

Operating Systems – 1BCS304

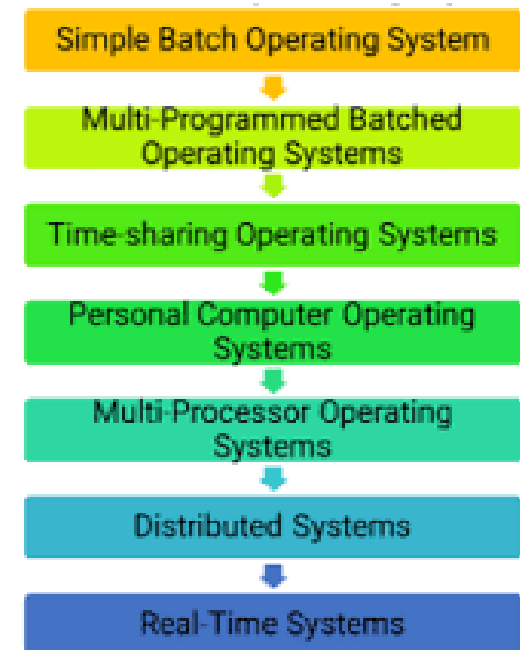
Course Coordinator:
Mrs. Pallavi A R
Assistant professor
Dept. of CSE-DS
ATMECE, Mysuru

Course Overview

- Operating Systems is a graduate-level introductory course.
- This course teaches the basic operating system abstractions, mechanisms, and their implementations.
- The core of the course contains concurrent programming (threads and synchronization), inter process communication, and an introduction to distributed operating systems.
- In this course, we dwell into how the OS manages to do all this in an efficient manner. This is an introductory course, for students with prior knowledge of computer organization.

Why this Course?

- Students will understand the rationale behind the current design and implementation decisions in modern OS's (like Linux) by considering the historic evolution of various OS constructs.
- Students will be exposed to theoretical knowledge regarding operating systems principles and implementation.
- Students will gain knowledge via experimenting and evaluating various OS aspects in a practical manner through case studies.



Course Information

Operating Systems	
Course Code	1BCS304
Number of Lecture Hours/Week	03 (Lecture)
Total Number of Lecture Hours	42 hours
Credits	3:0:0
CIE Marks	25+25
SEE Marks	50
Total Marks	100
Exam Hours	03

Course Syllabus

Modules

Module-1

Introduction to operating systems, System structures: What operating systems do; Computer System organization; Computer System architecture; Operating System structure; Operating System operations; Process management; Memory management; Storage management.

Operating System Services: User - Operating System interface; System calls; Types of system calls; System programs; Operating system design and implementation; Operating System structure;

Textbook 1: Chapter – 1 (1.1-1.8), 2 (2.2-2.7)

Number of Hours: 09

Module-2

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

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

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

Textbook 1: Chapter – 3 (3.1-3.4), 4 (4.1,4.3,4.4,4.6), 5 (5.1 -5.5)

Number of Hours: 08

Modules

Module-3

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

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

Textbook 1: Chapter – 6 (6.1-6.4, 6.6, 6.7), 7 (7.1 -7.7)

Number of Hours: 09

Module-4

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

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

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

Number of Hours: 08

Module-5

File System: File system: File concept; Access methods; Directory and Disk structure; File system mounting; File sharing.

Implementing File system: File system structure; File system implementation; Directory implementation; Allocation methods; Free space management.

Mass Storage Structure: Mass storage structures; Disk structure; Disk attachment; Disk scheduling; Disk management.

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

Number of Hours: 08

Suggested Learning Resources

Textbooks

1. Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System Principles, 9th edition, Wiley-India, 2015.

Reference Books

1. Ann McHoes Ida M Fylnn, Understanding Operating System, Cengage Learning, 6th Edition.
2. D.M Dhamdhere, Operating Systems: A Concept Based Approach 3rd Ed, McGraw- Hill, 2013.
3. William Stallings Operating Systems: Internals and Design Principles, 6th Edition, Pearson.

Course Outcomes (COs)

At the end of the course, the student will be able to:

1. Recall the structure, functions, and services of an operating system.
2. Explain process management concepts, CPU scheduling algorithms, and process synchronization mechanisms.
3. Apply appropriate CPU scheduling algorithms and process synchronization techniques to solve operating system problems.
4. Apply memory management techniques and deadlock-handling strategies to ensure efficient resource utilization.
5. Explain file system organization and secondary storage management techniques.
6. Describe information protection mechanisms, security principles, and access control methods in operating systems.

Course Assessment Policy

- **Continuous Internal Evaluation:**

Continuous Comprehensive Assessments (CCA)+ Internal Assessment Test (IA) = Continuous Internal Evaluation CIE [25+25=50 marks] A minimum of two Internal Assessment Tests (IA) shall be conducted, carrying a total of 25 marks. In addition, Continuous Comprehensive Assessment (CCA) shall be conducted for a total of 25 marks. It is recommended to include a maximum of two learning activities as part of the CCA to foster the holistic development of students.

Term Work (TW) and Self Learning (SL) components in Number of Hours per semester

Term work (includes assignments, seminars, micro projects, industrial visits, any other student learning activities etc.)

Sl. No.	Term Work (TW) Activity	Number of Hours / Semester
1.	Implementation of Operating System Concepts Using C Programming	30

SL: Self Learning, MOOCs, spoken tutorials, online educational resources etc.

Sl. No.	Self-Learning (SL) Activity	Number of Hours / Semester
1	Springboard Certification on: Introduction to Unix	18

Activity Based Learning Component

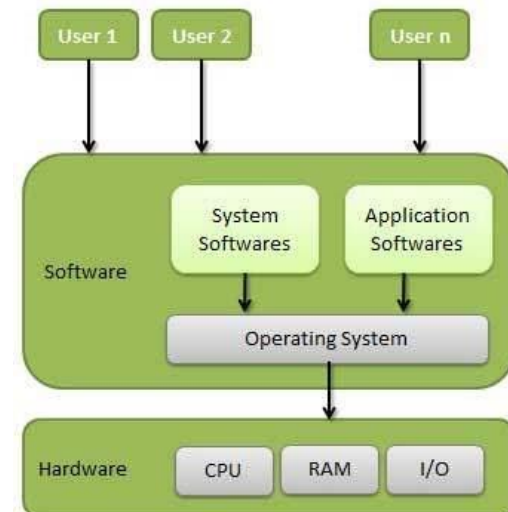
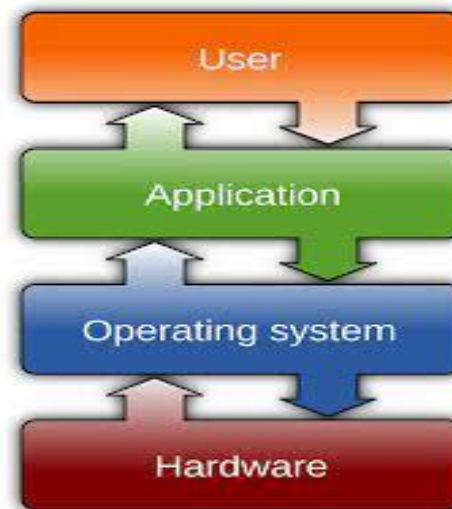
Module Number	Term Work (TW) Activity Component
Module 1	Active learning and problem solving 1. https://www.youtube.com/watch?v=vBURTt97EkA&list=PLBlnK6fEyqRiVhbXDGLXDk_OQAeuVcp2O 2. https://www.youtube.com/watch?v=a2B69vCtjOU&list=PL3-wYxbt4yCjpcfUDz-TgD_ainZ2K3MUZ&index=2
Module 2	Active Learning and problem solving 1. https://www.youtube.com/watch?v=HW2Wcx-ktsc 2. https://www.youtube.com/watch?v=9YRxhlvt9Zo
Module 3	Active Learning, Problem solving based on deadlock with animation 1. https://www.youtube.com/watch?v=MYgmmJJfdBg 2. https://www.youtube.com/watch?v=Y14b7_T3AEw&list=PLEJxKK7AcSEGPOCFtQTJh0_ElU44I_JAun&index=30

Activity Based Learning Component

Module Number	Term Work (TW) Activity Component
Module 4	Active learning about memory management and File system 1. https://www.youtube.com/watch?v=pJ6qrCB8pDw&list=PLIY8eNdw5tW-BxRY0yK3fYTYVqytw8qhp 2. https://www.youtube.com/watch?v=-orfFhvNBzY
Module 5	Active learning about case studies 1. https://www.youtube.com/watch?v=TTBkc5eiju4 2. https://www.youtube.com/watch?v=8hkvMRGTzCM&list=PLEAYkSg4uSQ2PAch478muxnoeTNz_QeUJ&index=36 3. https://www.youtube.com/watch?v=mX1FEur4VCw

Introduction

- An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.
- An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.



Examples of Operating systems



chromeOS



ubuntu



iOS

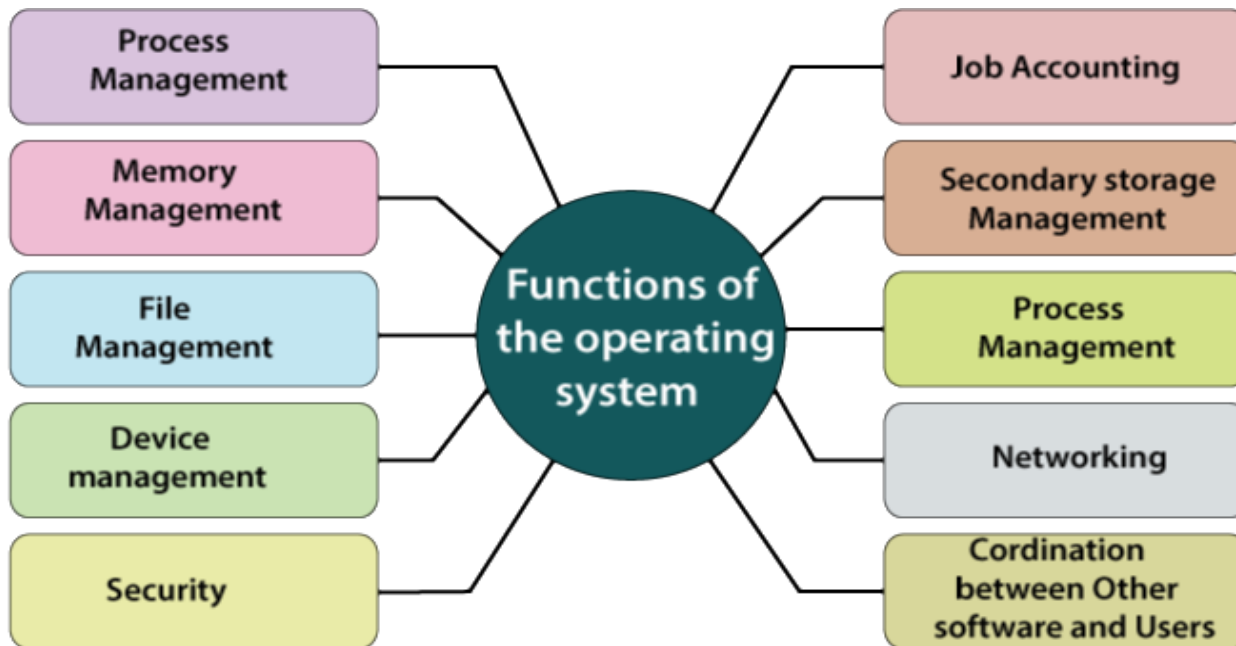
symbian
OS



UNIX

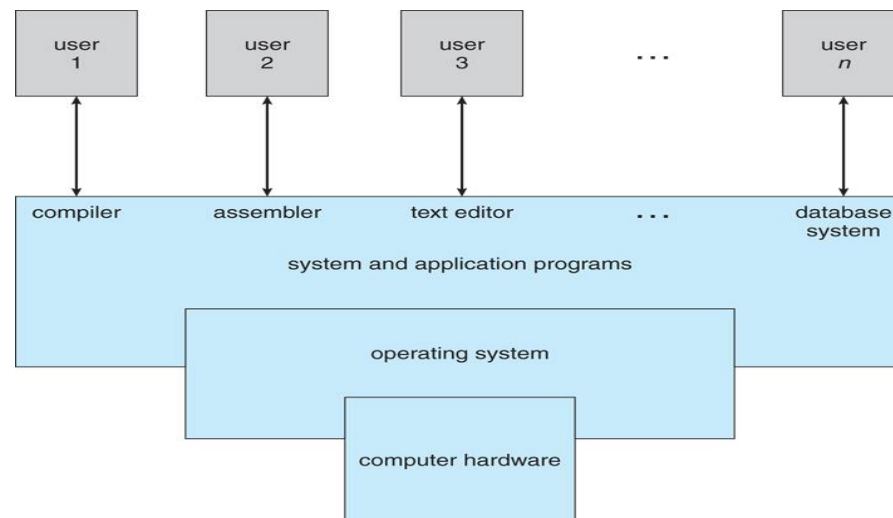
www.knowcomputing.com

Functions of OS



What operating systems do

- Let us understand the operating system's role in overall computer system.
- Computer system can be divided into four components:
 - Hardware
 - Operating system
 - Application programs
 - Users



What operating systems do

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

What operating systems do

- The computer system can also be viewed as consisting of hardware, software and data.
- The operating system provides the means for proper use of these resources in the operation of the computer systems.
- To understand the operating system's role better, let us consider the operating system from two viewpoints
 1. User
 2. System

Operating system: System View

- OS is a **resource allocator**
 - Manages all resources
 - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
 - Controls execution of programs to prevent errors and improper use of computer

Operating system: User View

- Users want convenience, **ease of use** – some attention paid to **performance** and none paid to **resource utilization**.
- **Performance** – optimized for single user experience rather than the requirements of multiple users.
- Being considered as **mainframes** or **minicomputers** – resource utilization through shared resources.
- Users of dedicate systems such as **workstations** have dedicated resources but frequently use shared resources from **servers**. (connecting to networks).

Operating system: User View

- **Handheld computers** such as mobiles and tablets are standalone units for individual users – OS is designed for individual usability and battery life.
- Some computers have little or no user interface, such as **embedded computers** in home devices and automobiles – OS is designed primarily to run without user intervention.
- With all these views – user and system, we can understand that the OS is designed for multiple purposes and cannot define the OS for a specific task – **No adequate definition for OS as the views are different.**

Computer System Organization

- Computer-system operation
 - One or more CPUs, device controllers connect through common bus providing access to shared memory (as shown in fig. below)
 - Concurrent execution of CPUs and devices competing for memory cycles

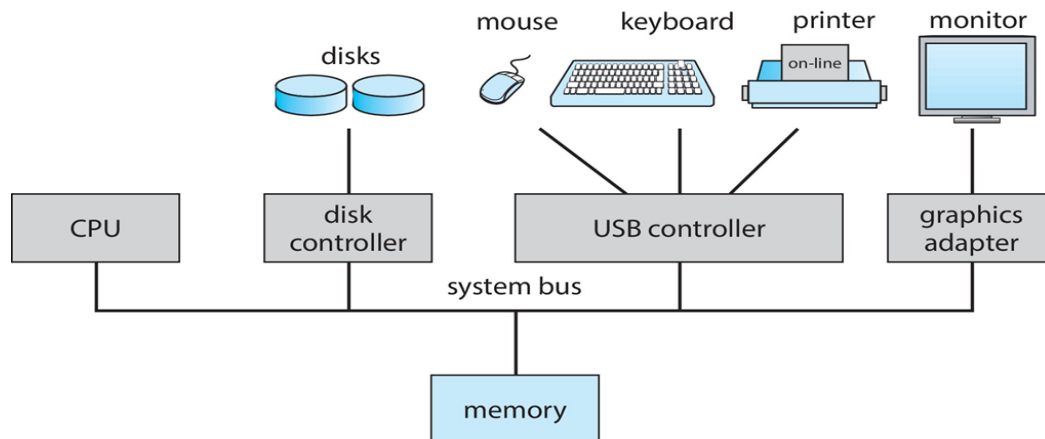


Fig. A Modern Computer System

Computer System Organization

- When the system is switched ON, **Bootstrap** program is executed. It is the initial program to run in the system which is typically stored in ROM or EEPROM.
- It initializes all aspects of the system, from CPU registers to device controllers to memory contents. The bootstrap program must know how to load the operating system & how to start executing the system. To do this, the bootstrap program must locate the operating system **kernel** and load it into memory.
- The OS, then starts with the first process to be executed (i.e., **init** process) and then wait for the interrupt from the user.

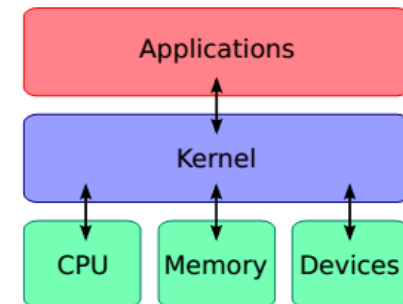
Computer System Organization

Switch on

'Bootstrap' program

- Initializes the registers, memory and I/O devices
-
- Locates & loads kernel into memory
- Starts with 'init' process
- Waits for interrupt from user.

Kernel is central component of an operating system that manages operations of computer and hardware. It basically manages operations of memory and CPU time. It is core component of an operating system. Kernel acts as a bridge between applications and data processing performed at hardware level using inter-process communication and system calls.



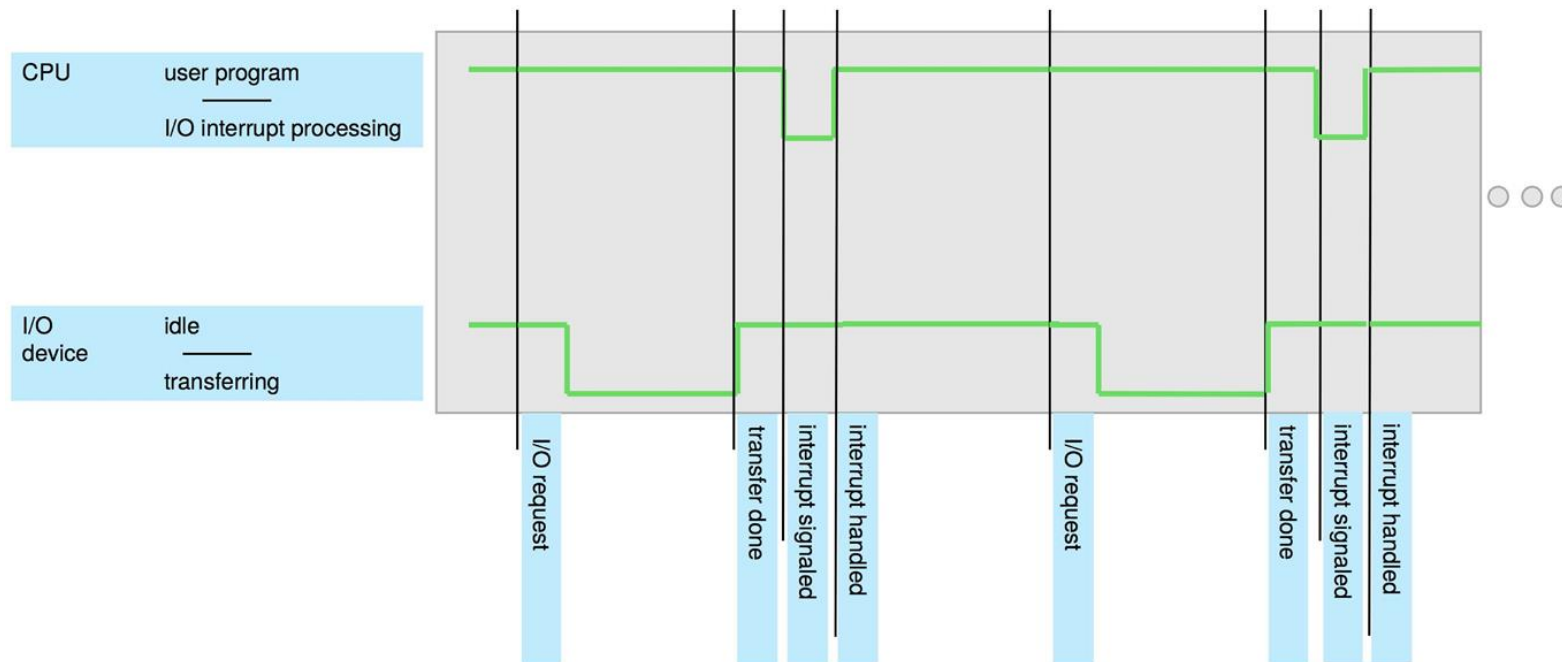
Computer System Operation

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

Common Functions of Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the **interrupt vector**, which contains the addresses of all the service routines
- Interrupt architecture must save the address of the interrupted instruction
- A **trap or exception** is a software-generated interrupt caused either by an error or a user request
- An operating system is **interrupt driven**

Interrupt Timeline



Interrupt Handling

- The occurrence of an event is usually signaled by an interrupt. The interrupt can either be from the hardware or the software. Hardware may trigger an interrupt at any time by sending a signal to the CPU. Software triggers an interrupt by executing a special operation called a system call (also called a monitor call).
- When the CPU is interrupted, it stops what it is doing and immediately transfers execution to a fixed location. The fixed location (Interrupt Vector Table) contains the starting address where the service routine for the interrupt is located. After the execution of interrupt service routine, the CPU resumes the interrupted computation.
- Interrupts are an important part of computer architecture. Each computer design has its own interrupt mechanism, but several functions are common. The interrupt must transfer control to the appropriate interrupt service routine.

Storage Structure

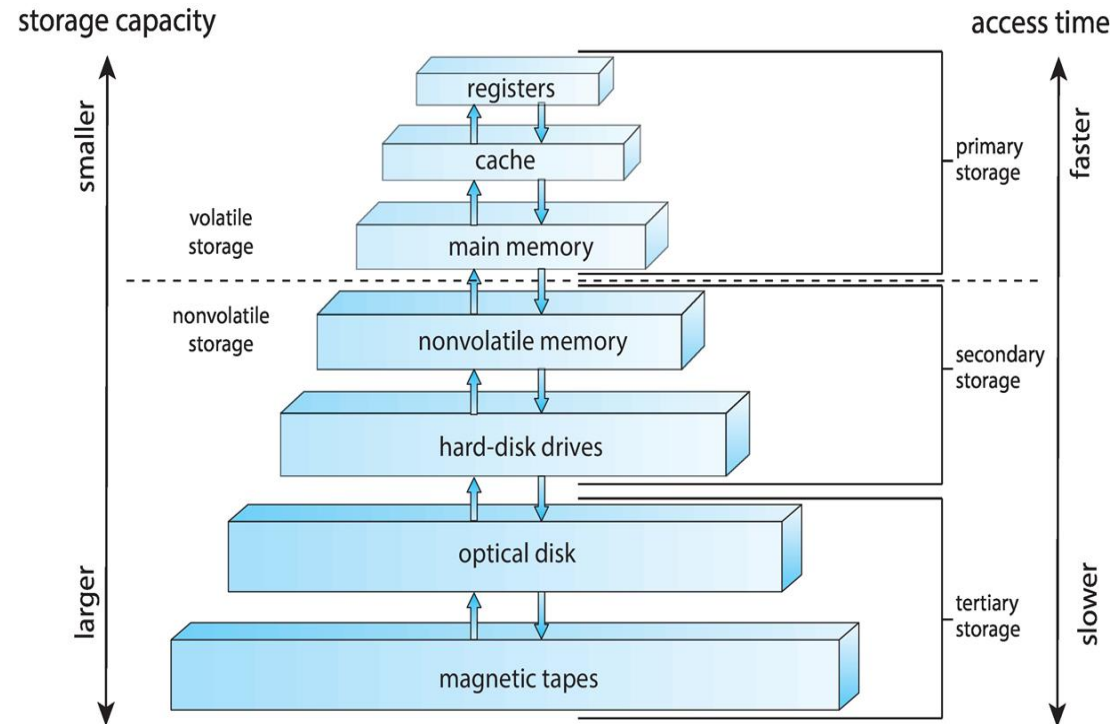
- Computer programs must be in **main memory (RAM)** to be executed.
- **Main memory** – only large storage media that the CPU can access directly
 - Random access
 - Typically volatile
 - Typically random-access memory in the form of Dynamic Random-access Memory (DRAM)
- All forms of memory provide an array of **memory words**. Each word has its own **address**. Interaction is achieved through a sequence of **load or store instructions** to specific memory addresses.
- A typical instruction-execution cycle, as executed on a system with a **Von Neumann architecture**, first fetches an instruction from memory and stores that instruction in the instruction register.

Storage Structure

- The instruction is then decoded and may cause operands to be fetched from memory and stored in some internal register. After the instruction on the operands has been executed, the result may be stored back in memory.
- Ideally, we want the programs and data to reside in main memory permanently. This arrangement usually is not possible for the following two reasons:
 1. *Main memory is usually too small to store all needed programs and data permanently.*
 2. *Main memory is a volatile storage device that loses its contents when power is turned off.*
- Thus, most computer systems provide secondary storage as an extension of main memory. The main requirement for secondary storage is that it will be able to hold large quantities of data permanently.

Storage Structure & Hierarchy

- The most common secondary-storage device is a **magnetic disk**, which provides storage for both programs and data.
- The wide variety of storage systems in a computer system can be organized in a hierarchy as shown in the figure, according to speed, cost and capacity.



Storage Hierarchy

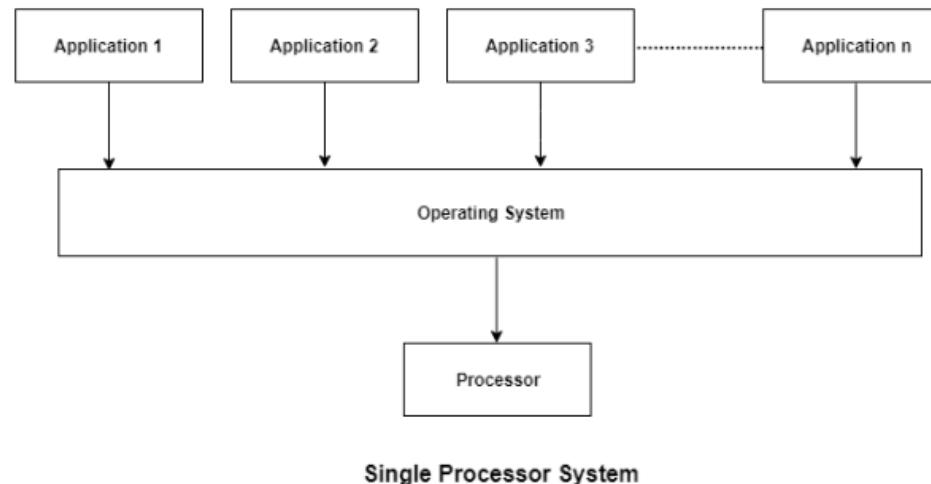
- The higher levels are expensive, but they are fast.
- As we move down the hierarchy, the cost per bit generally decreases, whereas the access time and the capacity of storage generally increases.
- In addition to differing in speed and cost, the various storage systems are either volatile or nonvolatile.
- Volatile storage loses its contents when the power to the device is removed.
- In the absence of expensive battery and generator backup systems, data must be written to nonvolatile storage for safekeeping.

Computer System Architecture

- Categorized roughly according to the number of general-purpose processors used.
 - Single-Processor Systems
 - Multi -Processor Systems (parallel systems or tightly coupled systems)
 1. Asymmetric multiprocessing
 2. Symmetric multiprocessing (SMP)
 - Clustered Systems
 1. Asymmetric clustering
 2. Symmetric clustering

Computer System Architecture - Single-Processor Systems

- The use of special-purpose microprocessors is common and does not turn a single-processor system into a multiprocessor. If there is only one general-purpose CPU, then the system is a single-processor system.



Computer System Architecture - Multi -Processor Systems

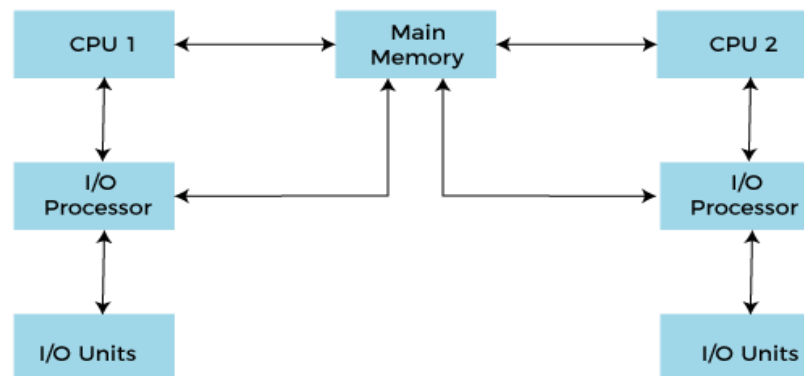
- **Multi -Processor Systems (parallel systems or tightly coupled systems)**
 - Systems that have two or more processors in close communication, sharing the computer bus, the clock, memory, and peripheral devices are the multiprocessor systems.
 - Multiprocessor systems have three main advantages:
 1. **Increased throughput** - In multiprocessor system, as there are multiple processors execution of different programs take place simultaneously. Even if the number of processors is increased the performance cannot be simultaneously increased. This is due to the overhead incurred in keeping all the parts working correctly and also due to the competition for the shared resources. The speed-up ratio with N processors is not N, rather, it is less than N. Thus the speed of the system is not has expected.

Computer System Architecture - Multi -Processor Systems

2. **Economy of scale** - Multiprocessor systems can cost less than equivalent number of many single-processor systems. As the multiprocessor systems share peripherals, mass storage, and power supplies, the cost of implementing this system is economical. If several processes are working on the same data, the data can also be shared among them.
 3. **Increased reliability** - In multiprocessor systems functions are shared among several processors. If one processor fails, the system is not halted, it only slows down. The job of the failed processor is taken up, by other processors.
- Two techniques to maintain 'Increased Reliability' –
 1. Graceful degradation
 2. Fault tolerant

Computer System Architecture - Multi -Processor Systems

1. **Graceful degradation** – As there are multiple processors when one processor fails other process will take up its work and the system go down slowly.
2. **Fault tolerant** – When one processor fails, its operations are stopped, the system failure is then detected, diagnosed, and corrected.

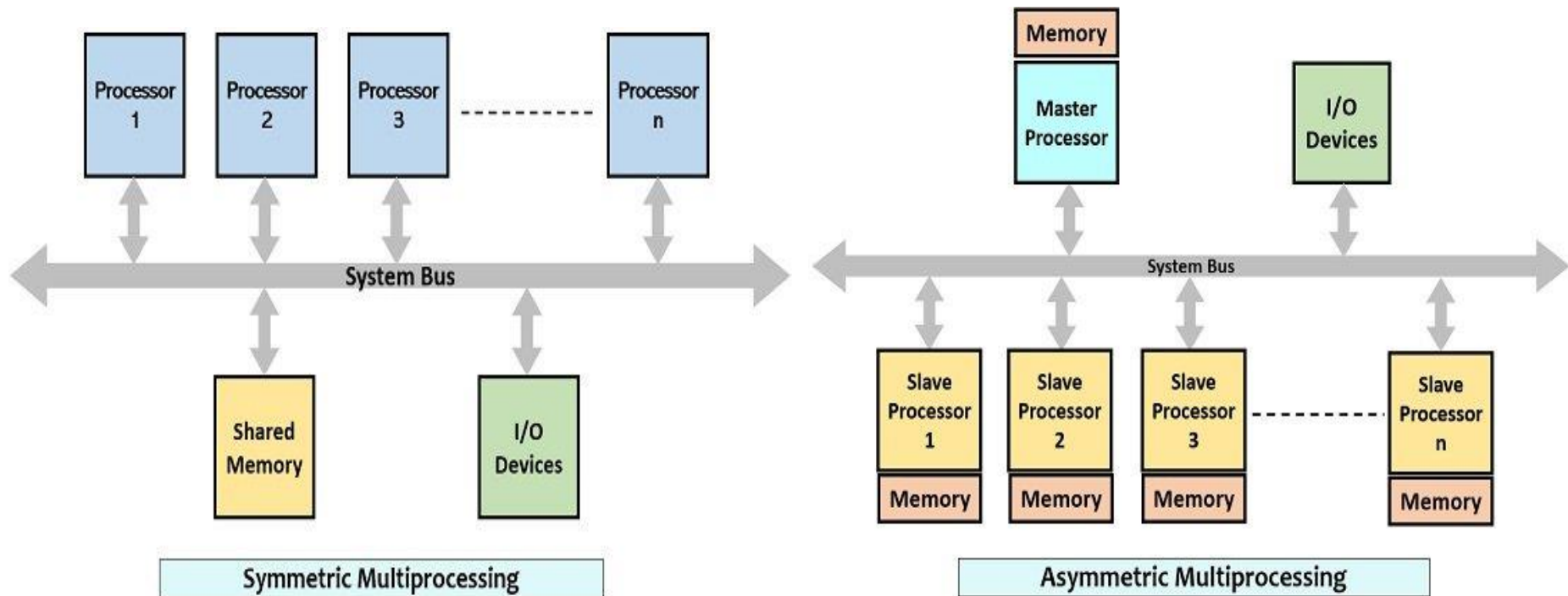


Working of Multiprocessor System

Computer System Architecture - Multi -Processor Systems

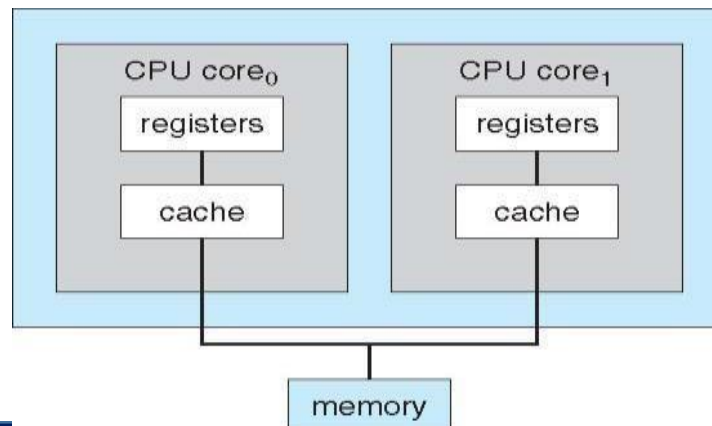
- Different types of multiprocessor systems
 1. Asymmetric multiprocessing
 2. Symmetric multiprocessing
- **Asymmetric multiprocessing – (Master/Slave architecture)**
 - Here each processor is assigned a specific task, by the master processor. A master processor controls the other processors in the system. It schedules and allocates work to the slave processors.
- **Symmetric multiprocessing (SMP)**
 - All the processors are considered peers. There is no master-slave relationship. All the processors have their own registers and CPU, only memory is shared.

Computer System Architecture - Multi-Processor Systems



Computer System Architecture - Multi-Processor Systems

- The benefit of this model is that many processes can run simultaneously. N processes can run if there are N CPUs—without causing a significant deterioration of performance. Operating systems like Windows, Windows XP, Mac OS X, and Linux—now provide support for SMP. A recent trend in CPU design is to include multiple compute cores on a single chip.
- The communication between processors within a chip is faster than communication between two single processors.

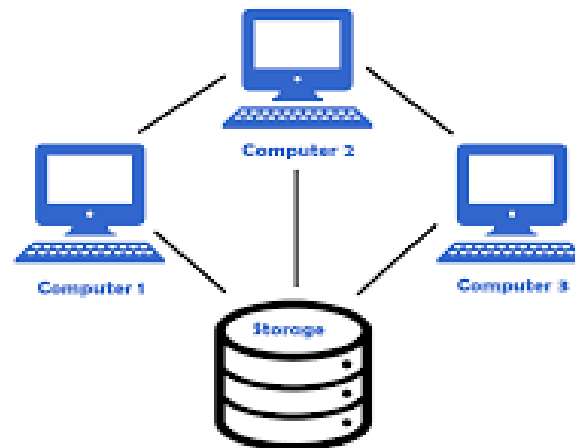


Computer System Architecture - Clustered Systems

- Clustered systems are two or more individual systems connected together via a network and sharing software resources. Clustering provides high availability of resources and services.
- The service will continue even if one or more systems in the cluster fail.
- High availability is generally obtained by storing a copy of files (s/w resources) in the system.
- There are two types of Clustered systems
 1. Asymmetric
 2. Symmetric

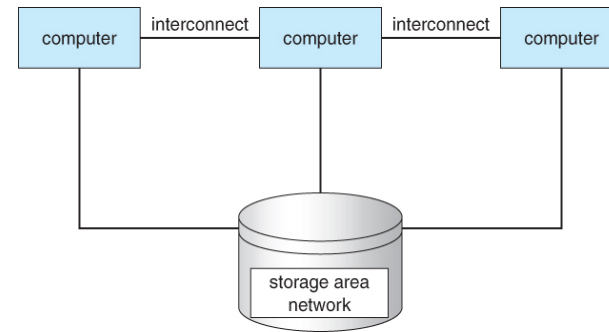
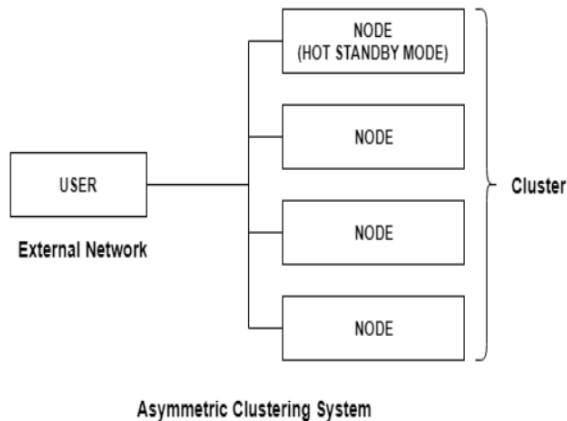
Computer System Architecture - Clustered Systems

- **Asymmetric clustering** – one system is in hot-standby mode while the others are running the applications. The hot-standby host machine does nothing but monitor the active server. If that server fails, the hot-standby host becomes the active server.
- **Symmetric clustering** – two or more systems are running applications, and are monitoring each other. This mode is more efficient, as it uses all of the available hardware. If any system fails, its job is taken up by the monitoring system.



Clustered System

Computer System Architecture - Clustered Systems



- Other forms of clusters include parallel clusters and clustering over a wide-area network (WAN). Parallel clusters allow multiple hosts to access the same data on the shared storage. Cluster technology is changing rapidly with the help of SAN (storage-area networks). Using SAN resources can be shared with dozens of systems in a cluster, that are separated by miles.

Operating System Structure

- **Multiprogramming**

- One of the most important aspects of operating systems is the ability to multiprogram.
- A single program cannot keep either the CPU or the I/O devices busy at all times.
- Multiprogramming increases CPU utilization by organizing jobs, so that the CPU always has one to execute.

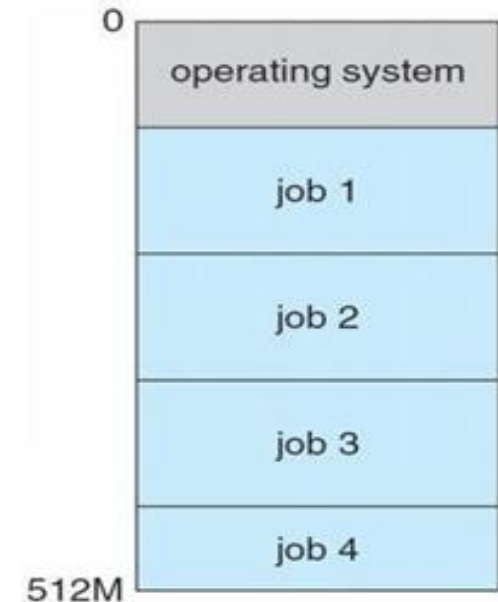


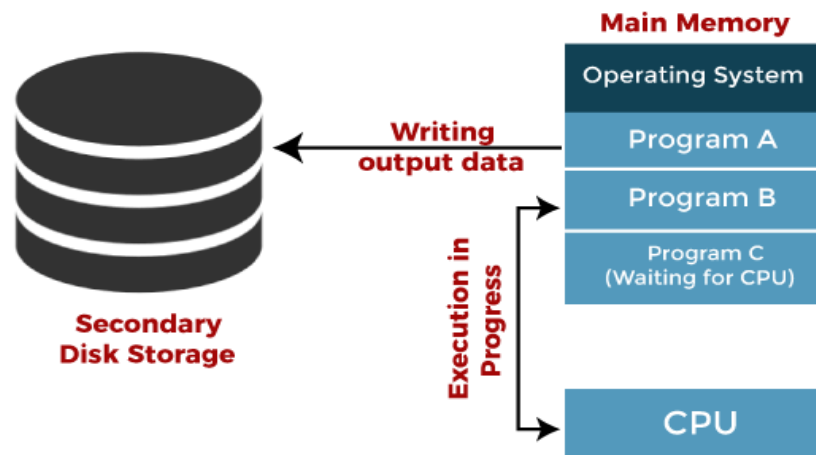
Fig - Memory layout for a multiprogramming system

Operating System Structure

- The operating system keeps several jobs in memory simultaneously as shown in figure. This set of jobs is a subset of the jobs kept in the job pool. Since the number of jobs that can be kept simultaneously in memory is usually smaller than the number of jobs that can be kept in the job pool (in secondary memory).
- The operating system picks and begins to execute one of the jobs in memory. Eventually, the job may have to wait for some tasks, such as an I/O operation, to complete.
- In a non-multiprogram system, the CPU would sit idle.
- In a multiprogrammed system, the operating system simply switches to, and executes, another job. When that job needs to wait, the CPU is switched to another job, and so on.
- Eventually, the first job finishes waiting and gets the CPU back. Thus, the CPU is never idle.

Operating System Structure

- Multiprogrammed systems provide an environment in which the various system resources (for example, CPU, memory, and peripheral devices) are utilized effectively, but they do not provide for user interaction with the computer system.

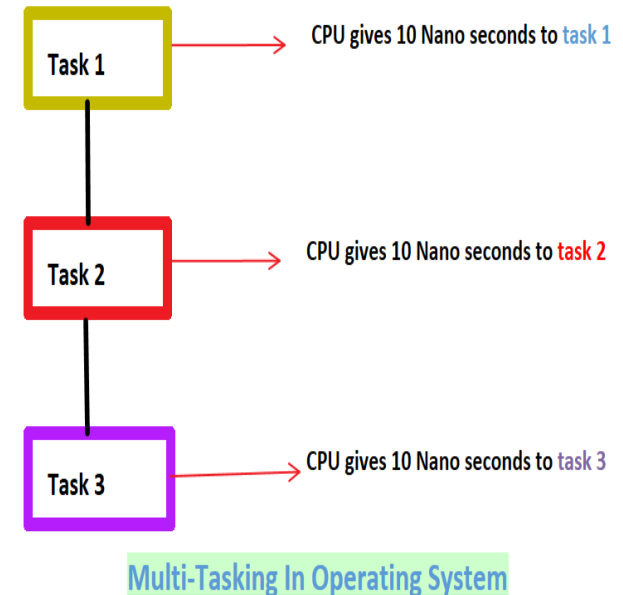


Jobs in multiprogramming system

Operating System Structure

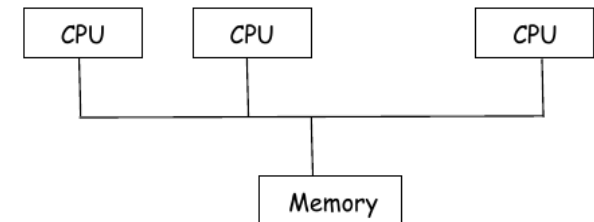
• Multitasking Systems

- In **Time sharing (or multitasking) systems**, a single CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running. The user feels that all the programs are being executed at the same time.
- Time sharing requires an **interactive (or hands-on)** computer system, which provides direct communication between the user and the system.



Operating System Structure

- The user gives instructions to the operating system or to a program directly, using a input device such as a keyboard or a mouse, and waits for immediate results on an output device. Accordingly, the response time should be short— typically less than one second.
- A time-shared operating system allows many users to share the computer simultaneously. As the system switches rapidly from one user to the next, each user is given the impression that the entire computer system is dedicated to his use only, even though it is being shared among many users.
- A **multiprocessor system** is a computer system having two or more CPUs within a single computer system, each sharing main memory and peripherals. Multiple programs are executed by multiple processors parallel.



Operating-System Operations

- Modern operating systems are **interrupt driven**. If there are no processes to execute, no I/O devices to service, and no users to whom to respond, an operating system will sit quietly, waiting for something to happen.
- Events are signaled by the occurrence of an **interrupt** or a **trap**.
- A **trap** (or an exception) is a software-generated interrupt. For each type of interrupt, separate segments of code in the operating system determine what action should be taken.
- An interrupt service routine is provided that is responsible for dealing with the interrupt.

Dual-Mode Operation

- Since the operating system and the user programs share the hardware and software resources of the computer system, it has to be made sure that an error in a user program cannot cause problems to other programs and the Operating System running in the system.
- The approach taken is to use a hardware support that allows us to differentiate among various modes of execution.
- The system can be assumed to work in two separate modes of operation:
 1. User mode
 2. Kernel mode (supervisor mode, system mode, or privileged mode).

Dual-Mode Operation

- A hardware bit of the computer, called the mode bit, is used to indicate the current mode: kernel (0) or user (1). With the mode bit, we are able to distinguish between a task that is executed by the operating system and one that is executed by the user.
- When the computer system is executing a user application, the system is in user mode. When a user application requests a service from the operating system (via a system call), the transition from user to kernel mode takes place.

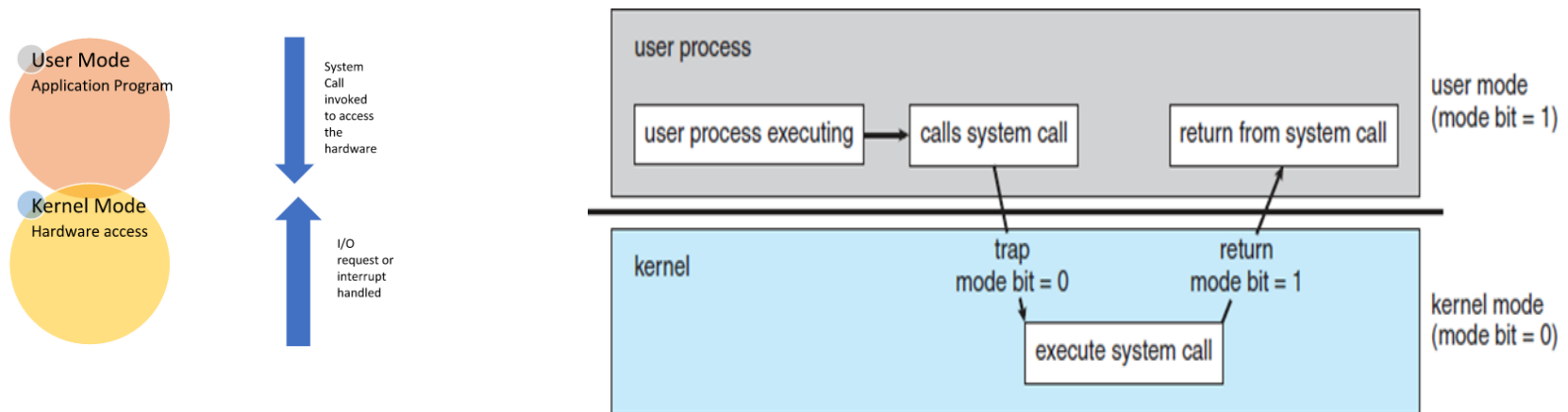


Figure Transition from user to kernel mode.

Dual-Mode Operation

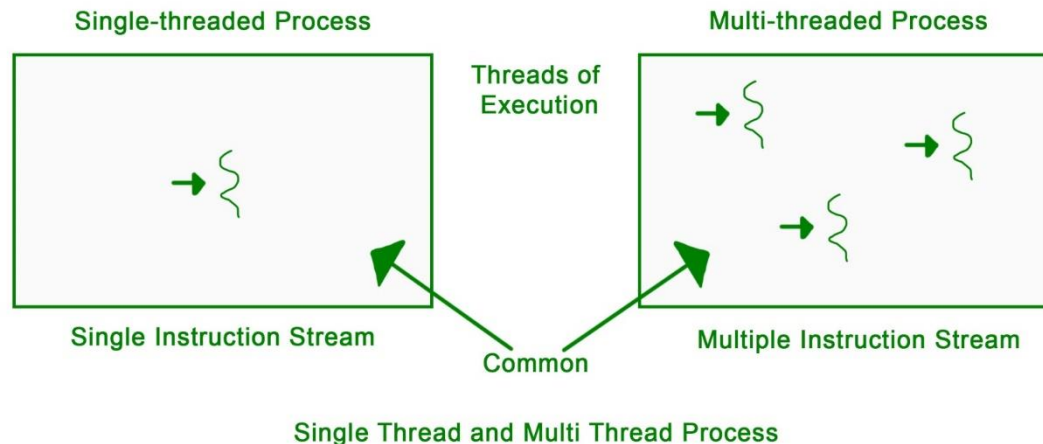
- At system boot time, the hardware starts in kernel mode.
- The operating system is then loaded and starts user applications in user mode.
- Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the mode bit from 1 to 0). Thus, whenever the operating system gains control of the computer, it is in kernel mode.

Process Management

- A program under execution is a process.
- A process needs resources like CPU time, memory, files, and I/O devices for its execution. These resources are given to the process when it is created or at run time.
- When the process terminates, the operating system reclaims the resources. The program stored on a disk is a **passive entity** and the program under execution is an **active entity**.
- A **single-threaded process** has one program counter specifying the next instruction to execute. The CPU executes one instruction of the process after another, until the process completes.
- A **multithreaded process** has multiple program counters, each pointing to the next instruction to execute for a given thread.

Process Management

- The concept of multi-threading needs proper understanding of these two terms – a **process** and a **thread**. A process is a program being executed. A process can be further divided into independent units known as threads. A thread is like a small light-weight process within a process. Or we can say a collection of threads is what is known as a process.



Process Management

- The operating system is responsible for the following activities in connection with process management:
 - Scheduling process and threads on the CPU
 - Creating and deleting both user and system processes
 - Suspending and resuming processes
 - Providing mechanisms for process synchronization
 - Providing mechanisms for process communication

Memory Management

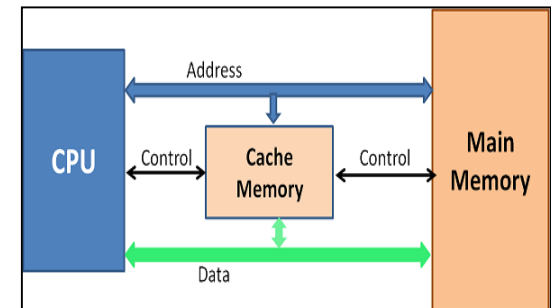
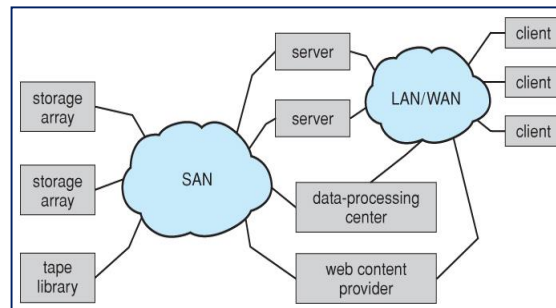
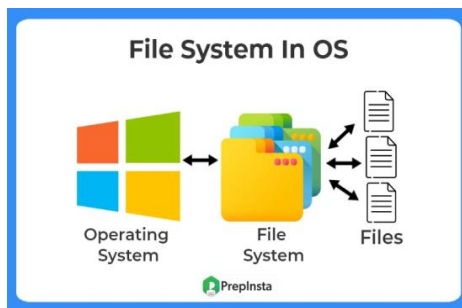
- Main memory is a large array of words or bytes. Each word or byte has its own address.
- Main memory is the storage device which can be easily and directly accessed by the CPU.
- As the program executes, the central processor reads instructions and also reads and writes data from main memory.
- To improve both the utilization of the CPU and the speed of the computer's response to its users, general-purpose computers must keep several programs in memory, creating a need for memory management.

Memory Management

- The operating system is responsible for the following activities in connection with memory management:
 - Keeping track of which parts of memory are currently being used by user.
 - Deciding which processes and data to move into and out of memory.
 - Allocating and deallocating memory space as needed.

Storage Management

- There are three types of storage management
 - File system management
 - Mass-storage management
 - Cache management



Storage Management - File-System Management

- File management is one of the most visible components of an operating system.
- Computers can store information on several different types of physical media.
- Magnetic disk, optical disk, and magnetic tape are the most common. Each of these media has its own characteristics and physical organization. Each medium is controlled by a device, such as a disk drive or tape drive, that also has its own unique characteristics.
- A file is a collection of related information defined by its creator. Commonly, files represent programs and data.
- Data files may be numeric, alphabetic, alphanumeric, or binary. Files may be free-form (for example, text files), or they may be formatted rigidly (for example, fixed fields).

Storage Management - File-System Management

- When multiple users have access to files, it may be desirable to control by whom and in what ways (read, write, execute) files may be accessed.
- The operating system is responsible for the following activities in connection with file management:
 - Creating and deleting files
 - Creating and deleting directories to organize files
 - Mapping files onto secondary storage
 - Backing up files on stable (nonvolatile) storage media



Storage Management - Mass-Storage Management

- As the main memory is too small to accommodate all data and programs, and as the data that it holds are erased when power is lost, the computer system must provide secondary storage to back up main memory.
- Most modern computer systems use disks as the storage medium for both programs and data.
- Most programs including compilers, assemblers, word processors, editors, and formatters are stored on a disk until loaded into memory and then use the disk as both the source and destination of their processing. Hence, the proper management of disk storage is of central importance to a computer system.

Storage Management - Mass-Storage Management

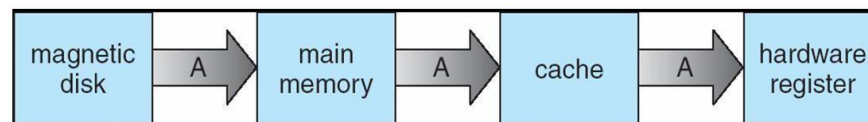
- The operating system is responsible for the following activities in connection with disk management:
 - Free-space management
 - Storage allocation
 - Disk scheduling

Storage Management - Caching

- Caching is an important principle of computer systems. Information is normally kept in some storage system (such as main memory).
- As it is used, it is copied into a faster storage system— the cache—as temporary data.
- When a particular piece of information is required, first we check whether it is in the cache. If it is, we use the information directly from the cache; if it is not in cache, we use the information from the source, putting a copy in the cache under the assumption that we will need it again soon.
- Because caches have limited size, cache management is an important design problem. Careful selection of the cache size and page replacement policy can result in greatly increased performance.

Storage Management - Caching

- The movement of information between levels of a storage hierarchy may be either explicit or implicit, depending on the hardware design and the controlling operating-system software.
- For instance, data transfer from cache to CPU and registers is usually a hardware function, with no operating-system intervention.
- In contrast, transfer of data from disk to memory is usually controlled by the operating system.
- In a hierarchical storage structure, the same data may appear in different levels of the storage system. For example, suppose to retrieve an integer A from magnetic disk to the processing program.



Storage Management - Caching

- Since the various CPUs can all execute concurrently, any update done to the value of A in one cache is immediately reflected in all other caches where A resides. This situation is called **cache coherency**, and it is usually a hardware problem (handled below the operating-system level).

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	< 16 MB	< 64 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

I/O Systems

- One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user.
- 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.

Operating-System Services

- An operating system provides an environment for the execution of programs. It provides certain services to programs and to the users of those programs. Fig. below shows one view of the various operating system services and how they interrelate.

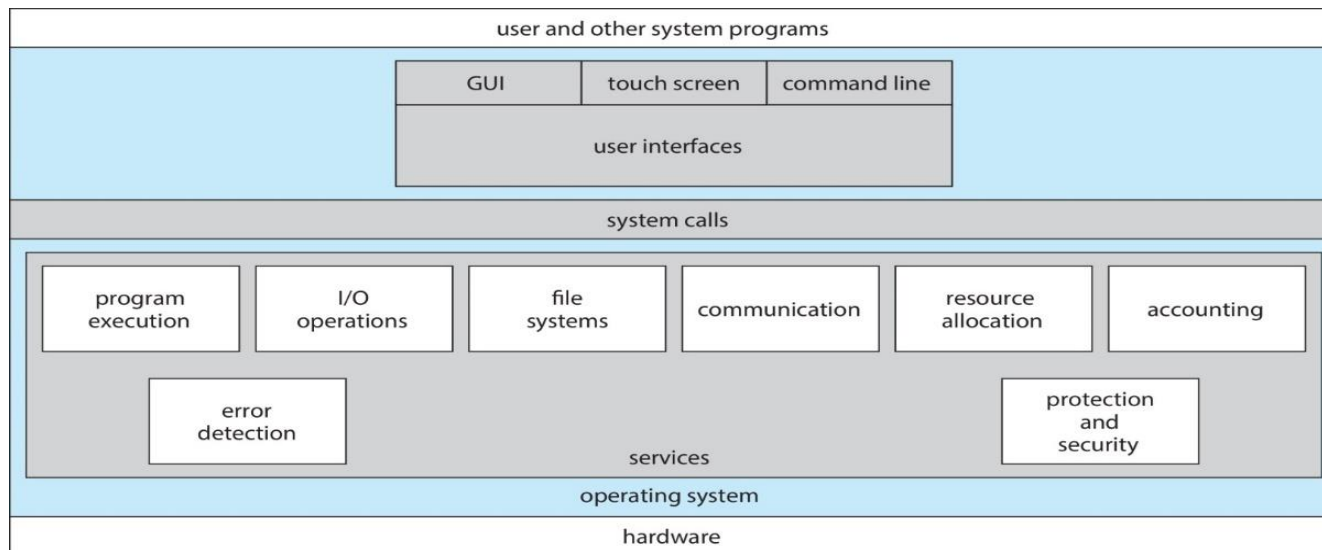


Fig. A view of operating system services

Operating-System Services

- One set of operating-system services provides functions that are helpful to the user:
- **User interface** - Almost all operating systems have a user interface (UI).
Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch interface (BI).
- **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error).
- **I/O operations** - A running program may require I/O, which may involve a file or an I/O device.
- **File-systems** - The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

Operating-System Services

- **Communications** - Processes may exchange information, on the same computer or between computers over a network.
Communications may be via shared memory or through message passing (packets moved by the OS).
- **Error detection** - OS needs to be constantly aware of possible errors.
May occur in the CPU and memory hardware, in I/O devices, in user program.
For each type of error, OS should take the appropriate action to ensure correct and consistent computing.
Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system.

Operating-System Services

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing:
- **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them.
Many types of resources - CPU cycles, main memory, file storage, I/O devices.
- **Logging** - To keep track of which users use how much and what kinds of computer resources.
- **Protection and security** - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other.
 - Protection** involves ensuring that all access to system resources is controlled.
 - Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts.

User Operating-System Interface

- There are several ways for users to interface with the operating system.
 1. **Command-line interface (CLI)**, or command interpreter, allows users to directly enter commands to be performed by the operating system.
 2. **Graphical user interface (GUI)**, allows users to interface with the operating system using pointer device and menu system.

Command-line interface (CLI)

- Command Interpreters are used to give commands to the OS. There are multiple command interpreters known as shells. In UNIX and Linux systems, there are several different shells, like the Bourne shell, C shell, Bourne-Again shell, Korn shell, and others.
- The main function of the command interpreter is to get and execute the user-specified command. Many of the commands manipulate files: create, delete, list, print, copy, execute, and so on.
- The commands can be implemented in two general ways-
 1. The command interpreter itself contains the code to execute the command. For example, a command to delete a file may cause the command interpreter to jump to a particular section of its code that sets up the parameters and makes the appropriate system call.
 2. The code to implement the command is in a function in a separate file. The interpreter searches for the file and loads it into the memory and executes it by passing the parameter. Thus by adding new functions new commands can be added easily to the interpreter without disturbing it.

Command-line interface (CLI)

```

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

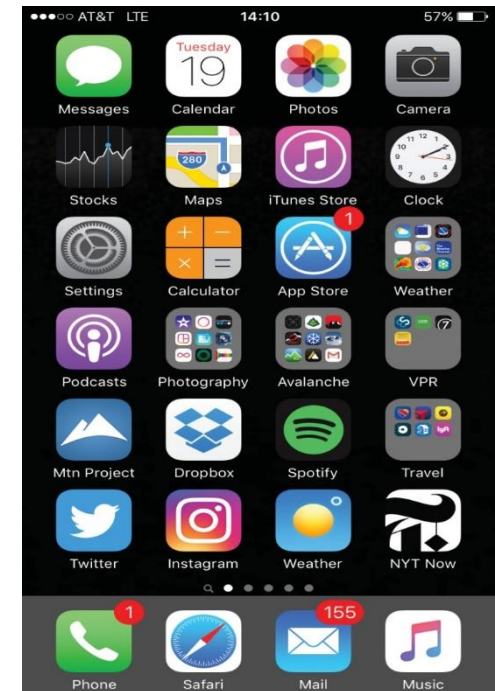
```

Graphical User Interfaces (GUI)

- A second strategy for interfacing with the operating system is through a user-friendly graphical user interface or GUI. Rather than having users directly enter commands via a command-line interface, a GUI allows provides a mouse-based window-and-menu system as an interface.
- A GUI provides a desktop metaphor where the mouse is moved to position its pointer on images, or icons, on the screen (the desktop) that represent programs, files, directories, and system functions.
- Depending on the mouse pointer's location, clicking a button on the mouse can invoke a program, select a file or directory—known as a folder— or pull down a menu that contains commands.
- Many systems now include both CLI and GUI interfaces
 - ✓ Microsoft Windows is GUI with CLI “command” shell
 - ✓ Apple Mac OS X is “Aqua” GUI interface with UNIX kernel underneath and shells available
 - ✓ Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)

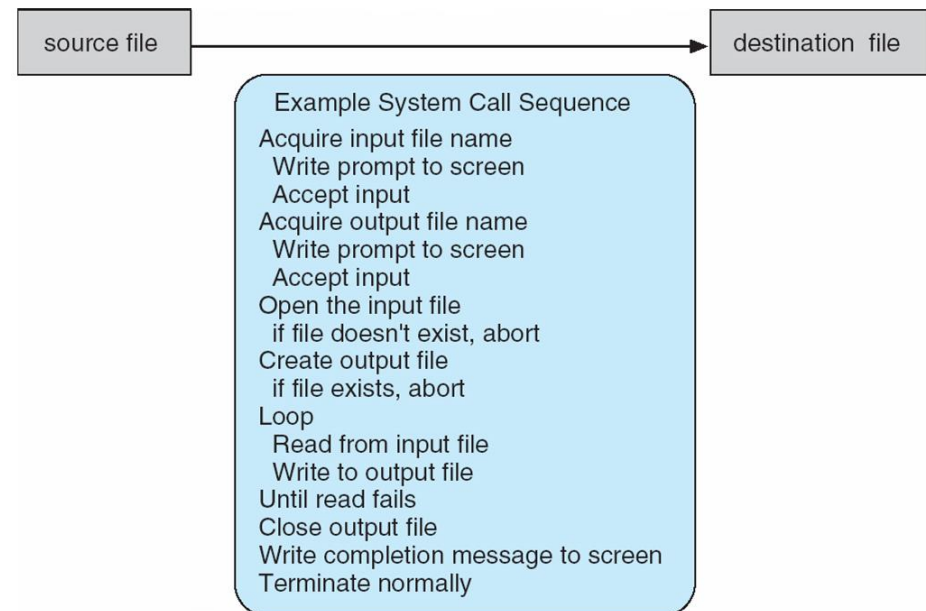
Graphical User Interfaces (GUI)

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



System Calls

- **System calls** provides an interface to the services of the operating system.
- These are generally written in C or C++, although some are written in assembly for optimal performance.
- The figure illustrates the sequence of system calls required to copy a file content from one file (input file) to another file (output file).



System Calls

- An example to illustrate how system calls are used: writing a simple program to read data from one file and copy them to another file
 - There are number of system calls used to finish this task. The first system call is to write a message on the screen (monitor). Then to accept the input filename. Then another system call to write message on the screen, then to accept the output filename.
 - When the program tries to open the input file, it may find that there is no file of that name or that the file is protected against access. In these cases, the program should print a message on the console (another system call) and then terminate abnormally (another system call) and create a new one (another system call).
 - Now that both the files are opened, we enter a loop that reads from the input file (another system call) and writes to output file (another system call).
 - Finally, after the entire file is copied, the program may close both files (another system call), write a message to the console or window (system call), and finally terminate normally (final system call).

System Calls

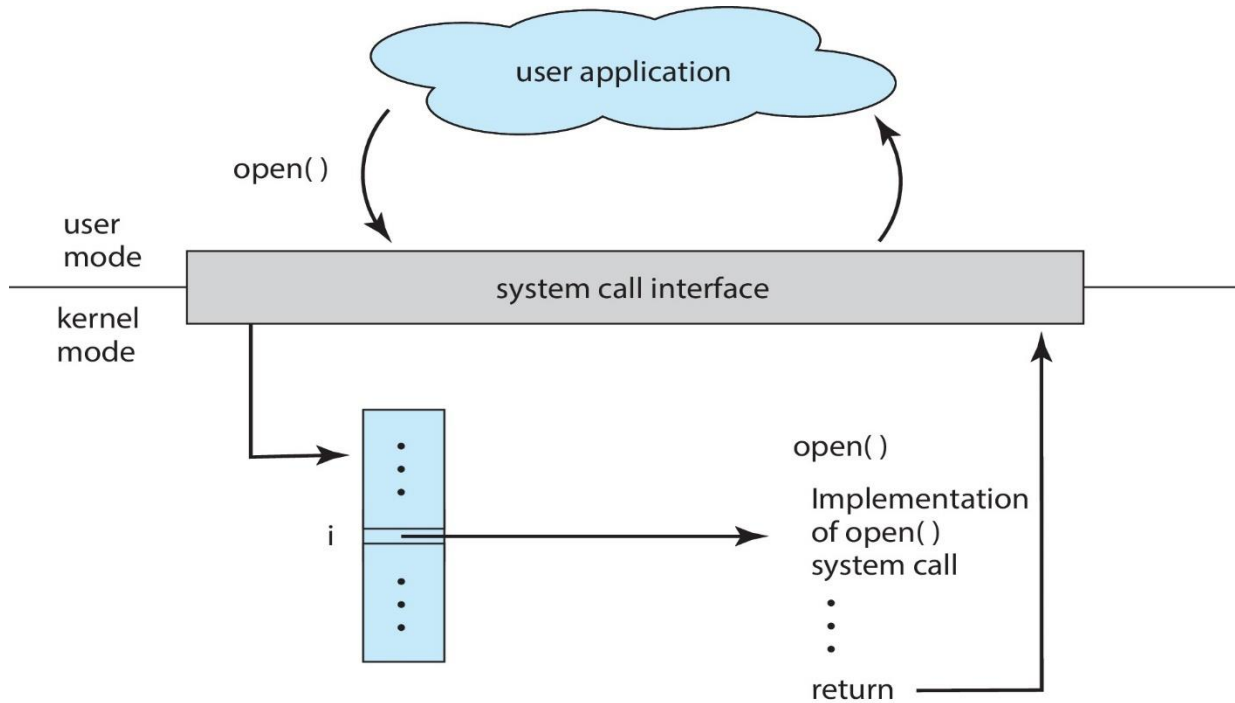


Fig. The handling of a user application invoking the open() system call

System Calls - Parameter Passing

- Three general methods used to pass parameters to OS are –
 - To pass parameters in registers.
 - If parameters are large blocks, address of block (where parameters are stored in memory) is sent to OS in the register. (Linux & Solaris).
 - Parameters can be pushed onto the stack by program and popped off the stack by OS.

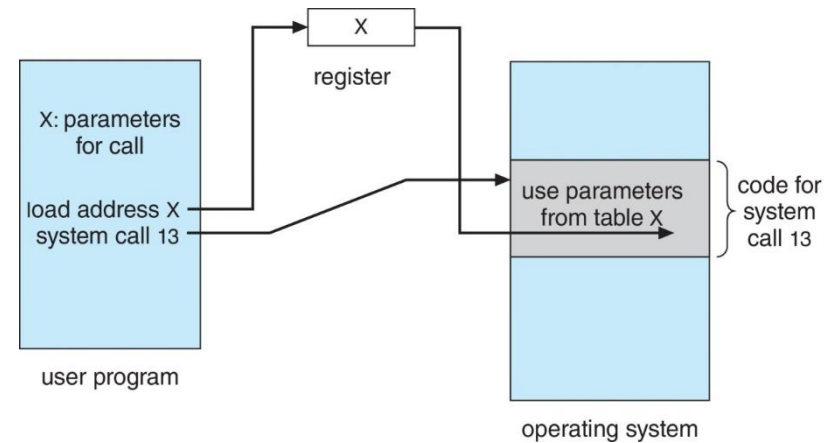


Fig. Passing of parameters as a table

Types of System Calls

- The system calls can be categorized into six major categories:

1. Process Control
2. File management
3. Device management
4. Information management
5. Communications
6. Protection

- 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

- **Process Control**

- Process control system calls include end, abort, load, execute, create process, terminate process, get/set process attributes, wait for time or event, signal event, and allocate and free memory.
- Processes must be created, launched, monitored, paused, resumed, and eventually stopped.
- When one process pauses or stops, then another must be launched or resumed.
- Process attributes like process priority, max. allowable execution time etc. are set and retrieved by OS.
- After creating the new process, the parent process may have to wait (wait time), or wait for an event to occur (wait event). The process sends back a signal when the event has occurred (signal event).

Types of System Calls

- **File Management**

- The file management functions of OS are –
 - File management system calls include create file, delete file, open, close, read, write, reposition, get file attributes, and set file attributes.
 - After creating a file, the file is opened. Data is read or written to a file.
 - The file pointer may need to be repositioned to a point.
 - The file attributes like filename, file type, permissions, etc. are set and retrieved using system calls.
 - These operations may also be supported for directories as well as ordinary files.

Types of System Calls

- **Device Management**

- Device management system calls include request device, release device, read, write, reposition, get/set device attributes, and logically attach or detach devices.
- When a process needs a resource, a request for resource is done. Then the control is granted to the process. If requested resource is already attached to some other process, the requesting process has to wait.
- In multiprogramming systems, after a process uses the device, it has to be returned to OS, so that another process can use the device.
- Devices may be physical (e.g. disk drives), or virtual/abstract (e.g. files, partitions, and RAM disks).

Types of System Calls

- **Information Maintenance**

- Information maintenance system calls include calls to get/set the time, date, system data, and process, file, or device attributes.
- These system calls are used to transfer the information between user and the OS. Information like current time & date, no. of current users, version no. of OS, amount of free memory, disk space etc. are passed from OS to the user.

Types of System Calls

- **Communication**

- Communication system calls create/delete communication connection, send/receive messages, transfer status information, and attach/detach remote devices.
- The **message passing model** must support calls to:
 - Identify a remote process and/or host with which to communicate.
 - Establish a connection between the two processes.
 - Open and close the connection as needed.
 - Transmit messages along the connection.
 - Wait for incoming messages, in either a blocking or non-blocking state.
 - Delete the connection when no longer needed.

Types of System Calls

- Communication

- The **shared memory model** must support calls to:
 - Create and access memory that is shared amongst processes (and threads)
 - Free up shared memory and/or dynamically allocate it as needed.
- Message passing is simpler and easier, (particularly for inter-computer communications), and is generally appropriate for small amounts of data. It is easy to implement, but there are system calls for each read and write process.
- Shared memory is faster, and is generally the better approach where large amounts of data are to be shared. This model is difficult to implement, and it consists of only few system calls.

Types of System Calls

- **Protection**

- Protection provides mechanisms for controlling which users / processes have access to which system resources.
- System calls allow the access mechanisms to be adjusted as needed, and for non- privileged users to be granted elevated access permissions under carefully controlled temporary circumstances.

Types of System Calls

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

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

	Windows	Unix
Process control	CreateProcess() 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() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

System Programs

- A collection of programs that provide a convenient environment for program development and execution (other than OS) are called **system programs** or **system utilities**.
- System programs may be divided into following categories:
- **File management** - programs to create, delete, copy, rename, print, list, and generally manipulate files and directories.
- **Status information** - Utilities to check on the date, time, number of users, processes running, data logging, etc. System registries are used to store and recall configuration information for particular applications.
- **Programming-language support** - E.g. Compilers, linkers, debuggers, profilers, assemblers, library archive management, interpreters for common languages, and support for make.

System Programs

- **File modification** - e.g. text editors and other tools which can change file contents.
- **Program loading and execution** - loaders, dynamic loaders, overlay loaders, etc., as well as interactive debuggers.
- **Communications** - Programs for providing connectivity between processes and users, including mail, web browsers, remote logins, file transfers, and remote command execution.

Operating-System 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: batch, time shared, single user, multiuser, distributed, real time, or general purpose.
- Beyond this highest design level, the requirements may be much harder to specify. The requirements can, however, be divided into two basic groups
 1. User goals (User requirements)
 2. System goals (System requirements)
- **User requirements** are the features that user care about and understand like system should be convenient to use, easy to learn, reliable, safe and fast.
- **System requirements** are written for the developers, ie. People who design the OS. Their requirements are like easy to design, implement and maintain, flexible, reliable, error free and efficient.

Operating-System Design and Implementation

- **Mechanisms and Policies**

- Policies determine what is to be done. Mechanisms determine how it is to be implemented.
- Example: in timer- counter and decrementing counter is the mechanism and deciding how long the time has to be set is the policies.
- Policies change overtime. In the worst case, each change in policy would require a change in the underlying mechanism.
- If properly separated and implemented, policy changes can be easily adjusted without re- writing the code, just by adjusting parameters or possibly loading new data / configuration files.

Operating-System Design and Implementation

- **Implementation**

- Traditionally OS were written in assembly language. In recent years, OS are written in C, or C++. Critical sections of code are still written in assembly language.
- The first OS that was not written in assembly language, it was the Master Control Program (MCP).
- The advantages of using a higher-level language for implementing operating systems are: The code can be written faster, more compact, easy to port to other systems and is easier to understand and debug.
- The only disadvantages of implementing an operating system in a higher-level language are reduced speed and increased storage requirements.

Operating-System Structure

- **Simple Structure**

- Many operating systems do not have well-defined structures. They started as small, simple, and limited systems and then grew beyond their original scope. Eg: MS-DOS.
- In MS-DOS, the interfaces and levels of functionality are not well separated. Application programs can access basic I/O routines to write directly to the display and disk drives. Such freedom leaves MS-DOS in bad state and the entire system can crash down when user programs fail.
- UNIX OS consists of two separable parts: the kernel and the system programs. The kernel is further separated into a series of interfaces and device drivers. The kernel provides the file system, CPU scheduling, memory management, and other operating-system functions through system calls.

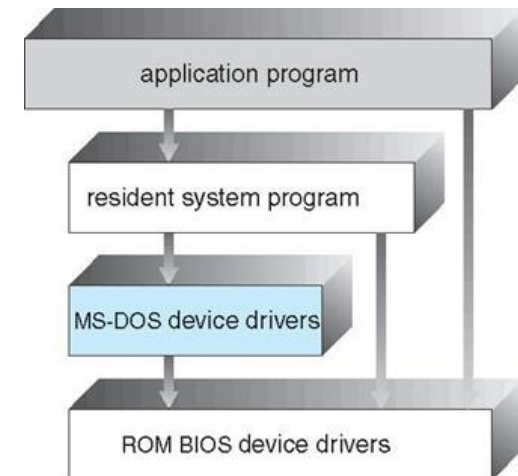


Fig. MS-DOS layer structure

Operating-System Structure

- **Layered Approach**

- The OS is broken into number of layers (levels). Each layer rests on the layer below it, and relies on the services provided by the next lower layer.
- Bottom layer (layer 0) is the hardware and the topmost layer is the user interface.
- A typical layer, consists of data structure and routines that can be invoked by higher-level layer.
- Advantage of layered approach is simplicity of construction and debugging.
- The layers are selected so that each uses functions and services of only lower-level layers. So simplifies debugging and system verification. The layers are debugged one by one from the lowest and if any layer doesn't work, then error is due to that layer only, as the lower layers are already debugged. Thus, the design and implementation are simplified.

Operating-System Structure

- **Layered Approach**

- A layer need not know how its lower-level layers are implemented. Thus hides the operations from higher layers.

- **Disadvantages of layered approach:**

- The various layers must be appropriately defined, as a layer can use only lower-level layers.
- Less efficient than other types, because any interaction with layer 0 required from top layer. The system call should pass through all the layers and finally to layer 0. This is an overhead.

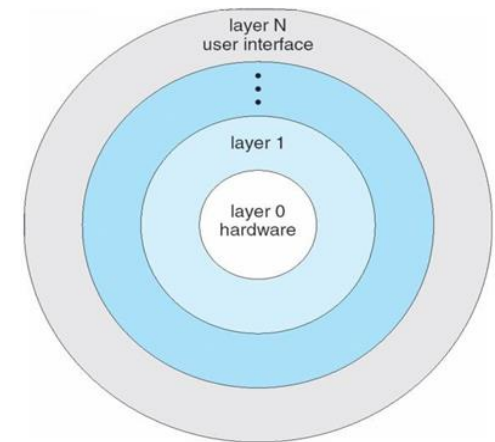


Fig. A layered Operating System

Operating-System Structure

- **Microkernels**

- This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs thus making the kernel as small and efficient as possible.
- The removed services are implemented as system applications.
- Most microkernels provide basic process and memory management, and message passing between other services.
- The main function of the microkernel is to provide a communication facility between the client program and the various services that are also running in user space.

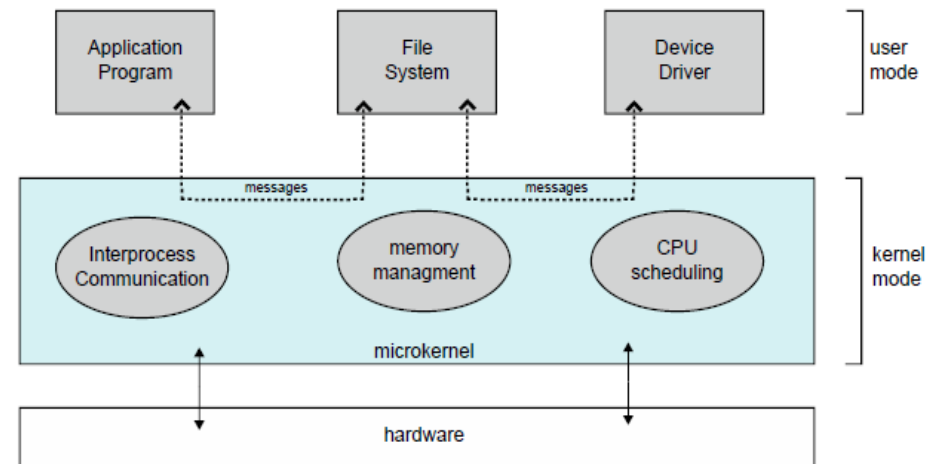
Operating-System Structure

- **Benefit of microkernel –**

- System expansion can also be easier, because it only involves adding more system applications, not rebuilding a new kernel.
- Mach was the first and most widely known microkernel, and now forms a major component of Mac OSX.

- **Disadvantage of Microkernel -**

- Performance overhead of user space to kernel space communication.



Operating-System Structure

- **Modules**

- Modern OS development is object-oriented, with a relatively small core kernel and a set of modules which can be linked in dynamically.
- Modules are similar to layers in that each subsystem has clearly defined tasks and interfaces, but any module is free to contact any other module, eliminating the problems of going through multiple intermediary layers.
- The kernel is relatively small in this architecture, similar to microkernels, but the kernel does not have to implement message passing since modules are free to contact each other directly. Eg: Solaris, Linux and Mac OSX.
- The Max OSX architecture relies on the Mach microkernel for basic system management services, and the BSD kernel for additional services. Application services and dynamically loadable modules (kernel extensions) provide the rest of the OS functionality.

Operating-System Structure

• Modules

- Resembles layered system, but a module can call any other module.
- Resembles microkernel, the primary module has only core functions and the knowledge of how to load and communicate with other modules.

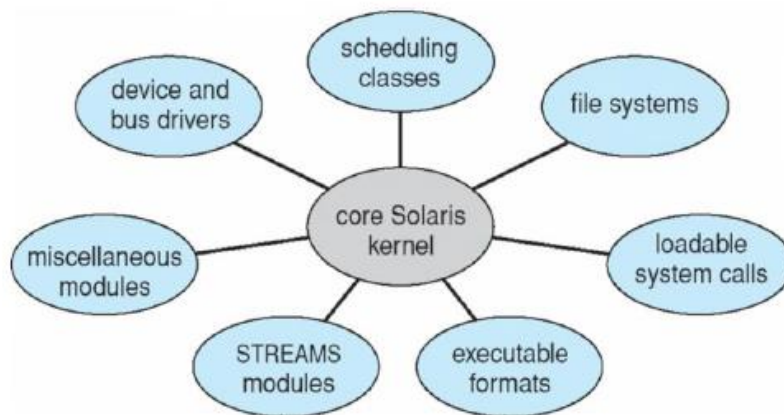


Fig. Solaris loadable modules

Question Bank

From Model Question Paper

- | | |
|---|---|
| 1 | Define operating Systems. Explain the dual-mode operating system with a neat diagram.
7 marks |
|---|---|

From Previous year question papers

- | | |
|---|--|
| 1 | Define operating Systems. Explain the dual-mode operating system with a neat diagram.
Dec'23/Jan'24 ,5 marks |
| 2 | Distinguish between the following terms.
(i) Multiprogramming and Multitasking
(ii) Multiprocessor System and Clustered System
Dec'23/Jan'24, 10 marks |

From Previous year question papers

3	Describe the differences between symmetric and asymmetric multiprocessing. What are the three advantages of multiprocessing system? Jan/Feb'2021 , 6 marks
4	Define an operating system. List the operating systems responsibilities in connection with process management and memory management. Aug/Sep'2020, 6 marks
5	Discuss the types of multiprocessor systems and advantages of multiprocessor systems. Jun/july'19 ,6 marks
6	Define operating system. Explain different views of operating system. Dec'18/Jan'19, 8 marks
7	Define operating system. Explain the role of OS with user and system viewpoints. Jun/July'18, 7 marks
8	What is an operating system? Explain the abstract view of component of a computer system. Dec'17/Jan'18, 7 marks

Question Bank

From Model Question Paper

- | | |
|---|--|
| 1 | Explain the services of the operating system that are helpful for the user and the system.
8 marks |
| 2 | What are system calls? Briefly point out its types with illustrations. 6 marks |
| 3 | Explain the layered approach of operating system structure with a supporting diagram.
6 marks |
| 4 | With a neat diagram, explain the concept of the virtual machine. 7 marks |

From Previous year question papers

- | | |
|---|--|
| 1 | Explain the operating system services with respect to users and system.
Dec'23/Jan'24 ,5 marks |
| 2 | What are system calls? List and explain different types of system calls.
Dec'23/Jan'24, 10 marks |
| 3 | With a neat diagram explain the concept of VM-WARE architecture.
Dec'23/Jan'24, 5 marks |

From Previous year question papers

4	Which calls provide an interface to the services made available by OS? Explain the same with suitable example. June/July 2019 6 marks
5	Explain the method of OS design which involves using object-oriented programming technique with a neat diagram. June/July 2019 8 marks
6	What are virtual machines? Explain VM-WARE architecture with a neat diagram. Dec'18/Jan'19 8 marks
7	Explain the concept of virtual machines. Bring out its advantages. Dec'17/Jan'18 7 marks
8	What are virtual machines? How are they implemented? Dec'16/Jan'17 5 marks
9	Explain the advantages of layered approach with a neat diagram. Dec'13/Jan'14 6 marks