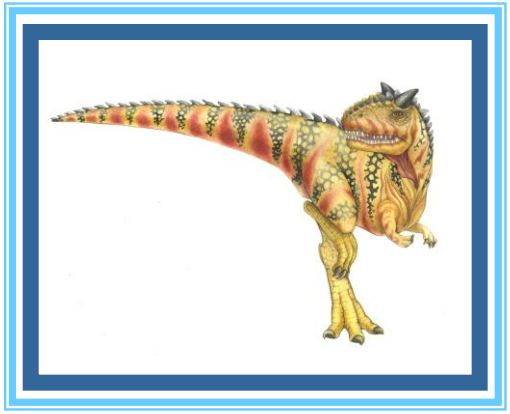
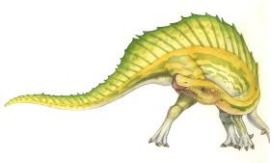
Chapter 1: Introduction



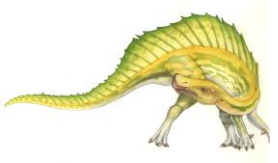
Outline

- What Operating Systems Do
- Computer-System Organization
- Computer-System Architecture
- Operating-System Operations
- Resource Management
- Security and Protection
- Virtualization
- Distributed Systems
- Kernel Data Structures
- Computing Environments
- Free/Libre and Open-Source Operating Systems



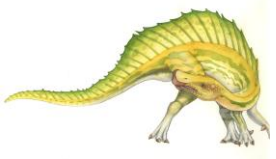
Objectives

- Describe the general organization of a computer system and the role of interrupts
- Describe the components in a modern, multiprocessor computer system
- Illustrate the transition from user mode to kernel mode
- Discuss how operating systems are used in various computing environments
- Provide examples of free and open-source operating systems

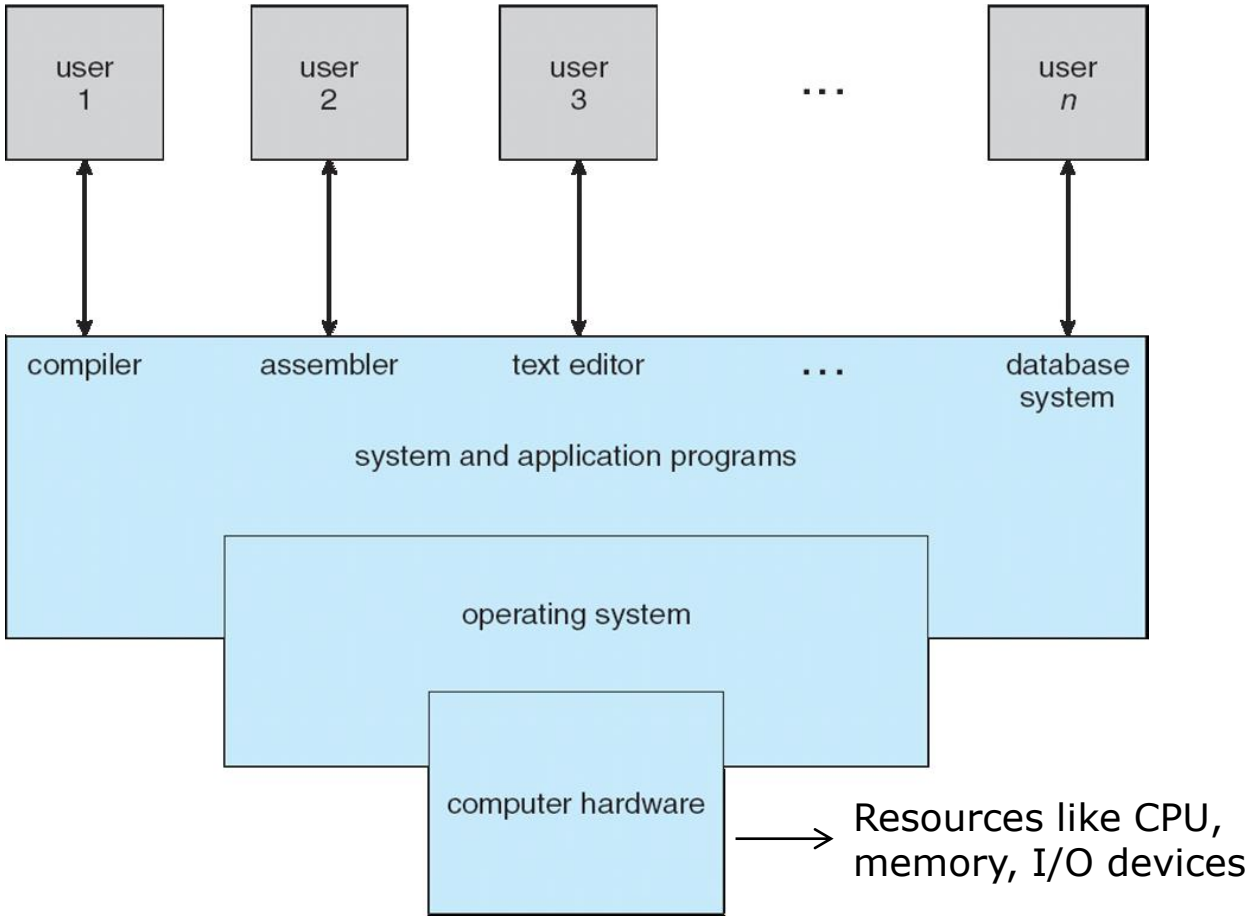


What is an Operating System?

- A program that manages the computer hardware.
- It also provides basis for application programs and acts as an intermediary between a user of a computer and the computer hardware

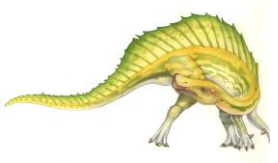


Abstract View of Components of Computer



Computer System Structure

- Computer system can be divided into four components:
  - Hardware – provides basic computing resources
    - CPU, memory, I/O devices
  - Operating system
    - Controls and coordinates use of hardware among various applications and users
  - Application programs – define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, video games
  - Users
    - People, machines, other computers



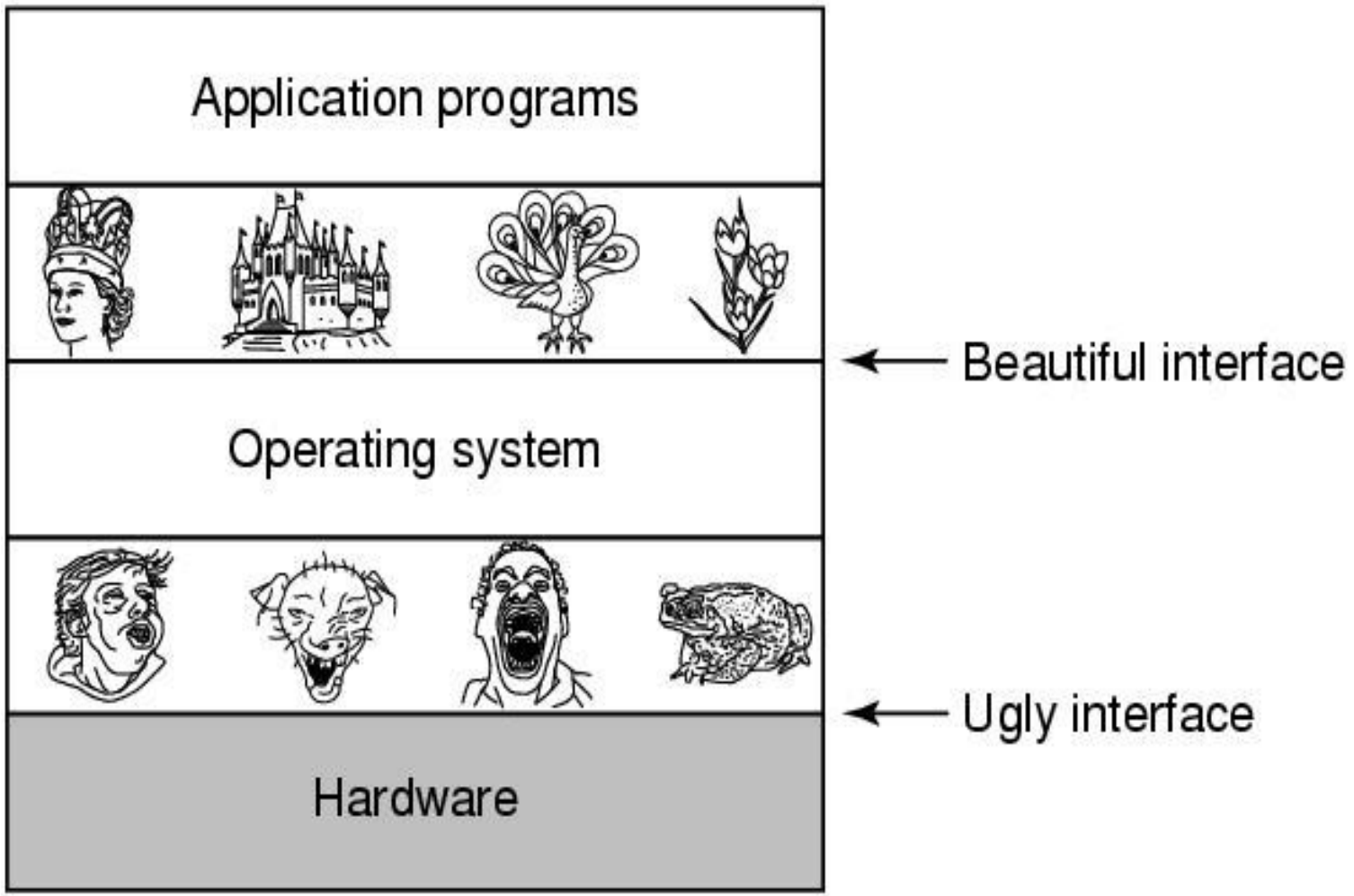
What is an Operating System?

- Types of OS:

Batch OS, Time sharing OS, Distributed OS, Network OS, Real-time OS, Multi programming/ Processing/ Tasking OS

- Operating system goals:
  - Make the computer system **convenient** to use
  - Use the computer hardware in an **efficient** manner
  - Both
- Operating system functions:
  - Interface between user and Hardware
  - Provide an environment in which user can easily interface with computer.
  - It is a resource allocator
  - Execute user programs and make solving user problems easier
  - Management of Memory, Security etc.

Abstract View of Components of Computer



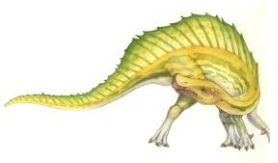
What Operating Systems Do

- Depends on the point of view
- User''s View (type of user)
- Standalone s/m - OS is designed for **ease of use** and **good performance**
- different terminals connected to **mainframe** or **minicomputer -** OS is designed to **maximize resource utilization**
- users of workstation, connected to networks and servers - **ease of use** and **resource utilization**
- users of handheld devices - **ease of use** and **good performance** per amount of battery

  - optimized for usability and battery life
- Embedded systems - designed for less user intearactions and **ease of use**

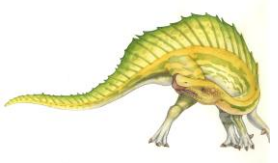
  - little or no user interface

  - designed to run primarily without user intervention



What Operating Systems Do (Cont.)

- System View:
  - 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



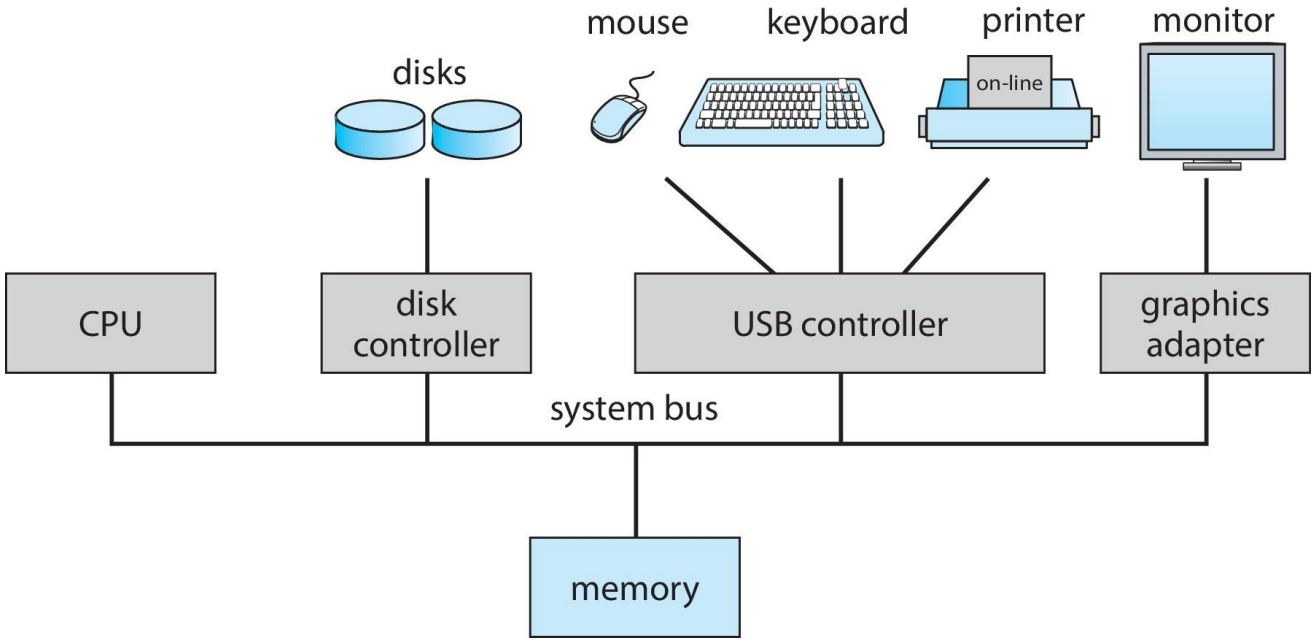
Operating System Definition

- No universally accepted definition
- "Everything a vendor ships when you order an operating system" is a good approximation
  - But varies wildly
- "The one program running at all times on the computer" is the **kernel,** which is part of the operating system
- Everything else is either
  - A **system program** (ships with the operating system, but not part of the kernel) , or
  - An **application program**, all programs not associated with the operating system
- Today's OSes for general purpose and mobile computing also include **middleware** – a set of software frameworks that provide addition services to application developers such as databases, multimedia, graphics etc.

Overview of Computer System Structure

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

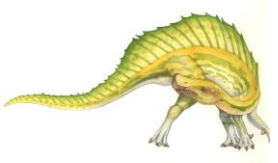


Computer-System Operation

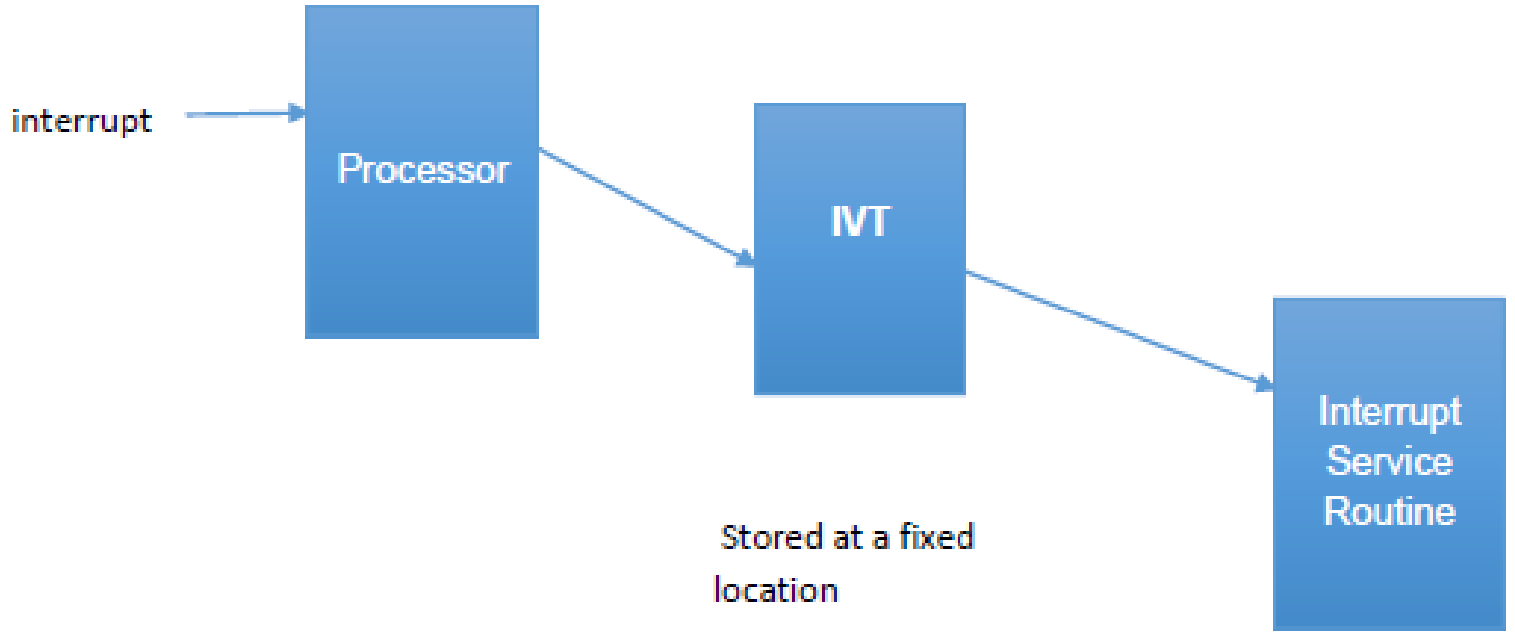
- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- Each device controller type has an operating system **device driver** to manage it
- CPU moves data from/to main memory to/from local buffers
- 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**

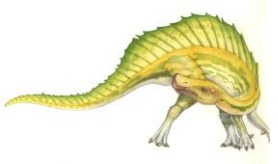
Interrupt Handling

- Occurence of an event is signalled by **interrupt**
- **Hardaware interrupt -** triggered by an event external to the processor
- **Software interrupt -** triggered by executing special intruction/operation (API or system calls)
- 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
- A **trap** or **exception** is a software-generated interrupt caused either by an error or a user request
- An operating system is **interrupt driven**



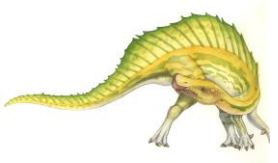
Interrupt Handling



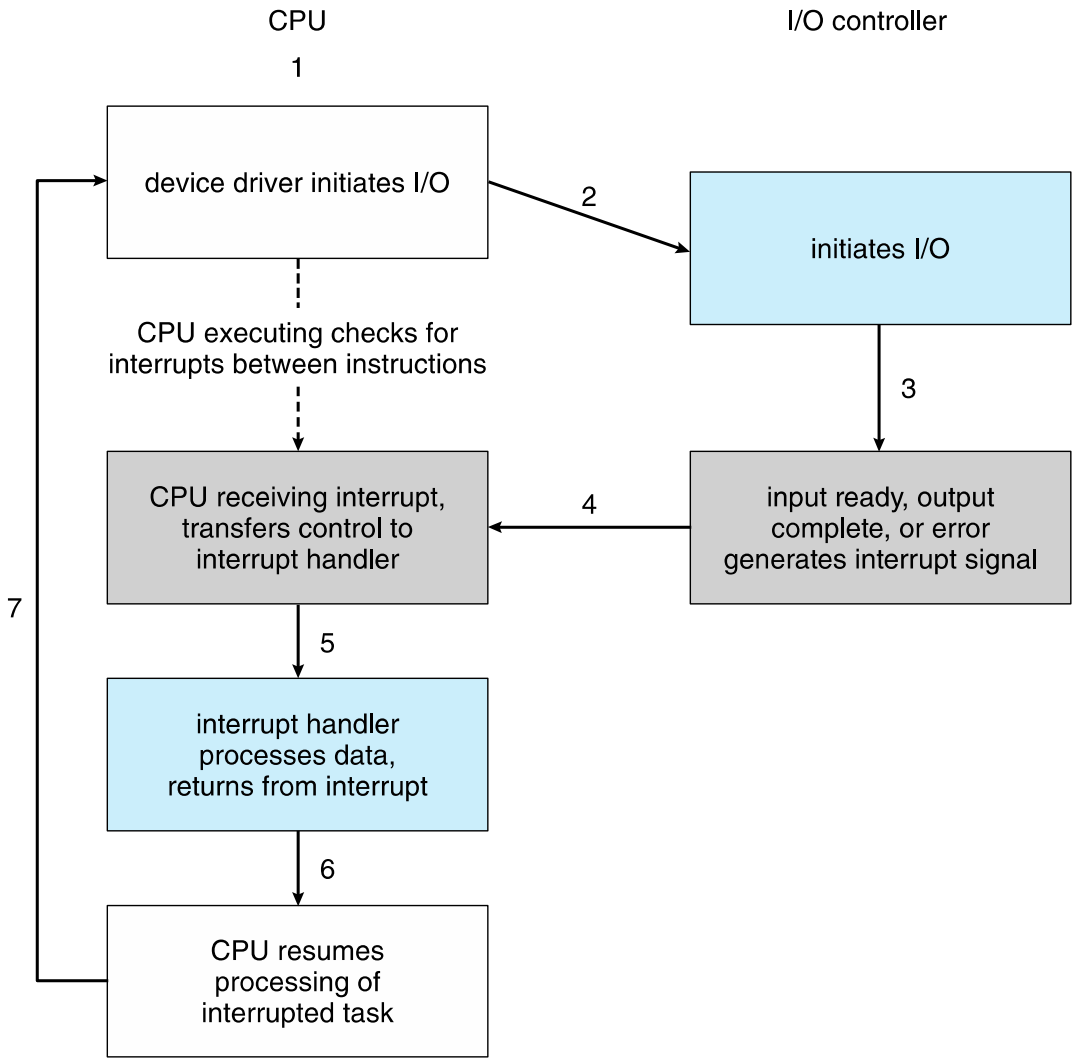


Interrupt Handling

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



Interrupt-drive I/O Cycle



Interrupt Driven Program Execution

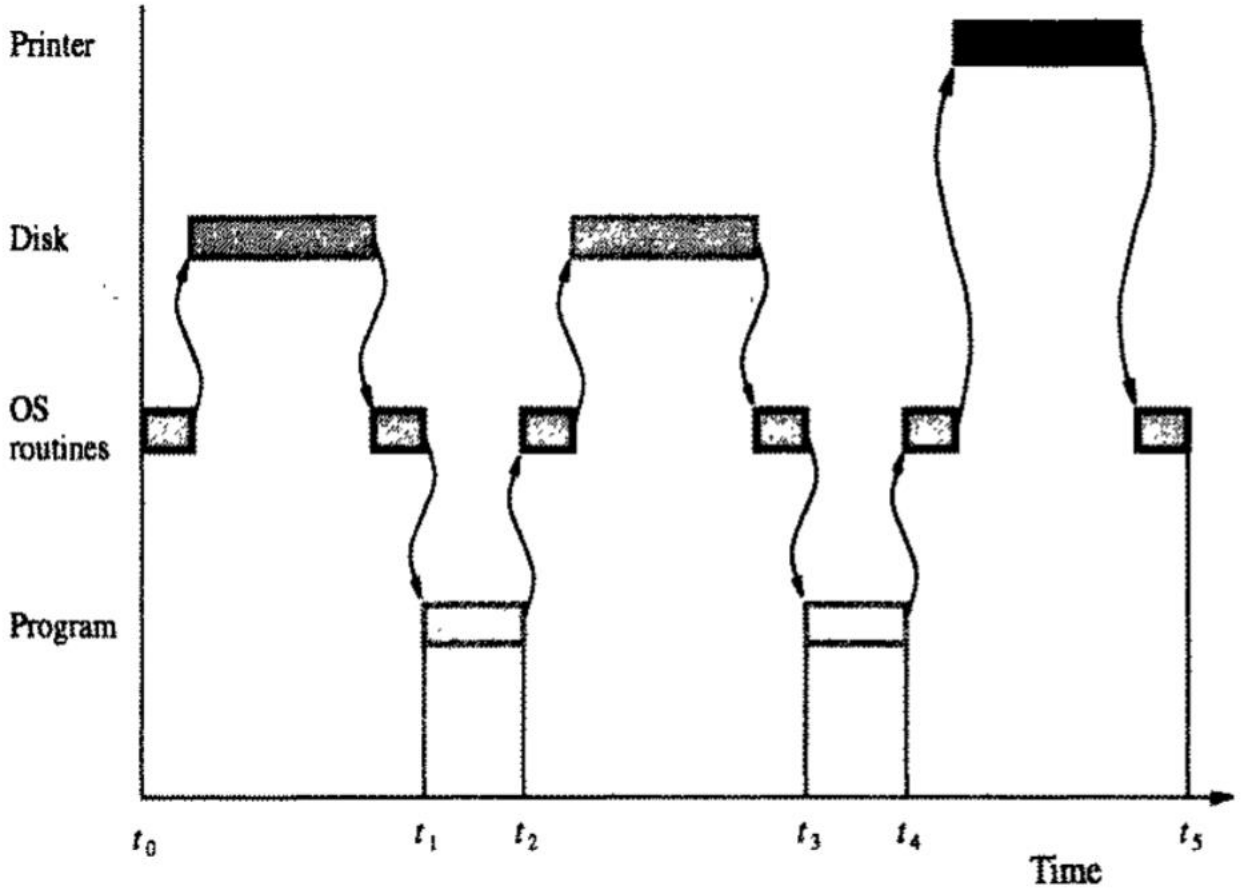
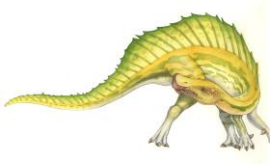
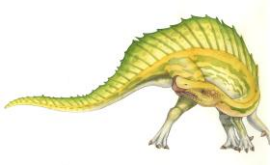


Figure 1.4 User program and OS routine sharing of the processor.



Storage Structure

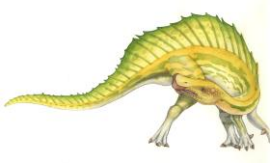


Storage Structure

- Main memory – only large storage media that the CPU can access directly
  - Typically, **volatile**
  - Typically, **random-access memory** in the form of **Dynamic Random-access Memory** (**DRAM**)
- Secondary storage – extension of main **memory** that provides large **nonvolatile** storage capacity

Storage Structure (Cont.)

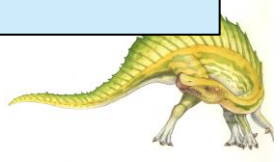
- **Hard Disk Drives** (**HDD**) – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into **tracks**, which are subdivided into **sectors**
  - The **disk controller** determines the logical interaction between the device and the computer
- **Non-volatile memory** (**NVM**) devices– faster than hard disks, nonvolatile
  - Various technologies
  - Becoming more popular as capacity and performance increases, price drops



Storage Definitions and Notation Review

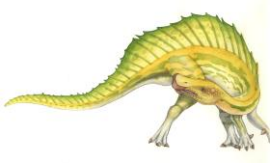
The basic unit of computer storage is the **bit** . A bit can contain one of two values, 0 and 1. All other storage in a computer is based on collections of bits. Given enough bits, it is amazing how many things a computer can represent: numbers, letters, images, movies, sounds, documents, and programs, to name a few. A **byte** is 8 bits, and on most computers, it is the smallest convenient chunk of storage. For example, most computers don't have an instruction to move a bit but do have one to move a byte. A less common term is **word**, which is a given computer architecture's native unit of data. A word is made up of one or more bytes. For example, a computer that has 64-bit registers and 64-bit memory addressing typically has 64-bit (8-byte) words. A computer executes many operations in its native word size rather than a byte at a time.

Computer storage, along with most computer throughput, is gen**er**ally measured and manipulated in bytes and collections of bytes. A **kilobyte** , or KB , is 1,024 bytes; a **megabyte** , or **MB** , is $1,024^2$ bytes; a **gigabyte** , or GB , is $1,024^3$ bytes; a **terabyte** , or **TB** , is $1,024^4$ bytes; and a **petabyte** , or **PB** , is $1,024^5$ bytes. Computer manufacturers often round off these numbers and say that a megabyte is 1 million bytes and a gigabyte is 1 billion bytes. Networking measurements are an exception to this general rule; they are given in bits (because networks move data a bit at a time).

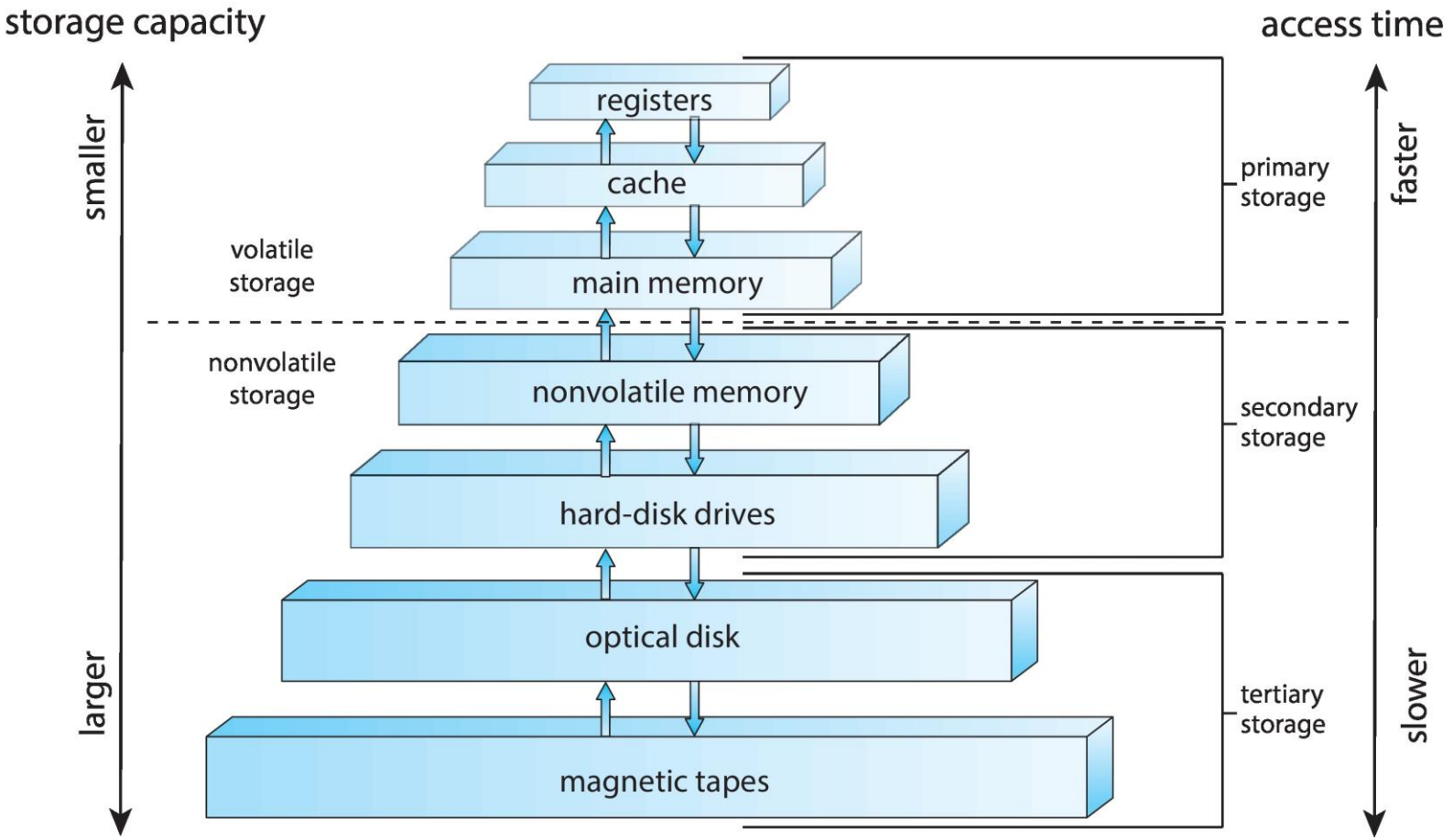


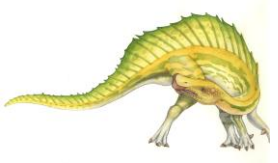
Storage Hierarchy

- Storage systems organized in hierarchy
  - Speed
  - Cost
  - Volatility
- **Caching** – copying information into faster storage system; main memory can be viewed as a cache for secondary storage
- **Device Driver** for each device controller to manage I/O
  - Provides uniform interface between controller and kernel



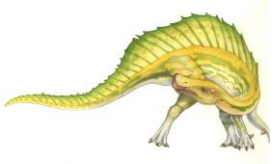
Storage-Device Hierarchy



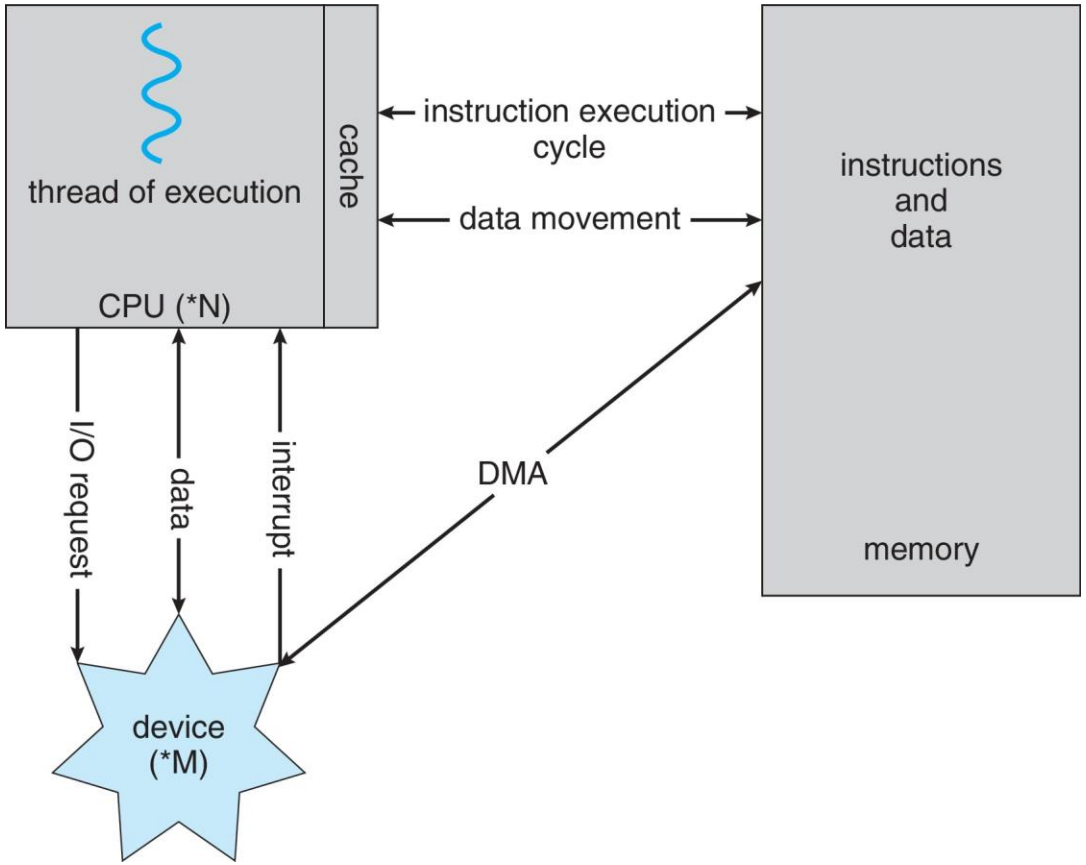


I/O Structure

- A large portion of operating system code is dedicated to managing I/O.
  - Varying nature of devices
  - Performance of a system
- Each device has device controller with local buffers and set of special purpose registers.
- OSs have device driver for each device controller.
  - Part of the OS kernel
  - Run with the same privilege level as the OS kernel
- To start an I/O operation, the device driver loads the registers within the device controller.
- The device controller, examines the contents of these registers and starts the transfer of data from the device to its local buffer.
- Finally, the device driver returns control to the operating system, and also returns the data and status information. (Interrupt driven I/O)



I/O Structure



A von Neumann architecture

Direct Memory Access Structure

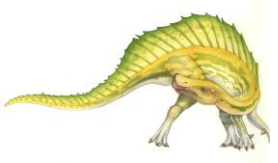
- Used for high-speed I/O devices able to transmit information at close to memory speeds
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only one interrupt is generated per block, rather than the one interrupt per byte

Computer-System Architecture

- Categorized based on number of general-purpose processor
- **Single-Processor Systems** use a single processor (PDAs to mainframes)
  - One main CPU executing user processes
  - Contain special purpose processors – run limited instructions, do not run user processes and managed by OS

    Eg1: Disk controller processor - implements its own disk queue and scheduling algorithm

    Eg2: Special processors in the keyboard, converts the keystrokes into codes to be sent to the CPU



Computer-System Architecture

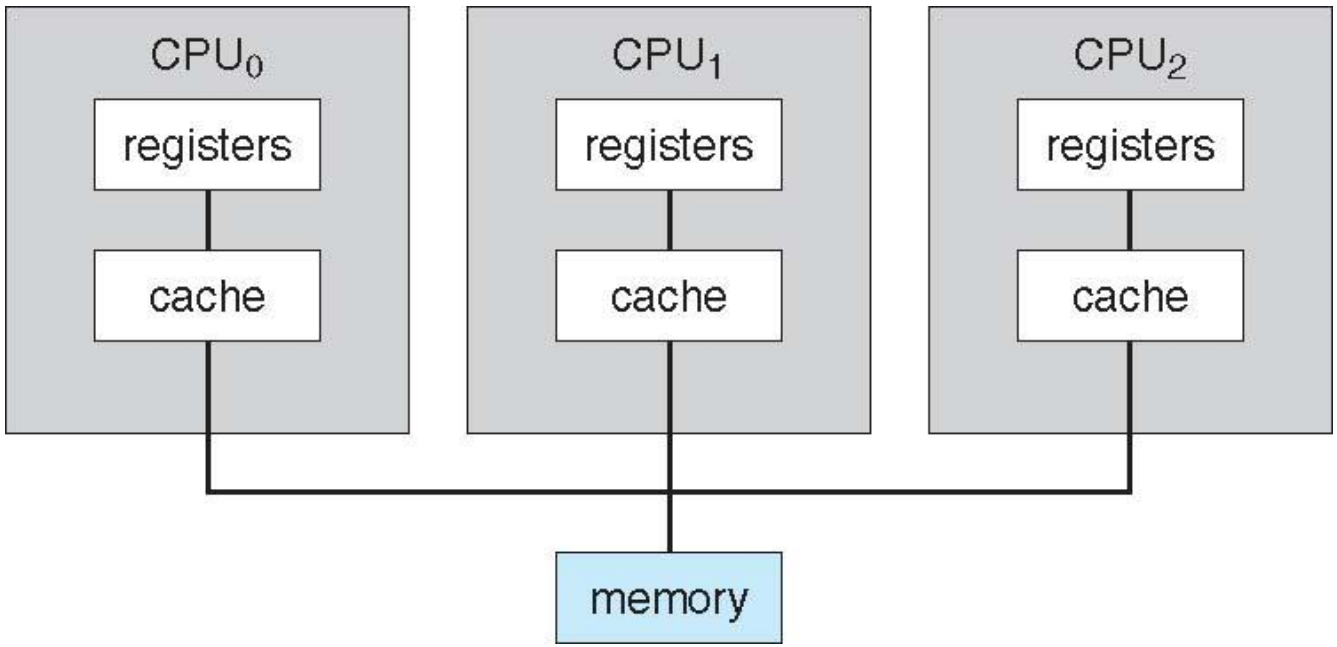
- **Multiprocessors** systems growing in use and importance
  - Also known as **parallel systems**, **tightly-coupled systems**
  - Systems that have two or more processors in close communication, sharing the computer bus, the clock, memory, and peripheral devices
  - Advantages include:
    1. **Increased throughput :** No. programs executed per unit time is more; increasing number of processors do not always guarantee increased performance due to the overhead of comlexity and cost
    2. **Economy of scale** – costs less than equivalent no. of many single processors due to sharing of peripherals, mass storage and power supplies. Even data can be shared among several processes.
    3. **Increased reliability** – if one processor fails, the s/m is not halted, it only slows down; the job of the failed processor is taken up by other processor

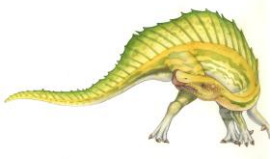
    - graceful degradation or fault tolerant

Symmetric Multiprocessing Architecture

Two types of multiprocessor systems:

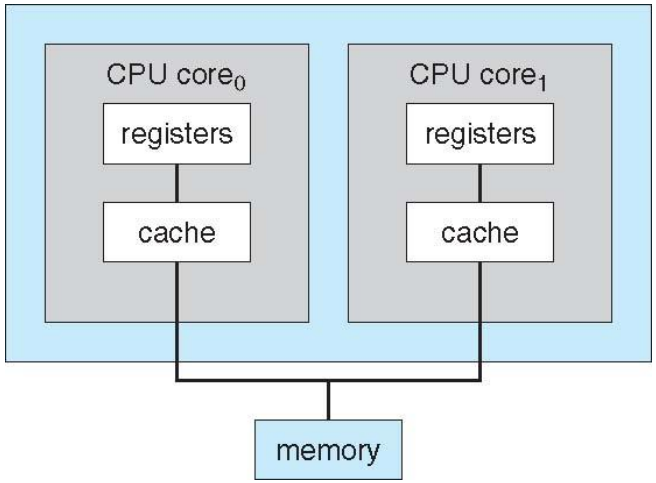
1. **Asymmetric Multiprocessing (Master/Slave)** – each processor is assigned a specific task by master processor. It schedules and allocates work to slave processors
2. **Symmetric Multiprocessing (SMP)**– each processor performs all tasks; all processors are considered peers; Windows, MacOS, Liux support SMP

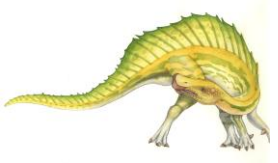




A Dual-Core Design

- Multiple processing units (computing cores) fabricated on a single chip - **multicore processors**
- The term processor is then used for the complete chip
- The communication b/w processors within a chip is more faster than communication between two single processors



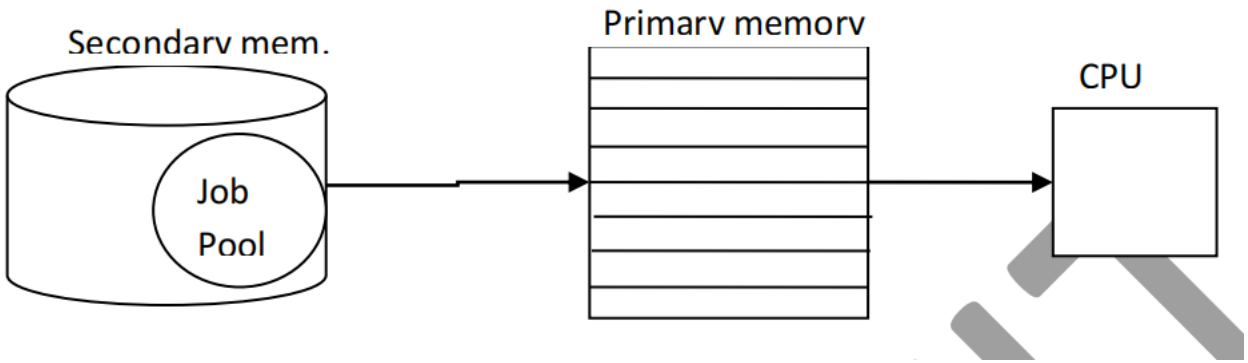


Clustered Systems

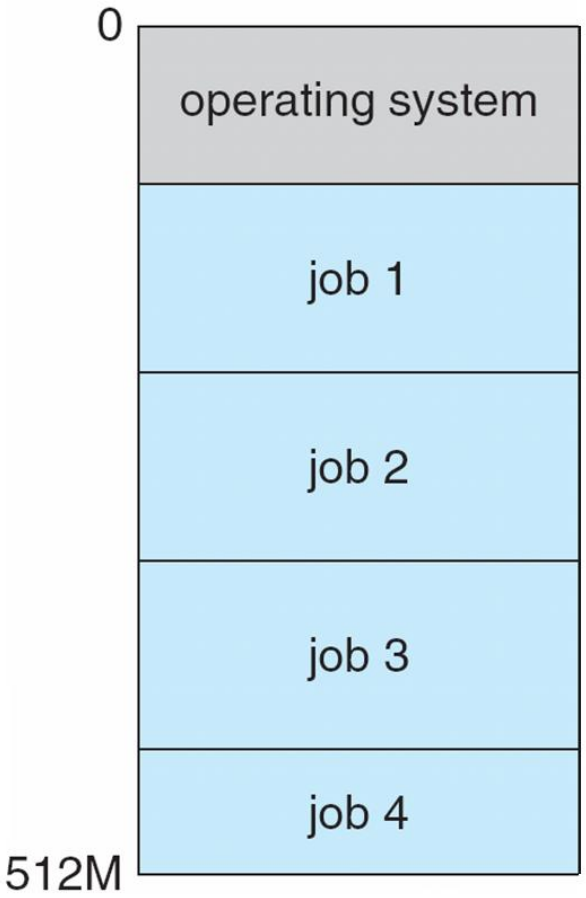
- Like multiprocessor systems, but multiple systems working together via network and sharing software resources
  - Provides a **high-availability** of resources and services which survive failures;the service continues even if one or more systems in the cluster fail; duplication of s/w resources
    - **Asymmetric clustering** has one system in hot-standby mode monitoring the server.
    - **Symmetric clustering** has multiple systems running applications, monitoring each other. Uses all of the available resources.
  - Some clusters are for **high-performance computing (HPC)**
    - Applications must be written to use **parallelization**
  - Parallel clusters allow multipe hosts to access same data on the shared storage. Cluster technology is changing rapidly with storage-area network (SAN) - where resources can be shared with dozens of systems in a cluster, that are separated by miles.

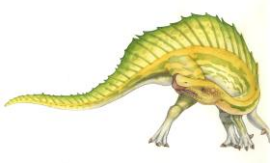
Operating System Structure

- **Multiprogramming** (**Batch system**) 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 and executes another job
  - Multiprogrammed s/ms provide envoronment in which various s/m resources are utilized effectively, but they do not provide user interaction with the computer system



Memory Layout for Multiprogrammed System



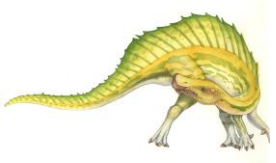


Operating System Structure

- **Timesharing** (**multitasking**) is logical extension in which a single CPU executes multiple jobs by switching among them

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

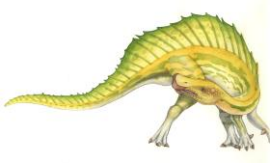


Operating-System Operations

- **Interrupt driven** (hardware and software)
  - Hardware interrupt by one of the devices
  - Software interrupt (**exception** or **trap):**
    - Software error (e.g., 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

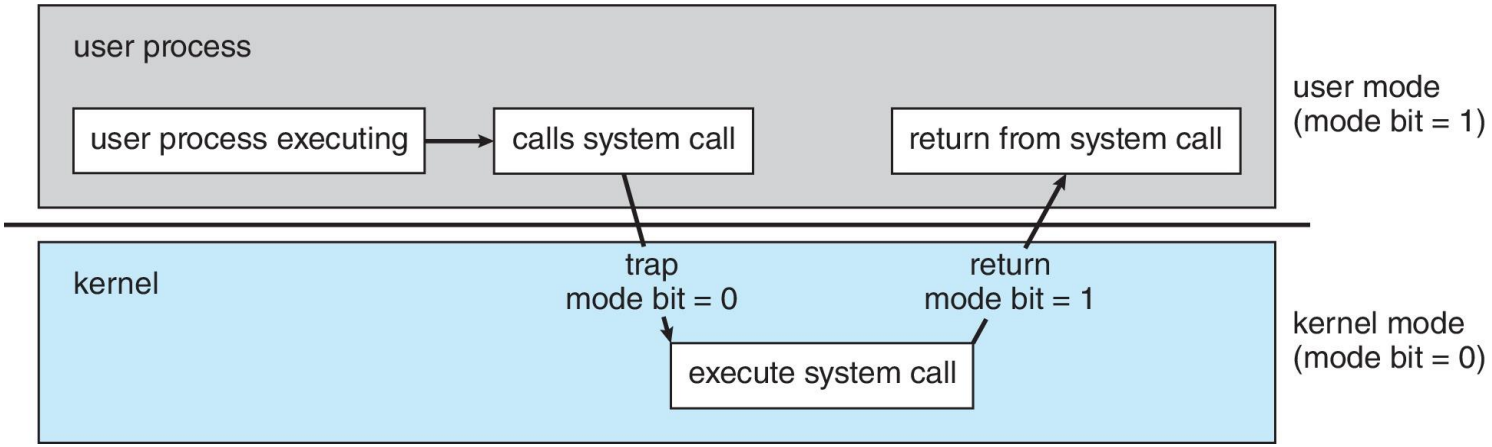
- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
- **Mode bit** provided by hardware
  - Provides ability to distinguish when system is running user code or kernel code.
  - When a user is running ➔ mode bit is “user” (set to 1)
  - When kernel code is executing ➔ mode bit is “kernel” (set to 0)
- Some instructions designated as **privileged**, only executable in kernel mode.
- Eg: changing the mode bit; changing the priority level etc.



Dual-mode Operation (Cont.)

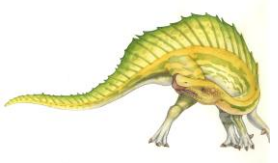
- How do we guarantee that user does not explicitly set the mode bit to “kernel”?
- When the system starts executing it is in kernel mode
- When control is given to a user program the mode-bit changes to “user mode”.
- When a user issues a system call it results in an interrupt, which trap to the operating system. At that time, the mode–bit is set to “kernel mode”.

Transition from User to Kernel Mode



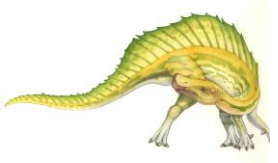
Timer

- Operating system uses timer to control the CPU. A user program cannot hold CPU for a long time, this is prevented with the help of timer.
- Timer to prevent infinite loop (or process hogging resources)
  - Timer is set to interrupt the computer after some time period
  - ➢ Fixed timer - After a fixed time, the process under execution is interrupted.`
  - ➢ Variable timer - Interrupt occurs after varying interval.
  - Fixed clock and a counter that is decremented every time the clock ticks
  - Operating system set the counter (privileged instruction)
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time



Process Management

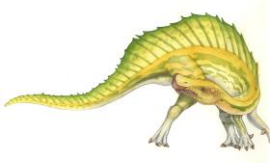
- A process is a program in execution. It is a unit of work within the system. Program is a ***passive entity;*** process is an ***active entity***.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Process termination requires reclaim of any reusable 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
- 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

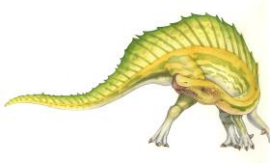
The operating system is responsible for the following activities in connection with process management:

- 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



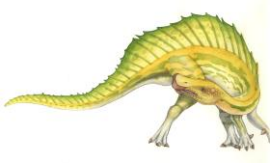
Memory Management

- To execute a program all (or part) of the instructions must be in memory
- All (or part) of the data that is needed by the program must be in memory
- To improve both the utilization of the CPU several programs are kept in memory, creating a need for memory management.
- 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 de-allocating memory space as needed



File-system Management

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - **file**
  - Different types of physical media - Magnetic disk, optical disk, and magnetic tape`
  - 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
  - Multiple users - Access control on most systems to determine who can access what
  - OS activities include
    - Creating and deleting files and directories
    - Supporting primitives to manipulate files and directories
    - Mapping files onto secondary storage
    - Backup files onto stable (non-volatile) storage media

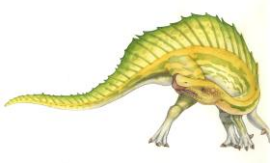


Mass-Storage Management

- Main memory is small and volatile – secondary storage back up main maemory
- 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
  - Mounting and unmounting
  - Free-space management
  - Storage allocation
  - Disk scheduling
  - Partitioning
  - Protection

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



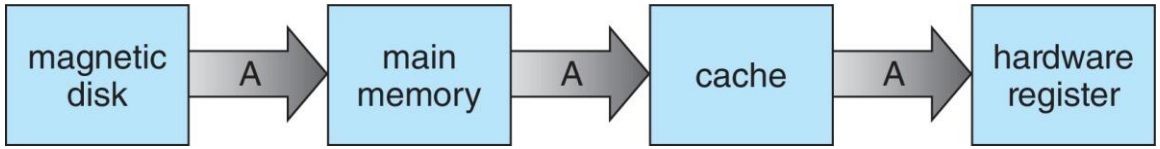
Characteristics of Various Types of Storage

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

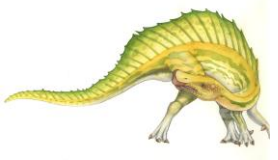
Movement between levels of storage hierarchy can be explicit or implicit

Migration of data “A” from Disk to Register

- Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy



- Multiprocessor environment must provide **cache coherency** in hardware such that all CPUs have the most recent value in their cache
- Distributed environment situation even more complex
  - Several copies of a datum can exist
  - Various solutions covered in Chapter 19

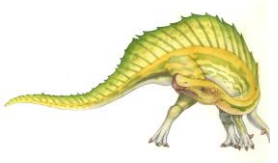


I/O Subsystem

- 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** – 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

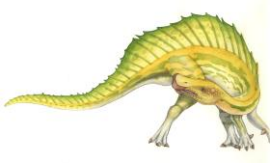


Protection

- 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

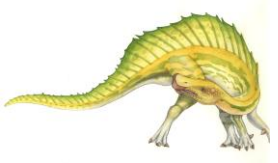
Computing Environments - Traditional

- Stand-alone general purpose machines
- But blurred as most systems interconnect with others (i.e., the Internet)
- **Portals** provide web access to internal systems
- **Network computers** (**thin clients**) are like Web terminals
- Mobile computers interconnect via **wireless networks**
- Networking becoming ubiquitous – even home systems use **firewalls** to protect home computers from Internet attacks



Computing Environments - Mobile

- Handheld smartphones, tablets, etc
- What is the functional difference between them and a “traditional” laptop? Limited physical memory, speed of processor, small screens, battery back-up, usage of I/O devices.
- Extra feature – more OS features (GPS, gyroscope)
- Allows new types of apps like ***augmented reality***
- Use IEEE 802.11 wireless, or cellular data networks for connectivity
- Leaders are **Apple iOS** and **Google Android**

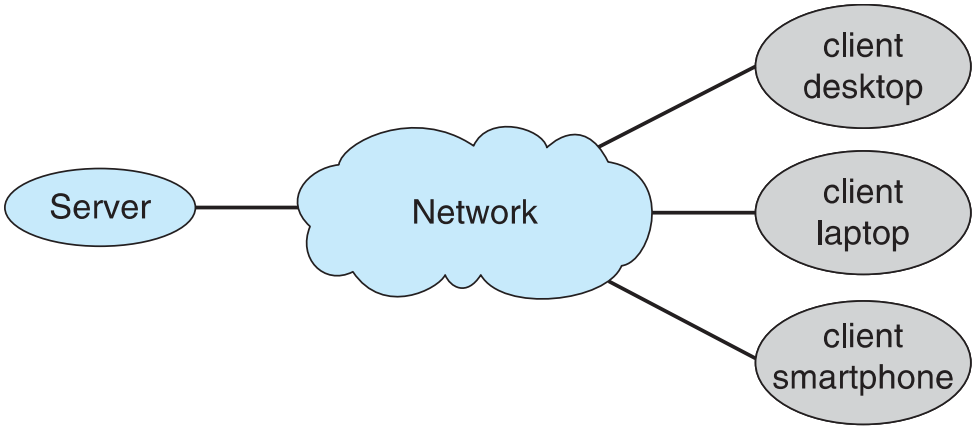


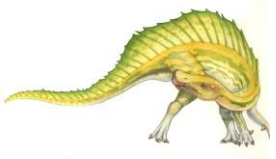
Computing Environments – Distributed

- Distributed computiing
  - Collection of separate, possibly heterogeneous, systems networked together to access various resources in the network.
    - **Network** is a communications path, **TCP/IP** most common
      - **Local Area Network** (**LAN**)
      - **Wide Area Network** (**WAN**)
      - **Metropolitan Area Network** (**MAN**)
      - **Personal Area Network** (**PAN**)
  - **Network Operating System** provides features between systems across network
    - Communication scheme allows systems to exchange messages
    - Illusion of a single system

Computing Environments – Client-Server

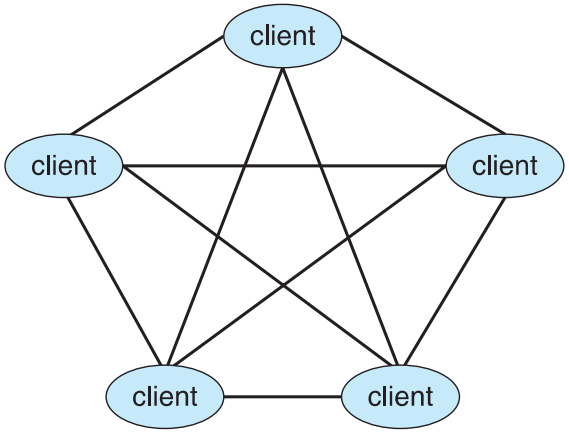
- Client-Server Computing – specialized distributed system
  - Dumb terminals supplanted by smart PCs
  - Many systems now **servers**, responding to requests generated by **clients**
    - **Compute-server system** provides an interface to client to request services (Eg. reading data)
    - **File-server system** provides interface for clients to store, create, update, delete, read, retrieve files (Eg. Web server that delivers files to clients running the web browser)

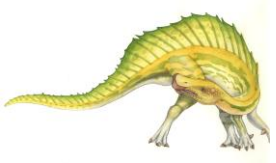




Computing Environments - Peer-to-Peer

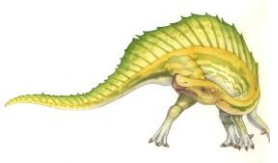
- Another model of distributed system
- P2P does not distinguish clients and servers
  - Instead all nodes are considered peers
  - May each act as client, server or both
  - Node must join P2P network
    - Registers its service with central lookup service on network, or
    - Broadcast request for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





Computing Environments – Real-Time Embedded Systems

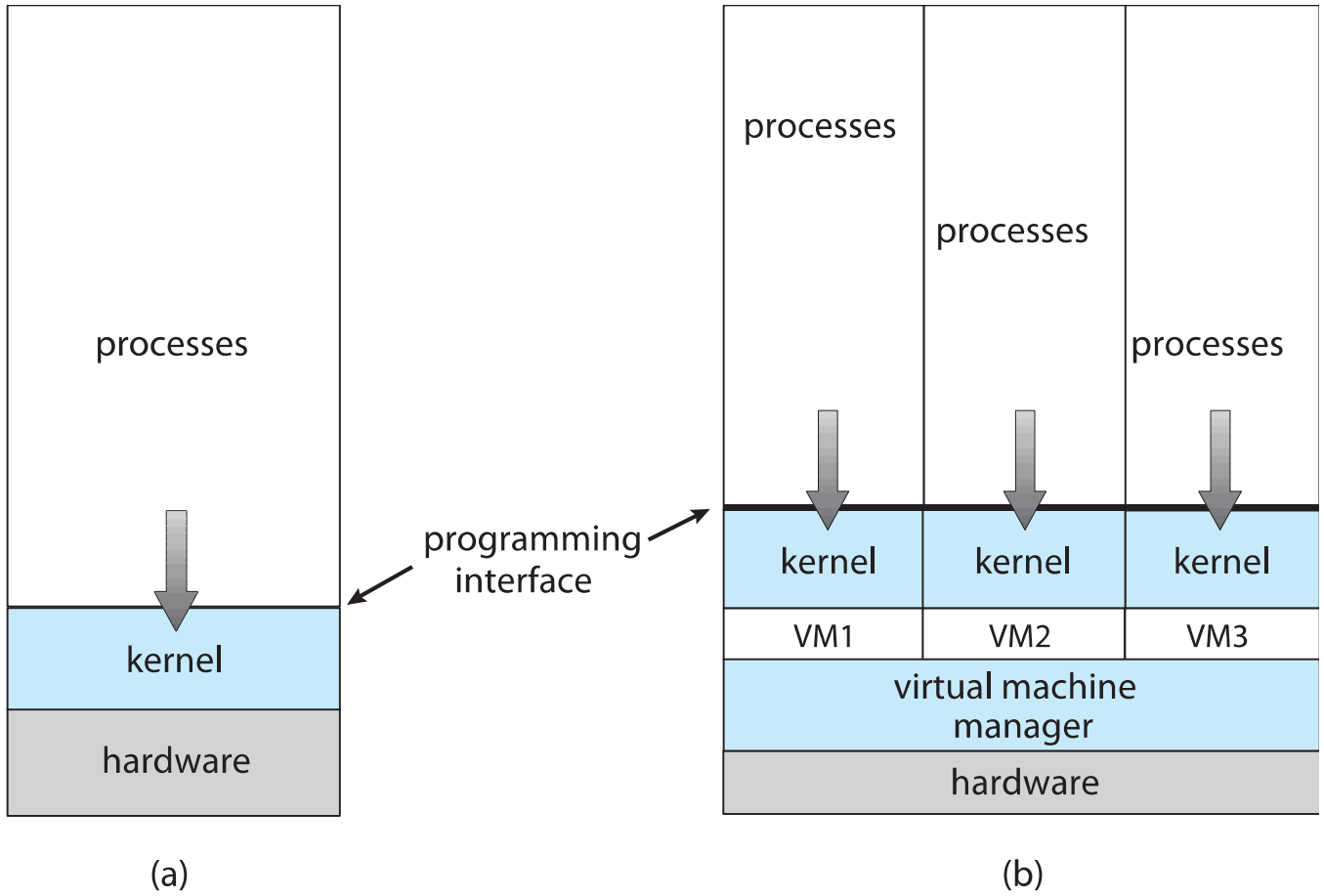
- Real-time embedded systems most prevalent form of computers
  - Vary considerable, special purpose, limited purpose OS, **real-time OS**
  - Use expanding
- Many other special computing environments as well
  - Some have OSes, some perform tasks without an OS
- Real-time OS has well-defined fixed time constraints
  - Processing ***must*** be done within constraint
  - Correct operation only if constraints met



Computing Environments - Virtualization

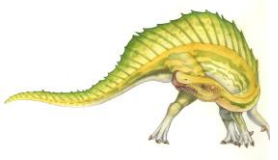
- **Virtualization** – OS natively compiled for CPU, running **guest** OSes also natively compiled
  - Consider VMware running WinXP guests, each running applications, all on native WinXP **host** OS
  - **VMM** (virtual machine Manager) provides virtualization services
- Use cases involve laptops and desktops running multiple OSes for exploration or compatibility
  - Apple laptop running Mac OS X host, Windows as a guest
  - Developing apps for multiple OSes without having multiple systems
  - Executing and managing compute environments within data centers
- VMM can run natively, in which case they are also the host

Computing Environments - Virtualization



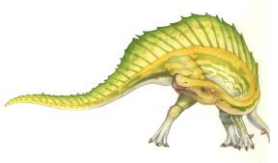
Computing Environments - Virtualization

- Emulation requires software bridge
- With virtualization harware can be directly accessed whereas in Emulator the GOS does not run on the physical hardware
- Emulator requires interpreter to translate the source code to host's readable format, to further process it.
- Emulators are slower than VMs
- Emulators do not rely on CPU while VMs make use of CPU
- VM solution is costlier and more complex than Emulation technoque
- VM provides more throughput, minimal overhead with a better backup and recovery solution



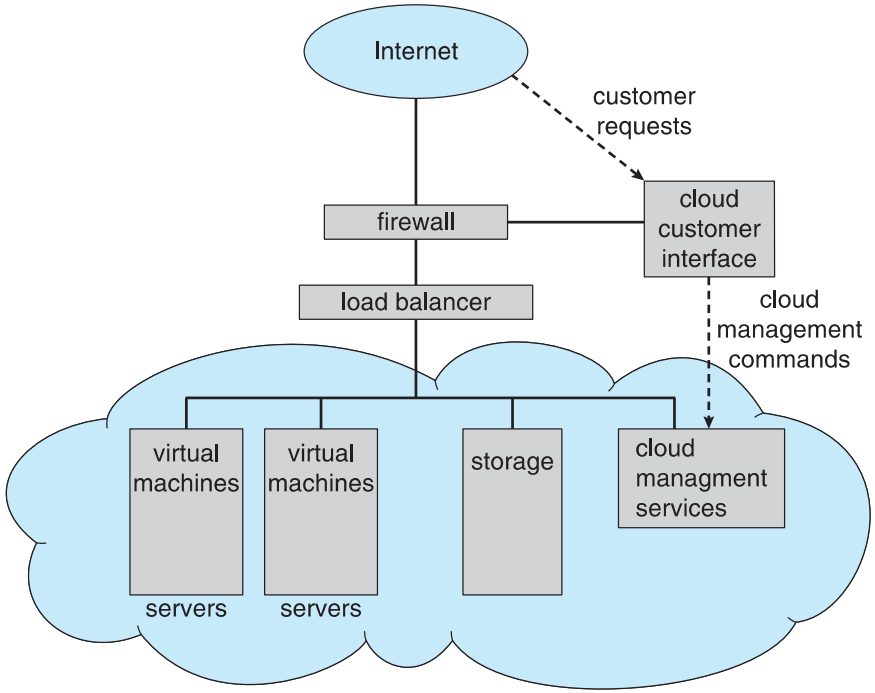
Computing Environments – Cloud Computing

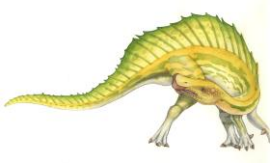
- Delivers computing, storage, even apps as a service across a network
- Logical extension of virtualization because it uses virtualization as the base for it functionality.
  - Amazon **EC2** has thousands of servers, millions of virtual machines, petabytes of storage available across the Internet, pay based on usage
- Many types
  - **Public cloud** – available via Internet to anyone willing to pay
  - **Private cloud** – run by a company for the company's own use
  - **Hybrid cloud** – includes both public and private cloud components
  - Software as a Service (**SaaS**) – one or more applications available via the Internet (i.e., word processor)
  - Platform as a Service (**PaaS**) – software stack ready for application use via the Internet (i.e., a database server)
  - Infrastructure as a Service (**IaaS**) – servers or storage available over Internet (i.e., storage available for backup use)



Computing Environments – Cloud Computing

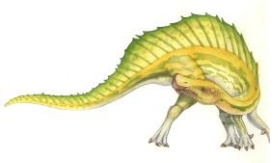
- Cloud computing environments composed of traditional OSes, plus VMMs, plus cloud management tools
  - Internet connectivity requires security like firewalls
  - Load balancers spread traffic across multiple applications





Open-Source Operating Systems

- Operating systems made available in source-code format rather than just binary **closed-source**
- Counter to the **copy protection** and **Digital Rights Management (DRM)** movement
- Started by **Free Software Foundation (FSF)**, which has "copyleft" **GNU Public License (GPL)**
- Examples include **GNU/Linux** and **BSD UNIX** (including core of **Mac OS X**), and many more
- Can use VMM like VMware Player (Free on Windows), Virtualbox (open source and free on many platforms - http://www.virtualbox.com)
  - Use to run guest operating systems for exploration



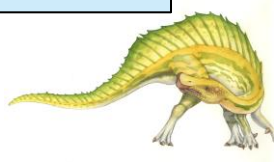
The Study of Operating Systems

There has never been a more interesting time to study operating systems, and it has never been easier. The open-source movement has overtaken operating systems, causing many of them to be made available in both source and binary (executable) format. The list of operating systems available in both formats includes Linux, BUSD UNIX, Solaris, and part of macOS. The availability of source code allows us to study operating systems from the inside out. Questions that we could once answer only by looking at documentation or the behavior of an operating system we can now answer by examining the code itself.

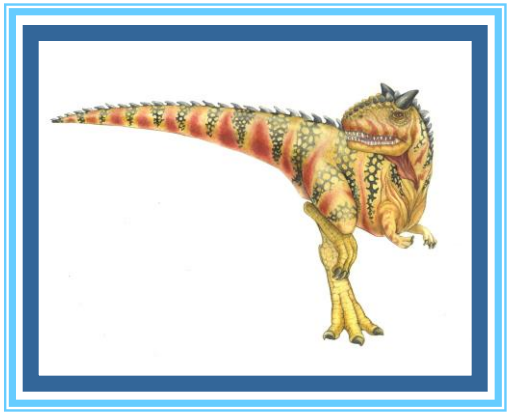
Operating systems that are no longer commercially viable have been open-sourced as well, enabling us to study how systems operated in a time of fewer CPU, memory, and storage resources. An extensive but incomplete list of open-source operating-system projects is available from https://curlie.org/Computers/Software/Operating_Systems/Open_Source/

In addition, the rise of virtualization as a mainstream (and frequently free) computer function makes it possible to run many operating systems on top of one core system. For example, VMware (http://www.vmware.com) providesa free "player" for Windows on which hundreds of free "virtual appliances" can run. Virtualbox (http://www.virtualbox.com) provides a free, open-source virtual machine manager on many operating systems. Using such tools, students can try out hundreds of operating systems without dedicated hardware.

The advent of open-source operating systems has also made it easier to make the move from student to operating-system developer. With some knowledge, some effort, and an Internet connection, a student can even create a new operating-system distribution. Just a few years ago, it was difficult or impossible to get access to source code. Now, such access is limited only by how much interest, time, and disk space a student has.

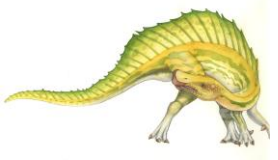


Chapter 2a: Operating-System Services



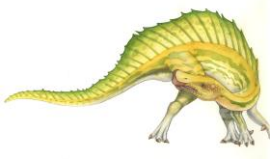
Outline

- Operating System Services
- User and Operating System-Interface
- System Calls
- System Services
- Linkers and Loaders
- Why Applications are Operating System Specific

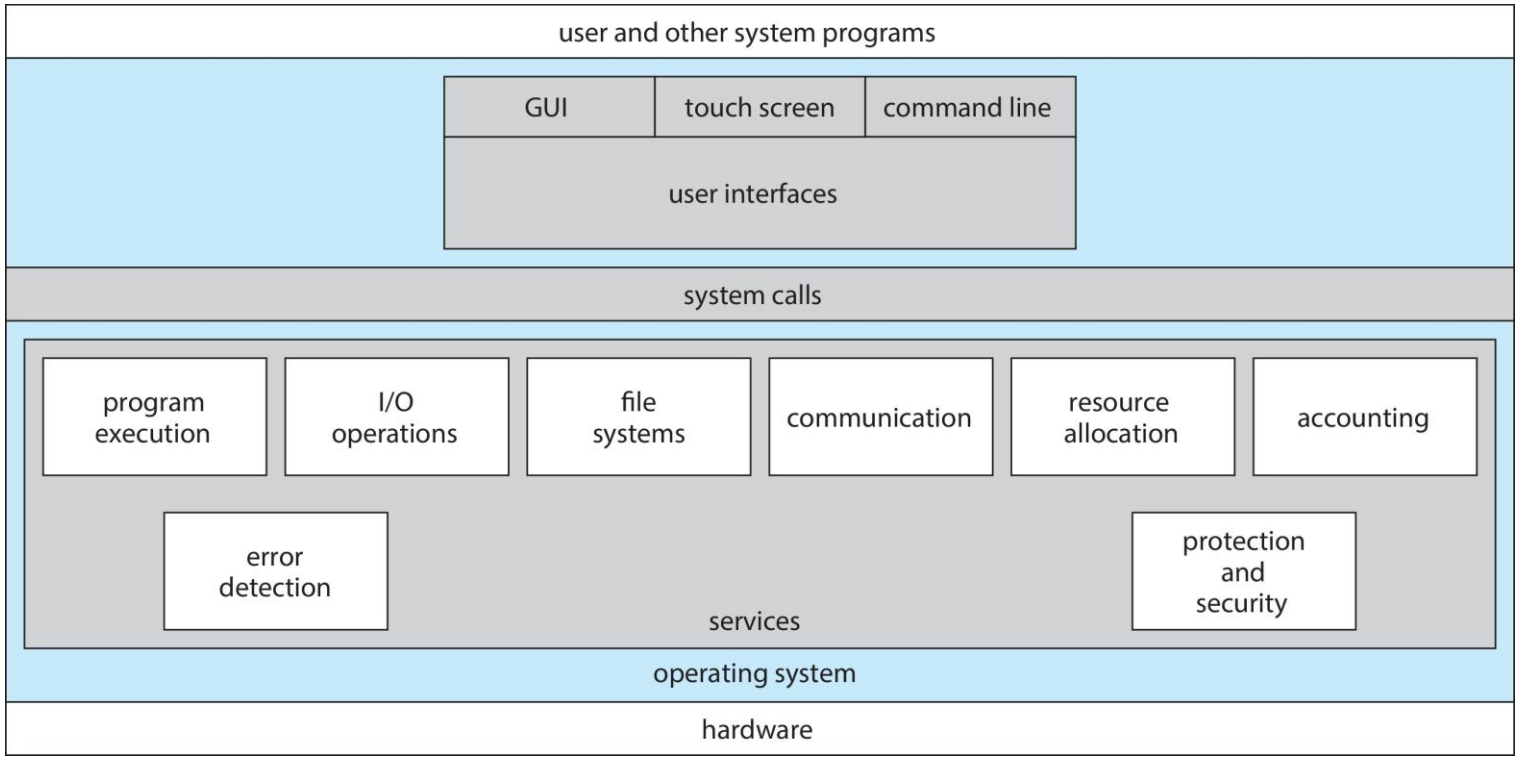


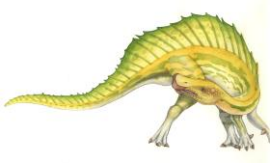
Objectives

- Identify services provided by an operating system
- Illustrate how system calls are used to provide operating system services



A View of Operating System Services





Operating System Services

- Operating systems provide an environment for execution of programs and services to programs and users.
- One set of operating-system services provides functions that are helpful to the user:
  - **User interfaces** – Means by which users can issue commands. Almost all operating systems have a user interface (**UI**). Depending on the OS UI
    - Varies between **Command-Line interface** (**CLI**), **Graphics User Interface** (**GUI**), **Touch screen, Batch command systems**
  - **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)
  - **I/O operations** - A running program may require I/O. The OS is responsible for transferring data to and from I/O devices, including keyboards, terminals, printers, and files.

.

Operating System Services (Cont.)

- One set of operating-system services provides functions that are helpful to the user (Cont.):
  - **File-system manipulation** - 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)

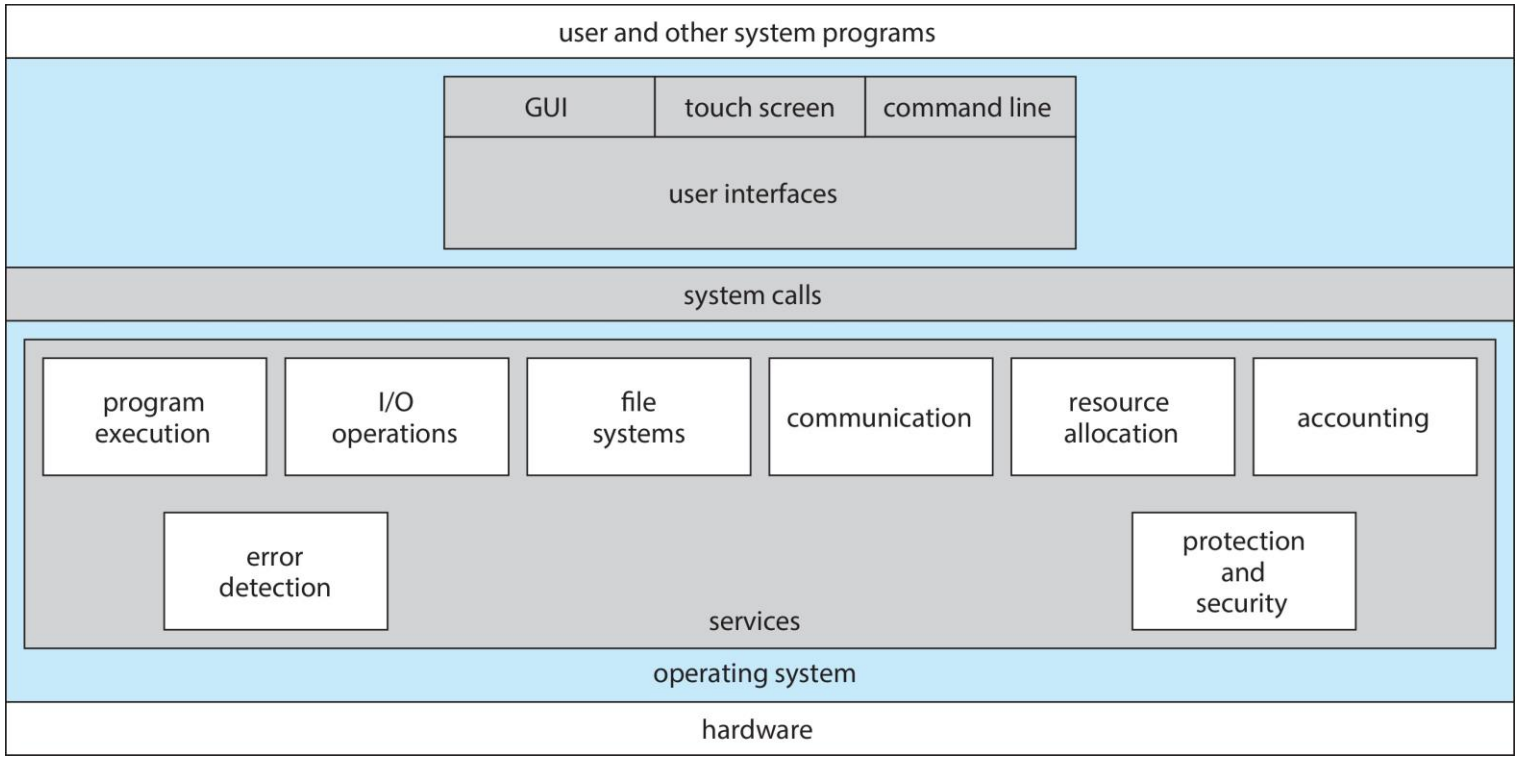
Operating System Services (Cont.)

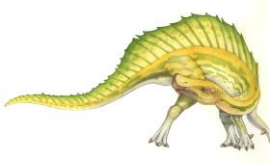
- One set of operating-system services provides functions that are helpful to the user (Cont.):
  - **Error detection** – OS needs to be constantly aware of possible errors (H/W and S/W errors)
    - May occur in the CPU and memory hardware, in I/O devices, in user program
    - For each type of error, 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 function exists 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
    - Many types of resources - CPU cycles, main memory, file storage, I/O devices.
  - **Logging/Accounting -** To keep track of which users use how much and what kinds of computer resources which may be used for billing or statistical record keeping
  - **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
    - **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

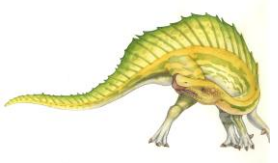
A View of Operating System Services





User Operating System Interface

- CLI -- command line interpreter
  - allows direct command entry
- GUI – graphical user interface
- Touchscreen Interfaces
- Batch



CLI

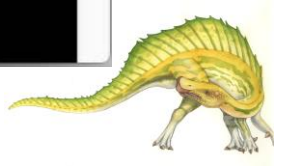
- Sometimes implemented in kernel, sometimes by systems program
- Command Interpreters are used to give commands to the OS.
- There are multiple command interpreters known as shells. *Bourne shell, C shell, Bourne-Again shell, Korn shell*
- Primarily fetches a command from user and executes it
- The commands can be implemented in two general ways
  - The command interpreter itself contains the code to execute the command.
  - The code to implement the command is in a function in a separate file. Thus by adding new functions new commands can be added easily to the interpreter.

Bourne Shell Command Interpreter

1. root@r6181-d5-us01:~ (ssh)

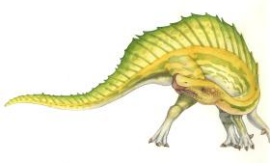
root@r6181-d5-u... ⌘1 | ssh ⌘2 | root@r6181-d5-us01... ⌘3

```
Last login: Thu Jul 14 08:47:01 on ttys002
iMacPro:~ pbg$ ssh root@r6181-d5-us01
root@r6181-d5-us01's password:
Last login: Thu Jul 14 06:01:11 2016 from 172.16.16.162
[root@r6181-d5-us01 ~]# uptime
 06:57:48 up 16 days, 10:52,  3 users,  load average: 129.52, 80.33, 56.55
[root@r6181-d5-us01 ~]# df -kh
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/vg_ks-lv_root
                       50G   19G   28G  41% /
tmpfs                 127G  520K  127G   1% /dev/shm
/dev/sda1             477M   71M  381M  16% /boot
/dev/dssd0000         1.0T  480G  545G  47% /dssd_xfs
tcp://192.168.150.1:3334/orangefs
                       12T  5.7T  6.4T  47% /mnt/orangefs
/dev/gpfs-test         23T  1.1T   22T   5% /mnt/gpfs
[root@r6181-d5-us01 ~]#
[root@r6181-d5-us01 ~]# ps aux | sort -nrk 3,3 | head -n 5
root      97653 11.2  6.6 42665344 17520636 ?   S<Ll Jul13 166:23 /usr/lpp/mmfs/bin/mmfsd
root      69849  6.6  0.0      0     0 ?        S    Jul12 181:54 [vpthread-1-1]
root      69850  6.4  0.0      0     0 ?        S    Jul12 177:42 [vpthread-1-2]
root       3829  3.0  0.0      0     0 ?        S    Jun27 730:04 [rp_thread 7:0]
root       3826  3.0  0.0      0     0 ?        S    Jun27 728:08 [rp_thread 6:0]
[root@r6181-d5-us01 ~]# ls -l /usr/lpp/mmfs/bin/mmfsd
-r-x------ 1 root root 20667161 Jun  3  2015 /usr/lpp/mmfs/bin/mmfsd
[root@r6181-d5-us01 ~]#
```



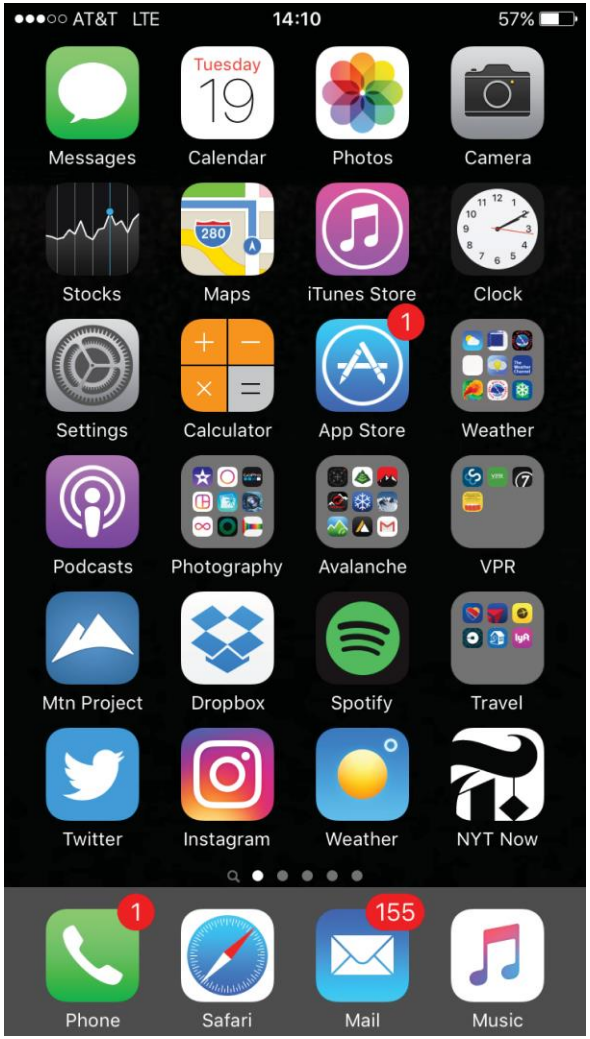
GUI

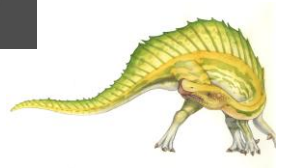
- User-friendly **desktop** metaphor interface
  - Usually mouse, keyboard, and monitor
  - **Icons** represent files, programs, actions, etc.
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**)
  - Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with CLI “command” shell
  - Apple Mac OS X is “Aqua” GUI interface with UNIX kernel underneath and shells available
  - Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)



Touchscreen Interfaces

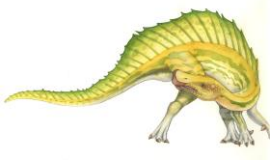
- Touchscreen devices require new interfaces
  - Mouse not possible or not desired
  - Actions and selection based on gestures
  - Virtual keyboard for text entry
- Voice commands





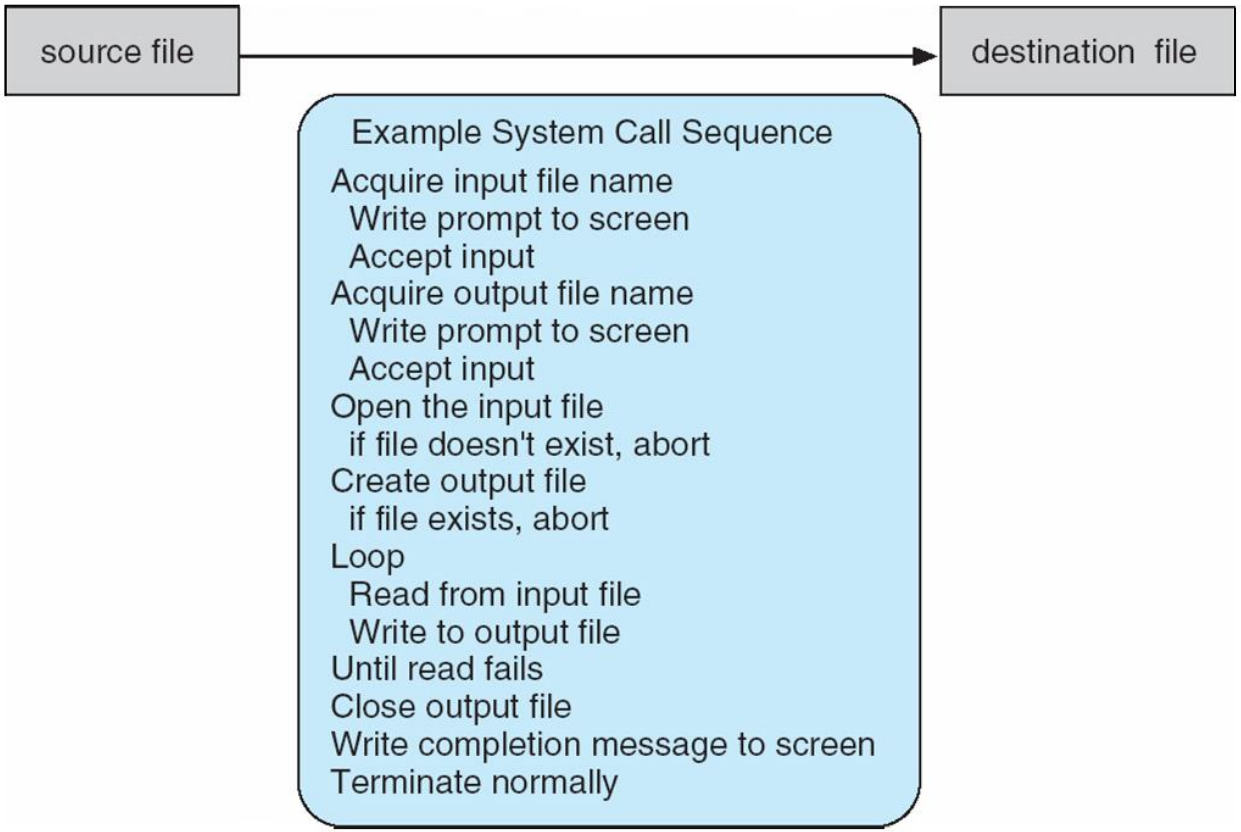
System Calls

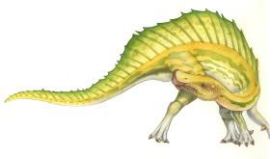
- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++), although some are written in assembly for optimal performance.
- Mostly accessed by programs via a high-level **Application Programming Interface** (**API**) rather than direct system call 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)



Example of System Calls

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





Example of Standard API

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

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

return value — function name — parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

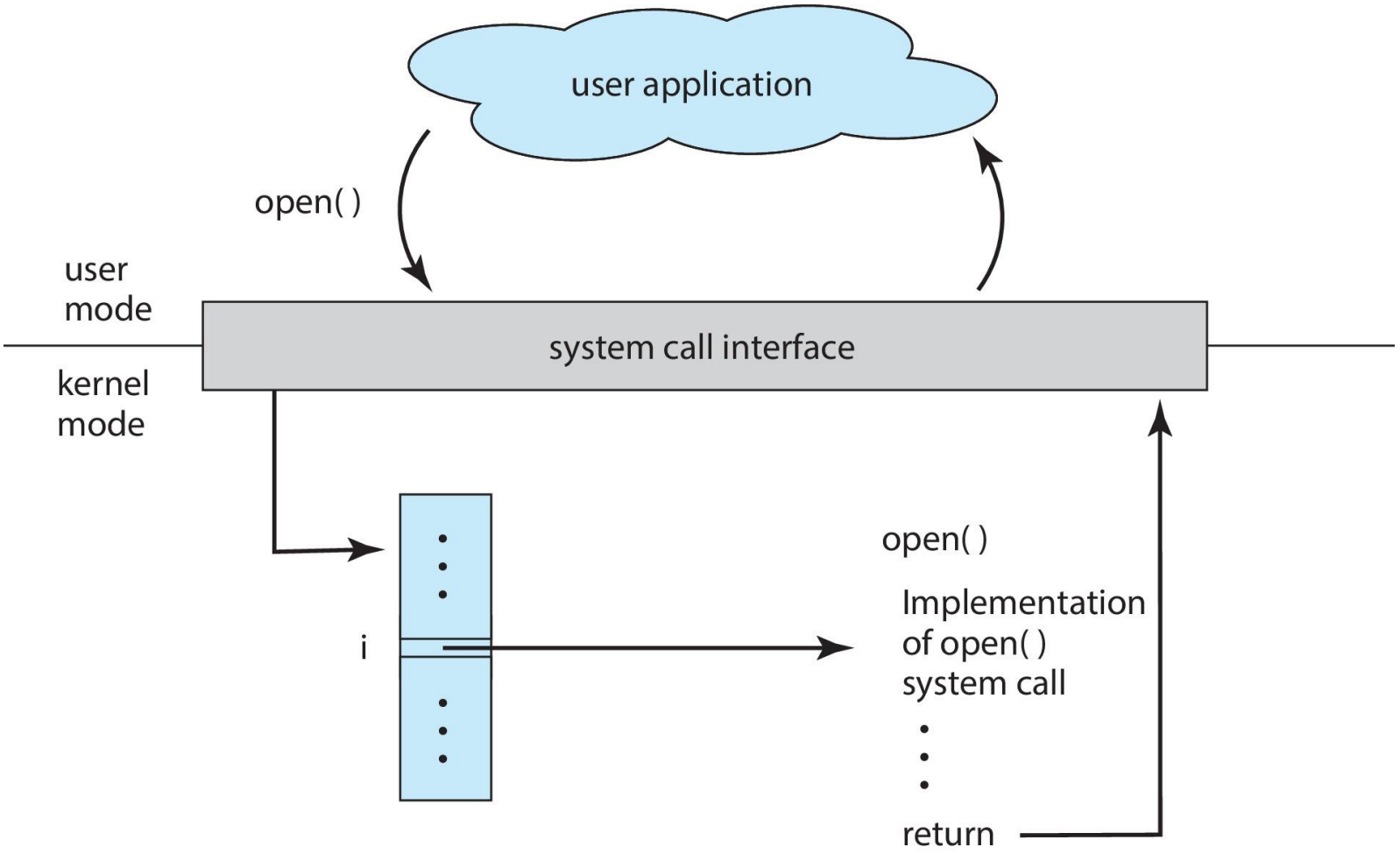
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer into which the data will be read
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

System Call Implementation

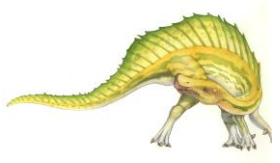
- Typically, a number is associated with each system call
  - **System-call interface** maintains a table indexed according to these numbers
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values
- The caller need not know anything 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 to Open a File

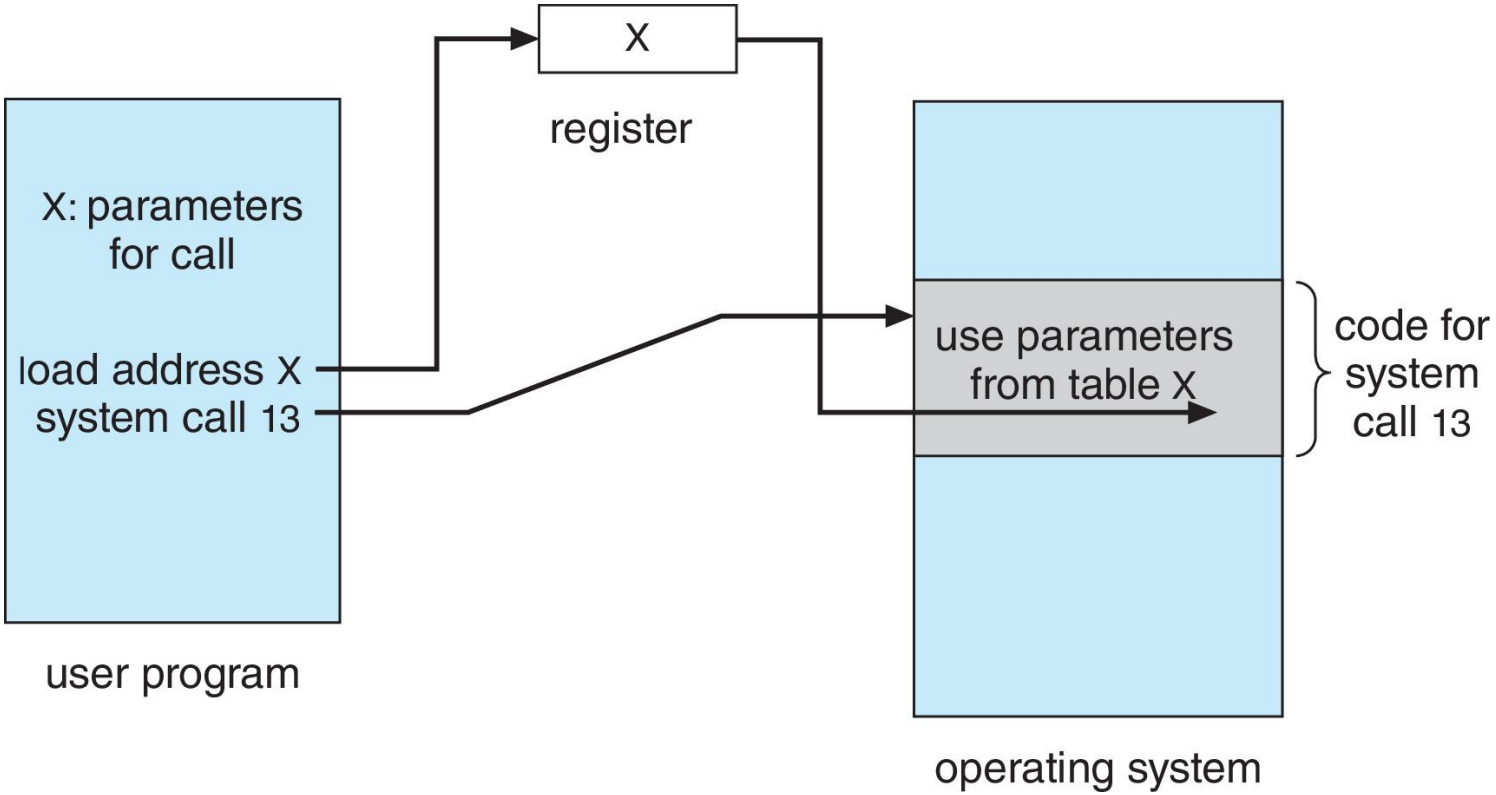


System Call Parameter Passing

- 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
  - Pass the parameters in registers
    - In some cases, there 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
    - 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

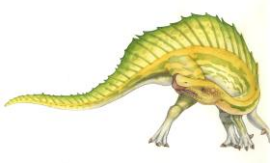


Parameter Passing via Table



Types of System Calls

- 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
  - **Debugger** for determining **bugs, single step** execution
  - **Locks** for managing access to shared data between processes

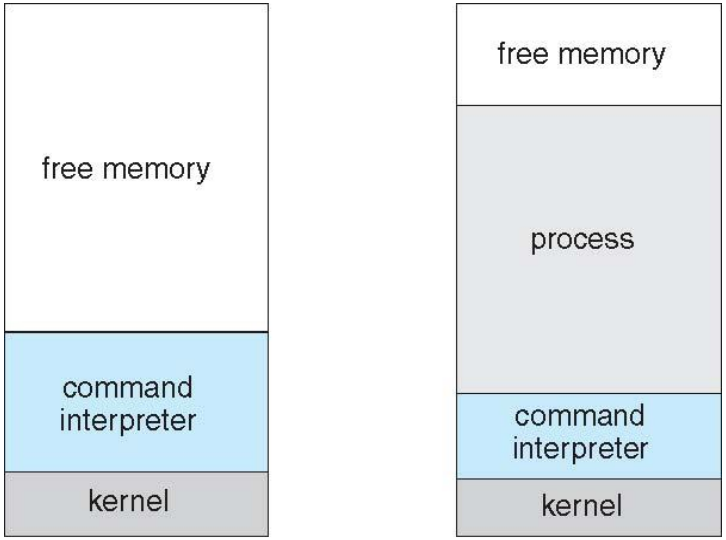


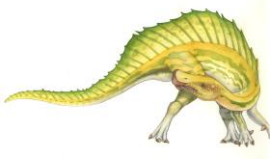
Example: DOS

- Process Control:

In DOS, the command interpreter loaded first. Then loads the process and transfers control to it. The interpreter does not resume until the process has completed, as shown in Figure

At system startup running a program

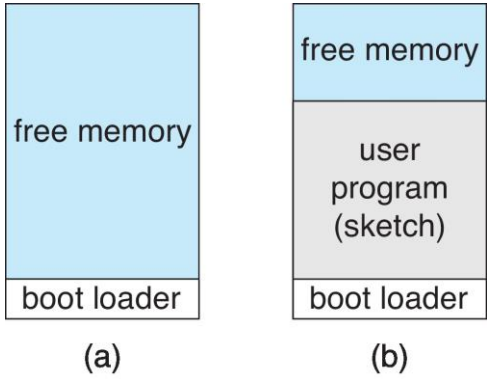




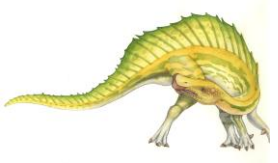
Example: Arduino

The Arduino is a simple hardware platform consisting of a microcontroller along with input sensors that respond to a variety of events, such as changes to light, temperature, and barometric pressure, etc.

- Single-tasking
- No operating system
- Programs (sketch) loaded via USB into flash memory
- Single memory space
- Boot loader loads program
- Program exit -> shell reloaded

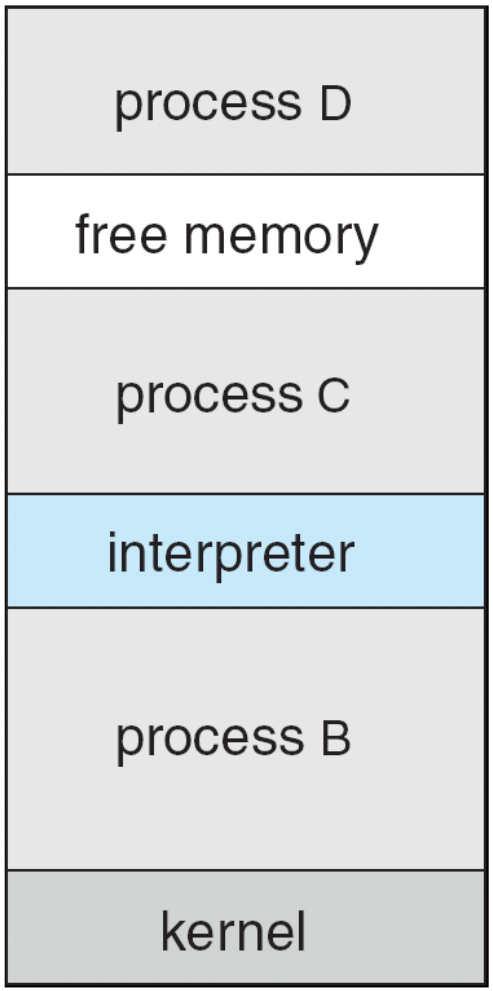


At system startup running a program



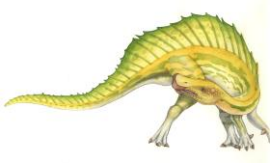
Example: FreeBSD

- Unix variant
- Multitasking
- User login -> invoke user's choice of shell
- Shell executes fork() system call to create process
  - Executes exec() to load program into process
  - Shell/parent (CI) waits for process to terminate or continues with user commands
- Process exits with:
  - code = 0 – no error
  - code > 0 – error code



Types of System Calls (Cont.)

- 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

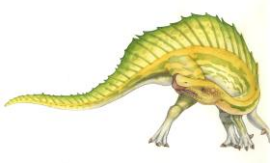


Types of System Calls (Cont.)

- 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
  - attach and detach remote devices

Types of System Calls (Cont.)

- Protection
  - control access to resources
  - get and set permissions
  - allow and deny user access



Examples of Windows and Unix System Calls

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

The following illustrates various equivalent system calls for Windows and UNIX operating systems.

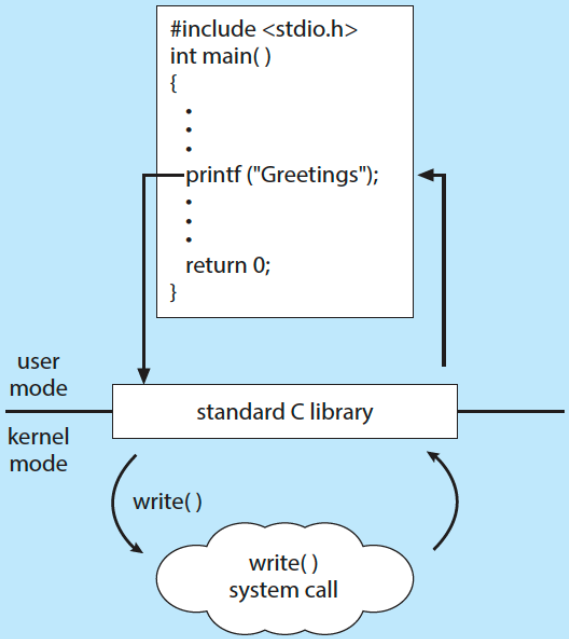
	Windows	Unix
Process control	`CreateProcess()`	`fork()`
	`ExitProcess()`	`exit()`
	`WaitForSingleObject()`	`wait()`
File management	`CreateFile()`	`open()`
	`ReadFile()`	`read()`
	`WriteFile()`	`write()`
	`CloseHandle()`	`close()`
Device management	`SetConsoleMode()`	`ioctl()`
	`ReadConsole()`	`read()`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

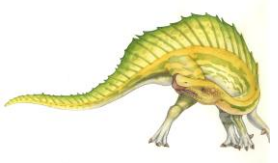
Standard C Library Example

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

THE STANDARD C LIBRARY

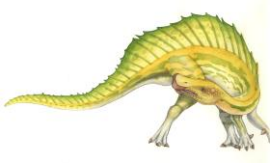
The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, let's assume a C program invokes the `printf()` statement. The C library intercepts this call and invokes the necessary system call (or calls) in the operating system—in this instance, the `write()` system call. The C library takes the value returned by `write()` and passes it back to the user program:





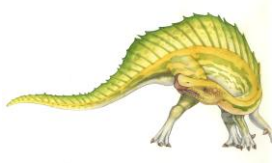
System Services

- System programs provide a convenient environment for program development and execution. They can be divided into:
  - File manipulation
  - Status information sometimes stored in a file
  - Programming language support
  - Program loading and execution
  - Communications
  - Background services
  - Application programs
- Most users' view of the operation system is defined by system programs, not the actual system calls



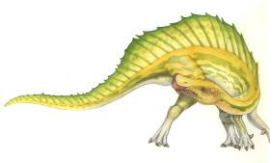
System Services (Cont.)

- Provide a convenient environment for program development and execution
  - Some of them are simply user interfaces to system calls; others are considerably more complex
- **File management** - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- **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



System Services (Cont.)

- **File modification**
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text
- **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided
- **Program loading and execution**- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- **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

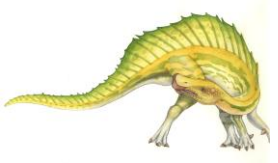


System Services (Cont.)

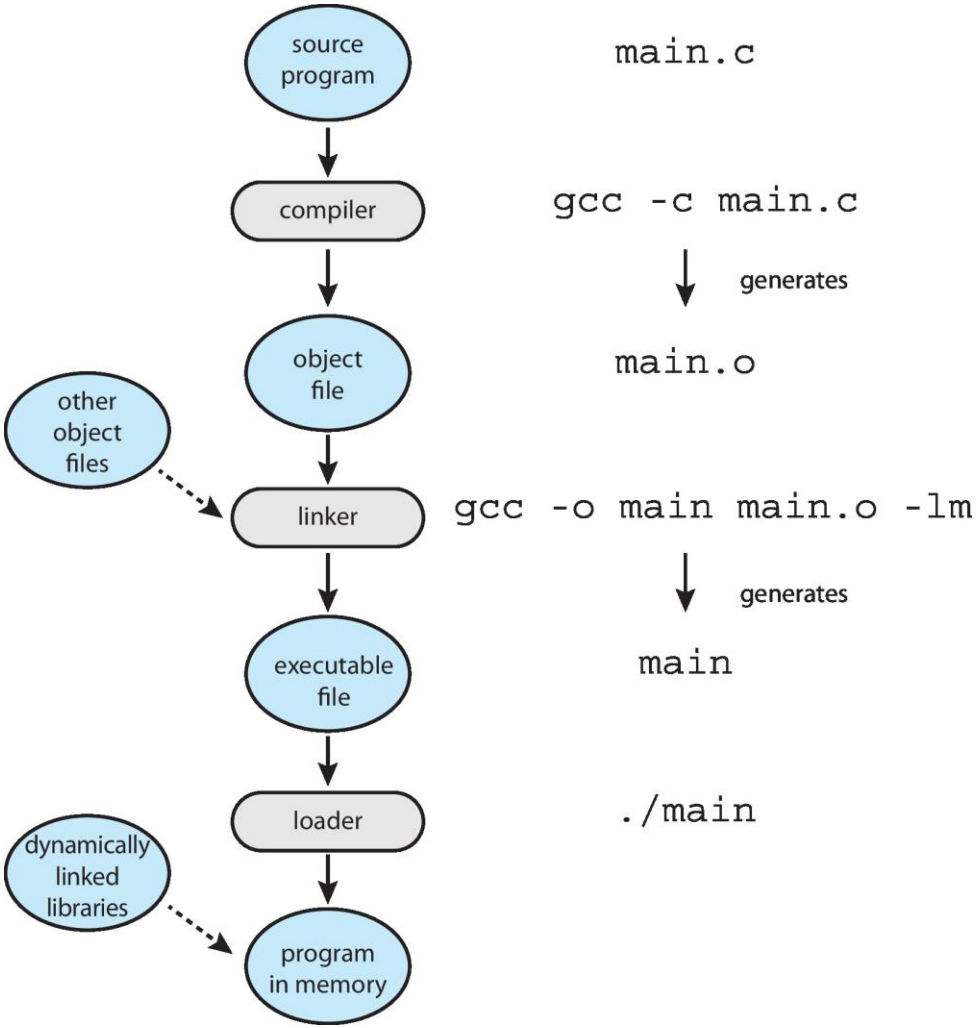
- **Background Services**
  - Launch at boot time
    - Some for system startup, then terminate
    - Some from system boot to shutdown
  - Provide facilities like disk checking, process scheduling, error logging, printing
  - Run in user context not kernel context
  - Known as **services**, **subsystems**, **daemons**
- **Application programs**
  - Don't pertain to system
  - Run by users
  - Not typically considered part of OS
  - Launched by command line, mouse click, finger poke

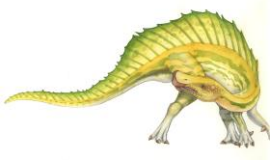
Linkers and Loaders

- Source code compiled into object files designed to be loaded into any physical memory location – **relocatable object file**
- **Linker** combines these into single binary **executable** file
  - Also brings in libraries
- Program resides on secondary storage as binary executable
- Must be brought into memory by **loader** to be executed
  - **Relocation** assigns final addresses to program parts and adjusts code and data in program to match those addresses
- Modern general-purpose systems don't link libraries into executables
  - Rather, **dynamically linked libraries** (in Windows, **DLLs**) are loaded as needed, shared by all that use the same version of that same library (loaded once)
- Object, executable files have standard formats, so operating system knows how to load and start them



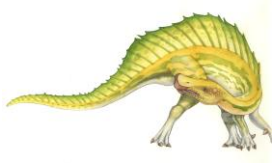
The Role of the Linker and Loader





Why Applications are Operating System Specific

- Apps compiled on one system usually not executable on other operating systems
- Each operating system provides its own unique system calls
  - Own file formats, etc.
- Apps can be multi-operating system
  - Written in interpreted language like Python, Ruby, and interpreter available on multiple operating systems
  - App written in language that includes a VM containing the running app (like Java)
  - Use standard language (like C), compile separately on each operating system to run on each
- **Application Binary Interface** (**ABI**) is architecture equivalent of API, defines how different components of binary code can interface for a given operating system on a given architecture, CPU, etc.



End of Chapter 2a