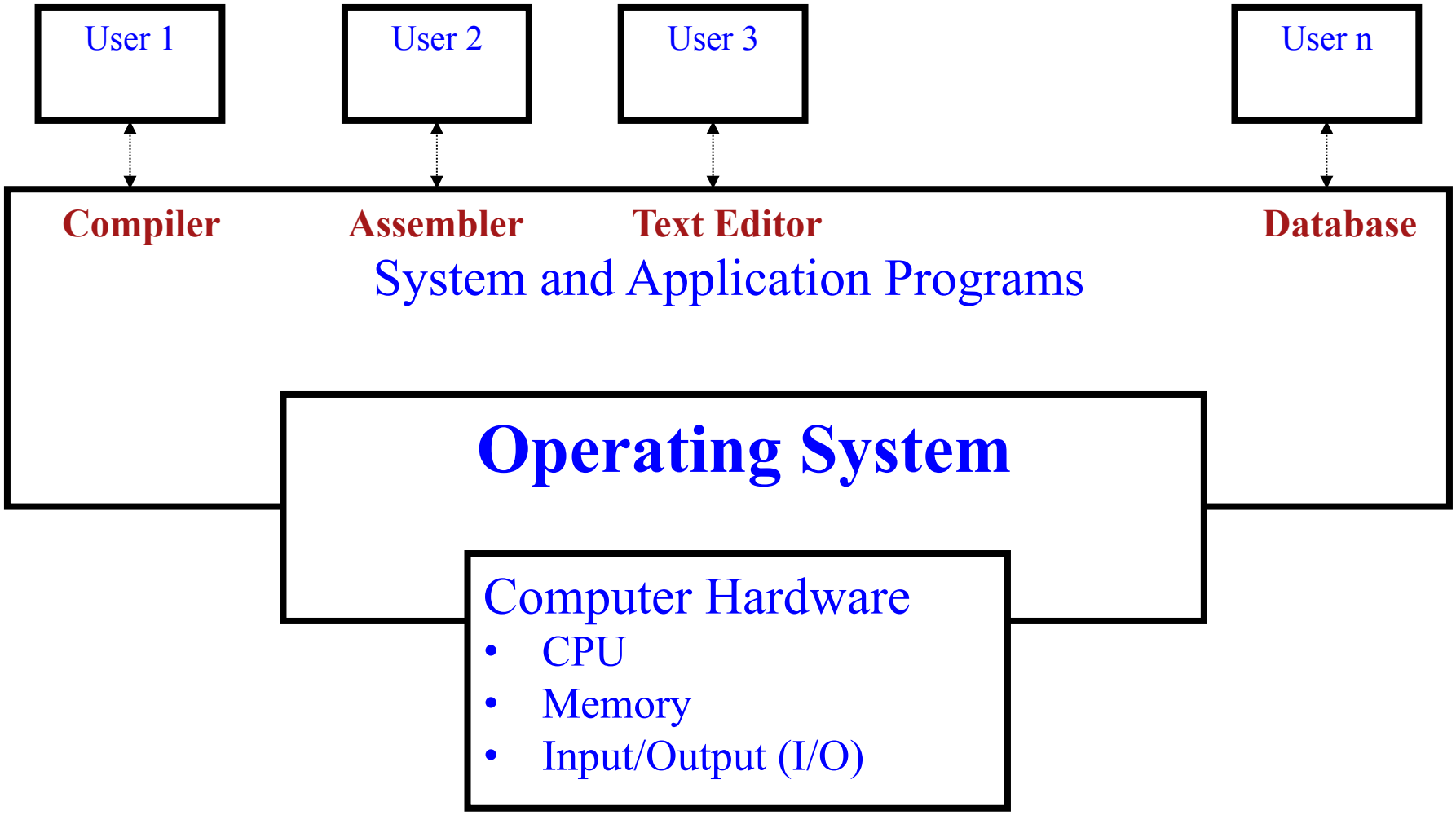
OS Components

- Process Management
- Main-memory management
- Secondary-storage management
- File Management
- I/O System Management
- Protection System
- Networking (Distributed Systems) – NOT discussed
- Command- interpreter System – Not discussed

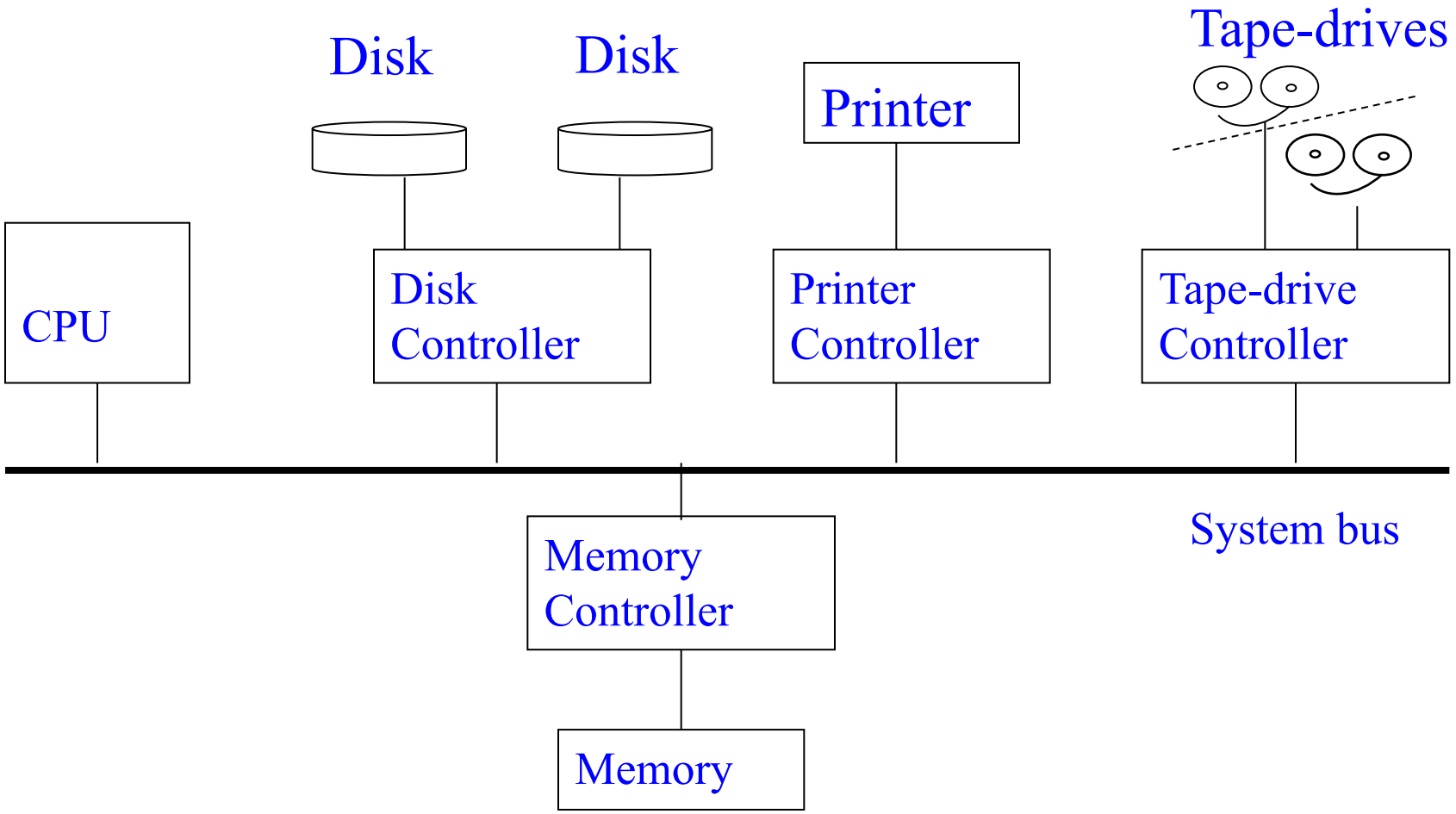
OS Purposes

1) To provide an environment for a user to execute programs conveniently and efficiently.
   - Efficient operation of computer system is important in large (expensive) multi-user machines.
   - Convenience and efficiency are often contradictory.
     - Aims for convenience sacrifice efficiency.
     - In the past → efficiency is more important.
     - Now → users need more conveniences?
2) To simulate features not available on hardware.
   - OS provides features factually not provided by the hardware.
   - What features?
3) To control all the computer's resources and to provide the base upon which the application programs can be written.

Abstract of Computer System



Computer System Organization



Computer System Organization (cont.)

- A general purpose computer system contains:
  - One or more CPU (Central Processing Unit): the 'brain' of the system.
  - Memory system
  - Input/output devices, e.g., disks, printers
- System *bus* connects the CPU, I/O devices, and Memory unit
  - All signals flow through the bus.
- Each device controller is in charge of a particular device type (e.g., disk drives, video displays, etc.), and each has a local *buffer*.
  - CPU moves data from/to main memory to/from local buffers.
- I/O is from the device to local buffer of controller.
  - Device controller informs CPU that it has finished its operation by causing an *interrupt*.
  - I/O devices and CPU can execute *concurrently*.
  - Memory controller is provided to synchronize access to memory.

Computer System Organization (cont.)

- Computer needs an initial program, called **bootstrap** program, to start running
  - When it is powered up
  - When it is rebooted
- Bootstrap is typically stored in hardware, called **firmware**
  - Read-only memory (ROM)
  - Electrically erasable programmable read-only memory (EEPROM)
- Bootstrap program initializes all aspects of system:
  - *CPU registers*
  - Memory contents
  - Device controllers
- Bootstrap program knows how to locate OS kernel, load it in memory, and run it.

Computer System Organization (cont.)

- Kernel runs all the time and provides services to the system and its users
  - System programs are loaded and start running at boot time
  - System programs that are running are called **system processes** or system **daemon** that run when the kernel is running
- In Linux, the first system process is called **init**
  - It runs other daemons
- Once the system is fully booted, it waits for some events to occur
  - An operating system is *interrupt-driven*.
  - The system knows when there occurs an event through *interrupt* either from hardware or software (called **trap**).
    - Hardware generates interrupt by sending *signal* to CPU via bus
    - Software sets interrupt using **system call** (also called **monitor call**)

Interrupt Mechanism

- Receiving an interrupt, CPU stops what it is doing, and jump to a specific location to execute the *interrupt service routine* (ISR)
  - The location is in the *interrupt vector table*, generally stored in low memory area – a protected area (i.e., can be accessed only by OS)
  - Interrupt vector table contains the addresses/pointers of all ISRs.
  - Each interrupt is known by its interrupt number/vector
    - Interrupts with lower numbers have higher *priority*
    - CPU computes the location by multiplying interrupt number with a fixed value, e.g., 4 → the starting address of ISR for interrupt 40 is 40 * 4 = 160.
- CPU resumes its execution on the completion of ISR
  - CPU must save the address of interrupted instruction before serving any interrupt, and processor states, e.g., register contents
    - The return address is stored on the *system stack*
  - The saved address is put into the *program counter*
    - Program counter contains the address of the next instruction to be executed by CPU
  - Using the saved processor states, CPU resumes the interrupted execution.

Intel Pentium interrupt vector table

Figure 13.4 (textbook)

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

Interrupt Controller

- Efficient interrupt handling is required for good system performance.
  - A server may handle hundreds of thousands of interrupts per second
- Use interrupt-controller hardware
  - It can defer interrupt handling during critical processing
  - It provides efficient dispatch to interrupt handler
  - It supports multi-level interrupts with priority
- With priority, a CPU can defer a low priority interrupt when a higher priority interrupt is being served
  - It is also possible to stop the execution of an interrupt to serve an incoming higher priority interrupt.

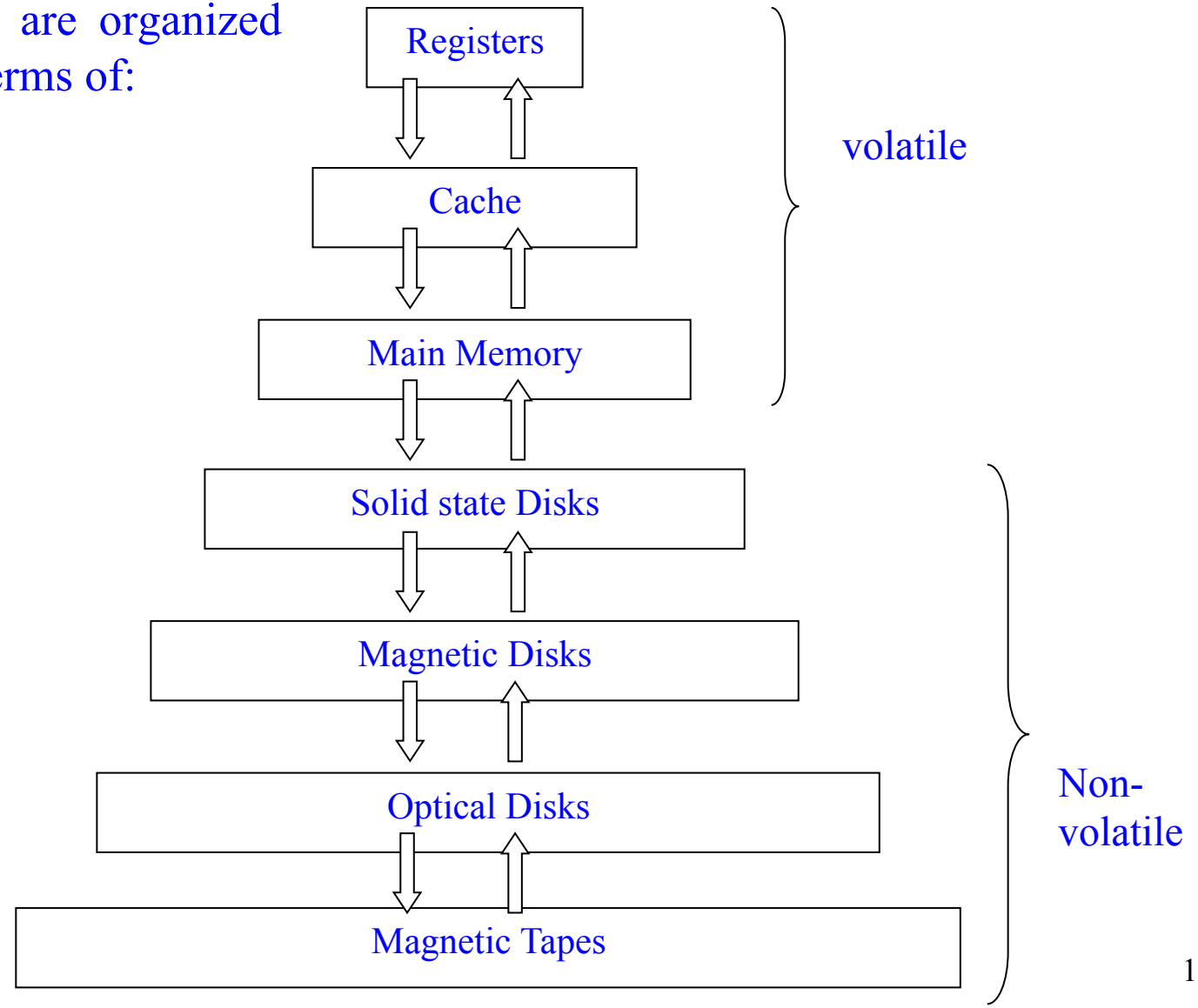
OS interaction with the interrupt mechanism

- **At boot time:** OS probes the hardware buses to see what devices are present, and installs the corresponding interrupt-handlers into the interrupt vector.
- **During I/O:** Various devices raise interrupts to indicate:
  - They are ready to provide service
  - They have completed output
  - The input data are available, or a failure has been detected
- Interrupt mechanism is also used for:
  - Exceptions: divide by zero, accessing protected memory address, page-fault, etc.
  - System call implementation; uses software interrupt or trap
    - Has lower priority than device interrupt
  - Managing flow of control within the kernel,
    - E.g., Copying data from kernel buffer to user buffer is not urgent, and thus can be done when higher priority task, e.g., reading the data from disk to the kernel buffer is completed.

Storage Structure

Storage systems are organized in hierarchy in terms of:

- Speed
- Cost
- Volatility
- Size/capacity



Performance of various levels of storage

Figure 1.11 (Textbook)

Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	25,000 - 50,000	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5,000 - 10,000	1,000 - 5,000	500	20 - 150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Storage Structure (cont.)

- CPU can execute instructions *only* from memory
  - Main memory – the only large storage media that the CPU can access directly
    - Memory keeps both programs and data (*von Neumann Architecture*).
- Execution cycle in Von Neumann Architecture
  - Fetch an instruction from memory and stores it in a register
    - The register is called the *instruction register* (IR)
    - The address/pointer of the next instruction to be fetched/executed is stored in the CPU's *Program Counter* (PC)
  - Decode the instruction
  - If the instruction needs operands, fetch the operands from memory, and store them in internal registers
  - Execute the instruction, and store the results in memory
  - Repeat the steps.

Storage Structure (cont.)

- General purpose computer uses rewritable memory for its main memory, called *random access memory* (RAM)
  - Usually implemented in *dynamic RAM* (DRAM) with access time: 80-250 ns
    - Other type of memory: *Read only Memory* (ROM), cannot be changed, is used to store static program, e.g., bootstrap
    - EEPROM: cannot be changed frequently, contains mostly static programs
- Memory can be viewed as a one-dimensional array of bytes
  - Its size is determined by the number of bits used to address each location in the memory.
    - Each byte has an address; the address starts from 0 to 2^n -1, for n bit address
    - E.g., address with 32 bits means the size is 2^{32} = 4 Giga bytes.

Note: one *word* is the CPU's native unit of data. A computer with 64 bit registers usually has 64-bit words. A computer usually executes intructions with operands of size words, rather than bytes.

Storage Structure (cont.)

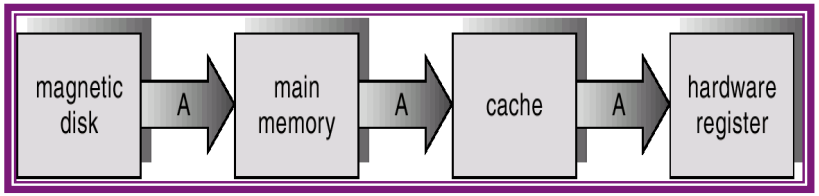
Unit standard by the International Electrotechnical Commission (IEC)

Binary			Decimal		
$2^{10}=1024$	KiB	kibibyte	1000	KB	kilobyte
$2^{20}=1024^2$	MiB	mebibyte	1000^2	MB	megabyte
$2^{30}=1024^3$	GiB	gibibyte	1000^3	GB	gigabyte
$2^{40}=1024^4$	TiB	tebibyte	1000^4	TB	terabyte
$2^{50}=1024^5$	PiB	pebibyte	1000^5	PB	petabyte
$2^{60}=1024^6$	EiB	exbibyte	1000^6	EB	exabyte
$2^{70}=1024^7$	ZiB	zebibyte	1000^7	ZB	zettabyte
$2^{80}=1024^8$	YiB	yobibyte	1000^8	YB	yottabyte

Note: The textbook and this slide uses 2^{10} bytes = KB, 2^{20} bytes = MB, etc.

Storage Structure (cont.)

- *Secondary storage* – extension of main memory that provides non-volatile storage and large capacity.
  - The most common secondary storage is *hard disk*
- Volatile storage loses its contents when it is not powered.
  - Registers, cache and main memory are volatile
  - Solid state disk, magnetic disk, optical disk and magnetic tape are non-volatile
    - Solid state disk uses DRAM in normal operation, and stores its contents to a hidden magnetic disk when it is powered off.
    - Non-volatile RAM (NVRAM) uses DRAM with backup battery
- Cache and registers are smaller but faster storage
  - CPU has a limited number of internal registers (access time: 0.25-0.5 ns) and limited sized cache (access time: 0.5 – 25 ns)

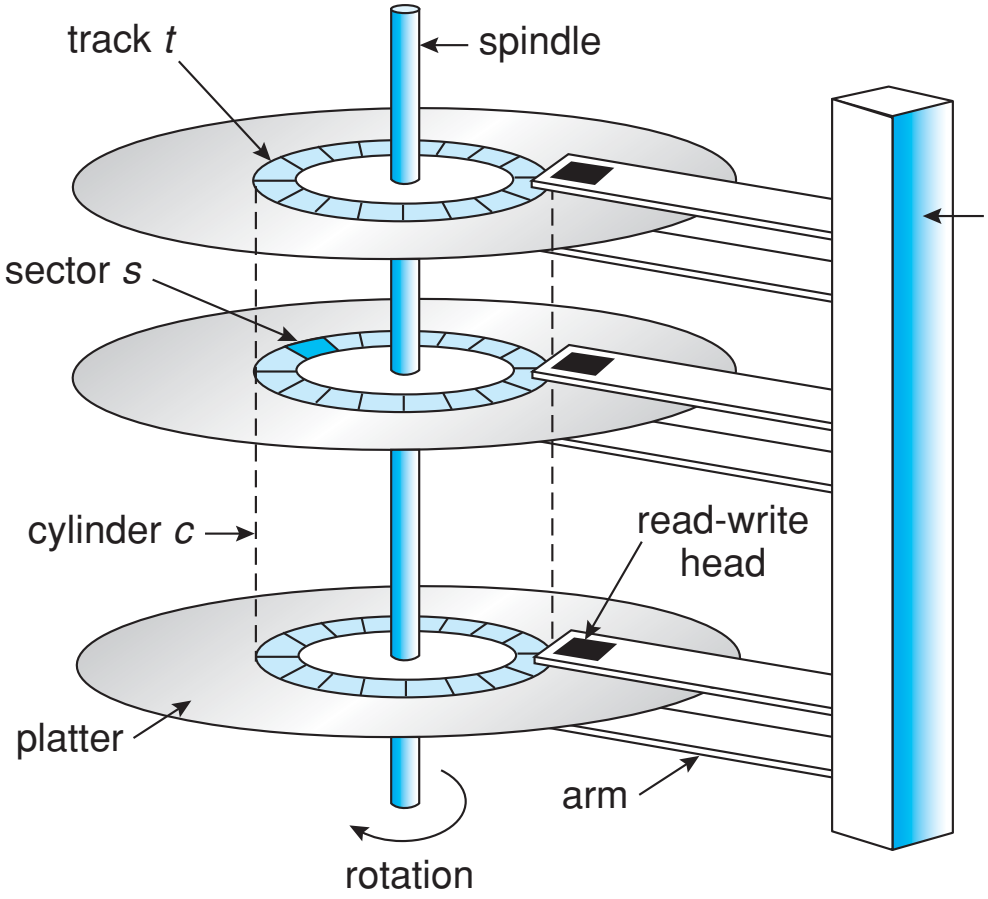


Storage Structure (cont.)

- Magnetic disks – rigid metal or glass platters covered with magnetic recording material; access time: in milliseconds, e.g., 5ms.
  - Disk surface is logically divided into circular *tracks*, which are subdivided into *sectors*.
    - The set of tracks that are in one arm position forms a *cylinder*.
  - Disk-controller determines the logical interaction between the device and the computer.
  - Disk speed has two parts:
    - *Transfer rate*: the rate at which data flow between the drive and computer (in MB per second).
    - Random access time = *seek time* + *rotational latency* (in millisecond).
      - Seek time – the time to move the disk arm to the desired cylinder.
      - Rotational latency – the time for the desired sector to rotate to the disk head.
- Magnetic tapes – its access time is about a thousand times slower than magnetic disk
  - Mainly used for backup.
- Caching – copying information into faster storage system
  - Main memory can be viewed as a fast cache (disk caching) for secondary storage.

Disk Mechanism

Figure 10.1 (Textbook)



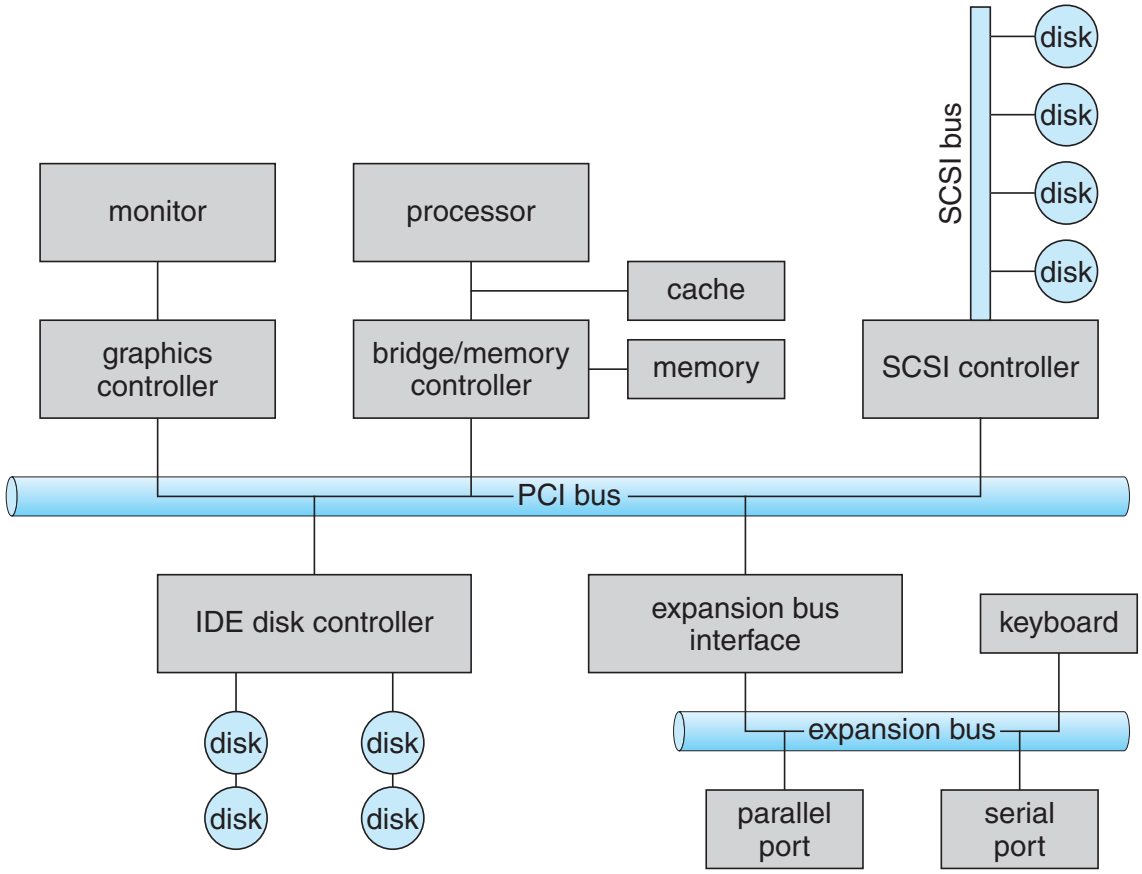
- The two sides of each platter are covered with magnetic materials to store data
  - Each read/write head, attached to a disk arm, 'flies' above each surface; the disk crashes if the head makes contact with the surface.
- Each platter is logically divided into tracks, and each track is divided into hundreds of sectors
  - A cylinder is a set of tracks that are at one arm position; there are thousands of cylinders
- Drive spins at 5400, 7200, 10000, or 15000 Rotations Per Minute (RPM)

I/O - Basic concepts

- An I/O device communicates with a computer system by sending signals over a cable or wireless via a *port.*
- A *device controller* is a collection of electronics that can operate a port, a bus, or a device.
  - A controller has registers for data and control signals.
    - *Status* register: contains bits indicating states of the device.
    - *Control* register: written by host to start a command or change the mode of a device.
    - *Data-in* register: read by host to get input.
    - *Data-out* register: written by host to send output.
  - Processor communicates with the controller by reading and writing bit patterns in these registers using special I/O instructions.
  - Some devices may have their own built-in controllers.
- Each device has address:
  - *I/O address space* in a direct I/O instructions.
  - *CPU address space* in a memory-mapped I/O; device-controller registers are mapped into CPU address space.
  - Some systems use both techniques.
- One or more devices may use a common set of wires called a *bus*.
  - A *bus* is a set of wires and a defined protocol that specifies a set of messages that can be sent on the wires.

I/O Structure

Figure 13.1 (Textbook)



Type of I/O devices:

- Storage; e.g., disks
- Transmission; e.g., network connection
- Human-interface; e.g., screen
- Other specialized devices.

Some I/O Port addresses

Fig. 13.2 (Textbook)

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

You need not memorize the numbers!

Host – Controller interactions

- How does a host communicate with an I/O controller?
  - Polling
  - Interrupts
  - Direct Memory Access
- *Polling* can be used to determine the state of a device.
  - The host repeatedly reads the **busy** bit in *status* register until it becomes not busy
  - The host sets **command ready** bit in **status** register and **command** bit (e.g., write) in **control** register when a command is available for the controller to execute, and write **one byte** in the data-out register.
  - The device controller sets **busy** bit in status register, read the command register, read the data-out register, and does I/O to the device.
  - The controller clears command ready bit, clear **error** bit in status register to indicate the device I/O succeeded, and clear busy bit.
- Polling is efficient if the device and its controller are fast.
  - Otherwise, the host should probably switch to another task.
  - Note: each poll needs only 3 CPU instructions: read register, logical AND, and jump.

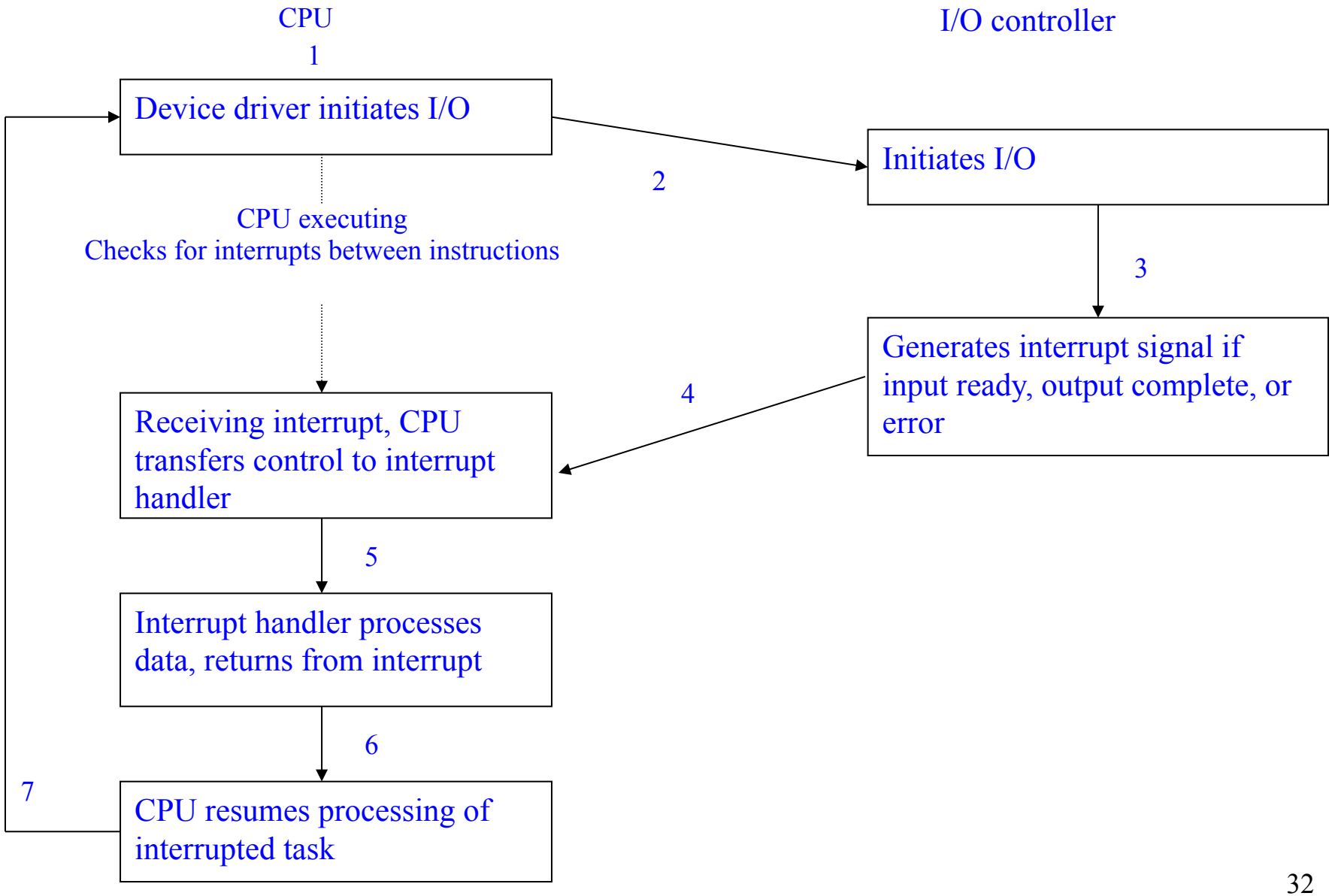
Host – Controller interactions (cont.)

- When polling is not efficient, use *interrupt*
  - Interrupt is a hardware mechanism that enables a device to notify the CPU that the device is ready for service.
- Interrupts are used in OS to handle asynchronous events and to trap to supervisor-mode routines in the kernel.
- Most CPU have two interrupt request lines:
  - **Non-maskable** interrupt: for events such as unrecoverable memory errors.
  - **Maskable** interrupt: can be turned off by CPU before execution of critical instruction sequences that must not be interrupted.

Basic interrupt I/O mechanism

1. Device controller **raises** an interrupt by asserting a signal on the CPU's interrupt request line.
2. CPU **catches** the interrupt and **dispatches** the interrupt to the *interrupt Service Routine* (ISR) or *interrupt handler*
   - CPU senses the interrupt request line after executing every instruction.
   - If CPU senses for interrupt request, it saves states, and jump to ISR at a fixed address in memory.
     - Use *interrupt vector table* to dispatch interrupt to correct handler.
3. Interrupt handler **clears** the interrupt by servicing the device
   - The handler determines the cause of the interrupt, performs necessary processing, and executes a return-from-interrupt instruction to return to CPU.

Interrupt-drive I/O cycle



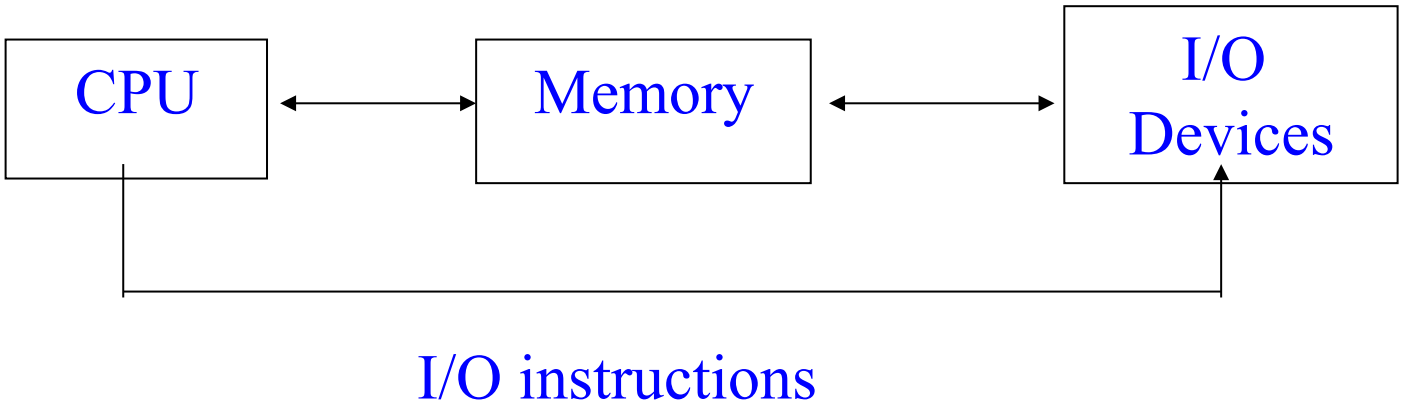
Direct memory access (DMA)

- DMA is used to improve the performance of large data transfers.
  - Use a special-purpose processor (DMA controller).
  - DMA bypasses CPU to transfer data directly between I/O device and memory.
  - Concurrently, CPU goes on with other work.
- CPU must give command to DMA controller, containing information:
  - A pointer to the source/address of the data being transferred,
  - A pointer to the destination/address of the transfer, and
  - A count of the number of bytes to be transferred.
- Handshaking between DMA controller and device controller uses: **DMA-request line** and **DMA-acknowledge line.**
  - Device controller puts a signal on the **DMA-request line** when a word of data is available for transfer.
  - Sensing the signal, DMA controller seizes the memory bus, sets the intended address on address bus, and puts a signal on the **DMA-acknowledge line.**
  - When the device controller receives the **DMA-acknowledge signal**, it transfers the data word to memory, and removes the **DMA-request signal**.
- After finishing the entire transfer, DMA controller interrupts the CPU.

DMA (cont.)

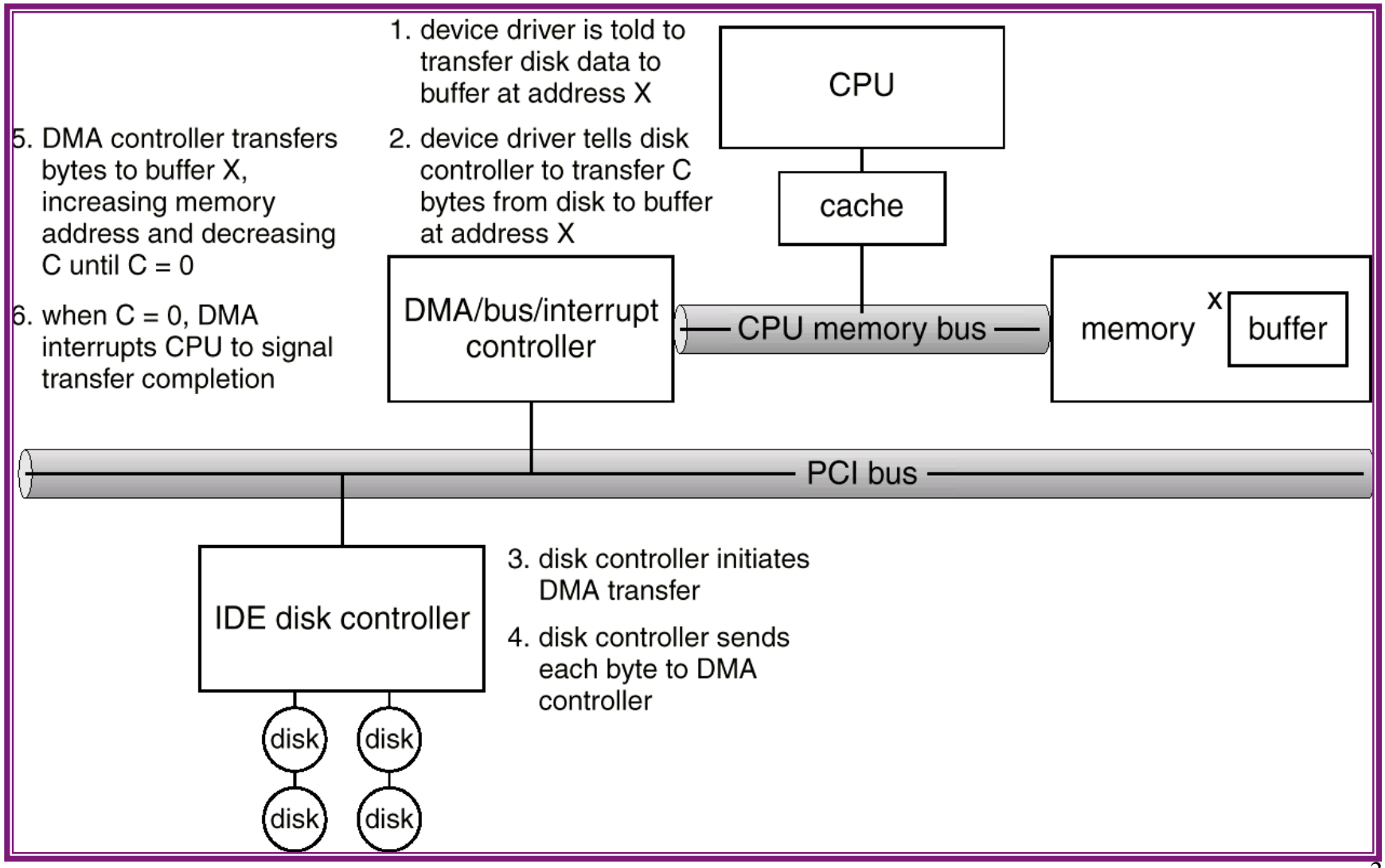
Steps:

- OS (Device driver) sets DMA controller registers to set source and destination addresses, read/write request, and transfer length.
- DMA controller transfers blocks of data from buffer storage directly to main memory without CPU intervention – CPU is free to do other tasks.
- DMA controller interrupts CPU after transfer has completed.
  - Only one interrupt is generated per block rather than one interrupt per byte.



Steps in DMA Transfer

Fig. 13.5 (textbook)



Computer-System Architecture

- A single-processor system contains only one CPU to execute general purpose instructions
  - However, it also contains special purpose processors
    - E.g., disk, keyboard, DMA, graphic controllers
    - These specialized processors do not run user processes
    - OS may be able to manage the processors, e.g., task disk controllers to use given scheduling algorithms.
- A multiprocessor system contains two or more processors working together
  - They may share computer bus, system clock, memory, I/O devices
  - Also called parallel system or multicore system

Multiprocessor (cont.)

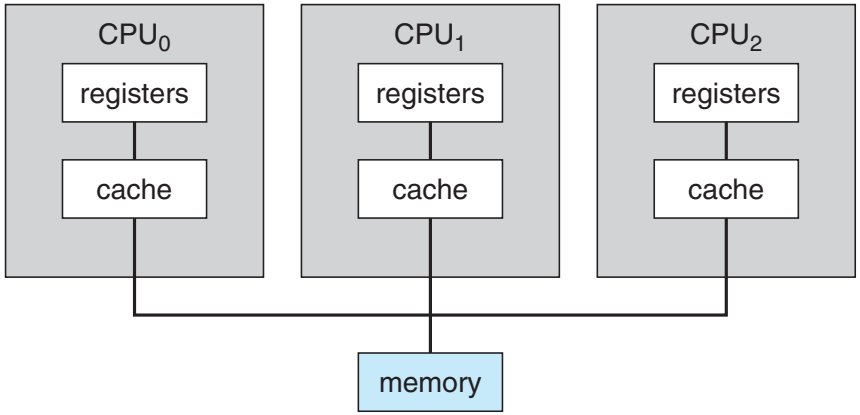
- Three main advantages of multiprocessor system
  - **Increased throughput**: get more work done in less time
    - The speed up in *n* processor system is NOT *n* due to overhead
  - **Economy of scale**: I/O devices, memory storage, and power supplies can be shared
  - **Increased reliability**: failure of one processor does not make the whole system down
    - **Graceful degradation:** the system can perform operations proportional to the level of operations of the surviving parts of the system
    - **Fault-tolerant:** the system can continue its function in the event of component failures
      - Require failure detection, diagnoses, and even correction
      - May use multiple hardware and software duplicates to execute the same tasks in parallel, and take as output the result from the majority of the duplicates

Multiprocessor (cont.)

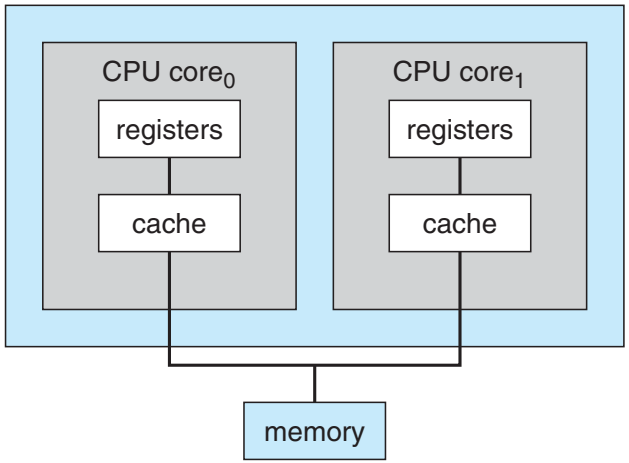
- Two types of multiprocessor system:
  - Asymmetric multiprocessing
  - Symmetric multiprocessing (SMP)
- Asymmetric multiprocessing:
  - Use a **master** processor to schedule and allocate work to (**slave**) processors.
  - Each slave processor waits for instruction from the master or has predefined task
  - More common in extremely large systems

Multiprocessor Systems (cont.)

- SMP – not the master-slave model; a more common system
  - Each processor runs an identical copy of the OS.
  - Each processor has its own registers and cache
    - However, they share the same memory
  - Many processes can run at once without significant performance deterioration
    - Need load balancing to improve performance
- Symmtric and Asymmetric may be the result of either hardware or software.



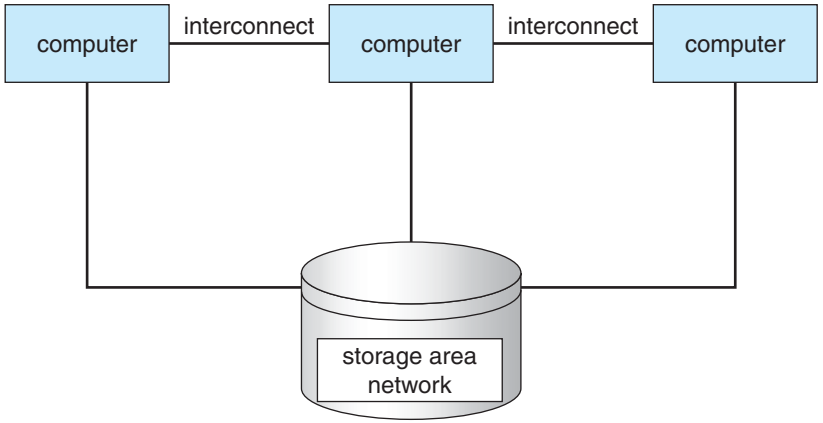
Multiprocessor Systems (cont.)



- Multicore system is a multiprocessor system but the processors are on a single chip
  - More efficient because on-chip communication is faster than inter-chip communication
  - Less power/energy
  - Note: not all multiprocessors are multicores
- Blade servers: multiple processor boards, I/O boards and networking boards are put in the same chassis.
  - Each processor board boots independently and runs its own OS
  - Each processor board can be multiprocessors

Clustered Systems

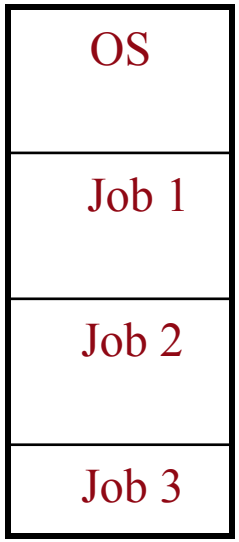
- A clustered system consists of multiple CPUs, like multiprocessors
  - However they are individual systems or nodes
  - Each node can be a single processor or a multicore
  - They share storage and communicate via LAN
  - It offers high-availability service
- Beowulf cluster consists of on-the-self hardware, e.g., personal computers, interlinked by LAN
  - Use open-source software for communication
  - Typically runs LINUX
  - Low cost for high performance computing tasks



Multiprogramming

- Goal: to increase the CPU utilization
- Several jobs are kept in main memory at the same time, and the CPU is multiplexed among them
- It needs these OS features:
  - Job scheduling → OS chooses jobs in the job pool and put them into memory.
  - Memory management → OS allocates the memory for each job.
  - CPU scheduling → OS chooses one among the jobs in memory (called processes) that is ready to run.
  - I/O allocation.

Memory partition



Time-Sharing/Multitasking Systems

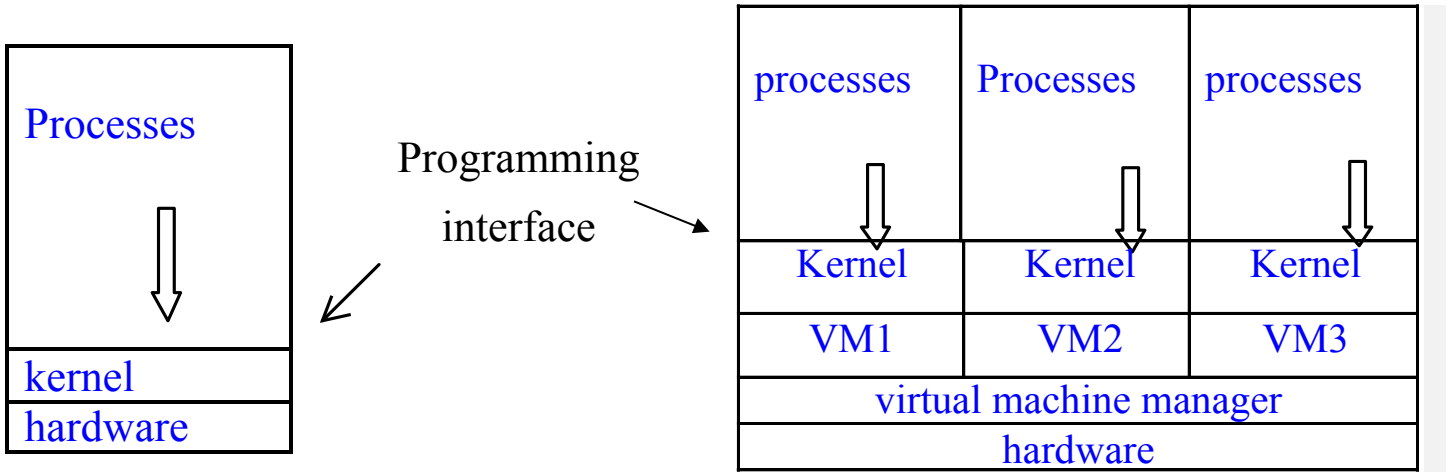
- **Goal:** to provide interactive use of computer system at reasonable cost (one computer – several users/jobs).
- It is a variant of multiprogramming → but user input is from on-line terminal.
- CPU is multiplexed among jobs in memory frequently (≤1 sec?) so that users can interact with their running program.
- It needs these OS features:
  - On-line communication between users and system
    - When OS finishes the execution of one command, it seeks the next *control statement* from the user's keyboard.
  - On-line file system, for users to access data and code.
- Today, most systems provide both batch processing and time sharing.

Real-time Systems

- Well-defined fixed-time constraint – the system is functional if it returns the correct result within the time constraint.
- Hard real-time system.
  - Guarantees that critical tasks complete on time.
  - Often used as a control device in a dedicated application
    - controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems
  - Secondary storage is limited or absent
    - data is stored in short-term memory, or in ROM.
  - Realtime vs. time-sharing systems?
- Soft real-time system
  - A critical-time task gets priority over others until it completes.
  - Limited utility in industrial control robotics.
  - Useful in applications (multimedia) requiring advanced OS features.

Virtualization

- Virtualization technology allows an OS to run as applications on other OS
  - It is a software that includes an **emulation** that is used when the type of source CPU is different from that of the target CPU
    - e.g., it allows applications compiled for a IBM power CPU to run on Intel CPU



OS Operation

- In multiprogramming, OS must ensure that an incorrect (or malicious) program cannot cause other programs to execute incorrectly.
- Many programming errors are detected by the hardware, and the errors are normally handled by the OS.

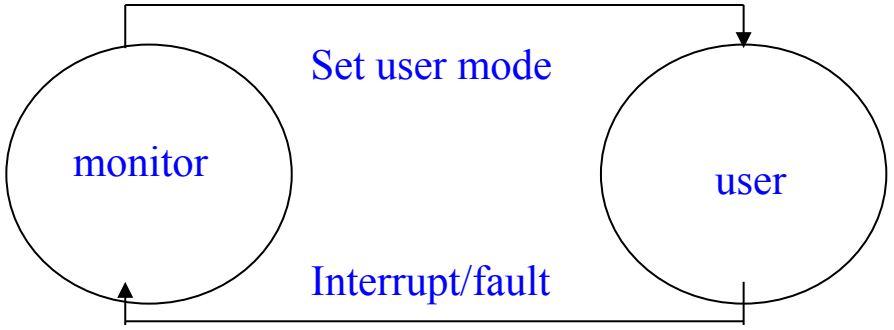
Dual-mode operation

- Hardware provides at least two modes of operations:
  - User mode – in which user programs run.
  - Monitor mode, also called Supervisor, System, or privileged mode.
- Mode bit is provided by the hardware to indicate the current mode: monitor (0) or user (1).

OS Operation (cont.)

Dual-mode operation (cont.)

- Some machine instructions that may cause harm are designated (by hardware) as **privileged instructions**.
  - E.g., the MSB of the machine code of the instruction is a bit '0'
- A privileged instruction can be executed only in monitor mode.
- At system boot-time the hardware starts in monitor mode → OS is loaded.
- OS starts user processes in user mode; a user process could never gain control of the computer in monitor mode.
- When an interrupt or fault occurs hardware switches to monitor mode.



OS Operation (cont.)

I/O

- All I/O instructions are privileged instructions.
- How does the user program perform I/O ? Use system call.
  - Usually takes the form of a trap to a specific location in the interrupt vector.
  - Control passes through the interrupt vector to a service routine in the OS, and the mode bit is set to a monitor mode.
  - The monitor verifies that the parameters are correct and legal, executes the request, and returns control to the instruction following the system call.

OS Operation (cont.)

Memory

- System must provide memory protection at least for the interrupt vector and the Interrupt Service Routine.
- Use two registers that determine the range of legal addresses a program may access:
  - Base register – holds the smallest legal physical memory address.
  - Limit register – contains the size of the range.
- Memory outside the defined range is protected.
- The load instructions for the base and limit registers are privileged instructions.
- OS has unrestricted access to monitor and user memory.

OS Operation (cont.)

CPU

- System must prevent one user program using CPU all the time
  - Because of getting stuck in an infinite loop, or non-fair users
- Use *timer* interrupt after specified period to ensure OS maintains control
  - Timer is decremented every clock tick.
  - When timer reaches value 0, an interrupt occurs.
- Timer is commonly used to implement time sharing.
  - Timer is also used to compute the current time.
  - Loading timer value is a privileged instruction. Why?

System Calls

- A system call is a user interface to the OS services
  - Available in assembly-language instructions or in high level languages for systems programming (e.g., C)
    - E.g., fork () is a system call to ask OS to create a new process
  - Application Program Interface (API): a set of functions available to an application programmer.
    - An API typically invokes the actual system calls for the application programmers.
    - Examples of API: Windows API, POSIX API, Java API
    - API is more portable and simpler to use

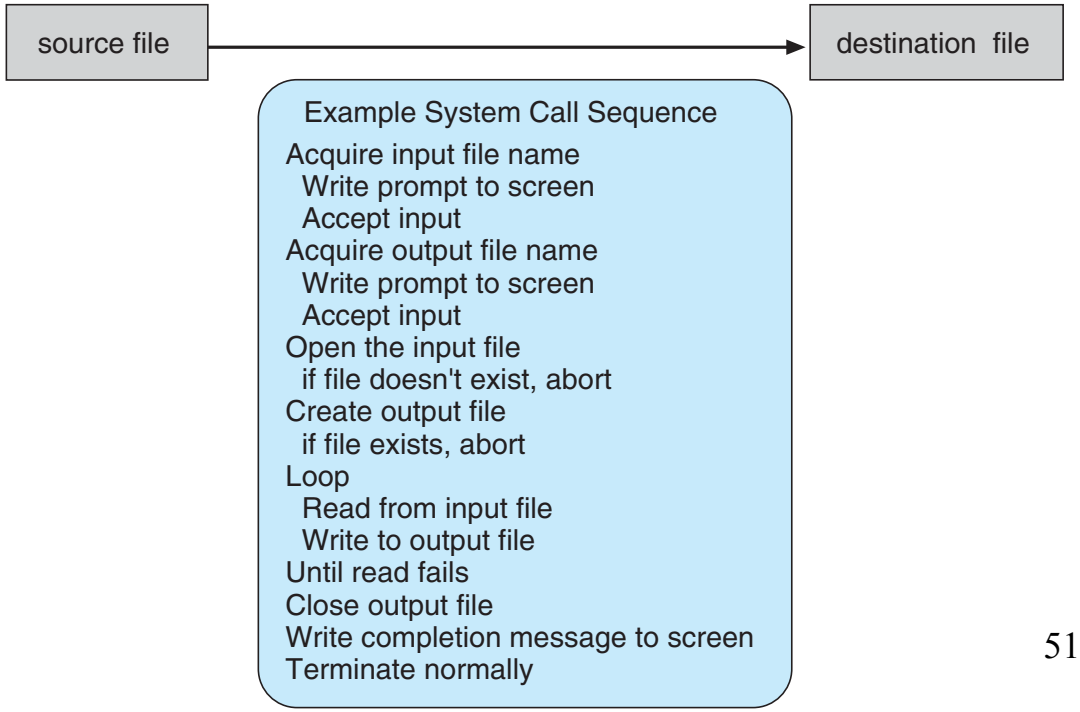
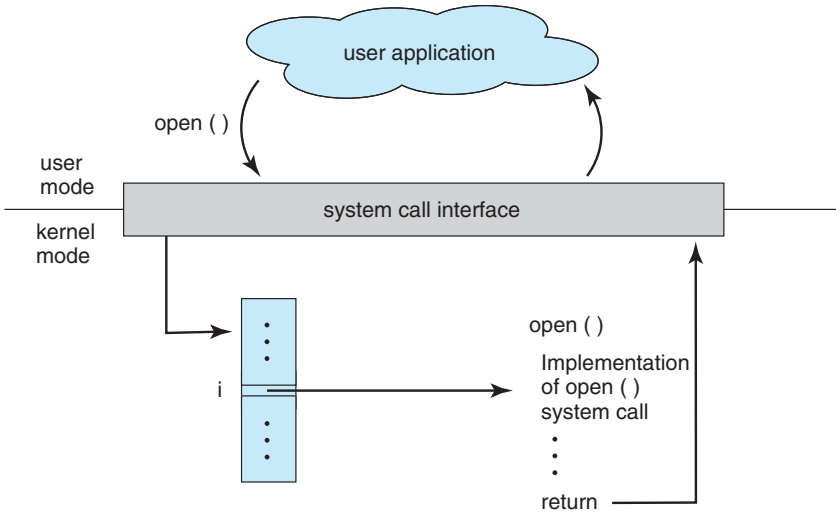


Figure 2.5 (Textbook)

System Calls (cont.)

- A system call is known by its number
  - System call interface maintains a vector table which stores pointers to system call routines
    - System call interface intercepts each API and calls all corresponding system calls to handle the API
- System call result/termination.
  - Normal termination (exit or return).
  - Terminate current program and return to the command interpreter.
  - Abnormal termination (trap) — program error.



How to pass parameters to OS?

- In registers
- In a block of memory or table – for many parameters
- In a stack

Figure 2.6 (Textbook)

System Calls (cont.)

Types of system calls

1) Process control
   - End, abort (running program); Load, execute; Create, terminate; Signal event, etc.
   - Examples: fork(), exit(), wait(), etc.
2) File management
   - Create, delete; Open, close; Read, write, reposition; Get and set file attributes, etc.
   - Examples: open(), read(), write(), close(), etc.
3) Device management (memory, tape drives etc.)
   - Request device; Release device; Read, write, reposition, etc.
   - Examples: ioctl(), read(), write(), etc.
4) Information maintenance
   - Get or set time or date; Get or set process attributes, etc.
   - Examples: getpid(), alarm(), sleep(), etc.
5) Communications
   - Create, delete communication connection; Send and receive messages, etc.
   - Examples: pipe(), shmget(), mmap(), etc.
6) Protection
   - Resource access control
   - Examples: chmod(), umask(), chown(), etc.

System programs or utilities

- Provide a convenient environment for program development and execution
  - Most users' view of the OS is defined by system programs, not the actual system calls.
  - Some of them are simple user interface to system calls.
- System programs can be divided into:
  - File manipulation: Create, delete, copy, rename, print, dump, list.
  - Status information: Date, time, memory, disk space, number of users.
  - File modification: Text editors to create and modify content of files.
  - Programming-language support: Compilers, interpreters, assemblers.
  - Program loading and execution: Absolute, Relocatable, Overlay loaders.
  - Communication: Programs that provide the mechanism for creating virtual connections among processes, users, and different computer systems.

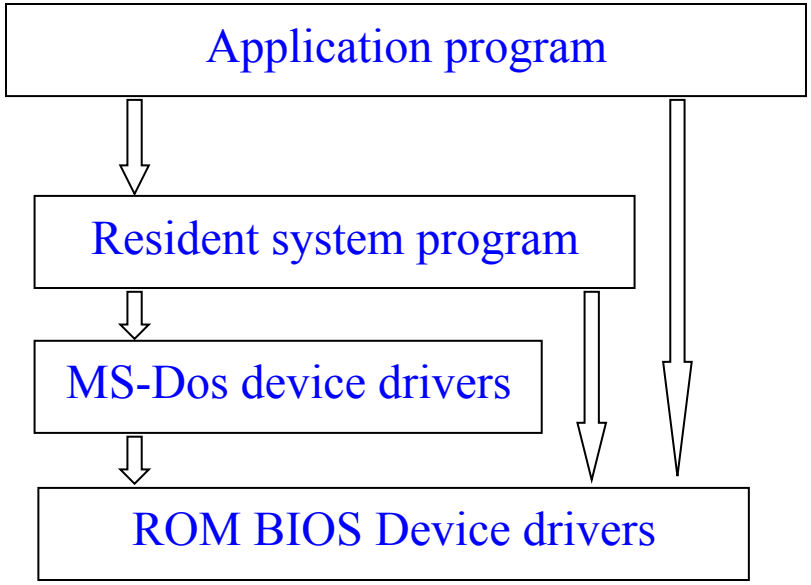
OS Structures

- A system as large and complex as a modern OS must be designed carefully such that it can function properly and be modified easily.
- Various structures:
  - Simple Structure/Monolithic
  - Layered Approach
  - Microkernels
  - Modules

Simple Structure

- No well-defined structures.
- MS-DOS - written to provide the most functionality in the least space.
- It is not divided into modules
- Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated

MS-DOS layer structure



UNIX system structure

(the users)		
•Shells and commands •Compilers and interpreters •System libraries		
System call interface to the kernel		
Signals terminal handling Character I/O system Terminal drivers	File system Swapping block I/O Disk/tape drivers	CPU scheduling Page replacement Demand paging Virtual memory
Kernel interface to the hardware		
Terminal controllers Terminals	Device controllers Disks and tapes	Memory controllers Physical memory

- The original Unix OS was limited by hardware functionality
  - it had limited structuring.
- The UNIX OS consists of two separable parts:
  - System programs
  - Kernel

- Kernel consists of everything below the *system-call interface* and above the *physical hardware*
  - It provides the file system, CPU scheduling, memory management, and other operating-system functions;
  - It contains many functions in one level → difficult to implement and maintain.

Layered approach

- The OS is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0) is the hardware, and the highest (layer N) is the user interface.
- With modularity, layers are selected such that each layer uses functions (operations) and services of only lower-level layers.
- A layered design was first used in the THE (Technische Hogeschool Eindhoven) operating system.
- **Advantage:** Modularity → easy to debug
- **Disadvantage:** less efficient, hard to define level functionality.

Six layers

Layer 5: User programs
Layer 4: Buffering for input and output devices
Layer 3: Operator-console device driver
Layer 2: Memory management
Layer 1: CPU scheduling
Layer 0: Hardware

Microkernel

- Smaller-sized kernel → only provides minimal services such as process and memory management, and communication.
- Put non-essential components of kernel as system/user programs
- Kernel function: as a communication facility between a client and the OS services using message passing.

Advantages:

- Easier to extend OS
- More portable
- Better security and reliability

Disadvantages:

- Using message passing increases system function overhead → reduce performance

Examples: Mach (CMU), Mac OS X kernel (Darwin), Digital Unix,

Modules

- Modular kernel - current method
  - Use Object-oriented programming techniques
- Kernel has a set of core components.
- Add additional modules (e.g., Scheduling, device drivers, file systems) during boot/run time to kernel.
- Like a layered system but more flexible
  - Any module can call any other module
- Like a microkernel but more efficient
  - Modules do not need message passing

Examples: Solaris, Linux, Mac OS X.