SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES
(AUTONOMOUS)

MCA DEPARTMENT

LECTURE NOTES

COURSE : OPERATING SYSTEMS

YEAR / BRANCH: I MCA

REGULATION: R24

PREPARED BY : Mrs. G. GEETHA

"An Operating System is a program that acts as an intermediary between a user of a computer and the computer hardware."

— Abraham Silberschatz

SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES (AUTONOMOUS)

MCA DEPARTMENT

I MCA–II SEMESTER			
COURSECODE:	**24MCA121**	CREDITS:	4
COURSETITLE:	**OPERATING SYSTEMS**	L-T-P:	3-1-0
PREREQUISITES: Basic knowledge on" Computer Organization"			

COURSE EDUCATIONAL OBJECTIVES:

CEO1 To be aware of the evolution and fundamental principles of operating system, processes and their communication.

CEO2 To understand the various operating system components like process management and memory management.

CEO3 To know about file management and the distributed file system concepts in operating systems.

CEO4 To be aware of components of operating system with relevant case study.

UNIT - I: OPERATING SYSTEMS INTRODUCTION *LectureHrs:10*

Definition & Views of OS- Operating Systems objectives and functions- Computer System Architecture - OS Structure - OS Operations. Evolution of Operating Systems: Simple Batch - Multi programmed - Time- shared- Parallel – Distributed Systems - Real-Time Systems –Operating System services - User OS Interface- System Calls - Types of System Calls - System Boot.

UNIT-II : PROCESS CONCEPTS ANDCPUSCHEDULING *LectureHrs:12*

Process Concepts: The Process - Process State - Process Control Block - Processes & Threads. Process Scheduling Principle: Scheduling Queues – Schedulers - Context Switch - Preemptive Scheduling – Dispatcher - Scheduling Criteria. CPU Scheduling: Scheduling algorithms –FCFS – SJF – Priority - Round Robin - Multi level Queue – Multiple processors.

UNIT-III : PROCESS COORDINATION &DEADLOCK *LectureHrs:12*

Process Coordination : Synchronization Background - The Critical Section Problem - Peterson's solution - Synchronization Hardware–Semaphores-Classic Problems of Synchronization. Deadlocks: System Model- Deadlock Characterization- Methods for Handling Deadlocks –Deadlock Prevention- Deadlock Avoidance- Deadlock Detection and Recovery from Deadlock.

UNIT- IV:MASS STORAGE STRUCTURE& MEMORY MANAGEMENT *LectureHrs:12*

Mass Storage Structure: Overview of Mass Storage Structure - Disk Structure - Disk Attachment - Disk Scheduling - Disk Management. Memory Management: Logical & Physical Address Space – Swapping- Contiguous Memory Allocation – Paging- Structure of Page Table – Segmentation - Page Replacement Algorithms.

UNIT - V: FILESYSTEM *LectureHrs:12*

File System Interface: The Concept of a File - Access methods – Directory & Disk Structure - File System Mounting - File Sharing – File System Implementation. Case Studies: The Linux System-Linux History-Design Principles.

Windows 2000 Operating system-History-Design Principles.

TEXT BOOKS:

1. Operating System Principles, Abraham Silberchatz, Peter B. Galvin, Greg Gagne, Wiley Student Edition, 8/e , 2009.
2. Operating Systems–Internals and Design Principles, 6/e,2008, W.Stallings,Pearson Education.

REFERENCE BOOKS:

1. Operating Systems-A a concept based Approach,2/e,2006,D.M.Dhamdhere,TMH,NewDelhi.
2. Operating Systems,3/e,2007,Deitel & Deitel, Pearson Education, New Delhi.
3. Operating Systems- A Modern Perspective, 2/e,2002, GaryNutt, Pearson Education.
4. Operating Systems-Design & Implementation, 3/e, 2007, Andrew S Tanenbaum, Pearson Education, New Delhi.
5. Principles of Operating Systems,1/e,2010,VRamesh, Laxmi Publications, New Delhi.

COURSE OUTCOMES: *On successful completion of this course, students will be able to:*		Pos related to COs
CO1	Demonstrate the basic knowledge of operating system components And services	PO1
CO2	Relate the different Process concepts and CPU scheduling Algorithms	PO1,PO2,PO3, PO8
CO3	Illustrate the different Process Synchronization and Deadlock methodology	PO1,PO2,PO3, PO8
CO4	Compare and Contrast different memory management techniques	PO1,PO2, PO3,PO8
CO5	Examine the various File management strategies andcomparative study of various operating systems	PO1,PO2,PO3, PO4,PO8

UNIT - I

OPERATING SYSTEMS INTRODUCTION

"An operating system is a layer of software that manages system resources and provides a convenient interface for users and applications."

- Gary Nutt

Operating Systems – Introduction

Unit I Overview

This unit introduces the fundamental concepts of Operating Systems. It explains the definition, objectives, and functions of an operating system and how it acts as an interface between the user and computer hardware. The unit also covers computer system architecture, operating system structure, and OS operations. Additionally, it discusses the evolution of operating systems from simple batch systems to modern distributed and real-time systems. Students will also learn about operating system services, user interfaces, system calls, and the system boot process.

Objectives of the Unit

1. To understand the basic concept and definition of an operating system.
2. To explain the objectives and functions performed by an operating system.
3. To study different computer system architectures.
4. To understand the structure and operations of operating systems.
5. To learn the evolution and types of operating systems.
6. To understand operating system services and user interfaces.
7. To study system calls and their types.
8. To understand the system boot process.

Learning Outcomes

After completing this unit, students will be able to:

1. Define an operating system and explain its role in a computer system.
2. Describe the objectives and major functions of operating systems.
3. Explain different computer system architectures and OS structures.
4. Analyze how operating systems manage system operations.
5. Identify different types of operating systems based on their evolution.
6. Explain operating system services and user interaction methods.
7. Classify and describe different types of system calls.
8. Explain the system booting process.

Importance of Studying this Unit

1. Helps students understand the basic concept and role of an Operating System in a computer system.
2. Provides knowledge about how the operating system manages hardware and software resources.
3. Builds a strong foundation for advanced topics such as process management, memory management, and file systems.

4. Explains the evolution of operating systems, helping students understand modern computing systems.
5. Enables students to learn how users interact with the system through interfaces and system calls.
6. Helps in understanding the booting process and internal working of a computer system.
7. Improves the ability to analyze and design efficient computing environments.
8. Provides essential knowledge required for software development, system administration, and computer engineering fields.

Key Concepts Covered

1. **Definition & Views of OS**
   - OS is an interface between user and hardware.
   - Acts as a resource manager and control program.
2. **Operating System Objectives**
   - Convenience – easy to use.
   - Efficiency – better resource utilization.
   - System evolution – supports future development.
3. **Functions of OS**
   - Process management
   - Memory management
   - File and device management
   - Security and protection
4. **Computer System Architecture**
   - Consists of CPU, memory, and I/O devices.
   - Includes single processor and multiprocessor systems.
5. **OS Structure**
   - Organization of OS components.
   - Examples: monolithic, layered, microkernel.
6. **OS Operations**
   - Interrupt handling
   - Dual mode operation (user/kernel mode)
   - Resource allocation

Evolution of Operating Systems

7. **Simple Batch Systems**
   - Jobs processed sequentially without user interaction.
8. **Multiprogrammed Systems**
   - Multiple programs in memory to increase CPU utilization.
9. **Time-Sharing Systems**
   - Multiple users share CPU using time slices.
10. **Parallel Systems**
    - Multiple processors work together for faster execution.
11. **Distributed Systems**
    - Network of computers sharing resources.
12. **Real-Time Systems**
    - Immediate response required within strict deadlines.

OS Interaction

13. Operating System Services

- Program execution
- I/O operations
- File management
- Communication and security

14. User OS Interface

- Interaction through CLI or GUI.

15. System Calls

- Requests made by programs to the OS for services.

16. Types of System Calls

- Process control
- File management
- Device management
- Communication

17. System Boot

- Startup process that loads OS into memory.

Definition & Views of OS

An operating system is a framework that enables user application programs to interact with system hardware.The operating system does not perform any functions on its own, but it provides an atmosphere in which various programs and apps can do useful work.

Viewpoints of the Operating System

The operating system may be observed from the point of view of the user or the system, and it is known as the user view and the system view.

There are mainly two types of views of the operating system.

- User view
- System view

User view

The user viewpoint focuses on how the user interacts with the operating system through the usage of various application programs. Some systems are designed for a single user to monopolize the resources to maximize the user's task. Therefore the Operating system is designed primarily for ease of use with little emphasis on quality and none of resource utilization.

Single user view point

These systems are much more designed for a single user experience and meet the needs of single user where the performance is not given focus as the multiple user systems. Most computer users use a monitor, keyboard, printer, mouse and other accessories to operate their computer system. In some cases the system is designed to maximize the output of a single user. As a result more attention is laid on accessibility, and resource allocation is less important.

Multiple user view point

These systems are designed for multiple user experience and meet the needs of multiple user. when there is one mainframe computer and many users on their computer trying to interact with their kernels over the mainframe to each other.

The client server architecture is a good example where many clients may interact through a remote server, and the same constraints of effective use of server resources may arise.

Handled user view point

In the handled user viewpoint smart phones interact via wireless devices to perform numerous operations but they are not as efficient as a computer interface, limiting their usefulness. Smart phones have given you the best handheld technology ever. However their operating system is a great example of creating a device focused on the user's point of view. The Touch screen era has given you the best handheld technology ever.

Embedded System user view Point

The embedded system lacks a user point of view. The remote control used to turn on or off the tv is all part of an embedded system in which the electronic device communicates with another program where the user view point is limited and allows the user to engage with the application.

System view

An operating system can also be considered as a program running at all times in the background of a computer system known as the kernel and handling all the application programs. The operating system may also be viewed as just a resource allocator.

A computer system comprises various sources, such as hardware and software which must be managed effectively. The operating system is responsible for managing hardware resources and allocating them to programs and users to ensure maximum performance. In the system viewpoint the operating system is more involved with hardware services -CPU time,memory space, I/O operation , and so on.

From the system point of view, we are more focused on how the hardware has to interact with the operating system than the user. The hardware and the operating system interact with each other for the various purpose some of them are

Resource Allocation

There are many resources which present in the hardware such as register, cache, RAM, ROM, processors, I/O interaction, etc. These resources demanded by the operating system when it is asked by any application program.

This resource allocation has to be done only by the operating system which has used many techniques and strategies such that it brings the most out of its processing and memory space. There are various techniques such as paging, virtual memory, caching,etc.

The operating system allocates resources when a program needs them. When the program terminates, the resources are unallocated and allocated to other programs that need them.
There are two resource allocation techniques

- **Resource partitioning approach**

  It divides the resources in the system to many resource partitions, where each partition may include various resources -**For example** ,1MB memory, disk blocks and a printer. Then it allocates one resource partition to each user program before the program's initiation. A resource table records the resource partition and its current allocation status.

- **puddle based approach**

  In the puddle based approach there is a common puddle of resources. The operating system checks the allocation status in the resource table whenever a program makes a request for a resource. If the resource is free, it allocates the resources to the program.

Operating Systems objectives and functions

An Operating System (OS) is system software that acts as an intermediary between the user and the computer hardware.

Objectives of an Operating System

1. Convenience: To make the computer system easy for users to interact with by providing a user-friendly interface like a Graphical User Interface (GUI).
2. Efficiency: To manage resources so they are used as effectively as possible (e.g., maximizing CPU utilization).
3. Ability to Evolve: To permit the effective development, testing, and introduction of new system functions without interfering with service.
4. Resource Management: To manage and allocate hardware resources (CPU, memory, storage) fairly among various users and programs.

Key Functions with Examples

- Process Management: Manages the creation, scheduling, and deletion of processes. It decides which process gets the CPU and for how long.
  - *Example:* When you run a browser and a music player simultaneously, the OS uses CPU Scheduling (like Round Robin) to switch between them so quickly that they appear to run at the same time.
- Memory Management: Tracks which part of the main memory is currently in use and by whom. It allocates and deallocates memory space as needed.
  - *Example:* When you open a heavy application like Photoshop, the OS finds a contiguous block of RAM to load it. If RAM is full, it may use Virtual Memory on the hard drive to keep the system running.

- File Management: Organizes, stores, and retrieves data on storage devices. It handles file operations like creation, deletion, and access control.
  - *Example:* Creating a folder named "Homework" and saving a Word document inside it involves the OS mapping that logical file to a physical location on your SSD.
- Device Management: Controls hardware devices through their respective drivers. It handles the communication between the computer and peripherals.
  - *Example:* When you hit "Print," the OS sends data to a Print Spooler (a buffer), allowing you to continue working while the printer slowly processes the job.
- Security: Protects data and programs from unauthorized access through authentication and access controls.
  - *Example:* Requiring a password or fingerprint to log into a Windows or macOS account prevents unauthorized users from accessing your private files.
- Error Detection: Constantly monitors the system for hardware or software malfunctions and alerts the user.
  - *Example:* Displaying a "Blue Screen of Death" (BSOD) on Windows when a critical system error occurs to prevent further data corruption.

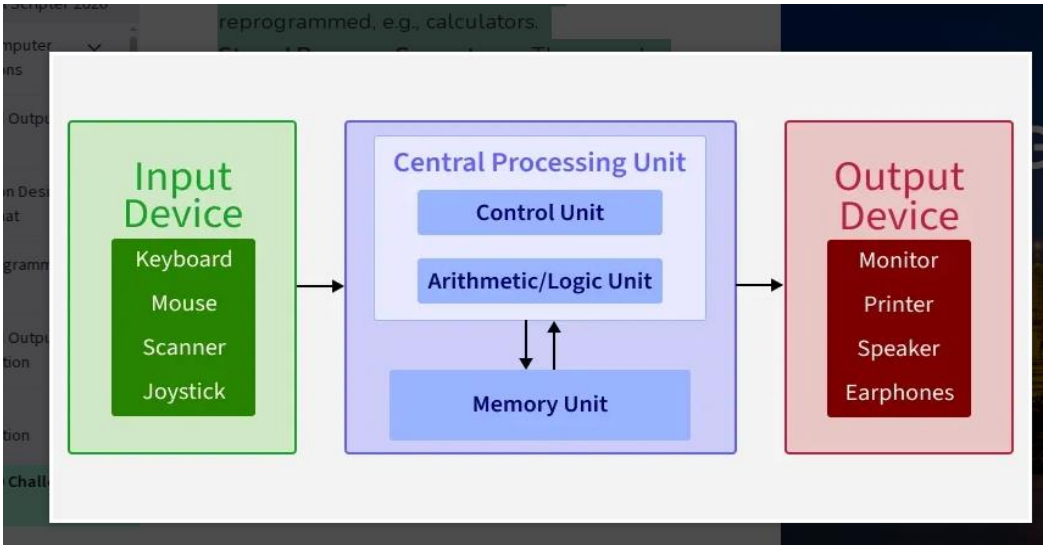
Examples of Operating Systems

- Desktop: Windows 11, macOS, and Ubuntu Linux.
- Mobile: Android and iOS.
- Specialized: RTOS (Real-Time OS) used in industrial robots or medical devices where timing is critical.

Computer System Architecture

Von Neumann architecture is a computer design where instructions and data are stored in the same memory space. This means the CPU fetches both instructions and data from the same memory, using the same pathways. Historically there have been 2 types of Computers:

- Fixed Program Computers - Their function is very specific, and they couldn't be reprogrammed, e.g., calculators.
- Stored Program Computers - These can be programmed to carry out many different tasks; applications are stored on them, hence the name.



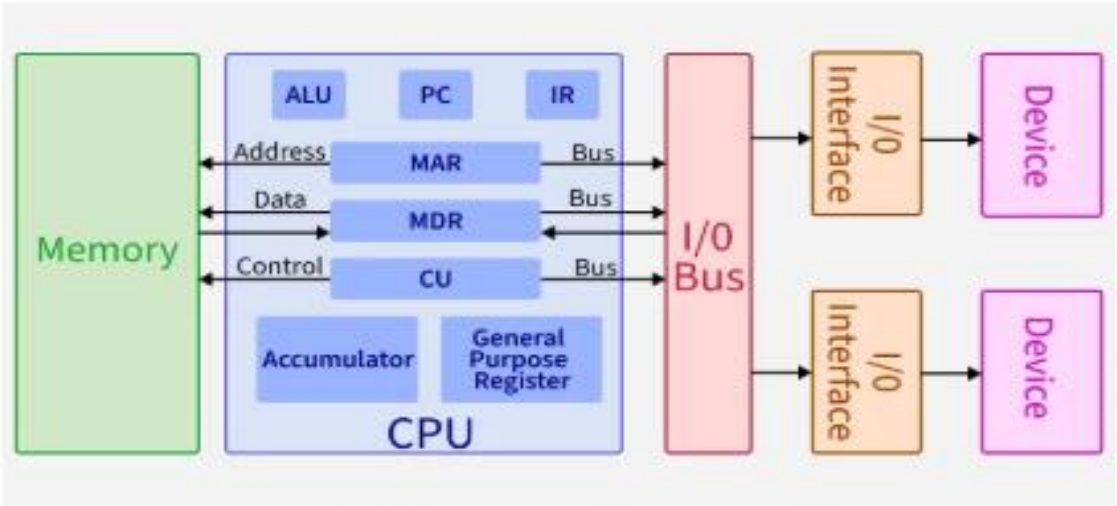
Components of Von Neumann Architecture

The structure described in the figure outlines the basic components of a computer system, particularly focusing on the memory and processor. It is made up with three maincomponents:

- CPU
- Memory
- I/O Devices

CPU (Central Processing Unit)

The central processing unit (CPU) is the main part of a computer that controls how it works. It is made up of the control unit, main memory, and the arithmetic logic unit (ALU). The CPU handles instructions from programs, processes data, stores information, and produces results. Without the CPU, a computer cannot perform tasks or run any applications.



Basic CPU structure, illustrating ALU

CU (Control Unit)

The control unit manages how the processor works by sending control signals. It decides how data should move inside the computer, controls input and output operations, and fetches the instructions from memory for execution.

ALU (Arithmetic and Logic Unit)

The arithmetic and logic unit is the part of the CPU that handles the calculations and decision-making tasks. It performs arithmetic operations like addition and subtraction, logical operations such as comparisons, and tasks like shifting bits in data.

Registers

Registers are the fastest type of memory located inside the CPU. They temporarily store information that the *processor is currently working on, making program execution and operations faster and more efficient*. Register serve as the CPU's primary working memory.

- PC (Program Counter): Keeps track of the address of the next instruction to be executed.
- IR (Instruction Register): Holds the current instruction being executed.
- MAR (Memory Address Register): Stores the address of the memory location being accessed.
- MDR (Memory Data Register): Temporarily holds data being transferred to or from memory.
- Accumulator: A register that stores intermediate results of arithmetic and logic operations.
- General Purpose Registers: Used for temporary storage of data during processing.

Bus

The bus is a communication system that *transfers data, addresses, and control signals* between the CPU, memory, and I/O devices. In Von Neumann architecture, a single bus is shared for both data and instructions, which can create a bottleneck (known as the Von Neumann bottleneck).

I/O Bus

- I/O Interface: Connects the CPU and memory to input/output devices.
- Device: Refers to external hardware like keyboards, monitors, or storage devices.

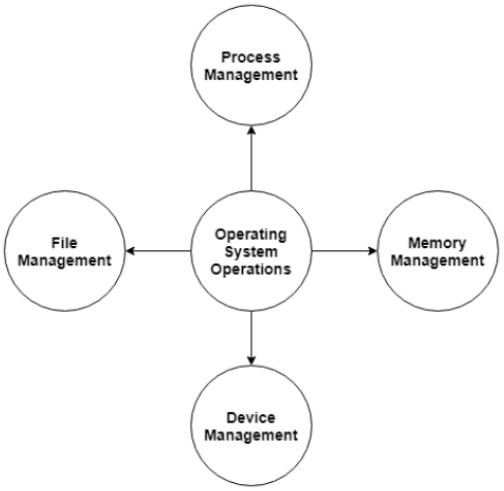
Key Characteristics

1. Single Memory for Data and Instructions: Both data and program instructions are stored in the same memory.
2. Shared Bus: A single bus is used for transferring data, addresses, and control signals, which can limit performance.
3. Sequential Execution: Instructions are executed one at a time in a sequential manner.

Operating System Operations

An operating system is a construct that allows the user application programs to interact with the system hardware. Operating system by itself does not provide any function but it provides an atmosphere in which different applications and programs can do useful work.

The major operations of the operating system are process management, memory management, device management and file management. These are given in detail as follows:



Process Management

The operating system is responsible for managing the processes i.e assigning the processor to a process at a time. This is known as process scheduling. The different algorithms used for process scheduling are FCFS (first come first served), SJF (shortest job first), priority scheduling, round robin scheduling etc.

There are many scheduling queues that are used to handle processes in process management. When the processes enter the system, they are put into the job queue. The processes that are ready to execute in the main memory are kept in the ready queue. The processes that are waiting for the I/O device are kept in the device queue.

Memory Management

Memory management plays an important part in operating system. It deals with memory and the moving of processes from disk to primary memory for execution and back again.

The activities performed by the operating system for memory management are

- The operating system *assigns memory to the processes as required*. This can be done using best fit, first fit and worst fit algorithms.

- All the memory is tracked by the operating system i.e. it nodes what memory parts are in use by the processes and which are empty.
- The operating system deallocated memory from processes as required. This may happen when a process has been terminated or if it no longer needs the memory.

Device Management

There are many I/O devices handled by the operating system such as mouse, keyboard, disk drive etc. There are different device drivers that can be connected to the operating system to handle a specific device. The device controller is an interface between the device and the device driver. The user applications can access all the I/O devices using the device drivers, which are device specific codes.

File Management

Files are used to provide a uniform view of data storage by the operating system. All the files are mapped onto physical devices that are usually non volatile so data is safe in the case of system failure.

The files can be accessed by the system in two ways i.e. sequential access and direct access −

- Sequential Access

  The information in a file is processed in order using sequential access. The files records are accessed on after another. Most of the file systems such as editors, compilers etc. use sequential access.
- Direct Access

  In direct access or relative access, the files can be accessed in random for read and write operations. The direct access model is based on the disk model of a file, since it allows random accesses.

OS Structure

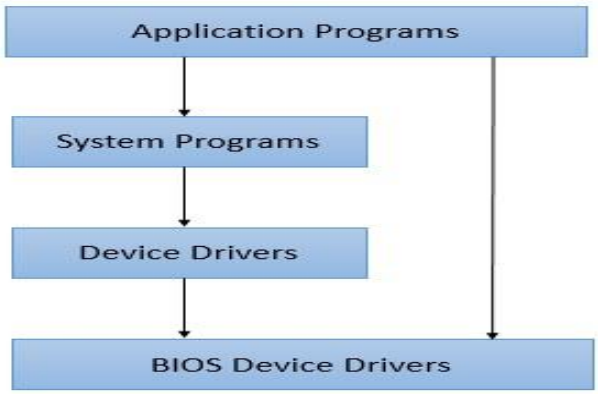
An operating system is a structure that allows the user application programs to interact with the system hardware. Since the operating system is such a complex structure, it should be created with utmost care so it can be used and modified easily. An easy way to do this is to create the operating system in parts. Each of these parts should be well defined with clear inputs, outputs and functions.

Following are various popular implementations of Operating System structures.

- Simple Structure
- Monolith Structure
- Micro-Kernel Structure
- Exo-Kernel Structure
- Layered Structure
- Modular Structure
- Virtual Machines

Simple Structure

There are many operating systems that have a rather simple structure. These started as small systems and rapidly expanded much further than their scope. A common example of this is MS-DOS. It was designed simply for a position amount for people. There was no indication that it would become so popular.



It is better that operating systems have a modular structure, unlike MS-DOS. That would lead to greater control over the computer system and its various applications. The modular structure would also allow the programmers to hide information as required and implement internal routines as they see fit without changing the outer specifications.

Advantages

Following are advantages of a simple operating system structure.

- Easy Development - In simple operation system, being very few interfaces, development is easy especially when only limited functionalities are to be delivered.
- Better Performance - Such a system, as have few layers and directly interacts with hardware, can provide a better performance as compared to other types of operating systems.

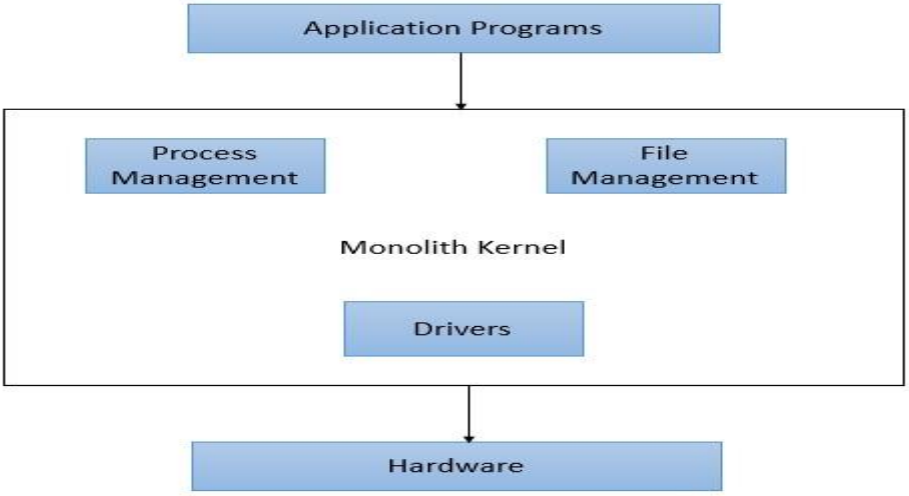
Disadvantages

- Frequent System Failures - Being poorly designed, such a system is not robust. If one program fails, entires operating system crashses. Thus system failures are quiet frequent in simple operating systems.
- Poor Maintainability - As all layers of operating systems are tightly coupled, change in one layer can impact other layers heavily and making code unmanageable over a period of time.

Monolith Structure

In monolith structured operating system, a central piece of code called kernel is responsible for all major operations of an operating system. Such operations includes file management, memory management, device management and so on. The kernal is the main component of an operating system and it provides all the services of an operating system to the application programs and system programs.

The kernel has access to the all the resources and it acts as an interface with application programs and the underlying hardware. A monolithic kernel structure promotes timesharing, multiprogramming model and was used in old banking systems.



Advantages

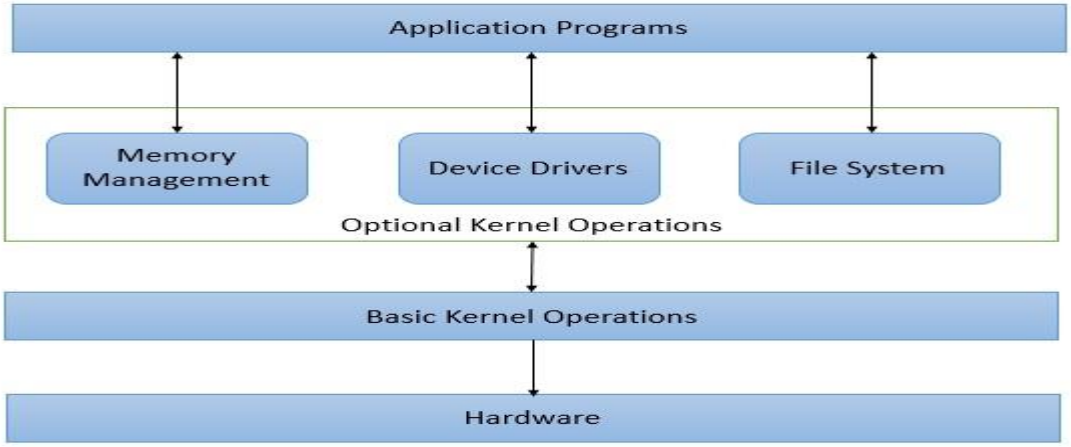
- Easy Development - As kernel is the only layer to develop with all major functionalities, it is easier to design and develop.
- Performance - As Kernel is responsible for memory management, other operations and have direct access to the hardware, it performs better.

Disadvantages

- Crash Prone - As Kernel is responsible for all functions, if one function fails entire operating system fails.
- Difficult to enhance - It is very difficult to add a new service without impacting other services of a monolith operating system.

Micro-Kernel Structure

As in case monolith structure, there was single kernel, in micro-kernel, we have multiple kernels each one specialized in particular service. Each microkernel is developed independent to the other one and makes system more stable. If one kernel fails the operating system will keep working with other kernel's functionalities.



Advantages

- Reliable and Stable - As multiple kernels are working simultaneously, chances of failure of operating sytem is very less. If one functionlity is down, operating system can still provide other functionalities using stable kernels.
- Maintainability - Being small sized kernels, code size is maintainable. One can enhance a microkernel code base without impacting other microkernel code base.

Disadvantages

- Complex to Design - Such a microkernel based architecture is difficult to design.
- Performance Degradation - Multi kernel, Multi-modular communication may hamper the performance as compared to monolith architecture.

Exo-Kernel Structure

Exo-Kernal Structured operating system was designed and developed at MIT. The aim of this design was to keep Kernel size minimal while allowing the application programs to manage hardware resources directly. The purpose of removing abstraction of operating system for hardware resources was to enable application programmer to write high performance code while exo-kernel handles other operations.

Advantages

- High Performance - As application program can allocate memory, a better designed code can make optimal use and perform better.
- Application Control - As resource management is not secured by operating system, application program has more control over system resources and can write custom operations on system resources.

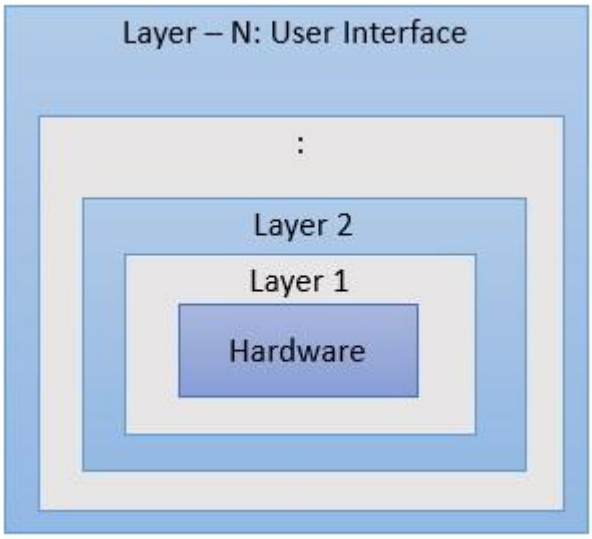
Disadvantages

- Unreliable and Unsafe - As security is in application program level, a poorly written code can damage the system.
- Complex Design - Exo-Kernel designing is complicated.

Layered Structure

One way to achieve modularity in the operating system is the layered approach. In this, the bottom layer is the hardware and the topmost layer is the user interface.

An image demonstrating the layered approach is as follows −



As seen from the image, each upper layer is built on the bottom layer. All the layers hide some structures, operations etc from their upper layers.

One problem with the layered structure is that each layer needs to be carefully defined. This is necessary because the upper layers can only use the functionalities of the layers below them.

Advantages

Following are advantages of a layered operating system structure.

- High Customizable - Being layered, each layer implmentation can be customized easily. A new functionality can be added without impacting other modules as well.
- Verifiable - Being modular, each layer can be verified and debugged easily.

Disadvantages

Following are disadvantages of a layered operating system structure.

- Less Performing - A layered structured operating system is less performing as compared to basic structured operating system.
- Complex designing - Each layer is to planned carefully as each layer communicates with lower layer only and a good design process is required to create a layered operating system.

Modular Structure

Modular structure operating system works on the similar princhiple as a monolith but with better design. A central kernal is responsible for all major operations of operating system. This kernal has set of core functionality and other services are loaded as modules dynamically to the kernal at boot time or at runtime. Sun Solaris OS is one of the example of
Modular structured operating system.

Advantages

- High Customizable - Being modular, each module implmentation can be customized easily. A new functionality can be added without impacting other modules as well.
- Verifiable - Being modular, each layer can be verified and debugged easily.

Disadvantages

- Less Performance - A modular structured operating system is less performance as compared to basic structured operating system.
- Complex designing - Each module is to planned carefully as each module communicates with kernal. A communication API is to be devised to facilitate the communication.

Virtual Machine Structure

In this kind of structure, hardware like CPU, memory, hard disks are abstracted into virtual machines. User can use them with actually configure them using execution contexts. Virtual machine takes a good amount of disk space and is to be provisioned. Multiple virtual machines can be created on a single physical machine.

Advantages

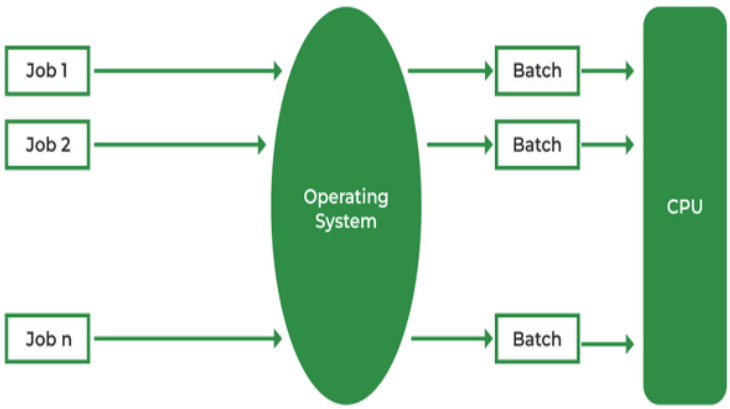
- High Customizable - Being virtual, functionality are easily accessible, can be customized on need basis.
- Secure - Being virtual, and no direct hardware access, such systems are highly secured.

Disadvantages

- Less advancement - A virtual structured operating system is less advancement as compared to modular structured operating system.
- Complex designing - Each virtual component of the machine is to planned carefully as each component is to abstract underlying hardware.

Batch-Processing Operating System

The batch-processing operating system was very popular in the 1970s. In batch operating system the jobs were performed in batches. This means Jobs having similar requirements are grouped and executed as a group to speed up processing. Users using batch operating systems do not interact with the computer directly. Each user prepares their job using an offline device **for example** a **punch card** and submits it to the computer operator.



Once the programmers have left their programs with the operator, they sort the programs with similar needs into batches.

The Batch operating system is a real-time operating system intended for batch processing. It structures a segmental architecture, which permits the addition of new segments without touching the current codebase.

A batch processing operating system (BPOS) is designed to handle and process large volumes of data in batches, making it ideal for organizations that require efficient and rapid data processing.

Unlike interactive systems, batch processing systems operate by executing a series of jobs without manual intervention, which enhances their speed and efficiency. This makes BPOS particularly suitable for businesses that consistently manage substantial data sets and need reliable, high-speed processing capabilities.

Features of Batch Processing Operating System

Batch OS is an operating system intended specifically for batch processing. It contains a command line interface, a library for scheduling tasks, and a user interface for managing tasks. Batch OS is designed to simplify the process of handling and scheduling tasks across a network of computers.

Batch OS contains a library for scheduling tasks. This library permits tasks to be scheduled in a ranked manner, which makes it easy to manage and schedule tasks across a network of computers. The user interface permits users to view and manage tasks in a graphical manner.

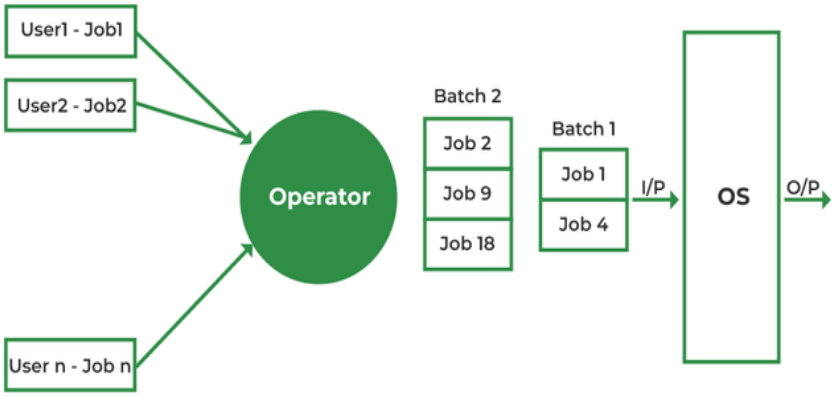
Working of Batch Processing Operating Systems

The Batch operating system is a new, open-source operating system that is being developed by the Berkeley Open Infrastructure for Network Computing (BOINC) project. A batch is a segmental operating system that can be collected from smaller pieces, allowing it to be modified to specific needs.

The batch is intended to be lightweight and efficient and is intended to be used primarily in grid computing environments.

The Batch project is presently in the progress stage, and there is still a lot of work to be done before the operating system is ready for use.

There are many types of batch operating systems. One popular type is the scheduled batch system. This type of system is used to control the execution of a series of tasks or jobs. Other types of batch systems include the interactive batch system, the real-time batch system, and the concurrent batch system.



Example of Batch Operating System

- IBM's z/OS
- Unisys MCP and Burroughs MCP/BCS

These systems are usually used in large organizations that require high-volume data processing, such as banks, airlines, and government agencies.

Advantages of Batch Operating System

The benefits of batch-processing operating systems include:

- **Resource Efficiency**: These systems improve the use of computation resources by processing jobs in groups and scheduling them during stages of resource accessibility.
- **High Throughput**: Batch processing systems can handle and complete a large number of tasks quickly, confirming quick turnaround times and high throughput.
- **Error Reduction**: Since these systems work without requiring user interference, they minimize the risk of faults that can occur with manual processing.
- **Simplified Management**: They restructure job management by automating the submission, scheduling, and implementation of tasks.
- **Cost Efficiency**: By producing well-organized use of resources and reducing processing time and errors, batch processing systems can be a cost-effective option.
- **Scalability**: These classifications can manage a huge number of tasks, making them scalable and appropriate for large organizations with significant data processing needs.

Disadvantages of Batch Operating System

There are many disadvantages to using batch operating systems, including:

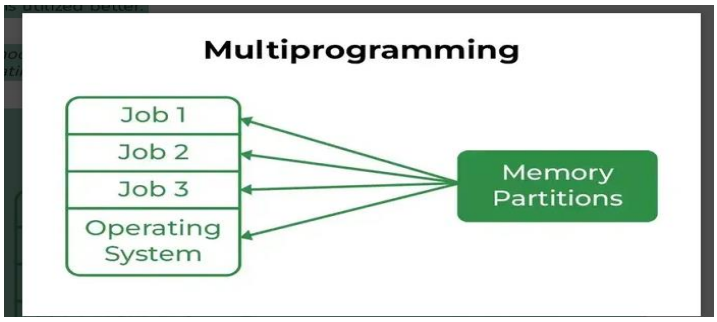
- Limited functionality: A batch operating system can solve only simple tasks not solve more complex tasks. this can make them difficult to use for certain tasks, like managing files or software.
- Security issues: Batch operating systems are not more secure because they are not typically used for day-to-day tasks, so they are not as secure as more common operating systems. This can lead to security risks if the system is used by people who should not have access to it.

- Interruptions Batch systems can be interrupted frequently, which can lead to missed deadlines or mistakes.
- Inefficiency: Batch systems are often slow and difficult to use, which can lead to inefficiency in the workplace.

Multiprogramming in Operating System

As the name suggests, Multiprogramming means more than one program can be active at the same time. Before the operating system concept, only one program was to be loaded at a time and run. These systems were not efficient as the CPU was not used efficiently.

All modern operating systems like MS Windows, Linux, etc are multiprogramming operating systems. Multiprogramming



Features of Multiprogramming

- Need Single CPU for implementation.
- Context switch between process.
- Switching happens when current process undergoes waiting state.
- CPU idle time is reduced.
- High resource utilization.
- High Performance.

Disadvantages of Multiprogramming

- If it has a large number of jobs, then long-term jobs will have to require a long wait.
- Memory management is needed in the operating system because all types of tasks are stored in the main memory.
- Using multiprogramming up to a larger extent can cause a heat-up issue.
- High Degree of Multiprogramming will cause Thrashing.
- Prior knowledge of scheduling algorithms (An algorithm that decides which next process will get hold of the CPU) is required.

These are of two types:

- *Preemptive Scheduling algorithm:* In the preemptive scheduling algorithm if more than one process wants to enter into the critical section then it will be allowed and it can enter into the critical section without any interruption only if no other progress is in the critical section.

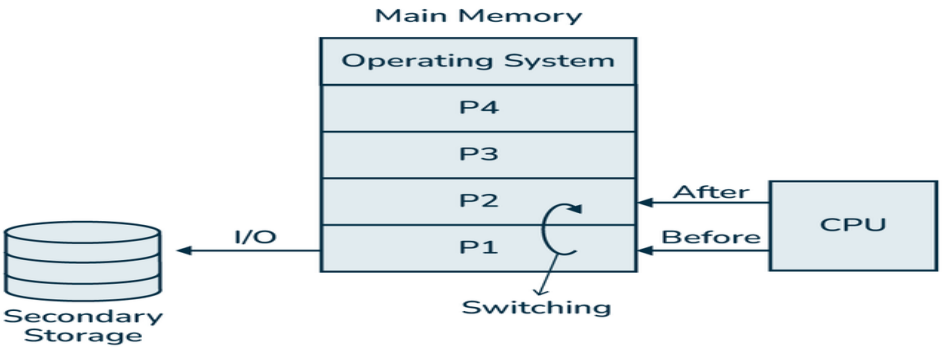
- *Non-Preemptive scheduling algorithm*: If a process gets a critical section then it will not leave the critical section until or unless it works gets done.

Working principle of Multiprogramming Operating System

In multiprogramming system, multiple programs are to be stored in memory and each program has to be given a specific portion of memory which is known as process. The operating system handles all these process and their states. Below are the steps occur when the process undergoes execution:

- The operating system selects a ready process by checking which one process should undergo execution.
- When the chosen process undergoes CPU execution, it might be possible that in between process need any input/output operation.
- At that time process goes out of main memory for I/O operation and temporarily stored in secondary storage.
- Then, CPU switches to next ready process.
- When the process which undergoes for I/O operation comes again after completing the work
- Then, CPU switches to this process.
- This switching is happening so fast and repeatedly that creates an illusion of simultaneous execution.

This switching is happening so fast and repeatedly that creates an illusion of simultaneous execution.



Working of Multi-Programming

Advantages of Multiprogramming

- Increase the resource utilization of a computer system.
- Support multiple simultaneously interactive users (terminals).
- Keeps multiple runnable jobs loaded in memory .
- Increase the throughput of the system.
- The waiting time for a program is reduced.
- Improve system's overall responsiveness.
- Improve the reliability and stability of the system.
- Make the system suitable for multitasking environments.

Time-Sharing Operating Systems

An operating system (OS) is basically a collection of software that manages computer hardware resources and provides common services for computer programs. Operating system is a crucial component of the system software in a computer system.

Time-Sharing Operating Systems is one of the important type of operating system.

Time-sharing enables many people, located at various terminals, to use a particular computer system at the same time. Multitasking or Time-Sharing Systems is a logical extension of multiprogramming.

Processor's time is shared among multiple users simultaneously is termed as time-sharing.

The main difference between Time-Sharing Systems and Multiprogrammed Batch Systems is that in case of Multiprogrammed batch systems, the objective is to maximize processor use, whereas in Time-Sharing Systems, the objective is to minimize response time.

Multiple jobs are implemented by the CPU by switching between them, but the switches occur so frequently. So, the user can receive an immediate response.

For an example, in a transaction processing, the processor executes each user program in a short burst or quantum of computation, i.e.; if n users are present, then each user can get a time quantum. Whenever the user submits the command, the response time is in few seconds at most.

An operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems which were designed primarily as batch systems have been modified to time-sharing systems.

Advantages of Timesharing operating systems are –

- It provides the advantage of quick response.
- This type of operating system avoids duplication of software.
- It reduces CPU idle time.

Disadvantages of Time-sharing operating systems are –

- Time sharing has problem of reliability.
- Question of security and integrity of user programs and data can be raised.
- Problem of data communication occurs.

Parallel Operating System

A Parallel Operating System (POS) manages multiple processors or CPU cores simultaneously to execute complex, large-scale, or multiple applications in parallel, significantly increasing speed, throughput, and efficiency. By dividing tasks into sub-tasks, these systems enhance resource utilization, though they suffer from complex architecture, high cost, and high power consumption.

Key Concepts of Parallel Operating Systems

- **Multiprocessing**: Utilizes multiple CPUs, cores, or nodes to execute tasks simultaneously.
- **Parallelization**: Breaks down complex tasks into smaller, independent sub-tasks or sub-processes.
- **Shared/Distributed Memory**: Components often share memory to communicate, although some configurations may be distributed.
- **Resource Management**: Efficiently distributes workload across available processors, preventing bottlenecking.
- **Examples**: Linux (with specific scheduling), Unix-based server OS, and HPC systems.

Advantages of Parallel Operating Systems

- **Higher Performance & Speed**: Drastically reduces processing time by running tasks concurrently, which is ideal for complex scientific simulations.
- **Increased Efficiency & Throughput**: Maximizes hardware usage, allowing more work to be completed in the same amount of time.
- **Reliability:** If one component fails, the system can often continue functioning, providing higher fault tolerance.
- **Scalability**: Systems can be scaled up by adding more processors to manage larger loads.
- **Memory Efficiency**: Parallel systems handle memory constraints more effectively compared to single-processor systems.

Disadvantages of Parallel Operating Systems

- High Cost & Resource Usage: Requires expensive hardware, specialized infrastructure, and increased energy consumption.
- Complex Architecture & Software Development: Designing, programming, and maintaining parallel operating systems is difficult due to synchronization needs.
- Maintenance & Cooling: Requires sophisticated, constant maintenance and improved cooling techniques to handle high temperatures.
- Parallel Slowdown: If communication overhead between processors outweighs the computation speedup, the system may actually become slower.

Distributed Operating System

A Distributed Operating System (DOS) manages a group of independent, networked computers and presents them to users as a single, cohesive system. It offers high scalability, reliability, and resource sharing by distributing tasks across multiple nodes, but introduces complexity in design, security, and data consistency.

Key Concepts of Distributed Operating Systems

- Resource Sharing: Users can access hardware (CPU, disks) and software resources across different sites.
- Transparency: Hides the distributed nature of the system from the user, making remote resources appear local. Types include location, migration, and replication transparency.
- Scalability: The ability to add new nodes to the network without disrupting existing operations, adjusting capacity as needed.
- Fault Tolerance/Reliability: If one node fails, the entire system does not crash, as other nodes continue to operate.
- Transparency/Transparency: Distributing workload among multiple nodes to enhance performance.

Advantages of Distributed Operating Systems

- High Reliability & Availability: The system remains functional even if multiple components fail, ensuring high dependability.
- Load Balancing: Distributes tasks efficiently across processors to speed up data processing.
- Scalability: Allows incremental growth, as resources can be easily added.
- Resource Sharing: Facilitates collaboration by allowing users at one location to access resources at another.
- Cost-Effective: Often relies on multiple, cheaper, interconnected machines (commodity hardware) rather than one massive mainframe.

Disadvantages of Distributed Operating Systems

- Complexity & Administration: Highly difficult to design, implement, and administer compared to centralized systems.
- Security Risks: Increased, as data is distributed across multiple, potentially vulnerable locations.
- Data Inconsistency: Maintaining the same, up-to-date data across all nodes is challenging (data inconsistency).
- Network Dependency: Performance is heavily dependent on the network; if the network becomes saturated, system performance suffers.
- Less Autonomy: The rate of autonomy for individual nodes is generally low.

Real-Time Operating System (RTOS)

A Real-Time Operating System (RTOS) is a specialized OS designed for time-critical, embedded applications requiring deterministic, predictable, and rapid response to events. Key concepts include strict priority-based scheduling, low latency, task management, and resource allocation. It ensures high reliability for, e.g., automotive/industrial systems, but is costly and limited in multitasking.

Key Concepts of RTOS

- **Determinism:** The ability to guarantee a maximum time for operations and provide consistent, predictable response times.
- **Task Scheduling**: Uses prioritized, preemptive, or cooperative scheduling to prioritize critical tasks, ensuring urgent tasks run first.
- **Interrupt Handling**: Rapidly handles external interrupts, crucial for systems like airbags or medical devices.
- **Hard vs. Soft Real-Time**: Hard RTOS strictly guarantees deadline completion to prevent system failure; Soft RTOS permits occasional missed deadlines with only degraded performance.
- **Resource Allocation**: Optimized for minimal memory usage and maximum CPU utilization.

Advantages of RTOS

- **High Reliability & Predictability**: Deterministic behavior guarantees task completion within deadlines.
- **Maximum Resource Utilization**: Efficiently manages resources, ideal for constrained embedded systems.
- **Low Latency & Fast Response**: Extremely fast task switching (3-10 microseconds).
- **Embedded Friendly**: Small, compact size allows for deployment in compact devices.
- **24/7 Reliability**: Designed for consistent, non-stop operation.

Disadvantages of RTOS

- **Limited Multitasking**: Not designed for managing many tasks simultaneously compared to general-purpose OS.
- **High Cost & Complexity**: Complex algorithms and high development costs, particularly for complex software.
- **Low-Priority Task Starvation**: Low-priority tasks may wait indefinitely if high-priority tasks monopolize the CPU.
- **Complex Development**: Requires specialized skills and strict adherence to timing, making debugging difficult.
- **Minimal Memory Separation**: Limited memory protection, meaning one task crash could potentially compromise the entire system.

Operating System Services

An Operating System (OS) provides various services that act as an interface between the user and computer hardware. These services help in the efficient, safe, and convenient execution of programs.

Program execution is a fundamental service of the operating system in which the OS loads a program into main memory, starts its execution, and ensures proper termination after completion.

For example, when a user opens a media player, the OS loads the program, allocates required resources, and closes it correctly when the user exits.

Process management refers to the OS service that handles the creation, scheduling, and termination of processes. Since multiple programs run simultaneously, the OS decides which process should use the CPU at a given time.

For example, while browsing the internet and listening to music at the same time, the OS manages both processes efficiently.

Memory management is the service through which the OS allocates memory to programs and frees it after use. It also ensures that one program does not access the memory of another.

For example, when a browser is opened, the OS assigns a portion of RAM to it and releases the memory when the browser is closed.

File system management allows the OS to create, delete, read, write, and organize files and directories. The OS also controls access permissions for files.

For example, saving a document, renaming a folder, or deleting a file are all handled by the file system service.

Input/Output (I/O) device management enables the OS to control and coordinate the use of hardware devices such as keyboards, printers, disks, and monitors. The OS uses device drivers to communicate with hardware.

For example, when a document is printed, the OS sends print commands to the printer through the printer driver.

Communication services allow processes to exchange data either within the same system or over a network. This service is essential in distributed and networked systems.

For example, a web browser communicates with a remote server to load a webpage using networking services provided by the OS.

Error detection and handling is an important OS service that identifies hardware and software errors and takes corrective action to prevent system failure.

For example, if a program crashes or a disk error occurs, the OS displays an error message and safely terminates the faulty process.

Protection and security services ensure that system resources are protected from unauthorized access. The OS controls user authentication and access rights.

For example, user login passwords and file permissions in Linux are managed by the OS to protect data.

User Operating System Interface

The User Operating System Interface is the mechanism through which a user interacts with the operating system to execute commands, run programs, and manage system resources. Since users cannot communicate directly with hardware, the operating system provides an interface that acts as a bridge between the user and the system. Through this interface, users can give instructions and receive output in a convenient and understandable form. Depending on the system design and user requirements, different types of interfaces are provided.

1. Command Line Interface (CLI)

A Command Line Interface (CLI) allows users to interact with the operating system by typing text-based commands using a keyboard. Each command performs a specific operation, and the user must remember correct syntax and command structure. CLI is fast and powerful for experienced users but difficult for beginners.

Example: In Linux, typing the command ls displays the list of files in a directory. In Windows, the Command Prompt allows users to type commands like dir to view files.

2. Graphical User Interface (GUI)

A Graphical User Interface (GUI) allows users to interact with the operating system using icons, windows, menus, and a mouse or touch input. GUI is user-friendly and does not require memorizing commands, making it suitable for general users. Most modern operating systems use GUI as the primary interface.
Example: Microsoft Windows desktop, where users click icons to open files, drag folders, and use menus to perform tasks.

3.Touch-Based Interface

A Touch-Based Interface allows users to interact with the operating system using touch gestures such as tapping, swiping, and pinching. This interface is commonly used in smartphones, tablets, and embedded devices. It provides fast and insightful interaction without the need for a keyboard or mouse.

Example: Android and iOS operating systems, where users open apps by tapping icons on the screen.

System Calls

System calls are the mechanism through which a user program requests services from the operating system kernel. A user program cannot directly access hardware or critical system resources for safety and security reasons. Therefore, when a program needs to perform operations such as **file access, process creation, or device control, it uses system calls** to switch from user mode to kernel mode. The operating system then performs the requested operation and returns the result to the user program. Thus, system calls act as a **bridge between user-level programs and the operating system.**

Example:

When a program wants to read data from a file, it cannot access the disk directly. Instead, it uses a system call like read(), and the OS performs the file-reading operation.

Types of System Calls

1. Process Control System Calls

Process control system calls are used to create, execute, terminate, and manage processes. These system calls allow a program to start another program, end execution, or wait for another process to complete.

Example: In UNIX/Linux, the system call **fork()** creates a new process, and **exit()** terminates a process. When you run a program from the terminal, the OS uses these system calls internally.

2. File Management System Calls

File management system calls are used to create, open, read, write, and delete files. They also control file permissions and attributes.

Example:

The system call **open() opens a file, read() reads data from the file, write() writes data, and close() closes the file**. Saving a document in a text editor uses these system calls.

3. Device Management System Calls

Device management system calls are used to request, release, read from, and write to hardware devices. The OS uses device drivers to communicate with hardware.

Example:Printing a document involves system calls that send data to the printer device. Calls like read() and write() are also used for I/O devices.

4. Information Maintenance System Calls

Information maintenance system calls are used to get or set system information, such as time, date, and system status. They also retrieve process and file attributes.

Example: The system call time() returns the current system time. Commands that display system information internally use these system calls.

5. Communication System Calls

Communication system calls are used for inter-process communication (IPC) and network communication. They allow processes to exchange data and synchronize their actions.

Example: The system calls pipe(), send(), and receive() are used for communication between processes. A web browser communicating with a server uses these calls.

6. Protection and Security System Calls

Protection system calls are used to control access permissions and ensure system security. They help the OS protect resources from unauthorized access.

Example: The system call chmod() changes file permissions in UNIX/Linux systems.

System boot

System boot is the process of starting a computer and loading the operating system into main memory (RAM) when the power is switched on or the system is restarted. When a computer is powered on, the hardware alone cannot run user programs. Therefore, a small initial program stored in non-volatile memory starts first and gradually loads the operating system. The boot process ensures that the kernel of the operating system is loaded, initialized, and ready to accept user commands.

The booting process begins with the Power-On Self Test (POST), where the system checks basic hardware components such as RAM, keyboard, and storage devices. After successful testing, the system firmware (BIOS or UEFI) locates the bootloader from the boot device (hard disk, SSD, or USB). The bootloader then loads the operating system kernel into memory, initializes system services and device drivers, and finally presents the login screen or desktop to the user.

Example of System Boot

When you switch on a laptop with Windows OS:

1. The system performs POST to check hardware.
2. BIOS/UEFI searches for a bootable device.
3. The Windows bootloader is loaded.
4. The Windows kernel is loaded into RAM.
5. System services start, and the login screen appears.

Only after this process is completed can the user open applications and use the system.

Types of Booting

Booting is the process of starting a computer and loading the operating system into memory. Based on **how the system is started**, booting is mainly classified into Cold Booting and Warm Booting.

Cold Booting: Cold Booting refers to starting the computer from a completely powered-off state. When the power button is switched on, the system performs a full Power-On Self Test (POST) to check hardware components such as RAM, keyboard, and storage devices.

After successful testing, the BIOS/UEFI loads the bootloader, and the operating system is loaded into memory. Cold booting is usually slower because all hardware checks are performed.

Example: Turning on a desktop computer in the morning after it was shut down.

Warm Booting: Warm Booting refers to restarting the computer without turning off the power. In this type, the system does not perform all hardware checks again, so the boot process is faster than cold booting. Warm booting is often used when the system becomes slow or after installing software updates.

Example: Clicking the *Restart* option in Windows or pressing Ctrl + Alt + Delete and choosing restart.

Question No.	Questions
Unit – I :OPERATING SYSTEMS INTRODUCTION	
PART – A (Two Marks Questions)	
1	Define an Operating System.
2	What are the different views of an Operating System?
3	List the objectives of an Operating System.
4	Write any four functions of an Operating System.
5	What is Computer System Architecture?
6	What is meant by OS Structure?
7	Define OS Operations.
8	What is a Simple Batch System?
9	Define Multiprogramming System.
10	What is a Time-Sharing System?
11	What are Distributed Systems?
12	Define System Call.
13	What is System Boot?
PART – B(Ten Marks Questions)	
1	Define Operating System and explain the different views of an Operating System.
2	Explain the objectives and functions of an Operating System.
3	Describe Computer System Architecture with its components.
4	Explain the structure of an Operating System in detail.
5	Explain the operations performed by an Operating System.
6	Describe the evolution of Operating Systems.
7	Explain Simple Batch System and Multiprogramming System with examples.
8	Explain the Time-Sharing Operating System with its features.
9	Describe Parallel Systems and Distributed Systems.
10	Explain Real-Time Operating Systems and their applications.
11	Explain the services provided by an Operating System.
12	Define System Call and explain the types of System Calls.
13	Explain the System Boot process in detail.
14	Define Operating System and explain the different views of an Operating System.
15	Explain the objectives and functions of an Operating System.
16	Describe Computer System Architecture with its components.

Unit –II

PROCESS CONCEPTS AND CPU SCHEDULING

“Scheduling algorithms determine the order in which processes access the CPU for execution.”
— Andrew S. Tanenbaum

PROCESS CONCEPTS AND CPU SCHEDULING

Unit –II-Overview

This unit explains the basic concepts of processes in an operating system and how the CPU manages multiple processes efficiently. A process is a program that is currently executing, and process management is one of the main responsibilities of the operating system. This unit discusses the process life cycle, process states, process control block, and the relationship between processes and threads.

It also introduces the principles of process scheduling, which determine how the CPU selects the next process for execution. Topics such as scheduling queues, different types of schedulers, context switching, preemptive scheduling, dispatcher, and scheduling criteria are included. In addition, the unit explains several CPU scheduling algorithms such as FCFS, SJF, Priority Scheduling, Round Robin, Multilevel Queue Scheduling, and scheduling in multiple processor systems.

Objectives of the Unit

1. To understand the concept of processes in operating systems.
2. To explain the different states of a process and the process life cycle.
3. To study the role and structure of the Process Control Block (PCB).
4. To understand the relationship between processes and threads.
5. To learn the principles of process scheduling.
6. To understand the concept of context switching and scheduling criteria.
7. To study different CPU scheduling algorithms used in operating systems.

Learning Outcomes

1. Students will be able to understand and explain the concept of a process in an operating system.
2. Students will be able to identify and describe different process states and their transitions.
3. Students will be able to explain the role and importance of the Process Control Block (PCB).
4. Students will be able to distinguish between processes and threads.
5. Students will be able to understand the concept of scheduling queues and the functions of different schedulers.
6. Students will be able to describe context switching and the concept of preemptive scheduling.
7. Students will be able to evaluate and compare different CPU scheduling algorithms.

Importance of Studying this Unit

1. It helps in understanding how operating systems manage and control program execution.
2. It improves knowledge about efficient CPU utilization and system performance.
3. It provides the foundation for multitasking and multiprogramming concepts.
4. It helps students understand real-world operating system scheduling techniques.
5. It forms the base for advanced topics like process synchronization and deadlocks.
6. It helps in analyzing the efficiency of different scheduling algorithms.

Key Concepts Covered

Process Coordination

1. Synchronization Background

- Multiple processes share data simultaneously.
- Proper synchronization ensures data consistency and correct execution.

2. Critical Section Problem

- A critical section is a part of code where shared resources are accessed.
- Only one process should execute it at a time.

3. Peterson's Solution

- A software solution for two processes.
- Ensures mutual exclusion using flags and turn variables.

4. Synchronization Hardware

- Uses hardware instructions like Test-and-Set.
- Provides atomic operations for mutual exclusion.

5. Semaphores

- Synchronization tool using integer variables.
- Uses wait() and signal() operations to control access.

6. Classic Problems of Synchronization

- Standard problems used to study synchronization:
    - Producer–Consumer Problem
    - Readers–Writers Problem
    - Dining Philosophers Problem

Deadlocks

7. System Model

- Processes compete for limited system resources.
- Deadlock may occur when resources are held and requested simultaneously.

8. Deadlock Characterization

Deadlock occurs if four conditions exist:

- Mutual exclusion
- Hold and wait
- No preemption

- Circular wait

9. Methods for Handling Deadlocks

- Deadlock prevention
- Deadlock avoidance
- Deadlock detection and recovery

10. Deadlock Prevention

- Prevent at least one of the four deadlock conditions.

11. Deadlock Avoidance

- System checks resource allocation to avoid unsafe states.

12. Deadlock Detection

- System identifies whether deadlock has occurred.

13. Recovery from Deadlock

- Recover by terminating processes or preempting resources.

Definition: Process

A process is a program under execution that consists of a number of elements including, program code and a set of data. To execute a program, a process has to be created for that program. Here the process may or may not run but if it is in a condition of running then that has to be maintained by the OS for appropriate progress of the process to be gained.

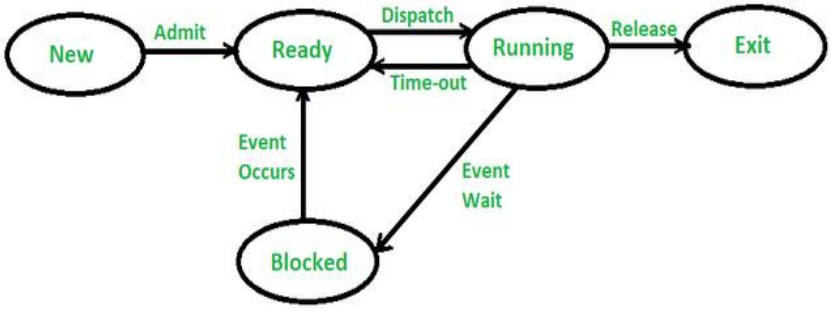
The five states that are being used in this process model are:

- **Running**: It means a process that is currently being executed. Assuming that there is only a single processor in the below execution process, so there will be at most one processor at a time that can be running in the state.
- **Ready**: It means a process that is prepared to execute when given the opportunity by the OS.
- **Blocked/Waiting:** It means that a process cannot continue executing until some event occurs like for example, the completion of an input-output operation.
- **New**: It means a new process that has been created but has not yet been admitted by the OS for its execution. A new process is not loaded into the main memory, but its **process control block (PCB**) has been created.
- **Exit/Terminate**: A process or job that has been released by the OS, either because it is completed or is aborted for some issue.

Execution of Process in Five-state Model

This model consists of five states i.e, running, ready, blocked, new, and exit. The model works when any new job/process occurs in the queue, it is first admitted in the queue after that it goes in the ready state. Now in the Ready state, the process goes in the running state. In the running state, a process has two conditions i.e., either the process goes to the event wait or the process gets a time-out.

If the process has timed out, then the process again goes to the ready state as the process has not completed its execution. If a process has an event wait condition then the process goes to the blocked state and after that to the ready state. If both conditions are true, then the process goes to running state after dispatching after which the process gets released and at last it is terminated.



Five State Process Model

Possible State Transitions

There can be various events that lead to a state transition for a process. The possible state transitions are given below:

- **Null -> New**: A new process is created for the execution of a process.
- **New -> Ready**: The system will move the process from new to ready state and now it is ready for execution. Here a system may set a limit so that multiple processes can't occur otherwise there may be a performance issue.
- **Ready -> Running:** The OS now selects a process for a run and the system chooses only one process in a ready state for execution.
- **Running -> Exit:** The system terminates a process if the process indicates that is now completed or if it has been aborted.
- **Running -> Ready:** The reason for which this transition occurs is that when the running process has reached its maximum running time for uninterrupted execution. An example of this can be a process running in the background that performs some maintenance or other functions periodically.
- **Running -> Blocked:** A process is put in the blocked state if it requests for something it is waiting. Like, a process may request some resources that might not be available at the time or it may be waiting for an I/O operation or waiting for some other process to finish before the process can continue.
- **Blocked -> Ready:** A process moves from blocked state to the ready state when the event for which it has been waiting.
- **Ready -> Exit:** This transition can exist only in some cases because, in some systems, a parent may terminate a child's process at any time.

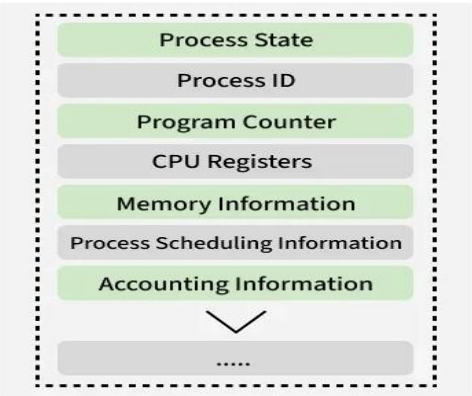
Process Control Block

A Process Control Block (PCB) is a data structure used by the operating system to keep track of process information and manage execution. It helps the OS monitor and control process execution

- Each process is given a unique Process ID (PID) for identification.
- The PCB stores details such as process state, program counter, stack pointer, open files, and scheduling info.
- During a state transition, the OS updates the PCB with the latest execution data.
- It also includes register values, CPU quantum, and process priority.
- The Process Table is an array of PCBs that maintains information for all active processes.

Structure of Process Control Block

- **Process state:** Stores whether the process is running, waiting, ready, or terminated
- **Process number Or PID**: Every process is assigned a unique id known as process ID or PID.
- **Program counter**: Program Counter stores the address of the next instruction that is to be executed for the process.
- **Register:** Registers in the PCB, it is a data structure. When a processes is running and it's time slice expires, the current value of process specific registers would be stored in the PCB and the process would be swapped out. When the process is scheduled to be run, the register values is read from the PCB and written to the CPU registers. This is the main purpose of the registers in the PCB.
- **Memory limits:** This field contains the information about memory management system used by the operating system. This may include page tables, segment tables, etc.
- **List of Open files:** This information includes the list of files opened for a process.



Applications

- Helps the operating system schedule processes and allocate CPU resources efficiently.
- Enables efficient resource utilization and sharing by providing detailed resource-usage information.
- Supports context switching through stored CPU registers and stack pointer information.
- Assists in process synchronization by keeping track of waiting states and required resources.
- Tracks process states and resource usage to determine which process should be executed next.

Processes & Threads

Processes

Processes are basically the programs that are dispatched from the ready state and are scheduled in the CPU for execution. PCB (Process Control Block) holds the context of process. A process can create other processes which are known as Child Processes. The process takes more time to terminate, and it is isolated means it does not share the memory with any other process. The process can have the following states new, ready, running, waiting, terminated and suspended.

Advantages of Processes

- Work independently in separate memory, improving security.
- Efficient allocation of CPU and memory resources.
- Can be prioritized for better task management.

Disadvantages of Processes

- Context switching can reduce system speed.
- Poor resource handling may cause deadlocks.
- Too many processes increase memory usage and management overhead.

Thread

Threads are often called "lightweight processes" because they share some features of processes but are smaller and faster. Each thread is always part of one specific process. A thread has three states: Running, Ready and Blocked.

A thread takes less time to terminate as compared to the process but unlike the process, threads do not isolate.

Examples:

When you use a mobile banking app, multiple threads work together to keep it smooth. One thread loads your latest balance, another handles your touch inputs, and another loads images or icons. Because these threads run at the same time, you can scroll and tap normally while the app continues loading data in the background.

Advantages of Threads

- Improve performance by running tasks in parallel, especially with I/O work.
- Faster to create and destroy than processes.
- Allow applications to handle multiple tasks at the same time.

Disadvantages of Threads

- Share memory, so errors in one thread can affect others.
- Shared resources may cause conflicts and unexpected behavior.
- Too many threads can reduce performance and exhaust memory.

Process Vs Thread

The table below represents the difference between process and thread.

Process	Thread
Program in execution	Part of a process
Takes more time to create & terminate	Takes less time to create & terminate
Context switching is slow	Context switching is fast
Heavyweight	Lightweight
Has its own memory space	Shares memory with other threads
Less efficient communication	More efficient communication
Blocking one process doesn't affect others	Blocking a user-level thread may block all
Uses system calls	Created using APIs (may not need OS call)
Has its own PCB, stack, address space	Shares PCB & address space, has own TCB & stack
Does not share data	Shares data with other threads

Process Scheduling Principle

Process scheduling determines which process runs on the CPU, maximizing efficiency by managing multitasking. Key principles include maximizing throughput, minimizing response/waiting time, and ensuring fairness. Scheduling is either preemptive (interruptible) or non-preemptive (run-to-completion), using algorithms like First Come, First Serve, Shortest Job First, or Round Robin to decide execution order.

Core Process Scheduling Principles

- Maximized CPU Utilization: Keeping the CPU as busy as possible.
- Maximum Throughput: Completing the maximum number of processes per time unit.
- Minimum Waiting Time: Reducing the time a process spends waiting in the ready queue.
- Minimum Response Time: Time from submission to the first response.
- Fairness: Giving each process a reasonable share of CPU time.
- Preemptive vs. Non-preemptive: Preemptive allows interrupting a process (e.g., higher priority), while Non- preemptive forces a process to finish.

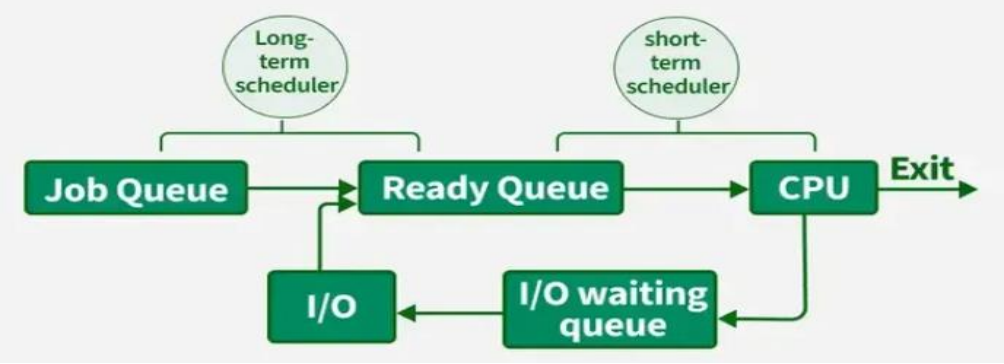
Scheduling Queues

The processes that are entering into the system are stored in the Job Queue. Suppose if the processes are in the Ready state are generally placed in the Ready Queue.The processes waiting for a device are placed in Device Queues. There are unique device queues which are available for every I/O device.

First place a new process in the job queue and then it waits in the ready queue till it is selected for execution.

Once the process is assigned to the CPU and is executing, any one of the following events occur – The process issue an I/O request, and then placed in the I/O queue.

- The process may create a new sub process and wait for termination.
- The process may be removed forcibly from the CPU, which is an interrupt, and it is put back in the ready queue.
- In the first two cases, the process switches from the waiting state to the ready state, and then puts it back in the ready queue. A process continues this cycle till it terminates, at which time it is removed from all queues and has its PCB and resources deallocated.



Core Types of Scheduling Queues

- Job Queue: Contains all processes in the system, including those waiting for memory allocation to begin execution.
- Ready Queue: Stores processes loaded into the main memory (RAM) that are prepared to execute but are awaiting CPU time.
- Device Queue (I/O Queue): A queue dedicated to a specific Input/ Output device (e.g., printer, disk). Processes wait here when they require I/O operations.

Example: A Working Scenario

Imagine a computer system performing several tasks simultaneously:

- **Job Queue**: User clicks on 3 applications (Browser, Music Player, Video Editor). All three enter the Job Queue.
- **Ready Queue**: The Operating System (Long-term scheduler) moves the Browser and Music Player into the Ready Queue in RAM.
- **CPU Execution**: The CPU picks the Browser first (FIFO) from the Ready Queue to execute.
- **Device Queue**: While playing music, the application needs to read a file from the disk. It leaves the CPU and moves to the Disk Device Queue.
- **Ready Queue (Re-entry)**: After the disk finishes reading the file, the Music Player moves back from the Device Queue to the Ready Queue, waiting for its next turn on the CPU.

Schedulers

Schedulers are special system software which handles process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Long Term Scheduler

It is also called a job scheduler. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler

It is also called as CPU scheduler. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

Medium-term scheduling is a part of swapping. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request.

A suspended process cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called swapping, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

Context Switch

Context switching is the procedure of storing the context or state of a process so that it can be reloaded when required and execution can be resumed from the same point as earlier. This is a feature of a multitasking operating system and allows a single CPU to be shared by multiple processes.

Context Switching Triggers

There are three major triggers for context switching. These are given as follows −

1. Multitasking

In a multitasking environment, a process is switched out of the CPU so another process can be run. The state of the old process is saved, and the state of the new process is loaded. On a pre-emptive system, processes may be switched out by the scheduler.

2. Interrupt Handling

The hardware switches a part of the context when an interrupt occurs. This happens automatically. Only some of the context is changed to minimize the time required to handle the interrupt.

3. User and Kernel Mode Switching

A context switch may take place when a transition between the user mode and kernel mode is required in the operating system.

Context Switching Steps

The steps involved in context switching are as follows −

- Save the context of the process that is currently running on the CPU. Update the process control block and other important fields.
- Move the process control block of the above process into the relevant queue, such as the ready queue, I/O queue, etc.
- Select a new process for execution.
- Update the process control block of the selected process. This includes updating the process state to running.
- Update the memory management data structures as required.
- Restore the context of the process that was previously running when it is loaded again on the processor. This is done by loading the previous values of the process control block and registers.

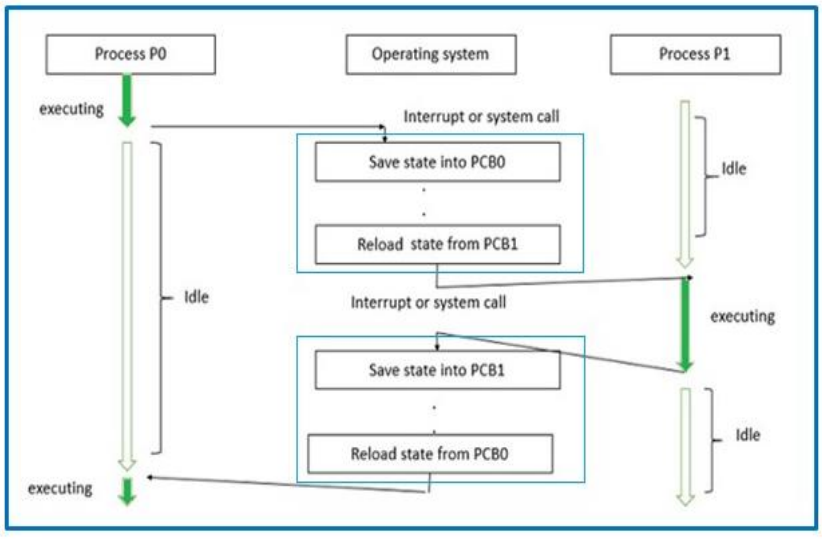
Need in Multitasking

It is essential in multitasking systems where many processes need CPU time.

- In multitasking, the CPU keeps switching between processes.
- This makes it seem like processes are running at the same time, even though the CPU works on one process at a time.
- Without context switching, one process could monopolize the CPU, and others would have to wait indefinitely

Example of Content Switching

Let us consider a scenario where context switch occurs from process P0 to process P1. The steps that occur are depicted in the following diagram .



As we can see from the diagram, two context switches occur here. The first context switch causes a switch from process P0 to P1. We can see that process P0 is initially executing.

A system call or interrupt occurs that requests P0 to be pre-empted from its executing state so that process P1 may execute. In order to do so, context switching takes place.

In order to keep the partial results of execution of P0, the state of P0 is saved in its process control block, PCB0.

The state of process P1 is reloaded to the CPU from its PCB1. Thus, P0 goes to an idle state, which P1 starts executing.

In the second context switch, the reverse procedure occurs, whereby the state of process P1 is saved in PCB1 and the saved state of P0 is loaded from PCB0.

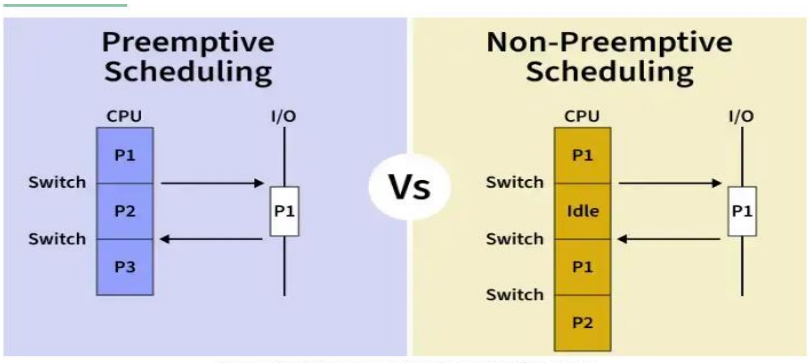
Preemptive Scheduling

CPU scheduling in operating systems is the method of selecting which process in the ready queue will execute on the CPU next. It aims to utilise the processor efficiently while minimising waiting and response times.

By determining an optimal execution order, CPU scheduling enhances overall system performance, supports smooth multitasking, and improves the user experience.

Scheduling can be broadly classified into **two types: Preemptive and Non-Preemptive.**

- ***Preemptive Scheduling****: After P1 goes for I/O, P2 comes into the CPU and utilizes it.*
- ***Non-Preemptive Scheduling:*** *After P1 goes for I/O, the CPU remains idle until P1 finishes I/O and comes back.*



Preemptive Scheduling vs Non-Preemptive Scheduling

In preemptive scheduling, the operating system can interrupt a running process to allocate the CPU to another process usually due to priority rules or time-sharing policies. A process may be moved from Running → Ready state before it finishes.

Advantages

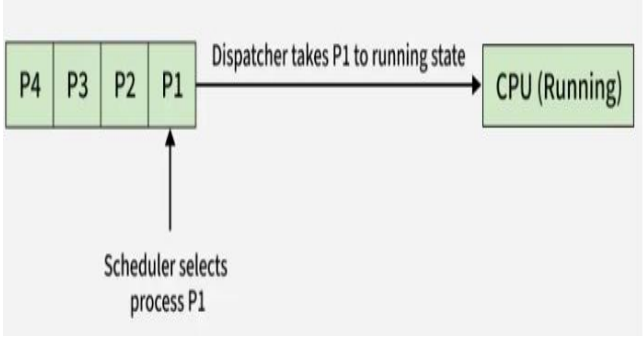
- Prevents one process from monopolizing CPU
- Better average response time in multi-user systems
- Widely used in modern OS (Windows, Linux, macOS)

Disadvantages

- More complex to implement
- Higher overhead from context switching
- Can cause starvation of low-priority processes
- Risk of concurrency issues if preempted during shared resource access.

Dispatcher

A dispatcher in an operating system is a small, high-priority module that gives control of the CPU to the process selected by the short-term scheduler. It enables multitasking by performing context switching, transitioning processes to user mode, and jumping to the correct program location to execute code.



Concepts of Dispatcher

• Process Switching: Switching the CPU from one process to another process.

• Context Switching: Saving the current process state and loading the next process state.

• CPU Allocation: Assigning CPU control to the process selected by the scheduler.

• Mode Switching: Changing the processor mode from kernel mode to user mode.

• Program Counter Update: Loading the correct instruction address so the process resumes correctly.

• Dispatcher Latency: The time taken by the dispatcher to stop one process and start another process.

Features of Dispatcher

• **Works with Short Term Scheduler**: The dispatcher operates after the CPU scheduler selects a process.

• **Fast Operation:** It performs its task quickly to reduce CPU idle time.

• **Efficient Context Switching:** Handles saving and restoring of process information efficiently.

• **Smooth Process Execution**: Ensures that processes start and resume without errors.

• **Supports Multiprogramming**: Allows multiple programs to share the CPU effectively.

• **Better CPU Utilization:** Keeps the CPU busy by assigning tasks continuously.

Importance of Dispatcher

• **Improves System Performance**: Fast switching increases overall system efficiency.

• **Enables Multitasking:** Multiple applications can run at the same time.

• **Reduces Waiting Time:** Processes receive CPU access faster.

• **Maintains Correct Execution:** Processes continue from the correct point of execution.

• **Improves System Responsiveness**: The system reacts quickly to user actions.

Example

Consider three processes in a system.

P1 – Word Processor,P2 – Web Browser, P3 – Music Player

Step 1: P1 is currently running on the CPU.

Step 2: The time slice of P1 finishes.

Step 3: The CPU scheduler selects P2.

Step 4 – Dispatcher Actions

• Saves the state of P1, • Loads the state of P2, • Switches to user mode, • Transfers CPU control

Step 5 - P2 begins execution.

Scheduling Criteria

CPU scheduling is essential for the system's performance and ensures that processes are executed correctly and on time. Different CPU scheduling algorithms have other properties and the choice of a particular algorithm depends on various factors. Many criteria have been suggested for comparing CPU scheduling algorithms.

1. CPU utilization: The main objective of any CPU scheduling algorithm is to keep the CPU as busy as possible. Theoretically, CPU utilization can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the load upon the system.

2. Throughput: A measure of the work done by the CPU is the number of processes being executed and completed per unit of time. This is called throughput. The throughput may vary depending on the length or duration of the processes.

3. Turnaround Time: Turn-around time is the sum of times spent waiting to get into memory, waiting in the ready queue, executing in CPU and waiting for I/O.

4. Waiting Time: A scheduling algorithm does not affect the time required to complete the process once it starts execution. It only affects the waiting time of a process i.e. time spent by a process waiting in the ready queue.

Waiting Time = Turnaround Time - Burst Time.

5. Response Time: In an interactive system, turn-around time is not the best criterion. A process may produce some output fairly early and continue computing new results while previous results are being output to the user. Thus another criterion is the time taken from submission of the process of the request until the first response is produced. This measure is called response time.

Response Time = CPU Allocation Time(when the CPU was allocated for the first) - Arrival Time

6. Completion Time: The completion time is the time when the process stops executing, which means that the process has completed its burst time and is completely executed.

7. Priority: If the operating system assigns priorities to processes, the scheduling mechanism should favor the higher-priority processes.

8. Predictability: A given process always should run in about the same amount of time under a similar system load.

Non- Preemptive Scheduling Algorithms

- First Come First Served (FCFS)
- Shortest Job First (SJF)
- Longest Job First (LJF)

Preemptive Scheduling Algorithms

- *Priority Scheduling*
- *Round Robin (RR)*
- Shortest Remaining Time First (SRTF)
- Longest Remaining Time First (LRTF)
- *Multilevel Queue Scheduling*
- Multilevel Feedback Queue Scheduling

CPU Scheduling Algorithms

Definition: CPU scheduling is one of the most important functions of an operating system. In a multiprogramming environment, many processes are kept in memory at the same time. The CPU can execute only one process at a time, so the operating system must decide which process should be executed next. This decision-making process is known as CPU scheduling.

The CPU scheduler selects a process from the ready queue and allocates the CPU to it. Efficient CPU scheduling improves system performance, increases CPU utilization, reduces waiting time, and ensures fairness among processes.

Objectives of CPU Scheduling

• Maximize CPU utilization
• Increase throughput
• Reduce waiting time
• Reduce turnaround time
• Improve response time
• Ensure fairness among processes

Common CPU Scheduling Algorithms

• First Come First Serve (FCFS)
• Shortest Job First (SJF)
• Priority Scheduling

• Round Robin Scheduling

• Multilevel Queue Scheduling

• Multiple Processor Scheduling

1. First Come First Serve (FCFS)

First Come First Serve (FCFS) is one of the simplest CPU scheduling algorithms used in Operating Systems. In this method, the process that arrives first in the ready queue is allocated the CPU first. It follows the principle of FIFO (First In, First Out).

FCFS is a non-preemptive scheduling algorithm, meaning once a process gets the CPU, it will continue execution until it completes or moves to the waiting state.

Concept of FCFS Scheduling

1. Processes are executed in the order they arrive.
2. The operating system maintains a queue of processes in the ready state.
3. The CPU is allocated to the process at the front of the queue.
4. When that process finishes, the next process in the queue gets the CPU.

Flow of FCFS

Process Arrival → Ready Queue → CPU Execution → Completion

Example: Consider the following processes:

Process	Arrival Time	Burst Time
P1	0	5
P2	1	3
P3	2	8
P4	3	6

Execution Order: Since FCFS executes based on arrival time:

P1 → P2 → P3 → P4

Gantt Chart

0 5 8 16 22

| P1 | P2 | P3 | P4 |

Process	Burst Time	Completion Time
P1	5	5
P2	3	8
P3	8	16
P4	6	22

Turnaround Time Calculation: Turnaround Time = Completion Time – Arrival Time

Process	Turnaround Time
P1	5
P2	7
P3	14
P4	19

Waiting Time Calculation: Waiting Time = Turnaround Time – Burst Time

Process	Waiting Time
P1	0
P2	4
P3	6
P4	13

Advantages of FCFS

1. Simple and easy to implement.
2. Processes are executed in a fair manner based on arrival.
3. No starvation occurs because every process gets a chance.
4. Suitable for batch processing systems.
5. Low overhead in scheduling.

Disadvantages of FCFS

1. Convoy Effect:Short processes may wait for a long process to complete.
2. Poor average waiting time:Especially when the first job is large.
3. Not suitable for time-sharing systems.
4. CPU utilization may decrease.
5. Interactive processes suffer delays.

Convoy Effect :If a long process arrives first, all shorter processes must wait behind it.

Example

P1 = 24 ms

P2 = 3 ms

P3 = 3 ms

Execution Order: P1 → P2 → P3 Here, shorter processes wait unnecessarily, increasing waiting time.

Applications of FCFS

1. Batch operating systems
2. Print queue management
3. Simple task scheduling
4. Background job processing

Round Robin (RR) Scheduling

Round Robin is a preemptive CPU scheduling algorithm mainly used in time-sharing operating systems. Each process is given a small unit of CPU time called a time quantum. After the time quantum expires, the process is preempted and placed at the end of the ready queue.This ensures that every process gets CPU time in a cyclic order.

Concept of Round Robin

1. Processes are stored in a circular queue.
2. CPU is assigned to the first process in the queue.
3. The process runs for a fixed time quantum.
4. If the process finishes, it leaves the system.
5. If it does not finish, the remaining burst time goes back to the queue.
6. The next process in the queue gets the CPU.

Example: Time Quantum = 4 ms

Process	Arrival Time	Burst Time
P1	0	10
P2	1	5
P3	2	8
P4	3	6

Step -by- step Execution:

P1 executes for 4 units → remaining 6

P2 executes for 4 units → remaining 1

P3 executes for 4 units → remaining 4

P4 executes for 4 units → remaining 2

Second cycle continues until completion.

Gantt Chart

0 | P1 | 4 | P2 | 8 | P3 | 12 | P4 | 16 | P1 | 20 | P2 | 21 | P3 | 25 | P4 | 27 | P1 | 29

Completion Time

Process	Completion Time
P1	29
P2	21
P3	25
P4	27

Turnaround Time: Turnaround Time = Completion Time − Arrival Time

Process	Turnaround Time
P1	29
P2	20
P3	23
P4	24

Waiting Time: Waiting Time = Turnaround − Burst

Process	Waiting Time
P1	19
P2	15
P3	15
P4	18

Advantages

1. Fair allocation of CPU.
2. Each process gets equal opportunity.
3. Suitable for interactive systems.
4. No starvation problem.
5. Good response time for users.

Disadvantages

1. Too many context switches.
2. Performance depends on time quantum.
3. Larger waiting time compared with SJF.
4. Small quantum reduces CPU efficiency.

Applications

- Time sharing systems
- Multiuser systems
- Online transaction systems

Shortest Job First (SJF) Scheduling

Shortest Job First scheduling selects the process with the smallest CPU burst time for execution. It is known to produce the minimum average waiting time.

Types:

1. Non-Preemptive SJF
2. Preemptive SJF (Shortest Remaining Time First)

Concept

1. All processes enter the ready queue.
2. The process with the shortest burst time is selected.

3. CPU executes that process completely (non-preemptive).
4. In preemptive version, a new shorter job can interrupt.

Example

Process	Arrival Time	Burst Time
P1	0	7
P2	1	4
P3	2	1
P4	3	4

Step-by-step Selection

At time 0 → only P1 available → start execution

When other processes arrive, the scheduler selects the shortest job.

Execution order: P1 → P3 → P2 → P4

Gantt Chart

0 | P1 | 7 | P3 | 8 | P2 | 12 | P4 | 16

Completion Time:

Process	Completion Time
P1	7
P3	8
P2	12
P4	16

Turnaround Time(TAT): Turnaround Time = Completion Time − Arrival Time

Process	TAT
P1	7
P2	11
P3	6
P4	13

Waiting Time(WT): Waiting Time = Turnaround − Burst

Process	WT
P1	0
P2	7
P3	5
P4	9

Average waiting time is smaller compared to other algorithms.

Advantages

1. Minimum average waiting time.
2. Efficient CPU utilization.
3. Faster execution of short processes.
4. Suitable for batch systems.

Disadvantages

1. Long processes may suffer starvation.
2. Difficult to estimate CPU burst time.
3. Not suitable for interactive systems.
4. Implementation complexity.

Applications

- Batch operating systems
- Long-term scheduling systems

Priority Scheduling

Priority Scheduling assigns each process a priority number. The CPU is allocated to the process with the highest priority. Lower number often indicates higher priority.

Types:

1. Preemptive Priority Scheduling
2. Non-Preemptive Priority Scheduling

Concept

1. Each process is given a priority.
2. The scheduler compares priorities.
3. The highest priority process executes first.
4. If a higher priority process arrives (in preemptive), CPU is taken from current process.

Example

Process	Arrival	Burst	Priority
P1	0	8	2
P2	1	4	1
P3	2	9	4
P4	3	5	3

(Smaller number = higher priority)

Execution Order: P2 → P1 → P4 → P3

Gantt chart

0 | P2 | 4 | P1 | 12 | P4 | 17 | P3 | 26

Completion Time(CT)

Process	CT
P2	4
P1	12
P4	17
P3	26

Turnaround Time(TAT)

Process	TAT
P1	12
P2	3
P3	24
P4	14

Waiting Time(WT):

Process	**WT**
P1	4
P2	0
P3	15
P4	9

Advantages

1. Important processes get executed first.
2. Useful in real-time systems.
3. Flexible scheduling.
4. System control is easier.

Disadvantages

1. Starvation of low priority processes.
2. Priority inversion problem.
3. Complex to maintain priorities.
4. May reduce fairness.

Solution to Starvation

Aging Technique: Gradually increase priority of waiting processes so that they eventually execute.

Example: If a process waits too long, its priority improves over time.

Multilevel Queue Scheduling

Multilevel Queue Scheduling is a CPU scheduling method in which the ready queue is divided into multiple queues based on process type, priority, or characteristics. Each queue works independently and may use a different scheduling algorithm.

Processes are permanently assigned to a queue when they enter the system.

Typical classification:

- System processes
- Interactive processes
- Batch processes
- Background processes

Each queue has its own scheduling policy.

Structure of Multilevel Queue

Ready Queue is divided as follows:

Queue 1 – System Processes (Highest Priority)

Queue 2 – Interactive Processes

Queue 3 – Batch Processes

Queue 4 – Background Processes (Lowest Priority)

CPU scheduling occurs in two ways:

1. **Fixed Priority Scheduling:** Higher priority queues are executed first.
2. **Time Slicing Between Queues:** Each queue receives a fixed portion of CPU time.

Example

System queue – Round Robin

Interactive queue – Round Robin

Batch queue – FCFS

Working of Multilevel Queue Scheduling

1. When a process enters the system, it is assigned to a queue.
2. Each queue follows its own scheduling algorithm.
3. CPU first selects processes from the highest priority queue.
4. If that queue is empty, the next queue is executed.
5. Lower queues execute only when higher queues are empty.

Example

Assume three queues: Queue 1 – System – Round Robin (Time quantum = 4),

Queue 2 – Interactive – Round Robin (Time quantum = 6),

Queue 3 – Batch – FCFS

Processes

Process	Queue	Burst Time
P1	System	10
P2	Interactive	6
P3	Batch	12
P4	System	4

Execution

First System Queue executes.

P1 runs for 4 units (remaining 6)

P4 runs for 4 units (completed)

P1 runs remaining 6

Then Interactive Queue executes.

P2 runs for 6 units and completes.

Finally Batch Queue executes.

P3 runs for 12 units.

Gantt Chart

0 | P1 | 4 | P4 | 8 | P1 | 14 | P2 | 20 | P3 | 32

Advantages

1. Easy to categorize processes.
2. Important processes get CPU quickly.
3. Better response time for interactive processes.
4. Flexible scheduling techniques.
5. Efficient for systems with mixed workloads.

Disadvantages

1. Starvation may occur in lower priority queues.
2. Processes cannot move between queues.
3. Complex scheduling management.
4. CPU may be idle for lower queues.

Applications

- Time-sharing systems
- Multiuser operating systems
- Large computing environments

Multiple Processor Scheduling

Multiple Processor Scheduling is used in systems with more than one CPU. The operating system must schedule processes efficiently across several processors.

These systems are known as:

- Multiprocessor systems
- Parallel processing systems

The goal is to improve system performance, reliability, and throughput.

Types of Multiprocessor Scheduling

1**. Asymmetric Multiprocessing (AMP**)

In this method, one processor acts as a master processor.

Master processor responsibilities:

- Scheduling processes
- Managing I/O
- System control

Other processors only execute tasks assigned by the master.

Example:

CPU1 – Master

CPU2 – Worker

CPU3 – Worker

2. Symmetric Multiprocessing (SMP)

In this method, all processors are equal.

Each processor:

- Schedules processes
- Executes processes

Processes are stored in a common ready queue and processors select tasks independently.

This model is used in most modern operating systems.

Key Concepts in Multiprocessor Scheduling

- **Load Sharing:** Processes are distributed among processors to balance workload.
- **Processor Affinity:** A process tends to run on the same processor to improve cache performance.
- **Load Balancing**: Ensures all processors are used efficiently.

Example

System with two processors

Processes

Process	Burst Time
P1	10
P2	4
P3	6
P4	8

Scheduling: CPU 1 executes: P1 → P3, CPU 2 executes: P2 → P4

Execution Timeline

CPU 1: 0 | P1 | 10 | P3 | 16

CPU 2: 0 | P2 | 4 | P4 | 12

Total execution time becomes faster compared to a single CPU system.

Advantages

1. Higher system throughput.
2. Faster execution of tasks.
3. Better resource utilization.
4. Increased system reliability.
5. Supports parallel processing.

Disadvantages

1. Complex operating system design.
2. Synchronization problems.
3. Higher hardware cost.
4. Communication overhead between processors.
5. Difficult scheduling algorithms.

Applications

- Supercomputers
- Scientific simulations
- Data centers
- Cloud computing
- Modern multi-core operating systems.

Question Number	Questions
Unit - II: PROCESS CONCEPTS AND CPU SCHEDULING	
PART – A (Two Marks Questions)	
1	Define a Process and distinguish it from a program.
2	List the five primary states of a process.
3	What is the purpose of a Process Control Block (PCB)?
4	State the key difference between Processes and Threads.
5	What is meant by a Context Switch?
6	Define the role of the Dispatcher in an operating system.
7	What is the difference between Preemptive and Non-preemptive scheduling?
8	Distinguish between the Long-term scheduler and the Short-term scheduler.
9	Define Scheduling Queues and mention the three main types.
10	Define Throughput and Turnaround Time as scheduling criteria.
11	What is the Convoy Effect in FCFS scheduling?
12	Define Starvation (Indefinite Blocking) in the context of priority scheduling.
13	Explain the concept of a Time Quantum in Round Robin scheduling.
14	What is the main objective of Multi-level Queue scheduling?
15	Contrast CPU Utilization and Response Time.
PART – B(Ten Marks Questions)	
1	Diagram and describe the various states of a process and the transitions between them.
2	Explain the internal structure of a Process Control Block (PCB) and describe the function of each component.
3	Compare User-Level Threads and Kernel-Level Threads, detailing the advantages and disadvantages of each.
4	Discuss the three types of Schedulers (Long, Medium, and Short-term) and how they interact to manage the degree of multiprogramming.
5	Explain the mechanical process of a Context Switch and analyze why it is considered system overhead.
6	Critically analyze the five main CPU Scheduling Criteria used to evaluate the efficiency of an algorithm.
7	Discuss the First-Come, First-Served (FCFS) and Shortest Job First (SJF) algorithms, including their performance trade-offs.
8	Explain Priority Scheduling and describe how the "Aging" technique is used to prevent process starvation.
9	Detail the working of the Round Robin (RR) algorithm and explain how the choice of time quantum impacts system performance.
10	Differentiate between Multi-level Queue and Multi-level Feedback Queue scheduling with appropriate examples.
11	Describe the complexities and approaches involved in Multiple-Processor Scheduling, focusing on Load Balancing and Processor Affinity.
12	Provide a comparative study of Preemptive and Non-preemptive scheduling algorithms using a sample process set.
13	Explain how modern operating systems handle scheduling for Multiple Processors using Symmetric Multiprocessing (SMP).

Unit III

PROCESS COORDINATION & DEADLOCK

“Hardware support such as test-and-set instructions can be used to implement mutual exclusion.”

— William Stallings

PROCESS COORDINATION & DEADLOCK

UNIT-III : Overview

In a modern OS, multiple processes run concurrently and often share resources like memory, files, or printers. **Process Coordination** focuses on "Synchronization"—the mechanics used to prevent "Race Conditions" where the timing of execution leads to unpredictable results. **Deadlocks** represent the ultimate failure of coordination, where a set of processes is stalled because each holds a resource the other needs.

2. Objectives

- To understand the **Critical Section Problem** and the requirements for a valid solution (Mutual Exclusion, Progress, and Bounded Waiting).
- To explore software-based (Peterson's) and hardware-based synchronization tools.
- To master the use of **Semaphores** and monitors in solving classic synchronization problems.
- To identify the four necessary conditions for **Deadlock** to occur.
- To evaluate different strategies for handling deadlocks, including prevention, avoidance, and recovery.

3. Learning Outcomes

By the end of this study, student will be able to:

- Identify potential race conditions in concurrent code.
- Implement synchronization primitives like Semaphores to protect shared data.
- Solve classic problems such as the **Producer-Consumer**, **Dining Philosophers**, and **Readers-Writers** problems.
- Construct and analyze **Resource Allocation Graphs** to detect deadlocks.
- Apply the **Banker's Algorithm** to determine if a system is in a "Safe State."

4 . Importance of Studying this Unit

Without synchronization, data integrity is impossible. A bank transaction might lose money, or a database might become corrupted simply because two processes updated a value at the exact same millisecond. Similarly, deadlocks can freeze an entire system, requiring a hard reboot. Understanding these concepts is fundamental for anyone building distributed systems, multi-threaded applications, or OS kernels.

Key concepts

1. Synchronization Background –Key concepts

- **Concurrency:** Multiple processes executing simultaneously in a system.
- **Shared Resources:** Resources like memory, files, or variables accessed by multiple processes.
- **Race Condition:** Situation where the final result depends on the order of process execution.
- **Process Synchronization:** Coordination of processes to ensure correct access to shared resources.
- **Mutual Exclusion:** Only one process can access the critical resource at a time.
- **Process Cooperation:** Processes working together while sharing data.
- **Data Consistency:** Maintaining correct data values when multiple processes access shared data.

2. Critical Section Problem – Key Concepts

- **Critical Section:** Part of a program where shared resources are accessed.
- **Entry Section:** Code that requests permission to enter the critical section.
- **Exit Section:** Code that releases the resource after execution.
- **Remainder Section:** Other parts of the program not involving shared resources.
- **Requirements of Critical Section Solution:**
    - **Mutual Exclusion**
    - **Progress**
    - **Bounded Waiting**

3. Peterson's Solution – Key Concepts

- **Software-based Synchronization Method** for two processes.
- Uses **two shared variables**:
    - **Flag array** (indicates interest of a process to enter critical section)
    - **Turn variable** (decides which process gets priority).
- Ensures:
    - **Mutual Exclusion**
    - **Progress**
    - **Bounded Waiting**
- Works mainly for **two processes only**.

4. Synchronization Hardware – Key Concepts

- **Atomic Operations:** Operations executed completely without interruption.
- **Test-and-Set Instruction:** Hardware instruction used to lock a resource.
- **Compare-and-Swap:** Compares memory value and swaps it atomically.
- **Interrupt Disabling:** Temporarily prevents interrupts to ensure exclusive access.
- **Hardware Locks:** Mechanisms provided by processors for synchronization.

5. Semaphores – Key Concepts

- **Semaphore:** Integer variable used to control access to shared resources.
- **Wait (P) Operation:** Decreases semaphore value and blocks process if needed.
- **Signal (V) Operation:** Increases semaphore value and wakes waiting processes.
- **Types of Semaphores:**
    - **Binary Semaphore:** Value 0 or 1 (used for mutual exclusion).
    - **Counting Semaphore:** Value can be greater than 1 (used for multiple resources).
- **Process Blocking and Waking:** Managing processes waiting for resources.

6. Classic Problems of Synchronization – Key Concepts

Common examples used to explain synchronization techniques:

- **Producer–Consumer Problem**
    - Coordination between producer generating data and consumer using it.
- **Readers–Writers Problem**
    - Multiple readers allowed simultaneously but writers require exclusive access.
- **Dining Philosophers Problem**
    - Demonstrates deadlock and resource allocation challenges.
- **Sleeping Barber Problem**
    - Synchronization between customers and service provider.

Deadlocks – Key Concepts

7. System Model – Key Concepts

- **Processes and Resources:** Processes compete for limited system resources.
- **Resource Allocation:** Assigning resources to processes.
- **Resource Types:** Each type may have one or multiple instances.
- **Resource Request and Release:** Processes request, use, and release resources.

8. Deadlock Characterization – Key Concepts

Deadlock occurs when processes wait indefinitely for resources.

Four Necessary Conditions for Deadlock:

1. **Mutual Exclusion:** Only one process can use a resource at a time.
2. **Hold and Wait:** Process holds resources while waiting for others.
3. **No Preemption:** Resources cannot be forcibly taken away.
4. **Circular Wait:** Processes form a circular chain waiting for resources.

9. Methods for Handling Deadlocks – Key Concepts

Operating systems use different approaches:

- **Deadlock Prevention**
- **Deadlock Avoidance**
- **Deadlock Detection**
- **Deadlock Recovery**
- **Ignoring Deadlocks (Ostrich Approach)**

10 .Deadlock Prevention – Key Concepts

Techniques used to **prevent one of the four deadlock conditions**.

- Remove **Mutual Exclusion** (if possible)
- Avoid **Hold and Wait**
- Allow **Resource Preemption**
- Prevent **Circular Wait** using ordered resource allocation

11. Deadlock Avoidance – Key Concepts

- System dynamically checks resource allocation to avoid unsafe states.
- Uses **safe state concept**.
- **Banker's Algorithm** is commonly used.
- Requires knowledge of **maximum resource demand**.

12. Deadlock Detection – Key Concepts

- Allows deadlock to occur but detects it later.
- Uses **Resource Allocation Graph**.
- Detection algorithms check for **cycles in process-resource graph**.
- Mainly used when systems allow deadlocks.

13. Deadlock Recovery – Key Concepts

Once deadlock is detected, the system must recover.

Recovery techniques:

- **Process Termination**
  - Abort all deadlocked processes
  - Abort one process at a time
- **Resource Preemption**

    - Take resources from some processes and allocate them to others.
- **Rollback**
    - Restore processes to a safe state.

Introduction : Process Coordination

Process coordination ensures that multiple processes can execute concurrently without conflicts, ensuring data consistency and system stability. Synchronization is required when processes share resources or execute critical operations simultaneously.

Synchronization Background

Definition :

Synchronization background refers to the need for coordinating multiple processes that **execute concurrently and share common resources** such as memory, files, or variables. When several processes access shared data at the same time, the **order of execution may affect the final result**, leading to problems like **race conditions**.

Process synchronization ensures that **only one process accesses the critical section at a time**, maintaining **data consistency and system reliability**.

Key Concepts

- **Concurrent execution** of processes
- **Shared resources** (memory, files, variables)
- Need for **mutual exclusion**
- Avoiding **race conditions**

Advantages

- Maintains **data consistency** when multiple processes access shared resources.
- Prevents **race conditions**.
- Ensures **correct and reliable execution** of programs.
- Enables **safe resource sharing** in multi-user systems.

Disadvantages

- Increases **system complexity**.
- Improper synchronization can lead to **deadlocks or starvation**.
- May cause **performance overhead** due to waiting and coordination.

Applications

- **Database management systems** for maintaining consistent data.
- **Banking systems** for safe account updates.
- **Operating systems** for process and thread coordination.
- **Multi-user systems** where multiple users access shared resources.

Example: Suppose two processes P1 and P2 **update the same bank account balance.**

- Initial Balance = ₹1000
- P1 withdraws ₹200
- P2 withdraws ₹300

Without synchronization, both processes may read the balance at the same time**, causing incorrect results.**

Correct result should be: 1000 − 200 − 300 = 500

Synchronization ensures that **one process completes its operation before the other begins**, producing the correct balance.

Lack of Synchronization in Inter Process Communication Environment leads to following problems:

1. **Inconsistency:** When two or more processes access shared data at the same time without proper synchronization. This can lead to conflicting changes, where one process's update is overwritten by another, causing the data to become unreliable and incorrect.
2. **Loss of Data:** Loss of data occurs when multiple processes try to write or modify the same shared resource without coordination. If one process overwrites the data before another process finishes, important information can be lost, leading to incomplete or corrupted data.
3. **Deadlock:** Lack of Synchronization leads to Deadlock which means that two or more processes get stuck, each waiting for the other to release a resource. Because none of the processes can continue, the system becomes unresponsive and none of the processes can complete their tasks.

Conditions that Require Process Synchronization

1. **Race Condition**
2. **The Critical Section Race**

Condition:

A race condition is a situation that may occur inside a critical section. This happens when the result of multiple process/thread execution in the critical section differs according to the order in which the threads execute. Race conditions in critical sections can be avoided if the critical section is treated as an atomic instruction. Also, proper thread synchronization using locks or atomic variables can prevent race conditions.

Let us consider the following example.

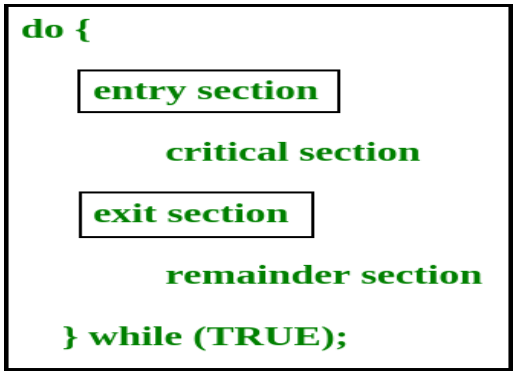
- There is a shared variable balance with value 100.
- There are two processes deposit (10) and withdraw (10). The deposit process does balance = balance + 10 and withdraw process does balance = balance – 10.
- Suppose these processes run in an interleaved manner. The deposit () fetches the balance as 100, then gets preempted.
- Now withdraw () get scheduled and makes balance 90.
- Finally, deposit is rescheduled and makes the value as 110. This value is not correct as the balance after both operations should be 100 only

We cannot notice that the different segments of two processes running in different order would give different values of balance.

The Critical Section Problem

A critical section is a code segment that can be accessed by only one process at a time. The critical section contains shared variables that need to be synchronized to maintain the consistency of data variables. So the critical section problem means designing a way for cooperative processes to access shared resources without creating data inconsistencies.

In the above example, the operations that involve balance variable should be put in critical sections of both deposit and withdraw.



In the entry section, the process requests for entry in the **Critical Section.**

Any solution to the critical section problem must satisfy three requirements:

- **Mutual Exclusion**: If a process is executing in its critical section, then no other process is allowed to execute in the critical section.
- **Progress**: If no process is executing in the critical section and other processes are waiting outside the critical section, then only those processes that are not executing in their remainder section can participate in deciding which will enter the critical section next, and the selection cannot be postponed indefinitely.
- **Bounded Waiting**: A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

The use of critical sections in a program can cause a number of issues, including:

- **Deadlock:** When two or more threads or processes wait for each other to release a critical section, it can result in a deadlock situation in which none of the threads or

processes can move. Deadlocks can be difficult to detect and resolve, and they can have a significant impact on a program's performance and reliability.

- **Starvation:** When a thread or process is repeatedly prevented from entering a critical section, it can result in starvation, in which the thread or process is unable to progress. This can happen if the critical section is held for an unusually long period of time, or if a high-priority thread or process is always given priority when entering the critical section.

- **Overhead:** When using critical sections, threads or processes must acquire and release locks or semaphores, which can take time and resources. This may reduce the program's overall performance.

Advantages of Critical Section in Process Synchronization

- Prevents race conditions
- Provides mutual exclusion
- Reduces CPU utilization
- Simplifies synchronization

Disadvantages of Critical Section in Process Synchronization

- Overhead
- Deadlocks
- Can limit parallelism
- Can cause contention

Peterson's Solution

Peterson's Algorithm is a classic solution to the critical section problem in process synchronization. It ensures mutual exclusion meaning only one process can access the critical section at a time and avoids race conditions. The algorithm uses two shared variables to manage the turn-taking mechanism between two processes ensuring that both processes follow a fair order of execution.

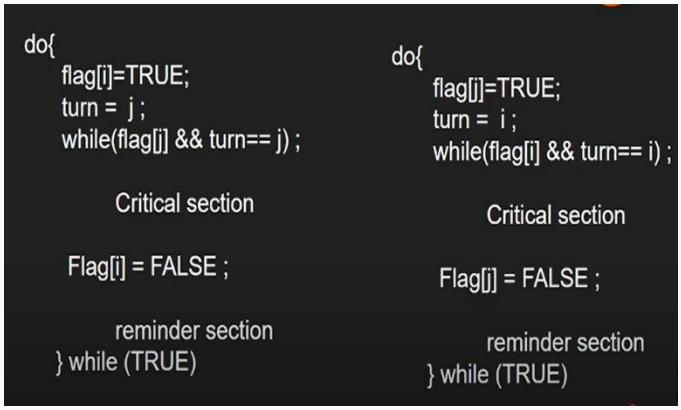
For example, let int a=6, and there are two processes p1 and p2 that can modify the value of a. p1 adds 2 to a a=a+2 and p2 multiplies a with 2, a=a*2. If both processes modify the value of a at the same time, then a value depends on the order of execution of the process. If p1 executes first, a will be 8; if p2 executes first, a will be 12. This change of values due to access by two processes at a time is the cause of the critical section problem.

The section in which the values are being modified is called the **critical section**. There are three sections except for the critical sections: the **entry section**, **exit section**, and the **reminder section**.

- The process entering the critical region must pass the entry region in which they request entry to the critical section.
- The process exiting the critical section must pass the exit region.
- The remaining code left after execution is in the remainder section.

Peterson's Algorithm

- The algorithm used for implementing Peterson's algorithm can be written in pseudocode as follows.



Peterson's solution provides a solution to the following problems,

- It ensures that if a process is in the critical section, no other process must be allowed to enter it. This property is termed **mutual exclusion**.
- If more than one process wants to enter the critical section, the process that should enter the critical region first must be established. This is termed **progress**.
- There is a limit to the number of requests that processors can make to enter the critical region, provided that a process has already requested to enter and is waiting. This is termed **bounding**.

Advantages of Peterson's Solution

- Peterson's solution allows multiple processes to share and access a resource without conflict between the resources.
- Every process gets a chance of execution.
- It is simple to implement and uses simple and powerful logic.
- It can be used in any hardware as it is completely software dependent and executed in the user mode.
- Prevents the possibility of a deadlock.

Disadvantages of Peterson's Solution

- The process may take a long time to wait for the other processes to come out of the critical region. It is termed as **Busy waiting**.
- This algorithm may not work on systems having multiple CPUs.
- The Peterson solution is restricted to only two processes at a time.

Synchronization Hardware

Synchronization hardware in operating systems provides atomic, uninterruptible instructions (like Test-and-Set or Swap) to manage concurrent processes and prevent race conditions when accessing shared resources. These hardware mechanisms ensure mutual exclusion by allowing only one process to modify a lock variable at a time.

Key Synchronization Hardware Techniques:

Test-and-Set Instruction: This atomic instruction tests a memory value and sets it to a new value simultaneously. If a process calls TestAndSet on a lock, it receives the old value and updates the lock, allowing it to enter the critical section only if the lock was previously false.

Example: A boolean variable lock is initialized to false.

A process executes while (TestAndSet(&lock));. If another process is in the critical section, lock is true, and the first process loops ("busy waiting"). Once the lock is released (false), the process breaks the loop and sets the lock to true.

Swap Instruction: This atomic instruction swaps the contents of two memory locations (a lock and a key). A process swaps its local key with the global lock variable; if the key becomes false, it enters the critical section.

Interrupt Disabling: In uniprocessor environments, a process can disable interrupts before entering its critical section and enable them upon exiting, ensuring it cannot be preempted by another process.

Advantages:

- Provides a straightforward way to solve the critical section problem.
- Ideal for multiprocessor environments.
- Highly efficient for short waiting times.

Disadvantages:

- Can lead to "busy waiting" (spinlocks), consuming CPU cycles while waiting.
- Potential for starvation if a process waits indefinitely.

Semaphores

In operating systems, semaphores are used to ensure proper process synchronization and to avoid race conditions when multiple processes access shared resources concurrently.

Semaphores For Process Synchronization

Semaphore is a variable (commonly an integer type) that is used to control access to a common resource by multiple processes in a concurrent system. By controlling access to shared resources, semaphores prevent critical section issues and ensure **process synchronization** in multiprocessing systems.

There are two atomic operations defined for semaphores: **wait(S)**, which decrements semaphore S, if S is positive, and **signal(S)**, which increments S, allowing process synchronization.

Wait Operation

The wait operation decrements its argument, S, if it is positive. If S is negative or zero, no operation is performed. This operation checks the semaphore's value. If the value is greater than 0, the process continues and S is decremented by 1. If the value is 0, the process is blocked(waits) until S becomes positive.

```
wait(S)
{
  while (S <= 0);
  S--;
}
```

Signal Operation

The signal operation increments its argument, S. After a process finishes using the shared resource, it performs the signal operation, which increases the semaphore's value by 1, potentially unblocking other waiting processes and allowing them to access the resource.

```
signal(S)
{
  S++;
}
```

Types of Semaphores

There are two main types of semaphores: counting semaphores and binary semaphores. Details about these are as follows −

- Binary Semaphores
- Counting Semaphores

1. Binary Semaphores

Binary semaphores are the semaphores whose value is **restricted to 0 and 1**. The wait operation only works when the semaphore is 0. Implementing binary semaphores is sometimes easier than counting semaphores.

Binary semaphores are implemented in the system where **single instances of resource** are available. For example, if there is only one printer in the system, a binary semaphore can be used to control access to the printer.

2. Counting Semaphores

Counting semaphores are integer-value semaphores have an **unrestricted value** domain. They are used to coordinate resource access, with the semaphore count representing the number of available resources. If resources are added, the semaphore count is incremented automatically; if resources are removed, the count is decremented.

The counting semaphore is used when **multiple instances of a resource** are available. For example, if there are 5 identical resources, the semaphore S is initialized to 5. Each time a process acquires a resource, the S is decremented by 1 and when it is released the S is incremented by 1. When S=0, no resources are available, and processes requesting resources are blocked until a resource is released.

Working of Semaphores

To understand working of semaphores, take a situation between two processes, P1 and P2, that need to access a shared resource. Initially, the semaphore S is set to 1, this indicates that the resource is available.

State 1 − Both P1 and P2 are in their non-critical sections.

State 2 − P1 wants to enter its critical section, so it performs wait(S). So S is decremented to 0.

State 3 − P1 is in critical section, now P2 also wants to enter its critical section, so it performs wait(S). Since S is 0, P2 is blocked and waits.

State 4 − P1 finishes its critical section and performs signal(S). So S is incremented to 1.

State 5 − P2 is unblocked and performs wait(S). So S is decremented to 0, and P2 enters its critical section.

State 6 − P2 finishes its critical section and performs signal(S). So S is incremented to 1.

The following diagram illustrates the above steps −

```
Initial Value: S = 1
P1: wait(S) -> S = 0 (P1 enters critical section)
P2: wait(S) -> S = 0 (P2 is blocked)
P1: signal(S) -> S = 1 (P1 exits critical section)
P2: wait(S) -> S = 0 (P2 enters critical section)
P2: signal(S) -> S = 1 (P2 exits critical section)
```

Classic problems of synchronization

Classic problems of synchronization in operating systems (OS) arise when concurrent processes share resources, requiring coordination to avoid data inconsistencies and deadlocks. Key examples include the **Producer-Consumer Problem, Readers-Writers Problem, and Dining Philosophers**

Problem. These are generally solved using synchronization tools like semaphores, mutexes, and monitors.

1. Bounded-Buffer (Producer-Consumer) Problem

This problem involves two processes: a **Producer** and a **Consumer**, who share a fixed-size buffer.

- **Problem:** The producer must not add data when the buffer is full, and the consumer must not remove data when the buffer is empty.
- **Example:** A print spooler where one process sends documents (produces) to a queue, and the printer (consumes) removes them to print. If the queue is full, the producer must wait; if empty, the consumer must wait.

2. Readers-Writers Problem

This problem occurs when a data object (like a database or file) is shared among multiple processes.

- **Problem:** Multiple readers can access the resource concurrently, but writers must have exclusive access. If a writer is writing, no other process can read or write.
- **Example:** A shared passenger flight schedule. Many passengers (readers) can check flight times simultaneously, but only one agent (writer) should update the schedule at a time.

3. Dining Philosophers Problem

This is a classic resource allocation problem illustrating deadlock and starvation.

- **Problem:** Five philosophers sit at a round table with five chopsticks (or forks) between them. They alternate between thinking and eating. To eat, a philosopher needs *two* chopsticks—the ones on their immediate left and right.
- **Example:** If all five philosophers pick up their left chopstick simultaneously, they will all wait indefinitely for the right one, creating a **deadlock**.

4. Sleeping Barber Problem

This problem illustrates the communication between the producer and consumer, where the consumer (barber) might sleep if there are no customers.

- **Problem:** A barber shop has one barber, one barber chair, and a waiting room with chairs. If no customers, the barber sleeps. If the barber is busy and all chairs are full, new customers leave.
- **Example:** A server handling requests. If there are no requests, it sleeps. If it is busy and the request queue is full, new requests are dropped.

Deadlocks: System Model

A deadlock occurs when a set of processes is stalled because each process is holding a resource and waiting for another process to acquire another resource. In the diagram below, for example, Process 1 is holding Resource 1 while Process 2 acquires Resource 2, and Process 2 is waiting for Resource 1.

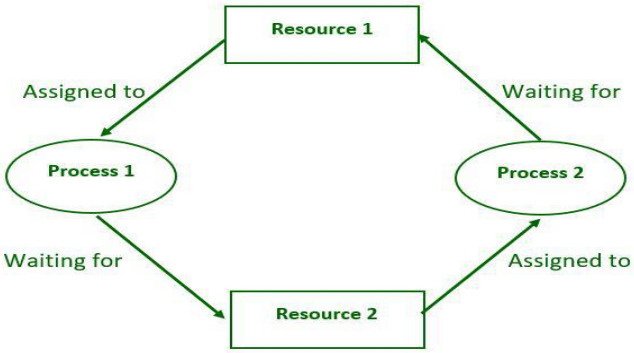


Figure: Deadlock in Operating system

System Model :

- For the purposes of deadlock discussion, a system can be modeled as a collection of limited resources that can be divided into different categories and allocated to a variety of processes, each with different requirements.
- Memory, printers, CPUs, open files, tape drives, CD-ROMs, and other resources are examples of resource categories.
- By definition, all resources within a category are equivalent, and any of the resources within that category can equally satisfy a request from that category. If this is not the case (i.e. if there is some difference between the resources within a category), then that category must be subdivided further. For example, the term "printers" may need to be subdivided into "laser printers" and "color inkjet printers."
- Some categories may only have one resource.
- The kernel keeps track of which resources are free and which are allocated, to which process they are allocated, and a queue of processes waiting for this resource to become available for all kernel-managed resources. Mutexes or wait() and signal() calls can be used to control application-managed resources (i.e. binary or counting semaphores.)
- When every process in a set is waiting for a resource that is currently assigned to another process in the set, the set is said to be deadlocked.

Deadlock Characterization

Deadlock in operating systems occurs when a group of processes are permanently blocked because each holds a resource while waiting for another resource held by another process in the group. Characterization requires four necessary conditions (Coffman conditions): mutual exclusion, hold and wait, no preemption, and circular wait.

Key Concepts in Deadlock Characterization

- **Resource Allocation Graph (RAG):** A directed graph used to detect deadlocks. If a RAG contains a cycle, a deadlock *might* exist; for single-instance resources, a cycle implies a deadlock.
- **System Model:** A process must request, use, and release resources. If a requested resource is unavailable, the process enters a waiting state that it may never leave.

- **Livelock:** A variation where processes constantly change state in response to each other but make no progress.

Necessary Conditions:

There are four conditions that must be met in order to achieve deadlock as follows.

1. **Mutual Exclusion -**
   At least one resource must be kept in a non-shareable state; if another process requests it, it must wait for it to be released.
2. **Hold and Wait -**
   A process must hold at least one resource while also waiting for at least one resource that another process is currently holding.
3. **No preemption -**
   Once a process holds a resource (i.e. after its request is granted), that resource cannot be taken away from that process until the process voluntarily releases it.
4. **Circular Wait -**
   There must be a set of processes P0, P1, P2,..., PN such that every P[I] is waiting for P[(I + 1) percent (N + 1)]. (It is important to note that this condition implies the hold-and-wait condition, but dealing with the four conditions is easier if they are considered separately).

Methods for Handling Deadlocks :

In general, there are three approaches to dealing with deadlocks as follows.

1. Preventing or avoiding deadlock by Avoid allowing the system to become stuck in a loop.
2. Detection and recovery of deadlocks, When deadlocks are detected, abort the process or preempt some resources.
3. Ignore the problem entirely.
4. To avoid deadlocks, the system requires more information about all processes. The system, in particular, must understand what resources a process will or may request in the future. (Depending on the algorithm, this can range from a simple worst-case maximum to a complete resource request and release plan for each process.)
5. Deadlock detection is relatively simple, but **deadlock recovery necessitates either aborting processes or preempting resources,** neither of which is an appealing option.
6. If deadlocks are not avoided or detected, the system will gradually slow down as more processes become **stuck waiting for resources that the deadlock has blocked** and other **waiting processes**. Unfortunately, when the computing requirements of a real-time process are high, this slowdown can be confused with a general system slowdown.

Deadlock Prevention

Deadlocks can be avoided by avoiding at least one of the four necessary conditions: as follows.

Condition-1 : Mutual Exclusion :

- Read-only files, for example, do not cause deadlocks.
- Unfortunately, some resources, such as printers and tape drives, require a single process to have exclusive access to them.

Condition-2 : Hold and Wait :

To avoid this condition, processes must be prevented from holding one or more resources while also waiting for one or more others. There are a few possibilities here:

- Make it a requirement that all processes request all resources at the same time. This can be a waste of system resources if a process requires one resource early in its execution but does not require another until much later.
- Processes that hold resources must release them prior to requesting new ones, and then re-acquire the released resources alongside the new ones in a single new request. This can be a problem if a process uses a resource to partially complete an operation and then fails to re-allocate it after it is released.
- If a process necessitates the use of one or more popular resources, either of the methods described above can result in starvation.

Condition-3 : No Preemption :

When possible, preemption of process resource allocations can help to avoid deadlocks.

- One approach is that if a process is forced to wait when requesting a new resource, all other resources previously held by this process are implicitly released (preempted), forcing this process to re-acquire the old resources alongside the new resources in a single request, as discussed previously.
- Another approach is that when a resource is requested, and it is not available, the system looks to see what other processes are currently using those resources and are themselves blocked while waiting for another resource. If such a process is discovered, some of their resources may be preempted and added to the list of resources that the process is looking for.
- Either of these approaches may be appropriate for resources whose states can be easily saved and restored, such as registers and memory, but they are generally inapplicable to other devices, such as printers and tape drives.

Condition-4 : Circular Wait :

- To avoid circular waits, number all resources and insist that processes request resources is strictly increasing (or decreasing) order.
- To put it another way, before requesting resource Rj, a process must first release all Ri such that I >= j.

- The relative ordering of the various resources is a significant challenge in this scheme.

Deadlock Avoidance

- The general idea behind deadlock avoidance is to avoid deadlocks by avoiding at least one of the aforementioned conditions.
- This necessitates more information about each process AND results in low device utilization. (This is a conservative approach.)
- The scheduler only needs to know the maximum number of each resource that a process could potentially use in some algorithms. In more complex algorithms, the scheduler can also use the schedule to determine which resources are required and in what order.
- When a scheduler determines that starting a process or granting resource requests will result in future deadlocks, the process is simply not started or the request is denied.
- The number of available and allocated resources, as well as the maximum requirements of all processes in the system, defines a resource allocation state.

Deadlock Detection and Recovery from Deadlock:

If deadlocks cannot be avoided, another approach is to detect them and recover in some way.

- Aside from the performance hit of constantly checking for deadlocks, a policy/algorithm for recovering from deadlocks must be in place, and when processes must be aborted or have their resources preempted, there is the possibility of lost work.

There are three basic approaches to getting out of a bind:

1. Inform the system operator and give him/her permission to intervene manually.
2. Stop one or more of the processes involved in the deadlock.
3. Prevent the use of resources.

Approach of Recovery From Deadlock :

Here, we will discuss the approach of Recovery From Deadlock as follows.

Approach-1 :Process Termination :

There are two basic approaches for recovering resources allocated to terminated processes as follows.

1. Stop all processes that are involved in the deadlock. This does break the deadlock, but at the expense of terminating more processes than are absolutely necessary.
2. Processes should be terminated one at a time until the deadlock is broken. This method is more conservative, but it necessitates performing deadlock detection after each step.

In the latter case, many factors can influence which processes are terminated next as follows.

1. Priorities in the process
2. How long has the process been running and how close it is to completion.
3. How many and what kind of resources does the process have? (Are they simple to anticipate and restore?)
4. How many more resources are required for the process to be completed?
5. How many processes will have to be killed?
6. Whether the process is batch or interactive.

Approach-2 : Resource Preemption :

When allocating resources to break the deadlock, three critical issues must be addressed:

1. **Selecting a victim -**
   Many of the decision criteria outlined above apply to determine which resources to preempt from which processes.
2. **Rollback -**
   A preempted process should ideally be rolled back to a safe state before the point at which that resource was originally assigned to the process. Unfortunately, determining such a safe state can be difficult or impossible, so the only safe rollback is to start from the beginning. (In other words, halt and restart the process.)
3. **Starvation -**
   To prevent process starvation due to repeated resource preemption; implement a priority mechanism that increases a process's priority upon each preemption, ensuring it eventually gains sufficient priority to avoid further interruption.

Question Number	Questions
Unit - II: PROCESS CONCEPTS AND CPU SCHEDULING	
PART – A (Two Marks Questions)	
1	What is process synchronization?
2	Define critical section.
3	What is a semaphore?
4	Explain the need for synchronization in OS.
5	What is the critical section problem?
6	Describe Peterson's solution briefly.
7	How are semaphores used to control access to shared resources?
8	Give an example of a synchronization problem in real life.
9	Differentiate between binary semaphore and counting semaphore.
10	Compare deadlock prevention and deadlock avoidance.
11	Why is mutual exclusion important in synchronization?
12	Evaluate the effectiveness of deadlock detection methods.
13	Construct a simple scenario illustrating the dining philosophers problem.
14	Design steps to recover from a deadlock situation.
15	Describe four necessary conditions for deadlock.

PART – B(Ten Marks Questions)	
1	(a) Define process synchronization. Why is synchronization necessary in operating systems? (b) Explain the race condition problem with a suitable example.
2	(a) What is the Critical Section Problem? (b) Demonstrate with an example how mutual exclusion can prevent race conditions.
3	(a) Explain Peterson's solution for two processes. (b) Analyze how Peterson's solution ensures mutual exclusion, progress, and bounded waiting.
4	List the hardware instructions used for synchronization with example.
5	(a) Define semaphores and explain the difference between binary and counting semaphores. (b) Explain the use of semaphores in solving the Producer–Consumer problem.
6	(a) List any three classic synchronization problems. (b) Explain the Readers–Writers problem and propose a semaphore- based solution.
7	(a) Define deadlock and explain the system model for deadlocks. (b) Analyze the four necessary conditions for deadlock with examples.
8	Compare Deadlock Prevention and Deadlock Avoidance methods with examples.
9	Explain the deadlock detection algorithm for single-instance resources with example.
10	(a) Explain the methods of recovering from deadlock. (b) Evaluate process termination and resource preemption as deadlock recovery techniques.

UNIT IV

MASS STORAGE STRUCTURE & MEMORY MANAGEMENT

"Efficient storage management is the backbone of modern operating systems."
– Andrew S. Tanenbaum

OVERVIEW

Mass Storage Structure and **Memory Management** are essential components of an operating system that deal with efficient handling of storage and memory resources. Mass storage focuses on secondary storage devices such as **disks,** explaining how data is **organized, accessed, and managed using disk structure, attachment methods, scheduling algorithms, and disk management** techniques. Memory management, on the other hand, deals with the **allocation and utilization of main memory,** ensuring that multiple processes can run efficiently. It includes concepts like **logical and physical address space, swapping, paging, segmentation, and page replacement** algorithms. Together, these topics help in improving **system performance, optimizing resource utilization, and enabling smooth multitasking** in modern computing systems.

OBJECTIVES

- To understand the structure and working of mass storage devices
- To learn disk scheduling techniques for performance improvement.
- To study different memory management methods
- To understand logical and physical address mapping
- To analyze paging, segmentation, and allocation techniques
- To learn page replacement algorithms and their efficiency.

LEARNING OUTCOMES

- Ability to explain disk structure and operations
- Ability to compare disk scheduling algorithms
- Understanding of logical vs physical address space
- Ability to apply paging and segmentation concepts
- Skill to analyze page replacement algorithms
- Capability to solve memory management problems

IMPORTANCE OF STUDYING THIS UNIT

This unit is very important as it provides a clear understanding of how an operating system manages memory and storage efficiently, which are critical for overall system performance. It helps in optimizing the **use of memory, reducing fragmentation**, and **improving the speed of data access from disks**. Knowledge of these concepts is essential for designing efficient systems, supporting **multitasking**, and ensuring better **resource utilization**. It is also highly useful in real-world applications such as database systems, cloud computing, and operating system development, making it a key area for academic learning as well as technical interviews.

Key Concepts

Mass Storage Structure

- **Disk Structure**

  A disk is divided into tracks, sectors, and cylinders where data is physically stored. Tracks are circular paths, and sectors are smaller divisions used to store data.

- **Disk Attachment**

  Refers to how disks are connected to the system. It can be direct attachment (like SATA) or network-based storage such as NAS.

- **Disk Scheduling**

  It determines the order in which disk I/O requests are processed to reduce seek time and improve performance. Common algorithms include FCFS, SSTF, SCAN, and C-SCAN.

- **Disk Management**

  Involves operations like disk formatting, boot block initialization, and handling bad sectors to ensure reliable storage.

Memory Management

- **Logical Address Space**

  The set of addresses generated by the CPU for a program during execution.

- **Physical Address Space**

  The actual addresses in main memory (RAM) where data is stored.

- **Swapping**

  A technique in which processes are temporarily moved from main memory to disk and brought back when needed to support multitasking.

- **Contiguous Memory Allocation**

  Memory is allocated in a continuous block to a process. It is simple but may lead to fragmentation.

- **Paging**

  Memory is divided into fixed-size pages and frames. It avoids external fragmentation and allows efficient memory use.

- **Page Table**

  A data structure that maps logical pages to physical frames in memory.

- **Segmentation**

  Memory is divided into logical segments like code, data, and stack, making program structure more meaningful.

- **Page Replacement Algorithms**

  Techniques used to replace pages when memory is full. Examples include FIFO, LRU and Optimal.

Mass Storage Structure:

Overview of Mass Storage Structure

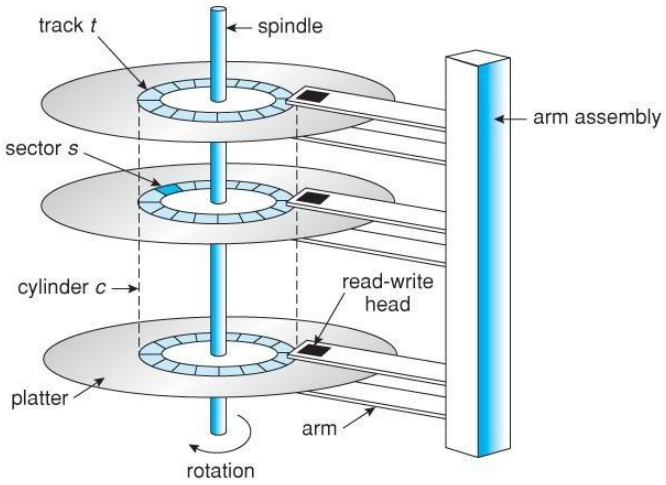
Mass storage is an essential component of modern computing systems, providing non-volatile storage for operating systems, applications, and user data. The primary concerns in mass storage management include:

- **Performance:** Efficient data access and transfer rates.
- **Reliability:** Protection against data corruption and failures.
- **Capacity:** Managing large volumes of data.
- **Cost-effectiveness:** Balancing performance with budget constraints.

Magnetic Disks

- Traditional magnetic disks have the following basic structure:
    - One or more **platters** in the form of disks covered with magnetic media. **Hard disk** platters are made of rigid metal, while "**floppy**" disks are made of more flexible plastic.
    - Each platter has two working **surfaces.** Older hard disk drives would sometimes not use the very top or bottom surface of a stack of platters, as these surfaces were more susceptible to potential damage.
    - Each working surface is divided into several concentric rings called **tracks.** The collection of all tracks that are the same distance from the edge of the platter, is called a **cylinder**.
    - Each track is further divided into **sectors,** traditionally containing 512 bytes of data each, although some modern disks occasionally use larger sector sizes.
    - The data on a hard drive is read by read-write **heads.** The standard configuration uses one head per surface, each on a separate **arm**, and controlled by a common **arm assembly** which moves all heads simultaneously from one cylinder to another.

The storage capacity of a traditional disk drive is equal to the number of heads, times the number of tracks per surface, times the number of sectors per track, times the number of bytes per sector. A particular physical block of data is specified by providing the head-sector-cylinder number at which it is located.



Moving-head disk mechanism

- In operation the disk rotates at high speed, such as 7200 rpm (120 revolutions per second.) The rate at which data can be transferred from the disk to the computer is composed of several steps:
    - The **positioning time**, a.k.a. the **seek time** or **random access time** is the time required to move the heads from one cylinder to another, and for the heads to settle down after the move. This is typically the slowest step in the process and the predominant bottleneck to overall transfer rates.
    - The **rotational latency** is the amount of time required for the desired sector to rotate around and come under the read-write head.This can range anywhere from zero to one full revolution, and on the average will equal one-half revolution. This is another physical step and is usually the second slowest step behind seek time.
        - The **transfer rate**, which is the time required to move the data electronically from the disk to the computer.
- Disk heads "fly" over the surface on a very thin cushion of air. If they should accidentally contact the disk, then a **head crash** occurs, which may or may not permanently damage the disk or even destroy it completely. For this reason it is normal to **park** the disk heads when turning a computer off, which means to move the heads off the disk or to an area of the disk where there is no data stored.
- Floppy disks are normally **removable**. Hard drives can also be removable, and some are even **hot-swappable**, meaning they can be removed while the computer is running, and a new hard drive inserted in their place.

- Disk drives are connected to the computer via a cable known as the **I/O Bus.** Some of the common interface formats include Enhanced Integrated Drive Electronics, EIDE; Advanced Technology Attachment, ATA; Serial ATA, SATA, Universal Serial Bus, USB; Fiber Channel, FC, and Small Computer Systems Interface, SCSI.

- The **host controller** is at the computer end of the I/O bus, and the **disk controller** is built into the disk itself. The CPU issues commands to the host controller via I/O ports. Data is transferred between the magnetic surface and onboard **cache** by the disk controller, and then the data is transferred from that cache to the host controller and the motherboard memory at electronic speeds.

Disk Structure

- The traditional head-sector-cylinder, HSC numbers are mapped to linear block addresses by numbering the first sector on the first head on the outermost track as sector 0. Numbering proceeds with the rest of the sectors on that same track, and then the rest of the tracks on the same cylinder before proceeding through the rest of the cylinders to the center of the disk. In modern practice these linear block addresses are used in place of the HSC numbers for a variety of reasons:

    1. The linear length of tracks near the outer edge of the disk is much longer than for those tracks located near the center, and therefore it is possible to squeeze many more sectors onto outer tracks than onto inner ones.
    2. All disks have some bad sectors, and therefore disks maintain a few spare sectors that can be used in place of the bad ones. The mapping of spare sectors to bad sectors in managed internally to the disk controller.
    3. Modern hard drives can have thousands of cylinders, and hundreds of sectors per track on their outermost tracks. These numbers exceed the range of HSC numbers for many operating systems, and therefore disks can be configured for any convenient combination of HSC values that falls within the total number of sectors physically on the drive.

- There is a limit to how closely packed individual bits can be placed on a physical media, but that limit is growing increasingly more packed as technological advances are made.

- Modern disks pack many more sectors into outer cylinders than inner ones, using one of two approaches:

    - With **Constant Linear Velocity, CLV,** the density of bits is uniform from

cylinder to cylinder. Because there are more sectors in outer cylinders, the disk spins slower when reading those cylinders, causing the rate of bits passing under the read-write head to remain constant. This is the approach used by modern CDs and DVDs.

- With **Constant Angular Velocity, CAV,** the disk rotates at a constant angular speed, with the bit density decreasing on outer cylinders.

Disk Attachment

Disk drives can be attached either directly to a particular host or to a network.

1. Host-Attached Storage

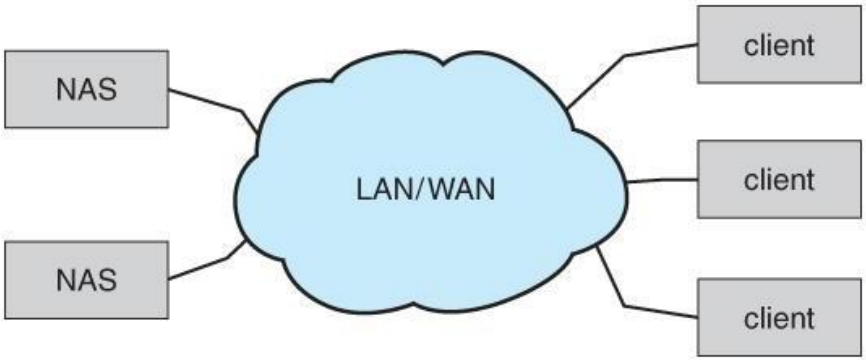
- Local disks are accessed through I/O Ports as described earlier.
- The most common interfaces are IDE or ATA, each of which allow up to two drives per host controller.
- SATA is similar with simpler cabling.
- High end workstations or other systems in need of larger number of disks typically use SCSI disks:
    - The SCSI standard supports up to 16 **targets** on each SCSI bus, one of which is generally the host adapter and the other 15 of which can be disk or tape drives.
    - A SCSI target is usually a single drive, but the standard also supports up to 8 **units** within each target. These would generally be used for accessing individual disks within a RAID array.
    - The SCSI standard also supports multiple host adapters in a single computer, i.e. multiple SCSI busses.
    - Modern advancements in SCSI include "fast" and "wide" versions, as well as SCSI-2.
    - SCSI cables may be either 50 or 68 conductors. SCSI devices may be external as well as internal.

    FC is a high-speed serial architecture that can operate over optical fiber or four-conductor copper wires, and has two variants:

    - A large switched fabric having a 24-bit address space. This variant allows for multiple devices and multiple hosts to interconnect, forming the basis for the **storage-area networks, SANs,** to be discussed in a future section.
    - The **arbitrated loop, FC-AL,** that can address up to 126

2. Network-Attached Storage

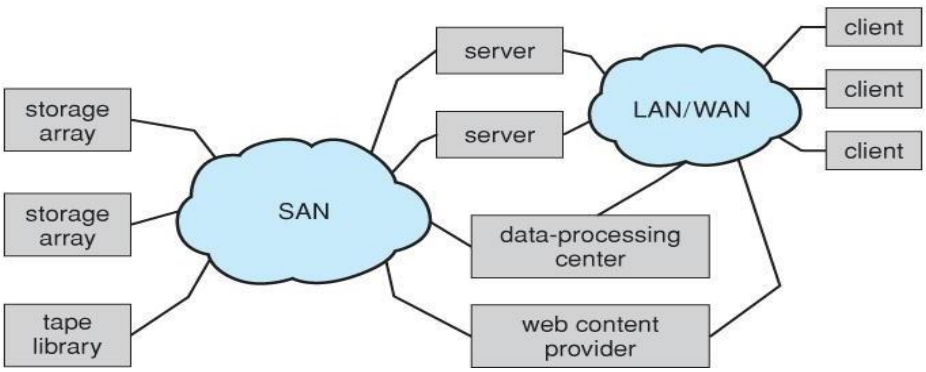
- Network attached storage connects storage devices to computers using a remote procedure call, RPC, interface, typically with something like NFS filesystem mounts. This is convenient for allowing several computers in a group common access and naming conventions for shared storage.
- NAS can be implemented using SCSI cabling, or **ISCSI** uses Internet protocols and standard network connections, allowing long-distance remote access to shared files.
- NAS allows computers to easily share data storage, but tends to be less efficient than standard host-attached storage.



Network-attached storage

3 . Storage-Area Network

- A **Storage-Area Network, SAN,** connects computers and storage devices in a network, using storage protocols instead of network protocols.
- One advantage of this is that storage access does not tie up regular networking bandwidth.
- SAN is very flexible and dynamic, allowing hosts and devices to attach and detach on the fly.
- SAN is also controllable, allowing restricted access to certain hosts and devices.



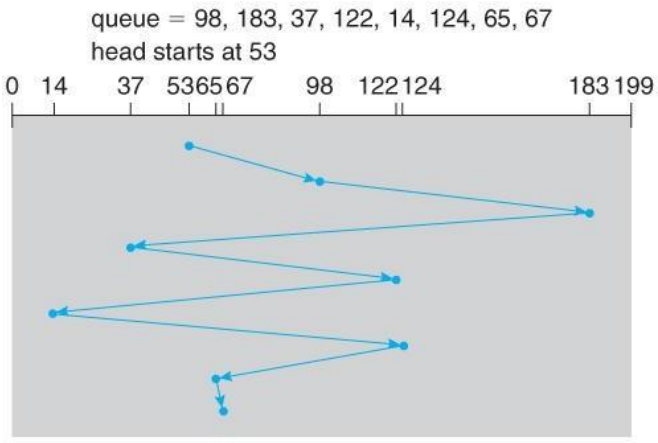
Storage-area network.

Disk Scheduling

- A disk transfer speeds are limited primarily by ***seek times*** and ***rotational latency.*** When multiple requests are to be processed there is also some inherent delay in waiting for other requests to be processed.
- ***Bandwidth*** is measured by the amount of data transferred divided by the total amount of time from the first request being made to the last transfer being completed.
- Both bandwidth and access time can be improved by processing requests in a good order.
- Disk requests include the disk address, memory address, number of sectors to transfer, and whether the request is for reading or writing.

1. FCFS Scheduling

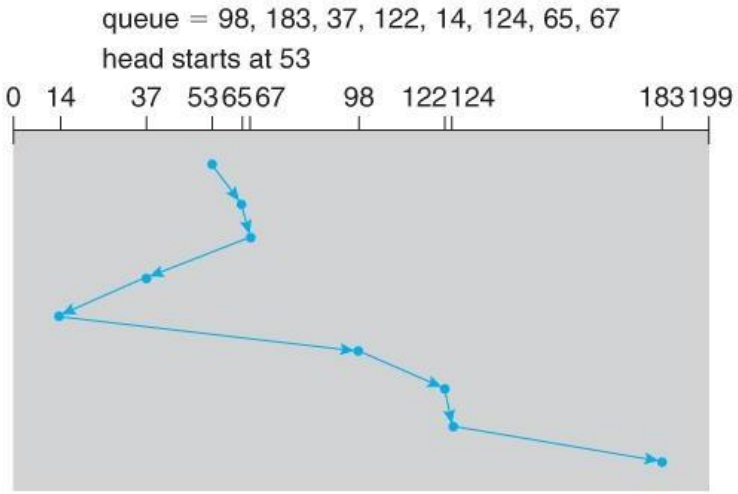
- ***First-Come First-Serve*** is simple and basically fair, but not very efficient. Consider in the following sequence the wild swing from cylinder 122 to 14 and then back to 124:



FCFS disk scheduling

1. SSTF Scheduling

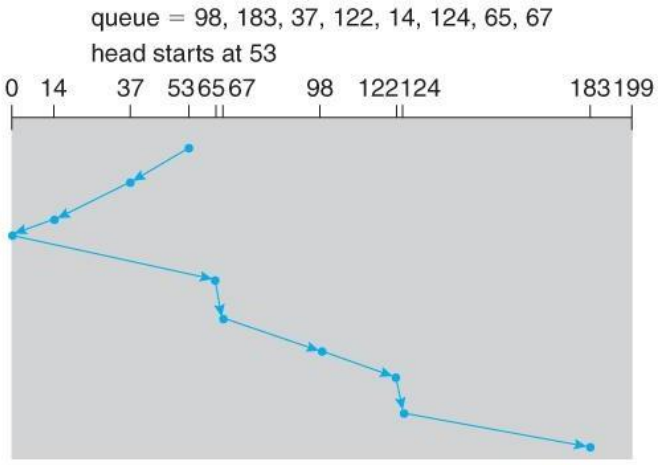
- ***Shortest Seek Time First*** scheduling is more efficient, but may lead to starvation if a constant stream of requests arrives for the same general area of the disk.
- SSTF reduces the total head movement to 236 cylinders, down from 640 required for the same set of requests under FCFS. Note, however that the distance could be reduced still further to 208 by starting with 37 and then 14 first before processing the rest of the requests.



SSTF disk scheduling

2. SCAN Scheduling

- The ***SCAN*** algorithm, a.k.a. the ***elevator*** algorithm moves back and forth from one end of the disk to the other, similarly to an elevator processing requests in a tall building.



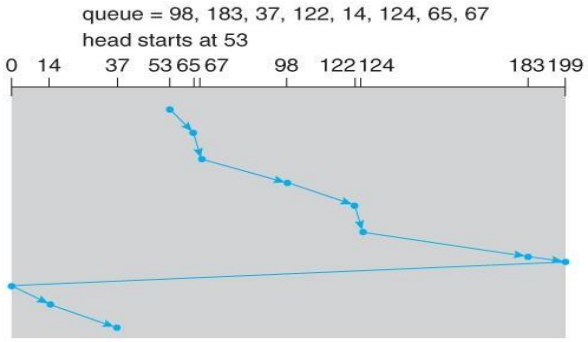
SCAN disk scheduling

Under the SCAN algorithm, If a request arrives just ahead of the moving head then it will be processed right away, but if it arrives just after the head has passed, then it will have to wait for the head to pass going the other way on the return trip. This leads to a fairly wide variation in access times which can be improved upon.

- Consider, for example, when the head reaches the high end of the disk: Requests with high cylinder numbers just missed the passing head, which means they are all fairly recent requests, whereas requests with low numbers may have been waiting for a much longer time. Making the return scan from high to low then ends up accessing recent requests first and making older requests wait that much longer.

3. C-SCAN Scheduling

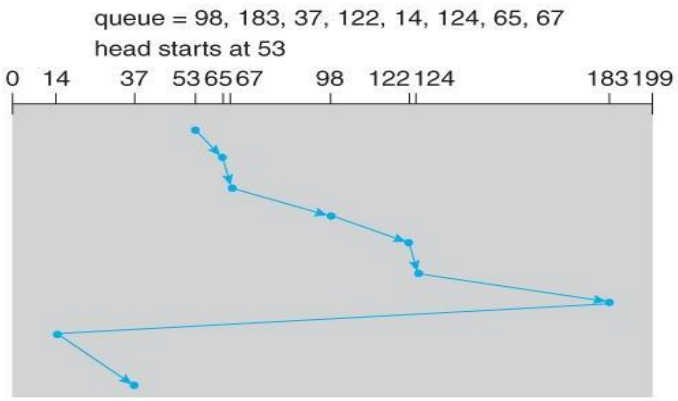
- The ***Circular-SCAN*** algorithm improves upon SCAN by treating all requests in a circular queue fashion - Once the head reaches the end of the disk, it returns to the other end without processing any requests, and then starts again from the beginning of the disk:



C-SCAN disk scheduling

4. LOOK Scheduling

- ***LOOK*** scheduling improves upon SCAN by looking ahead at the queue of pending requests, and not moving the heads any farther towards the end of the disk than is necessary. The following diagram illustrates the circular form of LOOK:



LOOK disk scheduling

5. Selection of a Disk-Scheduling Algorithm

- With very low loads all algorithms are equal, since there will normally only be one request to process at a time.
- For slightly larger loads, SSTF offers better performance than FCFS, but may lead to starvation when loads become heavy enough.
- For busier systems, SCAN and LOOK algorithms eliminate starvation problems.
- The actual optimal algorithm may be something even more complex than those discussed here, but the incremental improvements are generally not

worth the additional overhead.

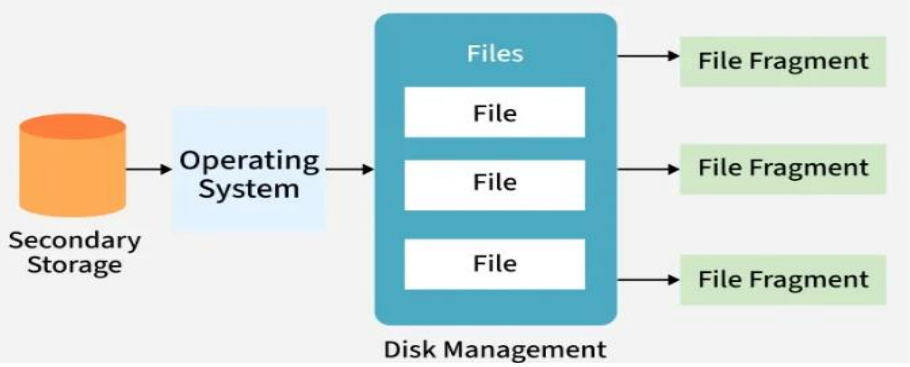
- Some improvement to overall filesystem access times can be made by intelligent placement of directory and/or inode information. If those structures are placed in the middle of the disk instead of at the beginning of the disk, then the maximum distance from those structures to data blocks is reduced to only one-half of the disk size. If those structures can be further distributed and furthermore have their data blocks stored as close as possible to the corresponding directory structures, then that reduces still further the overall time to find the disk block numbers and then access the corresponding data blocks.
- On modern disks the rotational latency can be almost as significant as the seek time, however it is not within the OSes control to account for that, because modern disks do not reveal their internal sector mapping schemes.
    - Some disk manufacturers provide for disk scheduling algorithms directly on their disk controllers, so that if a series of requests are sent from the computer to the controller, then those requests can be processed in an optimal order.
    - Unfortunately, there are some considerations that the OS must consider that are beyond the abilities of the on-board disk-scheduling algorithms, such as priorities of some requests over others, or the need to process certain requests in a particular order. For this reason, OSes may elect to spoon-feed requests to the disk controller one at a time in certain situations.

Disk management

Disk management is a critical function of the operating system (OS) that deals with organizing, optimizing, and securing data on secondary storage devices such as hard disk drives (HDDs) and solid-state drives (SSDs).

It ensures:

- Efficient use of storage space
- Safe and reliable read/write operations
- Faster access times



Why Disk Management is Important

Most computer systems use secondary storage devices (like hard disks, SSDs, tapes, optical media, and flash drives) to store programs and data at low cost and with non-volatile storage.

Data is stored in the form of files.

The operating system (OS) manages file storage by allocating disk space as needed. Files are not always stored in one continuous block, large files may be fragmented into parts stored in different disk locations, especially when space is limited.

The OS keeps track of where each file (and its fragments) is located, often handling thousands of such entries. It ensures:

- Files can be quickly located for read/write operations
- Safe and reliable access to stored data
- Efficient management of access times

Key Operations in Disk Management

Disk Formatting

- **Low-level (physical) formatting:** Divides the disk into sectors with headers, data, and error correction codes (ECC).
- **Logical formatting**: Creates a file system, defining free space and allocated space.
- Blocks are grouped into clusters for efficient I/O.
- Some systems allow raw I/O (direct access to disk blocks without a file system).

2. Booting from Disk

- The bootstrap program loads the OS kernel into memory when the computer is powered on.
- A small bootstrap loader resides in ROM.
- The full bootstrap code is stored in the boot block of the disk.
- A disk with a boot partition is called a boot disk (system disk).

3. Bad Block Management

Disks often have **bad sectors** due to manufacturing defects or usage.

Handled using:

- **Sector sparing (replacement):** faulty sectors are replaced with spare ones.
- **Error recovery:** for soft errors.
- **Manual intervention:** required for hard errors.

Severe disk failures may require **replacing the disk** and restoring from backup.

Some common disk management techniques used in operating systems include:

- **Partitioning:** Divides a physical disk into multiple logical partitions, each acting as a separate storage device for better organization.
- **Formatting:** Prepares a disk by creating a file system; erases all existing data.
- **File System Management:** Manages file systems (e.g., FAT, NTFS, ext4) to store and access data efficiently.
- **Disk Space Allocation:** Allocates space for files using methods like contiguous, linked, or indexed allocation.
- **Disk Defragmentation:** Rearranges scattered data blocks to improve performance.

Memory Management

Logical & Physical Address Space

Memory access in operating systems happens through two types of addresses: logical (virtual) and physical. The Memory Management Unit (MMU) sits between the CPU and the physical memory. Its job is to translate every logical address into the correct physical address.

- Programs to believe they have a large, continuous memory space
- The OS protects one process's memory from another process
- Efficient use of RAM through paging, segmentation, or both

Whenever the CPU accesses memory, it sends a logical address → the MMU converts it → the corresponding physical address in RAM is accessed.

Logical Address

A logical address is generated by the CPU while a program runs. It represents the address from the process's perspective and does not exist physically, hence it is also called a virtual address.

- The logical address space is the set of all logical addresses a process can generate.
- Programs use logical addresses to reference memory, and the MMU translates them into physical addresses when accessing actual memory.

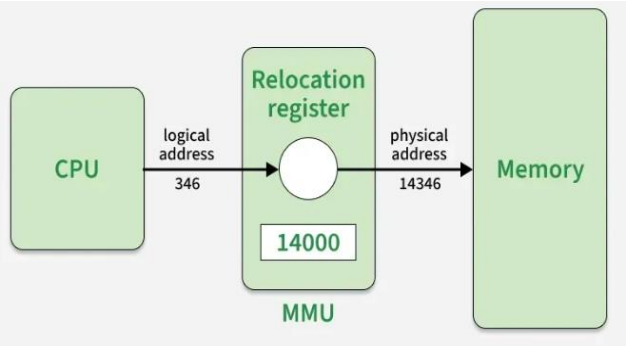
Physical Address

A physical address is the real location in main memory (RAM) where data or instructions are stored. The physical address space consists of all physical addresses corresponding to logical addresses.

- The MMU performs address translation using a page table, mapping each logical page to a physical frame.
- This allows processes to access memory transparently, without knowing actual memory locations.

Similarities in Logical and Physical Addresses

- Both logical and physical addresses are used to identify a specific location in memory.
- Each address type can be represented in different formats such as binary, hexadecimal, or decimal.
- They have a finite range, which is determined by the number of bits used to represent them.



Memory Management

Logical Address vs. Physical Address

Logical Address	Physical Address
Generated by the CPU during program execution	Generated by the Memory Management Unit (MMU)
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.
User can view and access the logical address of a program.	User can never view physical address of program.
Can change during program execution (due to relocation, paging, etc.)	Generally fixed once assigned in memory
The user can use the logical address to access the physical address. Logical address can be change.	The user can indirectly access physical address but not directly. Physical address will not change.
Virtual address.	Real address.

Swapping

Swapping is moving data between physical memory(RAM) and secondary memory. In computing, virtual memory is a management technique that combines a computer's hard disk space with its random access memory (RAM) to create a larger virtual address space. This can be useful if you have too many processes running on your system and not enough physical memory to store them.

While performing swapping, the **operating system** must allocate a memory block. It finds the first vacant block of physical memory.

As each new memory block is allocated, it replaces the oldest block in physical memory. When a program attempts to access a page that has been swapped out, the operating system copies the page from the disk into physical memory and updates the page table entry.

The purpose of operating system swapping is to increase the degree of multiprogramming and increase main memory usage.

There are two steps to changing the operating system:

- **Swap-in**: A swap-in process in which a process moves from secondary storage / hard disk to main memory (RAM).
- **Swap out:** Swap out takes a process from the main memory and places it in secondary memory.

Need of swapping in operating system

The main purpose of swapping in **memory management** is to allow more available memory than the computer hardware. Physical memory may be allocated, and the process may require additional memory. Rather than constraining the system to use only memory based on physical RAM, memory swapping allows the operating system and its users to extend the memory to disk.

Advantages of Swapping

- Swapping files can free up space and allow your programs to run more smoothly. It can also be helpful in cases where you have multiple programs open at the same time.
- Using a swap file, you can ensure that each program has its dedicated chunk of memory, which can help improve overall performance.
- Improve the degree of multi-programming.
- Better RAM utilization.

Disadvantages of Swapping

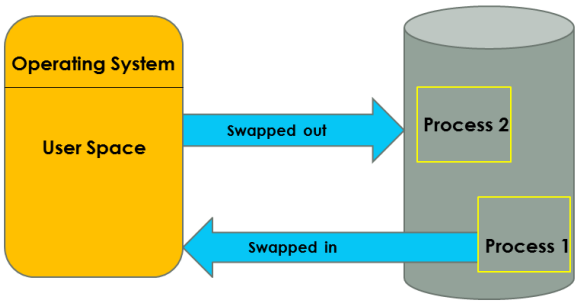
- If the computer system is turned off during high-paging activity, the user may lose all information related to the program.
- The number of page faults increases, which can reduce overall processing performance.
- When you make a lot of transactions, users lose information, and computers lose power.

Working of Swapping technique in Operating System

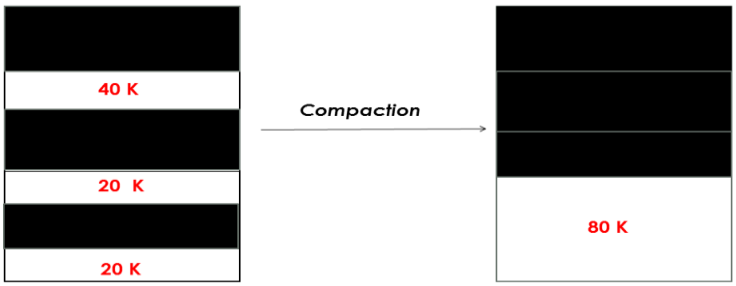
Swapping means **process exchange**. Process exchange also relies on priority-based preemptive scheduling. That means when a higher priority process arrives, the memory manager temporarily **swaps out** the lowest priority process to disk, and the highest priority process is **swapped in** the main memory for execution. When the highest priority process terminates, the

lower priority process is swapped back to memory and continues to run. This process is called **swapping.**

Due to the address binding method, processes paged out of main memory occupy the **same address space** when paged back to main memory if binding occurs during **load time**. If it had **run-time binding**, the address would have been calculated at run-time so that the process could use **any address space available** in the main memory.



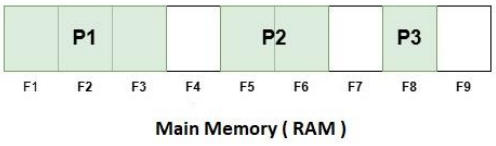
When the process is swapped in and out, it creates multiple holes in the memory. In that case, holes are combined to create a big memory space. This is done by moving all the processes **downwards** as far as possible. This technique is called **compaction**. However doing this requires a lot of CPU time. So, typically, this technique is not used. As in the below diagram, the free space available is 40+20+20 K. This available space is not available in a contiguous way. So this free space is compacted to make 80 K space.



Contiguous Memory Allocation

Contiguous memory allocation as the name suggests, allocates a single continuous block of memory to a process. The block of memory is allocated to a process is enough to hold the entire process. This means that the process can access all its memory locations without any interruptions.

The image below shows the main memory allocates storage to three processes P1, P2, and P3 using contiguous memory allocation −



In the image above, the process P1 is allocated 3 blocks of memory (F1, F2, and F3). P2 is allocated 2 blocks of memory (F5 and F6). Similarly, P3 is allocated 1 block of memory (F8). There some gaps in between the allocated memory blocks (at F4 and F7).

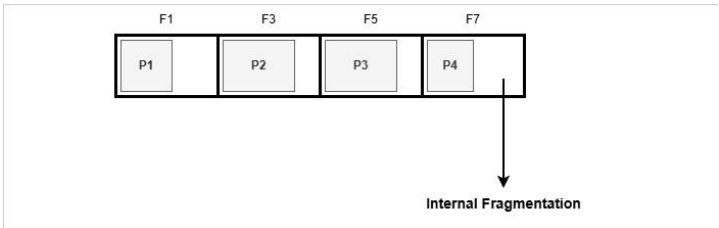
Two types of contiguous memory allocation techniques −

- Fixed Partitioning (Static)
- Dynamic Partitioning (Variable)

Fixed Partitioning in Contiguous Memory Allocation

The **fixed partitioning** or static partitioning is a memory allocation technique where the main memory is divided into partitions of fixed sizes. Each partition can hold exactly one process. When a new process loaded into memory, it is allocated a partition of fixed size. If the process is smaller than the partition size, the remaining memory in the partition will be wasted. If the process is larger than the partition size, it cannot be loaded into memory.

The image below shows internal fragmentation happening in main memory during fixed partitioning −

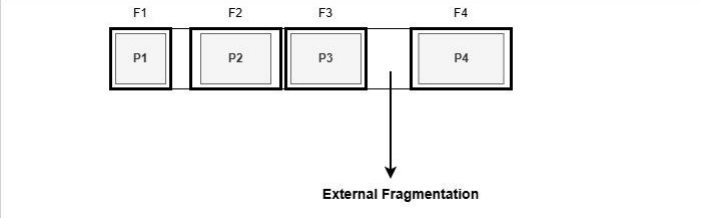


As the image above shows, one of main drawbacks of fixed partitioning is **internal fragmentation**. We can see that process are not using the entire allocated memory. This will lead to a situation where even though there is enough total memory available, it may not be possible to allocate memory to a new process, because the available memory may not be contiguous.

Dynamic Partitioning in Contiguous Memory Allocation

The **dynamic partitioning** or variable partitioning is a memory allocation technique where the main memory is divided into partitions of varying sizes depending on the size of the processes being loaded into memory. When a new process is loaded into memory, it is allocated to the first available partition that is large enough to hold the process. If no such partition is available, the process has to wait until a partition becomes free.

The image below shows external fragmentation happening in main memory during dynamic partitioning −



The major issue with dynamic partitioning is **external fragmentation**. In the image above, we can see that there are some gaps in the memory after allocating and deallocating memory to various processes. This situation happens when a process is terminated from memory, and it's neighboring processes are still running.

These gaps will add up over time as more processes are loaded and unloaded from memory. In such case, you need to restart the system to make the main memory free again.

Advantages of Contiguous Memory Allocation

- **Simplicity** − Contiguous memory allocation is simple to implement and manage. The operating system can easily keep track of the memory blocks allocated to each process.
- **Fast Access** − The entire process is stored in a single block of memory. No any calculation is needed to find the memory location of a process. So overall access time is faster.
- **Low Overhead** − Contiguous memory allocation has low overhead as there is no need for complex data structures to manage memory allocation.

Disadvantages of Contiguous Memory Allocation

Even though contiguous memory allocation is simple and fast, it comes with some drawbacks −

- **Wastage of Memory** − During contiguous memory allocation, there are gaps created in memory due loading and unloading of processes. This can lead to wastage of memory.
- **Fragmentation** − Contiguous memory allocation can lead to internal and external fragmentation. This reduce the overall efficiency of memory usage.
- **Limited Flexibility** − Contiguous memory allocation is not suitable for virtual memory systems and in the case where process are larger than the available memory itself.

Paging

Paging is a memory management technique used by modern operating systems to manage the allocation of memory to processes. In this technique, the physical memory (i.e., the RAM) is divided into blocks of fixed size called **frames**, and the logical memory (i.e., the memory where the process is stored) is divided into blocks of the same size called **pages**. When a process is executed, its pages are loaded into available frames in the physical memory.

The image below shows working of paging in operating system −

Key Concepts

- **Logical Address Space** − The logical address space is the set of all logical addresses generated by a process in hard disk. It is divided into pages.
- **Physical Address Space** − The physical address space refers to the set of all physical addresses in the main memory (RAM). It is divided into frames.
- **Page Table** − A page table is a data structure used by the operating system to direct logical addresses to physical addresses. It stores information such as the location of each page in the physical memory.
- **Page Number** − The page number is the index of a page in the logical address space.
- **Frame Number** − The frame number is the index of a frame in the physical address space.
- **Page Size** − The page size is the size of each page and frame. It is usually represented as a power of 2, For example, 4KB, 8KB, etc.

Address Translation in Paging

When a process generates a **logical address**, the operating system uses the **page table** to translate it into a **physical address**. This process is known as **address translation**. The logical address is divided into two parts −

- The page number
- The offset within the page

The **page number** is used to search the corresponding frame number in the page table, and the **offset** is the location within that frame. The physical address is then calculated by combining the frame number and the offset. Here is the formula to calculate the physical address:

Physical Address=(Frame Number × Page Size)+Offset

Structure of Page Table

The **page table** is a data structure inside the operating system that maps logical addresses and physical addresses. Each entry in the page table contains the following information:

- **Frame Number** − The frame number refer to the location of the page in physical memory.
- **Valid/Invalid Bit** − This bit is used to know if the page is in memory (valid) or not (invalid).
- **Protection Bits** − Bits that is used to indicate the permission of accessing the page (e.g., read, write, execute).
- **Dirty Bit** − A bit that is used to indicate if the page has been modified since it was loaded into memory.
- **Reference Bit** − A bit that is used to indicate whether the page has been accessed recently.

Example: Structure of a Page Table

Consider a system with following configuration −

- **System** A 32-bit system with a 4KB (2^{12} bytes) page size.
- **Logical Address Space** 2^{32} bytes (4GB).
- **Physical Address Space** 2^{30} bytes (1GB).
- **Number of Pages** $2^{32} / 2^{12} = 2^{20}$ pages.

In this case, the page table will have **2^{20} entries**. Each entry in the page table will contain the frame number, valid/invalid bit, protection bits, dirty bit, and reference bit. Consider that each entry takes 4 bytes, the total size of the page table will be −

Size of Page Table=Number of Pages×Size of Each Entry

=220×4 bytes=4 MB

Address Translation − Suppose a virtual address like **0 × 100000** is generated by a process. The page number can be calculated as:

Page Number=Virtual AddressPage Size=0×1000000×1000

=0×100

The offset within the page can be calculated as − Offset=Virtual AddressmodPage Size

=0×100000mod0×1000=0×000

So, the virtual address 0x100000 refers to,

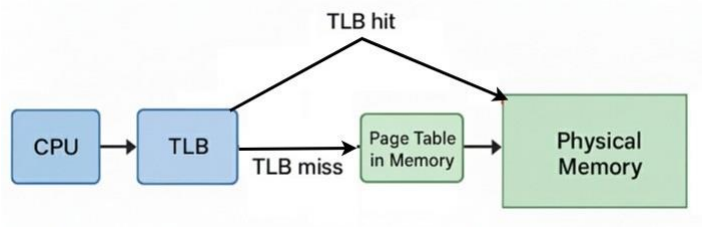
- Page Number: 0x100 (256 in decimal)
- Offset: 0x000 (0 in decimal)

The operating system will look up the page table entry for page number 0x100 to find the corresponding frame number. If the page is in memory (valid), it will get the frame number and calculate the physical address using the formula.

Translation Lookaside Buffer (TLB)

The **Translation Lookaside Buffer (TLB)** is a high speed cache inside CPU that stores recent values of translation made from virtual addresses to physical addresses. This will help to reduce the time taken for address translation, as it will store the most frequently used page table entries.

When a virtual address is generated, the TLB is checked first. If the entry is found in the TLB (a **TLB hit** occurs), the physical address can be obtained directly from the TLB. If the entry is not found in the TLB (a **TLB miss** occurs), the page table is used to get the frame number, and then the TLB is updated with this new entry.



Effective Access Time (EAT)

The **Effective Access Time (EAT)** is the average time taken to access a memory location, by considering both TLB hits and TLB misses. It can be calculated using the following formula −

EAT=(H×T)+(M×(T+M.A.T))

Where, **H = TLB Hit Ratio** − The percentage of memory accesses that leads to TLB hit.

- **M = TLB Miss Ratio** − The percentage of memory accesses that leads to TLB miss (1 - TLB Hit Ratio).
- **T = TLB Access Time** − The time taken to access the TLB.
- **M.A.T = Memory Access Time** − The time taken to access the main memory.

For example, if the TLB hit ratio is 90%, TLB access time is 10 nanoseconds, and memory access time is 100 nanoseconds, the EAT can be calculated as follows −

EAT=(0.9×10 ns)+(0.1×(10 ns+100 ns))

=9 ns+11 ns=20 ns

So, the effective access time in this case is 20 nanoseconds.

Advantages

- **Remove External Fragmentation** − Paging helps to reduce external fragmentation issues by dividing memory into fixed-size pages and frames.
- **Efficient Memory Utilization** − By implementing paging, memory can be utilized more efficiently, because pages can be loaded into any available frame in physical memory.
- **Simplified Memory Management** − Paging simplifies management of memory inside the operating system. It eliminates the need for contiguous allocation of memory.
- **Extra Physical Memory** − Paging allows the use of secondary storage (like hard disk) as an extension of physical memory, which is known as virtual memory.

Disadvantages

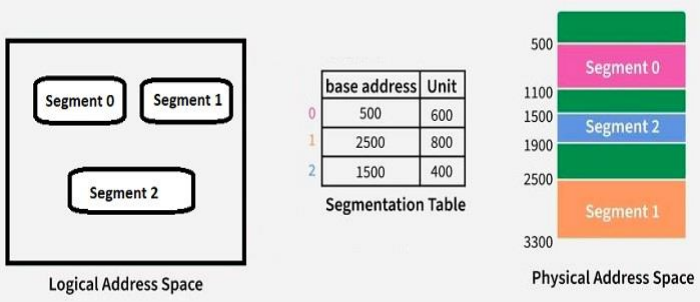
- **Overhead of Page Table** − If the size of the page table is large, it will consume large amount of memory and can lead to performance issues.
- **Increased Memory Access Time** − During address translation, if there is a TLB miss, then it will take more time to access the page table, which can increase the overall memory access time.

- **Extra Complexity** − Implementing paging adds extra complexity while creating operating system.

Segmentation

Segmentation is a memory management technique that divides a program's memory into different segments based on the logical divisions of the program. Each **segment** is used to store a specific part of the program, such as **functions**, **code**, or **variables**. This is different from **paging**, where memory is divided into fixed-size blocks. Here, the segments match the user's logical view of a program (functions, arrays, modules) to physical memory.

The image below shows how segmentation works in an operating system –



In the image, there are three segments - **Segment 0, Segment 1, and Segment 2** in logical memory. Each segment has a **base address** and **unit size** in physical memory, these values are stored in the **segment table**. When a program accesses a memory location, the operating system uses the segment table to translate the logical address into a physical address.

For example, the base address of Segment 0 is 500, and the unit size is 600. So the physical address for logical address (0, 600) will be calculated as −

Physical Address = Base Address + Offset
= 500 + 600
= 1100

Hence, The logical address (500, 1100) is given to segment 0 in physical memory.

Features of Segmentation

- **Variable sized segments:** Each segment can have a different size based on the program's requirements. Each segments represent meaningful units like code, stack, data, or modules.
- **Logical Addressing:** In segmentation, a logical address is represented as a tuple (segment number, offset) where the segment number identifies the segment and the offset specifies the location within that segment.
- **Segment Table:** The operating system maintains a segment table that maps segment numbers to physical memory addresses. Each entry in the segment table contains the base address and limit (size) of the segment.

- **Protection and Sharing:** Different segments can have access rights (read, write, execute) and can be shared among processes.
- **Fragmentation:** Segmentation can lead to external fragmentation, because of loading and unloading of segments of variable sizes. But it does not suffer from internal fragmentation since segments are not fixed in size.

Working of Segmentation

- Before a program is loaded into memory, the operating system divides it into segments based on its **logical structure**. For example, a program may be divided into **code segment**, **data segment**, **stack segment**, etc.
- When a program is executed, the operating system allocates memory for each segment in the **physical memory** (RAM) based on the size of the **segment**.
- The operating system keeps a data structure called the **segment table** that maps each segment number to its corresponding base address and limit in physical memory.
- When a program accesses a memory location, it provides a logical address consisting of a **segment number** and an **offset**. The operating system uses the segment table to translate the logical address into a physical address by adding the base address of the segment to the offset.
- If the offset exceeds the limit of the segment, then a **segmentation fault** occurs. This will prevent the program from accessing memory outside its allocated segment.

The Segment Table

The **segment table** is a data structure used by the operating system to keep track of the segments allocated to a process. It maps two-dimensional logical addresses into one-dimensional physical addresses. Each entry in the segment table contains the following information −

- **Segment Number** − A unique identifier for each segment.
- **Base Address** − The starting physical address of the segment in memory.
- **Limit** − The size of the segment, which defines the maximum offset that can be accessed within the segment.Example

Segment Number	Base Address	Limit
0	1000	500
1	2000	300
2	3000	400

Paging vs Segmentation

Feature	Paging	Segmentation
Definition	Memory management technique that divides memory into fixed-size pages that are mapped to physical frames.	Memory management technique that divides memory into variable-size segments based on the logical divisions of a program.
Division of Memory	Fixed-size pages	Variable-size segments
Addressing	One-dimensional (page number, offset)	Two-dimensional (segment number, offset)
Fragmentation	Likely to suffer from internal fragmentation	May suffer from external fragmentation
Logical Structure	No relation to logical structure of program	Matches logical structure of program, i.e., functions, modules etc
Protection and Sharing	Less protection and sharing capabilities	Better protection and sharing capabilities

Advantages

- **Logical Organization** − Segmentation allows programs to be divided into logical units. This makes it easier to manage and understand the program structure.
- **Protection and Sharing** − Sometimes in programs, different segments can have different access rights. Segmentation can ensure this by various segments having different protection levels.
- **Dynamic Growth** − Segments can grow or shrink dynamically as needed. This is useful for data structures like stacks and heaps.

Disadvantages

- **External Fragmentation** − When segments are loaded and unloaded from memory for long time, the memory blocks become fragmented. This can leads to wastage of memory.
- **Complexity** − The management of variable-sized segments can be more complex than fixed-size pages, leading to increased overhead in the memory management system.

- **Size Limitations** − Each segment has a size limit, which may restrict the amount of memory that can be allocated to a particular segment.

Page Replacement Algorithms

If an operating system uses **paging** for memory management, there is a chance for **page faults** to occur. During a page fault, if no free frame is available in physical memory(RAM), the operating system should decide which page to remove from physical memory to make space for the new page. This decision is made using a **page replacement algorithm**.

There are several page replacement algorithms used by operating systems. Some of these algorithms are good at reducing the number of page faults and some are easy to implement. So the choice of page replacement algorithm can affect the performance of your system. In this chapter, we will discuss following page replacement algorithms −

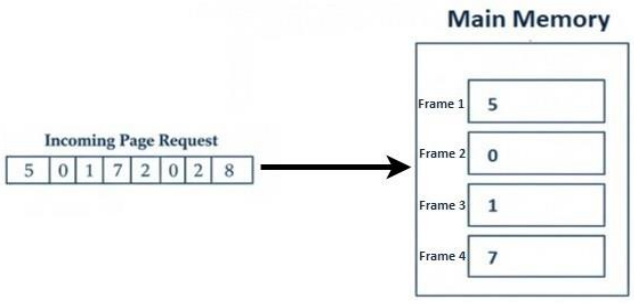
- First-In-First-Out (FIFO) Page Replacement
- Least Recently Used (LRU) Page Replacement
- Least Frequently Used (LFU) Page Replacement
- Optimal Page Replacement

First-In-First-Out (FIFO) Page Replacement

In the **FIFO** page replacement algorithm, the page that has entered in the memory earliest will be replaced first. Meaning the oldest page in memory will be removed to make space for the new page. This algorithm is simple to implement using a queue data structure. The operating system maintains a queue of pages in memory. When a page needs to be replaced, the page at the front of the queue (i.e., the oldest page) is removed, and the new page is added to the back of the queue.

Example

The image below shows how main memory behaves when a page reference string of **5, 0, 1, 7, 2, 0, 2, 8** is processed using FIFO page replacement algorithm with 4 frames:



In this example, the initial pages **5, 0, 1, 7** are loaded directly into the empty frames. Then, the next page **2** is referenced. Now, the frames are full, so according to FIFO, the oldest page **5** will be removed and **2** will be loaded into its place.

This process continues for the rest of the page references. The table below shows each step of the process, including the current page reference –

Step	Reference	Frames (Oldest to Newest)	Page fault?	Removed page
1	5	5	Yes (1)	-
2	0	5, 0	Yes (2)	-
3	1	5, 0, 1	Yes (3)	-
4	7	5, 0, 1, 7	Yes (4)	-
5	2	0, 1, 7, 2	Yes (5)	5 (oldest)
6	0	0, 1, 7, 2	No	-
7	2	0, 1, 7, 2	No	-
8	8	1, 7, 2, 8	Yes (6)	0 (oldest)

- Total Page Faults = 6
- Page Fault Rate = 6/8 = 75%
- Pages Removed = 5, 0

The main drawback of FIFO page replacement algorithm is **Belady's Anomaly**. It states that increasing the number of page frames can sometimes increase the number of page faults. We have a complete chapter on **Belady's Anomaly** where we explained this concept with examples.

Least Recently Used (LRU) Page Replacement

The **LRU** is another popular page replacement algorithm. In this algorithm, the page that has not been used for the longest period of time will be replaced first. Meaning, the page that was least recently used will be removed to make space for the new page. This work like memory cache, where the item that has not been touched for the longest time is removed first.

This algorithm uses **timestamps** to keep track of page usage. When a page is accessed, its timestamp is updated to the current time. When a page needs to be replaced, the page with the oldest timestamp (i.e., the least recently used page) is removed from memory. That's how the LRU algorithm works.

Example

Consider the a page reference string of **1, 2, 3, 1, 4, 5** and with 3 frames of physical memory. That is, Page Reference String: 1, 2, 3, 1, 4, 5

Number of Frames: 3

The initial pages **1, 2, 3** are loaded directly into the 3 empty frames. When the next page **1** is referenced, it is already in memory, so no page fault occurs. Next, page **4** is referenced, this can cause a page fault. So the LRU algorithm will remove page **2**. (If it was FIFO, it would have removed page **1**, because **2** is least recently used and **1** is oldest page).

This process continues for the rest of the page references. The table below shows all the steps of the process –

Step	Reference	Frames (Oldest to Newest)	Page fault?	Removed page
1	1	1	Yes (1)	-
2	2	1, 2	Yes (2)	-
3	3	1, 2, 3	Yes (3)	-
4	1	1, 2, 3	No	-
5	4	1, 3, 4	Yes (4)	2
6	5	3, 4, 5	Yes (5)	1

- Total Page Faults = 5
- Page Fault Rate = 5/6 = 83.33%
- Pages Removed = 2, 1

The main drawback of LRU algorithm is that it can be **expensive** and **difficult** to implement. We need to keep track of the order of page usage, which requires data structures like stacks or linked lists. Also, updating the timestamps for each page access can be costly in terms of time and space complexity. However, LRU is the **most popular** page replacement algorithm used in many operating systems.

Least Frequently Used (LFU) Page Replacement

The **LFU** page replacement algorithm **removes** the page that has been used the **least number of times**. In this algorithm, each page has a counter that keeps track of how many times it has been accessed. When a page needs to be replaced, the page with the lowest access count is removed from memory. If the access counts are the same, then you can use FIFO or LRU to break the tie (depending on the implementation).

Example

Consider the following page reference string and frames −

Reference string: 1, 2, 3, 1, 2, 4, 1, 2, 5

Frames = 3

The initial pages **1, 2, 3** are loaded directly into the empty frames. All of these pages will have an access count of 1. When the next page **1** and **2** are referenced, they are already in memory, so their access counts will be incremented to 2. Next page **4** is referenced, which causes a page fault. So the LFU algorithm will remove page **3** since the access count of page 3 is 1 (lowest).

This process continues for the rest of the page references. The table below shows all the steps of the process. Here the **1:2** means page 1 has a frequency of 2.

Step	Reference	Frequency Table (1,2,3,4,5)	Frames	Page Fault?
1	1	1:1	1	Yes
2	2	1:1, 2:1	1,2	Yes
3	3	1:1, 2:1, 3:1	1,2,3	Yes
4	1	1:2, 2:1, 3:1	1,2,3	No
5	2	1:2, 2:2, 3:1	1,2,3	No
6	4	1:2, 2:2, 3:1 → Remove 3 (lowest freq=1)	1,2,4	Yes
7	1	1:3, 2:2, 4:1	1,2,4	No
8	2	1:3, 2:3, 4:1	1,2,4	No
9	5	1:3, 2:3, 4:1 → Remove 4 (lowest freq=1)	1,2,5	Yes

- Total Page Faults = 5
- Page Fault Rate = 5/9 = 55.56%
- Pages Removed = 3, 4

Compared to FIFO and LRU, the LFU algorithm have improved page fault rate (55.56% vs 75% and 83.33%). But there is a drawback of LFU algorithm. It can lead to **starvation** of pages that are not regularly used. For example, if a page is accessed once at the beginning and then not used again, it will hard to get back into memory because its access count will remain low.

Optimal Page Replacement

The **optimal page replacement** algorithm is a theoretical concept which is very difficult to be implemented in real operating systems. This algorithm works by replacing the page that will not be used for the longest period of time in the future. Meaning, it involves predicting what pages will be used in the future and replacing the one that will not be used for the longest time. This is considered as the **best page replacement** algorithm.

Example

Consider the following page reference string and frames −

Page Reference String: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3

Frames: 4

Here, the first four pages **7, 0, 1, 2** are loaded directly into the first four empty frames. When the next page **0** is referenced, it is already in memory, so no page fault occurs. Next, page **3** is referenced, which causes a page fault. So the optimal algorithm will remove page **1** since it will not be used for the longest period of time in the future (it won't be used again).

We are able to implement this algorithm as we already know the entire page reference string. But that will not be the case in real-world scenarios. We never know what pages will be referenced in the future. That's why this algorithm is just a theoretical model.

The table below shows all the steps of the process –

7	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**3**
			2	2	2	2	2	2	2	2	2	2	2
		1	1	1	1	1	4	4	4	4	4	4	4
	0	0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	7	7	3	3	3	3	3	3	3	3	3
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit	Hit	Hit	Hit	Hit	Hit

Question No.	**Questions**
Unit – IV :	
PART – A (Two Marks Questions)	
1	What is mass storage structure?
2	Define disk scheduling.
3	What is paging?
4	Explain disk structure briefly.
5	What is the difference between logical and physical address space?
6	Describe contiguous memory allocation.
7	How does disk scheduling improve system performance?
8	Give an example of swapping in memory management.
9	Differentiate between paging and segmentation.
10	Compare FCFS and SSTF disk scheduling algorithms.
11	Why is page replacement necessary in memory management?
12	Evaluate the importance of disk management techniques.
13	Design a simple page table structure.
14	Construct a scenario illustrating disk scheduling.
15	Analyze the advantages of segmentation over paging.
PART – B(Ten Marks Questions)	
1	Describe the difference between primary, secondary, and tertiary storage with examples.
2	a)Explain the structure of a disk including tracks, sectors, and cylinders. b)Illustrate how logical block addressing maps to physical locations on a disk.
3	Describe the different methods of disk attachment with example.
4	Compare FCFS, SSTF, SCAN, and C-SCAN disk scheduling algorithms with examples.
5	What are the main functions of disk management in an operating system?
6	Explain the role of the Memory Management Unit (MMU) in mapping logical to physical addresses.
7	a)Explain the concept of swapping and its advantages. b)Analyze contiguous memory allocation methods and their limitations.
8	a)Explain the concept of paging in memory management. b)Describe the structure of a page table and how it is used to map logical addresses to physical addresses.
9	a)Define segmentation and explain how it differs from paging. b) Illustrate a segmented memory system with an example and explain how it manages program segments.
10	Compare and analyze the following page replacement algorithms: FIFO, LRU, Optimal, and LFU algorithm.

UNIT V

FILE SYSTEM

“Information is the oil of the 21st century, and analytics is the combustion engine.”

— Peter Sondergaard

Unit V- Overview

This unit focuses on how operating systems manage and organize data through file systems. It explains the concept of files, methods of accessing data, and how directories and disks are structured for efficient storage and retrieval. It also covers advanced concepts such as file system mounting, file sharing, and implementation techniques. Additionally, the unit includes case studies of Linux and Windows 2000 operating systems, highlighting their history and design principles. Overall, this unit provides a deep understanding of how data is stored, accessed, and managed in modern computer systems.

Objectives

- To understand the concept and attributes of files in an OS
- To learn different file access methods
- To study directory structures and disk organization
- To understand file system mounting and sharing mechanisms
- To analyze file system implementation techniques
- To explore real-world OS design through Linux and Windows 2000

Learning Outcomes

After studying this unit, students will be able to:

- Define and explain file system concepts
- Compare different file access methods
- Describe directory and disk structures
- Explain how file systems are mounted and shared
- Understand implementation of file systems
- Analyze design principles of Linux and Windows systems

Importance of Studying this Unit

Understanding file systems is essential because they are the backbone of data storage in any computer system. This unit helps students learn how operating systems efficiently manage large volumes of data, ensure data security, and provide fast access. It also builds practical knowledge required for system programming, database management, and real-world OS design. Studying Linux and Windows case studies gives insight into how theoretical concepts are applied in real operating systems, making it highly relevant for both academic and industry applications.

Key Concepts

1. Concept of a File

- A file is a collection of related data stored on secondary storage
- Attributes: name, type, size, location, permissions
- Types: text files, binary files, executable files

2. Access Methods

- **Sequential Access**: Data accessed in order
- **Direct Access**: Random access using index/position
- **Indexed Access**: Uses an index to locate data quickly

3. Directory Structure

- Organizes files in a hierarchical manner
- Types:
    - Single-level directory
    - Two-level directory
    - Tree-structured directory
- Helps in file organization and retrieval

4. Disk Structure

- Disk divided into tracks and sectors
- Blocks are the basic unit of storage
- OS manages allocation and free space

5. File System Mounting

- Process of attaching a file system to a directory
- Makes external or additional storage accessible
- Example: mounting USB drives

6. File Sharing

- Allows multiple users to access files
- Controlled using permissions:
    - Read
    - Write
    - Execute
- Ensures security and collaboration

7. File System Implementation

- Techniques used to manage files:
    - Allocation methods (contiguous, linked, indexed)
    - Free space management (bit map, free list)
- Improves efficiency and reliability

8. Linux System (Case Study)

- **History**: Developed in 1991 by Linus Torvalds
- **Design Principles**:
  - Open-source
  - Multiuser and multitasking
  - Strong security and stability
- Uses hierarchical file system

9. Windows 2000 OS (Case Study)

- **History**: Developed by Microsoft as part of Windows NT family
- **Design Principles**:
  - User-friendly interface
  - Robust security
  - Support for enterprise systems
- **Uses NTFS file system**

File System Interface: The Concept of a File

File System Interface

A **file system** provides a mechanism for storing, organizing, and retrieving data in a structured manner on mass storage devices. The file system interface defines how programs and users interact with the file system to manage files.

The Concept of a File

A **file** is a collection of data stored on a storage device that is identified by a name and can be accessed and manipulated. Files can store a wide range of data types, including text, images, videos, databases, and programs. In most file systems, files are abstracted as a sequence of bytes or blocks.

File Attributes:

- **File Name:** The unique identifier of a file in a directory.
- **File Type:** Specifies the type of file (e.g., text, executable, image).
- **File Size:** The size of the file (in bytes).
- **Creation Date and Time:** When the file was created.
- **Permissions:** Defines who can read, write, or execute the file.
- **Location:** The location of the file in physical storage.
- **Owner:** The user or process that owns the file.

File Operations in OS

The Operating system is responsible for performing the following file operations using various system calls:

1. Read
2. Write
3. Create
4. Delete
5. Truncate files
6. Reposition

Read files: The OS needs a read pointer to read a file from a specific location in the file.

To read files through command line, we use the **type** command. For huge files, we use the **more** command to read files one page at a time.

Write files: The system call uses the same pointers to write a file, it helps to save space and reduce complexity.

Create files: The user should have enough space to create a file. When a file is created, a directory entry is made.

To create files through command line, the syntax is **type nul > (filename).(filetype)**

Delete files: The user can look up the file name and delete it. It will release the space occupied by the file and remove the directory entries for the file.

The **del** command is used when we want to delete a file through command line.

Truncate files: The user can delete information from the file, instead of deleting it as a whole. This changes the file length, though the other attributes remain the same.

Repositioning files: The current file-position pointer can be repositioned to a new given value.

There are also other file operations like appending a file, creating a duplicate of the file, and renaming a file.

File types:

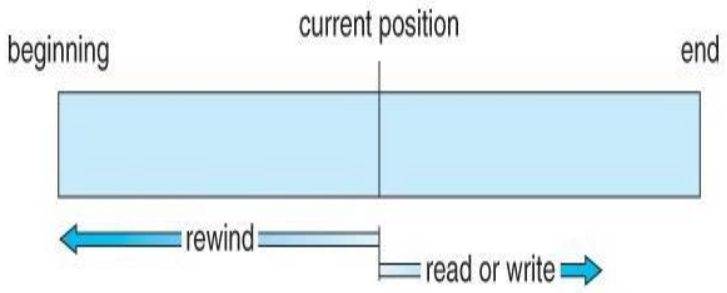
A common technique for implementing file types is to include the type as part of the file name. The name is split into two parts – a name and an extension separated by a period character. The system uses the extension to indicate the type of the file and the type of operations that can be done on that file.

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, 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, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Access Method

Sequential Access

- A sequential access file emulates magnetic tape operation, and generally supports a few operations:
    - read next - read a record and advance the tape to the next position.
    - write next - write a record and advance the tape to the next position.
    - rewind
    - skip n records - May or may not be supported. N may be limited to positive numbers, or may be limited to +/- 1.

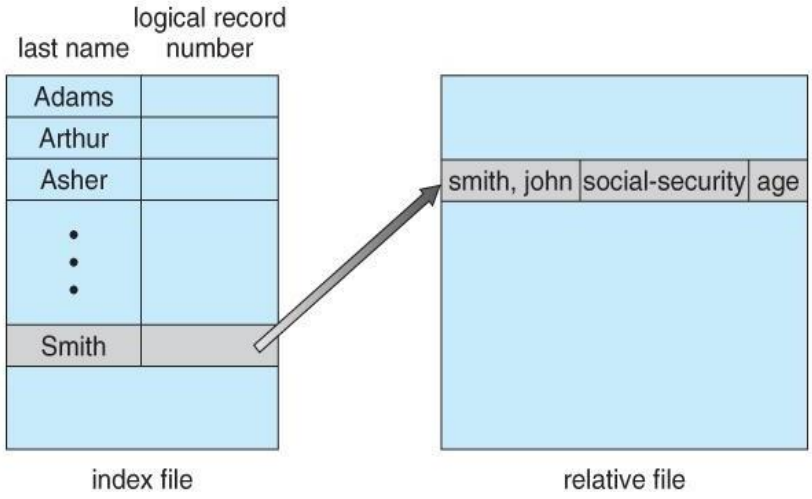


Direct Access

- Jump to any record and read that record. Operations supported include:
    - read n - read record number n. (Note an argument is now required.)
    - write n - write record number n. (Note an argument is now required.)
    - jump to record n - could be 0 or the end of file.
    - Query current record - used to return back to this record later.
    - Sequential access can be easily emulated using direct access. The inverse is complicated and inefficient.

Other Access Methods

- An indexed access scheme can be easily built on top of a direct access system. Very large files may require a multi-tiered indexing scheme, i.e. indexes of indexes.

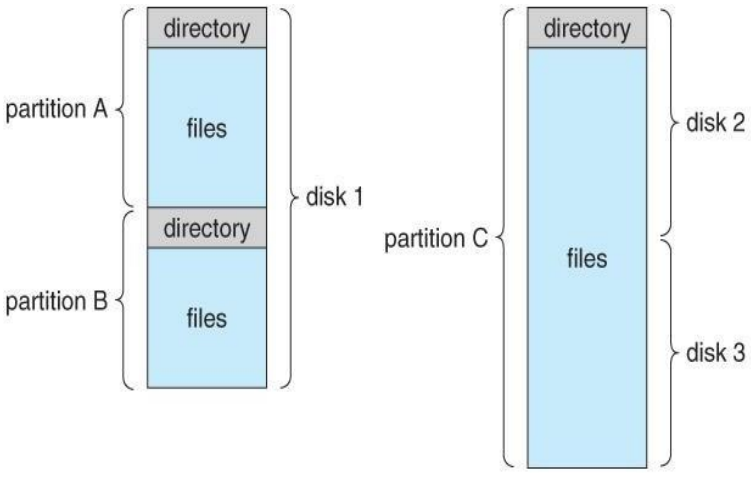


Example of index and relative files

Directory & Disk Structure

Storage Structure

- A disk can be used in its entirety for a file system.
- Alternatively, a physical disk can be broken up into multiple *partitions, slices, or mini-disks*, each of which becomes a virtual disk and can have its own filesystem. (or be used for raw storage, swap space, etc.)
- Or, multiple physical disks can be combined into one *volume*, i.e. a larger virtual disk, with its own filesystem spanning the physical disks.



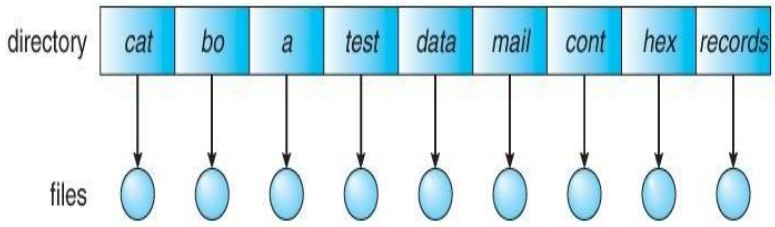
typical file-system organization

Directory Overview

- **Directory operations to be supported include:**
  - Search for a file
  - Create a file - add to the directory
  - Delete a file - erase from the directory
  - List a directory - possibly ordered in different ways.
  - Rename a file - may change sorting order
  - Traverse the file system.

1. Single-Level Directory

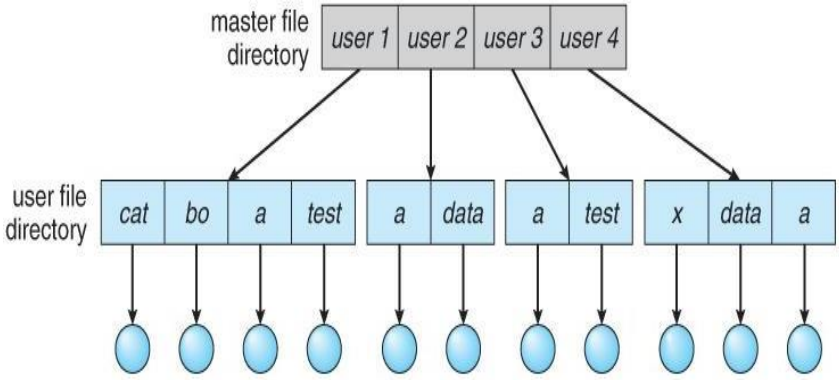
- Simple to implement, but each file must have a unique name.



Single-level directory

2. Two-Level Directory

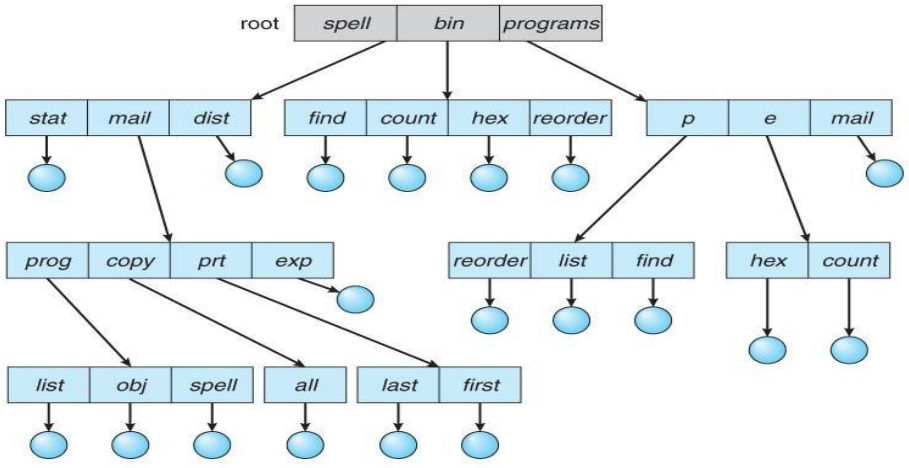
- Each user gets their own directory space.
- File names only need to be unique within a given user's directory.
- A master file directory is used to keep track of each users directory, and must be maintained when users are added to or removed from the system.
- A separate directory is generally needed for system (executable) files.
- Systems may or may not allow users to access other directories besides their own
  - If access to other directories is allowed, then provision must be made to specify the directory being accessed.
  - If access is denied, then special consideration must be made for users to run programs located in system directories. A *search path* is the list of directories in which to search for executable programs, and can be set uniquely for each user.



Two-level directory structure.

3. Tree-Structured Directories

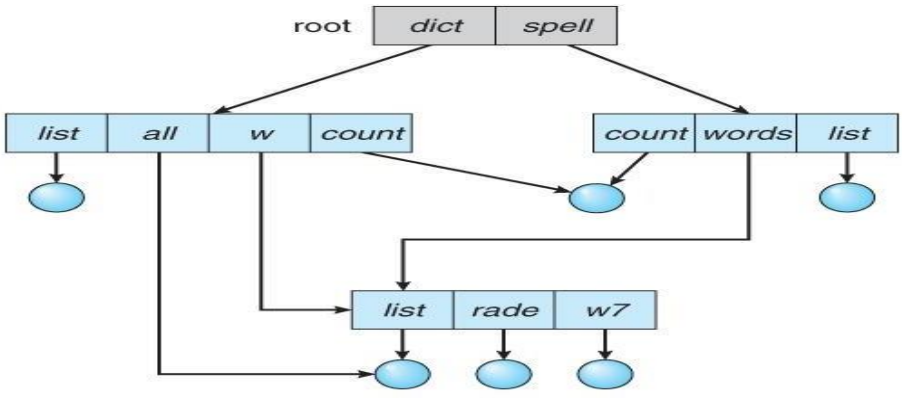
- An obvious extension to the two-tiered directory structure, and the one with which we are all most familiar.
- Each user / process has the concept of a *current directory* from which all (relative) searches take place.
- Files may be accessed using either absolute pathnames (relative to the root of the tree) or relative pathnames (relative to the current directory.)
- Directories are stored the same as any other file in the system, except there is a bit that identifies them as directories, and they have some special structure that the OS understands.
- One question for consideration is whether or not to allow the removal of directories that are not empty - Windows requires that directories be emptied first, and UNIX provides an option for deleting entire sub-trees.



Tree-Structured Directories

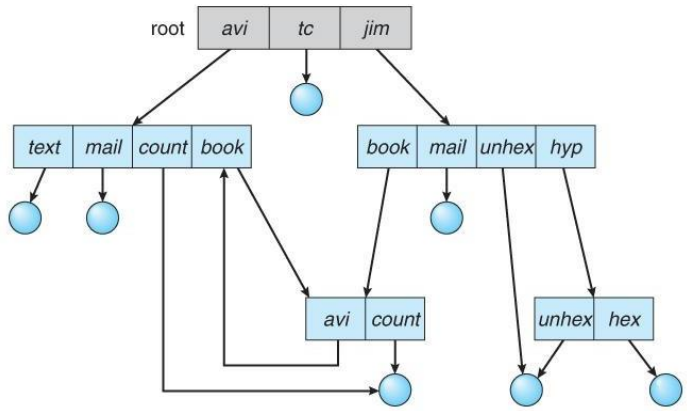
4. Acyclic-Graph Directories

- When the same files need to be accessed in more than one place in the directory structure (e.g. because they are being shared by more than one user / process), it can be useful to provide an acyclic-graph structure. (Note the *directed* arcs from parent to child.)
- UNIX provides two types of *links* for implementing the acyclic-graph structure. (See "man ln" for more details.)
    - A *hard link* (usually just called a link) involves multiple directory entries that both refer to the same file. Hard links are only valid for ordinary files in the same filesystem.
    - A *symbolic link*, that involves a special file, containing information about where to find the linked file. Symbolic links may be used to link directories and/or files in other filesystems, as well as ordinary files in the current filesystem.
- Windows only supports symbolic links, termed *shortcuts.*
- Hard links require a *reference count*, or *link count* for each file, keeping track of how many directory entries are currently referring to this file. Whenever one of the references is removed the link count is reduced, and when it reaches zero, the disk space can be reclaimed.
- For symbolic links there is some question as to what to do with the symbolic links when the original file is moved or deleted:
    - One option is to find all the symbolic links and adjust them also.
    - Another is to leave the symbolic links dangling, and discover that they are no longer valid the next time they are used.
    - What if the original file is removed, and replaced with another file having the same name before the symbolic link is next used?



5. General Graph Directory

- If cycles are allowed in the graphs, then several problems can arise:
    - o Search algorithms can go into infinite loops. One solution is to not follow links in search algorithms. (Or not to follow symbolic links, and to only allow symbolic links to refer to directories.)
    - o Sub-trees can become disconnected from the rest of the tree and still not have their reference counts reduced to zero. Periodic garbage collection is required to detect and resolve this problem. (chkdsk in DOS and fsck in UNIX search for these problems, among others, even though cycles are not supposed to be allowed in either system. Disconnected disk blocks that are not marked as free are added back to the file systems with made-up file names, and can usually be safely deleted.)

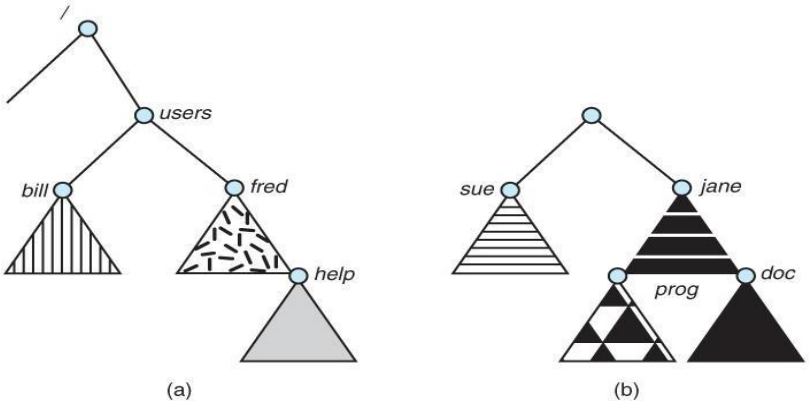


General graph directory

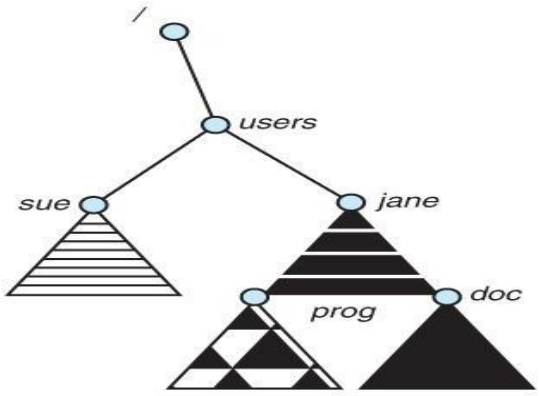
File System Mounting

- The basic idea behind mounting file systems is to combine multiple file systems into one large tree structure.
- The mount command is given a filesystem to mount and a *mount point* (directory) on which to attach it.
- Once a file system is mounted onto a mount point, any further references to that directory refer to the root of the mounted file system.
- Any files (or sub-directories) that had been stored in the mount point directory prior to mounting the new filesystem are now hidden by the mounted filesystem, and are no longer available. For this reason, some systems only allow mounting onto empty directories.

- Filesystems can only be mounted by root, unless root has previously configured certain filesystems to be mountable onto certain pre-determined mount points. (E.g. root may allow users to mount floppy file systems to /mnt or something like it.) Anyone can run the mount command to see what file systems are currently mounted.
- Filesystems may be mounted read-only, or have other restrictions imposed.



File system. (a) Existing system. (b) Unmounted volume.



Mount point

- The traditional Windows OS runs an extended two-tier directory structure, where the first tier of the structure separates volumes by drive letters, and a tree structure is implemented below that level.
- Macintosh runs a similar system, where each new volume that is found is automatically mounted and added to the desktop when it is found.
- More recent Windows systems allow filesystems to be mounted to any directory in the file system, much like UNIX.

File Sharing

File Sharing in an Operating System(OS) denotes how information and files are shared between different users, computers, or devices on a network; and files are units of data that are stored in a computer in the form of documents/images/videos or any others types of information needed.

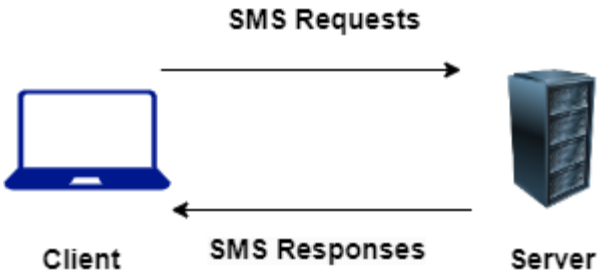
For Example: Suppose letting your computer talk to another computer and exchange pictures, documents, or any useful data. This is generally useful when one wants to work on a project with others, send files to friends, or simply shift stuff to another device. Our OS provides ways to do this like email attachments, cloud services, etc. to make the sharing process easier and more secure.

Various Ways to Achieve File Sharing

1. Server Message Block (SMB)

SMB is like a network based file sharing protocol mainly used in windows operating systems. It allows our computer to share files/printer on a network. SMB is now the standard way for seamless file transfer method and printer sharing.

Example: Imagine in a company where the employees have to share the files on a particular project . Here SMB is employed to share files among all the windows based operating system.orate on projects. SMB/CIFS is employed to share files between Windows-based computers. Users can access shared folders on a server, create, modify, and delete files.

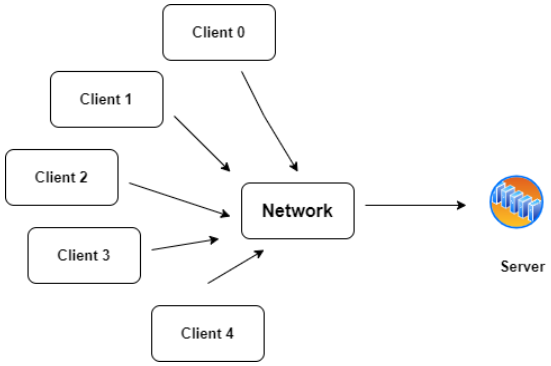


SMB File Sharing

2. Network File System (NFS)

NFS is a distributed based file sharing protocol mainly used in Linux/Unix based operating System. It allows a computer to share files over a network as if they were based on local. It provides a efficient way of transfer of files between servers and clients.

Example: Many Programmer/Universities/Research Institution uses Unix/Linux based Operating System. The Institutes puts up a global server datasets using NFS. The Researchers and students can access these shared directories and everyone can collaborate on it.

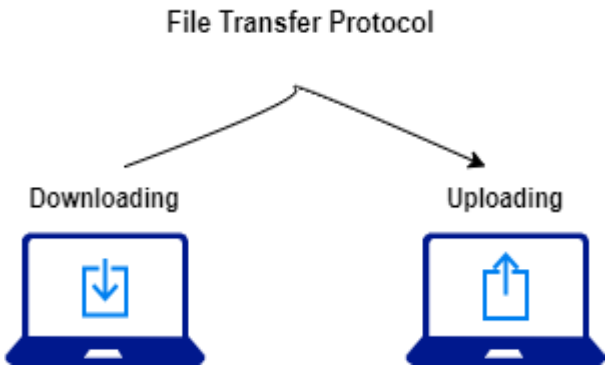


NFS File Sharing

3. File Transfer Protocol (FTP)

It is the most common standard protocol for transferring of the files between a client and a server on a computer network. FTPs supports both uploading and downloading of the files, here we can download,upload and transfer of files from Computer A to Computer B over the internet or between computer systems.

Example: Suppose the developer makes changes on the server. Using the FTP protocol, the developer connects to the server they can update the server with new website content and updates the existing file over there.



FTP File Sharing

4. Cloud-Based File Sharing

It involves the famous ways of using online services like Google Drive, DropBox , One Drive ,etc. Any user can store files over these cloud services and they can share that with others, and providing access from many users. It includes collaboration in realtime file sharing and version control access.

Ex: Several students working on a project and they can use Google Drive to store and share for that purpose. They can access the files from any computer or mobile devices and they can make changes in realtime and track the changes over there.

Cloud Based File Sharing

These all file sharing methods serves different purpose and needs according to the requirements and flexibility of the users based on the operating system.

File system implementation

File system implementation is a critical aspect of an operating system as it directly impacts the performance, reliability, and security of the system. Different operating systems use different file system implementations based on the specific needs of the system and the intended use cases. Some common file systems used in operating systems include NTFS and FAT in Windows, and ext4 and XFS in Linux.

Components of File System Implementation

The file system implementation includes several components, including:

- **File System Structure:** The file system structure refers to how the files and directories are organized and stored on the physical storage device. This includes the layout of file systems data structures such as the directory structure, file allocation table, and inodes.
- **File Allocation:** The file allocation mechanism determines how files are allocated on the storage device. This can include allocation techniques such as contiguous allocation, linked allocation, indexed allocation, or a combination of these techniques.
- **Data Retrieval:** The file system implementation determines how the data is read from and written to the physical storage device. This includes strategies such as buffering and caching to optimize file I/O performance.
- **Security and Permissions:** The file system implementation includes features for managing file security and permissions. This includes access control lists (ACLs), file permissions, and ownership management.
- **Recovery and Fault Tolerance:** The file system implementation includes features for recovering from system failures and maintaining data integrity. This includes techniques such as journaling and file system snapshots.

Different Types of File Systems

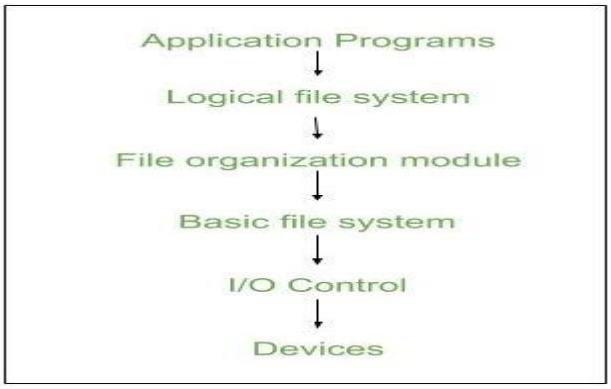
There are several types of file systems, each designed for specific purposes and compatible with different operating systems. Some common file system types include:

- **FAT32 (File Allocation Table 32):** Commonly used in older versions of Windows and compatible with various operating systems.
- **NTFS (New Technology File System):** Used in modern Windows operating systems, offering improved performance, reliability, and security features.

- **ext4 (Fourth Extended File System):** Used in Linux distributions, providing features such as journaling, large file support, and extended file attributes.
- **HFS+ (Hierarchical File System Plus):** Used in macOS systems prior to macOS High Sierra, offering support for journaling and case-insensitive file names.
- **APFS (Apple File System):** Introduced in macOS High Sierra and the default file system for macOS and iOS devices, featuring enhanced performance, security, and snapshot capabilities.
- **ZFS (Zettabyte File System):** A high-performance file system known for its advanced features, including data integrity, volume management, and efficient snapshots.

Layers in File System

A file system in an operating system is organized into multiple layers, each responsible for different aspects of file management and storage. Here are the key layers in a typical file system:



Layers in File System

- **Application Programs:** This is the topmost layer where users interact with files through applications. It provides the user interface for file operations like creating, deleting, reading, writing, and modifying files. Examples include text editors, file browsers, and command-line interfaces.
- **Logical File system -** It manages metadata information about a file i.e includes all details about a file except the actual contents of the file. It also maintains via file control blocks. File control block (FCB) has information about a file - owner, size, permissions, and location of file contents.
- **File Organization Module -** It has information about files, the location of files and their logical and physical blocks. Physical blocks do not match with logical numbers of logical blocks numbered from 0 to N. It also has a free space that tracks unallocated blocks.

- **Basic File system -** It Issues general commands to the device driver to read and write physical blocks on disk. It manages the memory buffers and caches. A block in the buffer can hold the contents of the disk block and the cache stores frequently used file system metadata.

- **I/O Control level -** Device drivers act as an interface between devices and OS, they help to transfer data between disk and main memory. It takes block number as input and as output, it gives low-level hardware-specific instruction.

    - **Devices Layer:** The bottommost layer, consisting of the actual hardware devices. It performs the actual reading and writing of data to the physical storage medium. This includes hard drives, SSDs, optical disks, and other storage devices.

Case Studies: The Linux System

Linux History

Linux is a free and open-source operating system that has become one of the most popular and widely used operating systems in the world. Its development traces back to the early 1990s, and its success lies in its open-source nature, flexibility, and community-driven approach.

Key Milestones in Linux History:

- **1991 - Creation of Linux:** Linux was created by **Linus Torvalds**, a Finnish student, who initially started the project as a personal endeavor. The first version, **Linux 0.01**, was released in 1991, and it was initially a clone of the Unix operating system.

- **1992 - Open Source Movement:** In 1992, the Linux kernel was officially released under the **GNU General Public License (GPL)**, allowing anyone to use, modify, and distribute the software freely. This shift to an open-source model was crucial for the system's growth.

- **1994 - Linux 1.0:** The first official version of the Linux kernel was released, marking a milestone in its development.

- **2000s - Widespread Adoption:** By the early 2000s, Linux began to be adopted by enterprises, server farms, and tech enthusiasts. Its stability, scalability, and low cost made it the system of choice for many.

- **2003 - Emergence of Linux Distributions:** Numerous distributions (distros) of Linux began to emerge, such as **Red Hat**, **Debian**, **Ubuntu**, and **Fedora**, each catering to different user needs.

- **2000s-Present - Dominance in Server and Cloud Computing:** Linux became the dominant operating system for servers, web hosting, cloud computing, and supercomputing.

- **2010s-Present - Mobile Computing and IoT:** The Linux-based **Android** operating system revolutionized mobile computing, becoming the most widely used mobile operating

system. Additionally, Linux expanded its use in Internet of Things (IoT) devices.

Design Principles of Linux

The Linux operating system is built around several core principles that distinguish it from other operating systems. These principles focus on flexibility, modularity, and openness, ensuring that Linux remains scalable, customizable, and adaptable to a wide range of use cases.

Core Design Principles:

Open Source and Free Software:

- Linux follows the principles of **open-source software**, which means that its source code is freely available for modification and distribution. The project adheres to the **GNU General Public License (GPL)**, ensuring that derivative works are also open- source.
- This has led to a **community-driven development model**, with thousands of developers contributing to its growth.

Modularity and Flexibility:

- The Linux kernel is designed to be **modular**, meaning that various features and functionalities (such as device drivers and file systems) can be added or removed based on specific needs. This modularity allows Linux to run on a wide variety of hardware, from small embedded devices to large servers.
- Users can customize their Linux system to a large extent, tailoring it to their use case by selecting the kernel, tools, and software that best fit their needs.

Multitasking and Multiuser Support:

- Linux is a **multiuser** operating system, allowing multiple users to use the system simultaneously while maintaining privacy and security. Each user has their own environment, which is isolated from other users.
- It also supports **preemptive multitasking**, allowing multiple applications to run concurrently, improving system performance and responsiveness.

Security:

- Security is a core principle of the Linux design. The system follows a **least privilege** model, meaning users and processes are only given the permissions they need to perform their tasks. The kernel enforces this model using mechanisms such as **user IDs (UIDs)** and **group IDs (GIDs)**.
- Linux also includes **Security Modules (SELinux, AppArmor)** that provide enhanced security by enforcing policies and restricting the actions of processes.

Portability:

- Linux is designed to be **highly portable**, meaning it can run on various hardware architectures, including x86, ARM, PowerPC, and others. The kernel is written in **C**, a language known for its portability across different platforms.
- This portability has made Linux ideal for diverse applications, from **embedded systems** to **mainframe computers**.

Performance and Efficiency:

- The Linux kernel is designed for **high performance**, ensuring efficient resource management. It is optimized to run efficiently on a wide range of hardware and offers high scalability, making it suitable for both low-resource embedded devices and high-performance computing environments.
- Linux uses efficient **memory management** and **disk scheduling** algorithms to optimize system resources.

Interoperability:

- Linux was designed to be **compatible with Unix systems**, ensuring that it could run applications designed for Unix-like systems. Linux also supports **POSIX standards**, allowing interoperability with other Unix-based systems.
- Additionally, Linux supports various networking protocols, file systems, and has tools for interoperability with Windows and macOS systems.

Unified System Interface:

- Linux provides a **single interface** for both system administration and user applications. It uses a **command-line interface (CLI)** along with powerful tools and shell scripting, giving users fine-grained control over the system.
- The **Graphical User Interface (GUI)** is available in various Linux distributions, such as **GNOME** and **KDE**, but the command-line interface remains a powerful tool for managing the system.

Community Collaboration:

- The Linux kernel is developed and maintained by a **global community of developers**, and decisions about the kernel are made by **Linus Torvalds** along with the kernel maintainers and contributors. This open collaboration model encourages innovation and continuous improvement.

Windows 2000 Operating System

Windows 2000 was a significant milestone in Microsoft's history, serving as the successor to Windows NT 4.0. It was a stable and secure operating system designed for both professional workstations and server environments. Released in February 2000, Windows 2000 integrated new features and enhancements that improved performance, security, and usability over previous versions.

History of Windows 2000

Development and Release:

- **Windows 2000** was part of the **Windows NT family**, designed primarily for business and enterprise use. It was aimed at providing stability, security, and scalability, with a focus on networking and reliability.
- **Release Date:** February 17, 2000.
- **Predecessor:** Windows NT 4.0 (released in 1996) was the previous version in the Windows NT series.
- **Successor:** Windows XP (released in 2001) was the direct successor to Windows 2000, bringing many of the same core features to home users with a more user-friendly interface.

Target Audience:

- windows 2000 was targeted at both **business** and **enterprise** environments, as well as **workstations**. It was designed to be stable enough for servers, while still providing a user-friendly environment for individual users.

Editions:

- Windows 2000 was available in several editions tailored to different use cases:
  - **Windows 2000 Professional:** Targeted at desktop workstations.
  - **Windows 2000 Server:** Targeted at server environments.
  - **Windows 2000 Advanced Server:** Aimed at more robustserver configurations with greater scalability.

Design Principles of Windows 2000

Windows 2000 was built with several core design principles to meet the growing demands of business, enterprise, and professional environments. These principles emphasized stability, performance, security, and ease of use.

Core Design Principles:

Stability and Reliability:

- Windows 2000 was designed to be a **highly stable operating system**, capable of running for long periods without crashing or requiring frequent reboots. This stability was a key feature for both **servers** and **workstations**.
- Windows 2000 leveraged the **Windows NT kernel**, which had been developed over several years to provide a robust, stable foundation for multi-user and multitasking environments.

Security:

- Windows 2000 incorporated **enhanced security** features to make it suitable for business and enterprise environments where data security was a top priority.
  - **User Authentication:** Improved login processes and authentication mechanisms, supporting **Kerberos** authentication and **smart cards**.
  - **Active Directory:** The introduction of Active Directory provided centralized management of users and resources across a network.
  - **Group Policy:** Administrators could apply security policies and settings across multiple machines.
  - **File Encryption (EFS):** Windows 2000 introduced the **Encrypting File System (EFS)**, allowing users to encrypt files to protect sensitive data from unauthorized access.

Networking and Internet Support:

Windows 2000 was designed to offer strong **networking capabilities**, suitable for small to large enterprise networks.

- **Support for TCP/IP, DHCP, and DNS**: These were critical for integration into modern networks and the Internet.
- **Improved Remote Access**: It included remote access features like **Virtual Private Network (VPN)** and **Terminal Services**, enabling remote administration and access to networked resources.
- **Internet Connection Sharing (ICS)**: Allowed sharing a single Internet connection among multiple devices on a local network.

Compatibility:

- Windows 2000 was designed to run older applications designed for **Windows NT 4.0**, but also included support for **Windows 9x** applications, making it a highly compatible OS for businesses transitioning from older systems.

- It supported the **Win32 API**, enabling compatibility with existing Windows software. Additionally, it maintained backward compatibility with DOS applications, and it supported **Plug and Play** for hardware installation.

Ease of Use:

- Windows 2000 introduced a more **user-friendly interface**, with improvements over previous versions, including **new UI elements**, such as better taskbars, window management, and **Start Menu enhancements**.
- While it retained much of the traditional Windows NT interface, it featured enhancements designed to streamline daily usage for both professionals and end users.

Scalability:

- Designed to handle both **entry-level workstations** and **high-performance servers**, Windows 2000 supported a wide range of hardware configurations and offered advanced scalability.

- Processor Support**:** It supported both **x86** and **Alpha processors**, with better scalability for systems with **multiple processors**.
- Servers could be scaled for large enterprise environments, with features like support for **large amounts of RAM** and more advanced memory management techniques

Multiuser Support:

- Like its predecessors, Windows 2000 was designed with **multi-user** capabilities, allowing multiple users to work on the same machine without interfering with each other. Windows 2000 allowed each user to have their own personalized settings, files, and preferences.

- **Terminal Services** allowed remote desktop sessions, enabling users to log in remotely and work as if they were physically present at the machine.

Hardware Support:

- Windows 2000 included support for the **latest hardware technologies** of its time, such as **USB 2.0**, **DVD drives**, and improved **video and audio drivers**. This ensured that it could run on a wide variety of hardware platforms.

Reliability for Servers:

- Windows 2000 was designed with enterprise and server environments in mind. It introduced several important features for servers:

  - **Clustering** and **load balancing**: To ensure high availability for business-critical applications.

  - **Active Directory** provided centralized management and a hierarchical structure for network resources.

  - **Backup and restore** features were improved to ensure data integrity.

Question No.	Questions
Unit – V : FILE SYSTEM	
PART – A (Two Marks Questions)	
1	What is a file in an operating system?
2	List any two file access methods.
3	What is file system mounting?
4	Explain the concept of a directory structure.
5	What is file sharing in OS?
6	Describe disk structure briefly.
7	How is sequential access used to read a file?
8	Give an example of file sharing in a multi-user system.
9	Differentiate between sequential and direct access methods.
10	Compare single-level and tree-structured directories.
11	Why is file system mounting important?
12	Evaluate the design principles of the Linux system.
13	Design a simple directory structure for storing student records.
14	Construct a basic model for file system implementation.
15	List out the design principles of Linux and Windows 2000 OS.
PART – B(Ten Marks Questions)	
1	Define a file and explain the difference between a text file and a binary file with examples.
2	Explain the sequential and direct access methods in file systems with example
3	a) Explain the concept of a root directory and subdirectories. b) Compare the tree-structured and acyclic-graph directory structures in terms of flexibility and complexity.
4	a)Explain the concept of disk blocks, sectors, and tracks. b) Illustrate how a file is stored across multiple disk blocks using linked allocation.
5	a)What is the purpose of mounting a file system in an OS? b)Analyze how mounting multiple file systems in Linux allows for a unified directory structure.
6	a)Explain the concept of shared files and access rights. b)Evaluate the security challenges involved in file sharing in multi- user systems.
7	Explain the structure and purpose of File Allocation Table (FAT) in a file system.
8	a)List the major distributions of Linux and their historical significance. b)Analyze how Linux's modular design supports flexibility and hardware compatibility.
9	Explain the concept of Windows 2000 Active Directory.
10	Compare the file management approaches in Linux and Windows 2000.