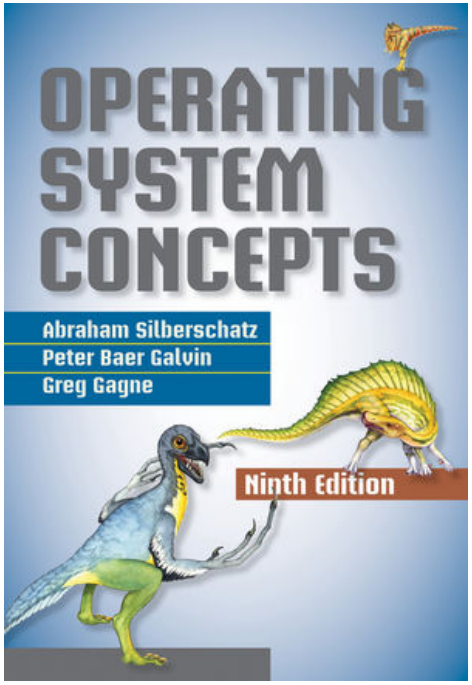
2 – System Architecture

EECE 315 (101)
ECE – UBC
2013 W2



Acknowledgement: This set of slides is partly based on the PPTs provided by the Wiley's companion website (including textbook images, when not explicitly mentioned/referenced).

Outline

- What Operating Systems (OS) Do
- Computer System
  - Architecture and Organization
- Operating-System Structure and Operations
  - Process Management
  - Memory Management
  - Storage Management
- OS Services and System Calls

Name a Few Operating Systems?

Google Chrome OS

Images sources: wikipedia.org
(Wikimedia Commons License)

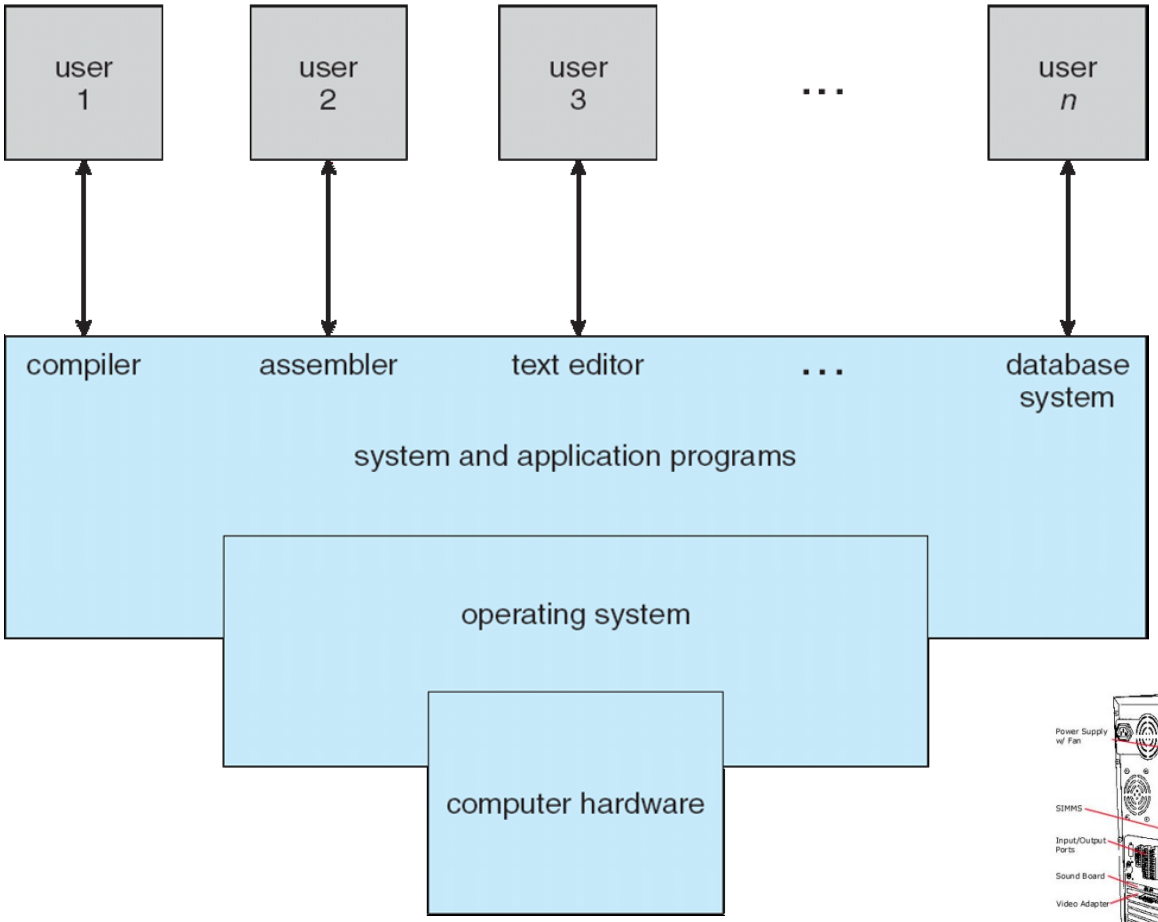
So … What is an Operating System?

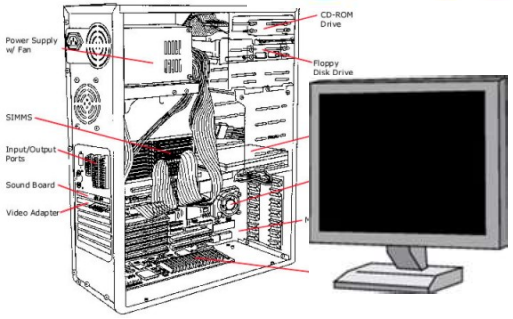
- Remember the projects you did in EECE 259
  - What did you need in your programs for a microcontroller?
  - Now think again …. What about the general-purpose computer or the smart-phone you use on a daily basis?

- An operating system is
  - a program
  - that acts as an intermediary between a user of a computer and the computer hardware
  - and provides an environment in which a user can execute programs.

- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient to use
  - Use the computer hardware in an efficient manner

OS's Role in the Overall Computer System

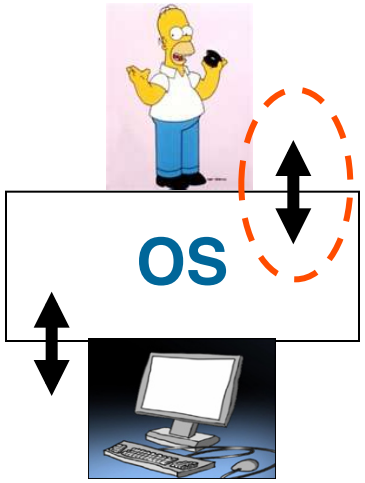




Four Components of a Computer System

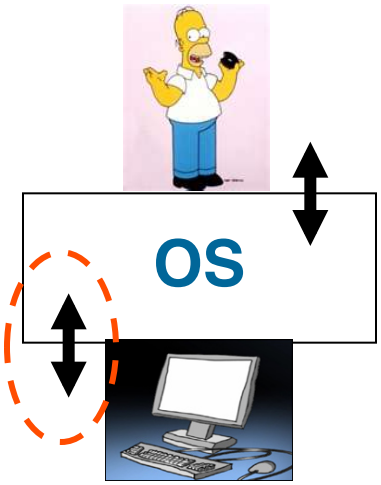
Operating System Definition

- **User View**:
  - User's view varies according to the interface
  - OS is a **control program**
  - It controls execution of programs, prevents errors and improper use of the computer
  - For a user in front of a
    - *PC* (single user): ease of use and performance
    - *terminal of a mainframe* (many users): maximize resource utilization
    - *network connected workstation*: mixed of usability and resource utilization
    - *handheld* (limited power, speed and interface): personal usability and performance (e.g., battery life)



Operating System Definition (cont)

- **System View**:
  - OS is a **resource allocator**
  - Manages the hardware and all resources (CPU, Memory, I/O, …)
  - Decides between conflicting requests for efficient and fair resource use

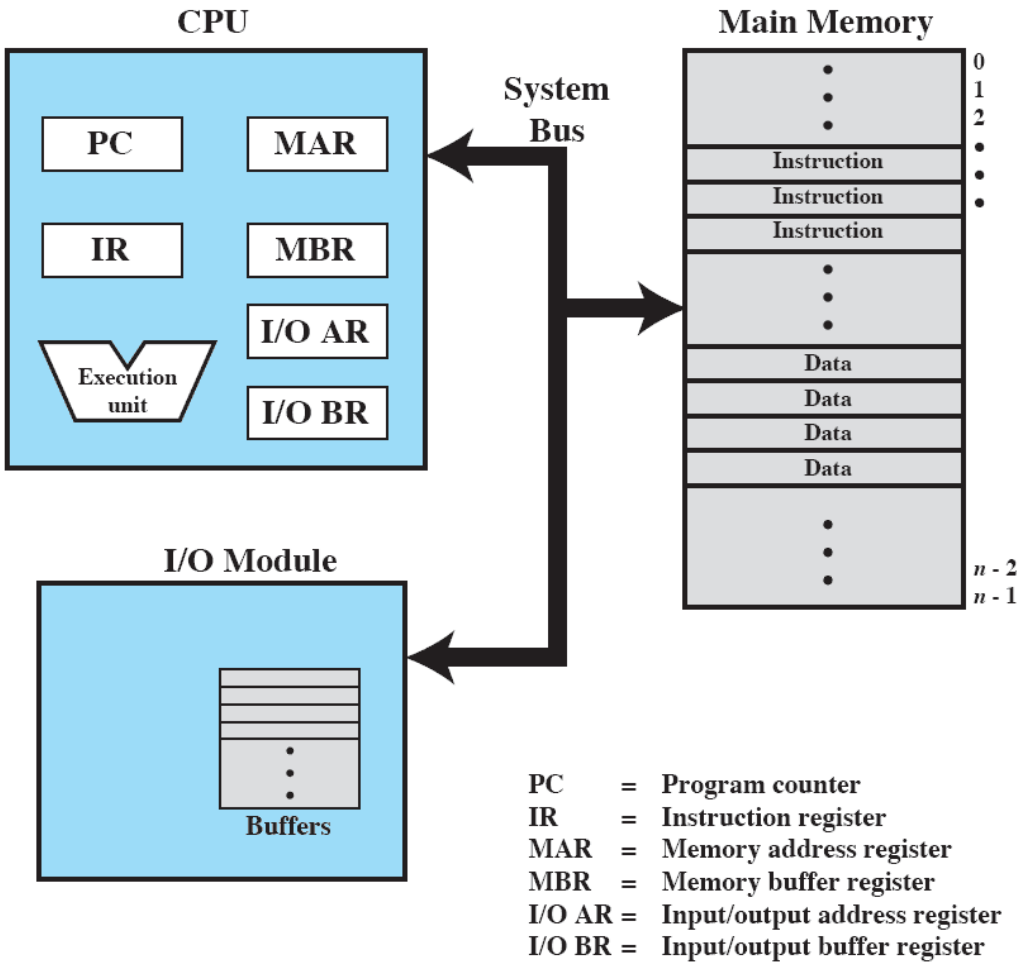


- In general, no completely adequate, universally accepted definition
- "The one program running at all times on the computer" is the **kernel.** Everything else is either a *system program* (ships with the operating system) or an *application program*.

Outline

- What Operating Systems (OS) Do
- Computer System
  - Architecture and Organization
- Operating-System Structure and Operations
  - Process Management
  - Memory Management
  - Storage Management
- OS Services and System Calls

Computer Components

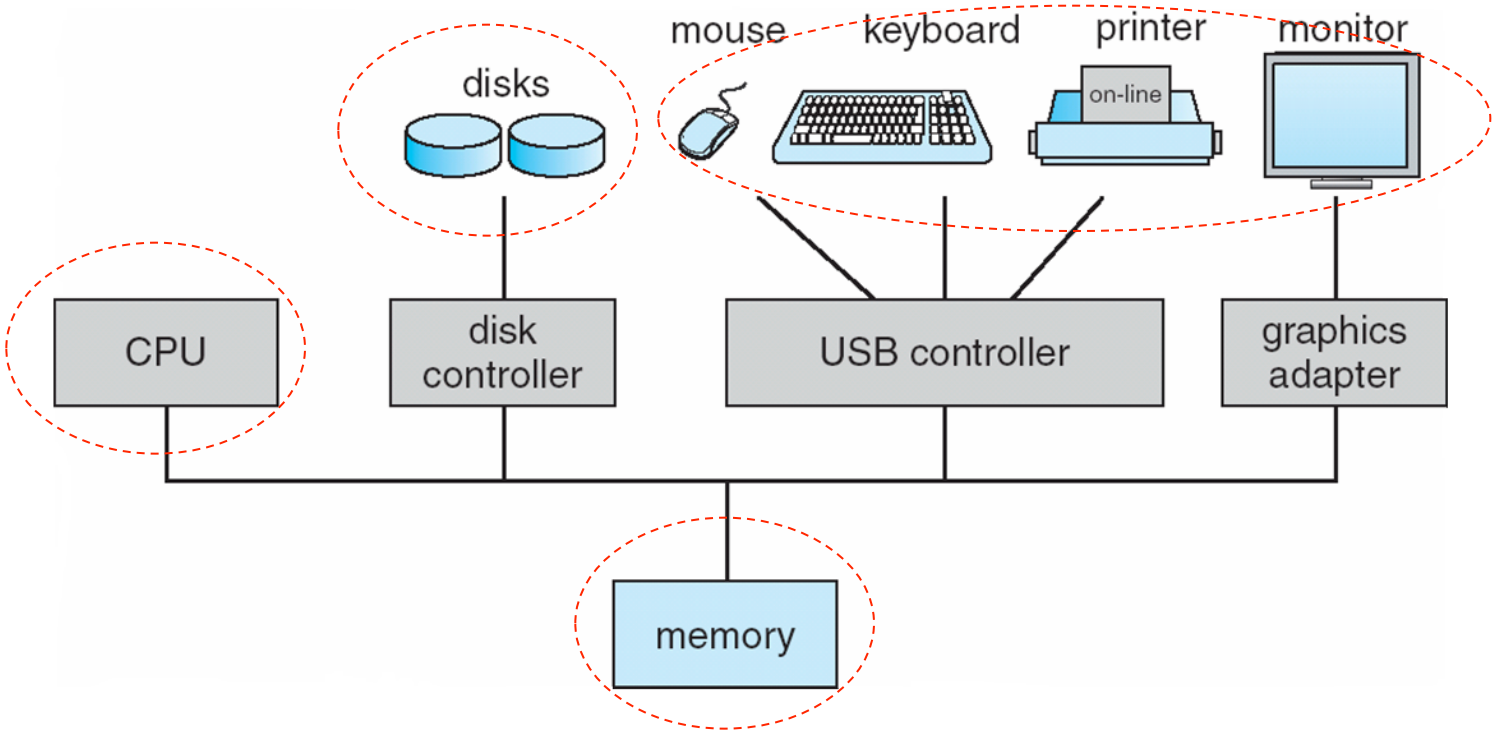


Computer Components: Top-Level View

source: Stallings'

Computer System Organization

- Computer-system operation
  - One or more CPUs, device controllers connect through common bus providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles



Computer-System Operation

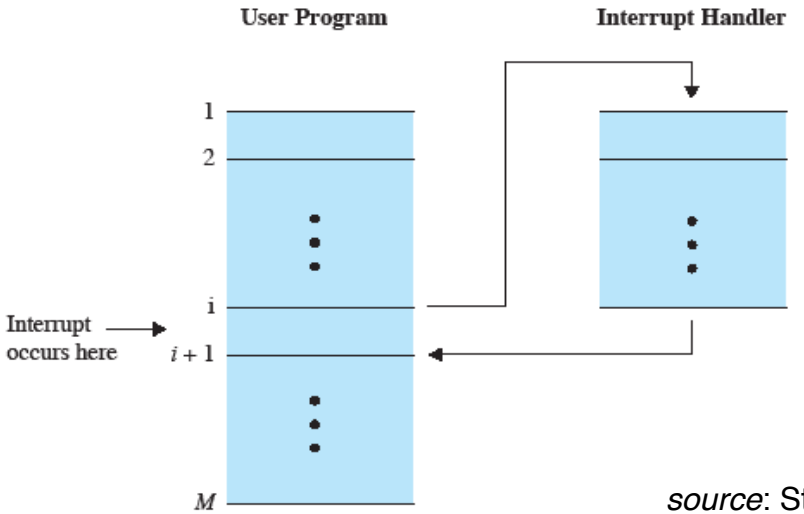
- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer

- 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*

Common Functions of Interrupts

- An operating system is **interrupt driven**
  - hardware may trigger an interrupt at any time, e.g. by sending a signal to the CPU
  - Software may trigger an interrupt by executing a special operation called a **system call**
  - A *trap* (or *exception*) is a software-generated interrupt caused either by an error or a user request

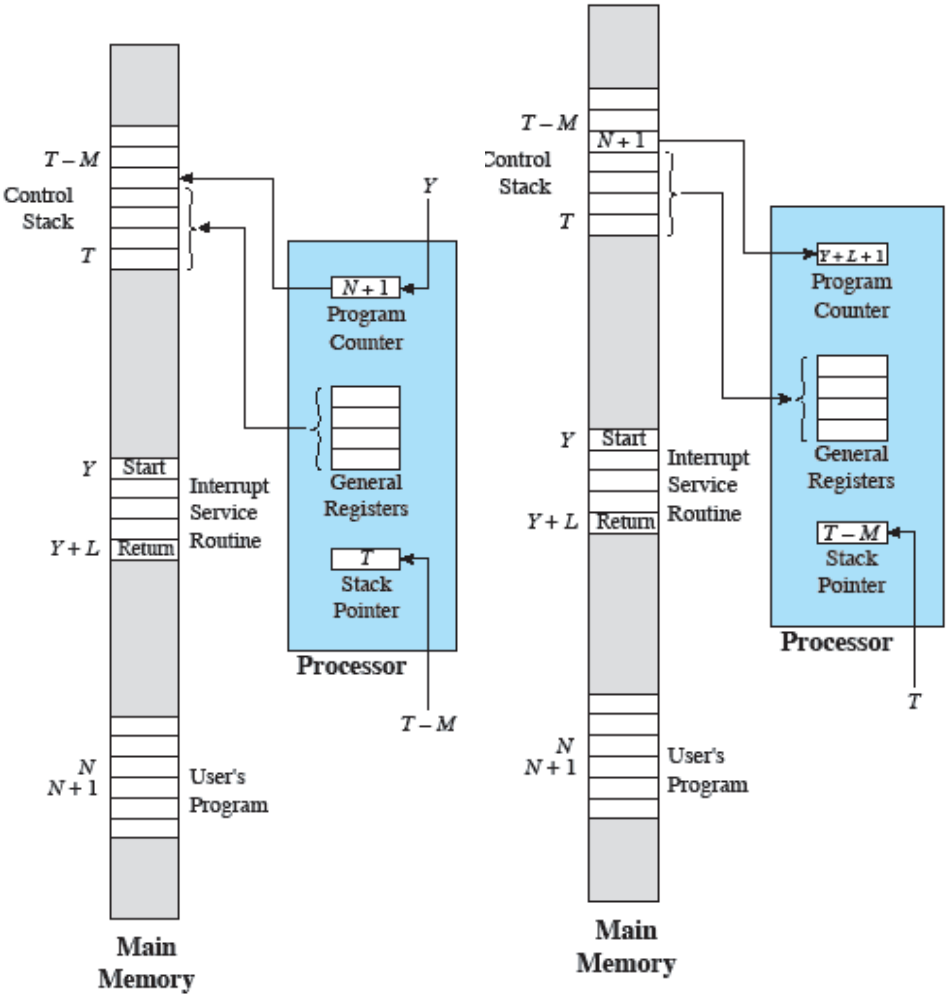


source: Stallings'

Transfer of Control via Interrupts

Common Functions of Interrupts (cont)

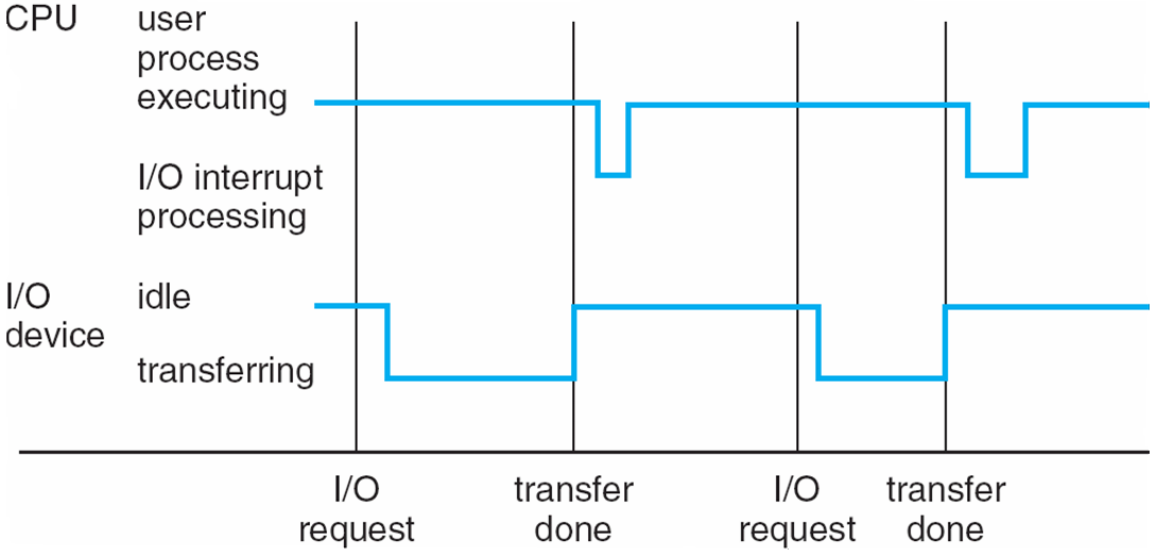
- Interrupt transfers control to the interrupt service routine generally, through the **interrupt vector**, which contains the addresses of all the service routines
- The operating system preserves the state of the CPU by storing registers and the program counter
- Interrupt architecture must save the address of the interrupted instruction
- Incoming interrupts may be *disabled* while another interrupt is being processed to prevent a *lost interrupt*



Interrupt Handling

- When a CPU is interrupted, it stops what it is doing and transfers execution to a fixed location.
- Determines which type of interruption has occurred:
  - polling
  - **vectored** interrupt system
- Separate segments of code determine what action should be taken for each type of interrupt

Interrupt Timeline:

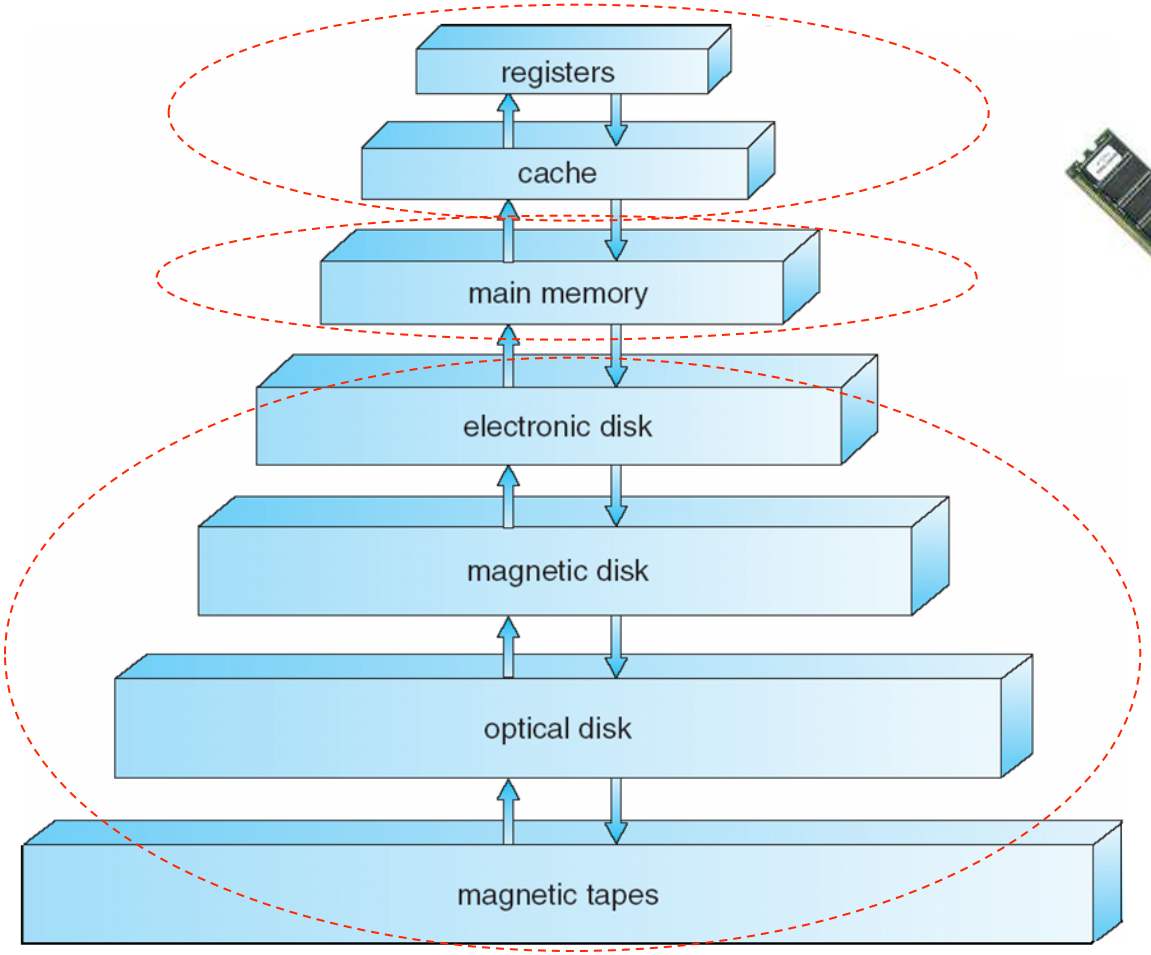


Computer Startup

- For a computer to start running (when powered up or rebooted), it needs to have an initial program to run.

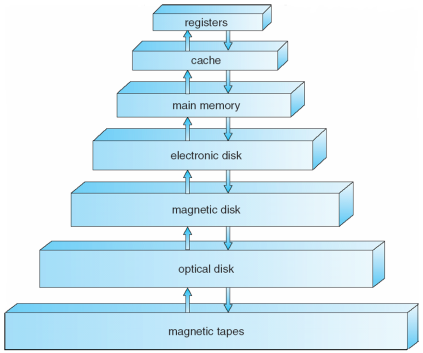
- **bootstrap program** is loaded at power-up or reboot
  - Typically stored in ROM or EEPROM, generally known as **firmware**
  - Initializes all aspects of system (CPU registers, memory contents and check, I/O, …)
  - Loads operating system kernel and starts its execution

Storage-Device Hierarchy



Storage Hierarchy

- Storage systems organized in hierarchy based on:
  - Speed
  - Cost
  - Volatility

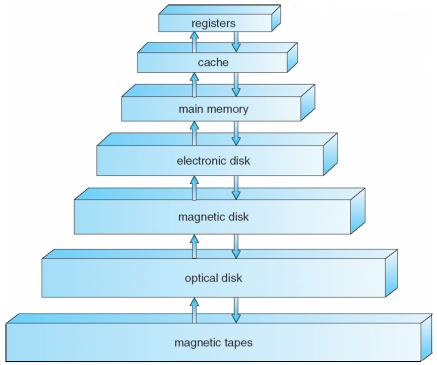


- In the previous figure: the higher levels are expensive, but they are fast.

- The design of a complete memory system must balance all factors:
  - using only as much as expensive memory as necessary while providing as much inexpensive, nonvolatile memory as possible.

Caching

- **Caching**
  - Information in use is copied from slower to faster storage temporarily
  - an important principle, performed at many levels in a computer (in hardware, operating system, software)
  - main memory can be viewed as a last *cache* for secondary storage
- Faster storage (cache) checked first 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
- Cache is smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy



Storage Structure

- **Main memory** is
  - the only large storage media that the CPU can access directly
  - too small (expensive, system limitations) to store all needed programs/ data
  - a volatile storage (i.e. loses its contents when the power is removed)
- **Secondary storage** is
  - extension of main memory that provides large nonvolatile storage capacity
- Magnetic disks – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into **tracks**, which are subdivided into **sectors**
  - The **disk controller** determines the logical interaction between the device and the computer

Semiconductor Memory

- The top four levels of memory in the previous figure may be constructed using semiconductor memory.

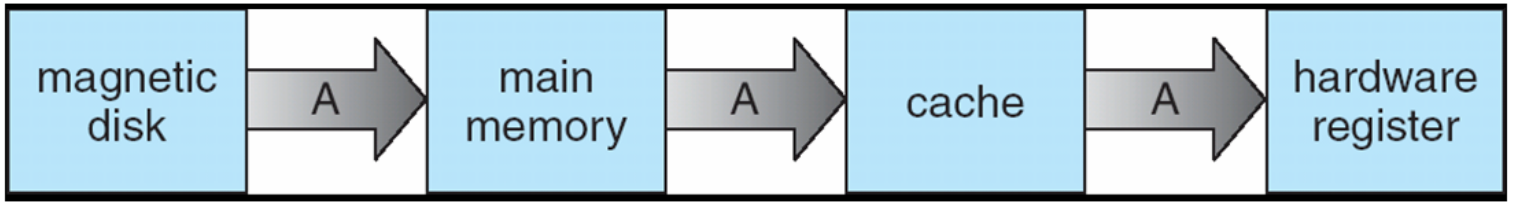
- **Electronic disk** can be either volatile or nonvolatile.
  - large DRAM array (volatile)
  - Flash memory (nonvolatile)
  - NVRAM (DRAM with battery backup, nonvolatile)

Performance of Various Levels of Storage

- Movement between levels of storage hierarchy can be explicit or implicit

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

- Example: migration of integer A from disk to register

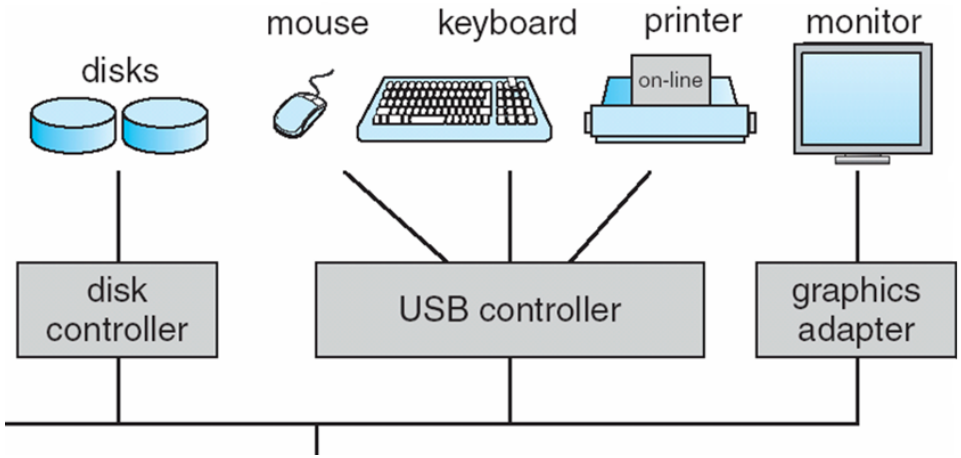


I/O Structure

- Storage is only one of many types of input/output devices.
- One purpose of OS is to hide peculiarities of hardware devices from the user
- A **device controller**
  - is in charge of a specific type of device and
  - is responsible for moving the data between the peripheral devices that it controls and its local buffer storage.

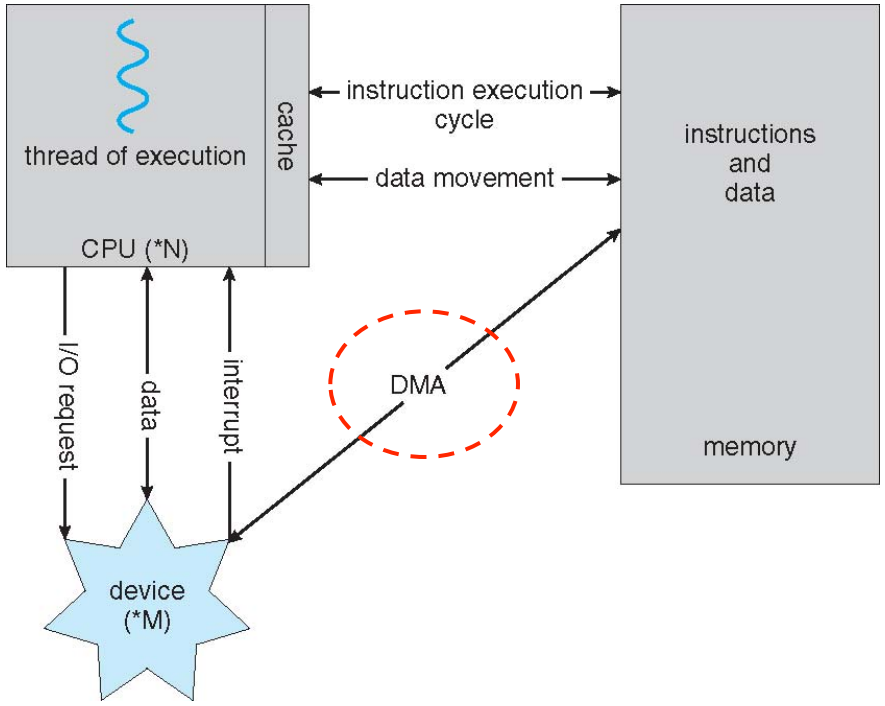
I/O Structure (cont.)

- Typically, operating systems have a **device driver** for each device controller.
  - The device driver understands the device controller and presents a uniform interface to it.
  - The device controller may inform the device driver via an interrupt that it has finished its operation.



Direct Memory Access Structure

- The interrupt-driven I/O is fine for moving small amount of data, but for bulk data transfer it can produce high overhead. DMA is used to solve this problem.
  - DMA is Used for high-speed I/O devices able to transmit information at close to memory speeds
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
  - Only one interrupt is generated per block, rather than the one interrupt per byte

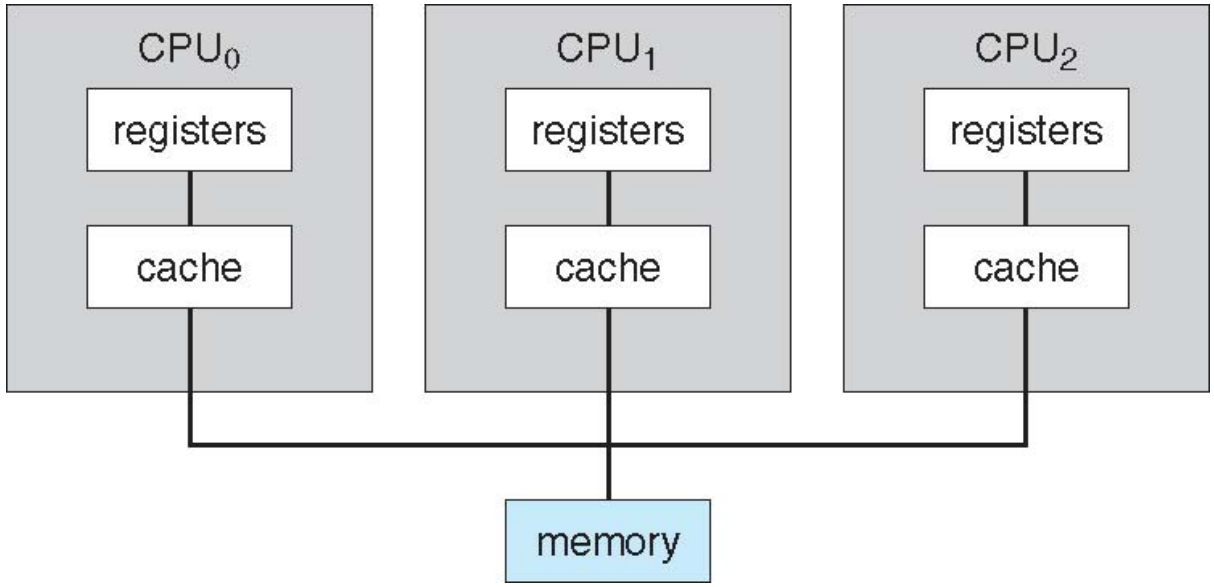


Computer-System Architecture

- Many systems use a single general-purpose processor (PDAs through mainframes)
  - Most systems have special-purpose processors as well
  - Operating system may or may not manage these special-purpose processors. (e.g. disk controller microprocessor and low-level microprocessor built into the hardware)
- Multiprocessors systems growing in use and importance
  - Have two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices.
  - Have three advantages:
    - Increased throughput
    - Economy of scale
    - Increased reliability (graceful degradation or fault tolerance)

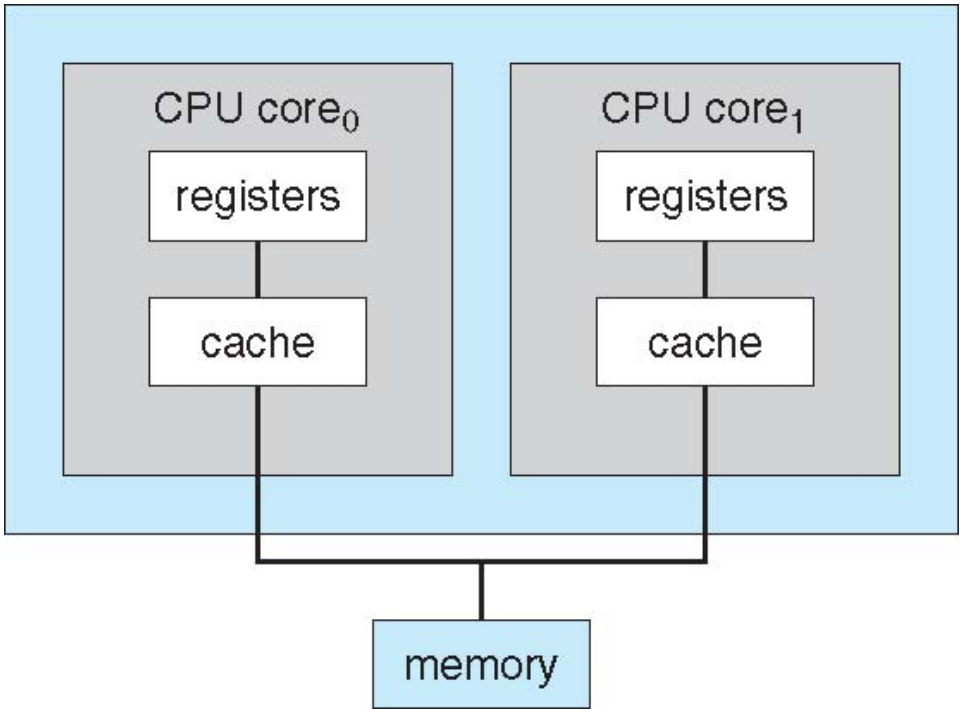
Multiprocessing Architecture

- There are two types of multiple-processor systems:
  - **Asymmetric Multiprocessing**:
    - Each processor is assigned a specific task. A master processor controls the system and other processors either look at the master or have predefined tasks.
  - **Symmetric Multiprocessing** (SMP):



Multi-core Processors

- A recent trend in CPU design is to include multiple computing **cores** in a single chip.
- Some Advantages:
  - On-chip communication is faster than between-chip communication.
  - uses significantly less power than multiple single-core chips.



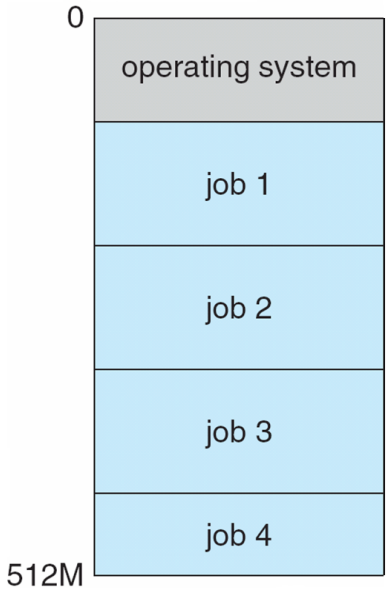
Outline

- What Operating Systems (OS) Do
- Computer System
  - Architecture and Organization
- Operating-System Structure and Operations
  - Process Management
  - Memory Management
  - Storage Management
- OS Services and System Calls

Operating System Structure

- One important aspect of operating systems is the ability to multiprogram.

- **Multiprogramming** increases CPU utilization and is needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - The jobs are kept initially on the disk in the **job pool**.
  - A subset of total jobs in the job pool is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job



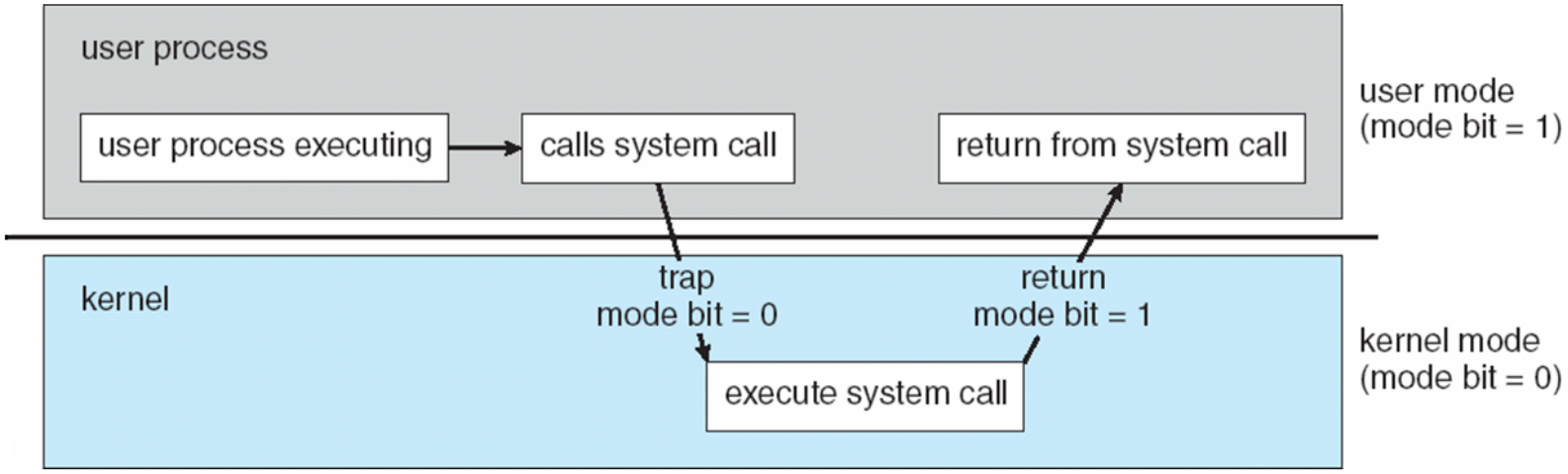
Operating System Structure (cont)

- **Timesharing (multitasking)**
  - is logical extension of multiprogramming
  - CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - **Response time** should be < 1 second
  - Each user has at least one program executing in memory ⇨**process**
  - If several jobs ready to run at the same time ⇨ **CPU scheduling**
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory
    - allows running programs that are larger than actual physical memory
    - frees programmers from concern over memory-storage limitations

Dual-mode Operation

- In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of OS code and user-defined code.
- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
    - When the computer is executing on behalf of a user application, the system is in user mode.
    - When a user application requests a service from the operating system (system call), a transition from user to kernel mode must be made. The system switches to user mode before passing control back to a user program.
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as **privileged**, only executable in kernel mode

Transition from User to Kernel Mode



- We must ensure that the OS maintains control over the CPU.
- A **timer** can be used to prevent infinite loop / process hogging resources
  - Set interrupt after specific period
  - Operating system decrements counter
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time

Process Management

- A process is a program in execution. It is a unit of work within the system. A program is a *passive entity*, process is an *active entity*.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - These resources are either given to a process when created or while running
  - When the process terminates, the OS reclaims any reusable resource
- The operating system is responsible for the following activities in connection with **process management**:
  - 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

- Main memory is central to the operation of a computer system:
  - All data in memory before and after processing
  - All instructions in memory in order to execute
- Memory management determines what is in memory and when
  - Optimizing CPU utilization and computer response to users
- **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 OS abstracts from the physical properties of its storage devices to define a logical storage unit, the **file**.

- Each medium is controlled by device (i.e., disk drive, tape drive)
  - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)

- File-System management
  - Files usually organized into directories
  - Access control on most systems to determine who can access what
  - OS activities include
    - 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

- Disks are used as the principal on-line storage medium for both programs and data
  - That does not fit in main memory or that must be kept for a “long” period of time
  - Proper management is of central importance
- OS activities in connection with **disk management**
  - Free-space management
  - Storage allocation
  - Disk scheduling
- Some storage need not be fast
  - Tertiary storage includes optical storage, magnetic tape
  - Still must be managed
  - Varies between WORM (write-once, read-many-times) and RW (read-write)

Protection and Security

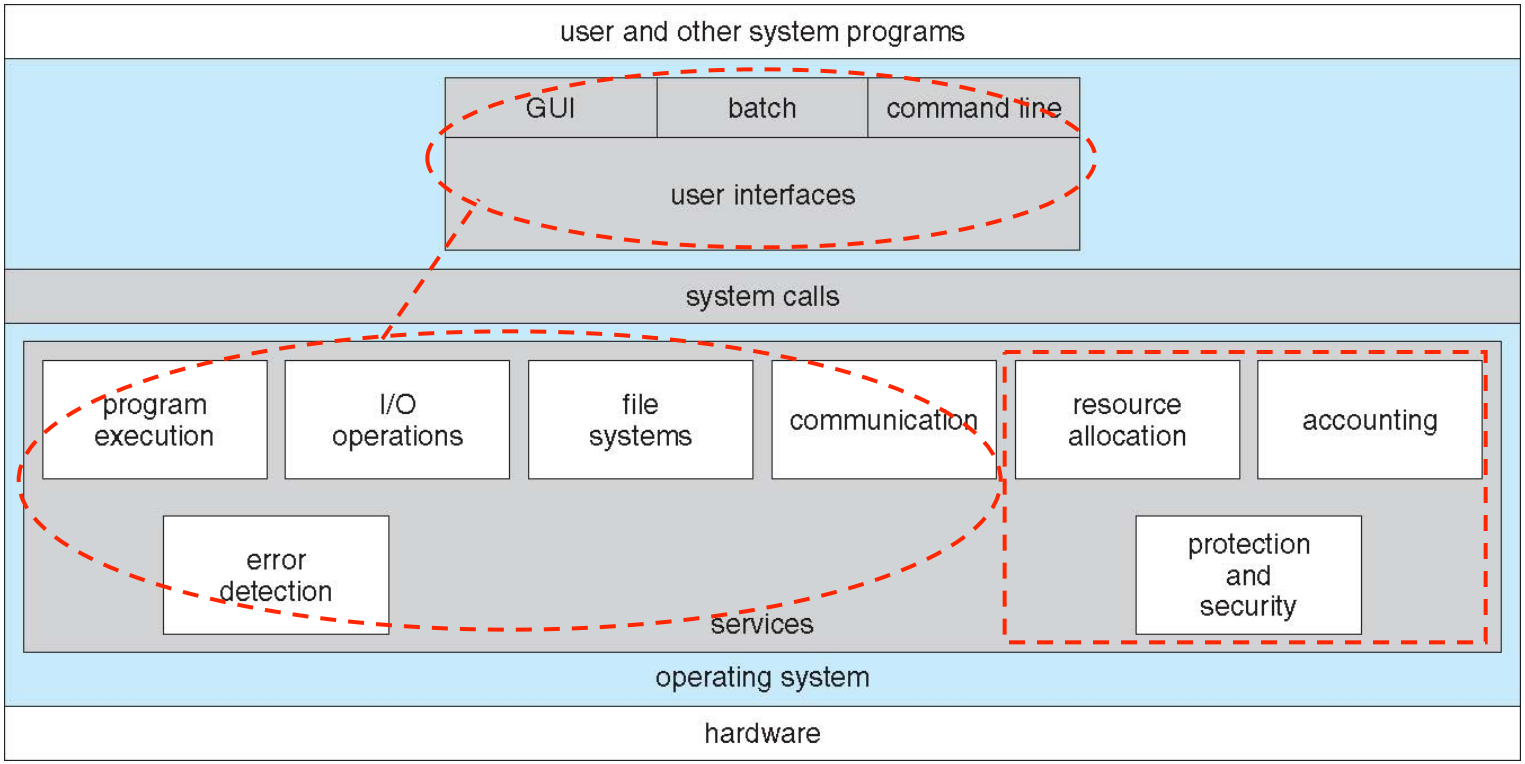
- If a computer system has multiple users and allows the concurrent execution of multiple processes, then access to data must be regulated.
  - **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
  - **Security** – defense of the system against internal and external attacks
    - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
  - User identities (**user IDs**, security IDs) include name and associated number, one per user
  - User ID 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 controls managed, then also associated with each process, file
  - **Privilege escalation** allows user to change to effective ID with more rights

Outline

- What Operating Systems (OS) Do
- Computer System
  - Architecture and Organization
- Operating-System Structure and Operations
  - Process Management
  - Memory Management
  - Storage Management
- OS Services and System Calls

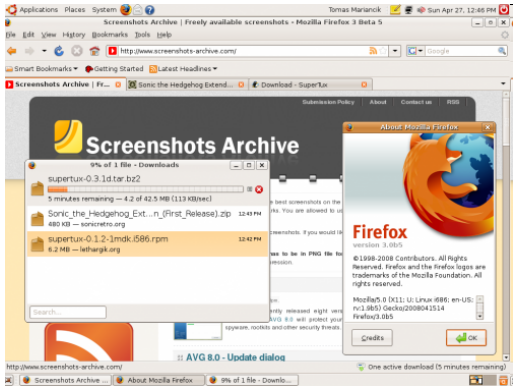
A View of Operating System Services

- As an environment for the execution of programs, an OS provides certain services to the programs and the users of those programs.



Operating System Services

- One set of OS services provides functions that are helpful to the user:
  - User interface
    - Command-Line (CLI),
    - Graphical User Interface (GUI),
    - and also Batch



  - Program execution
    - loading a program into memory and to run that program,
    - ending execution (either normally or abnormally (indicating error))
  - I/O operations - A running program may require I/O, which may involve a file or an I/O device
  - File-system manipulation
    - Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

Operating System Services (Cont)

- (Cont):
  - Communications – Processes may exchange information, on the same computer or between computers over a network
    - Communications may be via
      - shared memory or
      - message passing (packets moved by the OS)

  - Error detection – OS needs to be constantly aware of possible errors
    - May occur in the CPU and memory hardware, in I/O devices, in user program
    - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
    - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

Operating System Services (Cont)

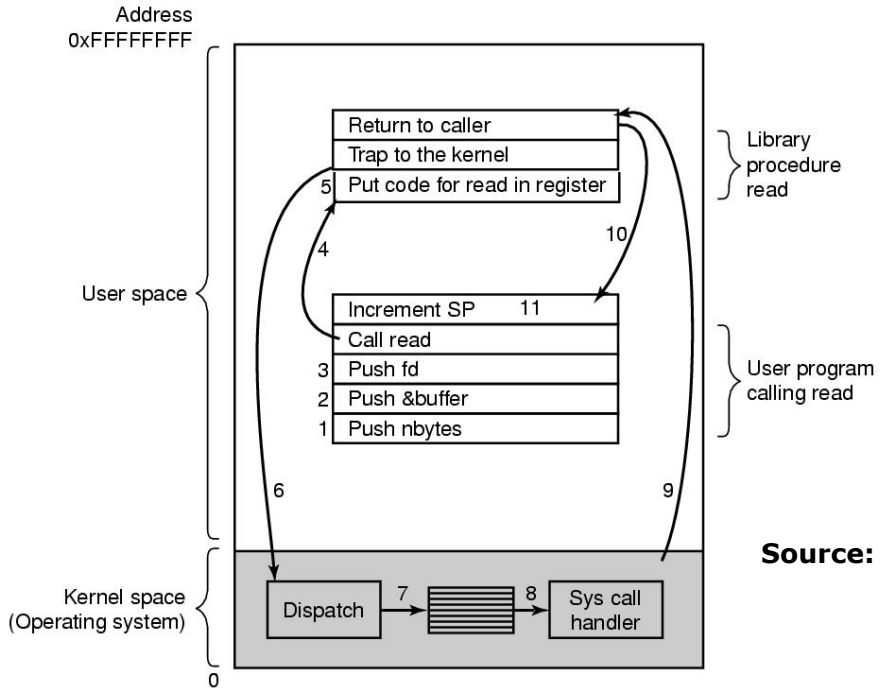
- Another set of OS functions exists not for helping the user but rather to ensuring the efficient operation of the system itself.
  - Resource allocation
    - When *multiple* users or *multiple* jobs running concurrently, resources must be allocated to each of them
    - Many types of resources are managed by OS.
      - e.g. CPU scheduling or request/release to allocate an I/O device
  - Accounting
    - To keep track of which users use how much and what kinds of computer resources

Operating System Services (Cont)

- (Cont):
  - Protection and security - The owners of information may want to **control use** of that information, or concurrent processes should not interfere with each other
    - If a system is to be protected and secure, precautions must be instituted throughout it.
      - A chain is only as strong as its weakest link.
    - *Protection* involves ensuring that all access to system resources is controlled
    - *Security* of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts

System Calls

- **System calls** provide programming interface to the services provided by the OS
  - Typically available as routines written in a C or C++ (a high level language and maybe some assembly)



Example - The steps in making the system call `read()`:

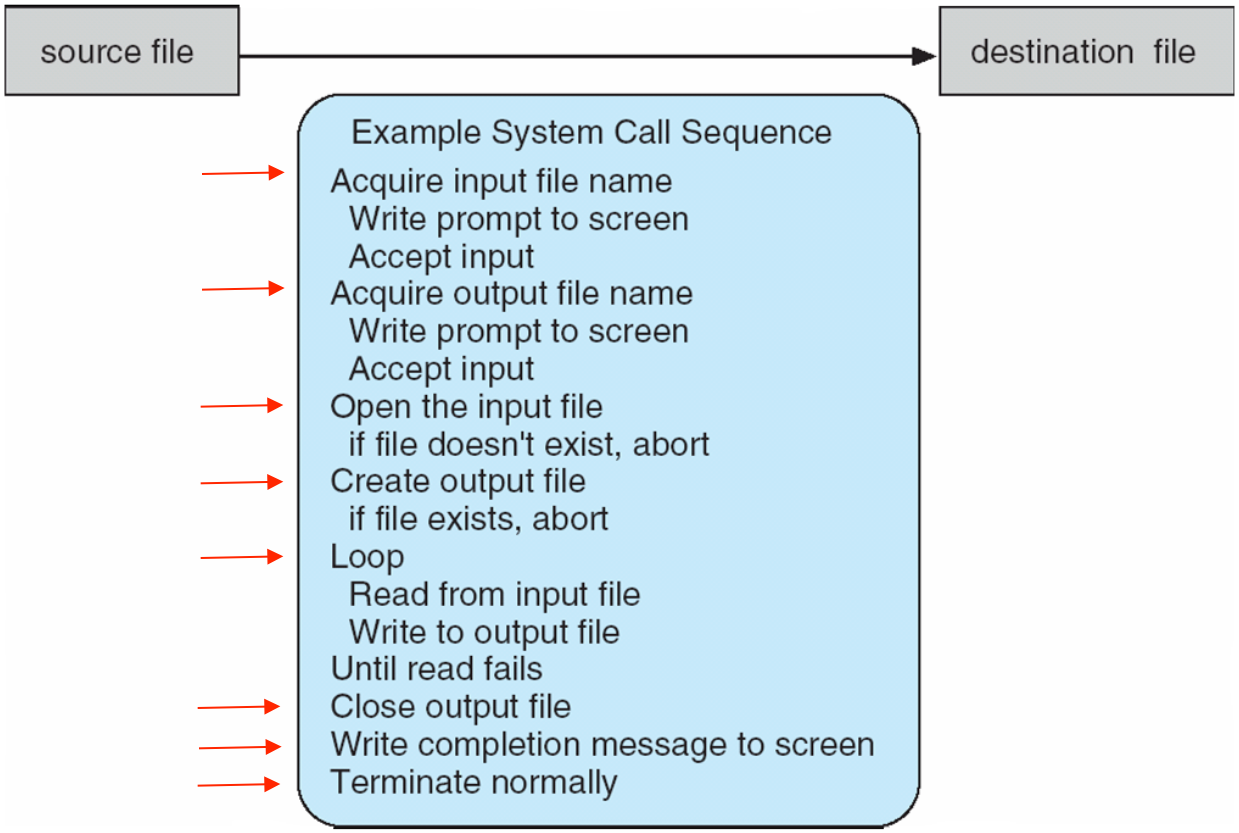
`read (fd, buffer, nbytes)`

Source: Tanenbaum

- Even a simple program may make heavy use of OS services.
  - e.g. see the example in the next slide

Example of System Calls

- The following image shows the sequence of system calls that are executed in order to copy the contents of one file to another file.



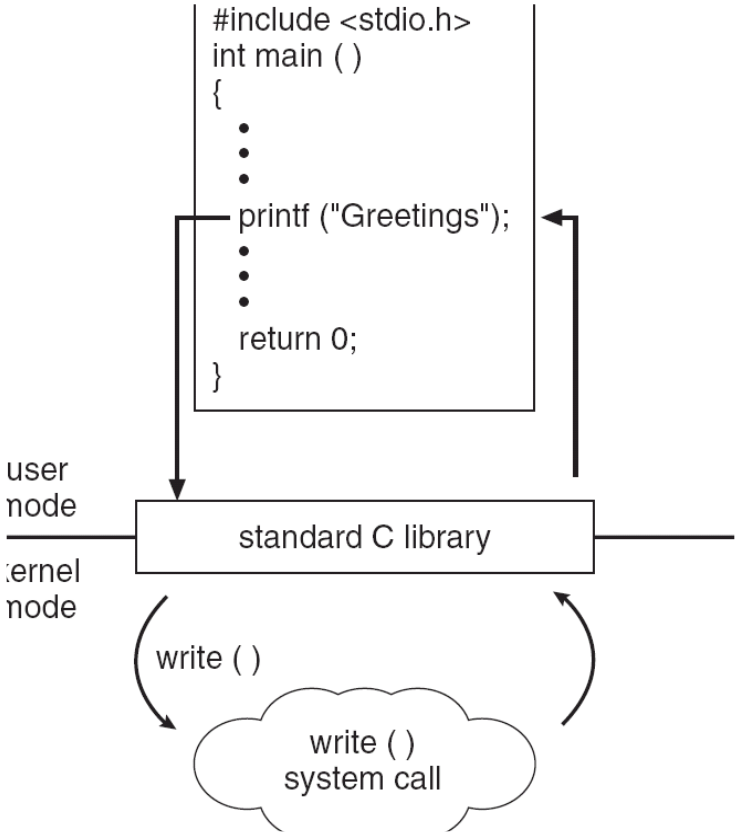
Note: the system-call names used throughout the textbook/lecture notes are generic, unless specified.

Application Program Interface (API)

- Most programmers never see the previously mentioned level of detail involved with the direct use of system calls.
- Typically, application developers design programs according to an Application Programming Interface (API) rather than a direct system call use
  - The API specifies a set of functions available
  - Why use APIs rather than system calls?
    - For many reasons, including 1) portability and 2) that system calls are usually more detailed and difficult to work with than the API.
- Three most common APIs are
  - **Win32 API** for Windows (Win64),
  - **POSIX API** for POSIX-based systems (UNIX, Linux, and Mac OS X)
  - **Java API** for the Java virtual machine (JVM)

Standard C Library Example

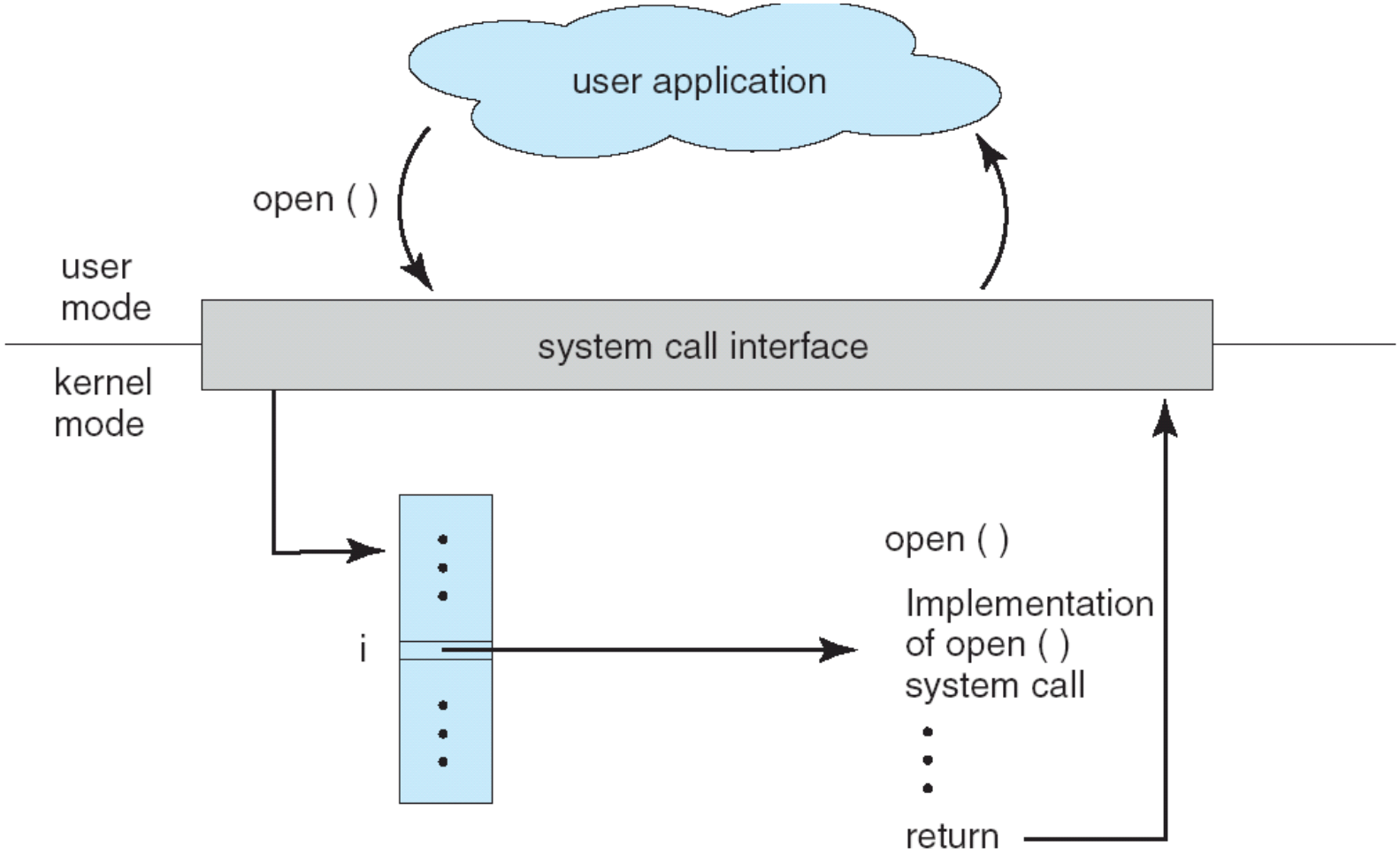
- The standard C library provides a portion of the system call interface for many versions of UNIX and Linux.
  - e.g. a C program invoking `printf()` library call, which calls `write()` system call



System Call Implementation

- The caller needs to know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
    - Managed by *run-time support library* (set of functions built into libraries included with compiler)
- The **system call interface** invokes intended system call in OS kernel and returns status of the system call and any return values
- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers

API – System Call – OS Relationship



Passing Parameters

- When a system call occurs, often more information is required than simply the identity of the desired system call
  - e.g. to get input, we may need to specify the file or device to use as the source
  - The exact type and amount of information vary according to the OS and the call

UNIX

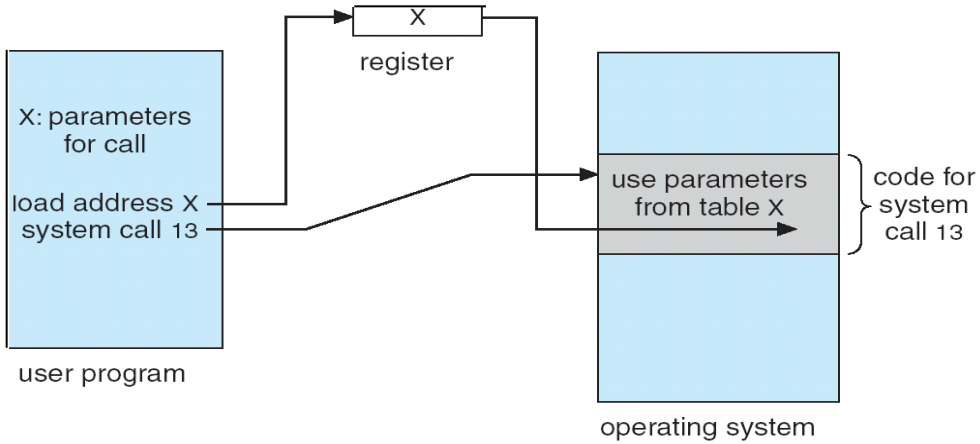
```
NAME

       fork - create a new process

SYNOPSIS

       #include <unistd.h>

       pid_t fork(void);
```

Windows

```
BOOL WINAPI CreateProcess(
  __in_opt      LPCTSTR lpApplicationName,
  __inout_opt   LPTSTR lpCommandLine,
  __in_opt      LPSECURITY_ATTRIBUTES lpProcessAttributes,
  __in_opt      LPSECURITY_ATTRIBUTES lpThreadAttributes,
  __in          BOOL bInheritHandles,
  __in          DWORD dwCreationFlags,
  __in_opt      LPVOID lpEnvironment,
  __in_opt      LPCTSTR lpCurrentDirectory,
  __in          LPSTARTUPINFO lpStartupInfo,
  __out         LPPROCESS_INFORMATION lpProcessInformation
);
```

Passing Parameters (cont)

- Three general methods are used to pass parameters to the OS
  1. pass the parameters in *registers* (simplest)
  2. Parameters stored in a *block,* or table, in memory, and address of block passed as a parameter in a register (used in Linux and Solaris)



  3. Parameters placed, or *pushed,* onto the *stack* by the program and *popped* off the stack by the operating system

- In the first method, in some cases there may be more parameters than registers. However, the last two methods (block or stack) do not limit the number or length of parameters being passed.

Types of System Calls

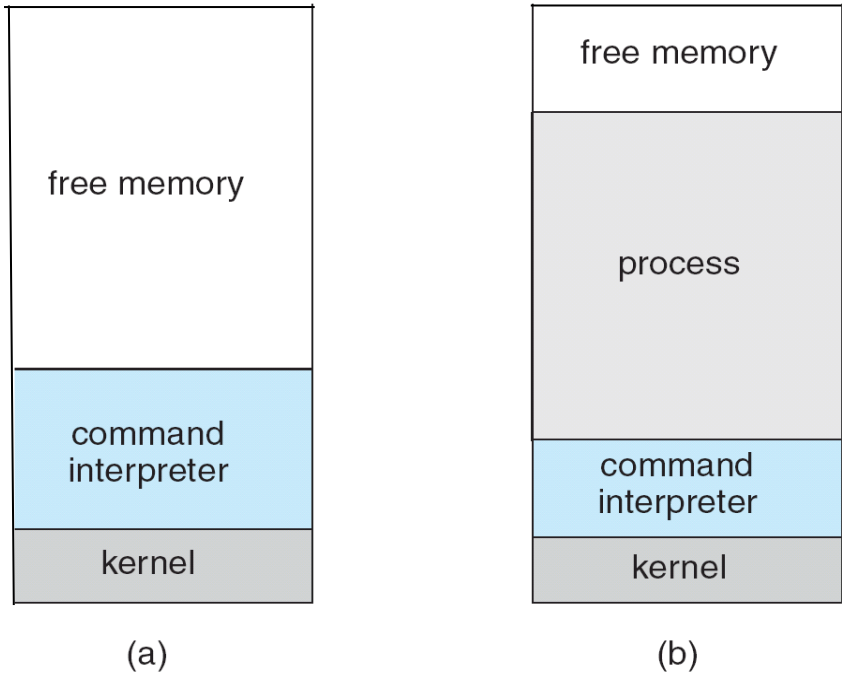
- System calls can be grouped roughly into six major categories:

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

MS-DOS Execution

- Let's take a look at two variations of process and job control: MS-DOS (single-tasking) and Free BSD (multitasking)

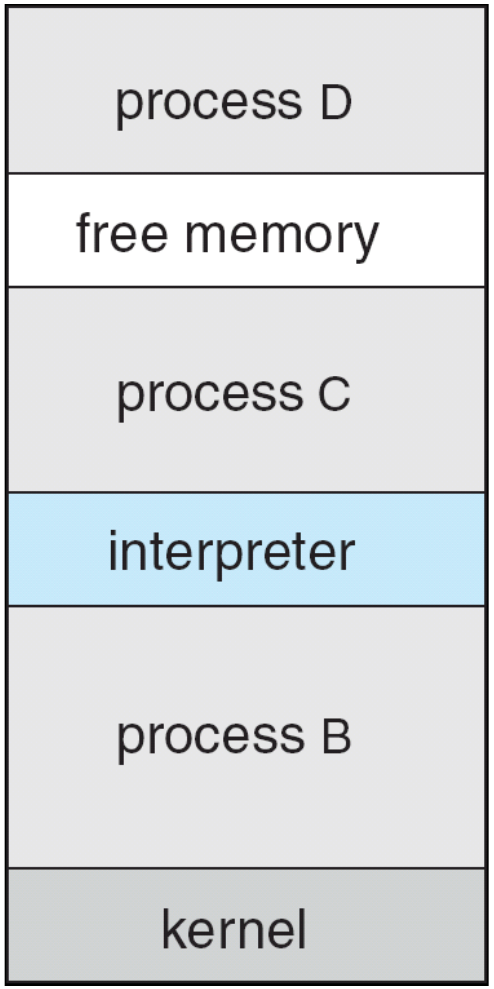
- MS-DOS command interpreter is invoked at the startup
- To run a program,
  - loads it into memory, writing over most of itself to give the program as much memory as possible.
  - sets the instruction pointer to the first instruction of the program
- After the program terminates, the command interpreter resumes.



(a) At system startup (b) running a program

FreeBSD Running Multiple Programs

- The shell is run at the login
- Since it is multitasking, the command interpreter may continue running while another program is executed
- To execute a program, it creates a new process (*fork()*) and then the selected program is loaded into memory
  - waits for the program or runs it "in the background"
- In the meanwhile, the user is free to ask the shell to run other programs
- When the process is done, it executes an *exit()* system call to terminate



Outline

- What Operating Systems (OS) Do
- Computer System
  - Architecture and Organization
- Operating-System Structure and Operations
  - Process Management
  - Memory Management
  - Storage Management
- OS Services and System Calls