CTBTCSE – SIV P4 – Operating System

"Introduction to Operating System"

Dr. Parag Shukla
Assistant Professor,
School of Cyber Security and Digital Forensics
National Forensic Sciences University

INTRODUCTION TO OPERATING SYSTEM

Presentation Outline

Introduction

History of Operating System

Types of Operating System

Operating System Concepts

Computer System Organization and Architecture

Resource Management Review

Security & Protection, Services, User & OS Interface

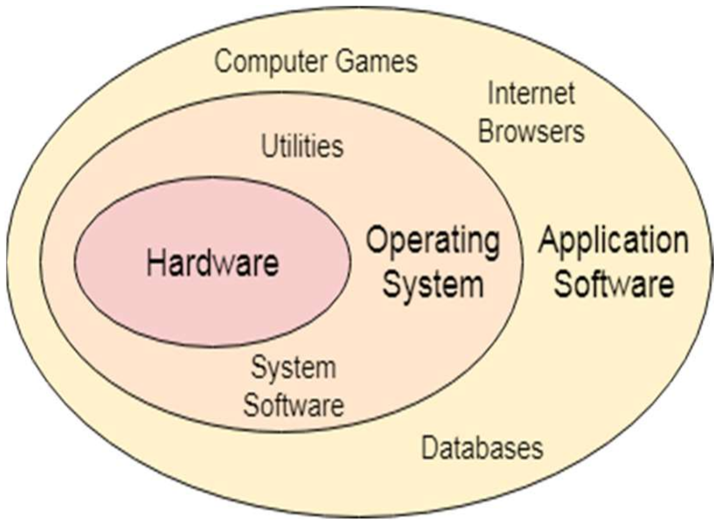
System Calls, Linkers & Loaders

OS Design & Implementation, OS Structure,

Building & Booting of OS, OS Debugging

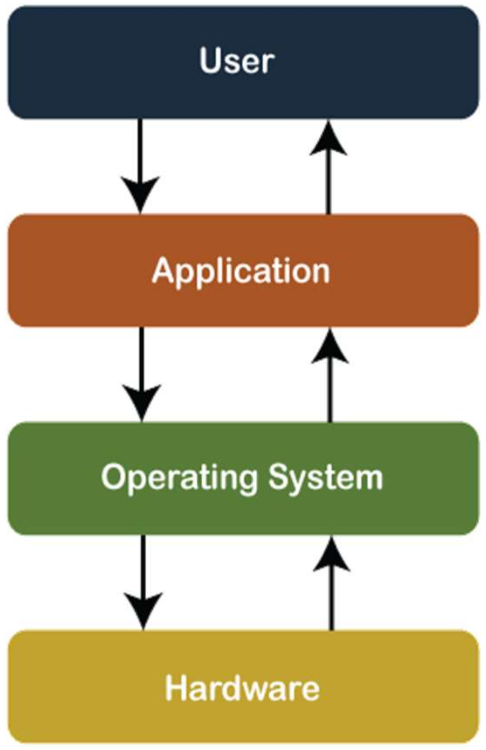
Introduction to Operating System

- ✓Operating System can be defined as an interface between user and the hardware.
- ✓It provides an environment to the user so that, the user can perform its task in convenient and efficient way.
- ✓In the Computer System (comprises of Hardware and software), Hardware can only understand machine code (in the form of 0 and 1) which doesn't make any sense to a naive user.
- ✓We need a system which can act as an intermediary and manage all the processes and resources present in the system.
- ✓An **Operating System** can be defined as an **interface between user and hardware**. It is responsible for the execution of all the processes, Resource Allocation, CPU management, File Management and many other tasks.
- ✓The purpose of an operating system is to provide an environment in which a user can execute programs in convenient and efficient manner



Introduction to Operating System

- **What is an Operating System?**
  - An operating system (OS) is the program that, after being initially loaded into the computer by a boot program, manages all of the other application programs in a computer.
  - The application programs make use of the operating system by making requests for services through a defined application program interface (API).
  - In addition, users can interact directly with the operating system through a user interface, such as a command-line interface (CLI) or a graphical UI (GUI).
  - Example: Apple macOS, Microsoft Windows, Google's Android OS, Linux Operating System, fedora, BlackBerry, etc



Introduction to Operating System

- **Why use an Operating System?**
  - An operating system brings powerful benefits to computer software and software development.
  - Without an operating system, every application would need to include its own UI and the comprehensive code needed to handle all low-level functionality of the underlying computer, such as disk storage, network interfaces, and so on.
  - Considering the vast array of underlying hardware available, this would vastly bloat the size of every application and make software development impractical.
  - Instead, many common tasks, such as sending a network packet or displaying text on a standard output device, such as a display, can be offloaded to system software that serves as an intermediary between the applications and the hardware.
  - The system software provides a consistent and repeatable way for applications to interact with the hardware without the applications needing to know any details about the hardware.

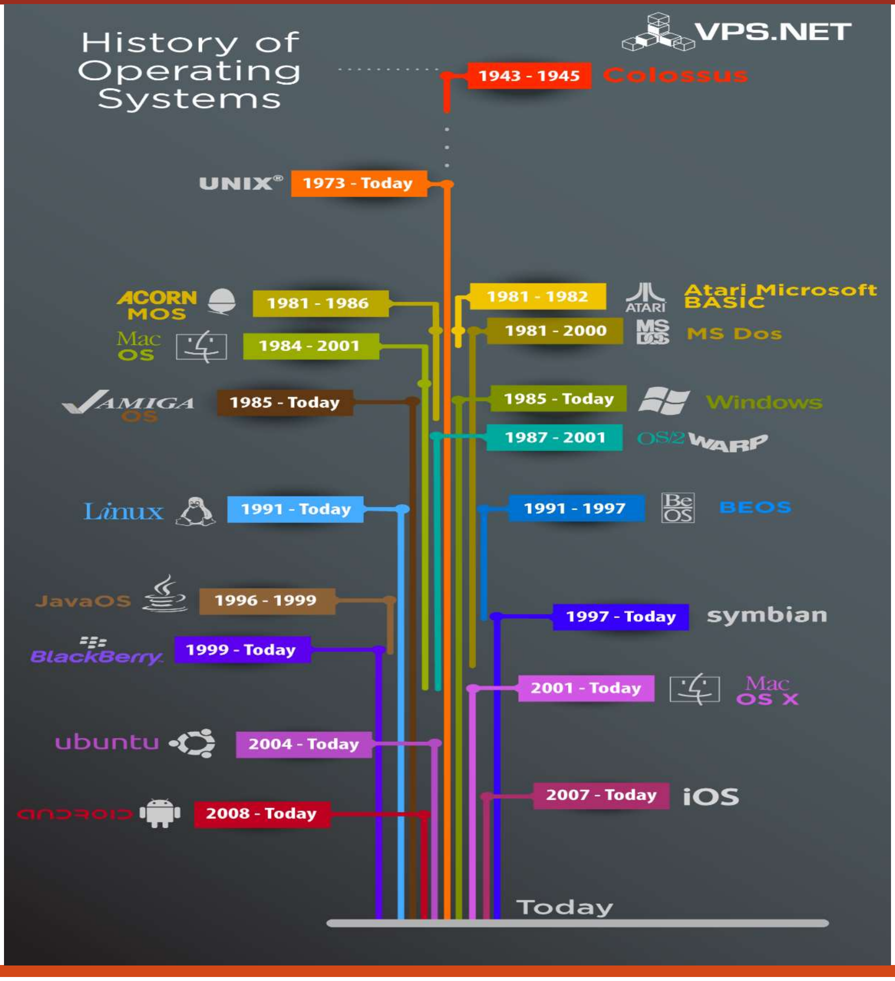
Introduction to Operating System

- **Why use an Operating System? (Continue…)**
  - As long as each application accesses the same resources and services in the same way, that system software -- the operating system -- can service almost any number of applications.
  - This vastly reduces the amount of time and coding required to develop and debug an application, while ensuring that users can control, configure and manage the system hardware through a common and well-understood interface.
  - Once installed, the operating system relies on a vast library of device drivers to tailor OS services to the specific hardware environment.
  - Thus, every application may make a common call to a storage device, but the OS receives that call and uses the corresponding driver to translate the call into actions (commands) needed for the underlying hardware on that specific computer.

Introduction to Operating System

S. No.	System Software	Operating System
1.	The software which manages the resources and makes the interaction between a user and machine possible is system software.	An operating system is a software that communicates with your computer hardware and provides a place to run the application.
2.	System software manages the system.	Operating System manages system as well as system software.
3.	Types of System Software- • Operating Systems- MS Windows, Mac OS, Linux, etc. • Device Drivers- Display drivers, USB drivers, etc. • Firmware- BIOS, Embedded Systems, etc. • Language Translators- Compiler, Interpreter, Assembler • Utility- Antivirus software, WinRAR, etc.	Types of Operating System- • Batch Operating System • Multiprocessing Operating System • Multiprogramming Operating System • Multi Tasking Operating System • Time Sharing Operating System • Real-Time Operating System • Distributed Operating System • Network Operating System
4.	It’s run only when required.	It runs all the time.
5.	It loads in the main memory whenever required.	It resides in the main memory all the time while the system is on.
6.	It is loaded by the operating system.	It resides in the main memory all the time while the system is on.
7.	Examples of system software are MacOS, Android, and Microsoft windows.	Examples of OS are Windows, OS X, and Linux.

History of Operating System



- **The First Generation (1940 to early 1950s)**
  - Electronic Digital Computer
  - Colossus was a set of computers developed by British codebreakers in the years 1943–1945 during world War II to help in the cryptanalysis of the Lorenz cipher
- **The Second Generation (1955 - 1965)**
  - Single Stream, Batch Processing
  - It collects all similar jobs in a batches and then submit to OS
  - General Motors has developed OS for IBM
- **The Third Generation (1965 - 1980)**
  - Multi-Programming
  - Simultaneously perform multiple task in single computer
- **The Fourth Generation (1980 - Present Day)**
  - Development of Personal Computer
  - Windows

Advantages and Disadvantages of OS

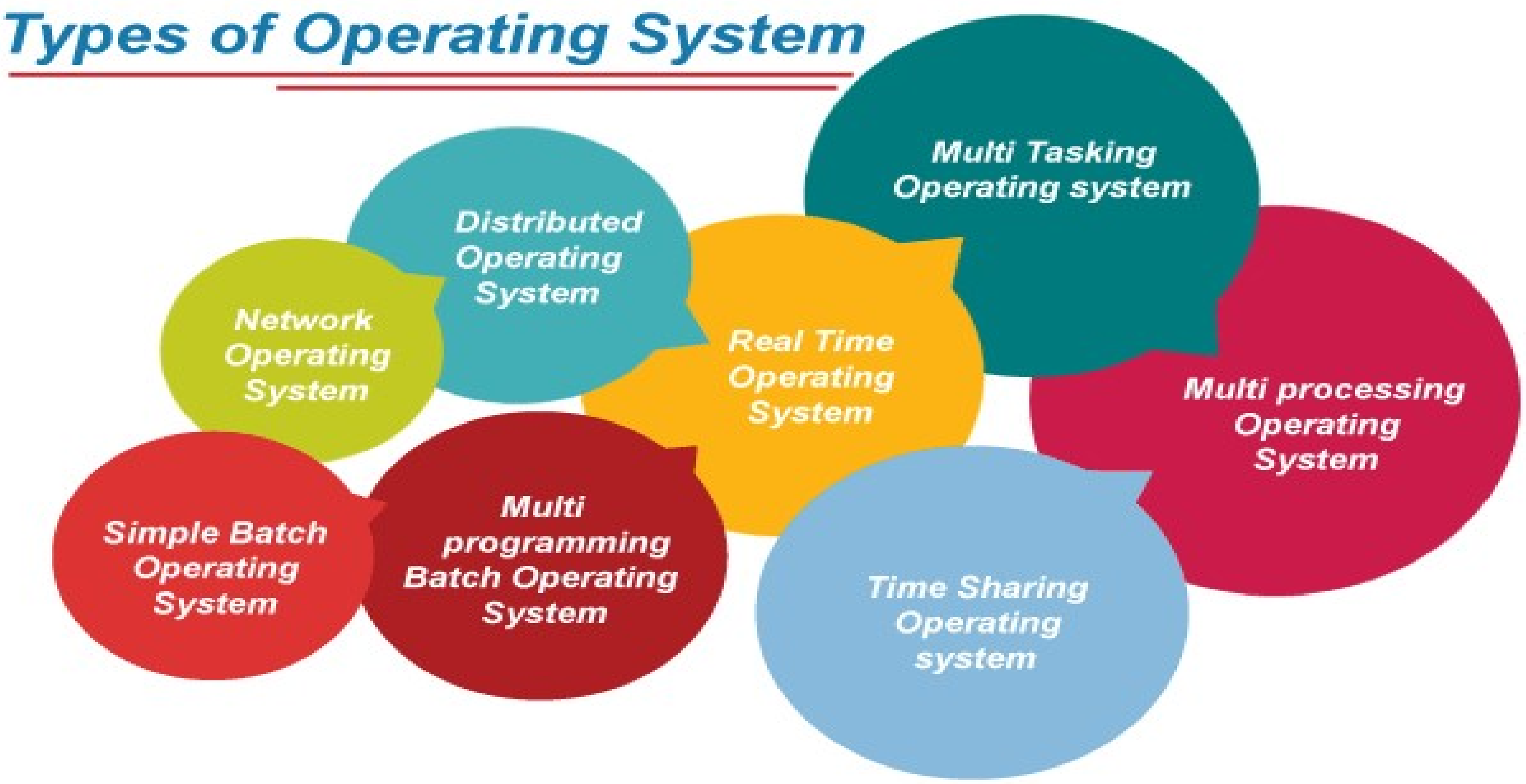
Advantages of Operating System

- It is helpful to monitor and regulate resources.
- It can easily operate since it has a basic graphical user interface to communicate with your device.
- It is used to create interaction between the users and the computer application or hardware.
- The performance of the computer system is based on the CPU.
- The response time and throughput time of any process or program are fast.
- It can share different resources like fax, printer, etc.
- It also offers a forum for various types of applications like system and web application.

Disadvantage of the Operating System

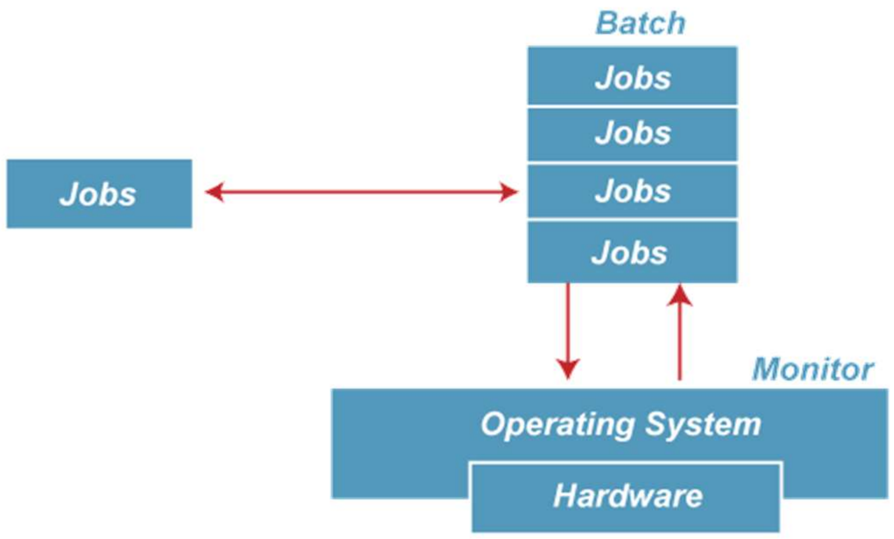
- It allows only a few tasks that can run at the same time.
- If any error occurred in the operating system; the stored data can be destroyed.
- It is a very difficult task or works for the OS to provide entire security from the viruses because any threat or virus can occur at any time in a system.
- An unknown user can easily use any system without the permission of the original user.
- The cost of operating system costs is very high.

Types of Operating System


Types of Operating System
Multi Tasking Operating system
Distributed Operating System
Network Operating System
Real Time Operating System
Multi processing Operating System
Simple Batch Operating System
Multi programming Batch Operating System
Time Sharing Operating system

Types of Operating System – Batch Operating System

- Batch Operating System
  - In the 1970s, Batch processing was very popular.
  - In this technique, similar types of jobs were batched together and executed in time.
  - People were used to having a single computer which was called a mainframe.
  - In Batch operating system, access is given to more than one person; they submit their respective jobs to the system for the execution.
  - The system put all of the jobs in a queue on the basis of first come first serve and then executes the jobs one by one.
  - The users collect their respective output when all the jobs get executed.



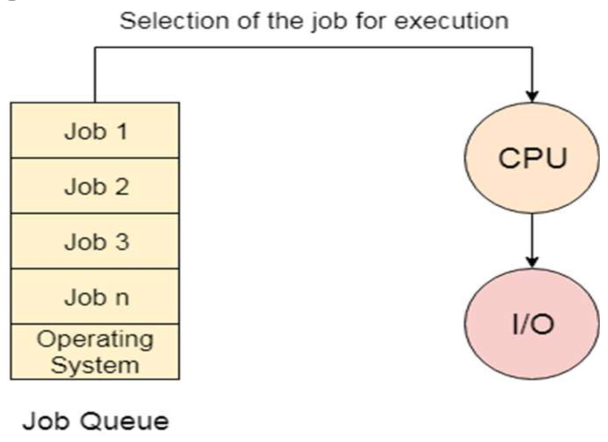
- The purpose of this operating system was mainly to transfer control from one job to another as soon as the job was completed.
- It contained a small set of programs called the resident monitor that always resided in one part of the main memory.
- The remaining part is used for servicing jobs.

Types of Operating System – Batch Operating System

- Advantages of Batch Operating System
  - The use of a resident monitor improves computer efficiency as it eliminates CPU time between two jobs.
- Disadvantages of Batch OS

1. Starvation

Batch processing suffers from starvation.



- There are five jobs J1, J2, J3, J4, and J5, present in the batch.
- If the execution time of J1 is very high, then the other four jobs will never be executed, or they will have to wait for a very long time.
- Hence the other processes get starved.

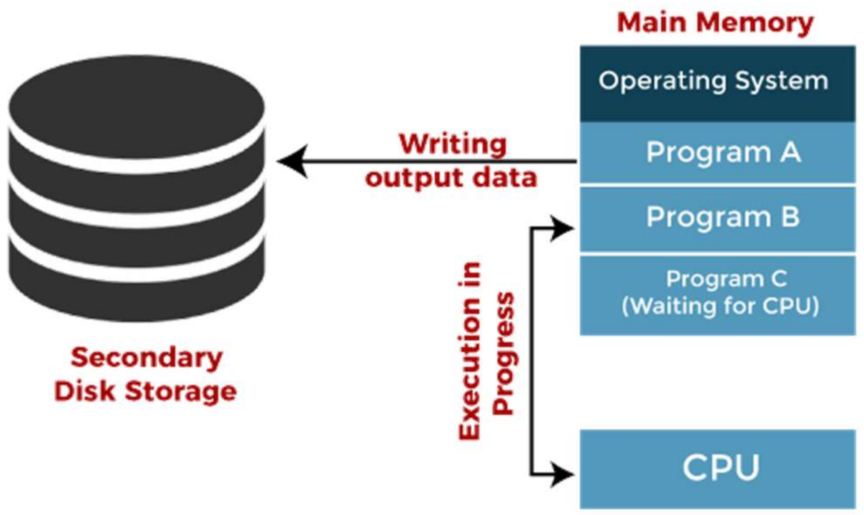
2. Not Interactive

- Batch Processing is not suitable for jobs that are dependent on the user's input.
- If a job requires the input of two numbers from the console, then it will never get it in the batch processing scenario since the user is not present at the time of execution.

Types of OS – Multi Programming Operating System

Multi Programming Operating System

- Multiprogramming is an extension to batch processing where the CPU is always kept busy.
- Each process needs two types of system time:
  - CPU time and
  - IO time.
- In a multiprogramming environment, when a process does its I/O, The CPU can start the execution of other processes.
- Therefore, multiprogramming improves the efficiency of the system.



Jobs in multiprogramming system

Advantages of Multiprogramming OS

- Throughout the system, it increased as the CPU always had one program to execute.
- Response time can also be reduced.

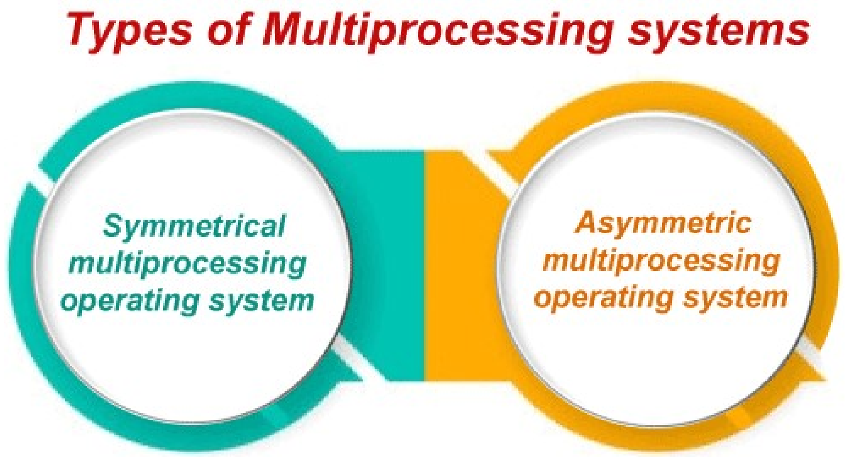
Disadvantages of Multiprogramming OS

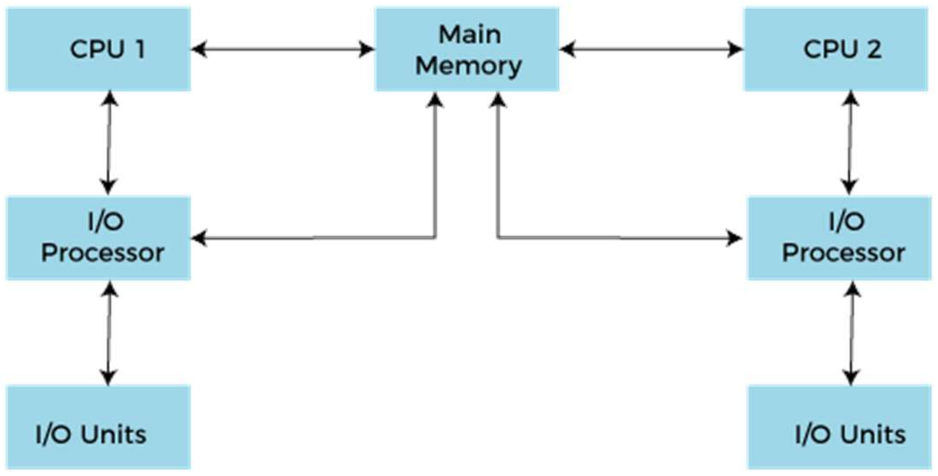
- Multiprogramming systems provide an environment in which various systems resources are used efficiently, but they do not provide any user interaction with the computer system.

Types of OS – Multi Processing Operating System

Multi Processing Operating System

- In Multiprocessing, Parallel computing is achieved.
- There are more than one processors present in the system which can execute more than one process at the same time.
- This will increase the throughput of the system.
- There are two types of multi-processing systems
  1. Symmetrical Multi processing operating system
  2. Asymmetric multiprocessing operating system


Types of Multiprocessing systems
Symmetrical multiprocessing operating system
Asymmetric multiprocessing operating system



Working of Multiprocessor System

Types of OS – Multi Processing Operating System

S. No.	Asymmetric Multiprocessing	Symmetric Multiprocessing
1.	In asymmetric multiprocessing, the processors are not treated equally.	In symmetric multiprocessing, all the processors are treated equally.
2.	Tasks of the operating system are done by master processor.	Tasks of the operating system are done individual processor.
3.	No Communication between Processors as they are controlled by the master processor.	All processors communicate with another processor by a shared memory.
4.	In asymmetric multiprocessing, process scheduling approach used is master-slave.	In symmetric multiprocessing, the process is taken from the ready queue.
5.	Asymmetric multiprocessing systems are cheaper.	Symmetric multiprocessing systems are costlier.
6.	Asymmetric multiprocessing systems are easier to design.	Symmetric multiprocessing systems are complex to design.
7.	All processors can exhibit different architecture.	The architecture of each processor is the same.
8.	It is simple as here the master processor has access to the data, etc.	It is complex as synchronization is required of the processors in order to maintain the load balance.

Types of OS – Multi Processing Operating System

Advantages of Multiprocessing OS

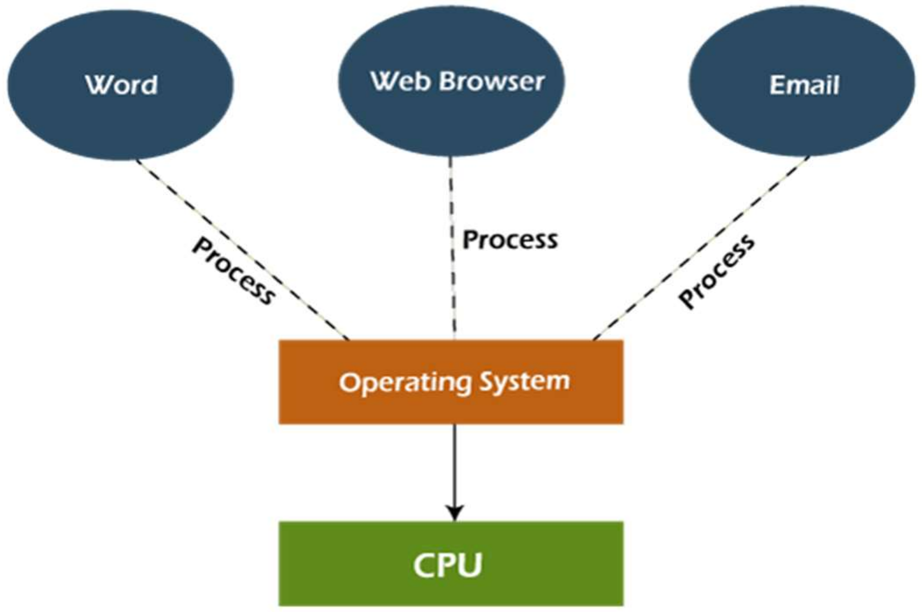
- **Increased reliability:**
  - Due to the multiprocessing system, processing tasks can be distributed among several processors.
  - This increases reliability as if one processor fails, the task can be given to another processor for completion.
- **Increased throughout:**
  - As several processors increase, more work can be done in less time.

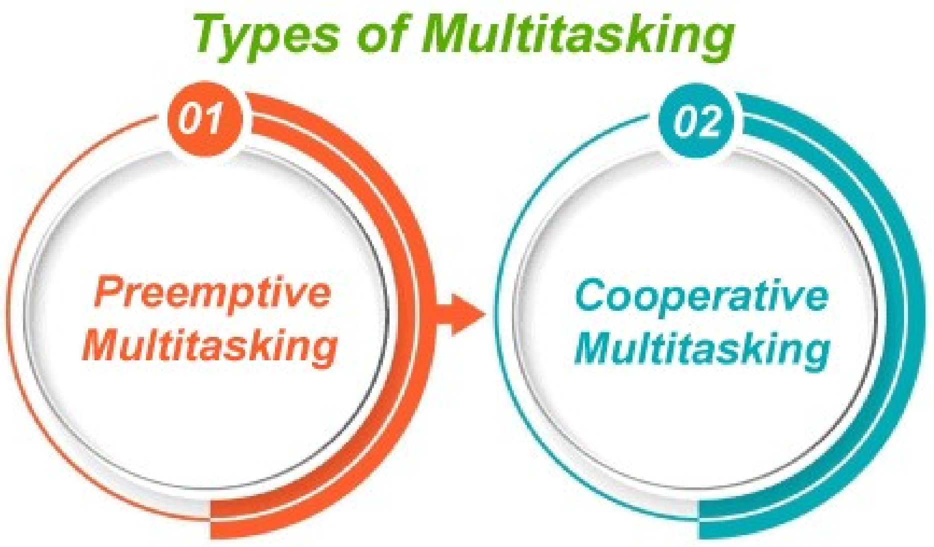
Disadvantages of Multiprocessing OS

- Multiprocessing operating system is more complex and sophisticated as it takes care of multiple CPUs simultaneously.

Types of OS – Multi Tasking Operating System

- The multitasking operating system is a logical extension of a multiprogramming system that enables **multiple** programs simultaneously.
- It allows a user to perform more than one computer task at the same time.





Types of OS – Multi Tasking Operating System

Features	Preemptive Multitasking	Cooperative Multitasking
Definition	The OS may begin context switching from one running process to another.	The OS never starts context switching from one executing process to another process.
Control	It interrupts programs and grants control to processes that are not under the control of the application.	It never unexpectedly interrupts a process.
Ideal	It is ideal for multiple users.	It is ideal for a single user.
Cores	It may use multiple cores.	It may use a single core.
Malicious Program	It occurs when a malicious application initiates an indefinite loop that only affects itself and has no effect on other programs or threads.	A malicious application may block the entire system in cooperative multitasking by busy waiting or performing an indefinite loop and refusing to give up control.
Applications	It forces apps to share the processor, whether they want to or not.	All applications must collaborate for it to work. If one program doesn't cooperate, it may use all of the processor resources.
Examples	UNIX, Windows 95, and Windows NT	Macintosh OS versions 8.0-9.2.2 and Windows 3.x

Types of OS – Multi Tasking Operating System

Advantages of Multitasking OS

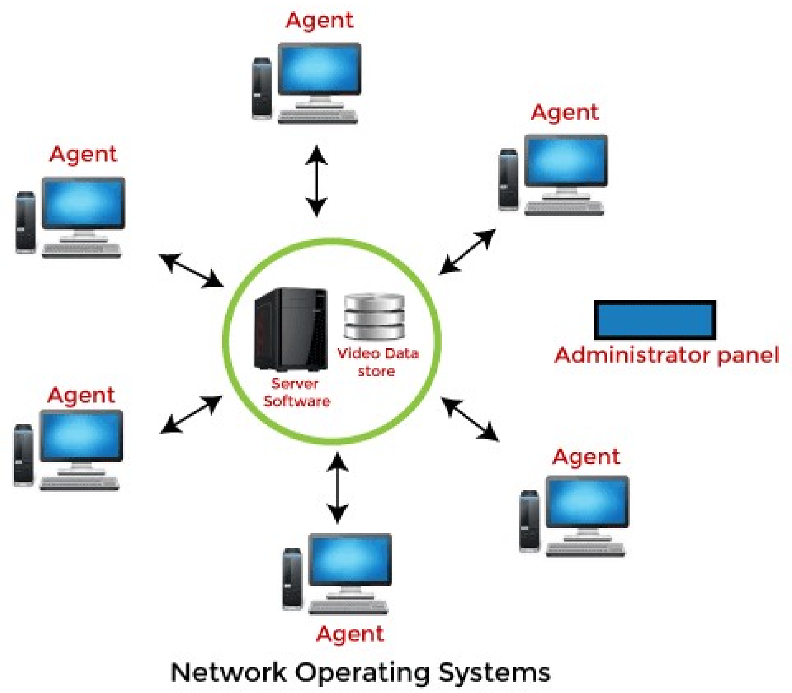
- This operating system is more suited to supporting multiple users simultaneously.
- The multitasking operating systems have well-defined memory management.

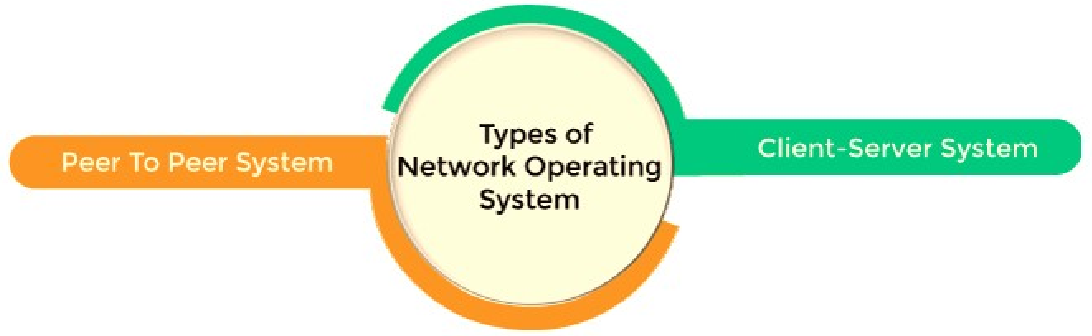
Disadvantages of Multitasking OS

- The multiple processors are busier at the same time to complete any task in a multitasking environment, so the CPU generates more heat.

Types of OS – Network Operating System

- An Operating system, which includes software and associated protocols to communicate with other computers via a network conveniently and cost-effectively, is called Network Operating System.





Types of OS – Network Operating System

Advantages of Network Operating System

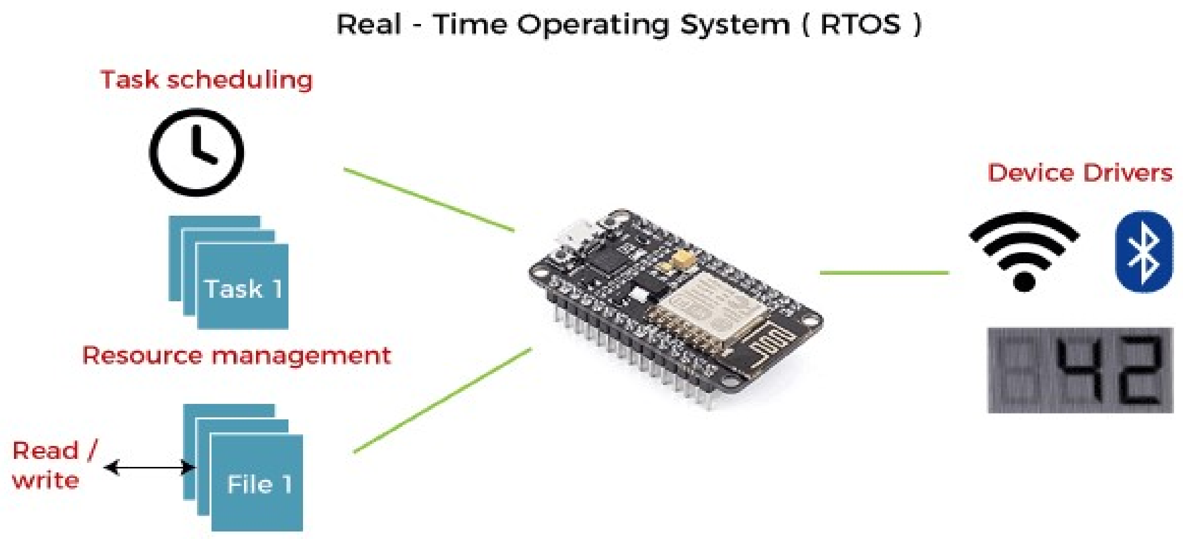
- In this type of operating system, network traffic reduces due to the division between clients and the server.
- This type of system is less expensive to set up and maintain.

Disadvantages of Network Operating System

- In this type of operating system, the failure of any node in a system affects the whole system.
- Security and performance are important issues. So trained network administrators are required for network administration.

Types of OS – Real-Time Operating System

- In Real-Time Systems, each job carries a certain deadline within which the job is supposed to be completed, otherwise, the huge loss will be there, or even if the result is produced, it will be completely useless.
- The Application of a Real-Time system exists in the case of military applications, if you want to drop a missile, then the missile is supposed to be dropped with a certain precision.



Types of OS – Real-Time Operating System

Advantages of Real-time operating system:

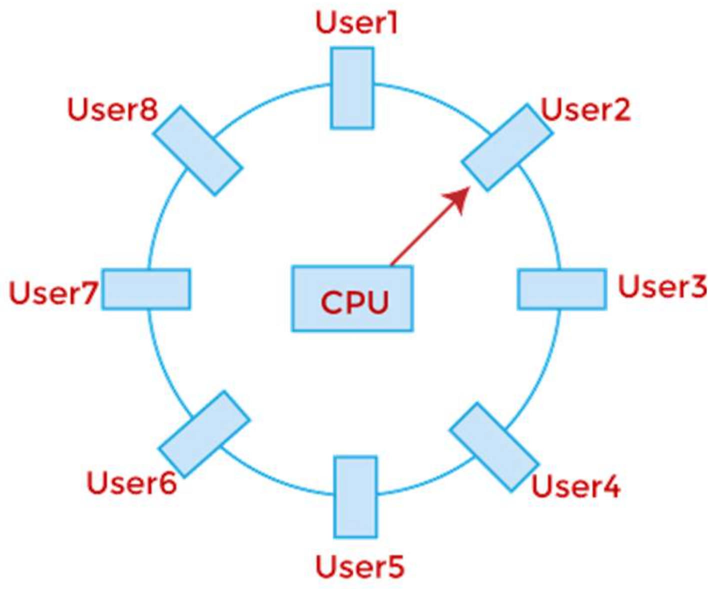
- Easy to layout, develop and execute real-time applications under the real-time operating system.
- In a Real-time operating system, the maximum utilization of devices and systems.

Disadvantages of Real-time operating system:

- Real-time operating systems are very costly to develop.
- Real-time operating systems are very complex and can consume critical CPU cycles.

Types of OS – Time Sharing Operating System

- In the Time Sharing operating system, computer resources are allocated in a time-dependent fashion to several programs simultaneously.
- Thus it helps to provide a large number of user's direct access to the main computer.
- It is a logical extension of multiprogramming.
- In time-sharing, the CPU is switched among multiple programs given by different users on a scheduled basis.
- A time-sharing operating system allows many users to be served simultaneously, so sophisticated CPU scheduling schemes and Input/output management are required.
- Time-sharing operating systems are very difficult and expensive to build.



Timesharing in case of 8 users

Types of OS – Time Sharing Operating System

Advantages of Time Sharing Operating System

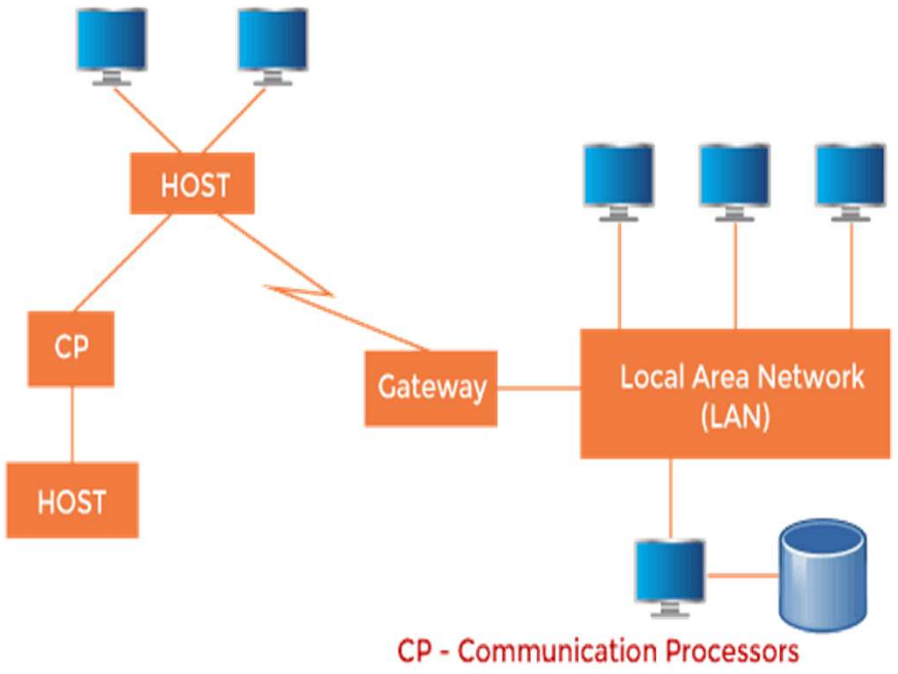
- The time-sharing operating system provides effective utilization and sharing of resources.
- This system reduces CPU idle and response time.

Disadvantages of Time Sharing Operating System

- Data transmission rates are very high in comparison to other methods.
- Security and integrity of user programs loaded in memory and data need to be maintained as many users access the system at the same time.

Types of OS – Distributed Operating System

- The Distributed Operating system is not installed on a single machine, it is divided into parts, and these parts are loaded on different machines.
- A part of the distributed Operating system is installed on each machine to make their communication possible.
- Distributed Operating systems are much more complex, large, and sophisticated than Network operating systems because they also have to take care of varying networking protocols.



A Typical View of a Distributed System

Types of OS – Distributed Operating System

Advantages of Distributed Operating System

- The distributed operating system provides sharing of resources.
- This type of system is fault-tolerant.

Disadvantages of Distributed Operating System

- Protocol overhead can dominate computation cost.

Operating System Concepts

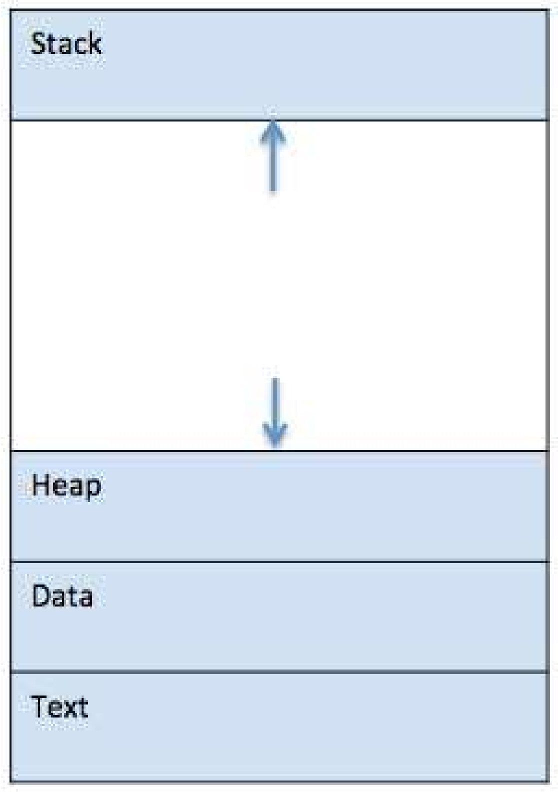
OS Concepts

- Process
  - Process Life Cycle
  - Process Control Block
- Address Space
- Input/Output
- Files

Process

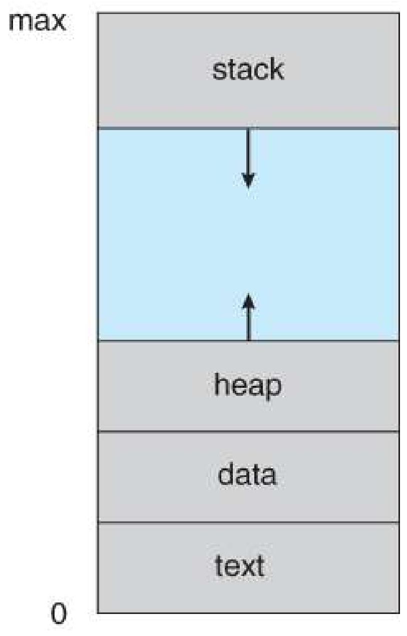
Process

- A process is basically a program in execution.
- The execution of a process must progress in a sequential fashion.
- A process is defined as an entity which represents the basic unit of work to be implemented in the system.
- To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.
- When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data.
- The image shows a simplified layout of a process inside main memory



Operating System Concepts – Process

S.N.	Component & Description
1	**Stack** The process Stack contains the temporary data such as method/function parameters, return address and local variables.
2	**Heap** This is dynamically allocated memory to a process during its run time.
3	**Text** This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
4	**Data** This section contains the global and static variables.

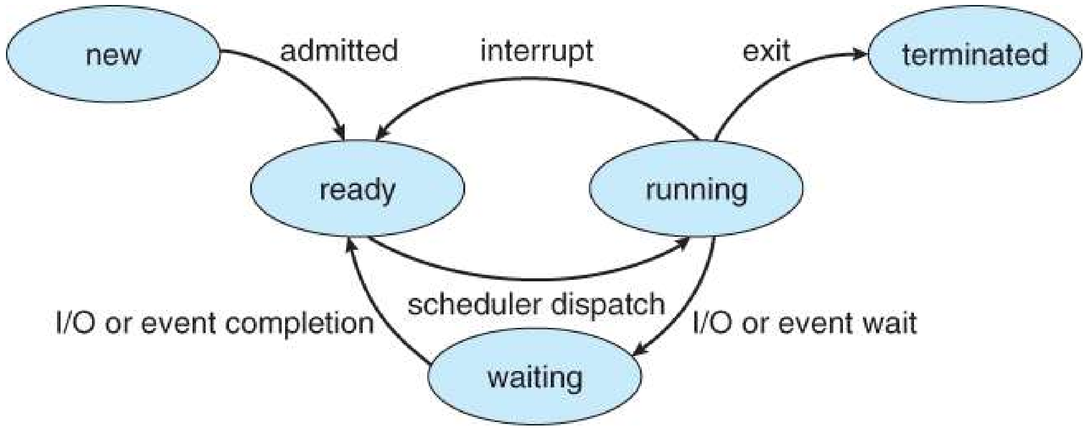


Process in Memory

Process Life Cycle or Process States

Processes may be in one of 5 states, as shown in image

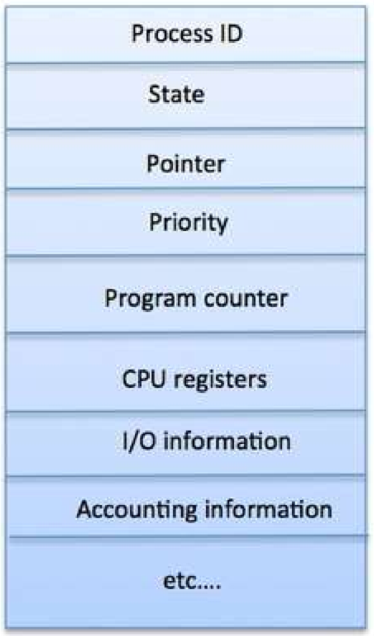
- **New** - The process is in the stage of being created.
- **Ready** - The process has all the resources available that it needs to run, but the CPU is not currently working on this process's instructions.
- **Running** - The CPU is working on this process's instructions.
- **Waiting** - The process cannot run at the moment, because it is waiting for some resource to become available or for some event to occur. For example the process may be waiting for keyboard input, disk access request, inter-process messages, a timer to go off, or a child process to finish.
- **Terminated -** The process has completed.



Process Control Block (PCB)

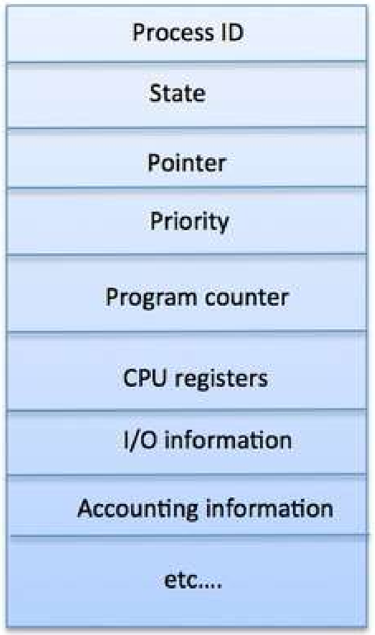
- A Process Control Block is a data structure maintained by the Operating System for every process.
- The PCB is identified by an integer process ID (PID).
- A PCB keeps all the information needed to keep track of a process as listed below in the table

S.N.	Information & Description
1	**Process State** The current state of the process i.e., whether it is ready, running, waiting, or whatever.
2	**Process privileges** This is required to allow/disallow access to system resources.
3	**Process ID** Unique identification for each of the process in the operating system.
4	**Pointer** A pointer to parent process.
5	**Program Counter** Program Counter is a pointer to the address of the next instruction to be executed for this process.
6	**CPU registers** Various CPU registers where process need to be stored for execution for running state.
7	**CPU Scheduling Information** Process priority and other scheduling information which is required to schedule the process.
8	**Memory management information** This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
9	**Accounting information** This includes the amount of CPU used for process execution, time limits, execution ID etc.
10	**IO status information** This includes a list of I/O devices allocated to the process.



Process Control Block (PCB)

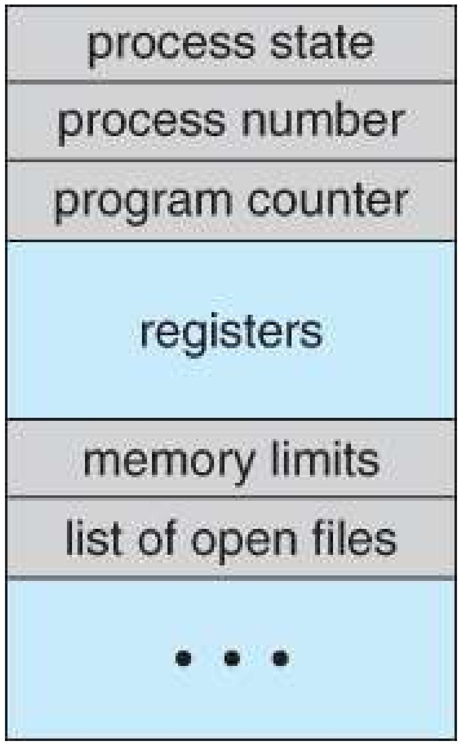
- The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems.
- Here is a simplified diagram of a PCB
- The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.



Process Control Block (PCB)

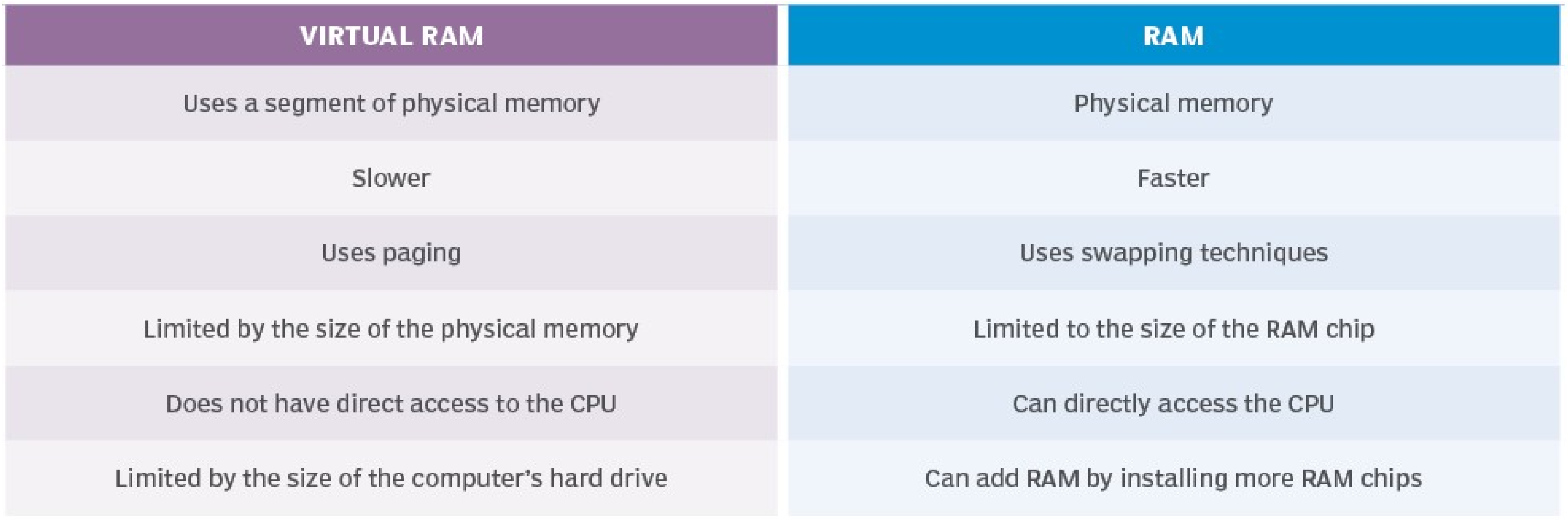
For each process there is a Process Control Block, PCB, which stores the following (types of) process-specific information, as illustrated in image (Specific details may vary from system to system.)

- **Process State** - Running, waiting, etc., as discussed in previous slide.
- **Process ID**, and parent process ID.
- **CPU registers and Program Counter** - These need to be saved and restored when swapping processes in and out of the CPU.
- **CPU-Scheduling information** - Such as priority information and pointers to scheduling queues.
- **Memory-Management information** - E.g. page tables or segment tables.
- **Accounting information** - user and kernel CPU time consumed, account numbers, limits, etc.
- **I/O Status information** - Devices allocated, open file tables, etc.



Address Space

- Address space is the amount of memory allocated for all possible addresses for a computational entity -- for example, a device, a file, a server or a networked computer.
- The system provides each device and process address space that holds a specific portion of the processor's address space.
- This can include either physical or virtual addresses accessible to a processor or reserved for a particular process.
- However, a memory management technique called virtual memory can increase the size of the address space to be higher than that of the physical memory.



VIRTUAL RAM	RAM
Uses a segment of physical memory	Physical memory
Slower	Faster
Uses paging	Uses swapping techniques
Limited by the size of the physical memory	Limited to the size of the RAM chip
Does not have direct access to the CPU	Can directly access the CPU
Limited by the size of the computer's hard drive	Can add RAM by installing more RAM chips

Address Space

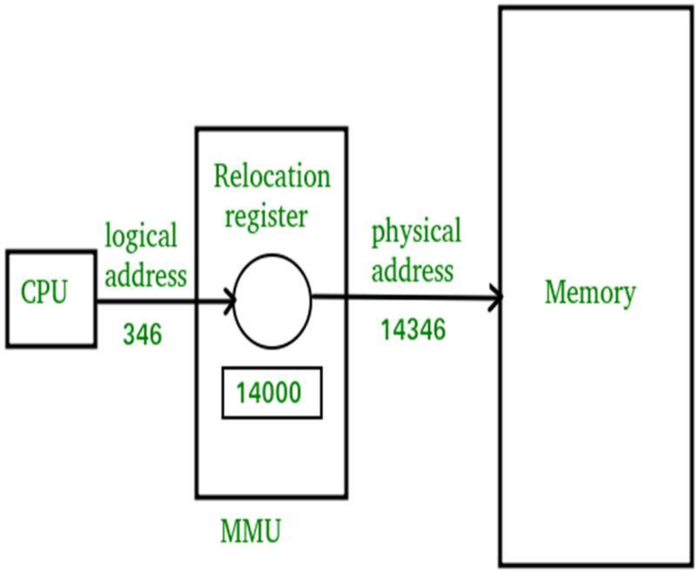
- Address space is a space in computer memory.
- Process Address Space means a space that is allocated in memory for a process.
- Every process has an address space
- Address Space can be of two types
  - Virtual Address Space (Logical Address Space)
  - Physical Address Space

Parameter	LOGICAL ADDRESS	PHYSICAL ADDRESS
Basic	Generated by CPU	location in a memory unit
Address Space	Logical Address Space is set of all logical addresses generated by CPU in reference to a program.	Physical Address is set of all physical addresses mapped to the corresponding logical addresses.
Visibility	User can view the logical address of a program.	User can never view physical address of program.
Generation	generated by the CPU	Computed by MMU
Access	The user can use the logical address to access the physical address.	The user can indirectly access physical address but not directly.
Editable	Logical address can be change.	Physical address will not change.
Also called	virtual address.	real address.

Address Space

- **Logical Address** is generated by CPU while a program is running.
- The logical address is virtual address as it does not exist physically, therefore, it is also known as Virtual Address.
- This address is used as a reference to access the physical memory location by CPU.
- The term Logical Address Space is used for the set of all logical addresses generated by a program's perspective.
- The hardware device called Memory-Management Unit is used for mapping logical address to its corresponding physical address

- **Physical Address** identifies a physical location of required data in a memory.
- The user never directly deals with the physical address but can access by its corresponding logical address.
- The user program generates the logical address and thinks that the program is running in this logical address but the program needs physical memory for its execution, therefore, the logical address must be mapped to the physical address by MMU before they are used.
- The term Physical Address Space is used for all physical addresses corresponding to the logical addresses in a Logical address space.



Input / Output

- The three major jobs of a computer are Input, Output, and Processing.
- In a lot of cases, the most important job is Input / Output, and the processing is simply incidental.
- For example, when you browse a web page or edit any file, our immediate attention is to read or enter some information, not for computing an answer.
- The primary role of the operating system in computer Input / Output is to manage and organize I/O operations and all I/O devices.
- The controlling of various devices that are connected to the computer is a key concern of operating-system designers.
- This is because I/O devices vary so widely in their functionality and speed (for example a mouse, a hard disk and a CD-ROM), varied methods are required for controlling them.
- These methods form the I/O sub-system of the kernel of OS that separates the rest of the kernel from the complications of managing I/O devices.

I/O Hardware

- Computers operate many huge kinds of devices.
- The general categories of storage devices are like disks, tapes, transmission devices (like network interface cards, and modems), and human interface devices (like screens, keyboards, etc.)
- A device that communicates with the operating system of a computer by transferring signals over cable or even through the air.
- The Peripheral devices that communicate with the machine through a connection point also called ports- (one example is a serial port).

Files in OS

- A file is a collection of related information that is recorded on secondary storage. Or file is a collection of logically related entities.
- From user's perspective a file is the smallest allotment of logical secondary storage.
- **The name of the file is divided into two parts as shown below:**
  - name
  - extension, separated by a period.

Files attributes and their operations:

Attributes	Types	Operations
Name	Doc	Create
Type	Exe	Open
Size	Jpg	Read
Creation Data	Xis	Write
Author	C	Append
Protection	Class	Delete
Last Modified	Java	Truncate

Files in OS

File Structure

- A File Structure should be according to a required format that the operating system can understand.
- A file has a certain defined structure according to its type.
- A text file is a sequence of characters organized into lines.
- A source file is a sequence of procedures and functions.
- An object file is a sequence of bytes organized into blocks that are understandable by the machine.
- When operating system defines different file structures, it also contains the code to support these file structure.
- Unix, MS-DOS support minimum number of file structure.

File Type

- File type refers to the ability of the operating system to distinguish different types of file such as text files source files and binary files etc.
- Many operating systems support many types of files.

Files in OS

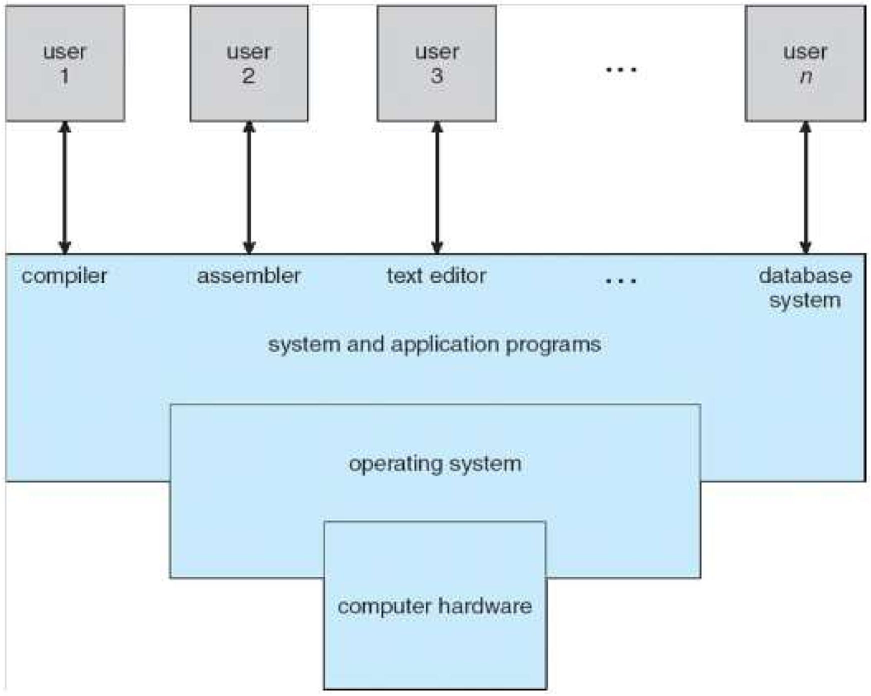
File type	Usual extension	Function
Executable	exe, com, bin	Read to run machine language program
Object	obj, o	Compiled, machine language not linked
Source Code	C, java, pas, asm, a	Source code in various languages
Batch	bat, sh	Commands to the command interpreter
Text	txt, doc	Textual data, documents
Word Processor	wp, tex, rrf, doc	Various word processor formats
Archive	arc, zip, tar	Related files grouped into one compressed file
Multimedia	mpeg, mov, rm	For containing audio/video information
Markup	xml, html, tex	It is the textual data and documents
Library	lib, a ,so, dll	It contains libraries of routines for programmers
Print or View	gif, pdf, jpg	It is a format for printing or viewing a ASCII or binary file.

Computer System Organization and Architecture

Computer System Structure

Computer system can be divided into four components:

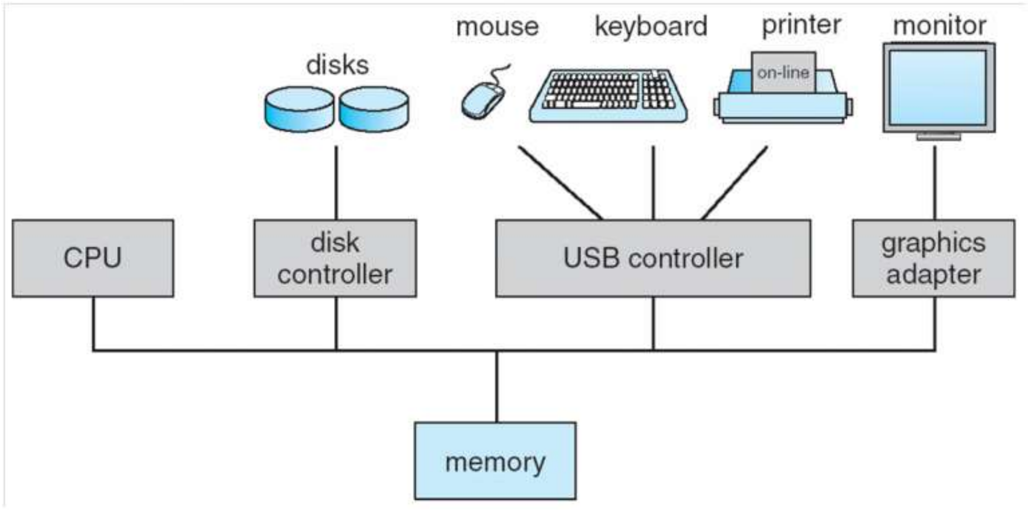
- **Hardware** – provides basic computing resources
  - CPU, memory, I/O devices
- **Operating system**
  - Controls and coordinates use of hardware among various applications and users
- **Application programs** – define the ways in which the system resources are used to solve the computing problems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



Four Components of a Computer System

Computer System Organization and Architecture

- Computer-system operation
  - One or more CPUs, and device controllers connect through a common bus providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles
  - I/O devices and the CPU can execute concurrently
  - Each device controller is in charge of a particular device type
  - Each device controller has a local buffer
  - CPU moves data from/to main memory to/from local buffers
  - I/O is from the device to local buffer of controller
  - Device controller informs CPU that it has finished its operation by causing an interrupt

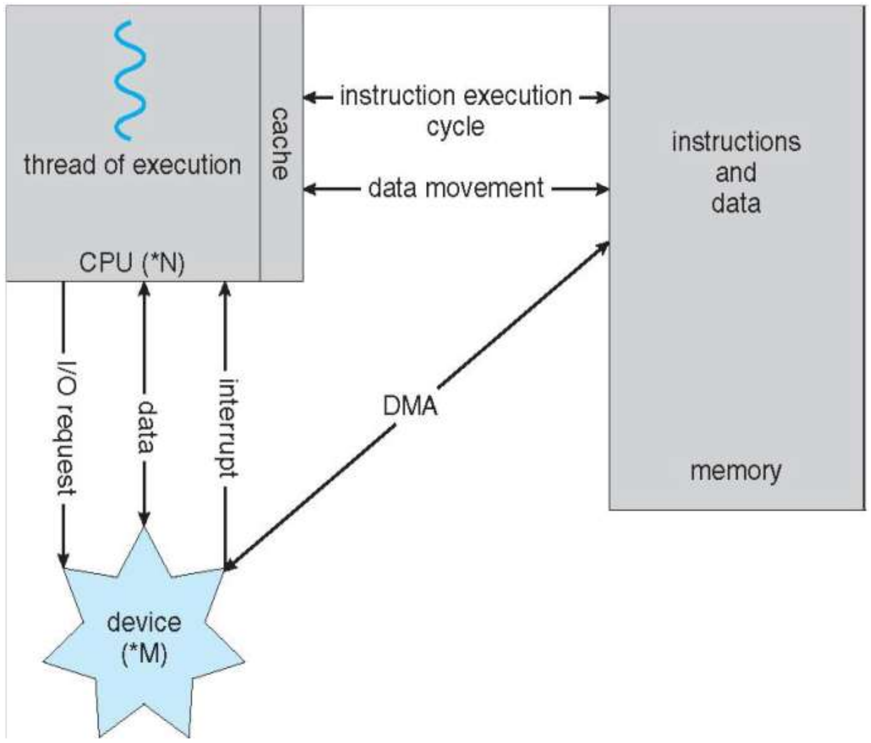


Computer System Organization and Architecture

Computer System Architecture

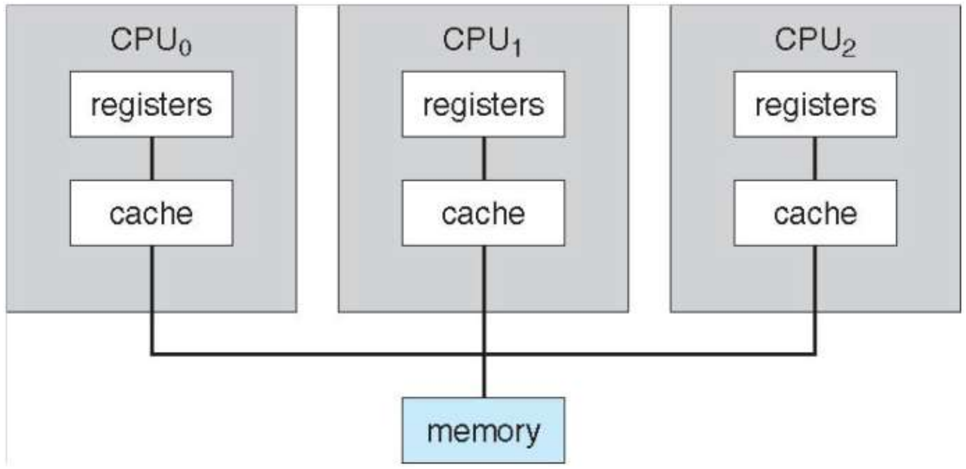
- Most systems use a single general-purpose processor (PDAs through mainframes)
  - Most systems have special-purpose processors as well
- Multiprocessors systems growing in use and importance
  - Also known as parallel systems, tightly-coupled systems
  - Advantages include:
    - Increased throughput
    - Economy of scale
    - Increased reliability – graceful degradation or fault tolerance
- Two types:
  - Asymmetric Multiprocessing
  - Symmetric Multiprocessing

How Modern Computer Works

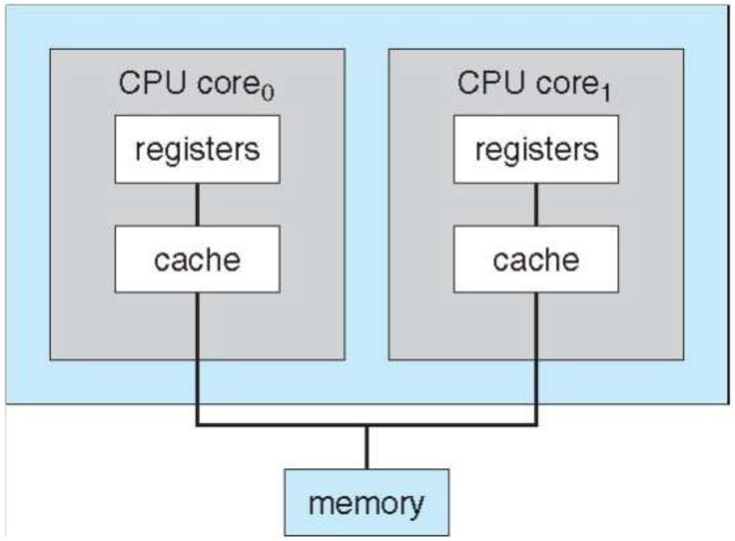


A von Neumann architecture

Computer System Organization and Architecture



Symmetric Multi-Processing Architecture



Dual Core Design

Resource Management

- An operating system is a resource manager.
- The system's CPU, memory space, file-storage space, and I/O devices are among the resources that the operating system must manage.
- **Resource Management**
  - Process Management
  - Memory Management
  - Storage Management

Resource Management – Process Management

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

Resource Management – Process Management

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

Resource Management – Memory Management

Memory Management

- All data in memory before and after processing
- All instructions in memory in order to execute
- Memory management determines what is in memory when
  - Optimizing CPU utilization and computer response to users
- Memory management activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed

Resource Management – Storage Management

Storage Management

- OS provides uniform, logical view of information storage
    - Abstracts physical properties to logical storage unit - file
    - Each medium is controlled by device (i.e., disk drive, tape drive)
        - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)

File-System management

- Files usually organized into directories
- Access control on most systems to determine who can access what
- OS activities include
    - Creating and deleting files and directories
    - Primitives to manipulate files and dirs
    - Mapping files onto secondary storage
    - Backup files onto stable (non-volatile) storage media

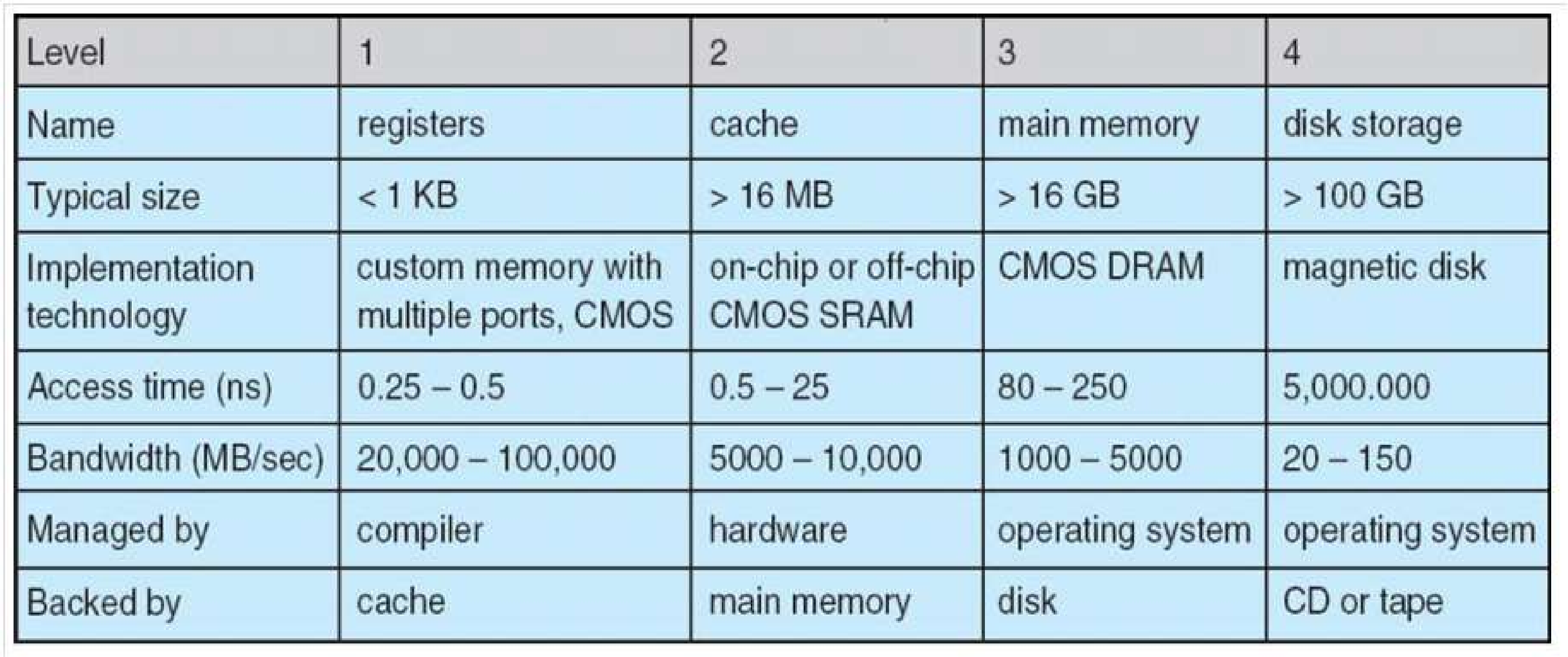
Resource Management – Mass Storage Management

Mass Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
  - Free-space management
  - Storage allocation
  - Disk scheduling
- Some storage need not be fast
  - Tertiary storage includes optical storage, magnetic tape
  - Still must be managed – by OS or applications
  - Varies between WORM (write-once, read-many-times) and RW (read-write)

Resource Management – Performance of level of Storage

Movement between levels of storage hierarchy can be explicit or implicit



Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

Performance of Various levels of Storage

Protection and Security

Protection and Security

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
- **Security** – defence of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
  - User identities (user IDs, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (group ID) allows set of users to be defined and controls managed, then also associated with each process, file
  - Privilege escalation allows user to change to effective ID with more rights

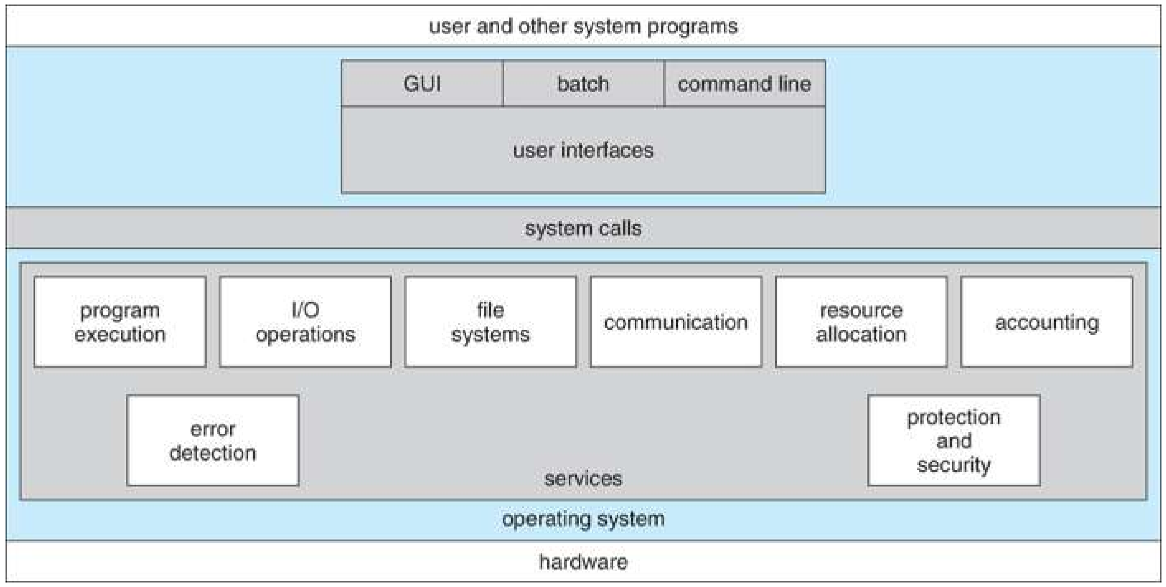
Protection and Security

- 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 the 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. That user ID is associated with all of the user's processes and threads.
- When an ID needs to be readable by a user, it is translated back to the user name via the user name list.
- In some circumstances, we wish to distinguish among sets of users rather than individual users.
- For example, the owner of a file on a UNIX system may be allowed to issue all operations on that file, whereas a selected set of users may be allowed only to read the file.

Operating System Services

OS provides environments in which programs run, and services for the users of the system, including:

- User Interfaces
- Program Execution
- I/O Operations
- File-System Manipulation
- Communications
- Error Detection



A view of OS Services

Operating System Services

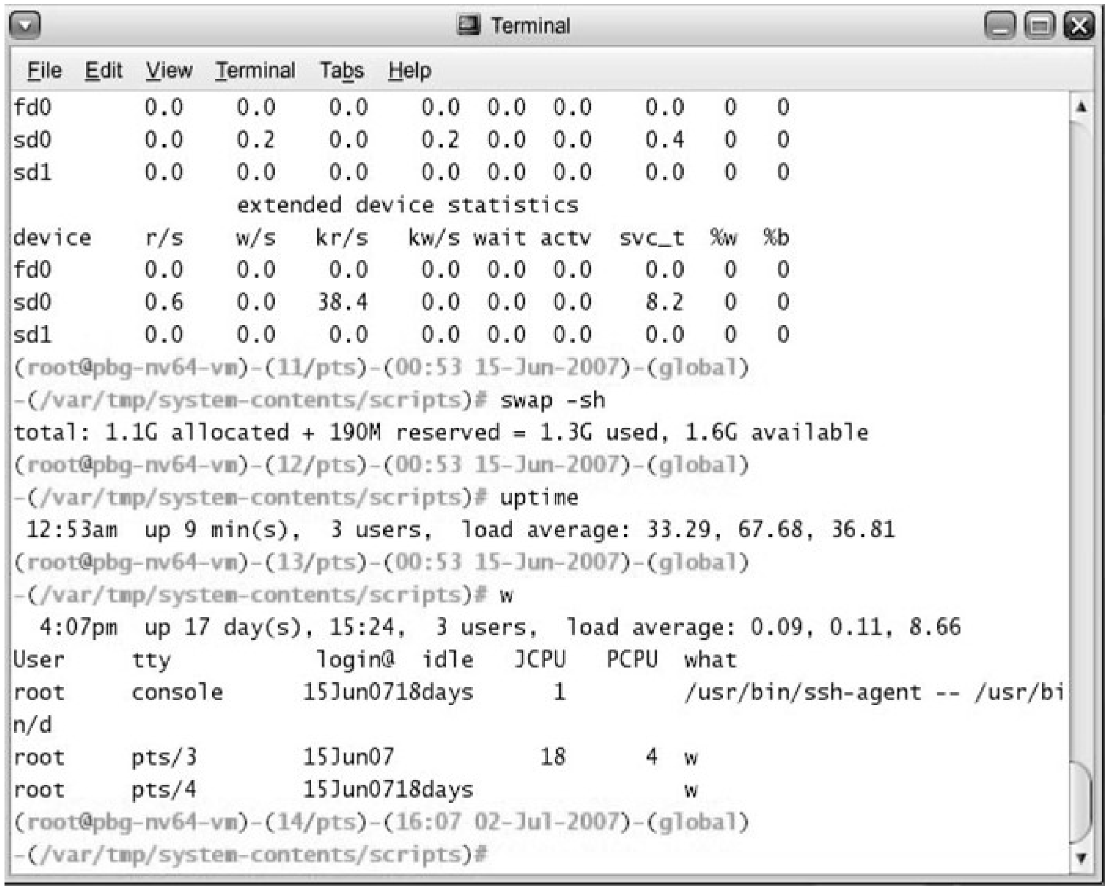
OS provides environments in which programs run, and services for the users of the system, including:

- **User Interfaces** - Means by which users can issue commands to the system. Depending on the system these may be a command-line interface (e.g. sh, csh, ksh, tcsh, etc.), a GUI interface (e.g. Windows, X-Windows, KDE, Gnome, etc.), or a batch command systems. The latter are generally older systems using punch cards of job-control language, JCL, but may still be used today for specialty systems designed for a single purpose.
- **Program Execution** - The OS must be able to load a program into RAM, run the program, and terminate the program, either normally or abnormally.
- **I/O Operations** - The OS is responsible for transferring data to and from I/O devices, including keyboards, terminals, printers, and storage devices.
- **File-System Manipulation** - In addition to raw data storage, the OS is also responsible for maintaining directory and subdirectory structures, mapping file names to specific blocks of data storage, and providing tools for navigating and utilizing the file system.
- **Communications** - Inter-process communications, IPC, either between processes running on the same processor, or between processes running on separate processors or separate machines. May be implemented as either shared memory or message passing, (or some systems may offer both.)
- **Error Detection** - Both hardware and software errors must be detected and handled appropriately, with a minimum of harmful repercussions. Some systems may include complex error avoidance or recovery systems, including backups, RAID drives, and other redundant systems. Debugging and diagnostic tools aid users and administrators in tracing down the cause of problems.

User and OS Interface

Command Interpreter

- Gets and processes the next user request, and launches the requested programs.
- In some systems the CI may be incorporated directly into the kernel.
- More commonly the CI is a separate program that launches once the user logs in or otherwise accesses the system.
- UNIX, for example, provides the user with a choice of different shells, which may either be configured to launch automatically at login, or which may be changed on the fly.
- (Each of these shells uses a different configuration file of initial settings and commands that are executed upon startup.)



```
fd0       0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd0       0.0    0.2    0.0    0.2  0.0  0.0    0.4   0   0
sd1       0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
                 extended device statistics
device    r/s    w/s   kr/s   kw/s wait actv  svc_t  %w  %b
fd0       0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd0       0.6    0.0   38.4    0.0  0.0  0.0    8.2   0   0
sd1       0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
(root@pbg-nv64-vm)-(11/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# swap -sh
total: 1.1G allocated + 190M reserved = 1.3G used, 1.6G available
(root@pbg-nv64-vm)-(12/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# uptime
 12:53am  up 9 min(s),  3 users,  load average: 33.29, 67.68, 36.81
(root@pbg-nv64-vm)-(13/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# w
  4:07pm  up 17 day(s), 15:24,  3 users,  load average: 0.09, 0.11, 8.66
User     tty           login@  idle   JCPU   PCPU  what
root     console      15Jun0718days      1         /usr/bin/ssh-agent -- /usr/bi
n/d
root     pts/3        15Jun07            18      4  w
root     pts/4        15Jun0718days                 w
(root@pbg-nv64-vm)-(14/pts)-(16:07 02-Jul-2007)-(global)
-(/var/tmp/system-contents/scripts)#
```

User and OS Interface

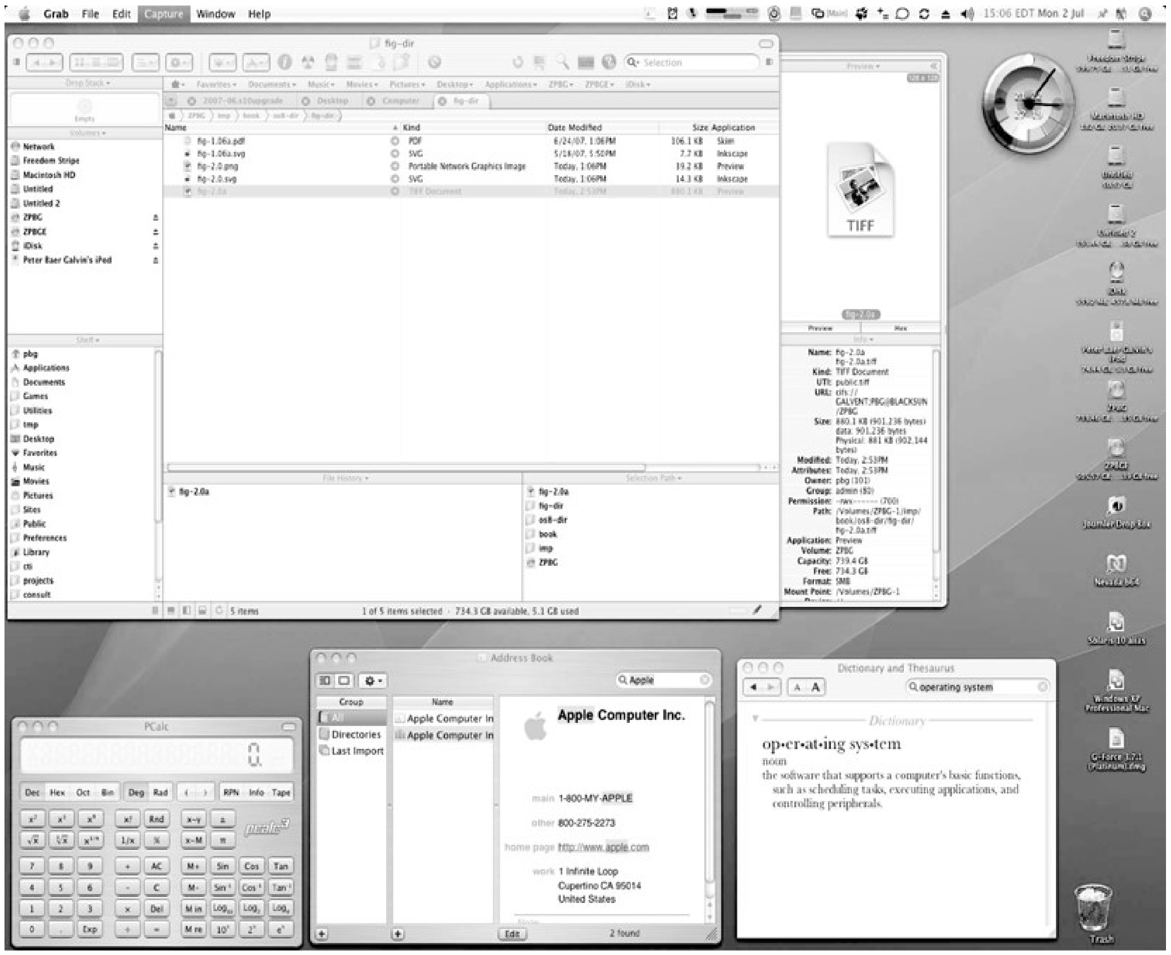
Graphical User Interface – GUI

- Generally implemented as a desktop metaphor, with file folders, trash cans, and resource icons.
- Icons represent some item on the system, and respond accordingly when the icon is activated.
- First developed in the early 1970's at Xerox PARC research facility.
- In some systems the GUI is just a front end for activating a traditional command line interpreter running in the background. In others the GUI is a true graphical shell in its own right.
- Mac has traditionally provided ONLY the GUI interface. With the advent of OSX (based partially on UNIX), a command line interface has also become available.

User and OS Interface

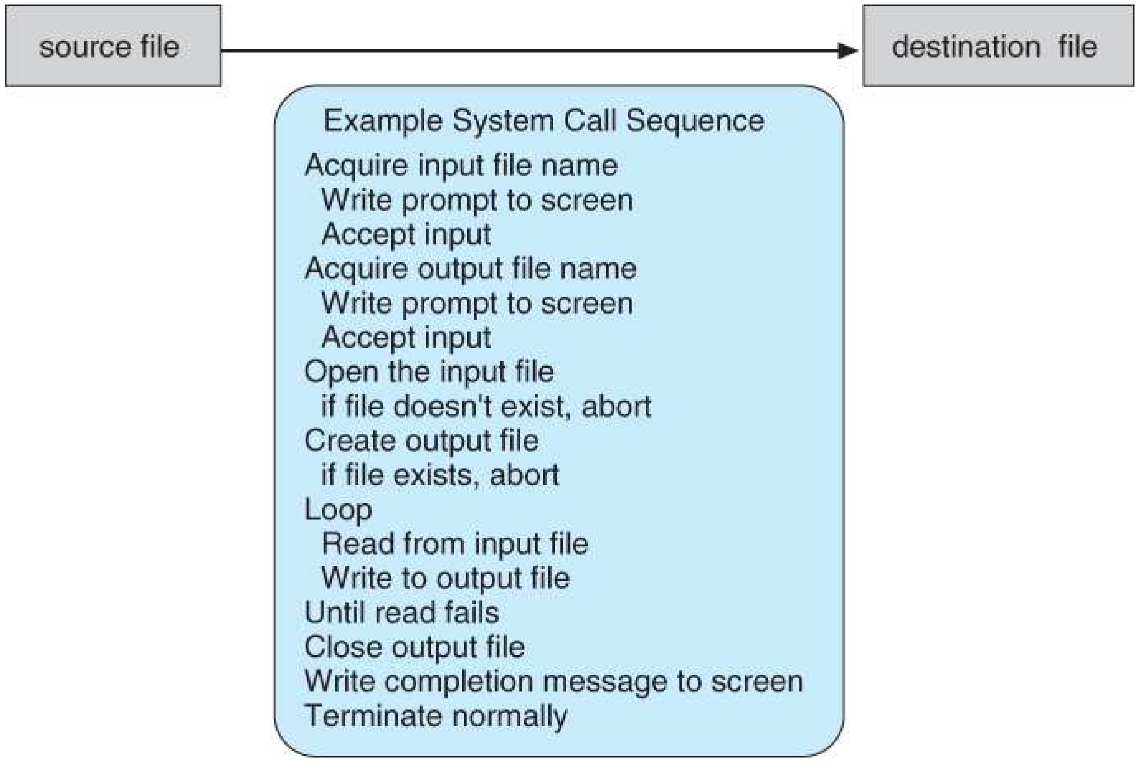
Choice of Interface

- Most modern systems allow individual users to select their desired interface, and to customize its operation, as well as the ability to switch between different interfaces as needed.
- System administrators generally determine which interface a user starts with when they first log in.
- GUI interfaces usually provide an option for a terminal emulator window for entering command-line commands.
- Command-line commands can also be entered into **shell scripts**, which can then be run like any other programs.


Grab File Edit Capture Window Help
fig-dir
PCalc
Address Book
Apple Computer Inc.
Dictionary and Thesaurus
op·er·at·ing sys·tem
noun
the software that supports a computer's basic functions, such as scheduling tasks, executing applications, and controlling peripherals.
TIFF
Trash

System Calls

- System calls provide a means for user or application programs to call upon the services of the operating system.
- Generally written in C or C++, although some are written in assembly for optimal performance.
- Figure illustrates the sequence of system calls required to copy a file:
- You can use "strace" to see more examples of the large number of system calls invoked by a single simple command.
- Read the man page for strace, and try some simple examples. (strace mkdir temp, strace cd temp, strace date > t.t, strace cp t.t t.2, etc.)

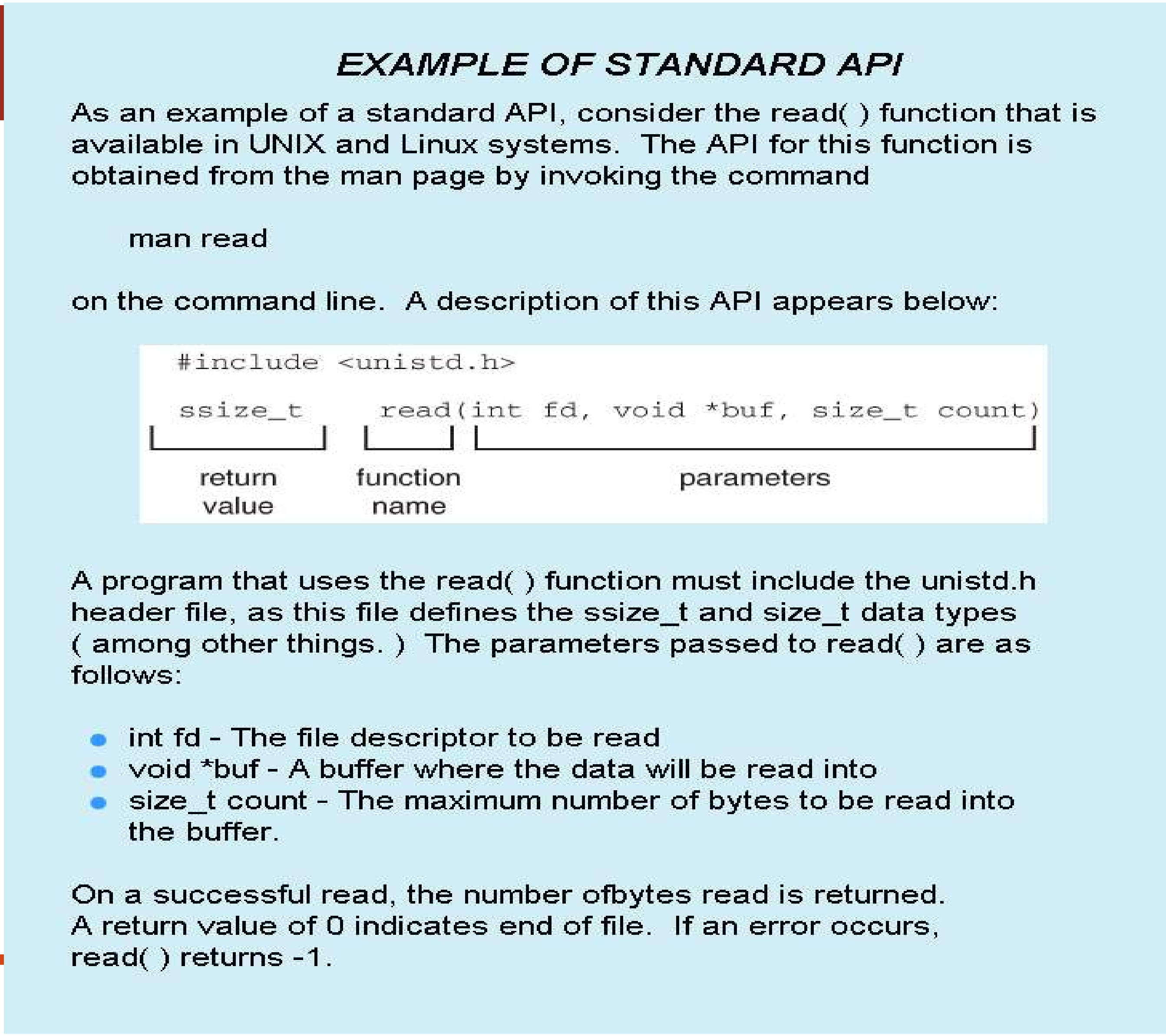


```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

Example of how system calls are used

System Calls

- Most programmers do not use the low-level system calls directly, but instead use an "Application Programming Interface", API.
- The following sidebar shows the read( ) call available in the API on UNIX based systems::



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)
```

$$\underbrace{\texttt{ssize\_t}}_{\text{return value}}\quad\underbrace{\texttt{read}}_{\text{function name}}\underbrace{\texttt{(int fd, void *buf, size\_t count)}}_{\text{parameters}}$$

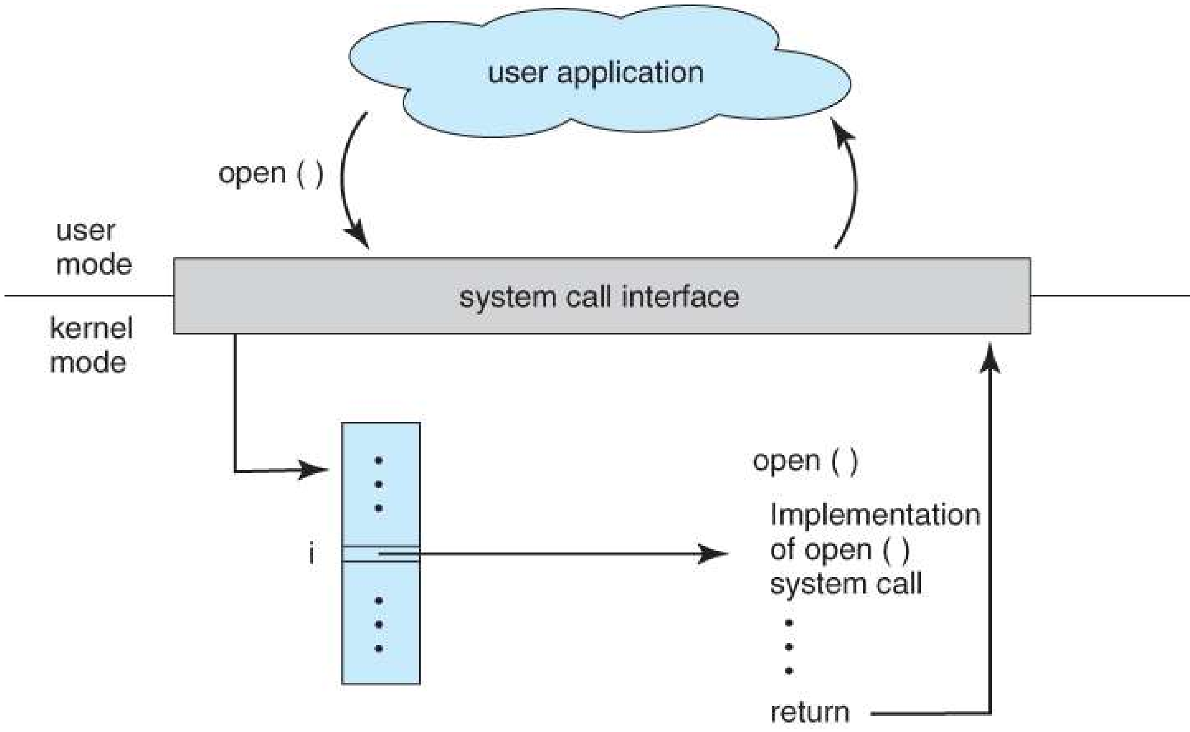
A program that uses the read( ) function must include the unistd.h header file, as this file defines the ssize_t and size_t data types (among other things.) The parameters passed to read( ) are as follows:

- int fd - The file descriptor to be read
- void *buf - A buffer where the data will be read into
- size_t count - The maximum number of bytes to be read into the buffer.

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

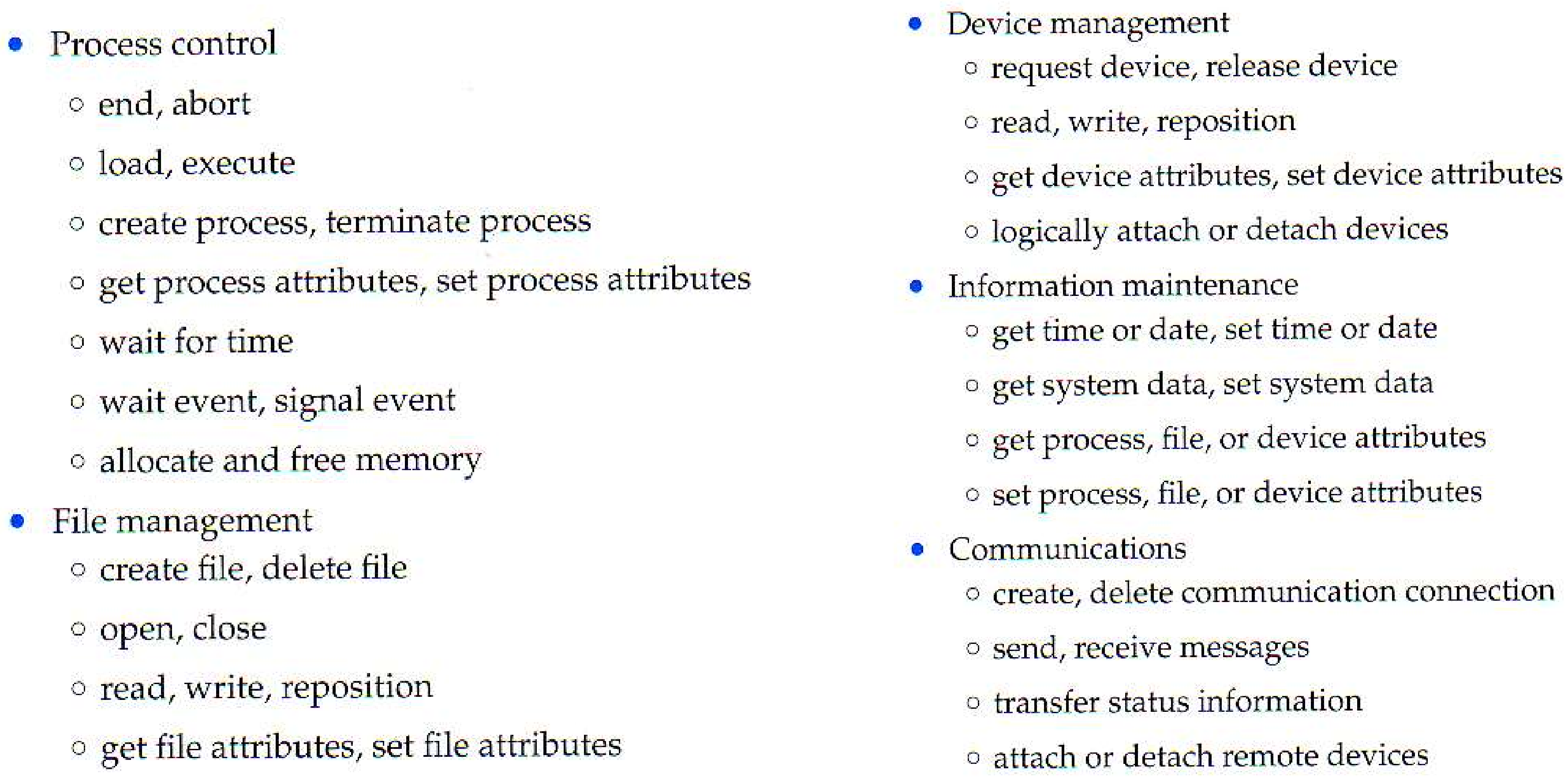
System Calls

- The use of APIs instead of direct system calls provides for greater program portability between different systems.
- The API then makes the appropriate system calls through the system call interface, using a table lookup to access specific numbered system calls, as shown in Figure:



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

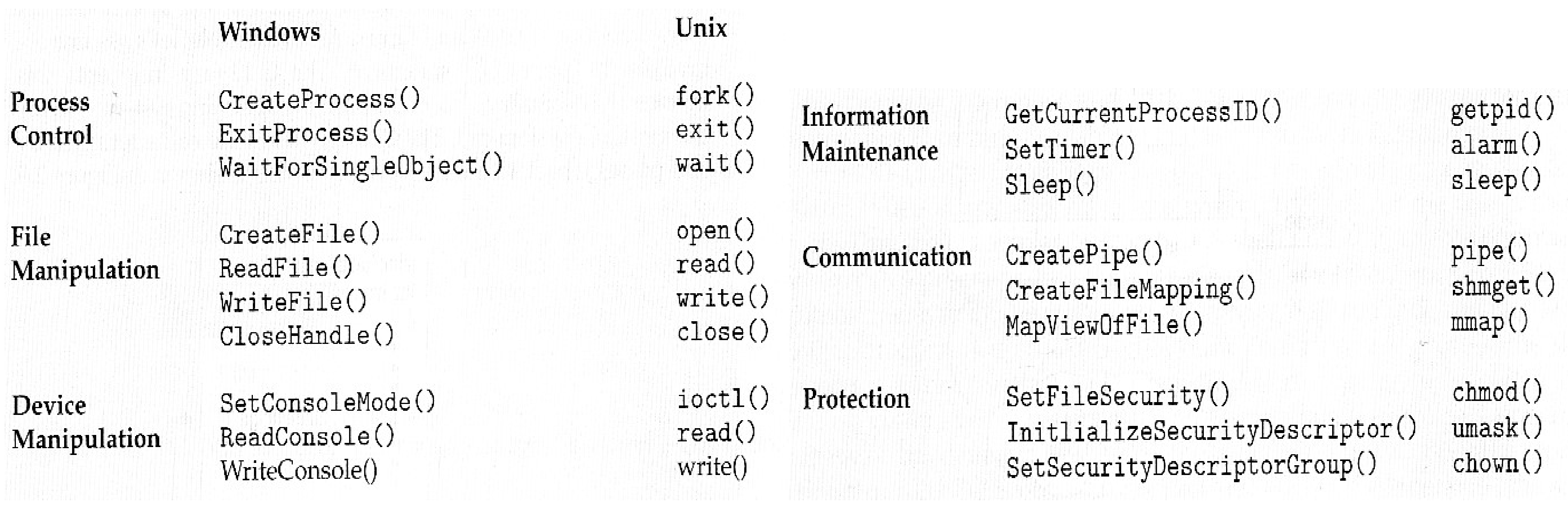
Types of System Calls



- Process control
  - end, abort
  - load, execute
  - create process, terminate process
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
- File management
  - create file, delete file
  - open, close
  - read, write, reposition
  - get file attributes, set file attributes
- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- 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
- Communications
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach or detach remote devices

Types of System Calls

Types of System Calls



	Windows	Unix

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

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

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

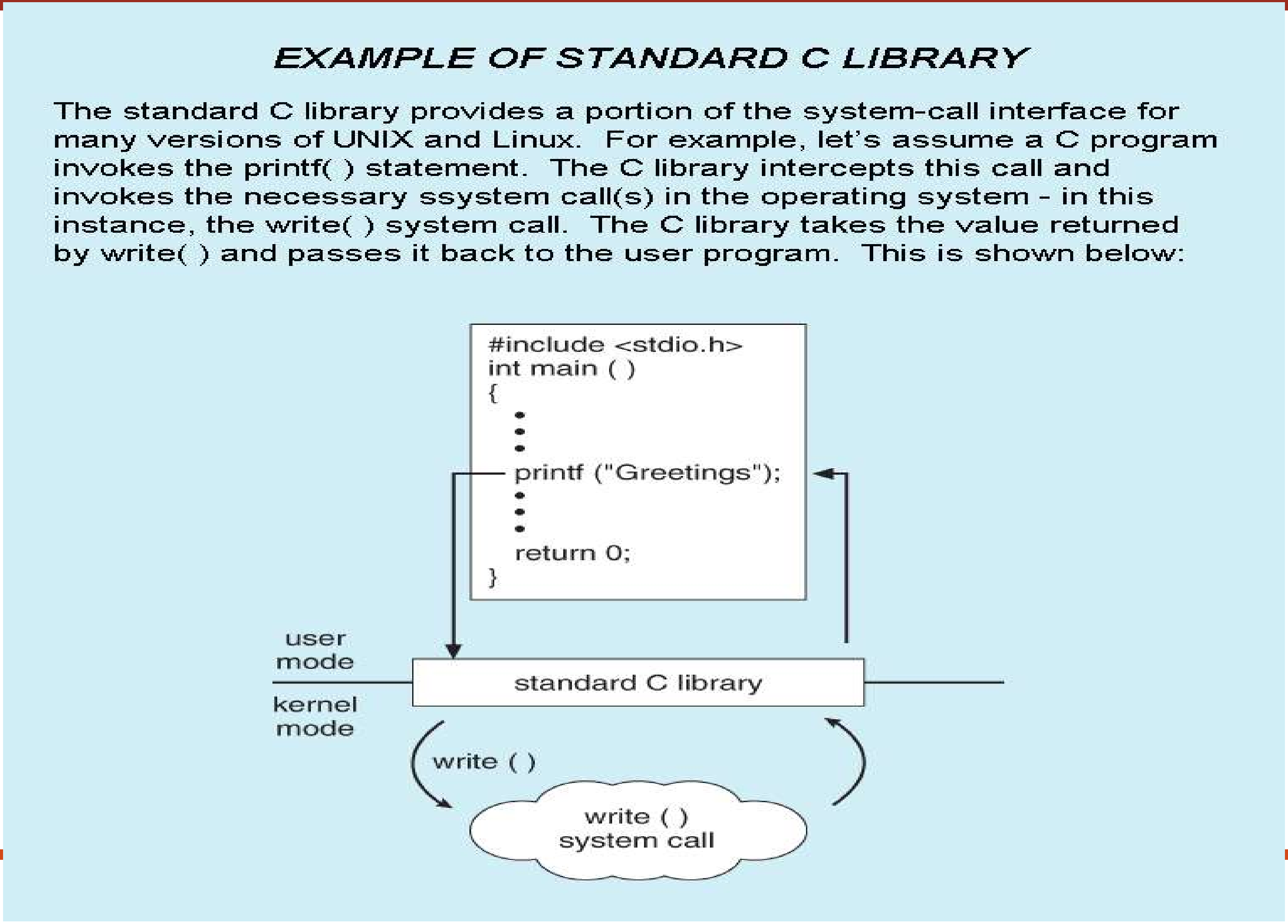
Information Maintenance	GetCurrentProcessID()
	SetTimer()
	Sleep()
		getpid()
		alarm()
		sleep()

Communication	CreatePipe()
	CreateFileMapping()
	MapViewOfFile()
		pipe()
		shmget()
		mmap()

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

Types of System Calls

System Calls – Example of Standard C Library



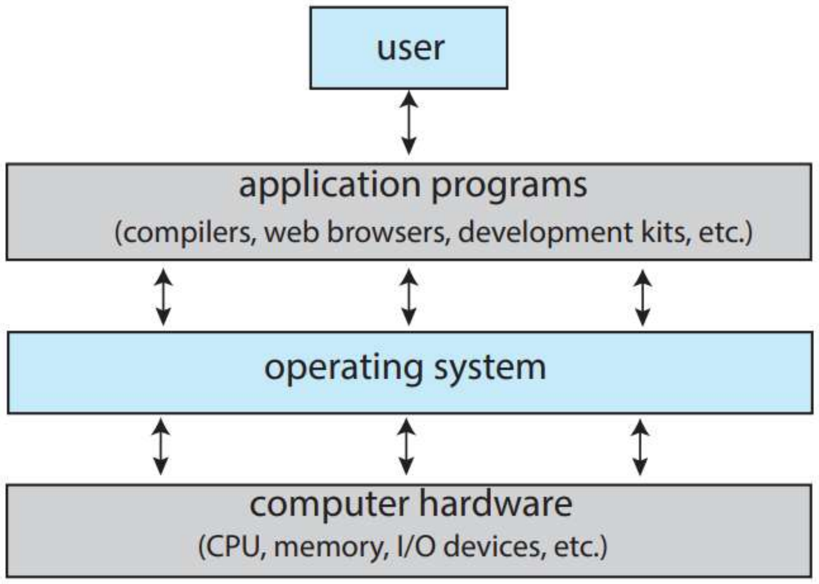
EXAMPLE OF STANDARD C LIBRARY

The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. For example, let's assume a C program invokes the printf( ) statement. The C library intercepts this call and invokes the necessary ssystem call(s) 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. This is shown below:

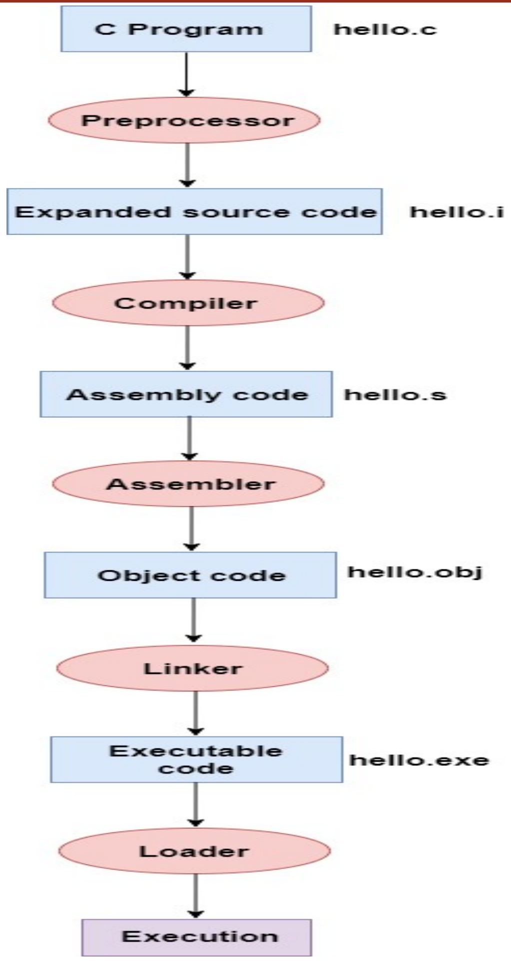
```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return 0;
}
```

System Services

- Another aspect of a modern system is its collection of system services.
- See the Figure, which depicted the logical computer hierarchy.
- At the lowest level is hardware.
- Next is the operating system, then the system services, and finally the application programs.
- System services, also known as system utilities, provide a convenient environment for program development and execution.
- Some of them are simply user interfaces to system calls.
- Others are considerably more complex.
- They can be divided into these categories:
    - File Management
    - Status Information
    - File Modification
    - Program Loading and Execution
    - Programming Language Support
    - Communication
    - Background Services


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

Linkers and Loaders

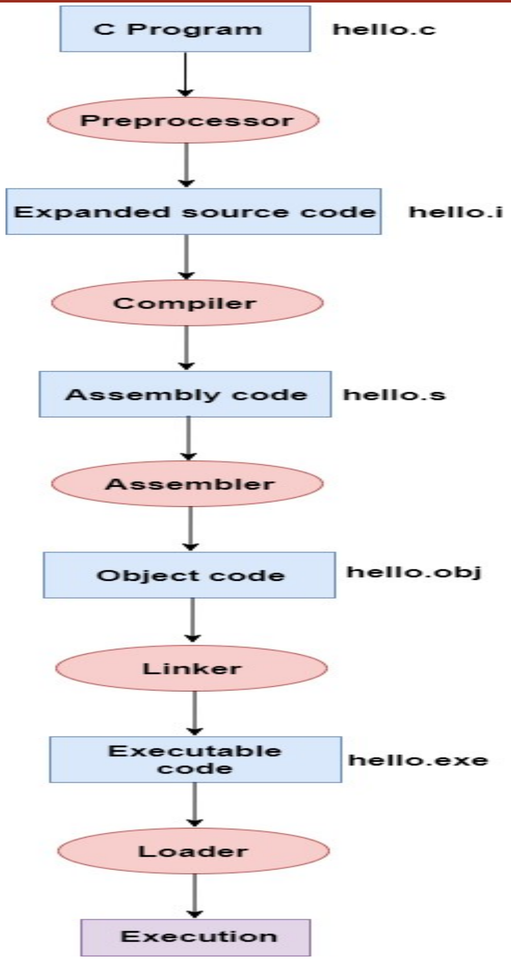


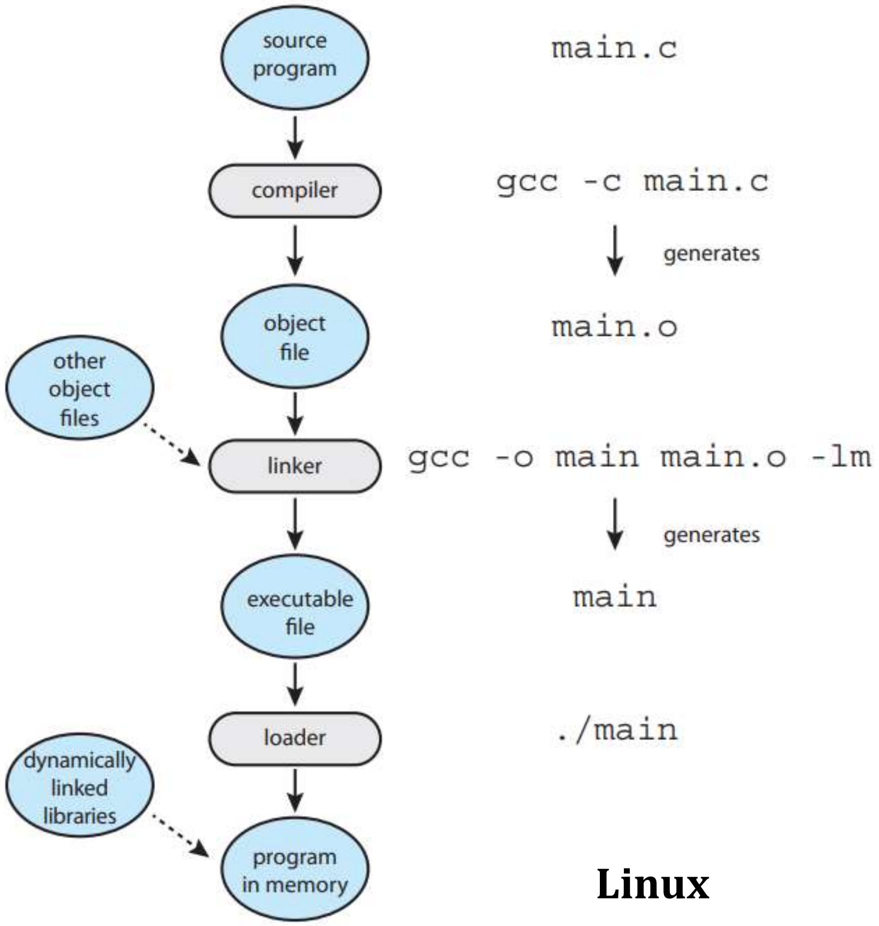
Windows

- Firstly, the input file, i.e., **hello.c,** is passed to the preprocessor, and the preprocessor converts the source code into expanded source code.
- The extension of the expanded source code would be **hello.i.**
- The expanded source code is passed to the compiler, and the compiler converts this expanded source code into assembly code.
- The extension of the assembly code would be **hello.s**
- This assembly code is then sent to the assembler, which converts the assembly code into object code.
- After the creation of an object code, the linker creates the executable file.
- The loader will then load the executable file for the execution.

Linkers and Loaders

Windows





Linux

Linkers and Loaders

- Usually, a program resides on disk as a binary executable file— for example, a.out or prog.exe.
- To run on a CPU, the program must be brought into memory and placed in the context of a process.
- Source files are compiled into object files that are designed to be loaded into any physical memory location, a format known as a relocatable object file.
- Next, the linker combines these relocatable object files into a single binary executable file.
- During the linking phase, other object files or libraries may be included as well, such as the standard C or math library (specified with the flag -lm).
- A loader is used to load the binary executable file into memory, where it is eligible to run on a CPU core.
- An activity associated with linking and loading is relocation, which assigns final addresses to the program parts and adjusts code and data in the program to match those addresses so that, for example, the code can call library functions and access its variables as it executes.

OS Design and Implementation

Design Goals

- The first problem in designing a system is to define goals and specifications.
- At the highest level, the design of the system will be affected by the choice of hardware and the type of system: traditional desktop/laptop, mobile, distributed, or real time.
- Beyond this highest design level, the requirements may be much harder to specify.
- The requirements can, however, be divided into two basic groups: user goals and system goals.
- Users want certain obvious properties in a system.
- The system should be convenient to use, easy to learn and to use, reliable, safe, and fast.
- Of course, these specifications are not particularly useful in the system design, since there is no general agreement on how to achieve them
- A similar set of requirements can be defined by the developers who must design, create, maintain, and operate the system.
- The system should be easy to design, implement, and maintain; and it should be flexible, reliable, error-free, and efficient.
- Again, these requirements are vague and may be interpreted in various ways.

OS Design and Implementation

Mechanisms and Policies

- One important principle is the separation of policy from mechanism.
- Mechanisms determine how to do something; policies determine what will be done.
- For example, the timer construct is a mechanism for ensuring CPU protection, but deciding how long the timer is to be set for a particular user is a policy decision.
- The separation of policy and mechanism is important for flexibility.
- Policies are likely to change across places or over time.
- In the worst case, each change in policy would require a change in the underlying mechanism.
- A general mechanism flexible enough to work across a range of policies is preferable.
- A change in policy would then require redefinition of only certain parameters of the system.
- For instance, consider a mechanism for giving priority to certain types of programs over others.
- If the mechanism is properly separated from policy, it can be used either to support a policy decision that I/O-intensive programs should have priority over CPU-intensive ones or to support the opposite policy.
- Policy decisions are important for all resource allocation. Whenever it is necessary to decide whether or not to allocate a resource, a policy decision must be made. Whenever the question is how rather than what, it is a mechanism that must be determined

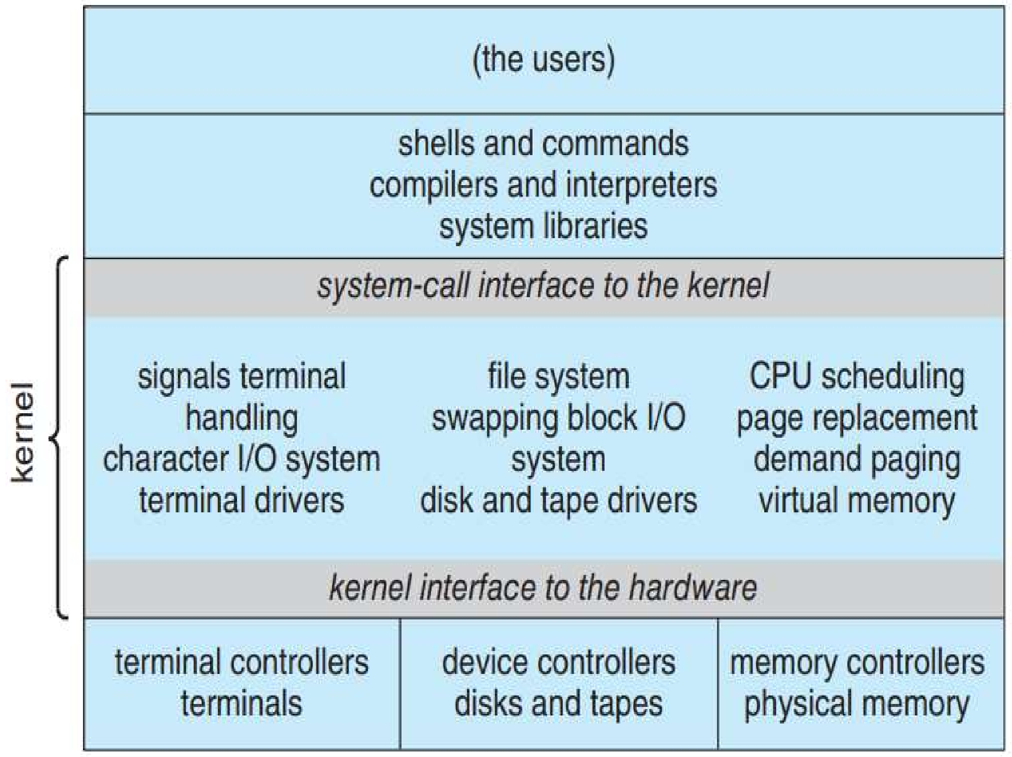
OS Design and Implementation

Implementation

- Once an operating system is designed, it must be implemented.
- Because operating systems are collections of many programs, written by many people over a long period of time, it is difficult to make general statements about how they are implemented.
- Early operating systems were written in assembly language.
- Now, most are written in higher-level languages such as C or C++, with small amounts of the system written in assembly language.
- In fact, more than one higher-level language is often used.
- The lowest levels of the kernel might be written in assembly language and C.
- Higher-level routines might be written in C and C++, and system libraries might be written in C++ or even higher-level languages.
- Android provides a nice example: its kernel is written mostly in C with some assembly language.
- Most Android system libraries are written in C or C++
- The only possible disadvantages of implementing an operating system in a higher-level language are reduced speed and increased storage requirements. This, however, is not a major issue in today's systems.

OS Structure

- A system as large and complex as a modern operating system must be engineered carefully if it is to function properly and be modified easily.
- A common approach is to partition the task into small components, or modules, rather than have one single system.
- Each of these modules should be a well-defined portion of the system, with carefully defined interfaces and functions.
- You may use a similar approach when you structure your programs: rather than placing all of your code in the main() function, you instead separate logic into a number of functions, clearly articulate parameters and return values, and then call those functions from main().



Traditional Unix Structure

OS Structure – Monolithic Structure

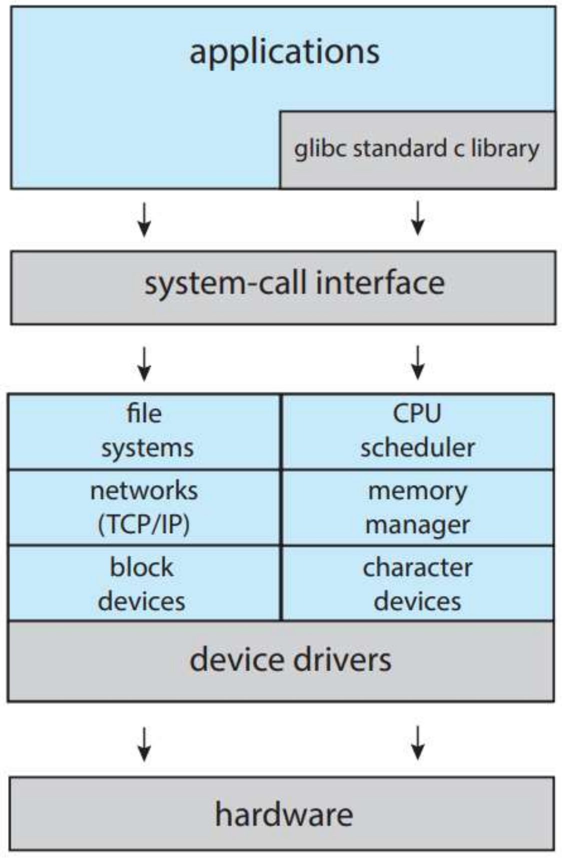
Monolithic Structure

- The simplest structure for organizing an operating system is no structure at all.
- That is, place all of the functionality of the kernel into a single, static binary file that runs in a single address space.
- This approach—known as a monolithic structure—is a common technique for designing operating systems.
- An example of such limited structuring is the original UNIX operating system, which consists of two separable parts: the kernel and the system programs.
- The kernel is further separated into a series of interfaces and device drivers, which have been added and expanded over the years as UNIX has evolved.
- We can view the traditional UNIX operating system as being layered to some extent, as shown in Figure on previous slide.
- Everything below the system-call interface and above the physical hardware is the kernel.
- The kernel provides the file system, CPU scheduling, memory management, and other operating system functions through system calls.
- Taken in sum, that is an enormous amount of functionality to be combined into one single address space

OS Structure – Monolithic Structure

Monolithic Structure

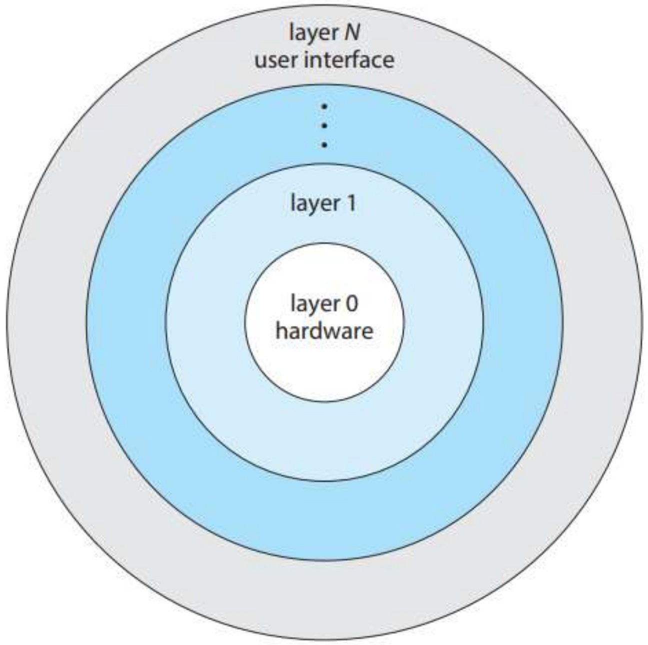
- The Linux operating system is based on UNIX and is structured similarly, as shown in this Figure.
- Applications typically use the glibc standard C library when communicating with the system call interface to the kernel.
- The Linux kernel is monolithic in that it runs entirely in kernel mode in a single address space, it does have a modular design that allows the kernel to be modified during run time.
- Despite the apparent simplicity of monolithic kernels, they are difficult to implement and extend.
- Monolithic kernels do have a distinct performance advantage, however: there is very little overhead in the system-call interface, and communication within the kernel is fast.



Linux System Structure

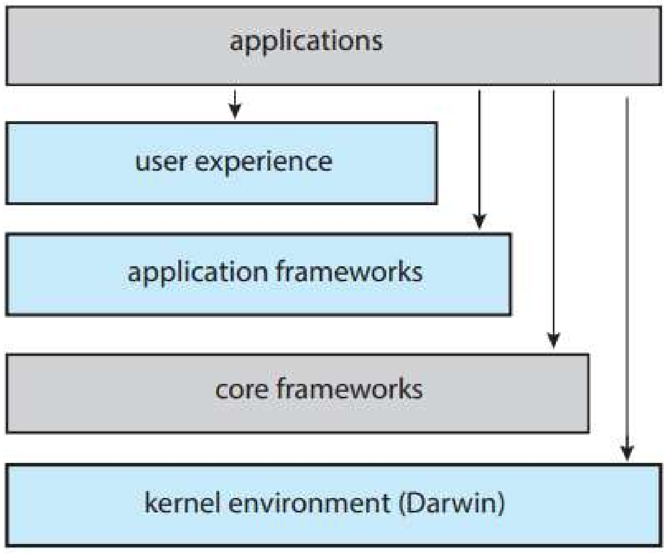
OS Structure – Layered Approach

- The monolithic approach is often known as a tightly coupled system because changes to one part of the system can have wide-ranging effects on other parts.
- Alternatively, we could design a loosely coupled system. Such a system is divided into separate, smaller components that have specific and limited functionality.
- All these components together comprise the kernel.
- The advantage of this modular approach is that changes in one component affect only that component, and no others, allowing system implementers more freedom in creating and changing the inner workings of the system.
- A system can be made modular in many ways. One method is the layered approach, in which the operating system is broken into a number of layers (levels). The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface.
- This layering structure is depicted in Figure

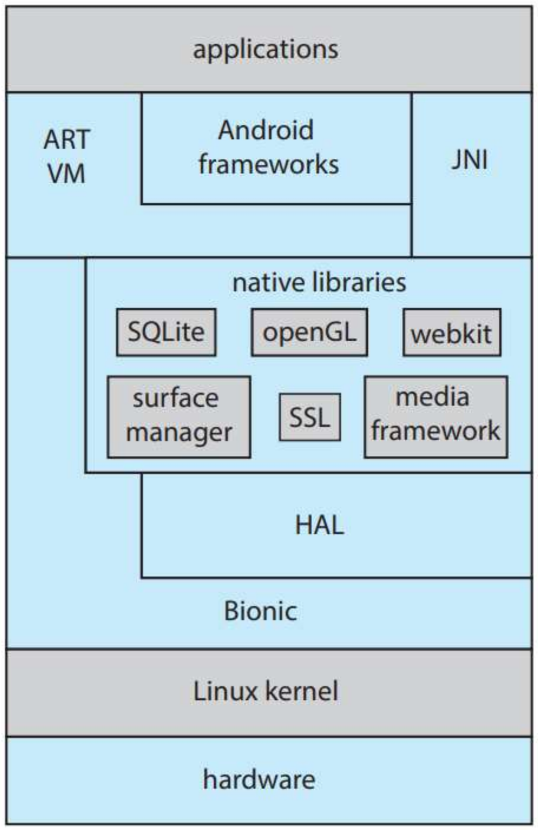


A layered OS

OS Structure



Architecture of Apple's macOS and iOS



Architecture of Google's Android

Building and Booting an Operating System

- It is possible to design, code, and implement an operating system specifically for one specific machine configuration.
- More commonly, however, operating systems are designed to run on any of a class of machines with a variety of peripheral configurations.

Operating System Generation

- Most commonly, a computer system, when purchased, has an operating system already installed.
- For example, you may purchase a new laptop with Windows or macOS preinstalled.
- But suppose you wish to replace the preinstalled operating system or add additional operating systems.
- Or suppose you purchase a computer without an operating system.
- In these latter situations, you have a few options for placing the appropriate operating system on the computer and configuring it for use.

Building and Booting an Operating System

- If you are generating (or building) an operating system from scratch, you must follow these steps:

1. Write the operating system source code (or obtain previously written source code).
2. Configure the operating system for the system on which it will run.
3. Compile the operating system.
4. Install the operating system.
5. Boot the computer and its new operating system.

- Configuring the system involves specifying which features will be included, and this varies by operating system.
- Typically, parameters describing how the system is configured is stored in a configuration file of some type, and once this file is created, it can be used in several ways.

Building and Booting an Operating System

- We now illustrate how to build a Linux system from scratch, where it is typically necessary to perform the following steps:

1. Download the Linux source code from http://www.kernel.org.
2. Configure the kernel using the “make menuconfig” command. This step generates the .config configuration file.
3. Compile the main kernel using the “make” command. The make command compiles the kernel based on the configuration parameters identified in the .config file, producing the file vmlinuz, which is the kernel image.
4. Compile the kernel modules using the “make modules” command. Just as with compiling the kernel, module compilation depends on the configuration parameters specified in the .config file.
5. Use the command “make modules install” to install the kernel modules into vmlinuz.
6. Install the new kernel on the system by entering the “make install” command

Building and Booting an Operating System

- When the system reboots, it will begin running this new operating system.
- Alternatively, it is possible to modify an existing system by installing a Linux virtual machine.
- This will allow the host operating system (such as Windows or macOS) to run Linux
- There are a few options for installing Linux as a virtual machine.
- One alternative is to build a virtual machine from scratch.
- This option is similar to building a Linux system from scratch; however, the operating system does not need to be compiled.
- Another approach is to use a Linux virtual machine appliance, which is an operating system that has already been built and configured.
- This option simply requires downloading the appliance and installing it using virtualization software such as VirtualBox or VMware.

Building and Booting an Operating System

- For example, to build the operating system used in the virtual machine provided with this text, the authors did the following:

1. Downloaded the Ubuntu ISO image from https://www.ubuntu.com/
2. Instructed the virtual machine software VirtualBox to use the ISO as the bootable medium and booted the virtual machine.
3. Answered the installation questions and then installed and booted the operating system as a virtual machine.

Building and Booting an Operating System

System Boot

- After an operating system is generated, it must be made available for use by the hardware.
- But how does the hardware know where the kernel is or how to load that kernel?
- The process of starting a computer by loading the kernel is known as booting the system.
- On most systems, the boot process proceeds as follows:

1. A small piece of code known as the bootstrap program or boot loader locates the kernel.
2. The kernel is loaded into memory and started.
3. The kernel initializes hardware.
4. The root file system is mounted.

Building and Booting an Operating System

- Some computer systems use a multistage boot process: When the computer is first powered on, a small boot loader located in non-volatile firmware known as BIOS is run.
- This initial boot loader usually does nothing more than load a second boot loader, which is located at a fixed disk location called the boot block.
- The program stored in the boot block may be sophisticated enough to load the entire operating system into memory and begin its execution.
- The boot process for mobile systems is slightly different from that for traditional PCs.
- Finally, boot loaders for most operating systems—including Windows, Linux, and macOS, as well as both iOS and Android—provide booting into recovery mode or single-user mode for diagnosing hardware issues, fixing corrupt file systems, and even reinstalling the operating system.

OS Debugging

- Debugging is the activity of finding and fixing errors in a system, both in hardware and in software.
- Performance problems are considered bugs, so debugging can also include performance tuning, which seeks to improve performance by removing processing bottlenecks.

Failure Analysis

- If a process fails, most operating systems write the error information to a logfilel to alert system administrators or users that the problem occurred.
- The operating system can also take a core dump—a capture of the memory of the process—and store it in a file for later analysis. (Memory was referred to as the "core" in the early days of computing.)
- Running programs and core dumps can be probed by a debugger, which allows a programmer to explore the code and memory of a process at the time of failure.

OS Debugging

- Debugging user-level process code is a challenge.
- Operating-system kernel debugging is even more complex because of the size and complexity of the kernel, its control of the hardware, and the lack of user-level debugging tools.
- A failure in the kernel is called a crash.
- When a crash occurs, error information is saved to a log file, and the memory state is saved to a crash dump.
- Operating-system debugging and process debugging frequently use different tools and techniques due to the very different nature of these two tasks.
- Consider that a kernel failure in the file-system code would make it risky for the kernel to try to save its state to a file on the file system before rebooting.
- A common technique is to save the kernel's memory state to a section of disk set aside for this purpose that contains no file system. If the kernel detects an unrecoverable error, it writes the entire contents of memory, or at least the kernel-owned parts of the system memory, to the disk area.

OS Debugging

Kernighan’s Law

“Debugging is twice as hard as writing the code in the first place.
Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.”

धन्यवाद