Operating System
BCS-303

Module-1
Chapter-1 Introduction to operating systems, System structures

Prepared By,
Darshini Y
Assistant Professor
Dept of CS&D
ATMECE,Mysuru

Syllabus

Module-1

Chapter-1

Introduction to operating systems, System structures: What operating systems do; Computer System organization; Computer System architecture; Operating System structure; Operating System operations; Process management; Memory management; Storage management; Protection and Security; Distributed system; Special-purpose systems; Computing environments.

Chapter-2

Operating System Services: User - Operating System interface; System calls; Types of system calls; System programs; Operating system design and implementation; Operating System structure; Virtual machines; Operating System debugging, Operating System generation; System boot.

Textbook 1: Chapter – 1 (1.1-1.12), 2 (2.2-2.11)

Module-2

Chapter-1

Process Management: Process concept; Process scheduling; Operations on processes; Inter process communication.

Chapter-2

Multi-threaded Programming: Overview; Multithreading models; Thread Libraries; Threading issues.

Chapter-3

Process Scheduling: Basic concepts; Scheduling Criteria; Scheduling Algorithms; Thread scheduling; Multiple-processor scheduling.

Textbook 1: Chapter – 3 (3.1-3.4),

4 (4.1-4.4) and 5 (5.1 -5.5)

Module-3

Chapter-1

Process Synchronization: Synchronization: The critical section problem; Peterson's solution; Synchronization hardware; Semaphores; Classical problems of synchronization;

Chapter-2

Deadlocks: System model; Deadlock characterization; Methods for handling deadlocks; Deadlock prevention; Deadlock avoidance; Deadlock detection and recovery from deadlock.

Textbook 1: Chapter – 6 (6.1-6.6), 7 (7.1 -7.7)

Module-4

Chapter-1

Memory Management: Memory management strategies: Background; Swapping; Contiguous memory allocation; Paging; Structure of page table; Segmentation.

Chapter-2

Virtual Memory Management: Background; Demand paging; Copy-on-write; Page replacement; Allocation of frames; Thrashing.

Textbook 1: Chapter -8 (8.1-8.6), 9 (9.1-9.6)

Module-5

Chapter-1

File System, Implementation of File System: File system: File concept; Access methods; Directory and Disk structure; File system mounting; File sharing; Implementing File system: File system structure; File system implementation; Directory implementation; Allocation methods; Free space management.

Chapter-2

Secondary Storage Structure, Protection: Mass storage structures; Disk structure; Disk attachment; Disk scheduling; Disk management; Protection: Goals of protection, Principles of protection, Domain of protection, Access matrix.

Textbook 1: Chapter – 10 (10.1-10.5) ,11 (11.1-11.5),12 (12.1-12.5), 14 (14.1-14.4)

Course Outcome

The students should be able to:

- To Demonstrate the need for OS and different types of OS.
- To discuss suitable techniques for management of different resources
- To demonstrate different APIs/Commands related to processor, memory, storage and file system management.

Text Books:

1.Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System Principles 8th edition, Wiley-India, 2015

Reference Books :

1. Ann McHoes Ida M Fylnn, Understanding Operating System, Cengage Learning, 6th Edition.
2. 2. D.M Dhamdhere, Operating Systems: A Concept Based Approach 3rd Ed, McGraw-Hill, 2013.
3. 3. P.C.P. Bhatt, An Introduction to Operating Systems: Concepts and Practice 4th Edition, PHI(EEE), 2014.
4. 4. William Stallings Operating Systems: Internals and Design Principles, 6th Edition, Pearson.

Module 1: Introduction

- What operating systems do.
- Computer System organization.
- Computer System architecture.
- Operating System structure.
- Operating System operations.
- Process management;.
- Memory management.
- Storage management.
- Protection and Security.
- Distributed system.
- Special-purpose systems;
- Computing environments.

What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware.
- 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.

Computer System Structure

- Computer system can be divided into four components

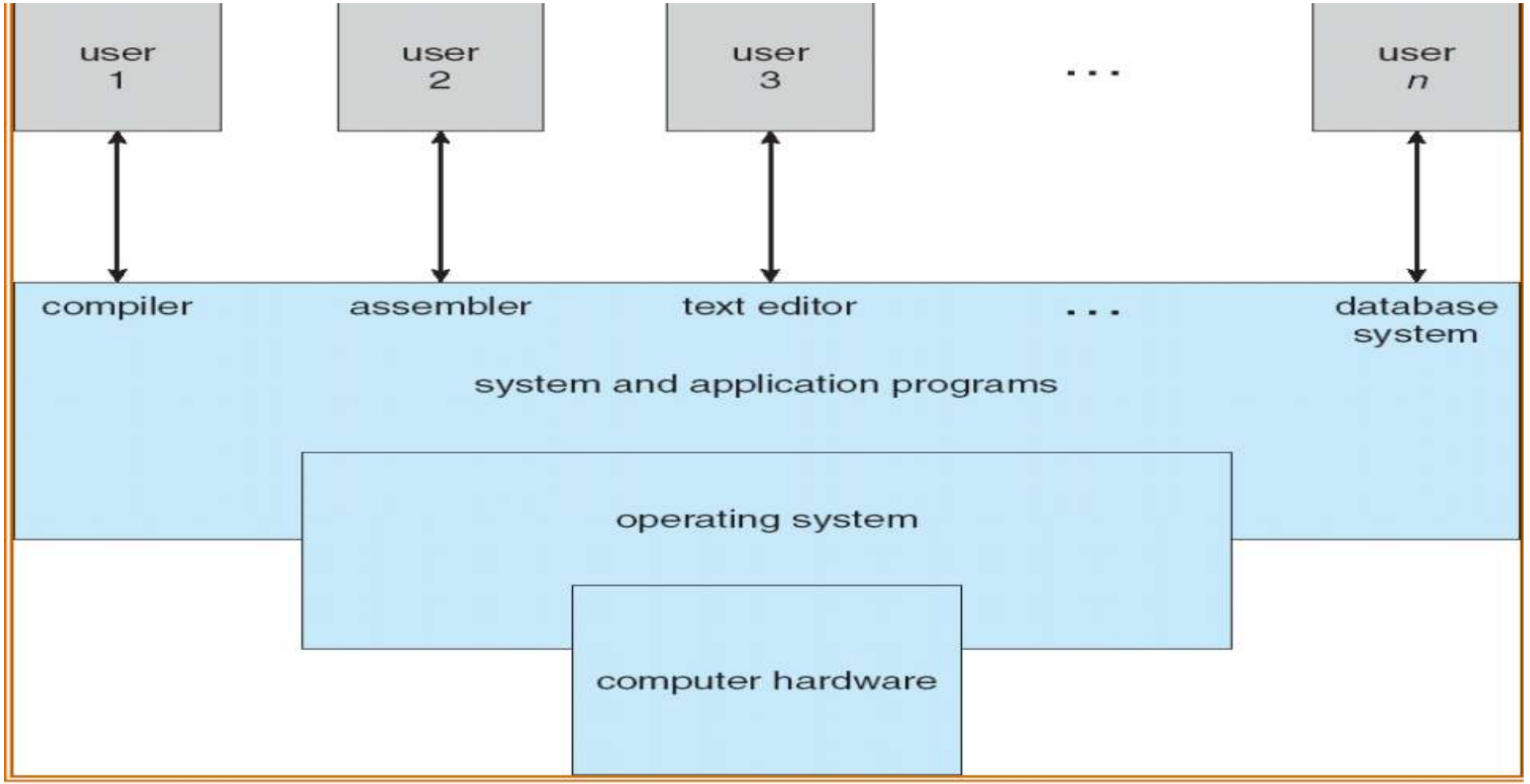
1. Hardware – provides basic computing resources
    - CPU, memory, I/O devices
2. Operating system
    - Controls and coordinates use of hardware among various applications and users
3. Application programs

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

4. Users
    - People, machines, other computers

Four Components of a Computer System



Operating System Definition

- 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 (Cont.)

- No universally accepted definition
- "Everything a vendor ships when you order an operating system" is good approximation – But varies wildly
- "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

Computer Startup

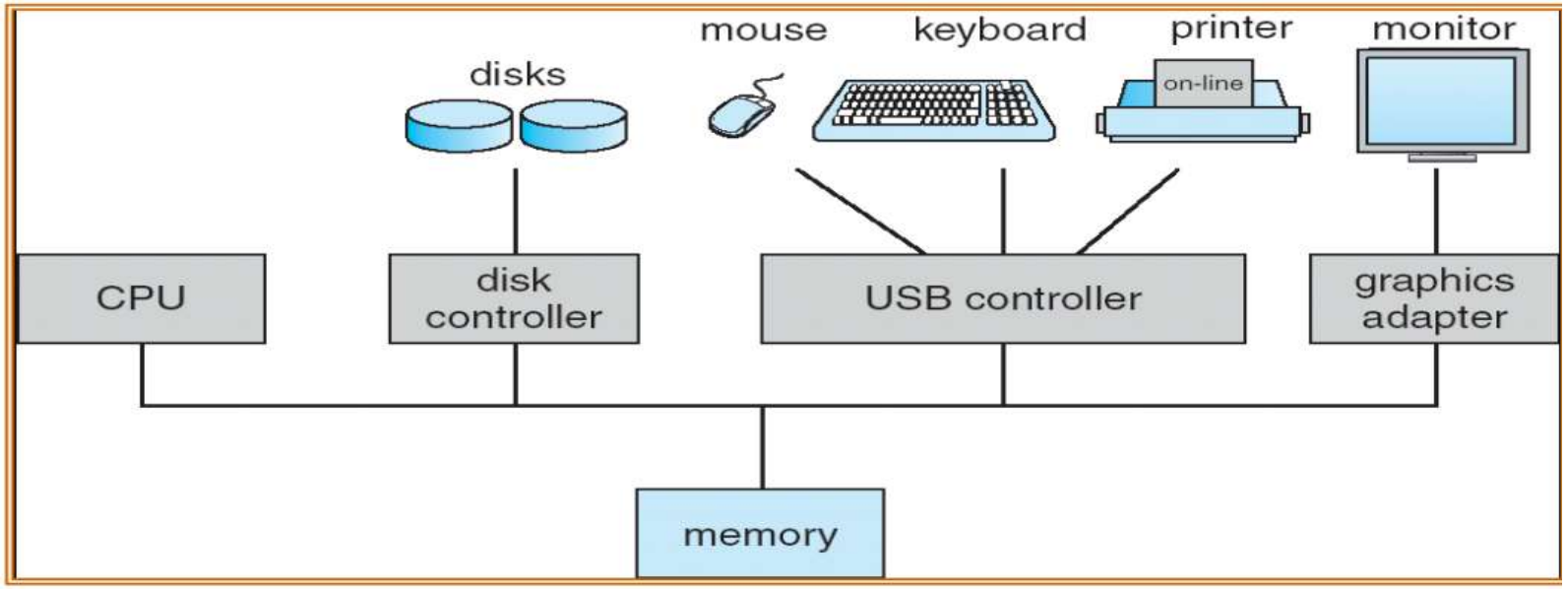
- **bootstrap program** is loaded at power-up or reboot
  - Typically stored in ROM or EEPROM, generally known as firmware
  - Initializates all aspects of system

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.

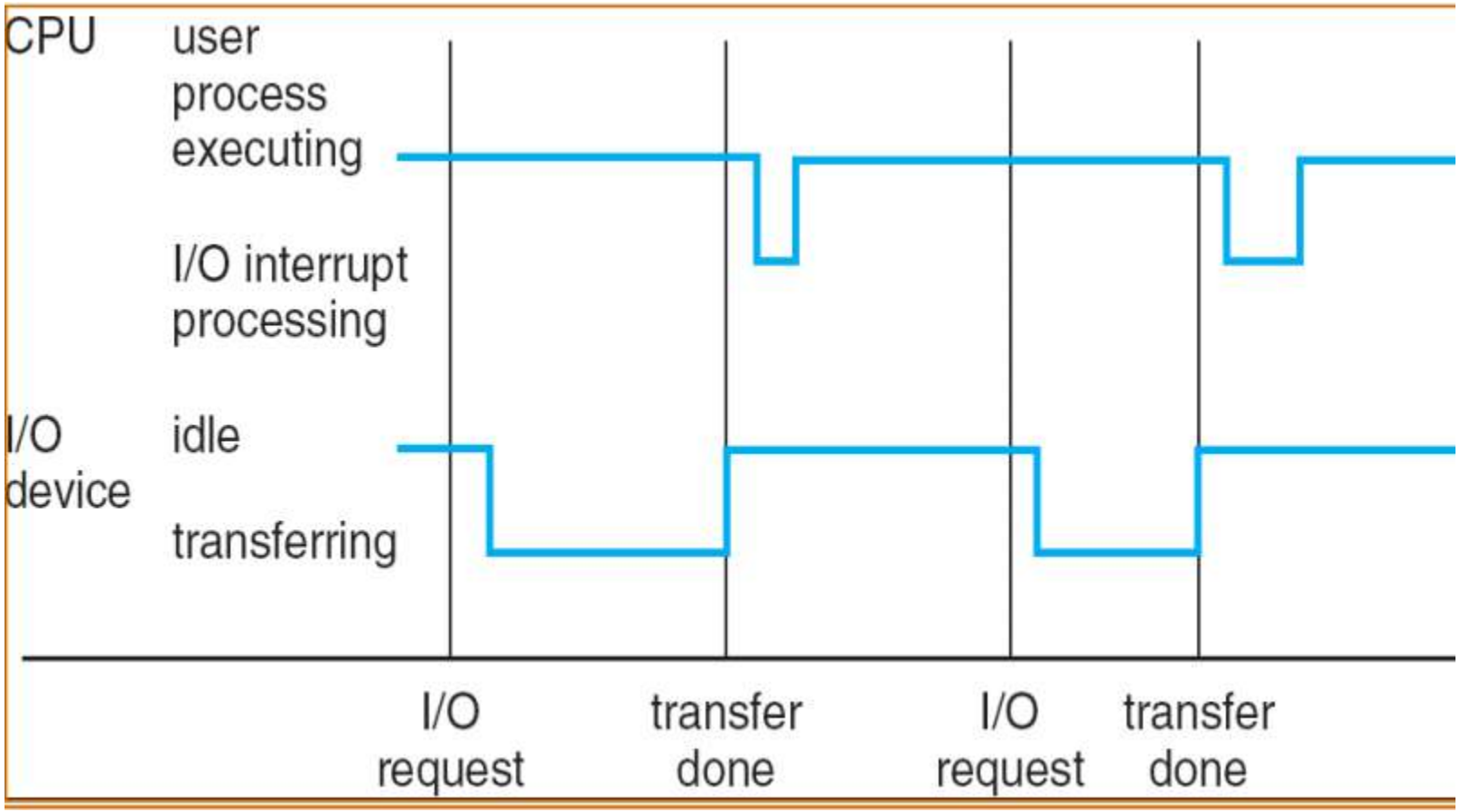
Common Functions of Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the interrupt vector, which contains the addresses of all the service routines.
- Interrupt architecture must save the address of the interrupted instruction.
- 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.
- An operating system is interrupt driven.

Interrupt Handling

- The operating system preserves the state of the CPU by storing registers and the program counter.
- Determines which type of interrupt has occurred:
  - polling
  - vectored interrupt system
- Separate segments of code determine what action should be taken for each type of interrupt

Interrupt Timeline



I/O Structure

- After I/O starts, control returns to user program only upon I/O completion.
  - Wait instruction idles the CPU until the next interrupt
  - Wait loop (contention for memory access).
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing.
- After I/O starts, control returns to user program without waiting for I/O completion.
  - System call.
  - request to the operating system to allow user to wait for I/O completion.
  - Device-status table contains entry for each I/O device indicating its type, address, and state.
  - Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt.

Direct Memory Access Structure

- 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 on interrupt is generated per block, rather than the one interrupt per byte

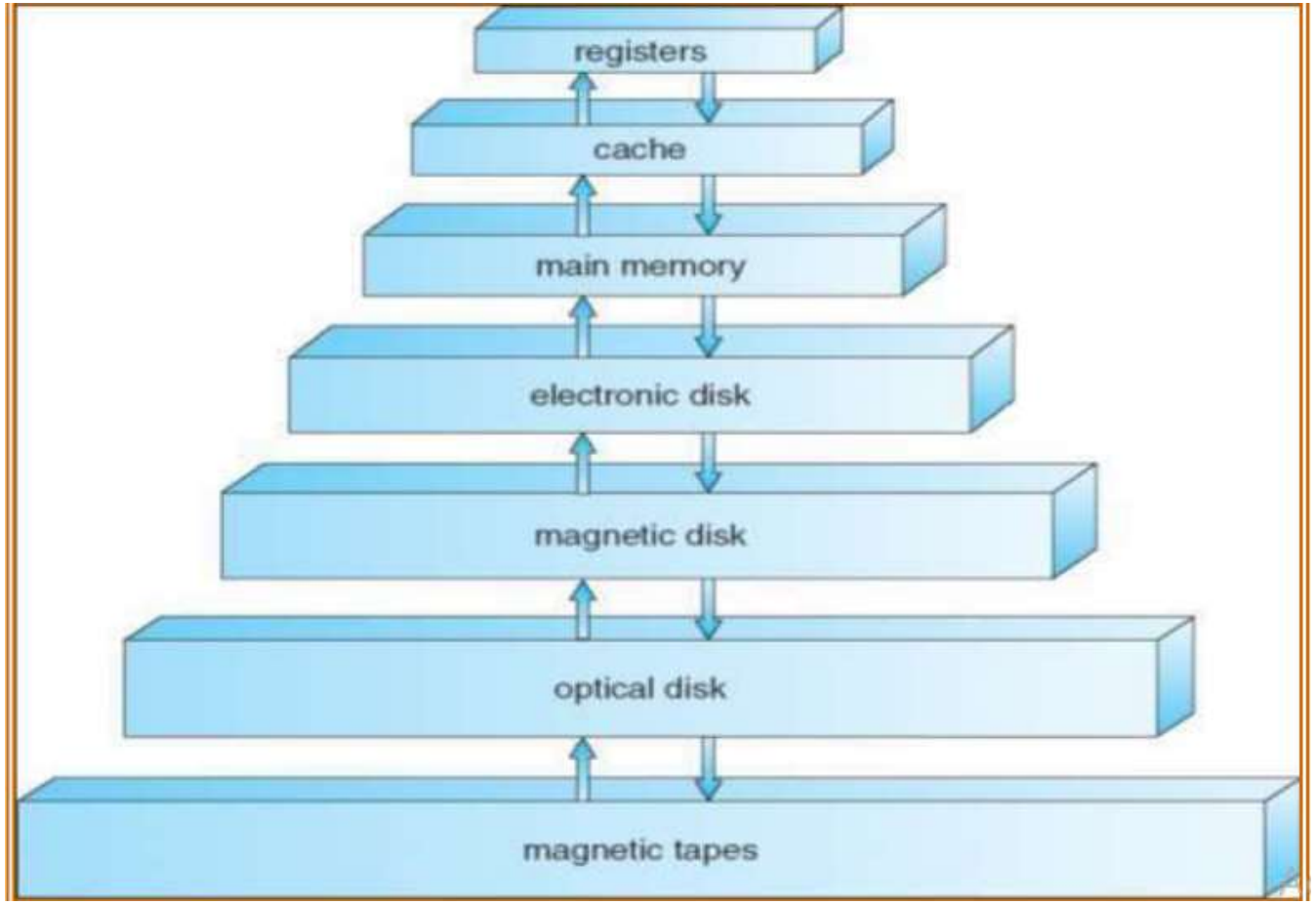
Storage Structure

- Main memory – only large storage media that the CPU can access directly.
- Secondary storage – 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.

Storage Hierarchy

- Storage systems organized in hierarchy.
  - Speed
    - Cost
  - Volatility
- Caching – copying information into faster storage system; main memory can be viewed as a last cache for secondary storage.

Storage-Device Hierarchy



Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- 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 smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy

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

Operating System Structure

- **Multiprogramming** 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
– A subset of total jobs in system 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

(cont)....

- **Timesharing (multitasking)** is logical extension in which 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

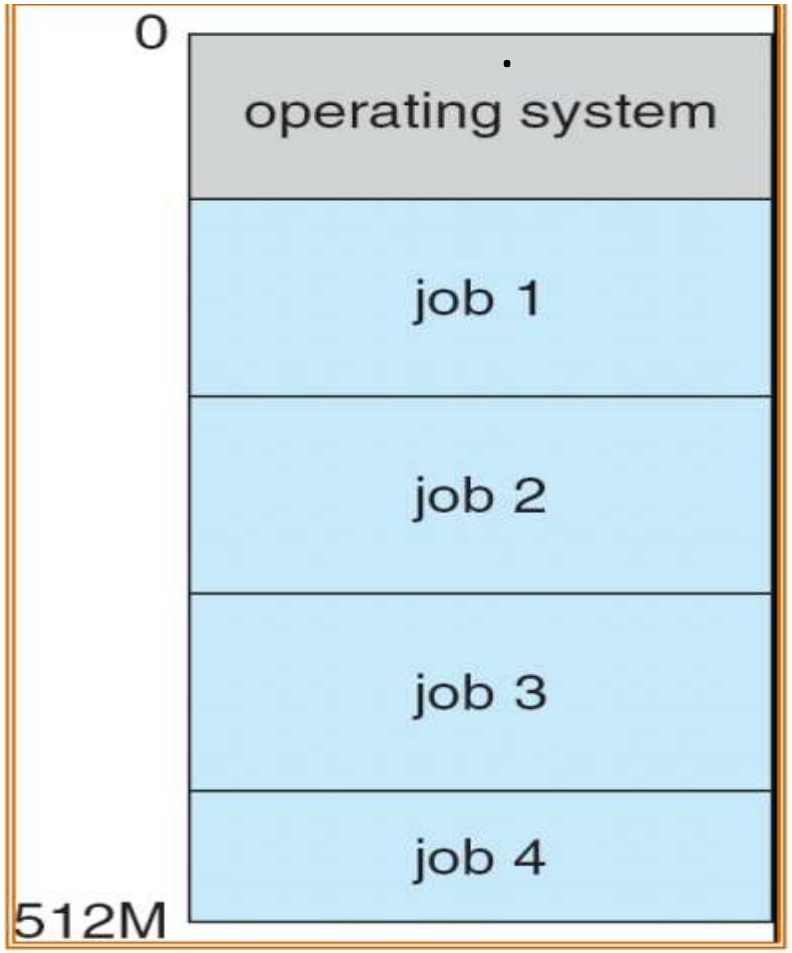
– Each user has at least one program executing in memory process

– 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

Memory Layout for Multiprogrammed System



Operating-System Operations

- Interrupt driven by hardware
- Software error or request creates exception or trap
  - Division by zero, request for operating system service
- Other process problems include infinite loop, processes modifying each other or the operating system
- Dual-mode operation allows OS to protect itself and other system components
  - User mode and kernel mode
  - Mode bit provided by hardware

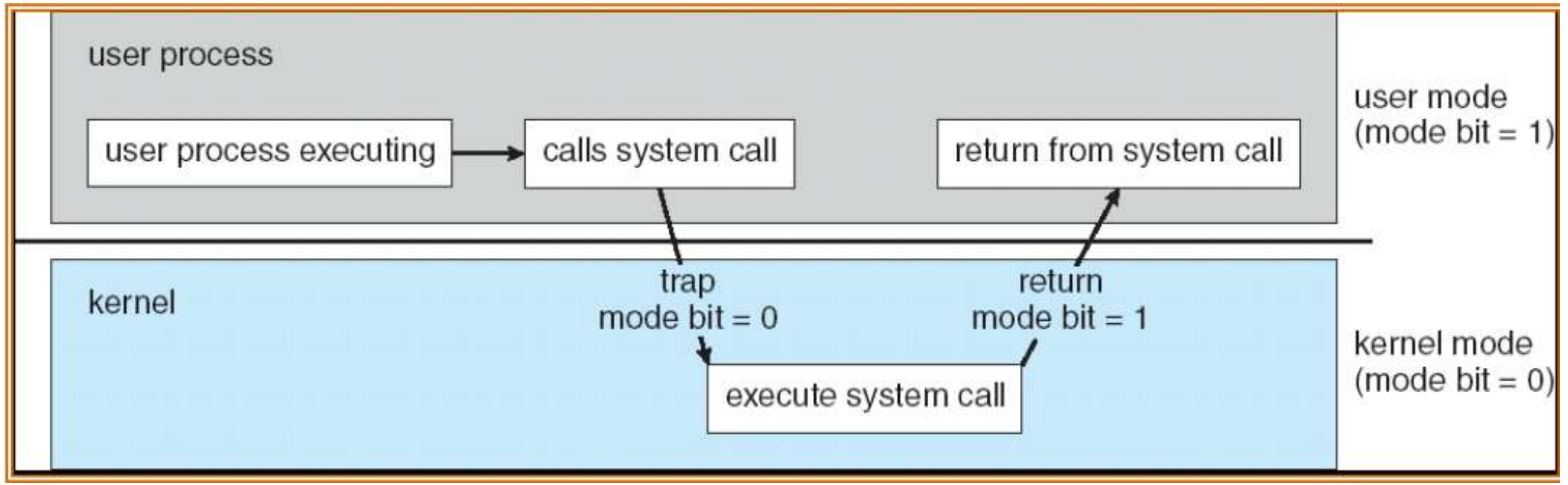
(Cont)…

- Provides ability to distinguish when system is running user code or kernel code
- Some instructions designated as privileged, only executable in kernel mode
- System call changes mode to kernel, return from call resets it to user.

Transition from User to Kernel Mode

- Timer 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

(cont)...


user process
user process executing
calls system call
return from system call
user mode
(mode bit = 1)
kernel
trap
mode bit = 0
return
mode bit = 1
execute system call
kernel mode
(mode bit = 0)

Process Management

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

(cont)…

- Multi-threaded process has one program counter per thread
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes / threads

Process Management Activities

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

- All data in memory before and after processing
- All instructions in memory in order to execute
- Memory management determines what is in memory 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 de allocating memory space as needed

Storage Management

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit – 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 dirs • Mapping files onto secondary storage • Backup files onto stable (non-volatile) storage media

Storage Management (cont)...

– OS activities include

- Creating and deleting files and directories
- Primitives to manipulate files and dirs.
- Mapping files onto secondary storage
- Backup files onto stable (non-volatile) storage media

Mass-Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time.
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities

– 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)

Mass-Storage Management (cont)…

- OS activities
  - 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)

I/O Subsystem

- One purpose of OS is to hide peculiarities of hardware devices from the user
- I/O subsystem responsible for
  - Memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance), spooling (the overlapping of output of one job with input of other jobs)
  - General device-driver interface
  - Drivers for specific hardware devices

Protection and Security

- **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

Protection and Security

– 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

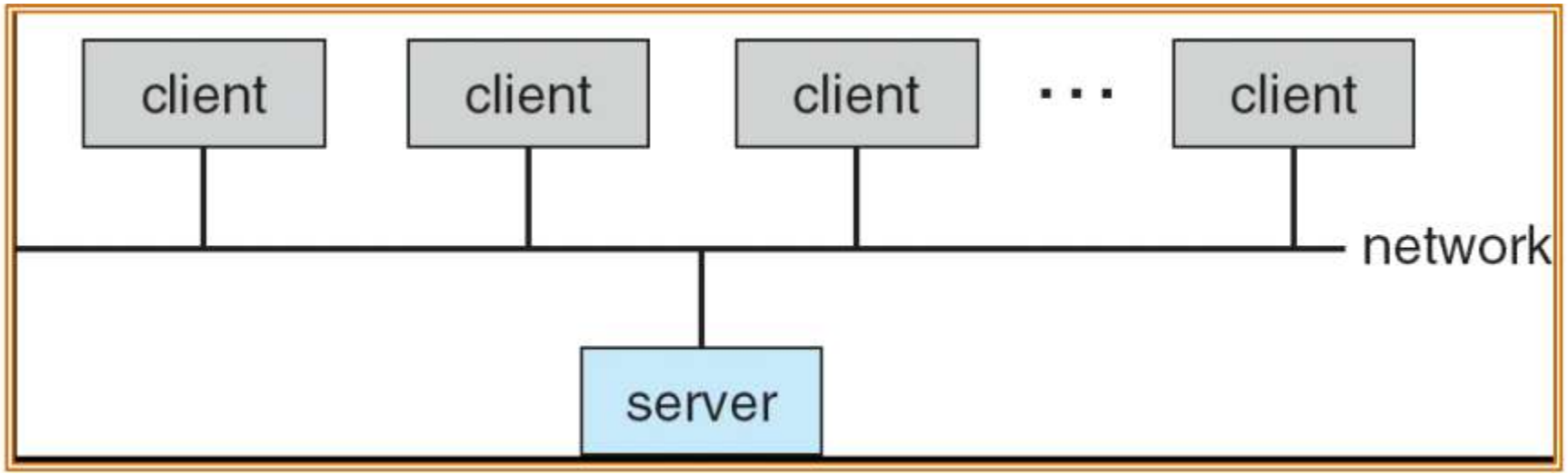
Computing Environments

- Traditional computer
  - Blurring over time
  - Office environment
- PCs connected to a network, terminals attached to mainframe or minicomputers providing batch and timesharing
- Now portals allowing networked and remote systems access to same resources
  - Home networks
- Used to be single system, then modems
- Now firewalled, networked Computing Environments

Computing Environments(cont)…

- Client-server computing
- Dumb terminals supplanted by smart PCs
- ➢ Many systems now servers, responding to requests generated by clients Compute-server provides an interface to client to request services (i.e. database)
- ➢ File-server provides interface for clients to store and retrieve files

Computing Environments(cont)...



Peer-to-Peer Computing

- Another model of distributed system
- P2P does not distinguish clients and servers
  - Instead all nodes are considered peers
  - May each act as client, server or both
  - Node must join P2P network
- Registers its service with central lookup service on network, or
- Broadcast request for service and respond to requests for service via discovery protocol
  - Examples include Napster and Gnutella

Web-Based Computing

- Web has become ubiquitous
- PCs most prevalent devices
- More devices becoming networked to allow web access
- New category of devices to manage web traffic among similar servers: load balancers
- Use of operating systems like Windows 95, client-side, have evolved into Linux and Windows XP, which can be clients and servers

Chapter-2
Operating System Services

- User-Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines
- Operating System Generation
- Operating System Debugging
- System Boot

Operating System Services

One set of operating-system services provides functions that are helpful to the user:

-User interface - Almost all operating systems have a user interface (UI)

• Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch Interface

– Program execution - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)

(cont)…

– **I/O operations** -A running program may require I/O, which may involve a file or an I/O device.

– **File-system manipulation -** The file system is of particular interest. Obviously, programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

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

(cont)…

– **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

(cont)…

OS functions for ensuring the efficient operation of the system itself via resource sharing

- **– Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
- **– Accounting -** To keep track of which users use how much and what kinds of computer resources
- **– Protection and security** - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other

(cont)…

- **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
- 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.

User Operating System Interface -CLI

CLI (Command Line Interpreter) allows direct command entry

- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented –shells
- Primarily fetches a command from user and executes it – Sometimes commands built-in, sometimes just names of programs
- Main function of the command interpreter is to get and execute the next user-defined command.

User Operating System Interface - GUI

- User-friendly desktop metaphor interface
  - Usually mouse, keyboard, and monitor
  - Icons represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a folder)
  - Invented at Xerox PARC

(cont)…

- Many systems now include both CLI and GUI interfaces

– Microsoft Windows is GUI with CLI “command” shell

– Apple Mac OS X as “Aqua” GUI interface with UNIX kernel underneath and shells available

– Solaris is CLI with optional GUI interfaces (Java Desktop, KDE)

System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)
- Why use APIs rather than system calls?

(Note that the system-call names used throughout this text are generic)

(Cont)…

- In computing, a system call is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on.
- A system call is a way for programs to interact with the operating system.
- A computer program makes a system call when it makes a request to the operating system's kernel. • System call provides the services of the operating system to the user programs via Application Program Interface(API).

(Cont)…

- It provides an interface between a process and operating system to allow user-level processes to request services of the operating system.
- System calls are the only entry points into the kernel system.
- All programs needing resources must use system calls.

Services Provided by System Calls :

- Process creation and management
- Main memory management
- File Access, Directory and File system management
- Device handling(I/O)
- Protection
- Networking, etc.

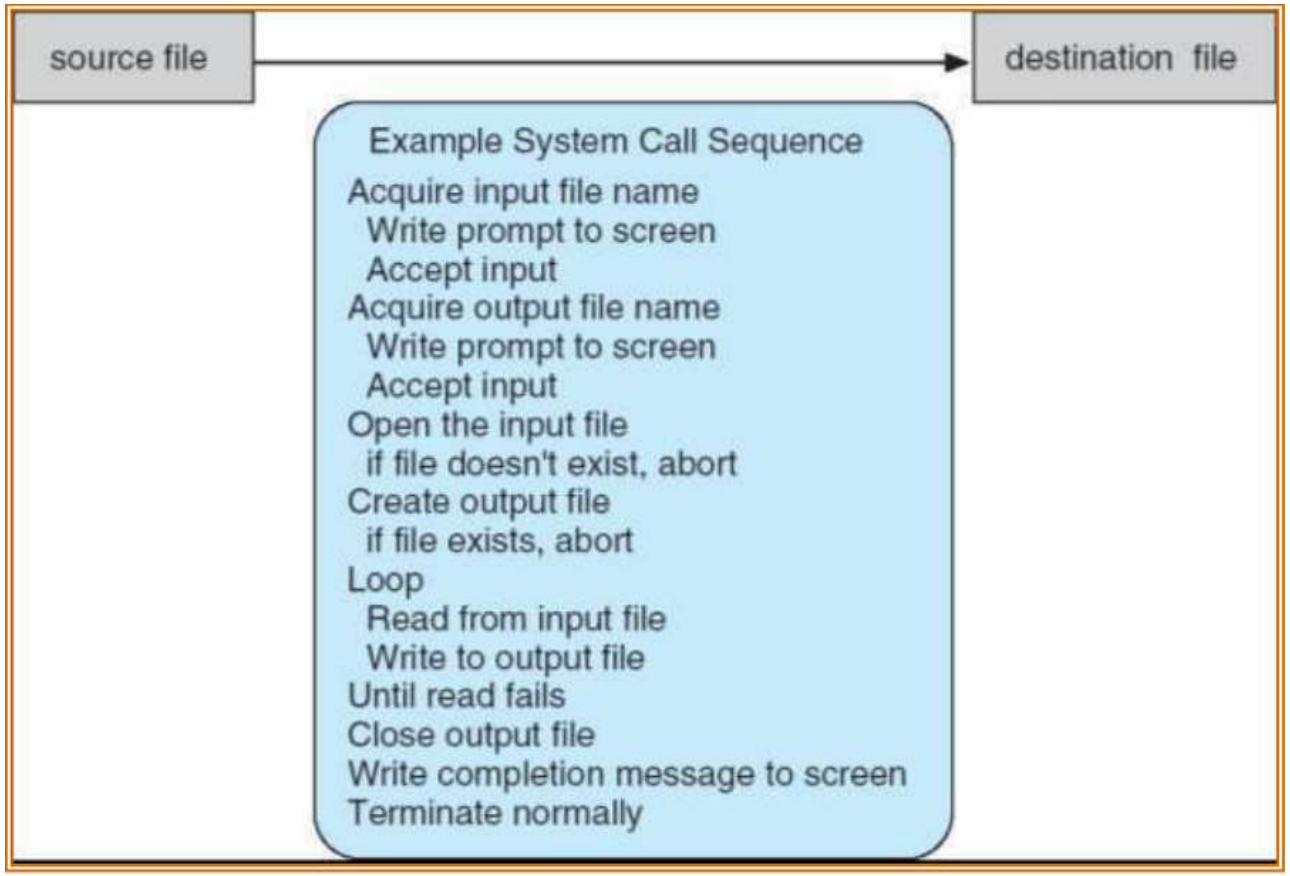
Why do you need System Calls in OS?

Following are situations which need system calls in OS:

- Reading and writing from files demand system calls.
- If a file system wants to create or delete files, system calls are required.
- System calls are used for the creation and management of new processes.
- Network connections need system calls for sending and receiving packets.
- Access to hardware devices like scanner, printer, need a system call.

Example of System Calls

System call sequence to copy the contents of one file to another file.



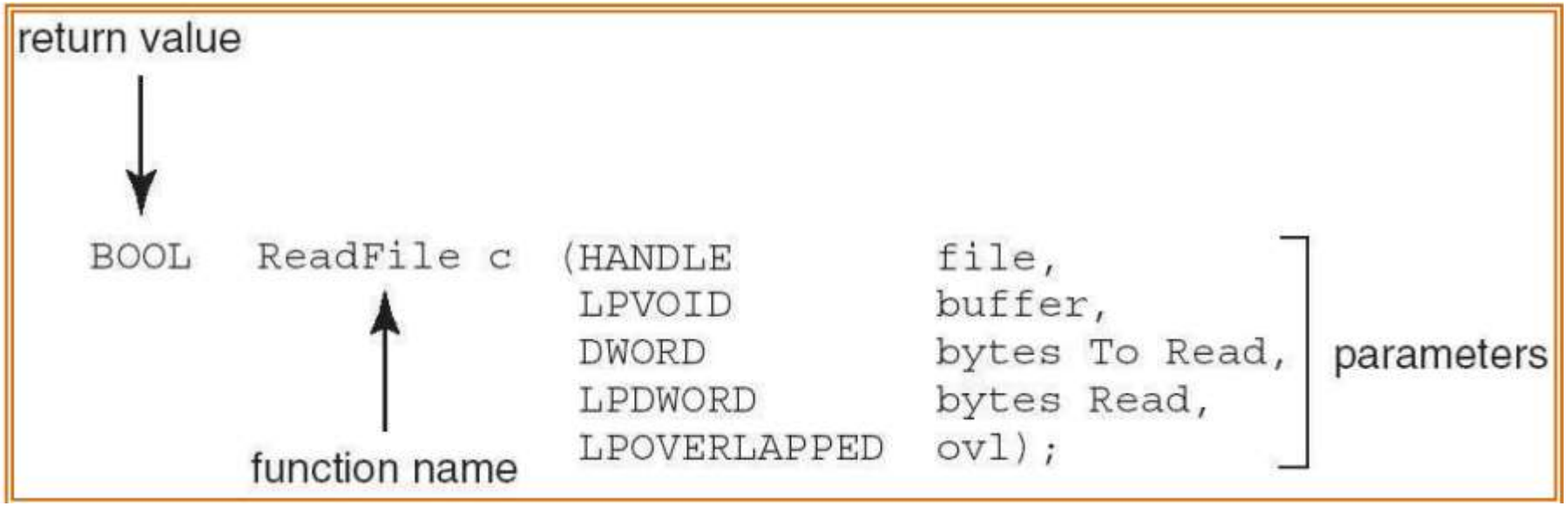
Example of System Calls

- Example: System call sequence to copy contents of one file to another

1. Display dialog to choose source file
2. Display dialog to choose destination folder
3. Display progress dialog
4. Open source file for reading
5. Open destination file for creating and writing
6. Loop while more source bytes remaining:
    1. Read *n* bytes from source file
    2. Write *n* bytes to destination file
    3. Update progress dialog
7. Close source file
8. Close destination file
9. Close progress dialog
10. Terminate normally

Example of Standard API

- Consider the ReadFile() function in the Win32 API—a function for reading from a file



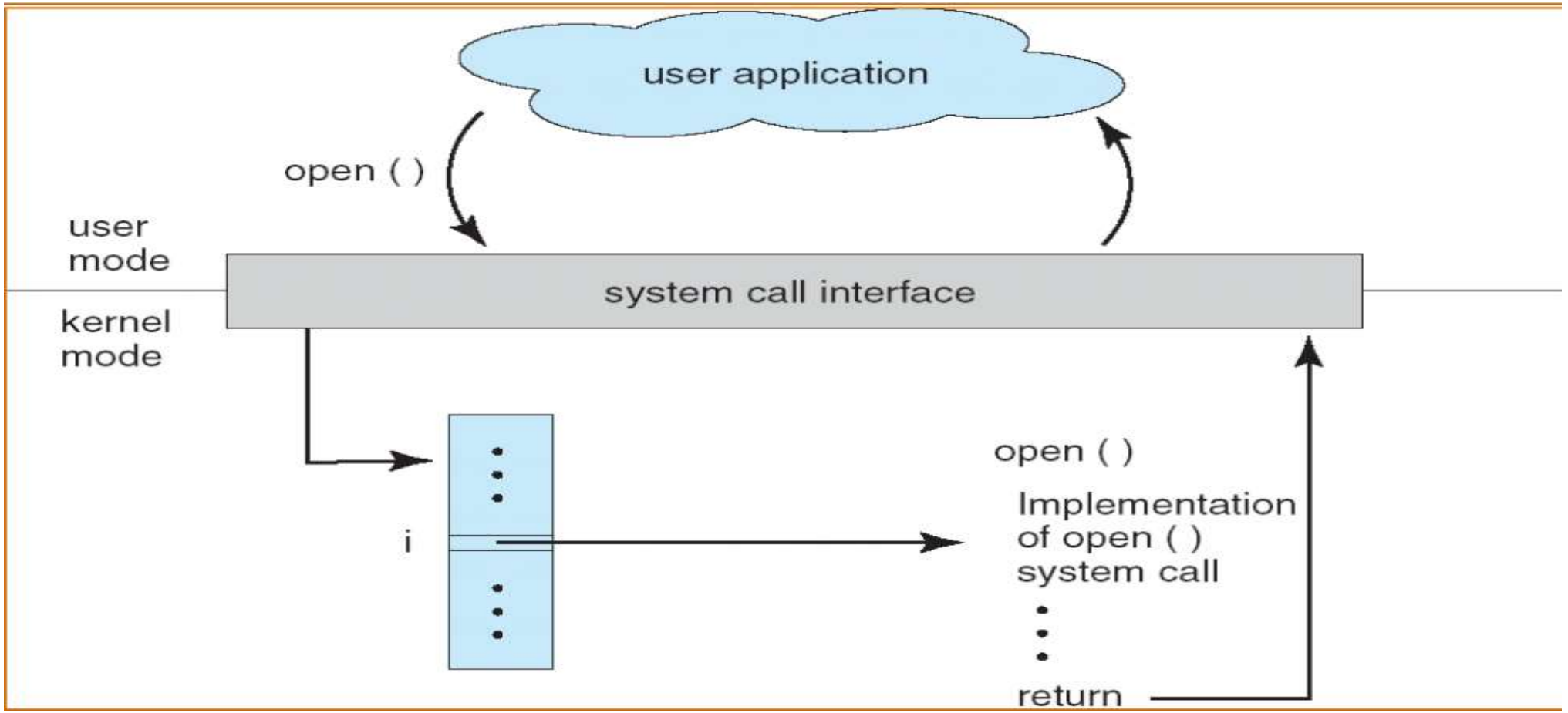
(Cont)…

- A description of the parameters passed to ReadFile()
  – HANDLE file
  —the file to be read
  – LPVOID buffer
  —a buffer where the data will be read into and written from
  – DWORD bytesToRead—the number of bytes to be read into the buffer
  – LPDWORD bytesRead—the number of bytes read during the last read
  – LPOVERLAPPED ovl—indicates if overlapped I/O is being used

System Call Implementation

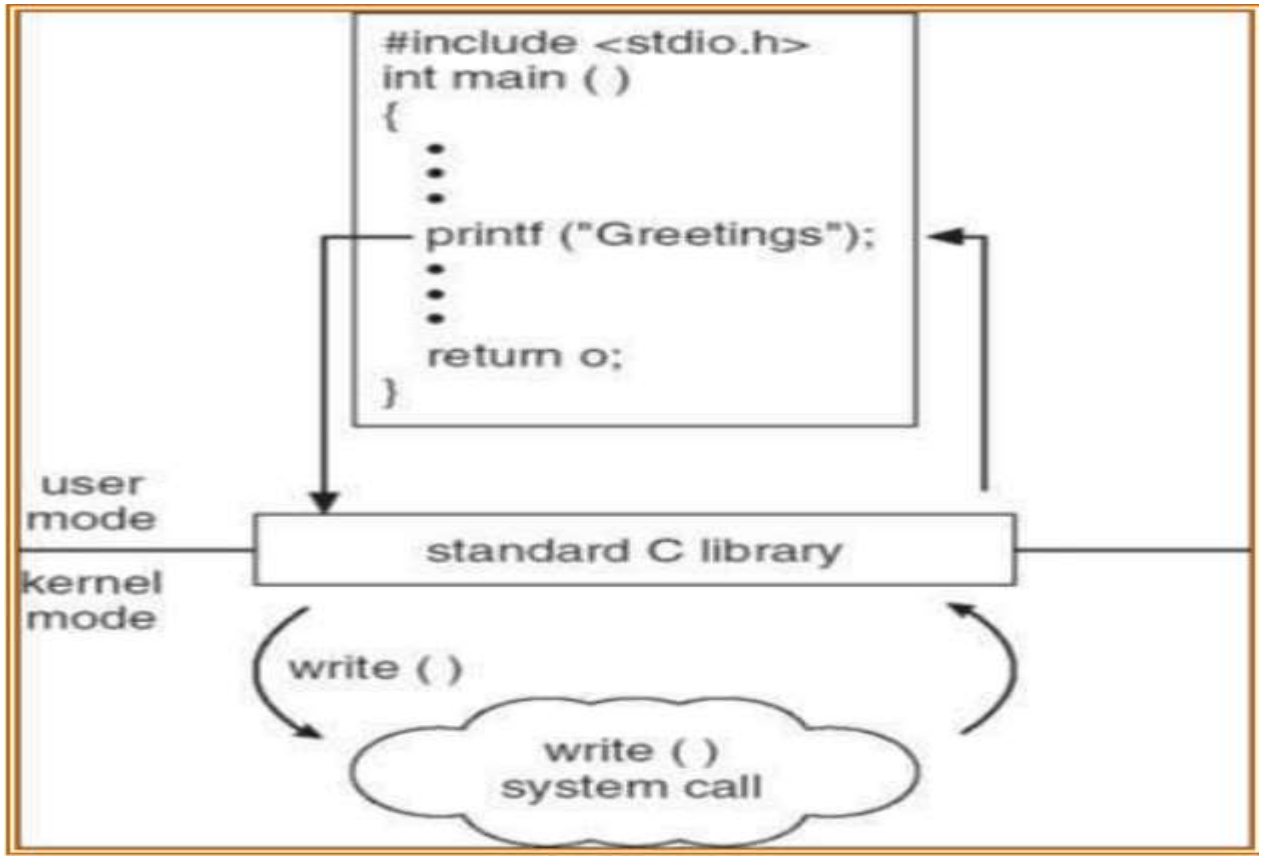
- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about 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)

API –System Call –OS Relationship



Standard C Library Example

- C program invoking printf() library call, which calls write() system call



System Call Parameter Passing

Often, more information is required than simply identity of desired system call

– Exact type and amount of information vary according to OS and call

- Three general methods used to pass parameters to the OS

– Simplest: pass the parameters in registers

    - In some cases, may be more parameters than registers

– Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register

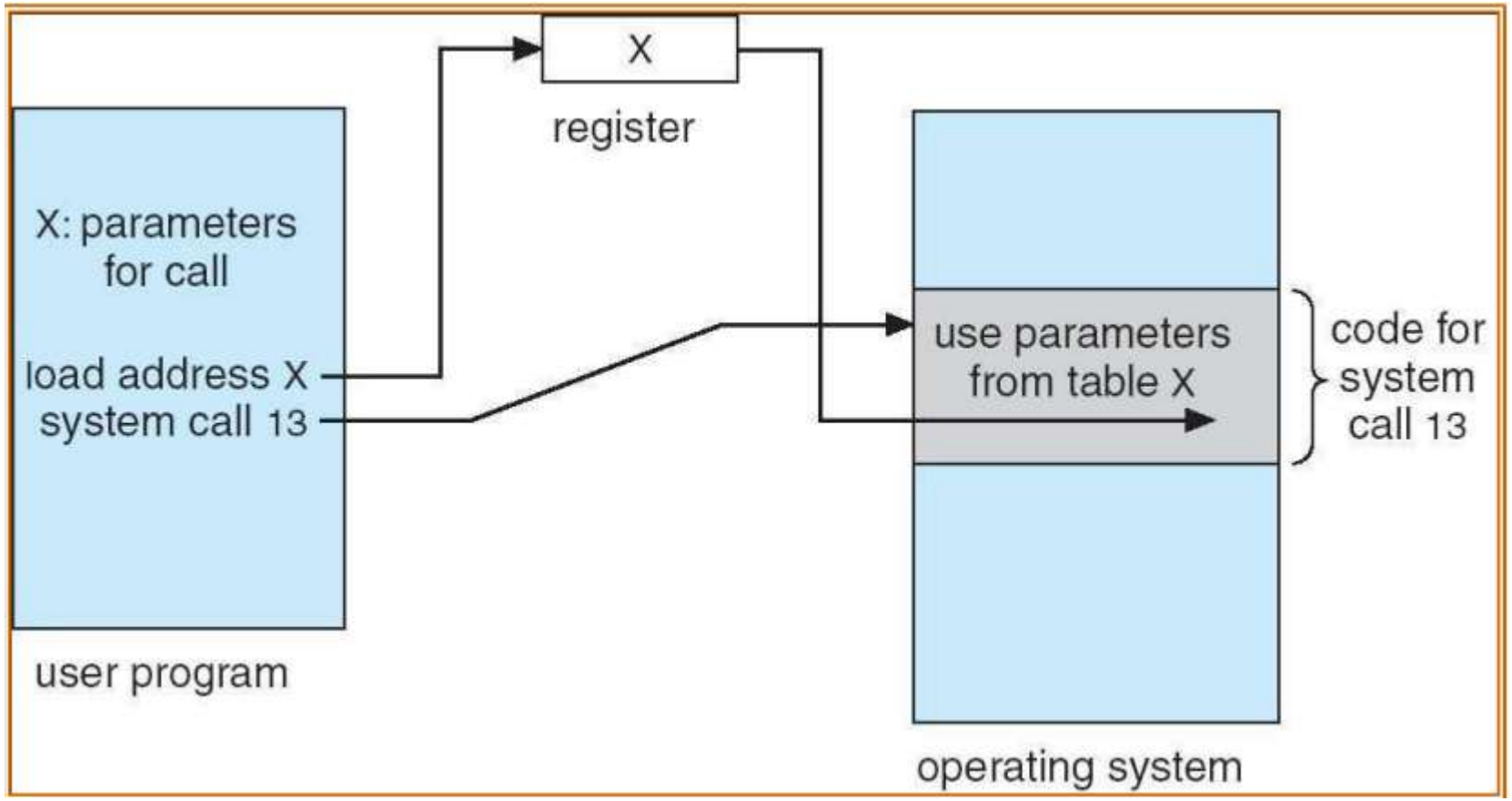
(cont)…

- This approach taken by Linux and Solaris

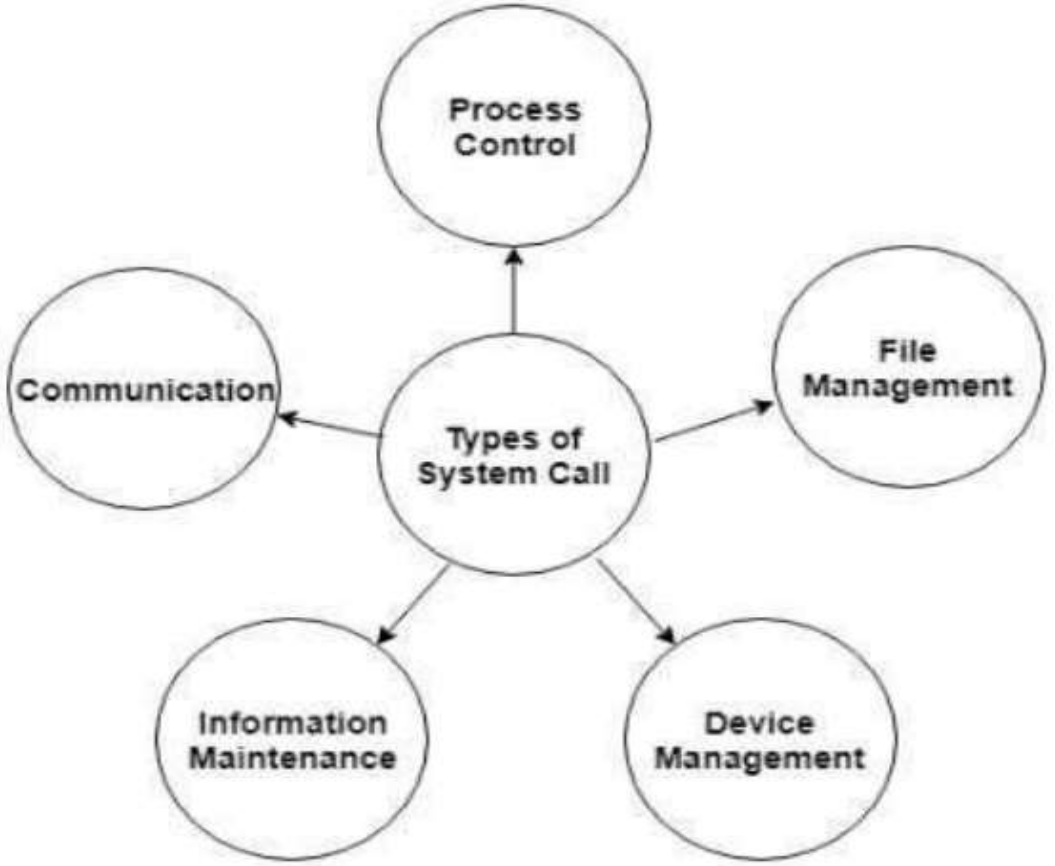
– Parameters placed, or pushed, onto the stack by the program and popped off the stack by the operating system

– Block and stack methods do not limit the number or length of parameters being passed

Parameter Passing via Table



Types of System Calls



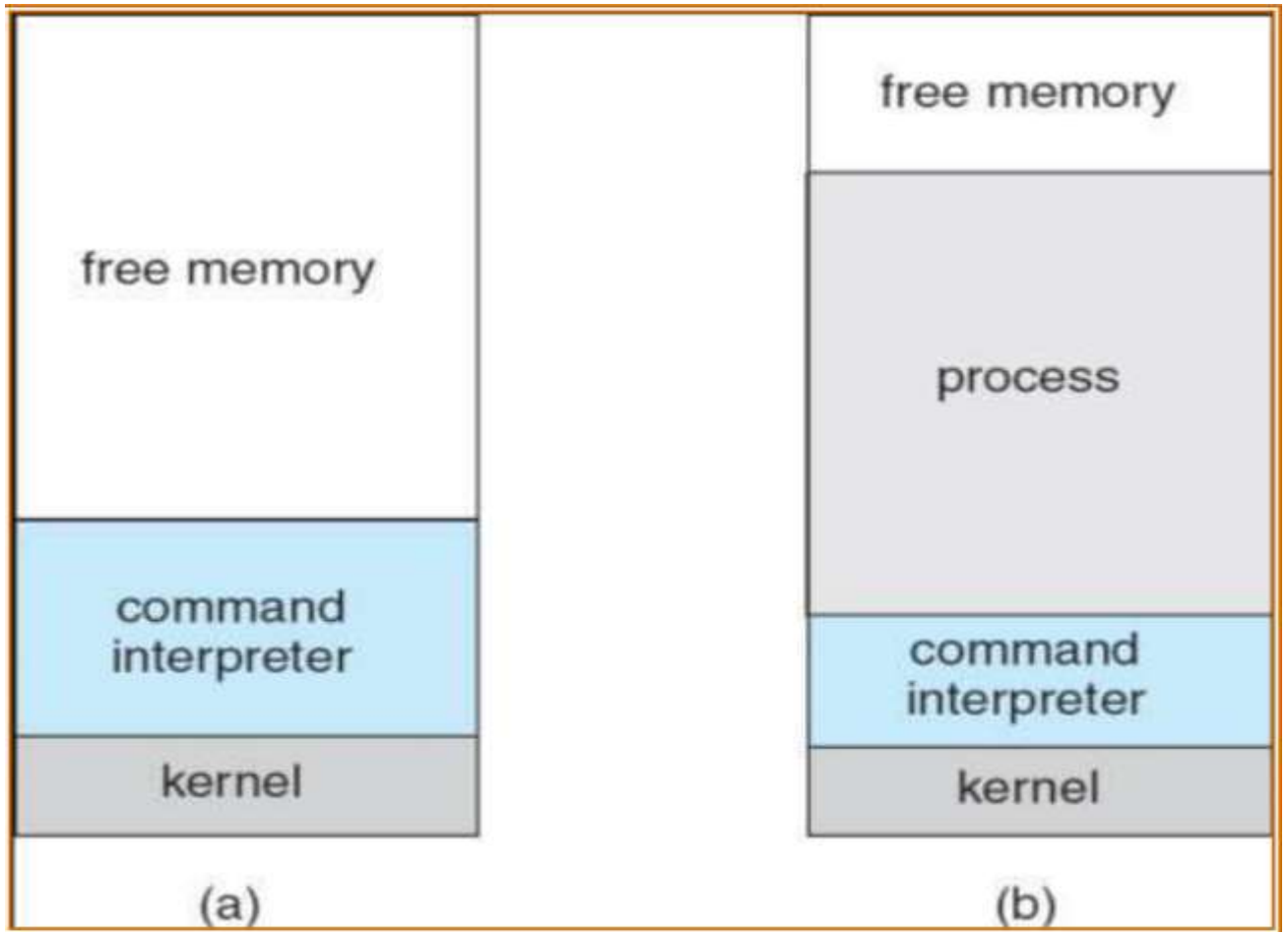
Types of System Calls

- **Process control**
  - create process, terminate process
  - end, abort
  - load, execute
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
  - Dump memory if error

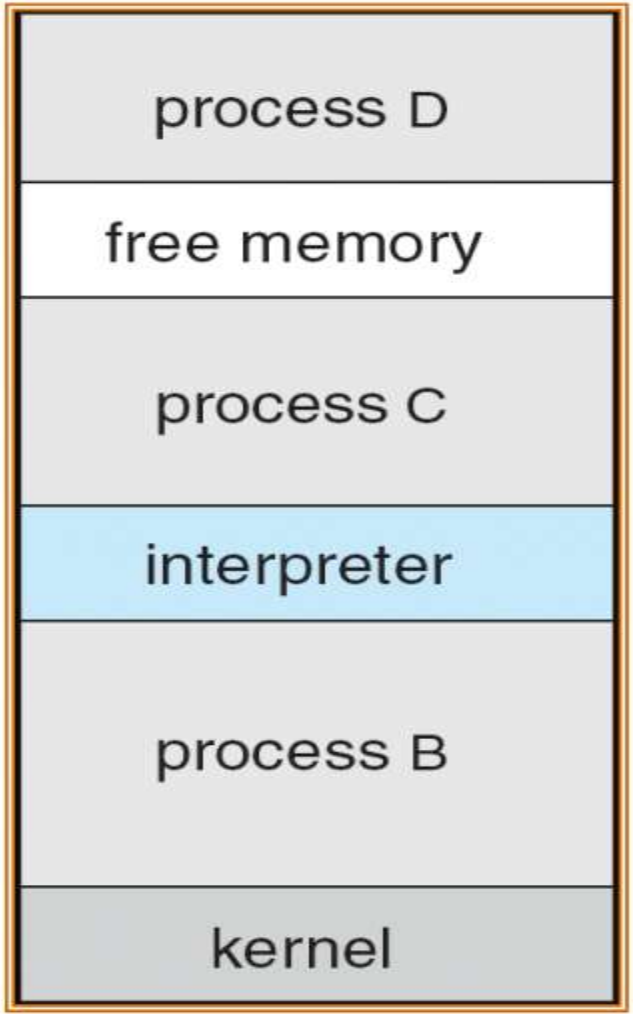
MS-DOS execution

(a)At system startup

(b) running a program



FreeBSD Running Multiple Programs



Types of System Calls

- **File management**
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes
- **Device management**
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices

Types of System Calls (cont)…

- **Information maintenance**
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes
- **Communications**
  - create, delete communication connection
  - send, receive messages if message passing model to host name or process name
    - From client to server
  - Shared-memory model create and gain access to memory regions
  - transfer status information

(cont)…

- **Protection**
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access.

Examples of Windows and Unix System Calls

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

System Programs

1.System programs provide a convenient environment for program development and execution. The can be divided into:

1. File manipulation
2. Status information
3. File modification
4. Programming language support
5. Program loading and execution
6. Communications
7. Application programs

(Cont)…

2. Most users' view of the operation system is defined by system programs, not the actual system calls System Programs

- Provide a convenient environment for program development and execution

– Some of them are simply user interfaces to system calls; others are considerably more complex

1.File management

- Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories

(Cont)…

2.Status information

– Some ask the system for info

- date, time, amount of available memory, disk space, number of users

– Others provide detailed performance, logging, and debugging information

– Typically, these programs format and print the output to the terminal or other output devices

– Some systems implement a registry

- used to store and retrieve configuration information

(Cont)…

3. File modification

– Text editors to create and modify files

– Special commands to search contents of files or perform transformations of the text

4. Programming-language support

- Compilers, assemblers, debuggers and interpreters sometimes provided

(Cont)…

5.Program loading and execution

- Absolute loaders, re-locatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language

6.Communications

- Provide the mechanism for creating virtual connections among processes, users, and computer systems

– Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

Operating System Design and Implementation

- Design and Implementation of OS not “solvable”, but some approaches have proven successful
- Internal structure of different Operating Systems can vary widely
- Start by defining goals and specifications
- Affected by choice of hardware, type of system.

(Cont)…

- User goals and System goals

– **User goals –** operating system should be convenient to use, easy to learn, reliable, safe, and fast

– **System goals –** operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

(Cont)…

- Important principle to separate

Policy: What will be done?

Mechanism: How to do it?

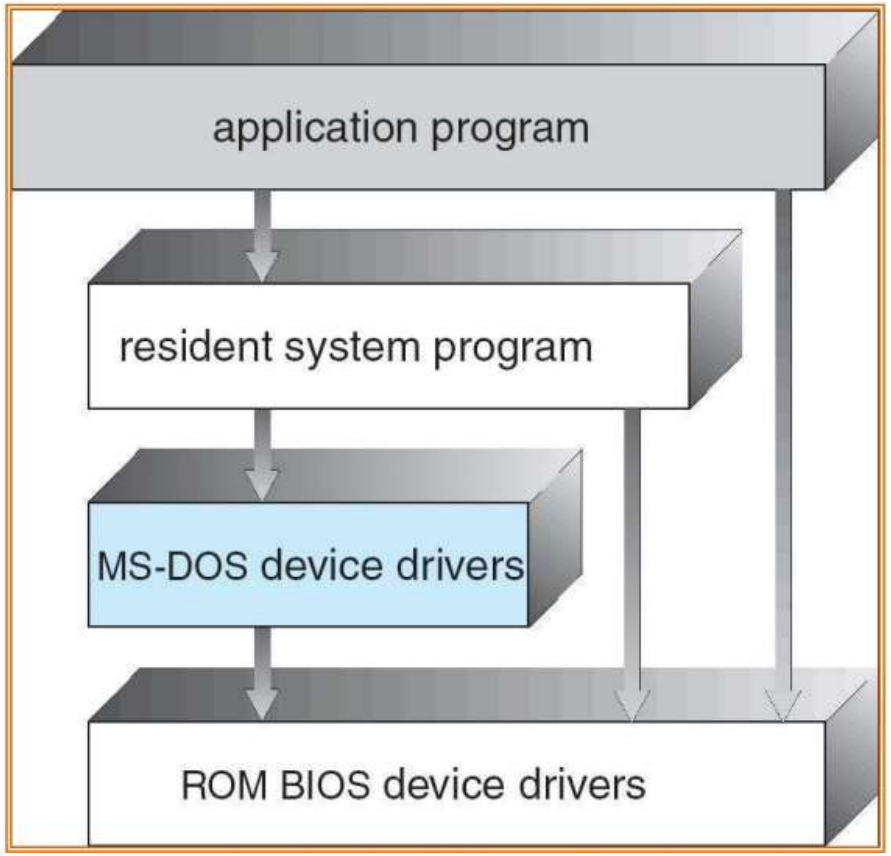
- Mechanisms determine how to do something, policies decide what will be done

– The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later

Simple Structure

- **MS-DOS**
  - written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated

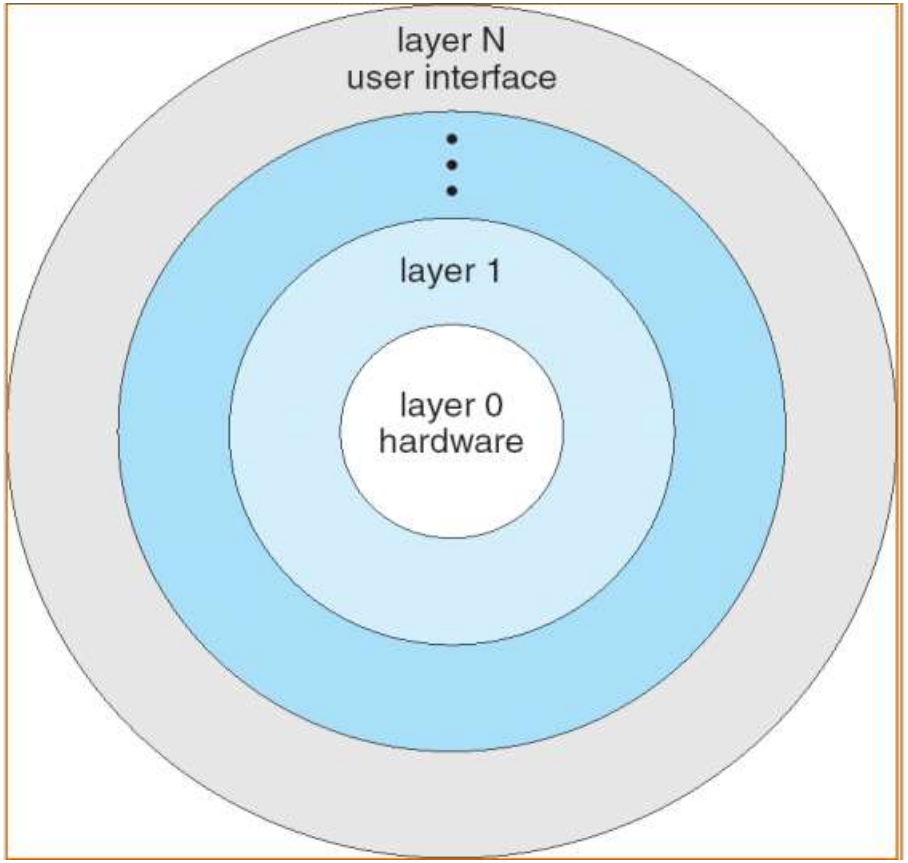
MS-DOS Layer Structure



Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers.
- The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers

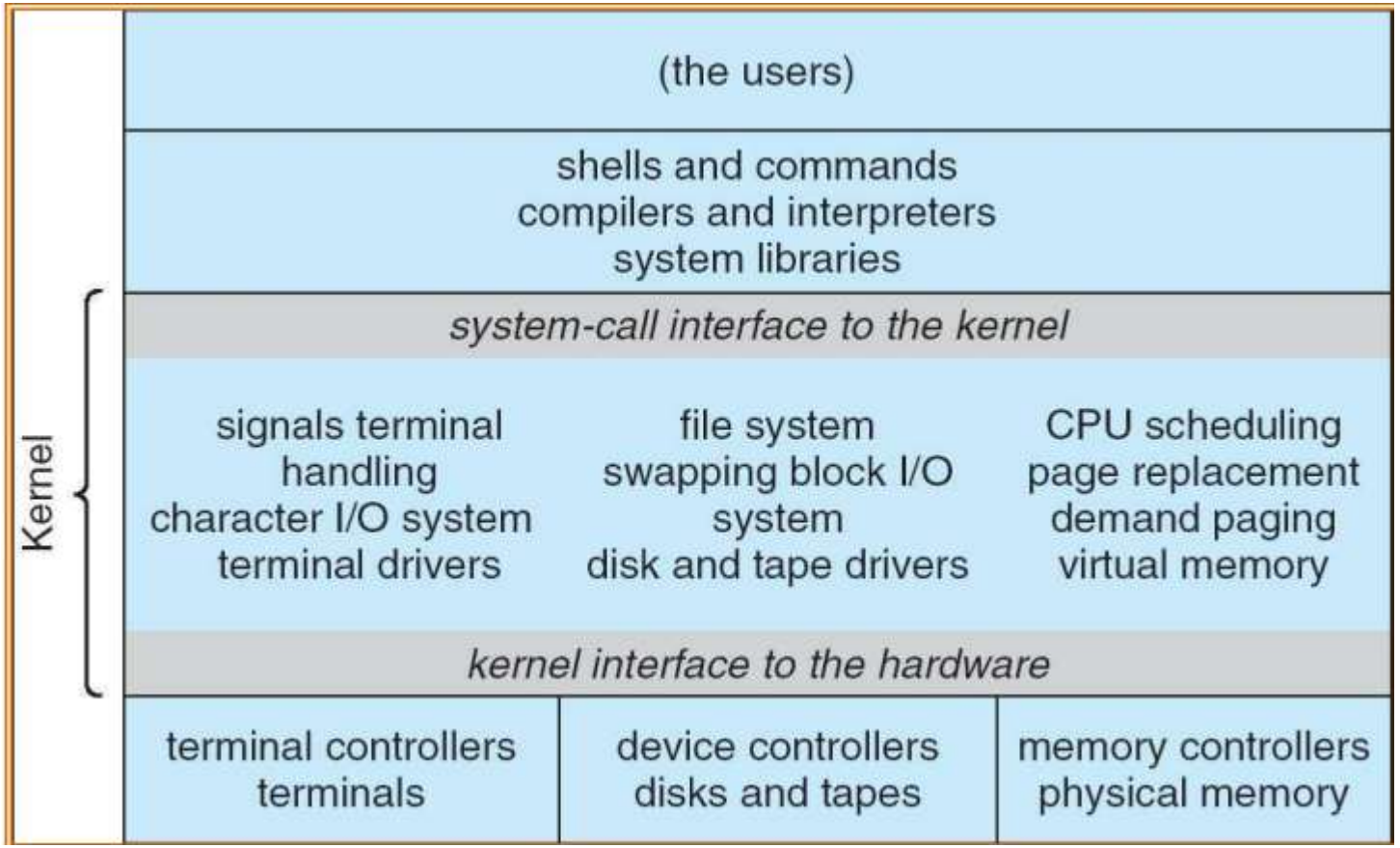
Layered Operating System



UNIX

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring.
- The UNIX OS consists of two separable parts
  - Systems programs
  - The kernel
- Consists of everything below the system-call interface and above the physical hardware
- Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level.

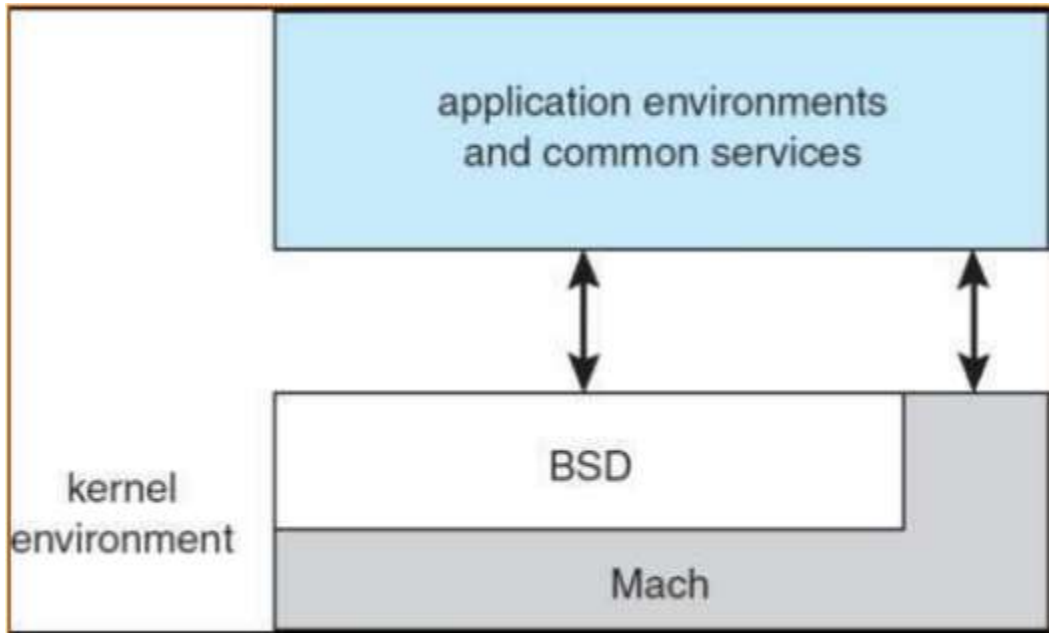
UNIX System Structure



Microkernel System Structure

- Moves as much from the kernel into “user” space
- Communication takes place between user modules using message passing
- **Benefits: –** Easier to extend a microkernel
  - – Easier to port the operating system to new architectures
  - – More reliable (less code is running in kernel mode)
  - – More secure
- **Detriments: –** Performance overhead of user space to kernel space communication

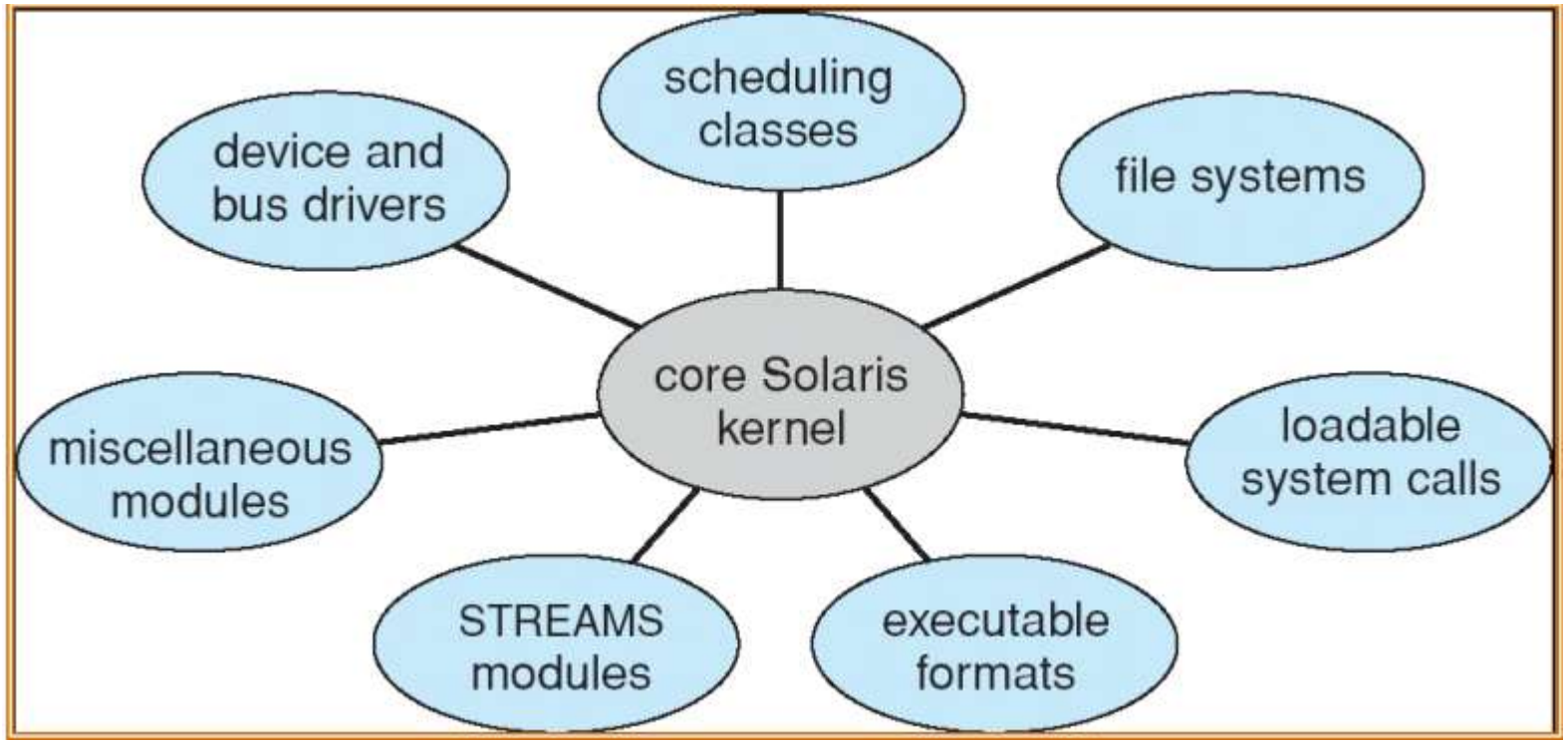
Mac OS X Structure



Modules

- Most modern operating systems implement kernel modules
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible

Solaris Modular Approach



Virtual Machines

- A virtual machine takes the layered approach to its logical conclusion.
- It treats hardware and the operating system kernel as though they were all hardware
- A virtual machine provides an interface identical to the underlying bare hardware
- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.

(Cont)…

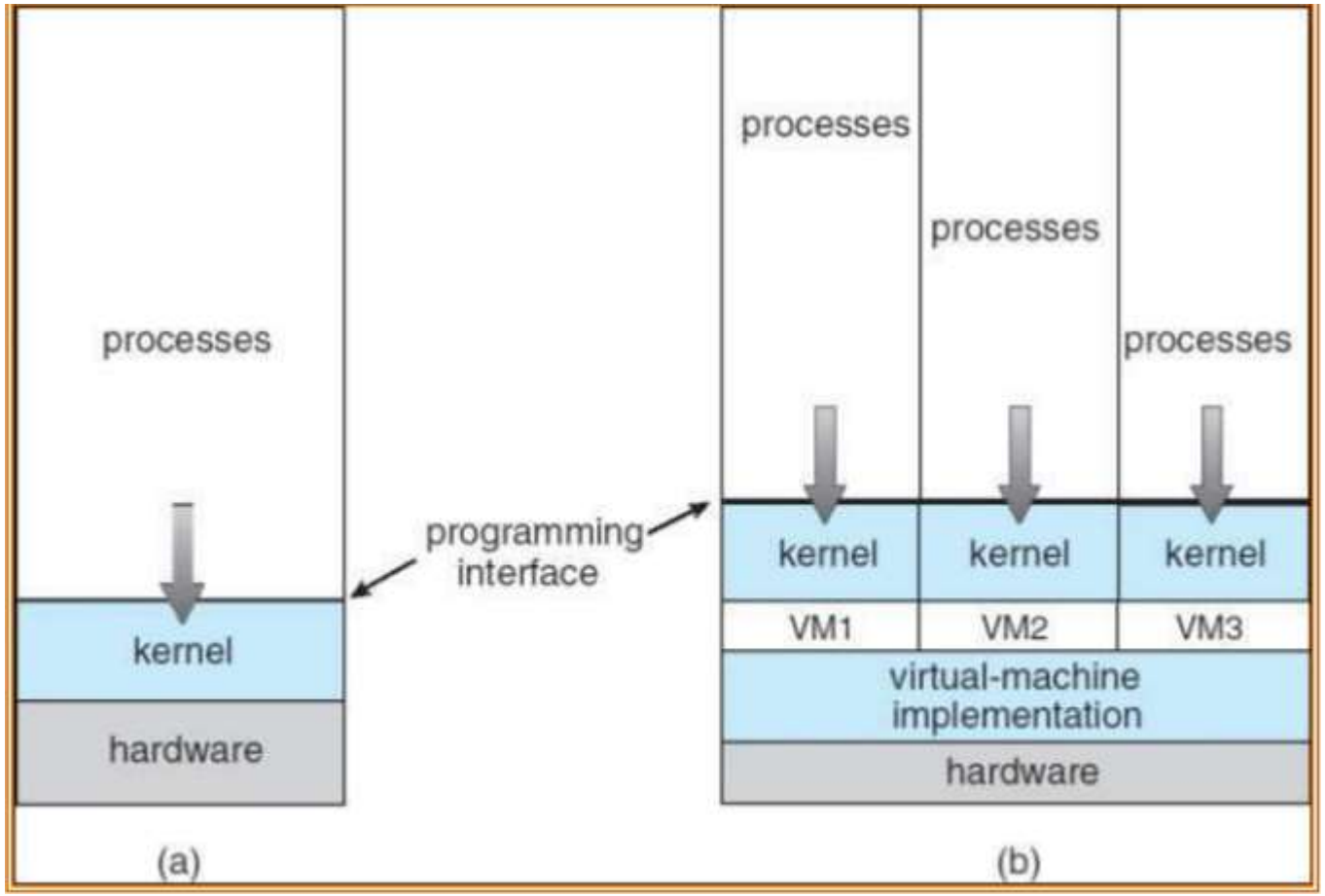
- The resources of the physical computer are shared to create the virtual machines

– CPU scheduling can create the appearance that users have their own processor

– Spooling and a file system can provide virtual card readers and virtual line printers

– A normal user time-sharing terminal serves as the virtual machine operator's console .

(Cont)…

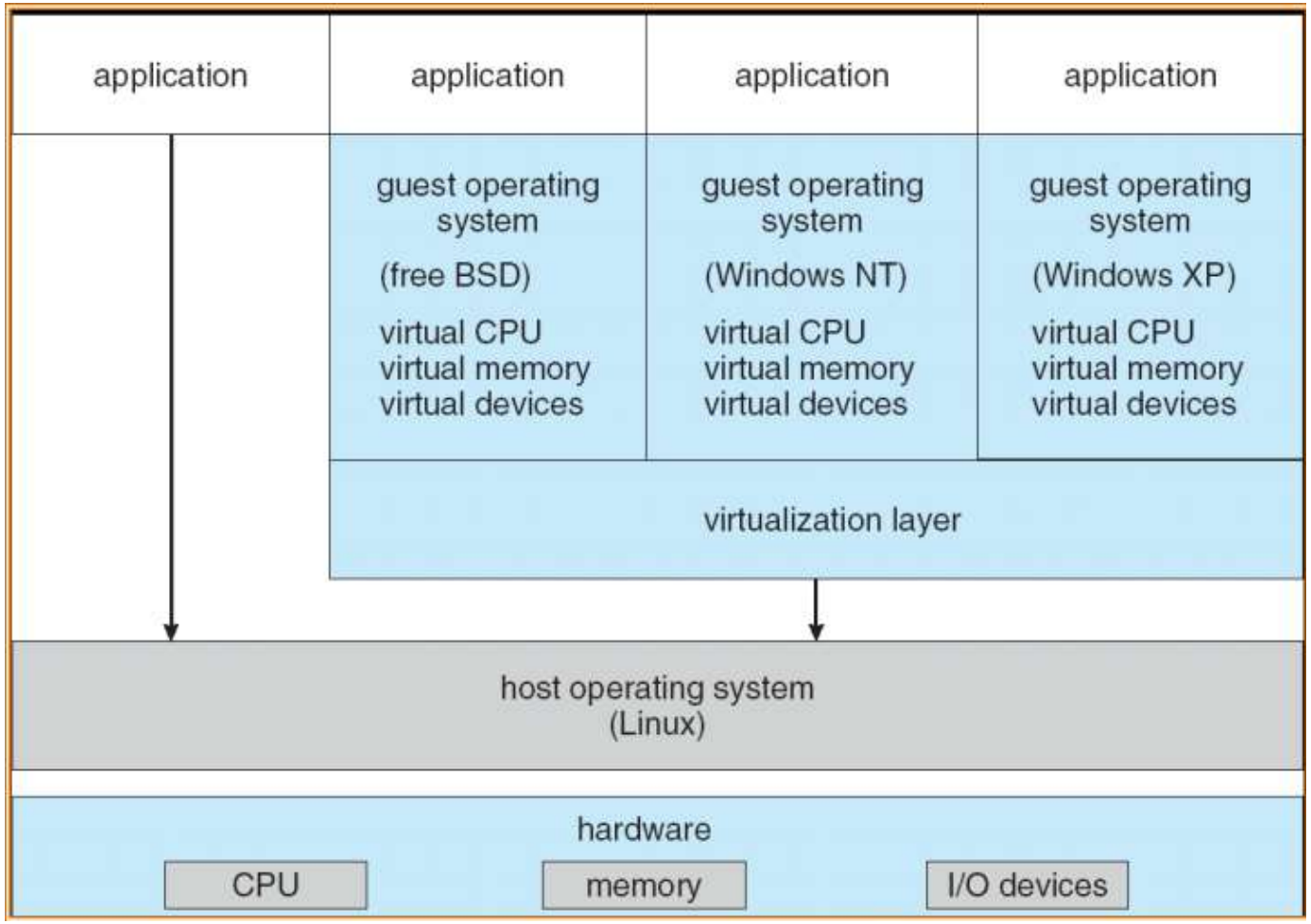


(a)Non-virtual machine
(b) virtual machine

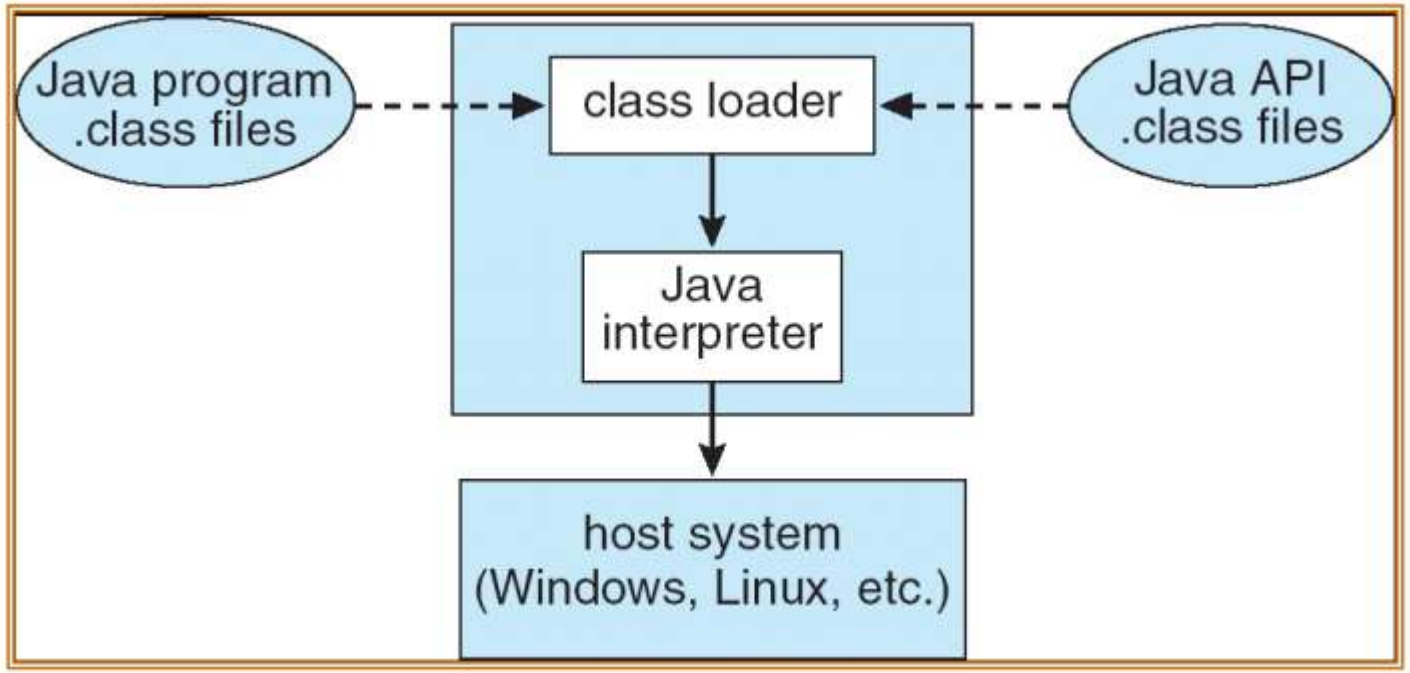
(Cont)…

- The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources.
- A virtual-machine system is a perfect vehicle for operating- systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
- The virtual machine concept is difficult to implement due to the effort required to provide an exact duplicate to the underlying machine

VMware Architecture



The Java Virtual Machine



Operating System Generation

- Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site
- SYSGEN program obtains information concerning the specific configuration of the hardware system
- Booting – starting a computer by loading the kernel
- Bootstrap program – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution.

Operating System Debugging

- Debugging is the activity of finding and fixing errors, or bugs in a system.
- Debugging seeks to find and fix errors in both hardware and software.

Stages of debugging:

1.Failure Analysis

2.Performance tuning.

System Boot

- Operating system must be made available to hardware so hardware can start it

– Small piece of code – **bootstrap loader**, locates the kernel, loads it into memory, and starts it

– Sometimes two-step process where boot block at fixed location loads bootstrap loader

– When power initialized on system, execution starts at a fixed memory location

- Firmware used to hold initial boot code.