CSE2008: Operating Systems

L1 L2 L3 L4 L5 & L6: Introductory Concepts

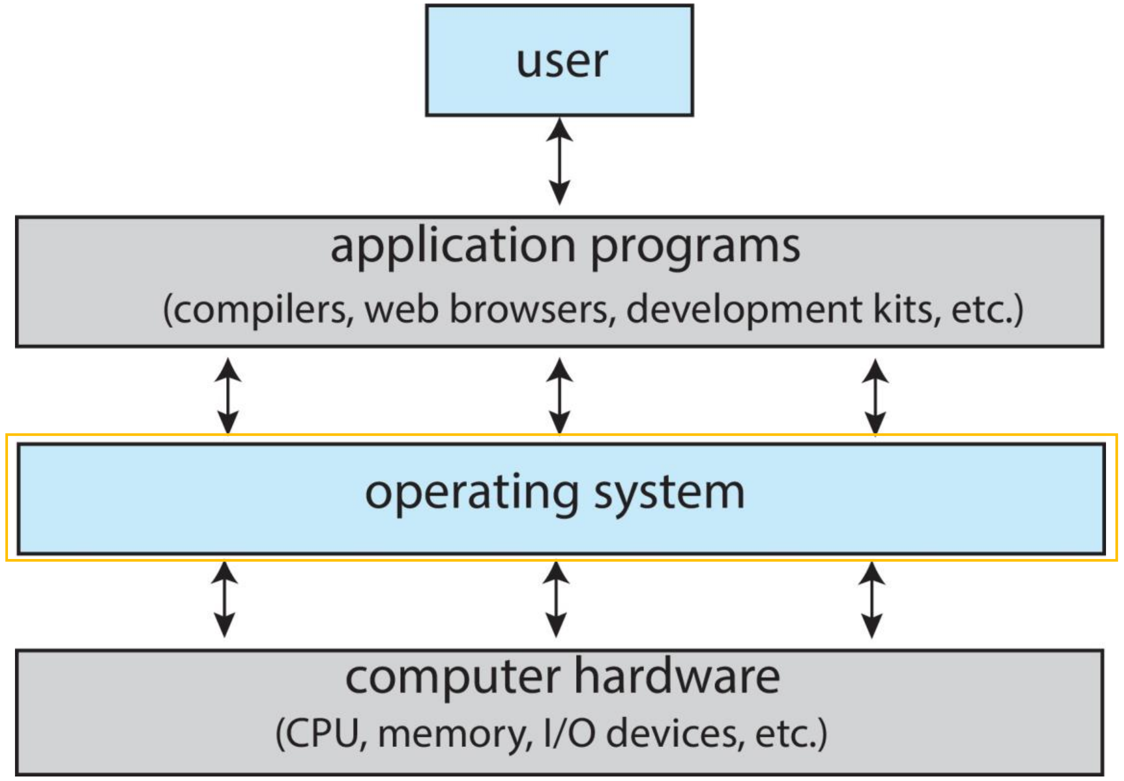
Dr. Subrata Tikadar
SCOPE, VIT-AP University

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 and Open-Source Operating Systems
- Overview of System Calls

What Operating Systems Do?

- Components of a computer system


user
application programs
(compilers, web browsers, development kits, etc.)
operating system
computer hardware
(CPU, memory, I/O devices, etc.)

Role of an OS – two viewpoints

User View

OS is designed mostly for ease of use, with some attention paid to performance and security and none paid to resource utilization

System View

OS acts as a resource allocator

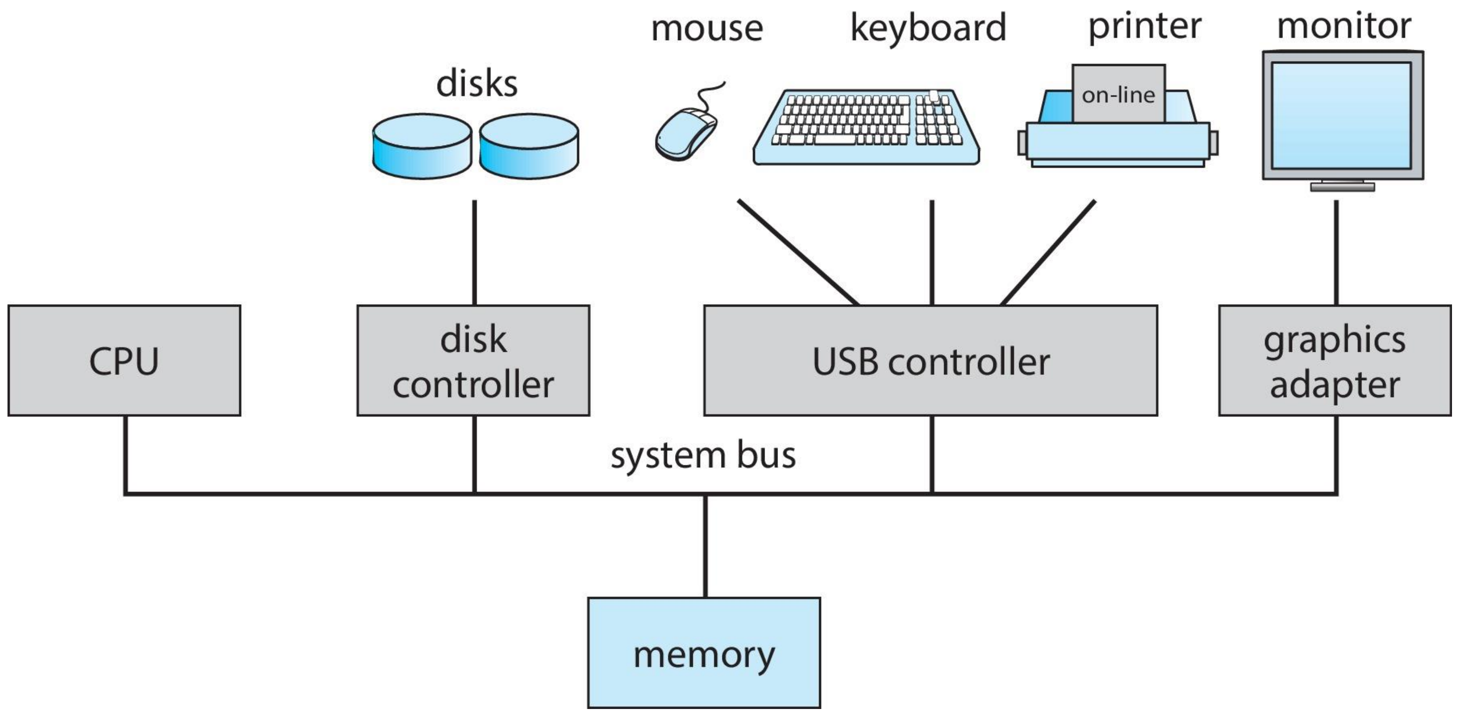
What Operating Systems Do?

- Defining Operating System
  - The operating system is the one program running at all times on the computer—usually called the **kernel**. Along with the kernel, there are two other types of programs: **system programs**, which are associated with the operating system but are not necessarily part of the kernel, and **application programs**, which include all programs not associated with the operation of the system

Note: Mobile operating systems often include not only a core kernel but also middleware—a set of software frameworks that provide additional services to application developers. For example, each of the two most prominent mobile operating systems—Apple's iOS and Google's Android—features core kernel along with middleware that supports databases, multimedia, graphics, etc.

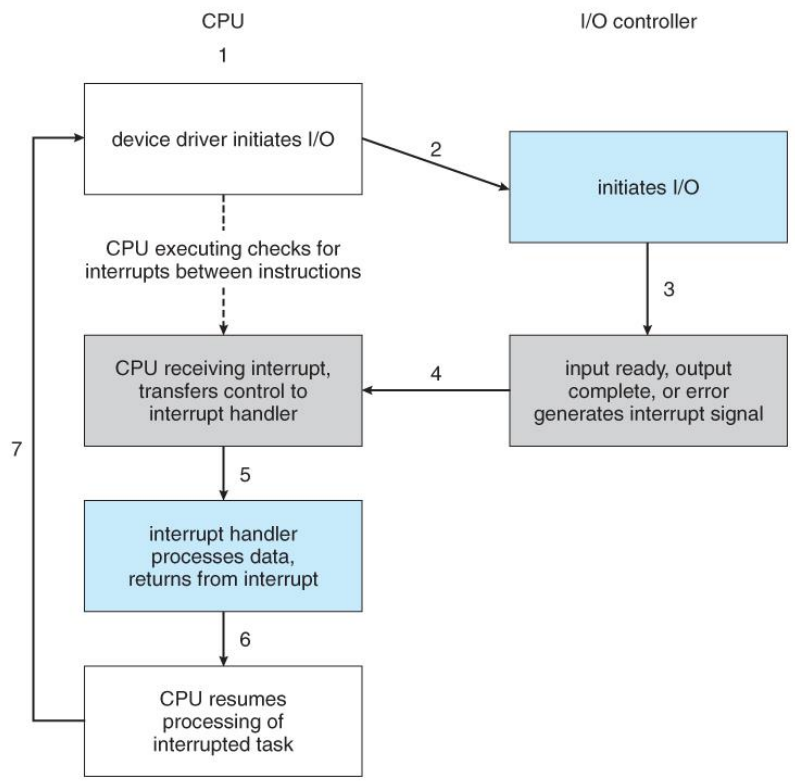
Computer System Organization

- Computer System Organization

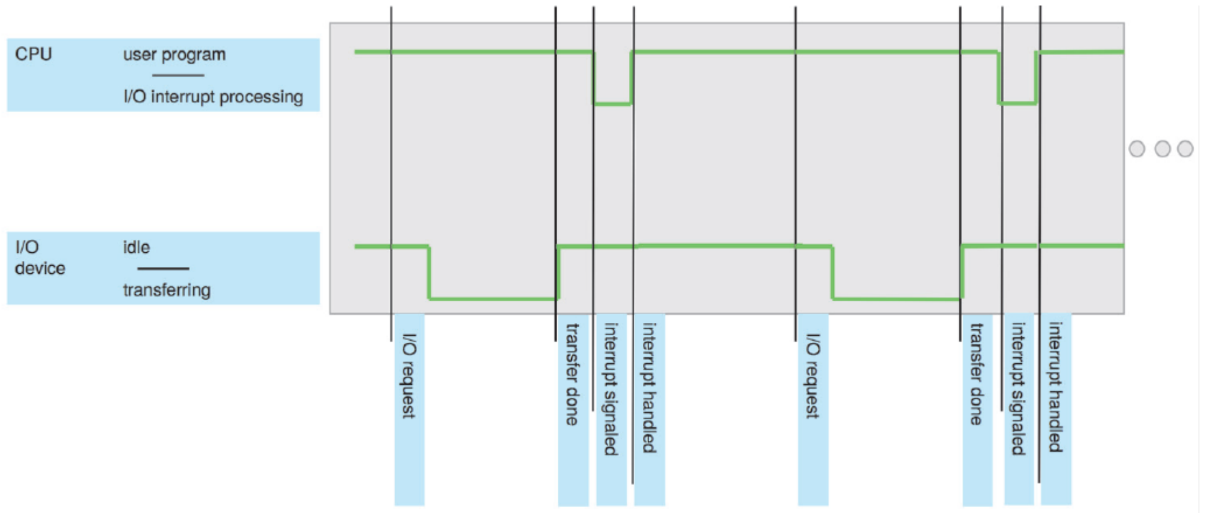


Computer System Organization

- Interrupt-driven I/O cycle

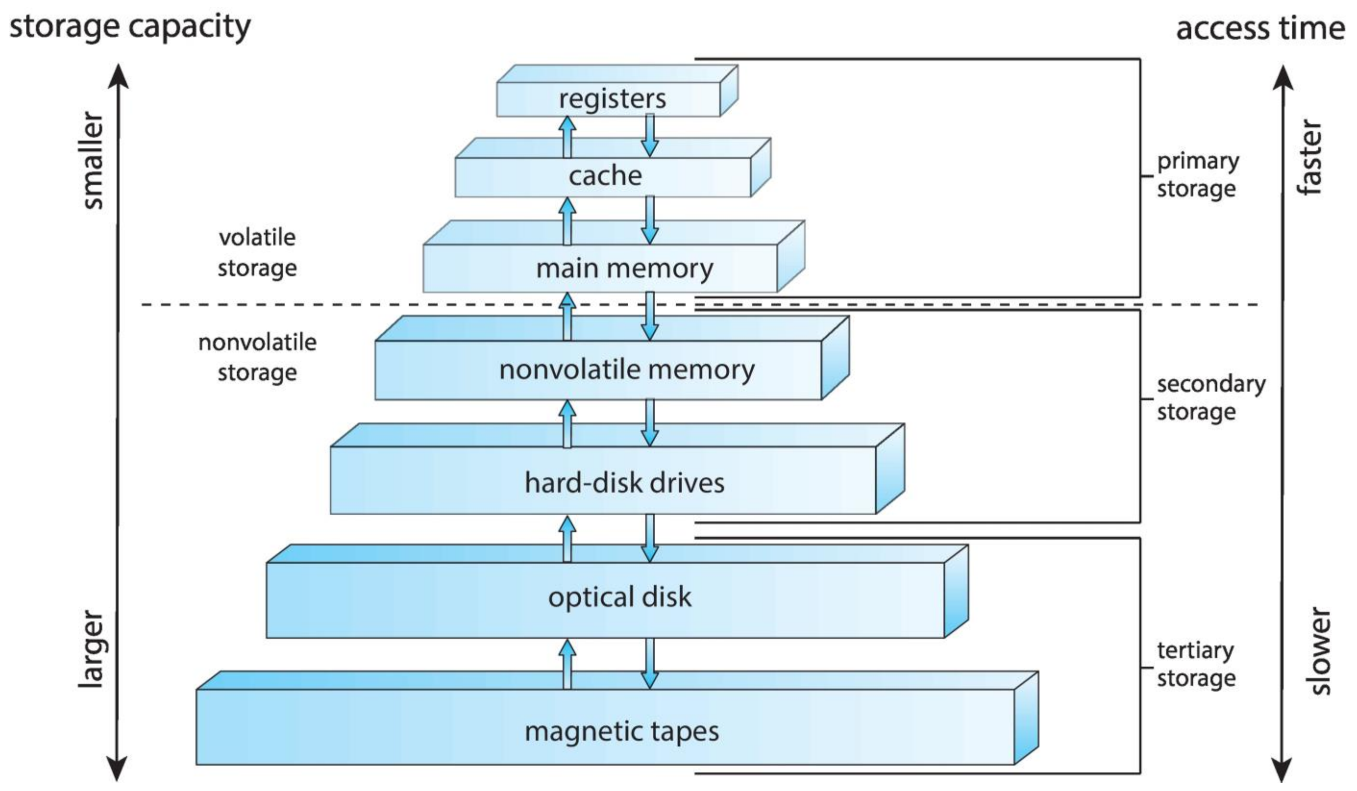


CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
CPU receiving interrupt, transfers control to interrupt handler
4
input ready, output complete, or error generates interrupt signal
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task
7



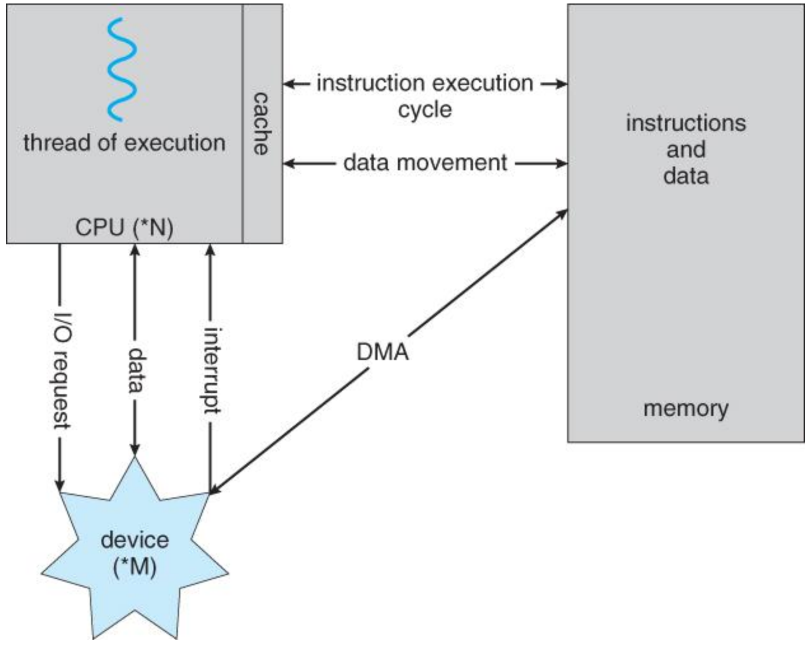
Computer System Organization

- Storage-Device Hierarchy



Computer System Organization

- I/O Structure - how a modern computer system works

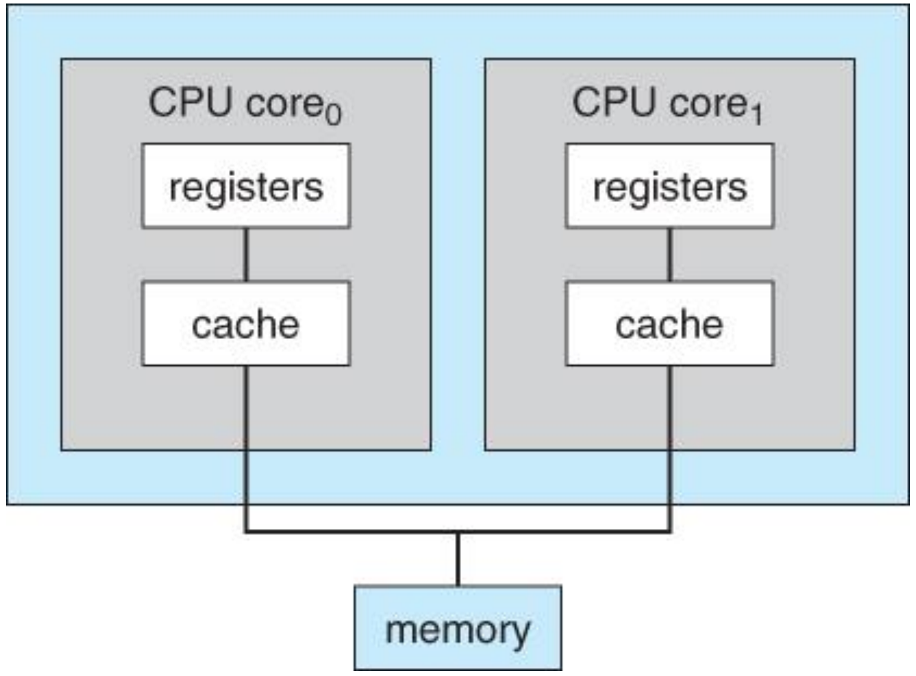


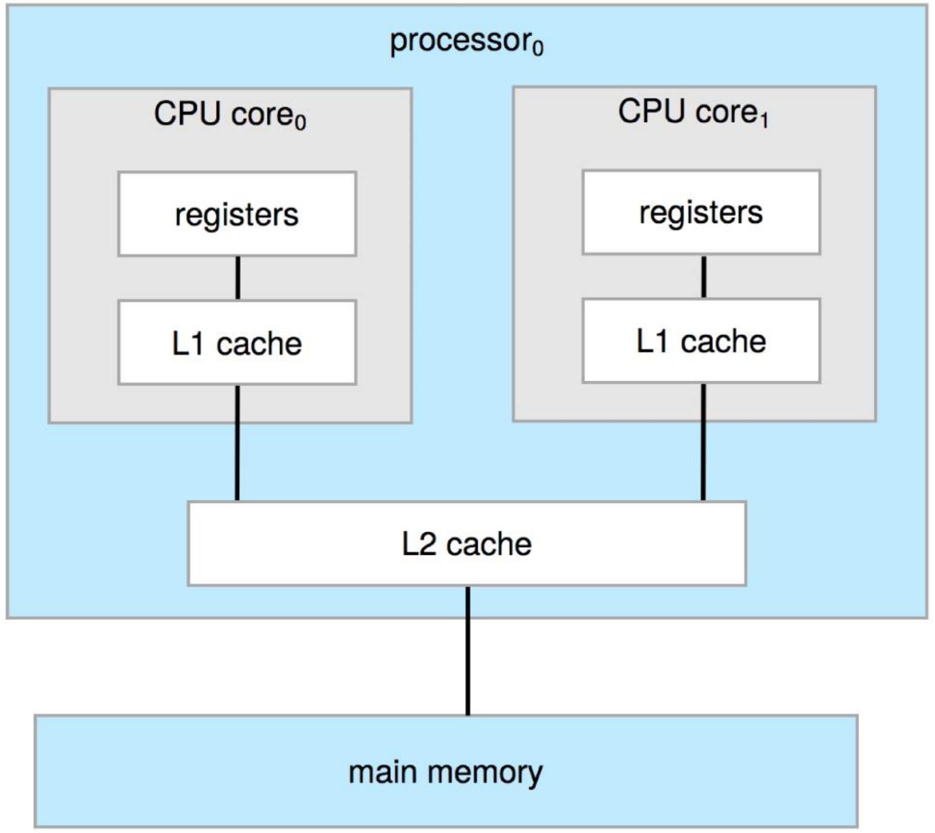
Computer-System Architecture

- Single-Processor Systems
- Multiprocessor Systems

Computer-System Architecture

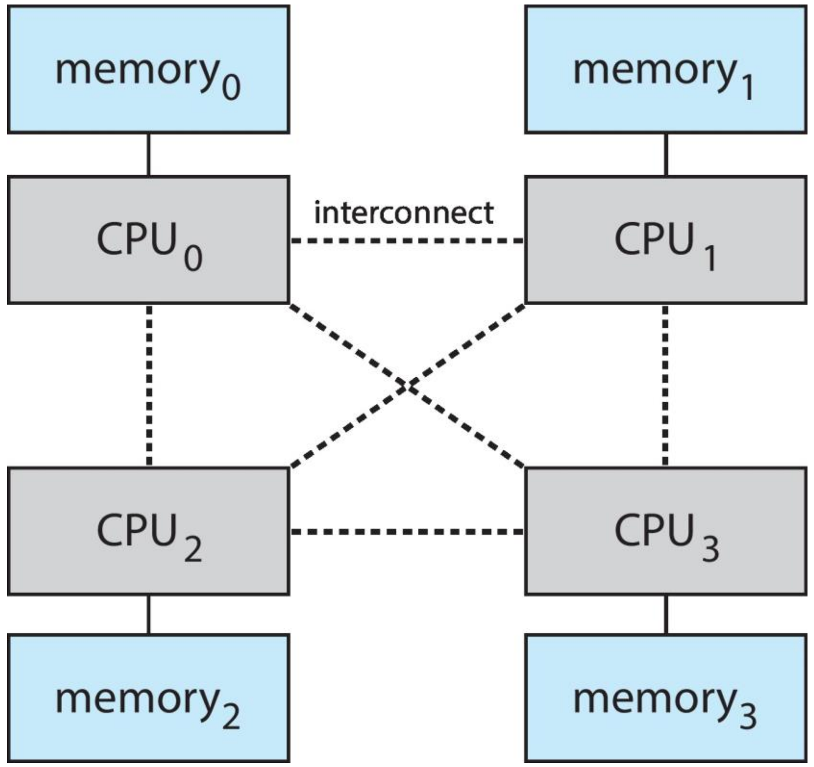
- Multiprocessor Systems
  - Symmetric multiprocessing architecture





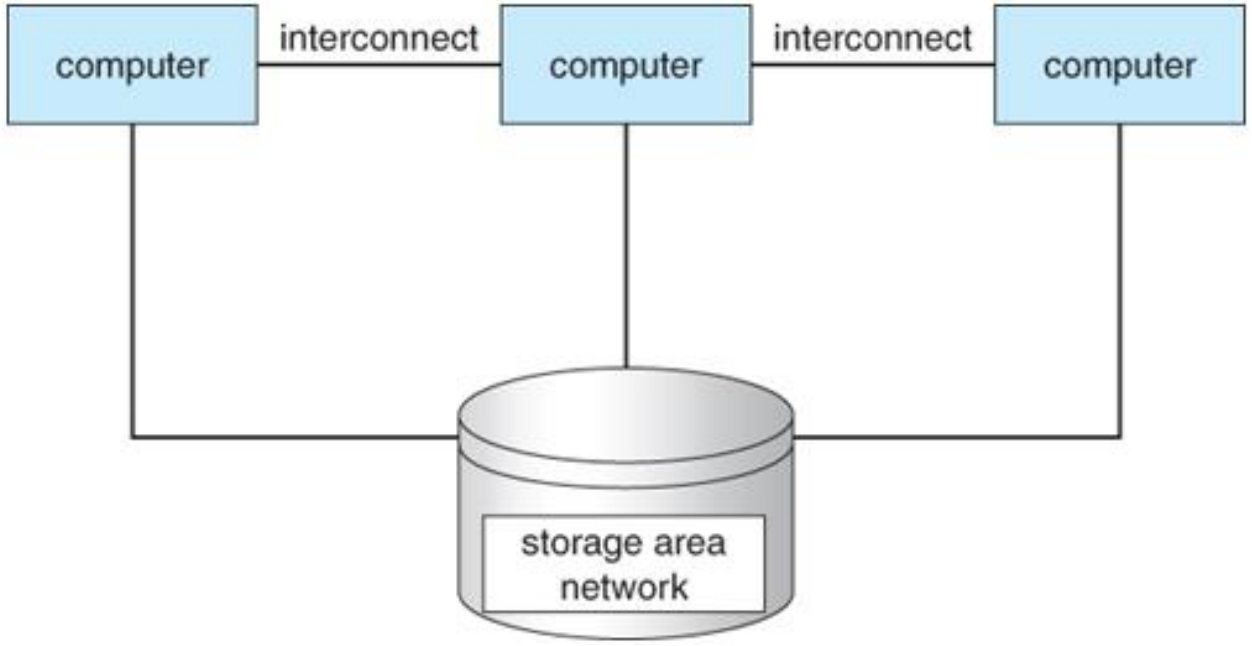
Computer-System Architecture

- Multiprocessor Systems
  - NUMA multiprocessing architecture



Computer-System Architecture

- Multiprocessor Systems
  - Clustered Systems



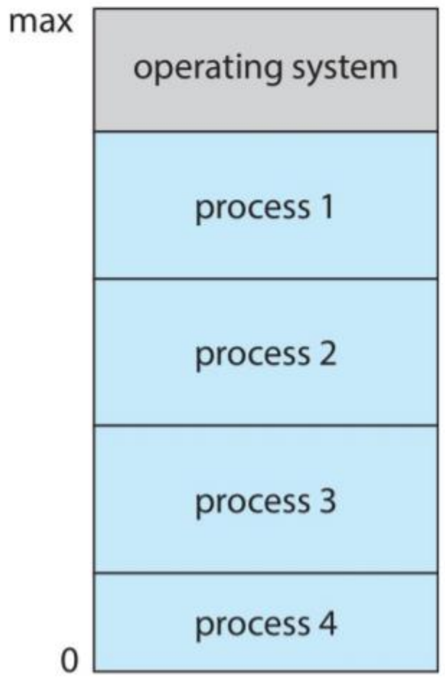
Operating-System Operations

- Bootstrap program
  - Stored within the computer hardware in firmware
  - Locate the operating-system kernel and load it into memory
- System daemons
  - Services that are provided outside of the kernel by system programs that are loaded into memory at boot time - run the entire time the kernel is running
- Trap/Exception
  - A software-generated interrupt caused either by an error (for example, division by zero or invalid memory access) or by a specific request from a user program
  - An operating-system service be performed by executing a special operation called a system call

Operating-System Operations

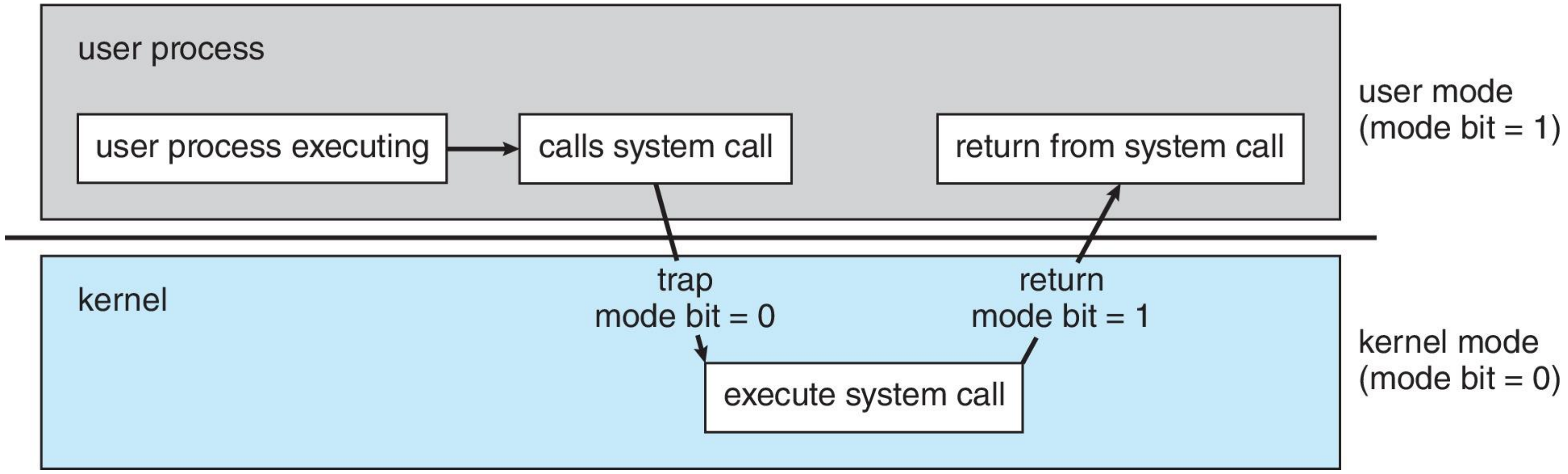
- Multiprogramming and Multitasking
  - Multiprogramming
    - increases CPU utilization, as well as keeping users satisfied by organizing programs so that the CPU always has one to execute
  - Multitasking
    - a logical extension of multiprogramming
    - the CPU executes multiple processes by switching among them, but the switches occur frequently, providing the user with a fast response time.

Note:** In a multiprogrammed system, a **program** in execution is termed a **process



Operating-System Operations

- Dual-Mode and Multimode Operation



Operating-System Operations

- Timer
  - We must ensure that the operating system maintains control over the CPU.
    - We cannot allow a user program to get stuck in an infinite loop or to fail to call system services and never return control to the operating system.
  - To accomplish this goal, we can use a timer. A timer can be set to interrupt the computer after a specified period.
    - Fixed (for example, 1/60 second) or Variable (for example, from 1 millisecond to 1 second). A variable timer is generally implemented by a fixed-rate clock and a counter.
    - The operating system sets the counter. Every time the clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs. For instance, a 10-bit counter with a 1-millisecond clock allows interrupts at intervals from 1 millisecond to 1,024 milliseconds, in steps of 1 millisecond.

Resource Management

- Resource Management
  - Process Management
  - Memory Management
  - File-System Management
  - Mass-Storage Management
  - Cache Management
  - I/O System Management

Resource Management

- Resource Management
  - Process Management
    - The OS is responsible for the following activities in connection with process management
      - Creating and deleting both user and system processes
      - Scheduling processes and threads on the CPUs
      - Suspending and resuming processes
      - Providing mechanisms for process synchronization
      - Providing mechanisms for process communication

Resource Management

- Resource Management
    - Memory Management
        - The OS is responsible for the following activities in connection with memory management:
            - Keeping track of which parts of memory are currently being used and which process is using them
            - Allocating and deallocating memory space as needed
            - Deciding which processes (or parts of processes) and data to move into and out of memory

Resource Management

- Resource Management
  - File-System Management
    - The OS is responsible for the following activities in connection with file management:
      - Creating and deleting files
      - Creating and deleting directories to organize files
      - Supporting primitives for manipulating files and directories
      - Mapping files onto mass storage
      - Backing up files on stable (nonvolatile) storage media

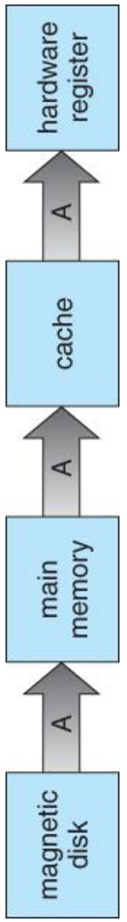
Resource Management

- Resource Management
  - Mass-Storage Management
    - The OS is responsible for the following activities in connection with secondary storage management:
      - Mounting and unmounting
      - Free-space management
      - Storage allocation
      - Disk scheduling
      - Partitioning
      - Protection

Resource Management

- Resource Management
  - Cache Management
    - Because caches have limited size, cache management is an important design problem. Careful selection of the cache size and of a replacement policy can result in greatly increased performance, as you can see by examining the following table

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



Resource Management

- Resource Management
  - I/O System Management
    - One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user. For example, in UNIX, the peculiarities of I/O devices are hidden from the bulk of the operating system itself by the I/O subsystem. The I/O subsystem consists of several components:
      - A memory-management component that includes buffering, caching, and spooling
      - A general device-driver interface
      - Drivers for specific hardware devices
    - Only the device driver knows the peculiarities of the specific device to which it is assigned.

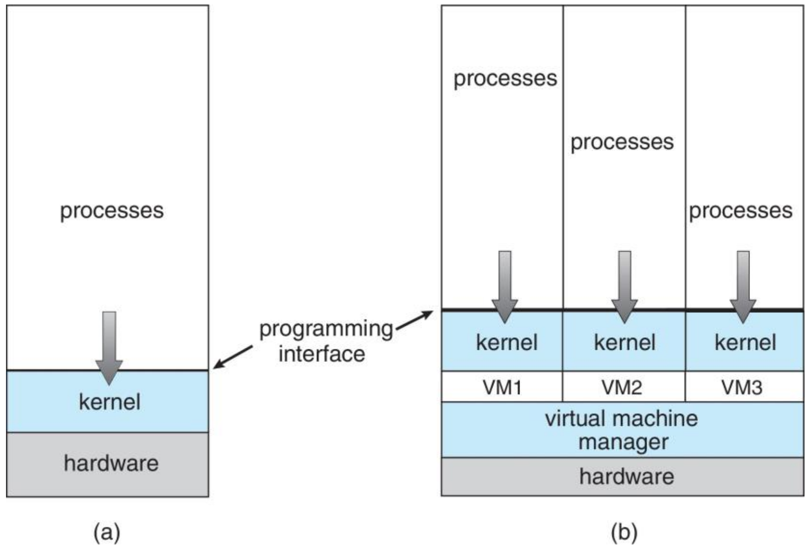
Security and Protection

- Security and Protection
  - Protection
    - Any mechanism for controlling the access of processes or users to the resources defined by a computer system.
  - Security
    - Defend a system from external and internal attacks.

Protection and security require the system to be able to distinguish among all its users. Most operating systems maintain a list of user names and associated user identifier (user IDs). In Windows parlance, this is a security ID (SID). These numerical IDs are unique, one per user. When a user logs in to the system, the authentication stage determines the appropriate user ID for the user

Virtualization

- Virtualization
  - A technology that allows us to abstract the hardware of a single computer (the CPU, memory, disk drives, network interface cards, and so forth) into several different execution environments, thereby creating the illusion that each separate environment is running on its own private computer.



Virtualization software is one member of a class that also includes emulation. Emulation, which involves simulating computer hardware in software, is typically used when the source CPU type is different from the target CPU type.

Distributed Systems

- A collection of physically separate, possibly heterogeneous computer systems that are networked to provide users with access to the various resources that the system maintains
  - Access to a shared resource increases computation speed, functionality, data availability, and reliability
    - Some operating systems generalize network access as a form of file access, with the details of networking contained in the network interface's device driver
    - Others make users specifically invoke network functions. Generally, systems contain a mix of the two modes—for example FTP and NFS

Note: *The protocols that create a distributed system can greatly affect that system's utility and popularity.*

Distributed Systems (cont…)

- Distributed systems depend on networking for their functionality
  - Networks vary by the protocols used, the distances between nodes, and the transport media

Distributed Systems (cont…)

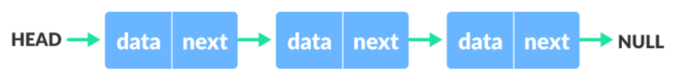
- Networks are characterized based on the distances between their nodes
  - Local-Area Network (LAN)
  - Wide-Area Network (WAN)
  - Metropolitan-Area Network (MAN)
  - Personal-Area Network (PAN)

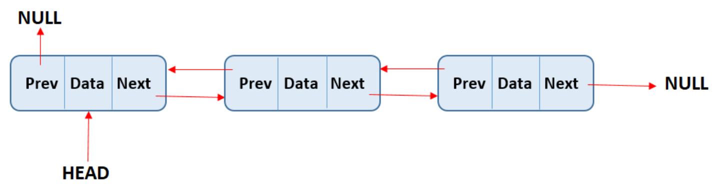
Distributed Systems (cont…)

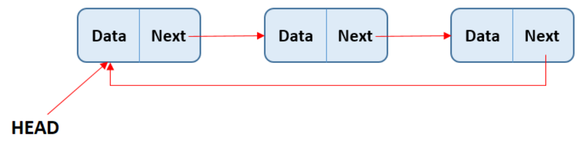
- Network OS vs Distributed OS
  - Network operating system
    - An operating system that provides features such as file sharing across the network, along with a communication scheme that allows different processes on different computers to exchange messages
    - A computer running a network operating system acts autonomously from all other computers on the network, although it is aware of the network and is able to communicate with other networked computers
  - A distributed operating system
    - Provides a less autonomous environment
    - The different computers communicate closely enough to provide the illusion that only a single operating system controls the network

Kernel Data Structures

- Lists, Stacks, and Queues
  - List
    - Singly linked list – each item points to its successor
    - Doubly linked list – a given item can refer either to its predecessor or to its successor
    - Circularly linked list – the last element in the list refers to the first element, rather than to null



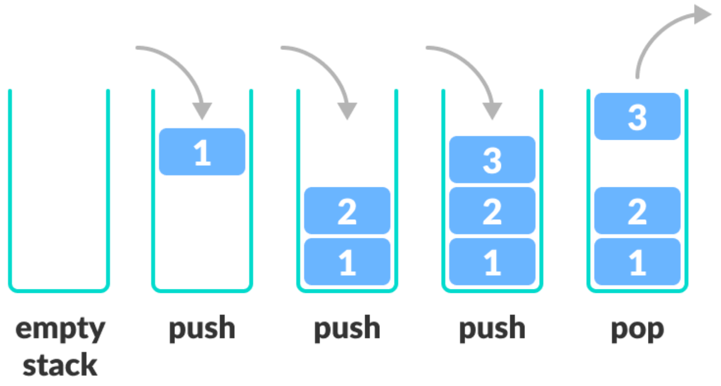




Lists are sometimes used directly by kernel algorithms – frequently, though, they are used for constructing more powerful data structures, such as stacks and queues

Kernel Data Structures

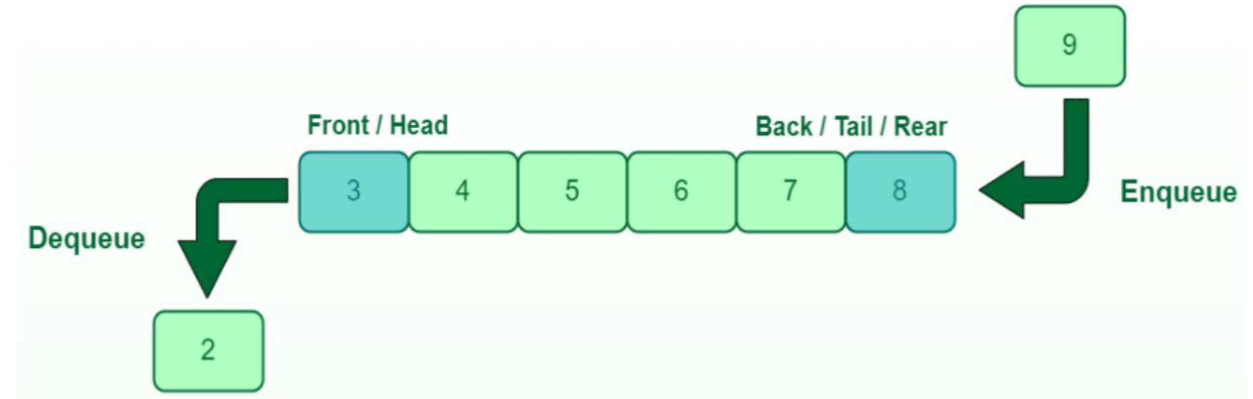
- Lists, Stacks, and Queues
  - Stack
    - LIFO (Last-In-First-Out) principle



An operating system often uses a stack when invoking function calls. Parameters, local variables, and the return address are pushed onto the stack when a function is called; returning from the function call pops those items off the stack

Kernel Data Structures

- Lists, Stacks, and Queues
  - Queue
    - FIFO (First-In-First-Out) principle

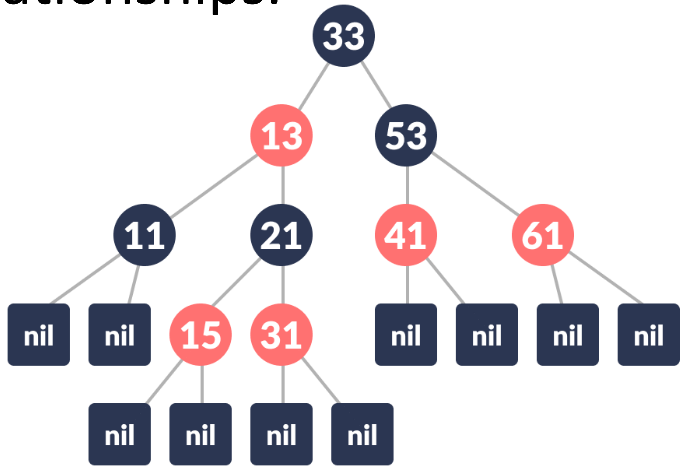


An operating system uses queues many a time

- *Tasks that are waiting to be run on an available CPU are often organized in queues*
- *Jobs that are sent to a printer are typically printed in the order in which they were submitted*

Kernel Data Structures

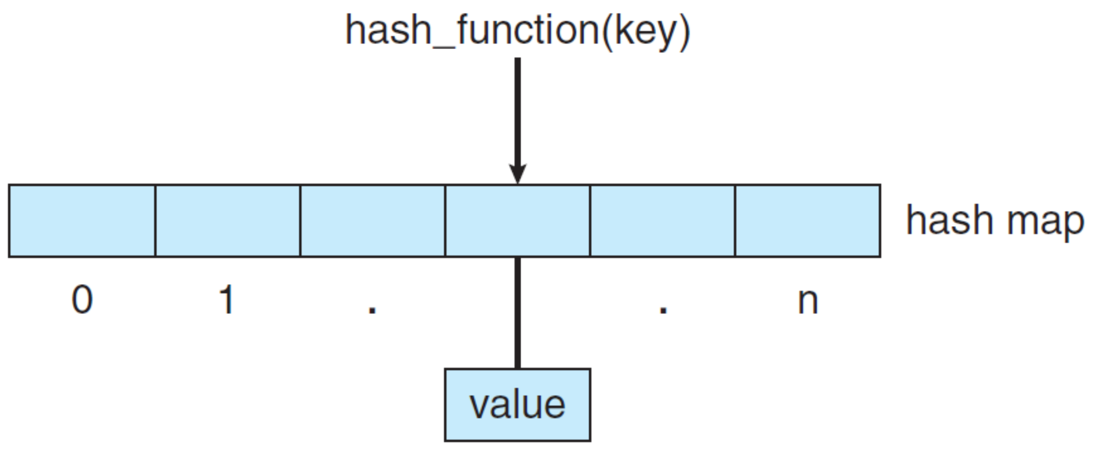
- Trees
  - A data structure that can be used to represent data hierarchically. Data values in a tree structure are linked through parent–child relationships.
    - General Tree
    - Binary Tree
    - Binary Search Tree
    - Balanced Binary Search Tree (AVL, Red-Black)



Application in OS: Linux uses the red-black tree as part its CPU-scheduling algorithm

Kernel Data Structures

- Hash Functions and Maps
  - Takes data as its input, performs a numeric operation on the data, and returns a numeric value. This numeric value can then be used as an index into a table (typically an array) to quickly retrieve the data



Kernel Data Structures

- Bitmaps
  - A string of n binary digits that can be used to represent the status of n items.
    - For example, suppose we have several resources, and the availability of each resource is indicated by the value of a binary digit: 0 means that the resource is available, while 1 indicates that it is unavailable (or vice versa). The value of the ith position in the bitmap is associated with the ith resource.

Q: *Consider the bitmap shown below and tell the meaning:*

001011101

A: Resources 2, 4, 5, 6, and 8 are unavailable; resources 0, 1, 3, and 7 are available

Computing Environments

- Traditional Computing
- Mobile Computing
- Client–Server Computing
- Peer-to-Peer Computing
- Cloud Computing
- Real-Time Embedded Systems

Computing Environments

- Traditional Computing
  - Earlier, PCs connected to a network, with servers providing file and print services

    [Remote access was awkward, and portability was achieved by use of laptop computers]

  - Now a days, network computers (or thin clients)—which are essentially terminals that understand web-based computing— are used in place of traditional workstations where more security or easier maintenance is desired

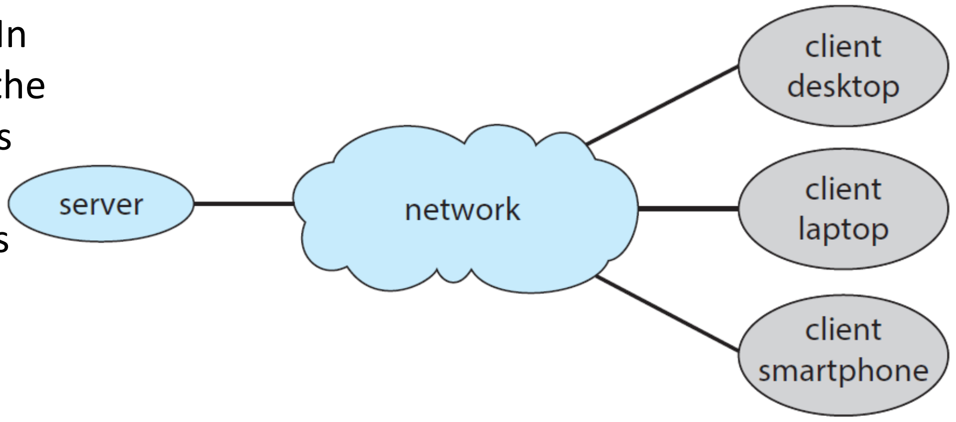
    [Mobile computers can synchronize with PCs to allow very portable use of company information]

Computing Environments

- Mobile Computing
  - Refers to computing on handheld smartphones and tablet computers - these devices share the distinguishing physical features of being portable and lightweight

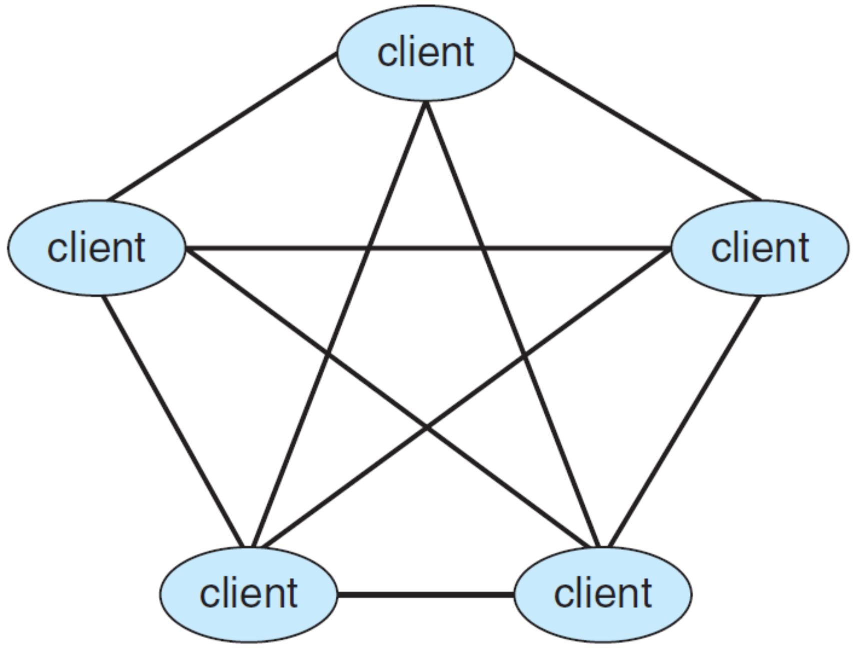
Computing Environments

- Client–Server Computing
  - Contemporary network architecture features arrangements in which server systems satisfy requests generated by client systems
  - A specialized distributed system
    - The compute-server system provides an interface to which a client can send a request to perform an action (for example, read data). In response, the server executes the action and sends the results to the client. A server running a database that responds to client requests for data is an example of such a system.
    - The file-serve system provides a file-system interface where clients can create, update, read, and delete files. An example of such a system is a web server that delivers files to clients running web browsers. The actual contents of the files can vary greatly, ranging from traditional web pages to rich multimedia content such as high-definition video



Computing Environments

- Peer-to-Peer Computing Model
  - Another structure for a distributed
  - Clients and servers are not distinguished from one another
    - All nodes within the system are considered peers, and each may act as either a client or a server, depending on whether it is requesting or providing a service
    - Offer an advantage over traditional client–server systems – *in a client–server system, the server is a bottleneck; but in a peer-to-peer system, services can be provided by several nodes distributed throughout the network*.



Computing Environments

- Cloud Computing
  - a type of computing that delivers computing, storage, and even applications as a service across a network
  - A logical extension of virtualization, because it uses virtualization as a base for its functionality

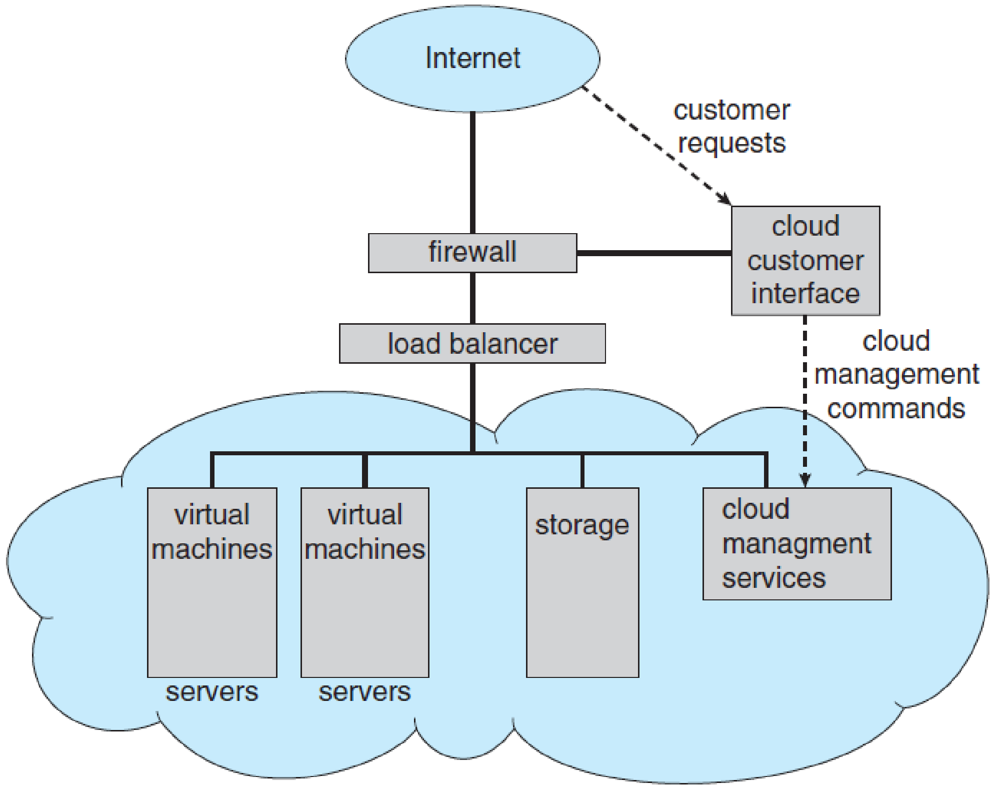
    Example: the Amazon Elastic Compute Cloud (ec2) facility has thousands of servers, millions of virtual machines, and petabytes of storage available for use by anyone on the Internet. Users pay per month based on how much of those resources they use.

Computing Environments

- Cloud Computing (cont…)
  - Types of Cloud-Computing
    - **Public cloud**—a cloud available via the Internet to anyone willing to pay for the services
    - **Private cloud**—a cloud run by a company for that company's own use
    - **Hybrid cloud**—a cloud that includes both public and private cloud components
    - **Software as a service (SaaS**)—one or more applications (such as word processors or spreadsheets) available via the Internet
    - **Platform as a service (PaaS)—**a software stack ready for application use via the Internet (for example, a database server)
    - **Infrastructure as a service (IaaS)—**servers or storage available over the Internet (for example, storage available for making backup copies of production data)

Computing Environments

- Cloud Computing (cont…)
    - **Example** - a public cloud providing IaaS



Computing Environments

- Real-Time Embedded Systems
    - run on are usually primitive, and so the operating systems provide limited features
    - have little or no user interface, preferring to spend their time monitoring and managing hardware devices, such as automobile engines and robotic arms
    - vary considerably
        - Some are general-purpose computers, running standard operating systems—such as Linux—with special-purpose applications to implement the functionality
        - Others are hardware devices with a special-purpose embedded operating system providing just the functionality desired

Free and Open-Source Operating Systems

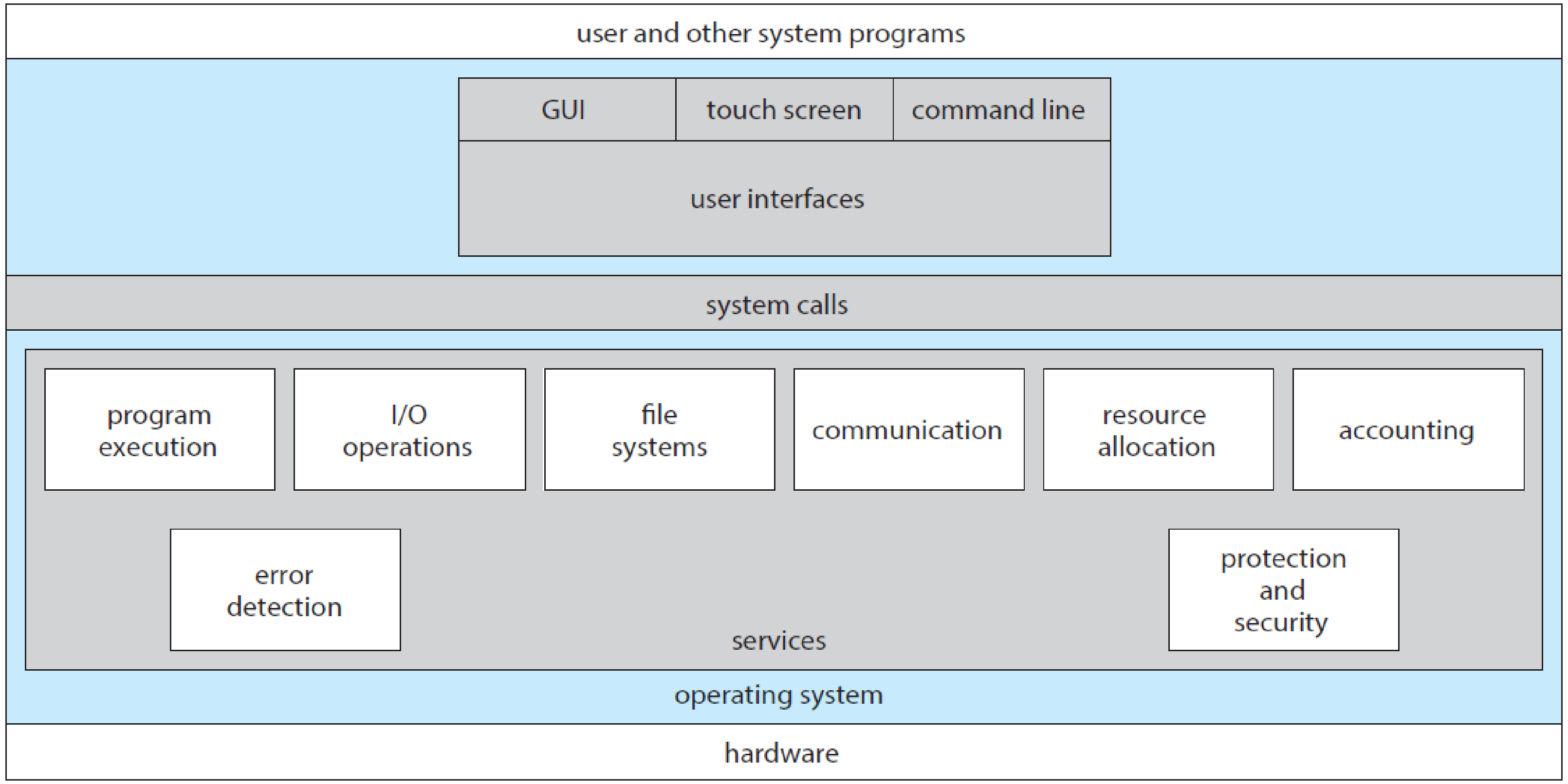
- Both free operating systems and open-source operating systems are available in source-code format rather than as compiled binary code
  - Free software (sometimes referred to as free/libre software) not only makes source code available but also is licensed to allow no-cost use, redistribution, and modification.
  - Open-source software does not necessarily offer such licensing.

➢All free software is open source, some open-source software is not “free.”

- GNU/Linux is the most famous open-source operating system, with some distributions free and others open source only
- Microsoft Windows is a well-known example of the opposite closed-source approach. Windows is proprietary software—Microsoft owns it, restricts its use, and carefully protects its source code.

Overview of System Calls

- A view of operating system services



Overview of System Calls

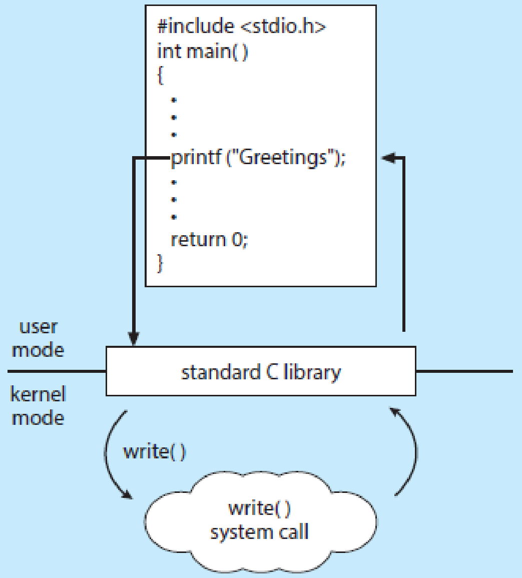
- Application Programming Interface
  - Windows API for Windows systems
  - POSIX API for POSIX-based systems (which include virtually all versions of UNIX, Linux, and macOS)
  - Java API for programs that run on the Java virtual machine.

Overview of System Calls

- Application Programming Interface

THE STANDARD C LIBRARY

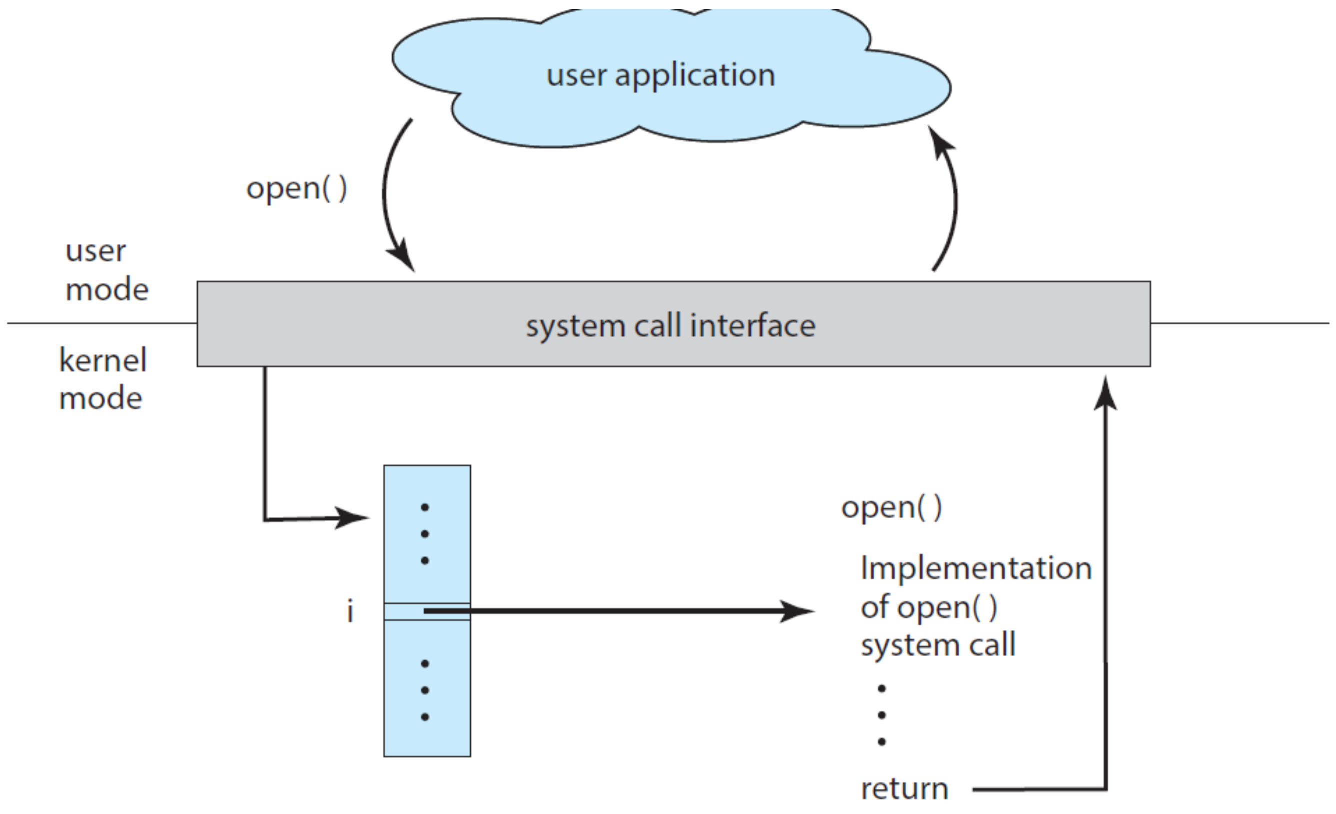
The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, let's assume a C program invokes the `printf()` statement. The C library intercepts this call and invokes the necessary system call (or calls) in the operating system—in this instance,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



```
#include <stdio.h>
int main( )
{
  •
  •
  •
  printf ("Greetings");
  •
  •
  •
  return 0;
}
```

Overview of System Calls

- The handling of a user application invoking the **open()** system call



Overview of System Calls

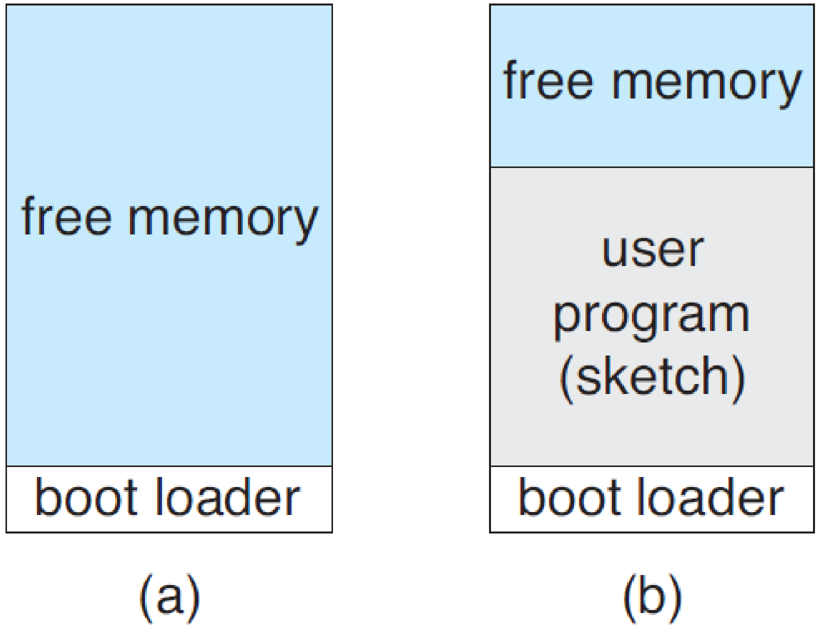
- Types of system calls
    - Process control
    - File management
    - Device management
    - Information maintenance
    - Communications
    - Protection

Overview of System Calls

- Types of system calls
  - Process control
    - create process, terminate process
    - load, execute
    - get process attributes, set process attributes
    - wait event, signal event
    - allocate and free memory

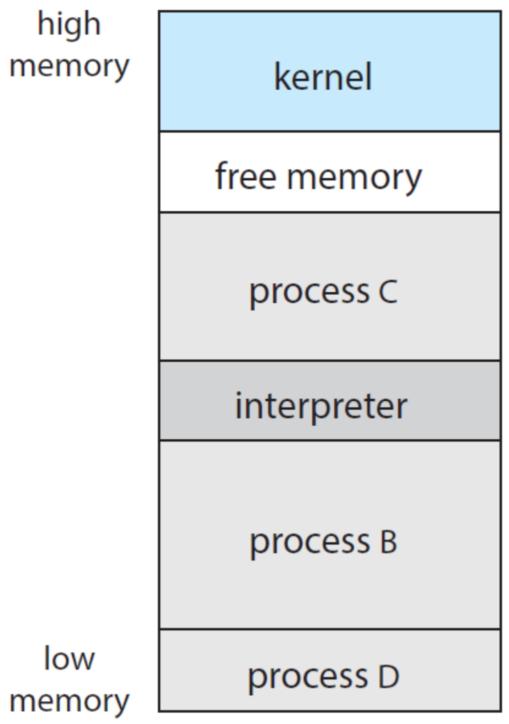
Overview of System Calls

- Types of system calls
  - Process control
    - Example 1: Arduino execution. (a) At system startup. (b) Running a sketch



Overview of System Calls

- Types of system calls
  - Process control
    - Example 2: FreeBSD running multiple programs



Overview of System Calls

- Types of system calls
  - File management
    - create file, delete file
    - open, close
    - read, write, reposition
    - get file attributes, set file attributes

Overview of System Calls

- Types of system calls
  - Device management
    - request device, release device
    - read, write, reposition
    - get device attributes, set device attributes
    - logically attach or detach devices

Overview of System Calls

- Types of system calls
  - Information maintenance
    - get time or date, set time or date
    - get system data, set system data
    - get process, file, or device attributes
    - set process, file, or device attributes

Overview of System Calls

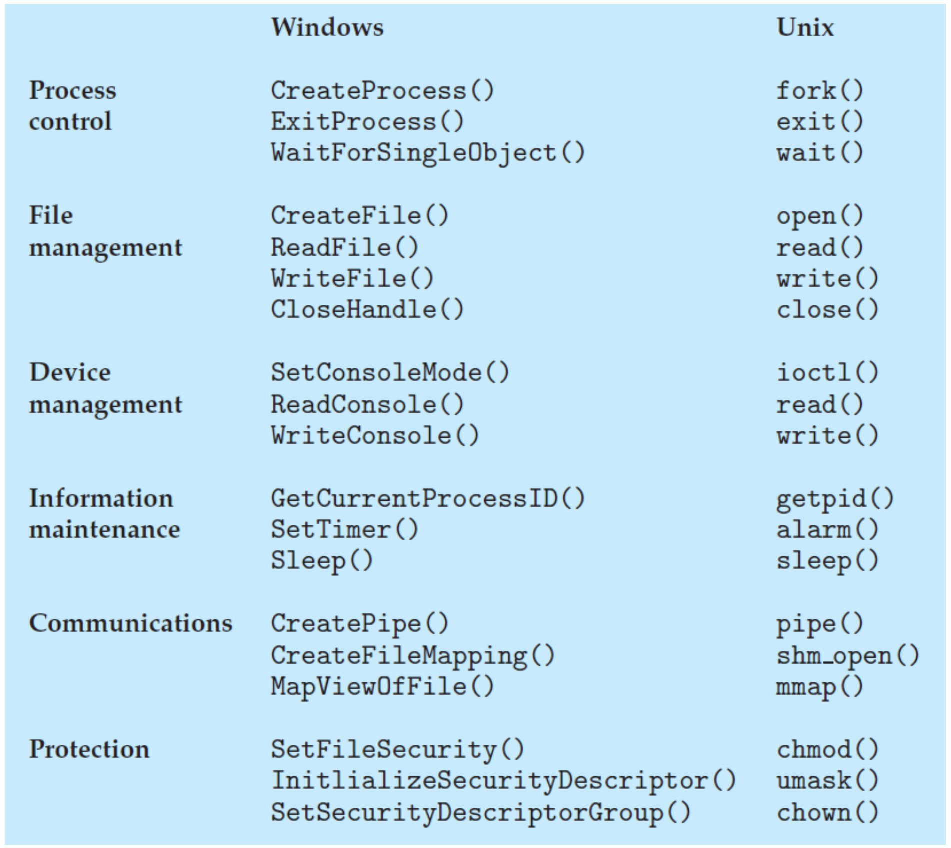
- Types of system calls
  - Communications
    - create, delete communication connection
    - send, receive messages
    - transfer status information
    - attach or detach remote devices

Overview of System Calls

- Types of system calls
  - Protection
    - get file permissions
    - set file permissions

Overview of System Calls

- Various equivalent system calls for Windows and UNIX OSs



	Windows	Unix

Process control	CreateProcess()
	ExitProcess()
	WaitForSingleObject()
		fork()
		exit()
		wait()

File management	CreateFile()
	ReadFile()
	WriteFile()
	CloseHandle()
		open()
		read()
		write()
		close()

Device management	SetConsoleMode()
	ReadConsole()
	WriteConsole()
		ioctl()
		read()
		write()

Information maintenance	GetCurrentProcessID()
	SetTimer()
	Sleep()
		getpid()
		alarm()
		sleep()

Communications	CreatePipe()
	CreateFileMapping()
	MapViewOfFile()
		pipe()
		shm_open()
		mmap()

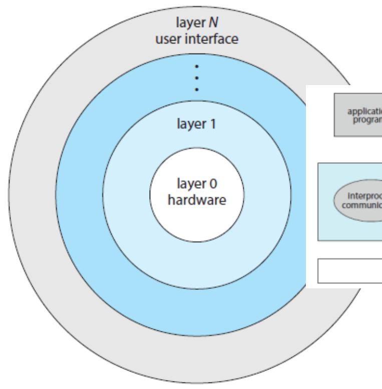
Protection	SetFileSecurity()
	InitlializeSecurityDescriptor()
	SetSecurityDescriptorGroup()
		chmod()
		umask()
		chown()

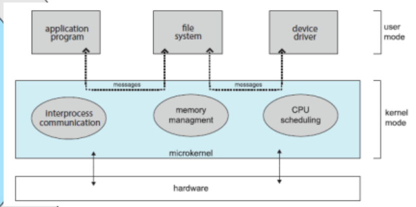
Reference

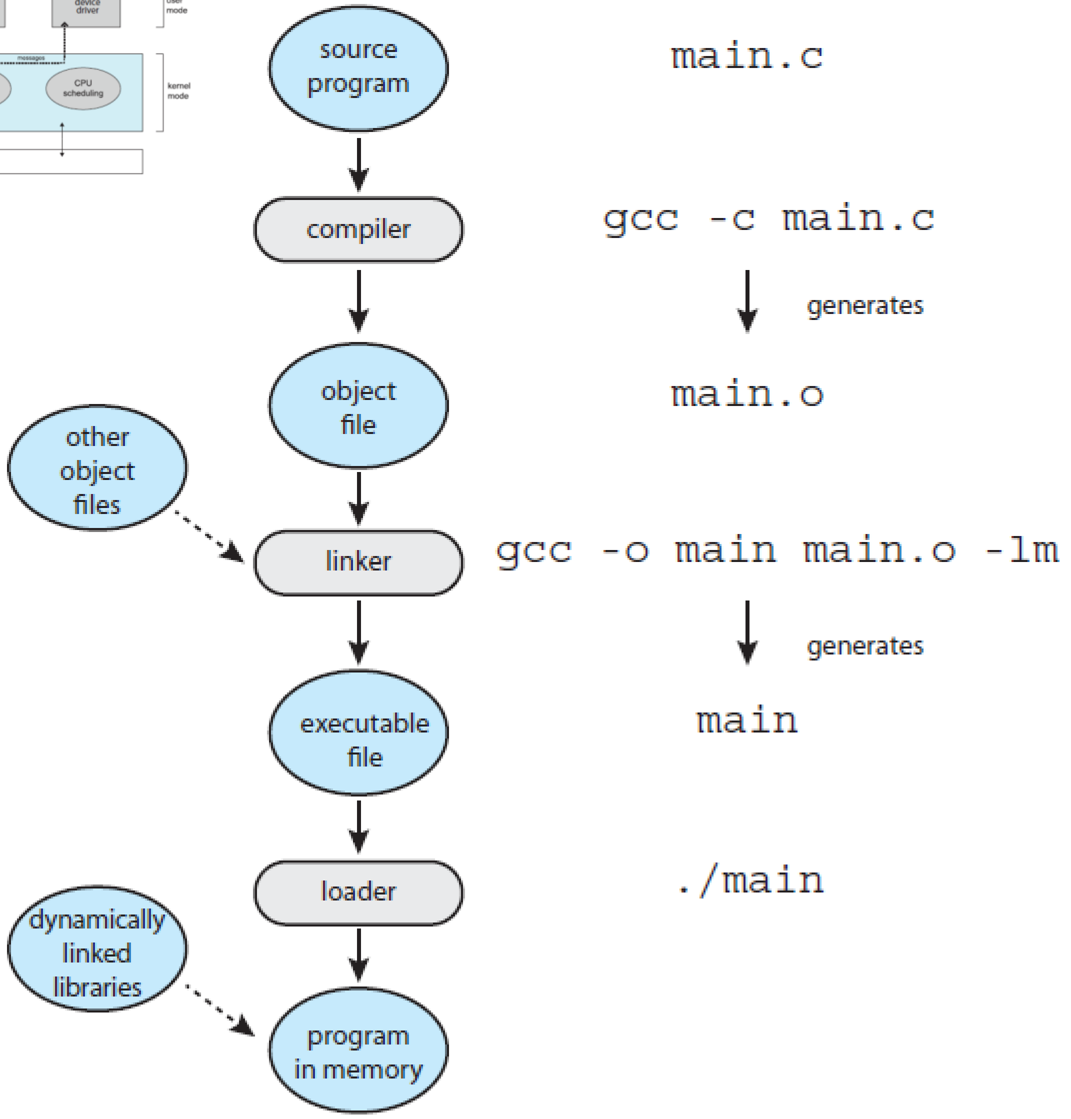
- Abraham Silberschatz, Peter B. Galvin, Greg Gagne, “Operating System Concepts”, Addison-Wesley, 10th edition, 2018
  - Chapter 1: Section 1.1 – 1.11
  - Chapter 2: Section 2.1 – 2.3

Self Study*

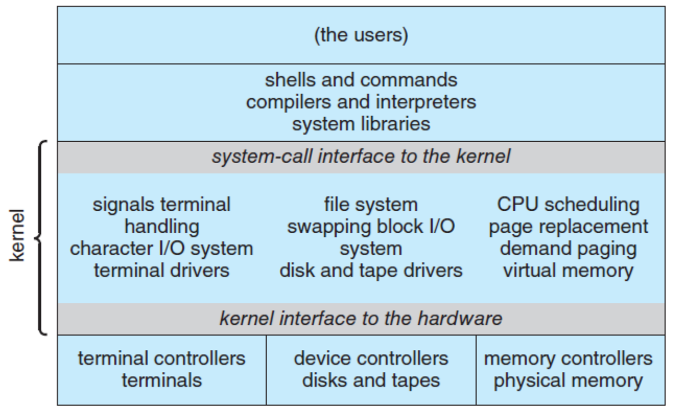
- Linkers and Loaders
- Operating System Structures

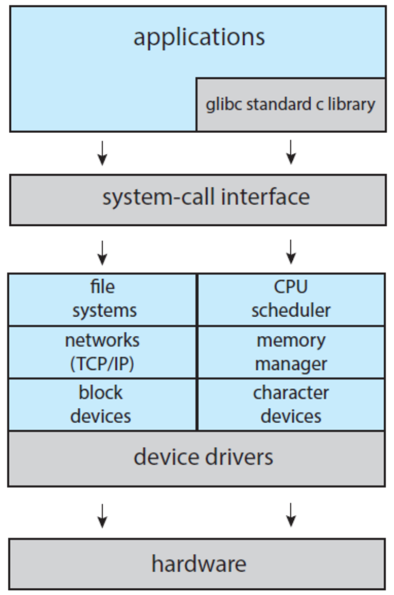


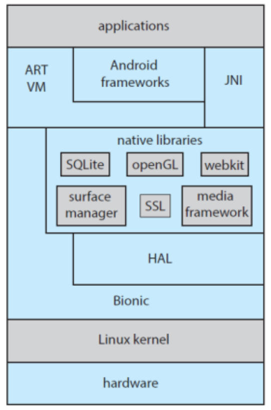




```
main.c
gcc -c main.c
main.o
gcc -o main main.o -lm
main
./main
```







Next

- Process and Threads (Module-2)

Thank You