Bachelor of Computer Applications (BCA)

OPRATING SYSTEM (DBCACO301T24)

Self-Learning Material (SEM III)

Jaipur National University
Centre for Distance and Online Education

Established by Government of Rajasthan
Approved by UGC under Sec 2(f) of UGC ACT 1956
&
NAAC A+ Accredited

TABLE OF CONTENTS

EXPERT COMMITTEE

Prof. Sunil Gupta
(Computer and Systems Sciences, JNU Jaipur)

Dr. Deepak Shekhawat
(Computer and Systems Sciences, JNU Jaipur)

Dr. Satish Pandey
(Computer and Systems Sciences, JNU Jaipur)

COURSE COORDINATOR

Mr. Satender Singh
(Computer and Systems Sciences, JNU Jaipur)

UNIT PREPARATION

Unit Writer(s)	**Assisting & Proofreading**	**Unit Editor**
Mr. Satender Singh (Computer and Systems Sciences, JNU Jaipur) (Unit 1- 4)	Ms. Rachana Yadav (Computer and Systems Sciences, JNU Jaipur)	Dr. Shalini Rajawat (Computer and Systems Sciences, JNU Jaipur)
Ms. Heena Shrimali (Computer and Systems Sciences, JNU Jaipur) (Unit 5-8)		

Secretarial Assistance

Mr. Mukesh Sharma

COURSE INTRODUCTION

"Clean code always looks like it was written by someone who cares."
- Robert C. Martin

The course begins with an introduction to the basic principles and history of operating systems, covering the evolution from simple batch systems to complex, multi-user, and multitasking systems. Students will learn about the architecture of operating systems, including the kernel, system calls, and user interfaces. Key concepts such as processes, threads, and concurrency are introduced early on to provide a foundation for understanding how operating systems handle multiple tasks simultaneously.

This course has 3 credits and is divided into 8 Units. In the process management module, students delve into the lifecycle of a process, including creation, scheduling, and termination. Topics such as context switching, process synchronization, and inter-process communication (IPC) are explored in detail. Practical lab sessions involve writing and analyzing programs that demonstrate these concepts, allowing students to observe firsthand how operating systems manage process execution and coordination.

Memory management is another critical area of focus. Students will learn about different memory management techniques, including paging, segmentation, and virtual memory. The course covers how operating systems allocate and deallocate memory, manage memory hierarchies, and ensure efficient use of available memory resources. Through practical assignments, students will implement and test memory management algorithms, gaining a deeper understanding of how operating systems optimize memory usage and maintain system stability.

Course Outcomes:
At the completion of the course, a student will be able to:

1. Recall the main components of an OS & describe the important computer system resources functions and the types of Operating Systems.
2. Explain the working of an OS as a resource manager, file system manager, process manager, memory manager and I/O manager and methods used to implement the different parts of OS and understand the factors in OS design.
3. Evaluate the requirement for process synchronization and coordination handled by operating system
4. Categorize memory organization and explain the function of each element of a memory hierarchy and analyze its allocation policies.
5. Conceptualize the components involved in designing a contemporary OS.

Acknowledgements:

The content we have utilized is solely educational in nature. The copyright proprietors of the materials reproduced in this book have been tracked down as much as possible. The editors apologize for any violation that may have happened, and they will be happy to rectify any such material in later versions of this book.

Unit 1: Introduction to Operating System

Learning Outcomes:

- Students will be able to understand the fundamental parts for an operating system.
- Students will be able to identify that an operating system virtualizes the CPU and memory.
- Students will be able to demonstrate the various scheduling along with swapping policies.
- Students will be able to acquire fundamental concurrent programming with C and assembly code.
- Students will be able to analyse how a rudimentary file system organizes data on a hard disc.

Structure:

1.1 Introduction of Operating System

An operating system (OS) acts as an intermediary between hardware architecture and users, overseeing all interactions between hardware components and software applications.Essentially, the operating system governs the entire operation of a computer system. It also controls tasks such as memory management, process management, and facilitates seamless communication between hardware and software components.

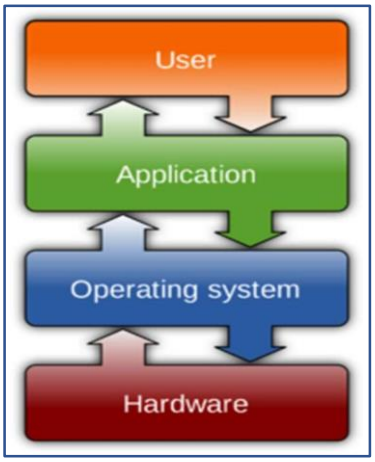


Fig. 1.1: Operating System

1.2 Operating system functions

1. Memory Management

Main memory serves as the core repository for active programs. Every program that runs must reside in main memory, which provides rapid access for the central processing unit (CPU). Once a program completes its task, the memory it occupied is released, becoming available for other programs to utilize. This capability enables multiple programs to run simultaneously, necessitating effective memory management to optimize system performance.

The Operating system:

- Memory allocator and releaser.
- Regularly monitors primary memory usage.
- Distributes RAM for multiprocessing.
- In multiprogramming, an operating system decides when and what amount of memory is allocated to each process.

2. Processor Management along with Scheduling

Every program that runs on a computer, whether in the foreground or background, utilizes the processor. Processor management involves overseeing how applications utilize the processing unit. The operating system monitors the status of both the processor and the processes running on it. It schedules operations to be executed by the processor, assigns resources to manage them, and deallocates those resources when processes are completed.

While numerous processes operate on the system in question, the OS decides when and how much each one uses the CPU. Thus, the name also relates to CPU Scheduling. Operating System:

- Processor resources are allocated and released to processes.
- Tracks CPU status.

The algorithms used for CPU planning are mentioned below.

- First come, first served (FCFS).
- Shortest Jobs First (SJF).
- Round-robin scheduling.
- Priority-based scheduling, etc.

3. Device Management

Drivers are essential components used by the operating system to manage and control device connections. These drivers enable processes within the operating system to interact with hardware devices efficiently. The operating system oversees the administration of these connections, ensuring that devices are properly utilized and that data can be transferred between software applications and hardware components seamlessly. Devices are allocated and de-allocated to various operations.

- Maintains records on the devices.
- Decides which method can utilize the equipment for how long.

4. File management

The operating system manages the allocation and deallocation of resources, specifying which operations can access documents and for how long. It also monitors data, settings, usage, and other conditions related to these groups of resources, collectively known as file systems. Files on a computer are stored in several directories. Operating System:

- Keeps track of file status as well as locations.
- Allocates and deallocates resources.
- Determines who receives resources.

5. Storage Management

Storage management maximizes device efficiency while securing data integrity. Key strategies include virtualization for resource abstraction, replication for data redundancy, and robust security measures. Compression reduces data size, while deduplication minimizes redundancy. Traffic analysis optimizes performance, automation streamlines operations, and provisioning allocates resources optimally. Memory management enhances data retrieval and system performance. These strategies collectively ensure efficient, secure, and reliable storage management. The operating system of a machine (also known as the OS) is in charge of saving and opening files. Storage management comprises creating folders and files, transferring and importing data from them, and moving file and folder contents from one area to another.

Storage management employed by the operating system:

- Improve the efficiency of data resource utilization.
- Optimizes the use of various storage mediums available.
- Helps prevent data loss and ensures compliance with retention laws.

1.3 Operating System Structure

Due to the intricate nature of operating systems, a structured approach is essential for tailoring them to specific requirements. Breaking down the operating system into manageable components is akin to dividing complex problems into smaller, more manageable subproblems. All elements of the operating system are encompassed within this structured framework. Operating system architecture is an approach for connecting and combining multiple operating system components within kernel. Operating systems run on a variety of buildings, which will be detailed below:

1. Simple structure:

It serves as the fundamental software architecture, yet its significance is minimal and is only suitable for small and limited systems. Because the communication pathways and degrees of capabilities in this framework are explicitly described, programmes can access I/O timelines, potentially resulting in unauthorized access to I/O operations.

The operating system known as MS-DOS has the following organisational structure:

- The operating system known as MS-DOS consists of four layers, each with its own set of characteristics.

- These layers contain device drivers for ROM BIOS and MS-DOS, as well as application and system programmes.
- Layering is beneficial to the operating system MS-DOS since each level may be established individually and communicate with one another as needed.

The following figure shows layering in a simple structure:

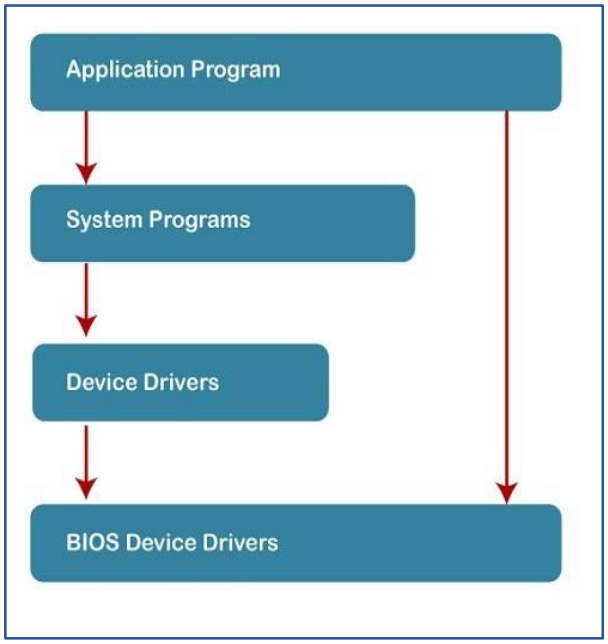


Fig. 1.2: Simplest operating system structure

Advantages of Simple Structures:

- Development is simplified due to the minimal number of interfaces and layers.
- Because there are fewer components separating the computer's hardware from its applications, performance is improved.

Limitations of simple structure:

- The operating system breaks as a whole if one user program fails.
- Due to the interdependence and communication between the layers, abstraction and data concealment are not feasible.

2. Monolithic structure

The single operating system manages all parts of an operating system's activities, such as handling files, handling memory, handling devices, and operational responsibility.

The kernel (OS) is a vital component of any computer operating system, providing core services to other system components. It serves as the primary interface between the operating system (OS) and the hardware. When the operating system (OS) is tightly integrated with specific hardware, such as a computer system with keyboard and mouse, the kernel grants direct access to all resources.

The diagram below depicts the monolithic structure:

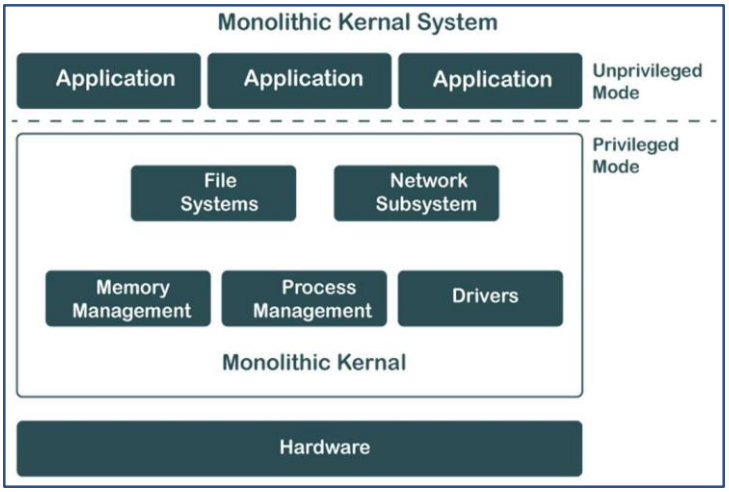


Fig. 1.3: Monolithic Structure

Advantages of monolithic structure:

- It is easy to design and implement since layering is superfluous and the kernel is in complete control of all operations.
- The monolithic kernel operates much faster than other systems since operations such as managing memory, managing files, scheduling processes, and so on can be performed in the same area of memory.

Disadvantages of a monolithic structure:

- The monolithic kernel's functions are linked together in address space and interact with one another, so if one fails, the entire system fails.
- It is not adaptive. Therefore, creating a new product is challenging.

1. Layered structure

This style of design separates the operating system's code into layers or levels. Layer 0 (the bottom depth) includes the hardware, while layer 1 (the top depth) houses the user interface (layer N). These levels are organized in a hierarchical structure, with higher-level layers utilizing lower-level skills and abilities.

This method separates the features of each individual layer while also allowing for abstraction. Because layers of software are centralised, debugging is simplified, thus all the lowest levels are evaluated before the highest level is examined. As an outcome, only the present layer needs to be checked because every one of the lower layers has already been reviewed.

The graphic below depicts how the OS is organised into layers:

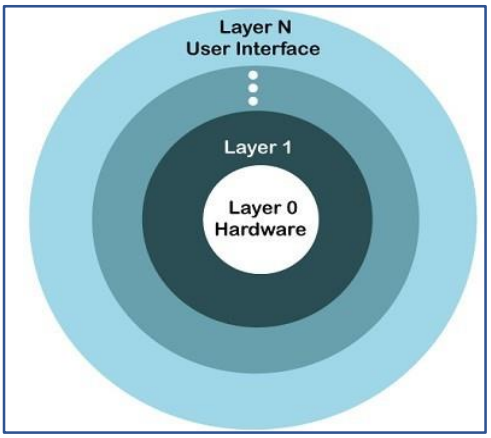


Fig. 1.4: Layered Structure

Advantages of Layered Structures:

- Work requirements are segregated as each layer has its own functionality and level of abstraction.
- Debugging is simplified by focusing on the lowest layers before moving on to the top layers.

Limitations of Layered Structure

- Layered architectures' performance suffers as a result.
- The layers must be carefully designed because higher layers only leverage the capabilities of lower tiers.

2. Micro-Kernel Structure

The operating system (OS) is structured using a micro-kernel architecture, which eliminates unnecessary components from the kernel. Optional kernel components are instead implemented through system and user-level applications. This approach gives rise to what are known as micro-kernels.

Each micro-kernel is developed and maintained independently from others. This design enhances the system's reliability and security. The remainder of the operating system is unaffected and keeps working normally even if one micro-kernel fails.

The graphic below displays the Micro-Kernel Operating System's Structure is

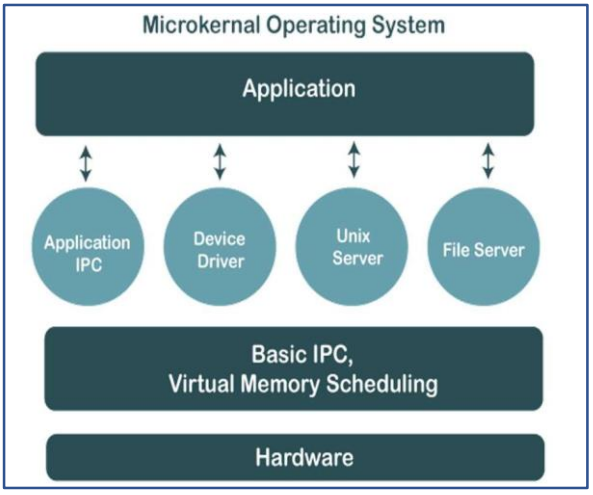


Fig. 1.5: Micro-Kernel Structure

Advantages of the Micro-Kernel Structure:

Facilitates portability of the operating system across different platforms.

Each micro-kernel operates in isolation, enhancing reliability and security.

Disadvantages of the Micro-Kernel Structure:

Increased inter-module communication can reduce system performance.

The construction of such systems is complex.

1.4 Types of Operating Systems:

1. Batch Operating System.

The hardware and this kind of computer operating system do not speak to each other directly. Rather, an operator collects jobs with comparable requirements and divides them into distinct batches. The operator is responsible for grouping jobs with comparable specifications.

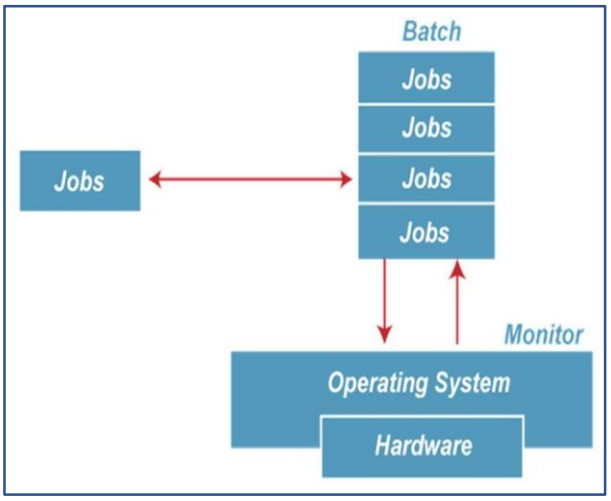


Fig. 1.6: Batch Operating System

Benefits of Batch Operating System:

- Multiple users may collaborate on batch systems.
- The batch system has relatively little idle time.

Limitations of Batch Operating Systems

- Familiarity with batch systems is important for computer operators.
- Debugging batch systems can be challenging.

Multiprogramming Operating System

Multiprogramming systems for operating systems may be simply defined as having multiple programmes in main memory, each of which can be executed. This is mostly used to improve resource utilisation.

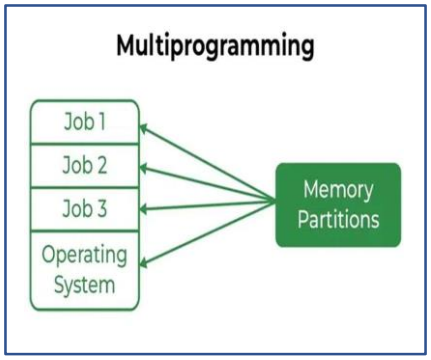


Fig. 1.7: Multiprogramming Operating System

Benefits of Multi-Programming Operating Systems

- Multi-programming boosts the system's throughput.
- It helps to shorten the response time.

Limitations of Multi-Programming Operating Systems

- There is no facility for users to interact with system resources.

2. Time-sharing operating systems:

Each assignment is assigned a specific amount of time to accomplish so that all tasks operate smoothly. Because they use the same system, each user is allocated CPU time. These systems are also known as performing multiple tasks Systems. The job may have been started by a single user or several people. Quantum denotes the quantity of time provided to each activity. Following this period, the operating system continues on to the subsequent task.

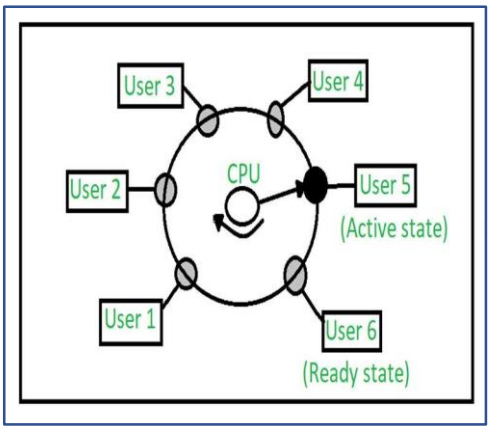


Fig. 1.8: Time-sharing operating systems

Benefits of Time-Sharing OS

- Each task has an equal opportunity.
- There are fewer risks of software duplication.
- The CPU's rest period can be minimised.

Limitations of Time-sharing OS

- Reliability issue.
- One must ensure the security and integrity of user programmes and data.

3. Desktop systems

The desktop operating system (OS) is the surroundings in which a user maintains their personal computer. It facilitates the control of system software as well as hardware assets. Examples include Mac OS, Windows, and a variety of Linux variants. It has fundamental capabilities including job scheduling, the printing process, I/O, peripheral management, and memory allocation.

On a desktop computer, the Operating System (OS) acts as an intermediary between applications and the hardware of the system. It facilitates the interaction between various programs and the underlying hardware components. While some operating systems need to be installed separately, others come pre-installed on new computers. Microsoft Windows, Mac OS X, and Linux are among the most widely used desktop operating systems. These modern operating systems typically feature a graphical user interface (GUI) to enhance user interaction and ease of use.

4. Multiprocessing Operating System

Multi-processing operating systems use more than one CPU to execute resources. It improves the system's throughput.

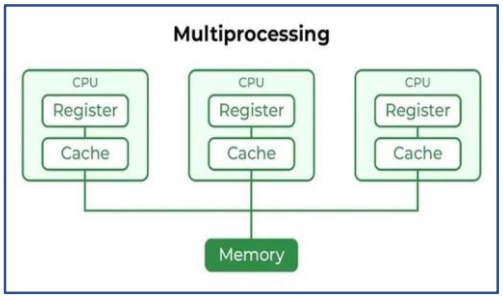


Fig. 1.9 Multiprocessing Operating System

Benefits of Multi-Processing Operating Systems

- It boosts the system's throughput.
- It has many CPUs, so if one fails, we can switch to another.

Disadvantages of a Multiprocessing Operating System

- Because of the various CPUs, it can become more complex and difficult to grasp.

5. Distributed Operating System

These operating systems are a recent innovation in computer technology that are rapidly gaining popularity around the world. Several autonomous networked computers communicate with one another over a shared network of communication. Independent systems have their own memory units and CPUs. These are known as loosely connected or dispersed systems. These systems' CPUs vary in dimension as well as function.

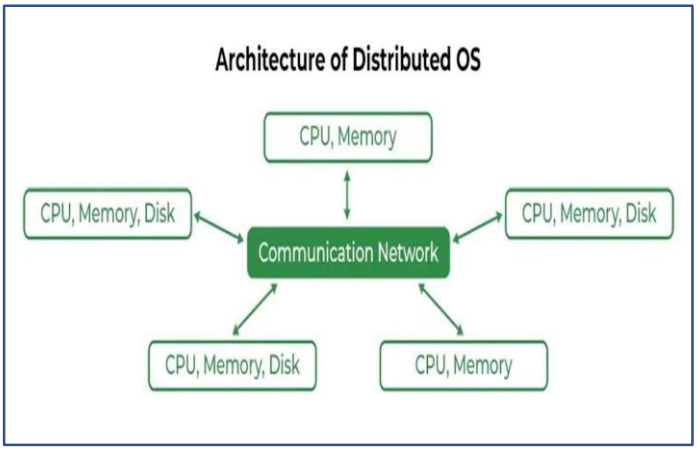


Fig. 1.10 Distributed Operating System

Benefits of Distributed Operating Systems:

- Because all systems were independent of one another, their failure would have no effect on the other's network connectivity.
- E-mail speeds up data communication.

Limitations of distributed operating systems

- The failure of the primary network will halt all communication.
- The language used to construct distributed systems is currently not clearly defined.

6. Clustered systems

Cluster systems and parallel systems are similar in that both require many CPUs. The major distinction is the fact that clustered systems consist of a minimum of two independent systems that are linked together. They have individual systems for computing and shared media for storage, and all systems collaborate to execute all tasks. To communicate with one another, all cluster nodes employ two different methods: message-passing interfaces (MPI) as well as parallel virtual machines (PVM).

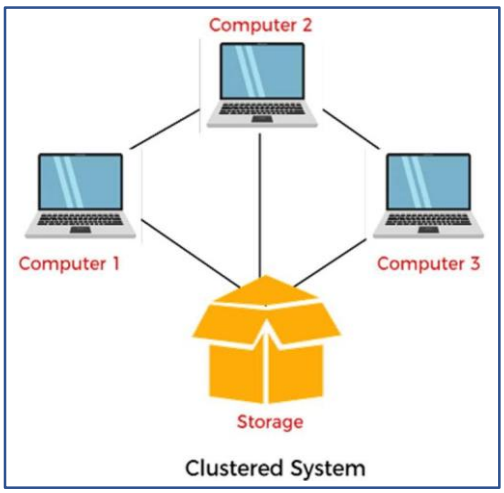


Fig. 1.11 Clustered System

7. Real-Time Operating System:

These operating systems are designed for real-time applications where the processing and response to inputs must occur very quickly. The time taken to process and respond to inputs is crucial and is referred to as response time. Real-time systems are utilized in applications that demand precise timing, such as weapons systems, air traffic control systems, robotics, and other critical systems where timing accuracy is essential.

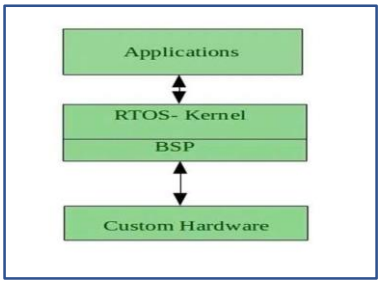


Fig. 1.12 Real-Time Operating System

Advantages of RTOS

- Error-free: These systems do not contain errors.
- Memory Allocation: Memory allocation works best in these systems.

The disadvantages of RTOS

- Resource-intensive: Real-time operating systems often require significant system resources, which can be costly and may not be readily available.
- Complex algorithms: Developing real-time systems involves designing and implementing complex algorithms that can be challenging for software designers to develop effectively.

8. Handheld systems

Handheld operating systems are accessible for all handheld devices, including smartphones and tablets. It is sometimes referred to as Personal Digital Assistance. Today's most popular handheld devices are Android and iOS.

Features of the Handheld Operating System:

- Its purpose is to deliver real-time operations.
- There is direct use of interruptions.
- The flexibility of input/output devices.
- Configurability.

Advantages of the Handheld Operating System:

- Lower cost, weight, and size.
- Reduced heat generation.
- More dependable.

Limitations of handheld operating systems:

- Less speed and smaller size.
- Input/Output System

9. Open-source operating systems

The concept of "open source" refers to applications or software for computers where the owners of the copyright allow users and third parties to view and alter the original source code. The source code of an operating system that is open-source is publicly available and changeable. Closed operating systems like iOS from Apple, Microsoft's Windows, and Apple's MacOS are proprietary and tightly controlled by their respective companies. Conversely, open-source software is distributed under a license that permits unrestricted use, modification, and distribution.

Open-source software generally requires fewer resources compared to commercial alternatives because it doesn't include licensing, marketing, branding, and other proprietary code that can bloat the system. This often results in more efficient and streamlined software solutions.

An open-source operating system allows users to freely access and utilize both publicly distributed and commercially available code. These operating systems provide open access to

their program source code, enabling users to modify and customize applications according to their specific requirements. Examples of open-source operating systems include Linux, OpenSolaris, FreeRTOS, OpenBSD, FreeBSD, Minix, among others. Users can leverage these systems to innovate, adapt, and develop new applications tailored to their needs, fostering a collaborative and transparent approach to software development.

1.5 Operating System Structures

System Components

An “operating system” is a vast and intricate system that is constructed by dividing it into smaller, well-defined components. Each component of the operating system should have clearly defined inputs, outputs, and functions. These components play a crucial role in facilitating the seamless interaction of various computer system elements. By breaking down the operating system into manageable components, it becomes easier to develop, maintain, and enhance its functionality, ensuring efficient operation of the entire computer system.

1. Process Management

Process management is a critical tool within an operating system, responsible for overseeing multiple processes that run concurrently on the machine. Every software application within an operating system consists of one or more processes.

When you initiate an online search using a web browser like Chrome, a process is launched for the browser software. Process management plays a crucial role in ensuring these processes operate smoothly. It allocates and monitors the amount of RAM (Random Access Memory) allocated to each process and terminates them when necessary to optimize system performance.

A process must be completed sequentially, which necessitates that a minimum of one instruction be executed on its behalf.

Functions of management of processes

- Process activation and deletion
- Suspended and resumed
- Synchronization procedure
- Communication Process

2. File Management

A file is an organized set of connected data that has been defined and created by the person who created it. Files are frequently used to store data and programs, both in source and object formats. Depending on their intended purpose, data files may contain alphanumeric, numeric, or alphabetic data.

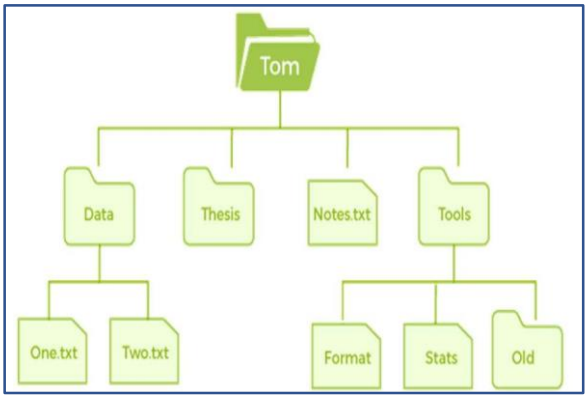


Fig. 1.13: File Management

The function of managing files:

The operating system performs several critical file management activities:

- Create and remove files and directories.
- To manage files and directories.
- Mapping files to supplementary storage.
- Back up your files to a dependable storage medium.

3. Network Management

Network management is the profession of overseeing and managing computer networks. It comprises performance management, network supplies, issue inquiry, and quality service monitoring.

A distributed system consists of devices or computers that do not share memory or clocks. In this form of system, each processor has internal memory and communicates with one another via various communication channels, such as optical fibers or telephone lines.

Functions of management of networks

- Distributed systems offer diverse computer resources, ranging in size and functionality. They could include small computers, microprocessors, and different general-purpose computer systems.
- A distributed system allows users to access the network's different resources.

4. Main Memory Management

Main memory consists of a large array of memory bytes that can be addressed individually. Memory management involves reading from and writing to specific memory addresses. Programs must be translated into absolute locations and loaded into memory before they can be executed. Various factors influence the choice of memory management approach.

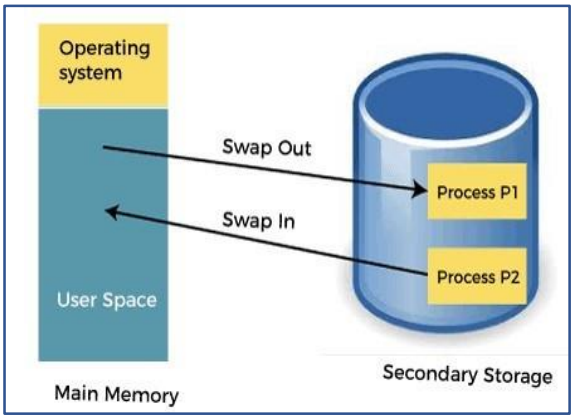


Fig. 1.14 Main Memory Management

5. Secondary Storage Management

The fundamental function of a computing system is to execute programs. These applications allow users to access and manipulate information stored in main memory during their execution. This computer's RAM is insufficient to permanently store every piece of information and programmes. The computer system includes extra space for storing the main memory.

Uses of Secondary Storage Management

The following are some primary responsibilities:

- Storage allocation
- Free space management
- Disc Scheduling

6. I/O Device Management

One essential function of an operating system is to abstract the differences between various hardware components, providing a uniform interface to the user.

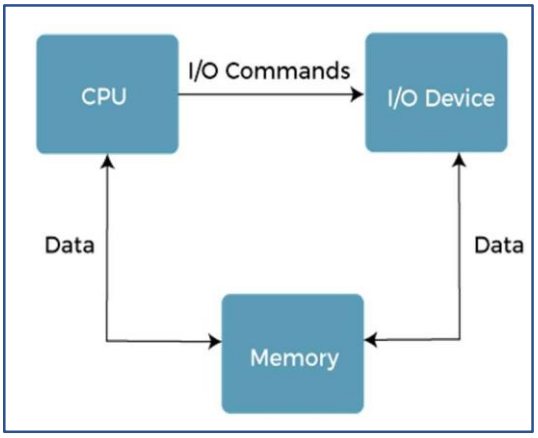


Fig. 1.15 I/O Device Management

Functions of I/O Monitoring

The I/O control system includes the following functions:

- It provides a buffer caching system.
- It gives general device driver codes.
- It provides solutions for certain hardware devices.
- I/O allows you to learn about the unique characteristics of a certain device.

1.6 Operating system services

1. **Programme Execution:** It's the Operating System that determines how a programme will be performed. It loads the programme into memory, which is then run. The CPU Scheduling Algorithm determines the sequence in which they are executed. A few examples include FCFS and SJF. When the programme is being executed, the Operating System handles deadlock, which occurs when two processes attempt to execute at the same moment.

2. **Input/Output Operations:** The operating system organises input-output operations and facilitates interaction between the user as well as device drivers. Device drivers are pieces of software that are related with hardware that is managed by the operating system to ensure

appropriate device synchronisation. It also allows a programme to access input-output devices as needed.

3. **Communication between processes:** The operating system oversees how processes communicate with each other, including data transfer. For processes on different computers connected via a network, the OS manages their communication.

4. **File Management:** The operating system helps manage files by granting access permissions such as read-only or read-write. It provides a platform for users to create and delete files.

5. **Memory Management:** The "operating system" initially checks if a program meets all memory requirements. If it does, the OS calculates the necessary storage capacity and loads the program into memory at a specific location. This ensures the program does not use excessive RAM.

6. **Networking:** This service enables communication between networked devices, allowing for internet access, data packet transfer, and network connection management.

7. **Error Handling:** The operating system additionally handles errors that arise in the CPU, I/O devices, and so on. It also guarantees that errors are not repeated frequently and corrects them. It also prevents the process from reaching an impasse.

1.7 System Calls

Through the use of a system call, an operating system program can ask the kernel for a service. It acts as a channel of communication between software and the operating system.

In essence, a system call is a request made by software to the kernel for a specific service. The Application Programming Interface (API) connects operating system functions to user programmers.

How System Calls Works

Applications execute within a designated area of memory referred to as user space. To interact with the operating system's kernel, located in kernel space, applications make system calls. When an application initiates a system call, it requests permission from the kernel. This process involves halting the current program execution and transferring control to the kernel.

A simple system call, like retrieving the system's time and date, might take only a few milliseconds to complete. Conversely, a more intricate process, such establishing a connection with a network device, may need several seconds. Most operating systems generate a distinct kernel thread for every system call in order to reduce bottlenecks.

Modern operating systems are multi-threaded, allowing them to manage multiple system calls simultaneously.

1.8 System programs

System programming is the act of developing software for systems using system languages for programming. According to the Computer Hierarchy, Hardware comes last. Then comes the operating system, followed by system programmes, and ultimately application programmes. System Programmes provide for easy programme development and execution. Some system programmes are simple user interfaces, while others are more complicated. It is traditionally located between the user's interface and calls to the system.

Types of System programmes

System programmes are classified into the following types:

Utility programme.

It administers, maintains, and regulates a variety of computer resources. Utility programmes are comparatively technical in nature and are intended for users with extensive technical understanding.

A few examples of utility programmes include antivirus software, backup software, and disc tools.

Device drivers

Device drivers operate as translators between the operating system and the connected devices, such as printers, scanners, and storage devices.\

Directory reporting tools

These tools are essential in an operating system to have certain applications to help users navigate the computer system.

Example: dir, ls, Windows Explorer, etc.

- **Knowledge Check 1**

Fill in the Blanks

- System programming is the act of developing _____ for systems using system languages for programming. (**software** / hardware)
- System calls is a way of communicating with an operating system through _____. (Windows / **programmes**)
- _____ administration refers to the process of controlling and overseeing computer networks. (**Network** / system)
- A _____ is a collection of connected information defined by the creator. (kernel /file)

- **Outcomes-Based Activity 1**

In this activity, students design, properly execute, and describe solutions to major computational challenges. Analyse and compare alternatives to computing.

1.9 Self- Assessment Questions:

1. What are the fundamental concepts covered in the introductory section of operating systems?
2. Discuss the primary functions performed by an operating system.
3. Explain the structure of an operating system and its key components.
4. What are the different types of operating systems? Provide examples for each type.

Unit 2: CPU Scheduling

Learning Outcomes:

- Students will be able to understand the CPU Scheduling.
- Students will be able to identify process concepts and operations.
- Students will be able to demonstrate the various scheduling criteria.
- Students will be able to analyse the multiple processor scheduling.

Structure:

2.1 CPU Scheduling

When a process waits for an input or output operation to be completed in uniprogramming systems such as MS DOS, the CPU remains idle. This is an expense as it wastes time and creates the issue of famine. However, in multi-programming systems, the central processing unit (CPU) does not remain idle throughout the process's wait period and instead begins executing other processes. The operating system must determine which process will be assigned to the CPU. In multi-programming systems as a whole, the operating system arranges operations on the CPU to maximize its utilisation, a method known as CPU scheduling. The Operating System schedules processes using a variety of algorithms.

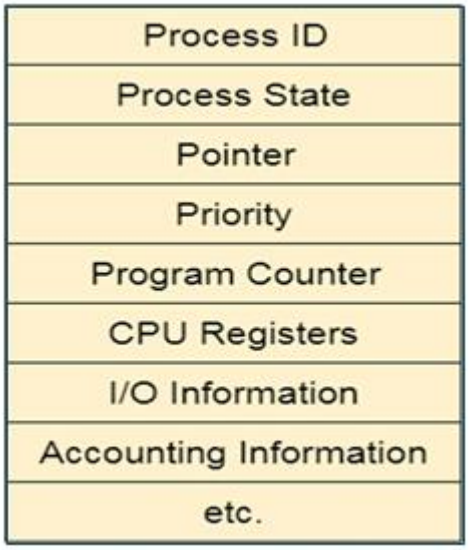


Fig. 2.1 CPU Scheduling

2.2 Process concepts

A process refers to a computer program executing on one or multiple threads, encompassing both the program's code and its ongoing activity. The operating system (OS) states that the process can have many execution threads that carry out instructions simultaneously.

The way is process memory utilised to ensure efficient operation ?

To ensure efficient operation, the procedure's memory is separated into four portions.

- The text segment comprises embedded program code retrieved from fixed memory during program initialization.
- The data segment includes both global variables, which retain their values throughout the program's execution, and dynamic variables, which are allocated and initialized before the main program begins.

- Heaps are used for flexible or dynamic allocation of memory and are handled by calls like new, delete, the malloc function, free, and so on.
- The stack of files is used to store local variables. When it is announced, space is reserved on the stack for local variables.

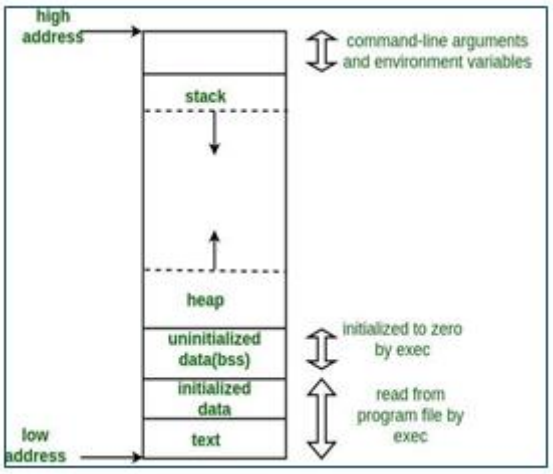


Fig. 2.2 Process of CPU Scheduling

2.3 Process operations

Procedure scheduling refers to the process by which the process manager selects a new process to execute on the CPU, replacing the current active process, using a defined method or approach.

Scheduling processes are an essential component of multiprogramming applications. Numerous processes can be loaded concurrently into accessible memory by these operating systems, with the loaded sharing CPU process consuming repetition time.

There are three kinds of process schedulers:

- Long-term or Job Scheduler
- Short-term or CPU Scheduler
- Medium-term Scheduler

2.4 Inter-process communication

"Inter-process communication is utilised for exchanging helpful data between numerous threads in multiple processes (or programmes)."

To comprehend inter-process communication, consider the graphic below, which highlights its importance:

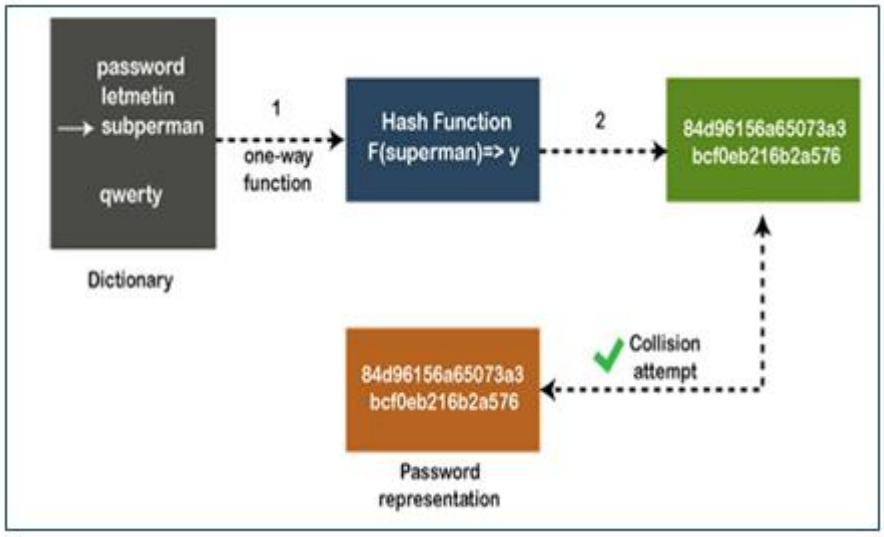


Fig. 2.3 Inter-process communication

Why do we require inter-process communication?

There are several reasons to employ inter process communication to share data. Here are a few of the most essential causes, as listed below:

1. It serves to speed up modularity.
2. Computational
3. Privilege separation.
4. Convenience
5. Helps operating systems interact with one another and synchronise their actions.

2.5 CPU scheduling criteria CPU utilisation

The main objective of every CPU scheduling strategy is to maximize CPU utilization. In practical terms, CPU utilization typically ranges from 40 to 90 percent in real-time operating systems, depending on the system's demands, although theoretically it can vary between 0 and 100 percent.

Throughput

The amount of processes executed and finished per unit of time is used to assess the CPU's workload. This is known as throughput. The speed of processing may vary according to the length or length of the processes.

Turnaround Time

The duration required to finish a particular process holds significant importance. Turnaround time refers to the interval between when a process is submitted and when it is finalized. It encompasses the total time spent waiting to access memory, waiting in the ready queue, processing in the CPU, and awaiting input/output operations.

"Turnaround Time" = "Completion Times" – "Arrival Time"

Waiting Time

Once applied, a scheduling method does not directly affect the actual time taken to complete an activity. Its main impact is on the waiting time of a process, which indicates the duration spent in the ready queue waiting for execution.

"Waiting time" = "Turnaround time" – "Burst time"

Response Time

In an interactive system, turnaround time may not hold the utmost importance. A process can yield initial outcomes relatively quickly and continue generating new results while simultaneously displaying earlier results to the user.

"Response Time" = "CPU Allocation Time (when the CPU was initially allocated)" – "Arrival Time"

Completion Time

The termination time of a process marks the moment when it ceases execution, signifying the completion of its burst time and full execution.

Priority

The scheduling mechanism ought to give the more crucial processes higher priority when the operating system designates them as such.

Predictability

Under a similar system load, a given process should ideally be completed in approximately the same amount of time as another process.

2.6 CPU scheduling algorithms

A Process Scheduler arranges various tasks to be allocated to the CPU using specific scheduling techniques.

1. First come, first serve (FCFS)

1. Tasks are executed in the order they arrive, following a first-come, first-served approach.
2. It operates as both a non-preemptive and preemptive scheduling algorithm.
3. Easy to comprehend and implement due to its simplicity.
4. Implementation relies on a FIFO (First In, First Out) queue.
5. Demonstrates poor performance due to extended average waiting times.

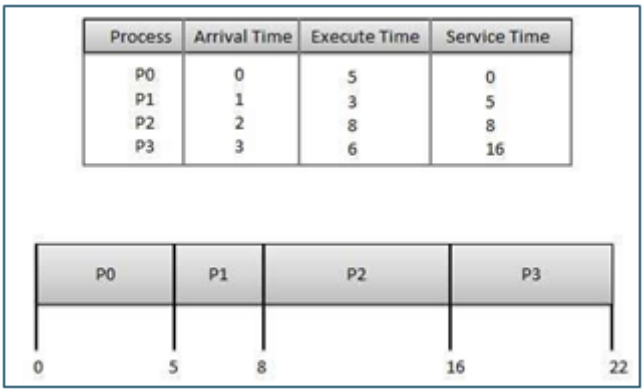


Fig. 2.4 First come, first serve (FCFS)

The wait times for each process are listed below:

"Process"	"Wait Time" = "Service Time - Arrival Time"
"P0"	"0 - 0 = 0"
"P1"	"5 - 1 = 4"
"P2"	"8 - 2 = 6"
"P3"	"16 - 3 = 13"

Average Wait Time: (0+4+6+13) divided by 4 equals 5.75.

2. Shortest Job Next (SJN).

- This is known as the "Shortest Job First" (SJF) algorithm.
- It can be used as both a non-preemptive and preemptive scheduling method.
- SJF is highly effective in minimizing waiting time.
- It is straightforward to implement in batch processing systems where CPU requirements are known in advance.

- Implementing it in interactive environments is impractical due to the unpredictability of required CPU time.
- The algorithm relies on the processor knowing the duration of each process beforehand.

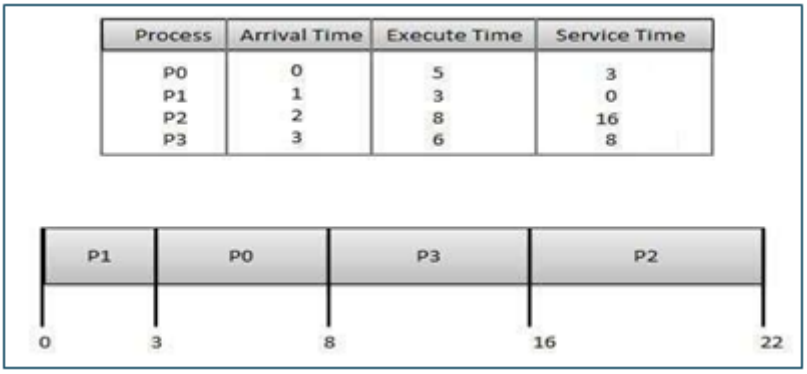


Fig. 2.5 Shortest Job Next (SJN)

Waiting times for each process are as follows:

"Process"	"Wait Time = Service Time - Arrival Time"
"P0"	"0 - 0 = 0"
"P1"	"5 - 1 = 4"
"P2"	"14 – 2 = 12"
"P3"	"8 – 3 = 5"

Average Wait Time: (0+4+12+5)/4 = 21/4 equals 5.25

3. Priority-Based Scheduling

- Priority scheduling is a non-preemptive scheduling method typically utilized in batch systems.
- Processes are assigned priorities, with those having higher priorities being executed before others.
- If multiple processes have the same priority, they are executed based on a first-come, first-served principle.
- Priorities can be determined by various factors, including memory usage, time requirements, or other resource demands.

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0

Fig. 2.6 Priority-Based Scheduling

Waiting times for each process are as follows:

"Process"	"Wait Time = Service Time - Arrival Time"
"P0"	"0 - 0 = 0"
"P1"	"11 – 1 = 10"
"P2"	"14 – 2 = 12"
"P3"	"5 – 3 = 2"

Average Wait Time: (0+10+12+2)/4 = 24/4 equals 6

4. Shortest Remaining Time

- The preemptive counterpart of the Shortest Job Next (SJN) algorithm is called Shortest Remaining Time (SRT).
- In this approach, the CPU is assigned to the process that has the least amount of time left until completion. However, this process can be interrupted if a new job arrives with a shorter remaining time.
- Implementing SRT in interactive environments is challenging due to the unpredictability of CPU time requirements.
- SRT is typically used in batch processing systems where prioritizing shorter processes is advantageous.

5. Round Robin Scheduling

- Round Robin is a preemptive scheduling algorithm for processes.
- Every process is allocated a specific time slice for execution.
- When a process finishes its quantum, it is preempted to allow another process to execute.
- Context switching is used to save and restore the state of preempted processes.

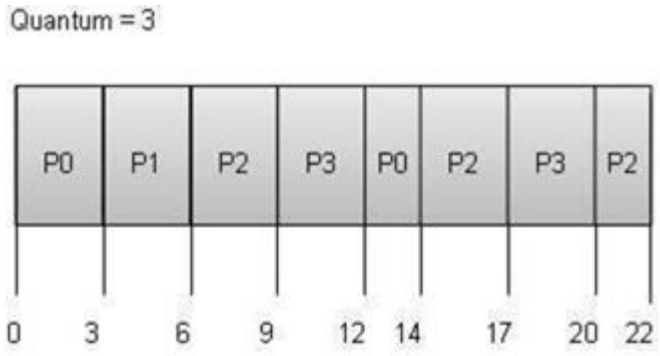


Fig. 2.7 Round Robin Scheduling

Waiting times for process are as follows:

"Process"	"Wait Time = Service Time - Arrival Time"
"P0"	"(0 – 0)+(12 – 3) = 9"
"P1"	"(3 – 1) = 2"
"P2"	"(6 – 2) + (14 – 9) + (20 - 17) = 12"
"P3"	"(9 – 3) + (17 – 12) = 11"

"Average Wait Time": (9+2+12+11)/4 = 8.5

6. Multiple-Level Queue Scheduling

- Multiple-level queues are not an independent scheduling algorithm but rather utilize existing algorithms to manage and schedule jobs with similar attributes.
- Processes are grouped into separate queues based on shared characteristics.
- Each queue can employ its own scheduling algorithm.
- Priority levels are assigned to each queue for efficient management of processes.

2.7 Comparative study of scheduling algorithms

Algorithm	Allocation is	Complexity	Average waiting time (AWT	Preemption	Starvation	Performance
FCFS	The CPU is assigned based on the process arrival times.	Not complex.	Large.	No	No	Slow performance .
SJF	Based on the	More sophisticate	Smaller than FCFS	No	Yes	Minimum Average
	shortest CPU burst time (BT).	d than FCFS.				Waiting Time
LJFS	Based on the maximum CPU time for bursts (BT)	More sophisticate d than FCFS	Depending on certain parameter s, such as time of arrival, process size, and so on	No	Yes	Significant turnaround time
LRTF SRTF	Similar to LJFS, CPU allocation depends on the greatest CPU burst time (BT). But it's preemptiv e.	More sophisticated than FCFS	Depending on certain parameters, such as time of arrival, process size, and so on.	Yes	Yes	Longer jobs are preferred.

RR	Based on the order In the procedure, arrives with a fixed time	The complexity depend on the TQ size.	Large when compared with SJF and Prioritise scheduling.	No	No	Each process has assigned a pretty defined time
	quantum (TQ).					

2.8 Multiple processor scheduling

Multiple processor scheduling, or multiprocessor scheduling, involves designing a system's scheduling mechanism to manage multiple processors. This method allows multiple CPUs to distribute the workload, enabling simultaneous execution of various tasks. Generally, scheduling for multiple processors is more complex than scheduling for a single processor.

In multiprocessor scheduling, several identical processors are available, and any task can be executed at any time.Approaches to Multiprocessor Scheduling

The operating system employs two approaches to multiple processor scheduling: symmetric multiprocessing & asymmetric multiprocessing.

1. Symmetric Multiprocessing:

In such systems, each processor is typically capable of self-scheduling. Processes can either share a single ready queue or each processor may maintain its own queue of ready processes. The scheduling process involves each processor's scheduler periodically examining its queue of ready processes and selecting one to execute based on predefined scheduling policies or algorithms. This distributed approach helps in efficiently utilizing multiple processors to handle concurrent tasks.

2. Asymmetric Multiprocessing:

It is utilised when a single processor, known as the Master Server, handles all scheduling and I/O processing. The remaining processors simply execute the user code. This is simple and lessens the need to share data; the entire situation is called Asymmetric Multiprocessing.

- **Knowledge Check 2**

  State True or False

  1. Multiple processor scheduling, also known as multiprocessor scheduling. (**True**)
  2. The SDN algorithm's preemptive variant is known as shortest remaining time. (**False**)
  3. Inter-process communication is utilised for exchanging helpful data between numerous threads in multiple processes. (**True**)
  4. Round Robin is a proactive process scheduling algorithm. (**True**)\

- **Outcomes-Based Activity 2**

In this activity, students are able to run several threads at the same time or run a specific thread faster. Running additional threads concurrently is simple; the threads are simply distributed among the processors.

2.9 Summary

- An operating system (OS) is software that mediates between computer hardware and users, enabling and overseeing interactions between software applications and a computer system's hardware components.
- The primary memory is a rapid storage location that the CPU can access directly.
- The operating system handles resource allocation as well as de-allocation.
- The operating system is built with a micro-kernel framework, which removes any unneeded components from the kernel.
- Multiprogramming systems for operating systems may be simply defined as having multiple programmes in main memory, each of which can be executed.
- Each assignment is given a certain amount of time to complete so that all tasks run properly. Because they share the same machine, each user receives CPU time. These systems are additionally known as Multitasking Systems.
- The desktop operating system (OS) is the surroundings in which a user maintains their personal computer.
- A file is an assemblage of related data that has been annotated by the author. Programs and data are frequently represented by it, both in source and object form.
- Network administration refers to the process of controlling and overseeing computer networks.

2.10 Self-Assessment Questions

1. What is the function of the operating system?
2. What are the types of Operating systems?
3. What is Micro-Kernel Structure?
4. What is Multiple processor scheduling?
5. Explain Round Robin Scheduling.

2.11 References / Reference Reading

- Stallings (2005). Operating Systems, Internal Structures, and Design Principles. Pearson: Prentice Hall. Page 6.
- "VII. Special-Purpose Systems - Operating System Concepts, Seventh Edition [Book]" . at www.oreilly.com. Archived from the original version on June 13, 2021. Retrieved February 8, 2021.
- Dhotre, I.A. (2009). Operating System Technical Publications. Page 1.

Unit 3: Memory Management

Learning Outcomes:

- Students will be able to comprehend swapping, paging, and segmentation, discerning theirdistinct advantages and limitations.
- Students will be able to grasp the importance of virtual memory and apply demand pagingprinciples.
- Students will be able to evaluate and utilize page replacement algorithms effectively.
- Students will be able to define file access methods, directory structures, and file-system mounting, recognizing the significance of file sharing and protection mechanisms.
- Students will be able to apply disk scheduling criteria and analyze algorithms like FCFS, SSTF,SCAN, and C-SCAN for enhanced disk performance.

Structure

3.1 Memory Management

Main memory, or the internal memory of the computer, is included in computer memory, whichis referred to as a collection of data expressed in binary format. It is distinguished from external mass storage devices, such as disk drives, by the term "main". The computer can alter data in the main memory, sometimes known as RAM. As a result, each program that is run and each file that is accessed needs to be copied into the main memory from a storage device.

To be executed, all programs loaded into the main memory. Programs can load all of their contents into memory at once, or only some sections or functions may load when the program calls upon them. Performance is improved by the Dynamic Loading procedure. Furthermore, instead of loading every program that depends on another when one program depends on another, the main executing program is as needed. This mechanism is termed Dynamic Linking.

This process facilitates understanding how data is stored in the computer system.

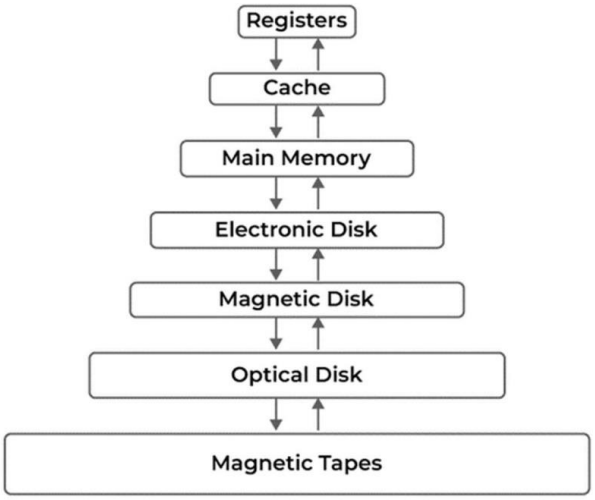


Figure 3.1: Main Memory

Memory Management in an Operating System

Coordinating and managing a computer's memory allocation is known as memory management in an operating system. In order to maximize system performance, it allows memory blocks to variousprograms that are currently running while monitoring the allocated or free status of each memory address.

This process determines when and how much memory each process receives, ensuring efficient memory utilization. It also protects processes from interfering with each other's operations and allocates space to different application routines.

The need for Memory Management in an OS includes:

- Efficient Memory Utilization: Placing programs in memory to maximize its usage.
- Process Isolation: Preventing processes from interfering with each other's operations.
- Space Allocation: Allocating space to different application routines.
- Fair Memory Allocation: Determining how much memory each process requires andallocating it accordingly.
- Memory Status Tracking: Monitoring the status of each memory location (allocated or free)and updating it as needed.

Logical and Physical Address Space

1. Logical Address Space: The address that the CPU generates, sometimes referred to as a "Virtual Address" or "Logical Address," establishes the boundaries of a process and is dynamically adjustable.
2. Physical Address Space: A "Physical Address" or "Real address" is the address that is entered into the memory address register and is accessible by the memory unit. The set of physical addresses that coincide with logical addresses is known as the physical address space.
3. The Memory Management Unit (MMU) is a piece of hardware that computes physical addresses. It manages the runtime translation of virtual to physical locations. Physical addresses don't change.

Static and Dynamic Loading

1. Static Loading: Entire programs are loaded into fixed addresses, requiring more memory space.
2. Dynamic Loading: In dynamic loading, programs and data are loaded into physical memory as needed for execution. The process size is limited to physical memory size, ensuring optimal memory utilization. Routines are not loaded until called, and unused routines are never loaded. This approach is beneficial for efficiently handling large amounts of code.

Static and Dynamic Linking

1. Static Linking: During static linking, the linker removes runtime dependencies by combining all required software modules into a single executable program. Certain operating systems regard system language libraries like any other object module and only allow static linking.

2. Dynamic Linking: A "Stub" is included for each library routine reference in dynamic linking, much like in dynamic loading. A stub is a brief code segment that, when run, determines whether required routine is present in memory. If not, the routine is loaded into memory by program.

Methods Involved in Memory Management

Methods involved in Memory Management include various techniques that enable the Operating System to intelligently manage memory allocation.

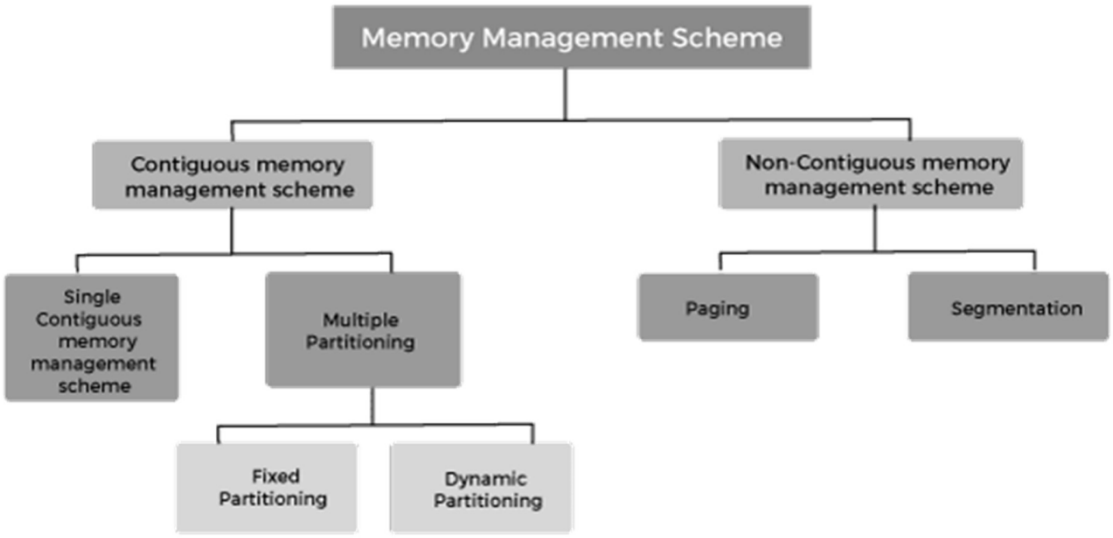


Figure 3.2: Classification of Memory Management Schemes

3.2 Swapping

A process must be in memory in order to be executed. A process can be temporarily moved from main memory to secondary memory with the help of swapping, which is quicker than transferring from secondary memory to main memory. This makes it possible for multiple processes to operate concurrently and fill memory at once. The transfer time, which is directly

correlated with the quantity of memory swapped, is the main feature of swapping.

Swapping, also known as roll-out or roll-in, occurs when the memory manager replaces a lower-priority process with a higher-priority one. The higher-priority process is loaded and executed, while the lower-priority process is temporarily moved out of memory. Once the higher-priority process is completed, the lower-priority process is reloaded into memory and resumes execution.

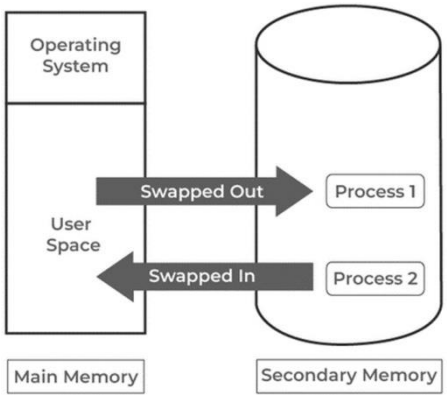


Figure 3.3: Swapping

Memory Management with Monoprogramming (Without Swapping):

In this memory management approach, the memory is divided into two sections:

1. Operating system section

2. User program section

A fence register is utilized to demarcate the boundary between the operating system and user program sections. The operating system keeps track of the first and last available locations for userprogram allocation.

Advantages: Simple management approach Disadvantages: Lack of support for multiprogrammingMemory wastage

Multiprogramming with Fixed Partitions (Without Swapping):

This memory management scheme introduces a fixed number of partitions to support multiprogramming, employing contiguous allocation.

Key Features:

- Memory partitioned into fixed-size partitions
- Each partition represents a block of contiguous memory

- The operating system maintains a partition table to track the partition status

Contiguous Memory Allocation

Assigning a single contiguous block of memory to every process is known as contiguous memory allocation. One process is stored in each of the fixed-size divisions that make up the memory. The system chooses the best hole from the collection of holes to allocate a process to. Free memory blocks are called holes.

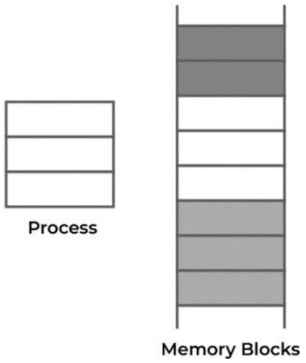


Figure 3.4: Contiguous Memory Allocation

Memory Protection

For a computer to have memory access permissions controlled, memory protection is necessary. Preventing processes from accessing memory that has not been assigned to them is its main goal. This results in a segmentation fault or storage violation exception rather than a process impacting other processes or the operating system.

Memory Allocation in an Operating System

Memory Allocation in an operating system involves assigning memory or space to computer programs. Following are three types:

- **First Fit Allocation**: Allocating the first hole that is large enough for the program.

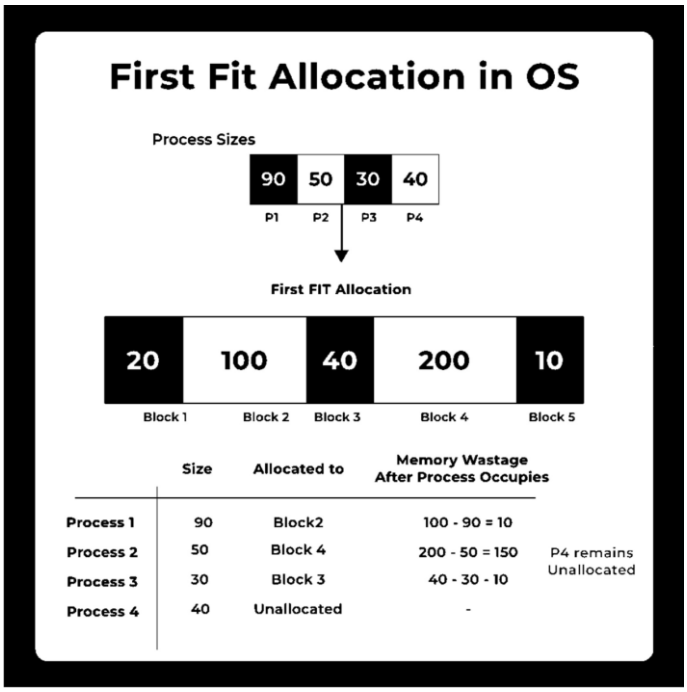


Figure 3.5: First Fit Allocation in OS

- **Best Fit Allocation**: Allocating the smallest hole that is large enough for the program.

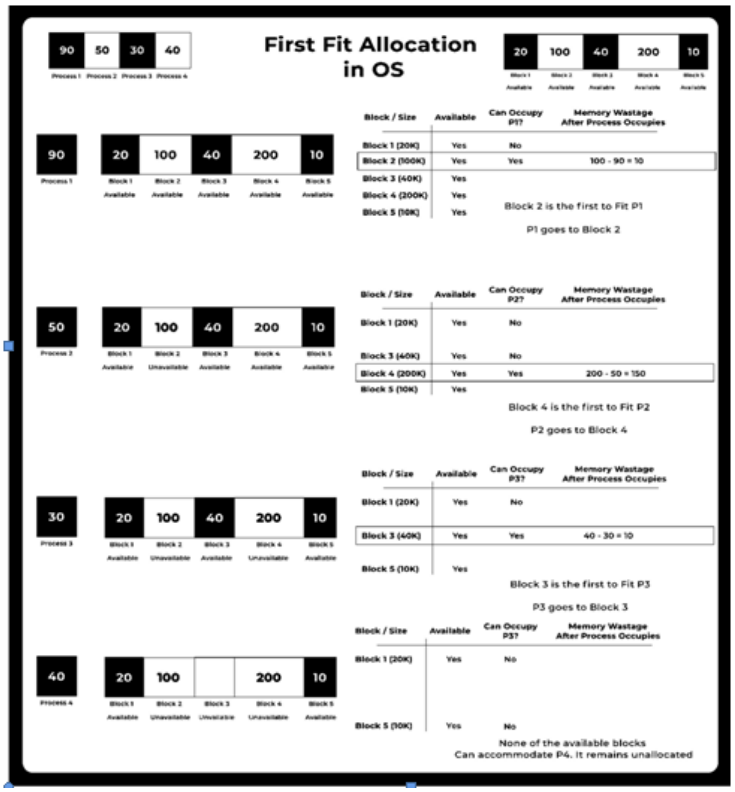


Figure 3.6: Best Fit Allocation in OS

Worst Fit Allocation: Allocating the largest hole that is large enough for the program.

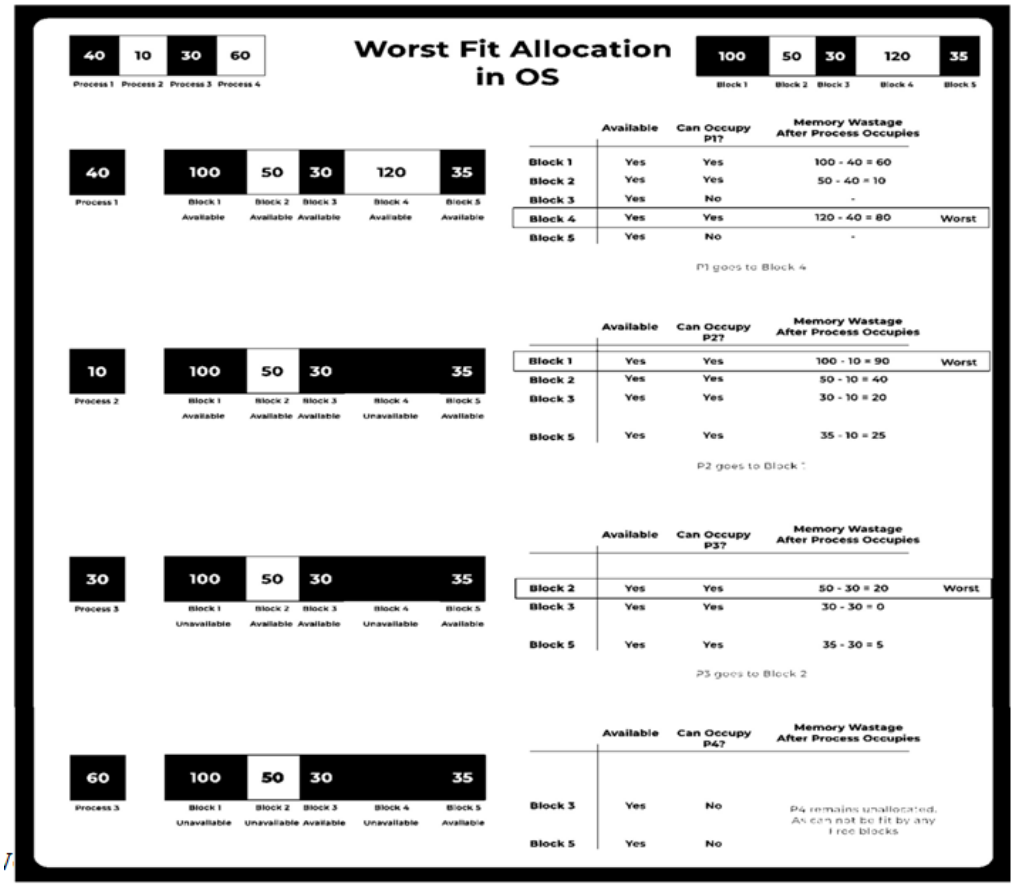


Figure 3.7: Worst Fit Allocation in OS

Fragmentation:

When processes are loaded and then removed from memory after execution, fragmentation takes place resulting in small free holes that cannot be utilized by new processes due to either lack of consolidation or failure to meet the memory requirements. To enable multiprogramming effectively, minimizing memory wastage and addressing fragmentation issues are crucial. There are two types of fragmentation:

1. Internal Fragmentation: This occurs when memory blocks allocated to processes exceed their requested sizes, leaving unused space and creating an internal fragmentation problem. For instance, in a fixed partitioning scheme where memory blocks of 3MB, 6MB, and 7MB are available, allocating a 2MB block for a process results in 1MB of wasted memory, which cannot be allocated to other processes.

2. External Fragmentation: In this scenario, although free memory blocks exist, they cannotbe assigned to processes because the blocks are not contiguous. For example, if processes of sizes 2MB, 4MB, and 7MB receive memory blocks of sizes 3MB, 6MB, and 7MB, respectively, the remaining 1MB and 2MB cannot be allocated to a new process demandinga 3MB block due to non-contiguous free memory space.

Both the first-fit and best-fit memory allocation systems are susceptible to external fragmentation. Compaction is used to address external fragmentation by consolidating all free memory space into a single large block, allowing efficient utilization by other processes.

Alternatively, another solution to external fragmentation involves allowing processes' logical address spaces to be non-contiguous, thereby enabling processes to be allocated physical memory wherever available.

3.3 Paging

One way to allocate non-contiguous memory is by paging. There is a predetermined size for this partitioning strategy. Both primary memory and secondary memory are split into equal-sized chunks during paging.

These segments are called pages in secondary memory and frames in main memory.

Using paging, pages from secondary memory are loaded into main memory. Each segment in this process is equal in size to a page. The process is divided into segments, and the size of the last segment can be the same as a page. The frames in main memory contain the pages of the process.

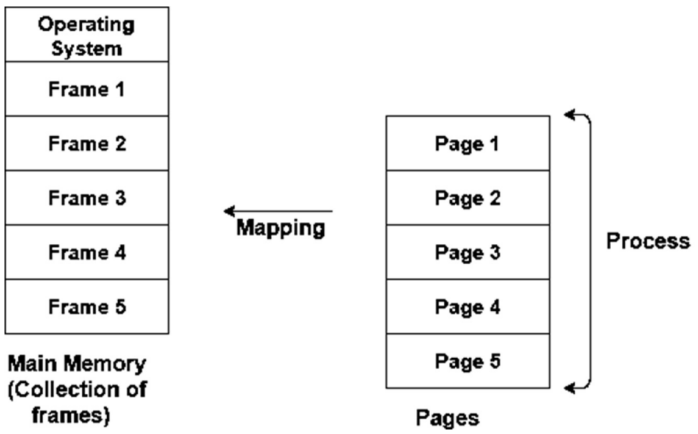


Figure 3.8: Paging

3.4 Segmentation

Segmentation is another non-contiguous memory allocation technique like paging. Segmentation divides a process into mounted (fixed) size pages at random, just like paging does. It's a dividing motif that comes in different sizes. Segmentation, like paging, does not create divisions of the same size between main and secondary memory. Secondary memory area units are divided into segments. Information for each segment is stored in a structure called a segmentation table. This table holds crucial details about each segment, specifically the segment's length (Limit) and its starting address (Base). During the segmentation process, the CPU produces a logical address composed of a segment number and an offset. The address is deemed valid if the offset is less than the segment's length.

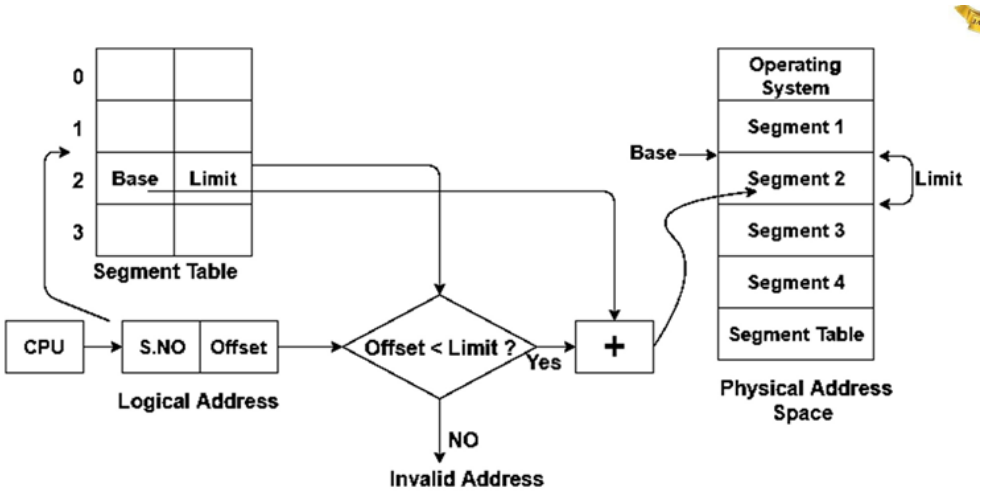


Figure 3.9: Segmentation

Difference between Paging and segmentation

Operating systems use two memory management strategies, paging and segmentation, to facilitate the efficient utilization of memory resources. However, they differ in their approach and organization of memory.

1. Segmentation:

a. Basic Concept: Segmentation divides the logical address space into variable-sized segments, which can represent different parts of a program such as code, data, stack, etc.

b. Size of Segments: Segments can vary in size, and each segment is typically defined by its starting address and length.

c. Address Translation: When a software requests memory access, its logical address is

divided into a segment number and an offset inside that segment.

d. . To find the base addressof a segment in physical memory, utilize the segment number as an index into a segment table or segment descriptor table. The physical address is then created by adding the offset to this base address.
e. Fragmentation: Segmentation can lead to internal fragmentation, where memory within a segment is not fully utilized.
f. Protection and Sharing: Segments can be protected and shared independently, allowing for finer control over memory access rights and sharing among processes.

2. Paging:

a. Basic Concept: Paging divides the physical memory into fixed-size blocks called pages and divides the logical address space into fixed-size blocks called page frames.
b. Size of Pages: Pages are typically of a fixed size determined by the hardware architecture (e.g., 4 KB, 8 KB).
c. Address Translation: When a program issues a memory access request, its logical address is divided into a page number and an offset within that page. To determine the physical frame number corresponding to the page in physical memory, the page number is used as an index in a page table. By adding the offset to the base address of the frame, the physical address is then generated.
d. Fragmentation: Paging can lead to external fragmentation, where small free memory spacesscattered throughout the memory cannot be effectively utilized due to their size not matching the page size.
e. Protection and Sharing: Paging facilitates memory protection and sharing at the page level,allowing for more coarse-grained control compared to segmentation.

3.5 Virtual memory concepts

Users can create the illusion of having ample main memory through a technique called virtual memory, where a portion of secondary memory acts as primary memory. This allows users to load larger processes than the physical main memory can accommodate by creating the appearance of additional RAM availability for loading processes. Rather than loading a single large process entirely into main memory, the operating system loads individual components of multiple processes as needed.

As a result, there will be more multiprogramming, which will result in higher CPU usage.

How Does Virtual Memory Operate?

In the current world, virtual memory is now commonly employed. In this scheme, the operating system scans the RAM for pages that have not been recently used or referenced, and transfers those pages to secondary memory. This frees up space in the main memory to accommodate new pages when the number of pages needing to be loaded exceeds the memory's capacity.

This operation is entirely automatic, so the computer seems to have an endless amount of RAM.

3.6 Demand Paging

Demand paging is a widely used technique for managing virtual memory. It involves keeping less frequently accessed pages of a process in secondary memory. When a page fault occurs or there's a demand for a page, that page is then copied into the main memory. Pages to be replaced are selected using various page replacement algorithms, which will be discussed in detail later.

Consider two processes, P1 and P2, each with four pages of 1 KB size. The main memory is divided into eight frames, each also 1 KB in size. The first two frames are reserved for the operating system. The third frame contains the first page of P1, with the remaining frames allocated for pages from different running processes in main memory.

Each page table for P1 and P2 is 1 KB in size and can fit within one frame. The page table base address (5 for P1 and 7 for P2) is stored in a CPU register. When accessing the actual page table entry corresponding to a logical address, the page number is added to this page table base address.

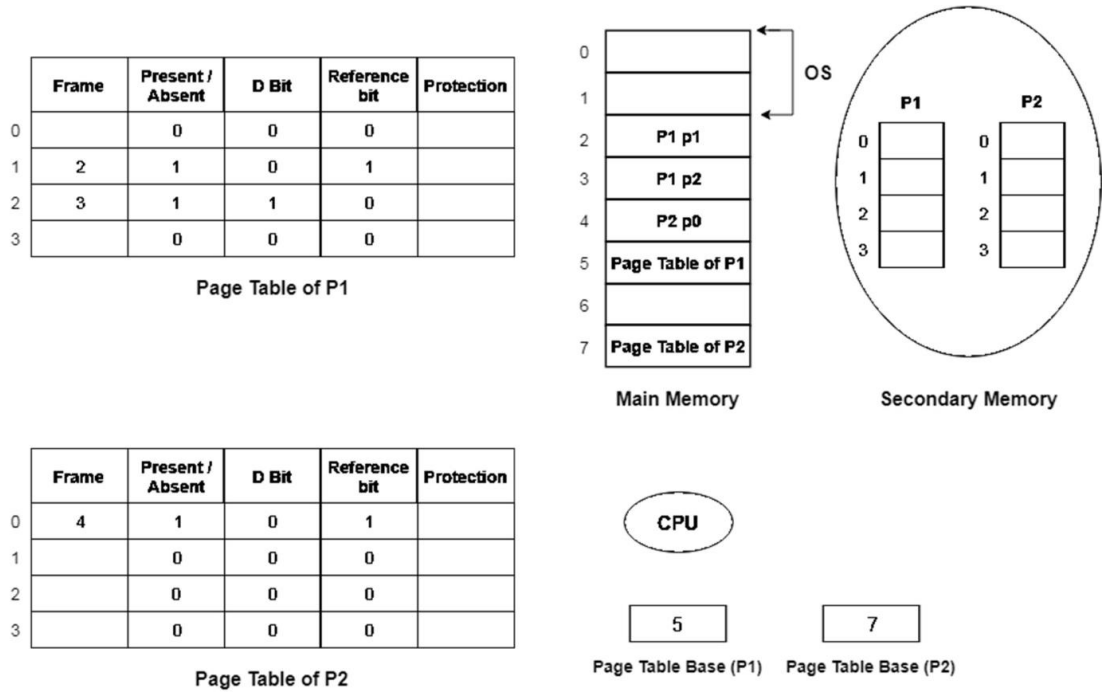


Figure 3.10: Demand Paging

Benefits of Virtual Memory

- There will be more multiprogramming to a greater extent.
- Large applications can be executed by the user with less real RAM.
- Purchasing more memory RAMs is not necessary.

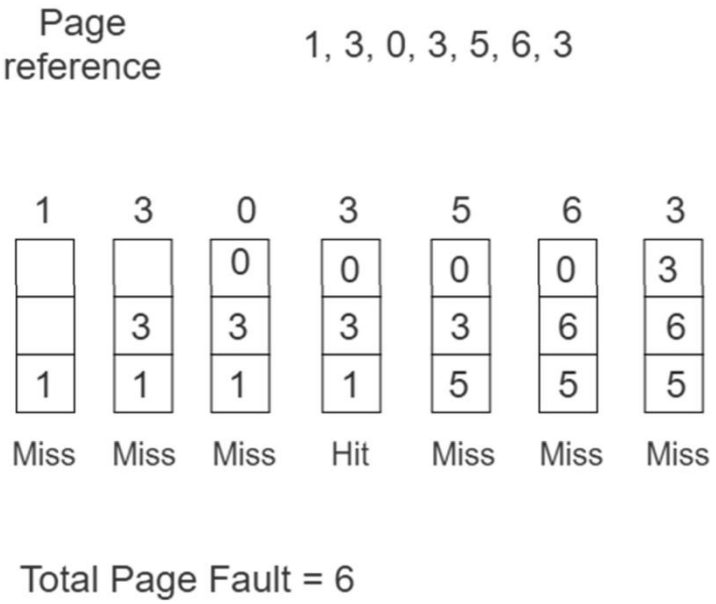
Negative aspects of virtual memory

- Since switching takes time, the system gets slower.
- The process of moving between applications takes longer.
- Less hard disk space will be available for use by the user.

3.7 Page replacement Algorithms

First In First Out (FIFO) – The simplest page replacement algorithm involves the operating system maintaining a queue to track every page in memory, with the oldest page positioned at the front of the queue. When a page replacement is necessary, the operating system selects the page at the front of the queue (the oldest page) for eviction.

Example 1 With three page frames, have a look at the page reference string 1, 3, 0, 3, 5, 6. Count the number of page errors.



Since there are always empty spaces at first, 1, 3, and 0 are assigned to the empty slots, which leads to 3 Page Faults.

Since it is already in memory when 3 arrives, there are 0 page faults.

Next comes number five, which takes the place of page slot one as it is not present in memory.

>Fault on One Page.

When page number six appears, it also takes up memory, therefore it replaces page number three, or >1 Page Fault.

Ultimately, it is unavailable when 3 arrives, thus it replaces 0 1 page error.

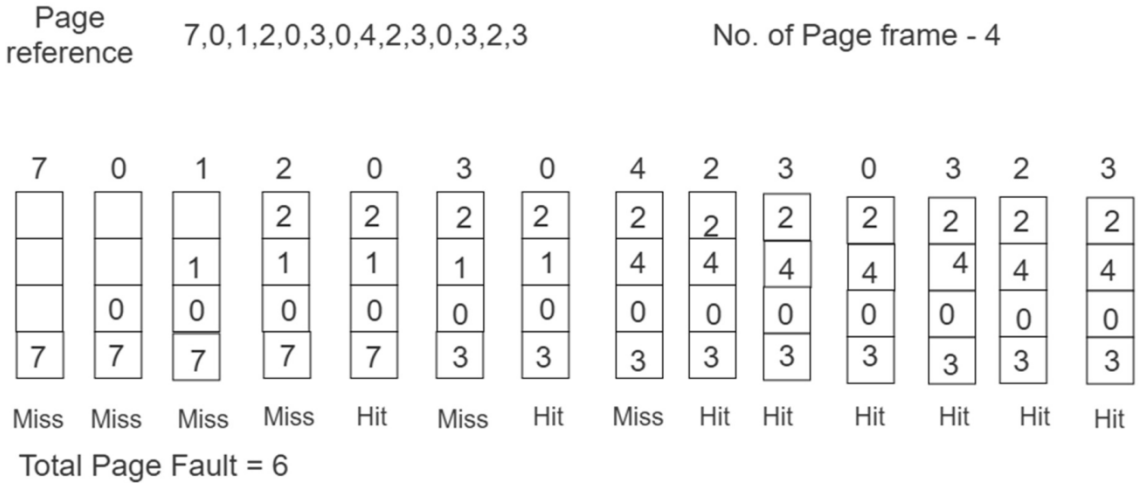
Belady's anomaly: This anomaly demonstrates that when employing the First in First Out (FIFO) page replacement mechanism and increasing the number of page frames, it is feasible to have morepage faults.

For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.

Optimal Page replacement –

Pages that are least likely to be used in the future are replaced using this method.

Second example: Examine the references on pages 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, and the four-page frame. Determine the page fault number.



Since all slots are initially empty, there are four page faults when the pages 7, 0, 1, and 2 are assigned to these empty slots.

As “0” already exists, there is “0” Page fault.

Since “3” is not used for the longest period of time in the future, it will replace “7” when it arrives.>One-page error.

Since “0” already exists, there is “0” Page fault.1 —> 1 Page Fault will be replaced with 4.

Since they are already in the memory, the next page reference string is —> “0” Page fault. Although page replacement is ideal, it not achievable in reality since the operating system isunable to anticipate future requests. Optimal Page Replacement is used to establish standard bywhich alternative replacement algorithms can be measured.

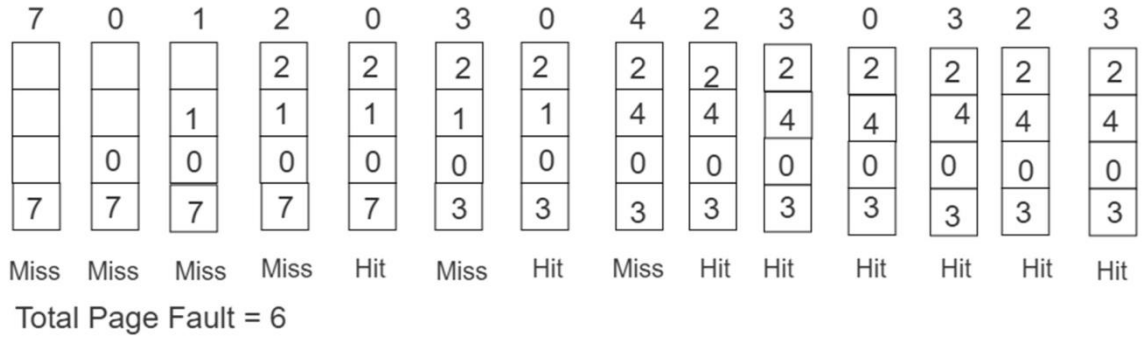
- **Least Recently Used –**

The page that has been used the least will be replaced using this algorithm.

Example #3 With four page frames, have a look at the page reference string “7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2”. Count the number of page errors.

.

Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3 No. of Page frame - 4



Here LRU has same number of page fault as optimal but it may differ according to question.

Since there are no slots at first when “7, 0, 1, 2” are assigned to the empty slots, there are 4 page faults;nevertheless, since “0” is already there, there are “0” page faults.
Since it is the least used (>1 Page fault), “3” will replace “7” when it arrives.Since “0” is already in memory, there is “0” page fault.
“1” —> 1 Page Fault will be replaced with “4”.

Since they are already in the memory, the next page reference string is —> “0” Page fault.

3.8 Thrashing

1. Introduction to Thrashing:

- Thrashing is a phenomenon in operating systems where excessive paging activity occurs, leading to severe degradation in system performance.
- It occurs when the system is unable to meet the memory demands of running processes, causing frequent page faults and excessive disk I/O operations.

2. Causes of Thrashing:

- Inadequate Physical Memory: This occurs when the system's RAM, or physical memory,is insufficient to support the current set of operating programs.
- Overcommitment of Memory: This occurs when the total amount of memory allotted to active programs surpasses the physical memory and swap space that are available.
- Inefficient Page Replacement: When the page replacement algorithm used by the

operating system is unable to effectively manage memory, leading to frequent page faults andunnecessary disk I/O operations.

- Excessive Multiprogramming: When the system attempts to execute more processes concurrently than can be accommodated by the available physical memory, resulting in excessive context switching and memory contention.

3. Symptoms of Thrashing:

- High Disk Activity: Excessive paging activity leads to a significant increase in disk I/Ooperations as the system continuously swaps pages between disk and RAM.
- Severe Performance Degradation: The system becomes unresponsive and sluggish as aresult of frequent page faults and disk thrashing.
- Decreased Throughput: The overall system throughput decreases as the CPU spends moretime handling page faults and disk I/O operations than executing useful work.
- Increased Response Time: Processes experience delays in accessing memory due to thehigh contention for memory resources caused by thrashing.

4. Detection and Prevention:

- Monitoring System Performance: Administrators can monitor system performance metrics such as CPU utilization, disk I/O rates, and page fault rates to detect signs of thrashing.
- Adjusting Memory Allocation: Increasing the amount of physical memory or adjusting memory allocation policies to better match the memory requirements of running processes can help prevent thrashing.
- Optimizing Page Replacement: Employing efficient page replacement algorithms like LRU (Least Recently Used) or CLOCK can effectively reduce thrashing occurrences by making optimal decisions on which pages to remove from memory.
- Limiting Multiprogramming: Controlling the number of concurrent processes or adjusting scheduling policies to prevent excessive multiprogramming can help alleviate thrashing by reducing memory contention.

5. Mitigation Strategies:

- Swapping and Swapping Policies: Swapping out less frequently used pages to disk and implementing intelligent swapping policies can help reduce the impact of thrashing by freeing up physical memory for more critical pages.
- Priority-Based Memory Allocation: Assigning higher priority to critical processes or memory-intensive applications can ensure that they receive sufficient memory resourcesto prevent thrashing.

- Dynamic Memory Management: Implementing dynamic memory management techniques such as memory ballooning or memory hotplug to adjust memory allocation dynamically based on workload demands can help mitigate thrashing.

Knowledge Check 1

Fill in the Blanks.

1. In_____, the logical address space is divided into fixed-size blocks called pages. (Swapping/**Paging**)
2. _____is a memory management technique used by operating systems to facilitate the efficient utilization of memory resources. (**Virtual memory concepts**/Page replacement Algorithms)
3. _____ divides the logical address space into variable-sized segments, representing different parts of a program. (Paging/**Segmentation)**
4. _____ is a phenomenon in operating systems where excessive paging activity occurs, leading to severe degradation in system performance. (**Thrashing**/Demand Paging)
5. _____is a technique used in virtual memory systems to efficiently manage memory by loading only the necessary pages into RAM. (**Demand Paging/** Page replacement Algorithms)

Outcome-Based Activity 1

How can participants develop a practical understanding of memory management concepts in operating systems through interactive activities and discussions?

3.9 Self-Assessment Questions:

1. Explain the concept of memory management in operating systems. How does it contribute to efficient resource allocation and utilization?
2. Compare and contrast swapping and paging techniques in memory management. What are the advantages and disadvantages of each approach?
3. Define segmentation in the context of memory management. How does it differ from paging? Provide examples of scenarios where segmentation is preferred over paging.

Unit 4 Storage Management

Learning Outcomes:

- Students will be able to understand the Storage Management.
- Students will be able to identify File concepts and access methods.
- Students will be able to demonstrate the Directory Structure.
- Students will be able to analyse the File system mounting.

Structure:

4.1 Storage management

4.2 File concepts

4.3 File access methods

4.4 Directory structure

4.5 File-system mounting

4.6 File sharing

4.7 Protection

4.8 File system structure and implementation

4.9 Directory implementation

4.10 File allocation methods

4.11 Recover

- Knowledge Check 2
- Outcome-Based Activity 2

4.12 Disk scheduling criteria and algorithms

4.13 Summary

4.15 Self-Assessment Questions

4.16 References / Reference Reading

4.1 Storage Management

Storage Management is the administration of data storage equipment used for storing user or computer-generated data. It serves as a tool or set of processes employed by administrators to safeguard data and storage equipment. The process allows users to optimize storage device usage and ensure data integrity across various media. Storage management encompasses subcategories such as security and virtualization, and it involves different types of provisioning or automation, constituting the broader storage management software market.

Key Attributes of Storage Management:

1. Performance: Ensures efficient and effective utilization of storage capacity within the system.
2. Reliability: Maintains the consistent and dependable functionality of storage resources.
3. Recoverability: Facilitates the recovery of data in the event of loss or system failures.
4. Capacity: Manages the overall storage capacity of the system.

Features of Storage Management:

1. It optimizes the utilization of storage devices within the system.
2. Storage management is treated as an allocated and managed resource for optimal corporatebenefits.
3. It serves as a fundamental system component in information systems.
4. Used to enhance the performance of data storage resources.

Advantages of Storage Management:

1. Simplifies the management of storage capacity.
2. Reduces time consumption associated with storage-related tasks.
3. Improves overall system performance.
4. Enhances organizational agility in virtualization and automation technologies.

Limitations of Storage Management:

1. Limited Physical Storage Capacity: The quantity of data that may be stored is limited by thephysical storage capacity that operating systems can only manage.
2. Performance Decline with Increasing Storage Utilization: Increased disk access times and othervariables can cause a system's performance to decline as more data is stored.
3. Complexity: Storage management complexity grows with the size of the storage environment.

4. Cost: Storing large amounts of data can be expensive, with additional storage capacity costsaccumulating.

5. Security Issues: Storing sensitive data poses security risks, requiring robust security features toprevent unauthorized access.

6. Backup and Recovery Challenges: Backup and recovery can be challenging, particularly whendealing with data stored across multiple systems or devices.

4.2 File Concepts:

- Definition: In computing, a file is a collection of related information stored on a storage device.
- Attributes: Files typically have attributes such as name, type, size, location, and permissions.
- Types of Files: Files can be categorized into different types based on their content and usage,such as text files, binary files, executable files, etc.
- File Operations: Common file operations include creation, reading, writing, updating, anddeletion.
- File Metadata: Metadata associated with files includes creation date, last modified date, owner,permissions, etc.
- File Systems: Files are organized and managed within file systems, which define how data isstored, accessed, and managed on storage devices.

4.3 File Access Methods:

- **Sequential Access:**
  - Definition: Sequential access refers to the method of reading or writing data from or to a file in a linear manner, from the beginning to the end of the file.
  - Description: In sequential access, data is processed or manipulated one piece at a time, following the order in which it is stored in the file. Each piece of data is accessed sequentially after the previous one.
  - Usage: Sequential access is commonly used in scenarios where data is processed or retrieved in the same order as it was written or stored. For example, reading log files, processing text files line by line, or streaming media files sequentially.

  Advantages:
  - Simple and easy to implement.
  - Suitable for tasks that involve reading or writing data sequentially, such as data processingor streaming.

Disadvantages:

- Inefficient for accessing specific data points within large files, as it requires reading through the entire file from the beginning.
- Not suitable for applications that require random access to data.

- **Random Access:**
  - Definition: Random access refers to the method of accessing data directly at any point within a file, without the need to sequentially read through the entire file.
  - Description: With random access, data can be accessed or manipulated at specific locations within the file using file pointers or offsets. This allows for faster access to specific portionsof data, regardless of their position in the file.
  - Usage: Random access is essential for applications that require quick access to specific data points within large files, such as database operations, file editing, or random data retrieval.

Advantages:

- Allows for efficient access to specific data points within large files.
- Ideal for applications that require frequent access to random data locations, such as databases or file editors.

Disadvantages:

- More complex to implement compared to sequential access.
- Requires additional overhead to manage file pointers or offsets.

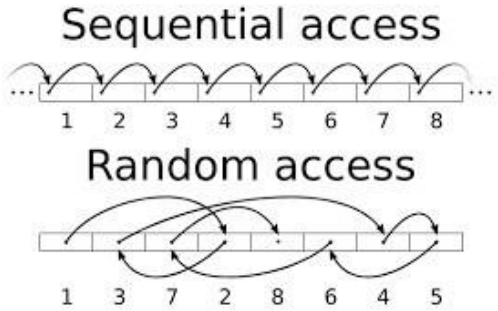


Figure 4.1 : File Access Methods

- **Methods:**

- Sequential Access Methods: Sequential access methods involve reading data from or writing data to a file in a linear, sequential manner. This includes operations such as reading from afile line by line or writing data to a file sequentially.

- Random Access Methods: Random access methods enable accessing specific locations within a file directly, without the need to sequentially read through the entire file. This includes methods like using file pointers or offsets to access specific data points within the file.

Applications:

- Sequential Access Applications: Sequential access is suitable for tasks that involve processing data in the order it is stored in the file. Examples include reading log files, parsing text files, or streaming media files sequentially.
- Random Access Applications: Random access is essential for applications that require quick access to specific data points within large files. Examples include database operations, file editing, or random data retrieval in scientific applications.

4.4 Directory Structure:

- **Definition:**

A directory structure refers to the hierarchical organization of files and directories within a computer's file system.

It provides a systematic way to arrange and manage files and directories, facilitating efficient storage and retrieval of data.

- **Components:**
  - **Directories (Folders):**

Directories are containers used to organize and group related files and subdirectories.

They serve as logical partitions within the file system, allowing users to categorize and managefiles based on their purpose or content.

Directories can contain files, other directories (subdirectories), or a combination of both.

  - **Files:**

Files are individual units of data stored on a storage device.

They represent documents, programs, multimedia files, or any other type of digital content. Files are organized within directories based on their content or functionality.

- **Navigation:**
  - Users can navigate through the directory structure using various methods, including:
  - Command-Line Interface (CLI): Users can use commands such as cd (change directory), ls (list directory contents), and mkdir (make directory) to navigate and

manipulate the directory structure.

- Graphical User Interface (GUI): Graphical file managers provide visual representations of the directory structure, allowing users to navigate through folders and files using mouse clicks and drag-and-drop operations.
- Navigation allows users to traverse directories, access files, and perform file management tasks such as copying, moving, renaming, and deleting.

- **Pathnames:**
  - Pathnames are used to specify the location of a file or directory within the directory structure.
  - A pathname consists of a series of directory names separated by slashes (/).
  - There are two types of pathnames:
  - Absolute Pathname: Specifies the complete path from the root directory to the desired file or directory. For example, /home/user/documents/file.txt.
  - Relative Pathname: Specifies the path relative to the current working directory. For example, documents/file.txt.
  - Pathnames allow users and applications to locate and access files and directories within the directory structure. They provide a standardized way to reference file locations regardless of the underlying file system or platform.

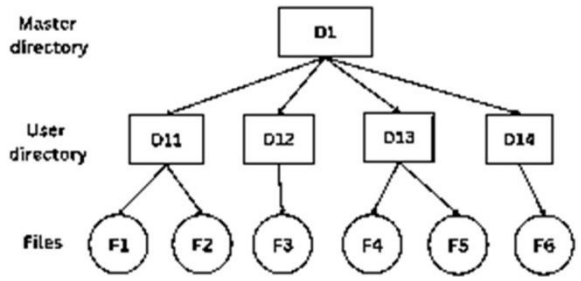


Figure 4.2: Directory Structure

4.5 File-System Mounting:

- **Definition:**
  - File system mounting is the process of integrating a file system into the operating system'sfile hierarchy, making its contents accessible to users and applications.
  - When a file system is mounted, its directory structure and files become part of the overallfile system structure of the operating system.

- **Mount Point:**
  - A mount point is a directory within the existing file system where the newly mounted filesystem is attached.
  - It serves as the access point for the mounted file system, allowing users and applicationsto navigate and interact with its contents.
  - The mounted file system becomes a subtree rooted at the mount point within the existingfile system hierarchy.
- **Types:**
  - **Temporary Mounting:**

    Temporary mounting involves attaching a file system to the operating system's file hierarchy for a specific duration or session.

    Examples include mounting a removable USB drive or a CD-ROM, where the file system remains accessible until it is unmounted or the system is restarted.
  - **Permanent Mounting:**

    Permanent mounting involves integrating a file system into the file hierarchy in a persistentmanner, so it remains accessible across system reboots.

    Examples include mounting a network file system (NFS) or a partition on the hard drive, ensuring that the file system remains mounted even after system shutdown or restart.
- **Commands:**

- Operating systems provide commands to manage the mounting and unmounting of file systems:

- Mount Command:

- The mount command is used to attach a file system to the file hierarchy at a specified mountpoint.

- Syntax: `mount [options] <source> <target>`

- Example: `mount /dev/sdb1 /mnt/usb`

- Umount Command:

- The umount command is used to detach a mounted file system from the file hierarchy, makingit inaccessible to users and applications.

- Syntax: `umount [options] <target>`

- Example: `umount /mnt/usb`

- These commands allow administrators to manage file system mounting operations, including mounting and unmounting removable devices, network file systems, and disk partitions. They are essential for managing storage resources and ensuring data accessibility within the operating system.

4.6 File Sharing:

- **Definition:**

File sharing refers to the capability of allowing multiple users or computers to access and modify the same files simultaneously.

It enables users to share and collaborate on documents, data, and resources, fostering efficient communication and productivity.

- **Network File Systems:**

Network file systems (NFS) facilitate file sharing over a network, enabling remote users to access files stored on network-attached storage (NAS) devices or file servers.

NFS allows users to access files as if they were stored locally on their own computers, regardless of their physical location.

This technology enables seamless access to shared files and resources across distributed networks, promoting collaboration and data sharing.

- **Access Control:**

File-sharing systems incorporate mechanisms for access control to regulate and manage user permissions for accessing shared files and resources.

Access control mechanisms include authentication, authorization, and permission management to ensure data security and integrity.

Administrators can define access policies, grant specific permissions to users or groups, and restrict unauthorized access to sensitive data.

- **Collaboration:**

File sharing facilitates collaboration among users by providing a platform for sharing documents, data, and resources.

It allows multiple users to work on the same files simultaneously, enabling real-time collaboration and feedback.

Users can share files, make edits, leave comments, and track revisions, promoting efficient teamwork and communication.

Collaboration features such as version control, file locking, and document sharing enhance productivity and streamline workflow processes.

4.7 Protection:

- **Definition:** Protection in the context of file systems refers to mechanisms implemented to control access to files and directories, ensuring data security and preventing unauthorized modifications.
- **Access Control:**

Access control lists (ACLs) and permissions are commonly used for protection.

ACLs specify which users or system processes are granted access to objects, along with the type of access (read, write, execute).

- **Encryption:**

Encryption can be employed to protect sensitive data within files, making it unreadable without the proper decryption key.
It adds an extra layer of security to prevent unauthorized access to the file content.

4.8 File System Structure and Implementation:

- **File System Structure:**

Describes how files and directories are organized within a file system.
Common structures include hierarchical, flat, and tree structures.

- **Implementation:**

File systems are implemented using data structures and algorithms to manage file organization, storage allocation, and access methods.
Common implementations include FAT32, NTFS, ext4, and more, each with its own features and optimizations.

4.9 Directory Implementation:

- Definition: Directory implementation refers to the method used to store and manage directorieswithin a file system.
- Data Structures: Directories are typically implemented using data structures such as B-trees, hash tables, or linked lists to efficiently organize and retrieve file and subdirectory information.

- Metadata: Directories store metadata, including file names, attributes, and pointers to the actualfile data or subdirectories.

4.10 File Allocation Methods:

File allocation methods dictate the strategy employed to allocate disk space for storing the contents of files within a file system. These methods are pivotal in efficient storage management and influence the organization and retrieval of data within the file system.

Methods of File Allocation:

1. Contiguous Allocation:

- Description: Contiguous allocation assigns a contiguous block of disk space to store theentirety of a file's contents.
- Process: When a file is created, a contiguous block of sufficient size is allocated from thedisk space. The file's data is then stored contiguously within this allocated block.

Advantages:

- Simple and straightforward implementation.
- Sequential access to file data is efficient, as data blocks are contiguous.

Disadvantages:

- Fragmentation can occur over time, leading to inefficient utilization of disk space.
- Difficulty in finding contiguous free blocks for larger files, potentially leading to filefragmentation.

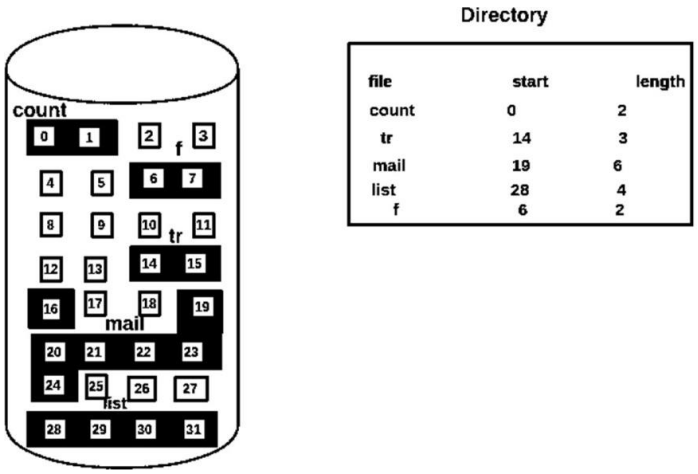


Figure 4.3 : Contiguous Allocation

2. Linked Allocation:

- Description: Linked allocation utilizes pointers to link together blocks of data scattered across the disk.
- Process: Each file is represented by a linked list of disk blocks, with each block containinga portion of the file's data and a pointer to the next block in the sequence.

Advantages:

- Flexibility in allocating non-contiguous disk space, reducing fragmentation.
- Efficient for handling variable-sized files, as blocks can be allocated dynamically.

Disadvantages:

- The overhead of storing pointers for each block, increasing storage overhead.

Random access to file data can be inefficient due to scattered disk blocks.

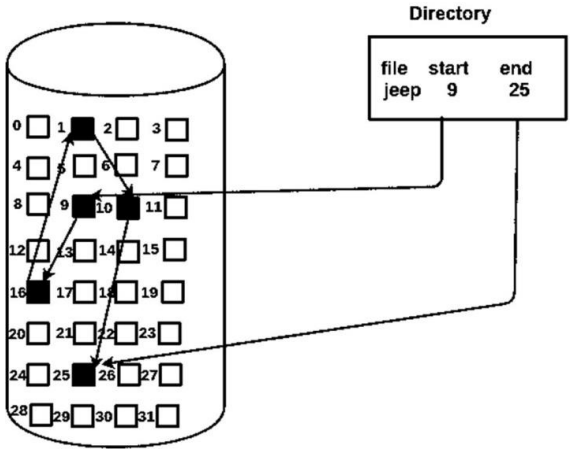


Figure 4.4 :Linked Allocation

3. Indexed Allocation:

- Description: Indexed allocation employs an index table to locate data blocks associatedwith each file.
- Process: Each file has an index table (or index block) containing pointers to the disk blocksholding the file's data.

Advantages:

- Efficient for both sequential and random access to file data.
- Reduced storage overhead compared to linked allocation, as only one index block is required per file.

Disadvantages:

- Limited by the size of the index table, affecting scalability for large files or a large numberof files.
- Additional overhead in managing and updating the index table, particularly for dynamic file size changes.

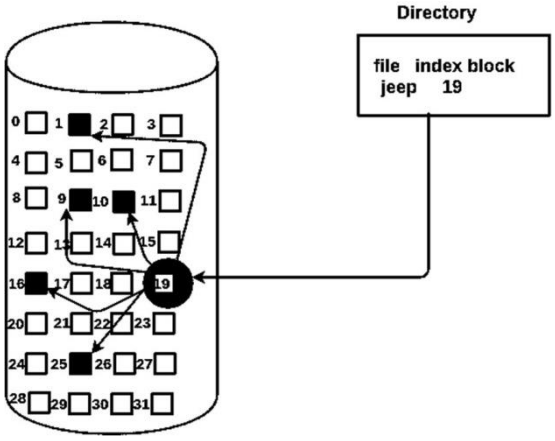


Figure 4.5 : Indexed Allocation

These file allocation methods play a crucial role in optimizing disk space utilization, managing file storage efficiently, and influencing file access performance within a file system. The selection of an appropriate file allocation method depends on factors such as file system requirements, file size distribution, and access patterns.

4.11 Recover:

Recovery mechanisms within file systems are engineered to mitigate the adverse effects of system failures, crashes, or inadvertent deletions on data integrity. These mechanisms are integral components of file system architecture aimed at ensuring the preservation and restoration of datain the event of unexpected incidents or errors.

Backup and Restore:

Regular backups constitute a fundamental aspect of data recovery strategies within file systems. It involves creating duplicate copies of critical data and storing them separately from the primary storage infrastructure. These backup copies serve as a safeguard against data loss caused by systemfailures, hardware malfunctions, or human errors such as accidental deletions. In the event of data corruption or loss, these backup copies can be utilized to restore the affected files or directories, thereby reinstating data integrity and continuity of operations.

Journaling:

Journaling file systems employ a specialized technique to maintain a detailed log or journal of changes made to the file system before they are finalized or committed. This journal serves as a chronological record of transactions, capturing modifications such as file updates, metadata changes, or directory reorganizations. In the event of a system failure or unexpected shutdown, the journal can be utilized to replay or roll back transactions, ensuring the consistency and integrity of the file system. Journaling enhances recovery efficiency by minimizing data inconsistencies and reducing the likelihood of file system corruption following abrupt interruptions.

Check and Repair Utilities:

File systems often incorporate built-in utilities designed to inspect, validate, and repair file system integrity. These utilities, such as fsck (File System Consistency Check), are tasked with identifying and rectifying errors, inconsistencies, or structural abnormalities within the file system. During routine maintenance or system boot-up procedures, these utilities are invoked to perform comprehensive checks on the file system's structural integrity, data consistency, and metadata accuracy. By identifying and resolving potential issues proactively, these utilities contribute to maintaining the stability, reliability, and functionality of the file system, thereby facilitating data recovery and ensuring optimal system performance.

These recovery mechanisms collectively form an essential component of file system resilience strategies, safeguarding data integrity, and enabling prompt restoration of data in the face of unforeseen events or errors. By leveraging backup and restore procedures, journaling techniques, and check and repair utilities, file systems can mitigate the impact of disruptions and uphold data reliability and availability.

- **Knowledge Check 2**

  State True and False

1. File access methods determine how disk space is allocated to store the content of files.(False)
2. File-system mounting is the process of making a file system available for access within theoperating system's file hierarchy. (True)
3. File sharing allows multiple users or computers to access and modify the same files simultaneously. (True)
4. Protection in file systems ensures data security by encrypting all files stored within thesystem. (False)

5. Directory implementation refers to the method used to store and manage files within a filesystem. (False)

- **Outcome-Based Activity 2**

  Compare and contrast different file access methods and justify which method(s) would be mostsuitable for the intended use cases of the operating system.

4.12 Disk scheduling criteria and algorithms

Disk scheduling is a crucial component of operating systems responsible for managing and optimizing the access to disk storage devices. It involves determining the order in which disk I/O requests are serviced to minimize seek time, reduce latency, and improve overall disk performance. Various criteria and algorithms are employed to schedule disk operations efficiently.

Disk Scheduling Criteria:

1. Minimization of Seek Time:

- Seek time refers to the time taken by the disk arm to move to the desired track where datais located.
- Disk scheduling algorithms aim to minimize seek time by scheduling I/O requests in amanner that reduces the distance the disk arm needs to travel.

2. Fairness:

- Fairness ensures that all processes or users receive a fair share of disk access and are notunfairly starved of disk resources.
- Disk scheduling algorithms should strive to provide equitable access to disk I/O operationsamong competing processes or users.

3. Throughput:

- Throughput measures the rate at which data is transferred between the disk and the CPU.
- Disk scheduling algorithms should maximize throughput by efficiently utilizing disk bandwidth and minimizing idle time.

4. Latency Reduction:

- Latency refers to the time delay between initiating a disk I/O request and receiving therequested data.
- Disk scheduling aims to reduce latency by prioritizing and optimizing the order of I/Orequests to minimize waiting time for data retrieval.

Disk Scheduling Algorithms:

1. "First-Come First-Served (FCFS)":

- "FCFS" is a simple disk scheduling algorithm that services disk I/O requests in the order they arrive.
- It is easy to implement but may result in poor performance, especially if there are long seektimes or if requests are not spatially localized.

Example:

Disk Request Sequence: 45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25

Disk Tracks: 100

Starting Head Position: 50 (moving left)

To find the number of head movements in cylinders using FCFS scheduling, we need to follow the First-Come, First-Served approach where the requests are serviced in the order they are received.

1. Start at the initial head position: 50
2. Move left to the first request: 45 (distance moved: 5)
3. Move left to the second request: 21 (distance moved: 24)
4. Move right to the third request: 67 (distance moved: 46)
5. Move right to the fourth request: 90 (distance moved: 23)
6. Move left to the fifth request: 4 (distance moved: 86)
7. Move right to the sixth request: 50 (distance moved: 46)
8. Move right to the seventh request: 89 (distance moved: 39)
9. Move left to the eighth request: 52 (distance moved: 37)
10. Move left to the ninth request: 61 (distance moved: 9)
11. Move right to the tenth request: 87 (distance moved: 26)
12. Move left to the eleventh request: 25 (distance moved: 62)

Total number of head movements in cylinders = 5 + 24 + 46 + 23 + 86 + 46 + 39 + 37 + 9 + 26 + 62 = 403 cylinders.

2. “Shortest Seek Time First (SSTF)”:

- “SSTF” selects the disk I/O request that requires shortest seek time from the current head position.
- It aims to minimize seek time and reduce disk arm movement, leading to improved performance compared to FCFS.

Example:

Disk Request Sequence: 45, 21, 67, 90, 4, 89, 52, 61, 87, 25

Disk Tracks: 100

Starting Head Position: 50

To find the number of head movements in cylinders using Shortest Seek Time First (SSTF) scheduling, we need to service the requests based on the shortest seek time from the current head position.

1. Start at the initial head position: 50

2. Calculate the seek time (absolute difference) between the current head position (50) and each request:

   Seek time for each request:

   - 45: |50 - 45| = 5

   - 21: |50 - 21| = 29

   - 67: |50 - 67| = 17

   - 90: |50 - 90| = 40

   - 4: |50 - 4| = 46

   - 89: |50 - 89| = 39

   - 52: |50 - 52| = 2

   - 61: |50 - 61| = 11

   - 87: |50 - 87| = 37

   - 25: |50 - 25| = 25

3. Select the request with the shortest seek time: 52 (seek time: 2)

4. Move the head to the selected request: 52

5. Repeat the process with the remaining requests, considering the current head position as thestarting point.

Seek Time Calculation:

|Current Head Position - Requested Track|

Total number of head movements in cylinders using SSTF scheduling = 5 + 29 + 17 + 40 + 46 + 39 + 2 + 11 + 37 + 25 = 251 cylinders.

3. "SCAN (Elevator) Algorithm":

- The "SCAN" algorithm moves disk arm in one direction (e.g., from the innermost to outermost track), servicing requests along the way.
- After reaching at end, direction is reversed, and requests in opposite direction are serviced.
- "SCAN" prevents starvation of requests located at outer tracks but may result in increased average seek time for requests at the inner tracks.

Example

Disk Request Sequence: 98, 137, 122, 183, 14, 133, 65, 78

Disk Tracks: 100

Starting Head Position: 54 (moving left)

To find the number of head movements in cylinders using SCAN (Elevator) scheduling, we need to follow the SCAN algorithm where the head moves in one direction, servicing requests along the way, and reverses direction when reaching end.

1. Start at the initial head position: 54
2. Move left to the end of the disk: 0 (servicing any requests encountered along the way)
3. Reverse direction to the right and continue servicing requests until reaching the end of the disk:98, 137, 122, 65, 78
4. Reverse direction to the left and service any remaining requests: 14, 133, 65 (including backtracking to the start to ensure all requests are serviced)

Total number of head movements in cylinders using SCAN scheduling = 54 (to the end) + 98 + 137 + 122 + 65 + 78 + 14 + 133 + 65 = 684 cylinders.

4. “C-SCAN Algorithm”:

- “C-SCAN” is a variant of the “SCAN” algorithm that only services requests in one direction (e.g., from innermost to outermost track) and then jumps to the beginning of the disk to service requests again.
- It aims to reduce seek time by avoiding unnecessary movement of the disk arm in the reverse direction.

To solve the disk request sequence using the C-SCAN (Circular-SCAN) algorithm, let's assume the disk arm moves in the left direction from the starting head position.

Example

Given disk request sequence: 45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25Starting head position: 50 (moving in the left direction)

Since C-SCAN moves the disk arm in one direction and then jumps to other end of disk without servicing requests in between, we need to simulate this behaviour.

1. Start at the initial head position: 50
2. Move left to the end of the disk (track 0), servicing requests along the way: 45, 21, 4, 25
3. Jump to the right end of the disk (track 99) without servicing requests in between.
4. Move left again from the right end to the last serviced request (track 4), servicing requests alongthe way: 50, 52, 61, 67, 87, 89, 90
5. Jump back to the starting head position (track 50) without servicing requests in between.

The resulting order of servicing the requests is: 50, 52, 61, 67, 87, 89, 90, 45, 21, 4, 25

To calculate the number of head movements in cylinders, we need to sum the distances between consecutive requests:

= |50 - 52| + |52 - 61| + |61 - 67| + |67 - 87| + |87 - 89| + |89 - 90| + |90 - 45| + |45 - 21| + |21 - 4| +

|4 - 25|

= 2 + 9 + 6 + 20 + 2 + 1 + 45 + 24 + 17 + 21

= 147 cylinders.

So, the number of head movements in cylinders using the C-SCAN algorithm is 147.

5. “LOOK Algorithm”:

- “LOOK” is similar to “SCAN” but does not scan the entire disk in one direction. Instead, itreverses direction when there are no pending requests in the current direction.
- “LOOK” reduces seek time compared to “SCAN” by only servicing requests in the directionthe disk arm is already moving.

Example:

To solve the disk request sequence using the LOOK algorithm, let's assume the disk arm moves in the left direction from the starting head position.

Given disk request sequence: 45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25Starting head position: 50 (moving in the left direction)

The “LOOK” algorithm services requests in the direction of movement until there are no requests in that direction. Then reverses direction and services requests in the opposite direction until there are no more requests.

1. Start at the initial head position: 50
2. Move left to the minimum request (21), servicing requests along the way: 45, 21, 4
3. Reverse direction and move right to the maximum request (90), servicing requests along theway: 50, 52, 61, 67, 87, 89, 90
4. Since there are no more requests in the left direction, stop.

The resulting order of servicing the requests is: 45, 21, 4, 50, 52, 61, 67, 87, 89, 90

To calculate the number of head movements in cylinders, we need to sum the distances between consecutive requests:

= |50 - 45| + |45 - 21| + |21 - 4| + |4 - 50| + |50 - 52| + |52 - 61| + |61 - 67| + |67 - 87| + |87 - 89| + |89 - 90|

= 5 + 24 + 17 + 46 + 2 + 9 + 6 + 20 + 2 + 1

= 132 cylinders.

So, the number of head movements in cylinders using the LOOK algorithm is 132.

These disk scheduling algorithms vary in complexity, efficiency, and suitability for different workload patterns. The choice of algorithm depends on factors such as disk workload characteristics, seek time considerations, and system requirements for fairness and throughput.

Disk scheduling plays a vital role in optimizing disk performance and improving the overall responsiveness of the operating system.

4.13 Summary

- A technique for freeing up space in main memory by transferring a process from main memoryto secondary memory.
- Using secondary memory as an extension of main memory is possible for the operating systemthanks to a memory management strategy.
- A method of managing memory that separates memory into segments and allocates a distinctprocess to each segment.
- A method where pages only enter main memory when the CPU requests or needs them invirtual memory systems.
- Algorithms that determine which page, when a new page is received, needs to be replaced.
- A state in which the system pages more frequently than it executes.
- A designated grouping of pertinent data stored on secondary storage.
- The method used to read and access a file or written to.
- A hierarchical structure for folders and files.
- The act of enabling users to access a file system.
- The capacity for several users to access the same directory or file.
- The procedure makes sure only authorized users can access files.
- The arrangement and storage of files on a file system.
- The manner in which file systems implement directories.
- The file system's method of allocating files.
- The procedure for restoring lost or corrupted data.
- The order in which disk I/O requests are handled is determined by disk scheduling methodsand criteria.

4.14 Self-Assessment Questions

1. In memory management, what distinguishes paging from swapping?
2. What is the virtual memory demand paging purpose?
3. In storage management, what distinguishes file allocation techniques from file access techniques?
4. What distinguishes the disk scheduling algorithms SCAN and C-SCAN from one another?
5. What is the memory management goal of segmentation?

4.15 References / Reference Reading

- "Deitel H.M., Operating Systems, Pearson Education."
- "Stallings William, Operating System, PHI Learning."
- "Godbole A.S., Operating Systems, Tata McGraw-Hill, New Delhi."

Unit 5: Security and Threats

Learning Outcomes:

- Students will be able to understand protection goals, domains, and access matrices.
- Students will be able to recognize and address security problems and threats.
- Students will be able to select and implement security tools effectively.
- Students will be able to comprehend distributed system types, structures, and protocols.
- Students will be able to manage distributed file access, replication, and synchronization efficiently.

Structure

5.1 Protection

Protection in an operating system is a crucial aspect of security, governing system access by restricting file access for users. It safeguards resources like CPU and memory, ensuring authorized users can access them. Essential in multiprogramming systems, protection prevents interference between users, intercepts malicious breaches, and enforces policies for fair resource usage. Its role includes preserving user data confidentiality and preventing unauthorized access or manipulation by others. Overall, protection is vital for maintaining system integrity, security, and adherence to established policies.

5.2 Goals of Protection

1. **Resource Access Control:** The primary goal of protection mechanisms in an operating system is to control and regulate access to system resources such as memory, files, devices, and other objects. This ensures that only authorized users and processes can access or modify these resources, thereby preventing unauthorized access or misuse.
2. **Data Confidentiality:** Protection mechanisms aim to maintain the confidentiality of sensitive data by preventing unauthorized users from accessing or viewing it. This is typically achieved through encryption techniques and access control mechanisms that restrict data access to authorized users or processes.
3. **Data Integrity:** Another key goal of protection is to ensure the integrity of data, meaning that data remains accurate and unaltered during storage, transmission, or processing. Protection mechanisms help prevent unauthorized modification or tampering of data by enforcing access controls and implementing integrity checks.
4. **System Stability:** Protection mechanisms contribute to the stability and reliability of the operating system by preventing unauthorized users or processes from disrupting system operations. By controlling access to critical system resources, protection mechanisms help maintain system stability and prevent unauthorized activities that could lead to system crashes or failures.
5. **User Accountability:** Protection mechanisms also facilitate user accountability by logging and monitoring user activities within the operating system. This helps in identifying and tracing unauthorized actions or security breaches to specific users or processes, thereby promoting accountability and deterring malicious activities.
6. **Isolation of Processes:** Protection mechanisms ensure the isolation of processes running on the system, preventing one process from interfering with or accessing the memory or

resources of another process. This isolation is crucial for ensuring system stability, security, and reliability, especially in multi-user or multi-tasking environments.

7. **Fault Tolerance:** Protection mechanisms contribute to fault tolerance by minimizing the impact of system failures or errors. By isolating processes and enforcing access controls, protection mechanisms help contain errors or failures within individual processes or components, thereby preventing cascading failures that could affect the entire system.
8. **Adaptability and Flexibility:** Protection mechanisms should be adaptable and flexible to accommodate changing security requirements and evolving threats. Operating systems should allow administrators to configure and customize protection settings based on their specific security needs, ensuring that protection mechanisms remain effective in addressing emerging security challenges.
9. **Performance Optimization:** While ensuring security and protection, operating systems should also strive to optimize system performance. Protection mechanisms should be designed and implemented in a way that minimizes overhead and resource usage, ensuring that system performance is not significantly impacted by security measures.
10. **Compliance and Regulation:** Finally, protection mechanisms should enable operating systems to comply with relevant security regulations, standards, and best practices. This includes ensuring adherence to industry-specific security standards and regulatory requirements, thereby ensuring legal and regulatory compliance.

5.3 Domains of Protection

1. **User Mode and Kernel Mode:**
    a. User Mode: Applications run in user mode, which restricts direct access to hardware resources and critical system functions.
    b. Kernel Mode: The operating system kernel runs in kernel mode, allowing direct access to hardware and privileged system operations. It controls and manages system resources on behalf of user applications.
2. **Memory Protection:**
    a. Address Spaces: Each process operates within its own virtual address space, ensuring isolation and preventing any process from accessing the memory of another.
    b. Memory Segmentation: Memory segmentation divides the address space into segments, such as code segment, data segment, and stack segment, with access control for each segment to prevent unauthorized access.

3. **Process Isolation:**
    a. Process Boundaries: Processes are isolated from each other, preventing interference and unauthorized access to the resources of other processes.
    b. Inter-Process Communication (IPC): Mechanisms like message passing and shared memory facilitate controlled communication between processes while maintaining isolation.

4. **File System Protection:**
    a. File Permissions: Access control lists (ACLs) and file permissions regulate access to files and directories, allowing or denying read, write, and execute permissions based on user roles.
    b. File Ownership: Assigning ownership to files and directories helps enforce accountability and control access rights.

5. **Device Protection:**
    a. Device Drivers: Device drivers manage access to hardware devices and enforce protection mechanisms to prevent unauthorized access or misuse.
    b. I/O Privileges: Access to I/O devices is restricted to privileged processes or through controlled system calls to ensure proper operation and prevent conflicts.

6. **Security Mechanisms:**
    a. Authentication: User authentication mechanisms like passwords and cryptographic keys verify user identities before granting access to system resources.
    b. Authorization: Authorization mechanisms define access rights based on user roles or permissions, ensuring that only authorized users can access specific resources.

7. **Error Handling and Exception Handling:**
    a. Error Detection: The operating system detects errors such as invalid memory access or illegal instructions, triggering appropriate actions to prevent system instability.
    b. Exception Handling: Exception handling mechanisms manage unexpected events or errors, ensuring system stability and preventing unauthorized access to critical resources.

8. **Privilege Levels:**

   a. Superuser Privileges: The superuser or root user has elevated privileges to perform administrative tasks and manage system resources.
   b. User Privileges: Regular users have limited privileges and are restricted from performing critical system operations to maintain system integrity and security.

These domains of protection in an operating system ensure the security, stability, and integrity of system resources while facilitating efficient and controlled access for user applications and processes.

5.4 Access matrix

The Access Matrix is a security model utilized in computer systems, depicted as a matrix that delineates the permissions granted to each process within a domain for different objects. Rows in the matrix represent domains, and columns represent objects. Each cell within the matrix specifies the access rights that a process in a particular domain has for a specific object. For instance, entry (i, j) specifies the operations that a process in domain Di can execute on object Oj.

Types of access rights include:

a. Read: Permission for a process to read the file.

b. Write: Authorization for a process to write into the file.

c. Execute: Ability for a process to execute the file.

d. Print: Access for a process to use the printer.

Domains may possess multiple rights, including combinations of the aforementioned rights.

Here's an example to illustrate how an access matrix functions.

	F1	**F2**	**F3**	**Printer**
D1	read		read	
D2				print
D3		read	execute	
D4	read write		read write	

“Observations of the above matrix”:

- The matrix consists of four domains and four objects: three files (F1, F2, F3) and one printer.
- A process in domain D1 can read files F1 and F3.Domain D4 has the same rights as D1 but also includes the right to write to files.
- The printer is accessible only to a process in domain D2.
- A process in domain D3 can read file F2 and execute file F3.

- **“Mechanism of access matrix”:**
- The access matrix mechanism involves various policies and semantic properties.
- It ensures that process in domain Di can only access objects listed in row i.
- Policies determine the rights included in the (i, j) entry of the access matrix.
- The operating system typically determines the domain in which each process executes.
- Users specify contents of access matrix entries.
- The association in between domains and processes can be either static or dynamic.
- The access matrix provides mechanism for managing association between domains and processes.

“Switch operation”:

- Switching process from one domain to another requires executing switch operation on the domain object.
- Domain switching is managed by including domains among the objects in the access matrix.
- Processes can switch from domain Di to domain Dj only if the switch right is granted in the access matrix entry (i, j).

	F1	F2	F3	Printer	D1	D2	D3	D4
D1	read		read			switch		
	F1	**F2**	**F3**	**Printer**	**D1**	**D2**	**D3**	**D4**
D2				print			switch	switch
D3		read	execute					
D4	read write		read write		switch			

As per matrix provided:

- A “process” running in “domain D2” can transition to domains “D3” and “D4”.
- A “process” operating in “domain D4” can transition to domain “D1”.
- A “process” in “domain D1” is permitted to switch to domain “D2”.

5.5 Security

Every computer system and software design must address security risks and implement measures to enforce security policies. However, finding a balance is crucial, as stringent security measures can raise costs and restrict system usability and functionality. Therefore, system designers must ensure efficient performance without sacrificing security.

Operating System Security involves ensuring the availability, confidentiality, and integrity of the operating system. It encompasses measures to safeguard against various threats like viruses, malware, and unauthorized access. These measures aim to prevent the theft, modification, or deletion of system assets in the event of a security breach.

Security in the context of operating systems involves protecting computer system resources such as software, CPU, memory, and disks from threats like viruses and unauthorized access. This protection is achieved by ensuring the integrity, confidentiality, and availability of the operating system. If a malicious user executes a computer application, it can lead to severe damage to the computer or the data stored within it.

5.6 Security problem

1. **Vulnerabilities:** Operating systems often contain vulnerabilities that attackers can exploit to gain unauthorized access, disrupt system operations, or steal sensitive information.

   These vulnerabilities may arise due to flaws in the operating system's design, implementation, or configuration.
2. **Malware:** Operating systems are susceptible to various forms of malware, including viruses, worms, Trojans, and ransomware. Malware can infect the system through malicious software downloads, email attachments, or compromised websites, compromising system security and integrity.
3. **Unauthorized Access:** Weak authentication mechanisms or misconfigured access controls can lead to unauthorized access to system resources. Attackers may exploit these vulnerabilities to gain unauthorized privileges, access sensitive data, or execute malicious activities on the system.

4. Data Breaches: Sensitive information may be accessed, stolen, or leaked by unauthorized parties as a result of inadequate data protection procedures. This can result in financial losses, privacy violations, and damage to the reputation of individuals or organizations.

5. **Denial-of-Service (DoS) Attacks:** Operating systems are vulnerable to denial-of-service attacks, where attackers flood the system with excessive traffic or requests, overwhelming its resources and causing disruptions in service availability. These attacks can render the system inaccessible to legitimate users and disrupt critical operations.
6. **Insider Threats:** Insider threats present a substantial security risk to operating systems, involving authorized users or employees with privileged access who may misuse their credentials to steal data, sabotage systems, or compromise security from within the organization.
7. **Inadequate Patch Management:** Failure to promptly apply security patches and updates leaves operating systems vulnerable to known vulnerabilities and exploits. Attackers can exploit these unpatched vulnerabilities to compromise system security and launch cyber attacks.
8. **Social Engineering:** Operating system security can be compromised through social engineering techniques, where attackers manipulate users into divulging confidential information, clicking on malicious links, or performing actions that compromise system security unintentionally.

9. **Insecure Network Communication:** Insecure network protocols and communication channels can expose operating systems to eavesdropping, man-in-the-middle attacks, and data interception. Encrypting network traffic and implementing secure communication protocols can mitigate these security risks.
10. **Lack of User Awareness:** Insufficient user awareness regarding security best practices, such as maintaining strong passwords, recognizing phishing attempts, and adopting safe browsing habits, can increase vulnerabilities in operating system security. Educating users about these risks and promoting security awareness can play a crucial role in mitigating such vulnerabilities.

These security problems highlight the importance of implementing robust security measures, regularly updating systems, enforcing access controls, educating users, and maintaining vigilance to protect operating systems from various security threats and vulnerabilities.

5.7 Threats

In computer security, a threat refers to any potential danger or risk that could compromise the confidentiality, integrity, or availability of computer systems, networks, data, or information. It encompasses malicious activities, vulnerabilities, or events that may lead to harm or compromise the security of digital assets.

Threats can come from various sources, including malicious actors (such as hackers, malware authors, or insiders), natural disasters, technical failures, or human errors. Common types of threats include malware (such as viruses, worms, Trojans), unauthorized access, denial-of-service (DoS) attacks, data breaches, social engineering attacks, and physical theft or damage to hardware.

Understanding and identifying threats is essential for implementing effective security measures to protect against potential risks and vulnerabilities. This includes implementing robust security controls, such as access controls, encryption, intrusion detection systems, firewalls, and security patches, to mitigate the impact of threats and ensure the security of computer systems and data.

Types of Threats:

1. **Program Threats:**
    a. Occur when user programs manipulate operating system processes to perform malicious actions.
    b. Examples:
    c. Virus: Self-replicating code that can modify/delete user files and crash computers.

d. Trojan Horse: Captures user login credentials and transfers them to a malicious user.
e. Logic Bomb: Software that behaves abnormally under specific conditions, otherwise functioning normally.
f. Trap Door: Program with a security weakness allowing unauthorized actions without user knowledge.

2. **System Threats:**

a. Involve the misuse of system services and network connections to disrupt user activities.
b. Can trigger program threats across a network, known as program attacks.
c. Examples:
d. Port Scanning: Identifies system vulnerabilities by connecting to specific ports via TCP/IP.
e. Worm: Process that consumes system resources by creating multiple clones, disrupting other processes and potentially halting a network.
f. Denial of Service (DoS): Prevents legitimate users from accessing system resources, such as the internet, by flooding the system with requests or traffic.

These threats pose risks to the security and integrity of computer systems, requiring effective security measures to mitigate their impact and protect against potential vulnerabilities.

Threats to Operating Systems:

1. **Malware:**

a. Includes viruses, worms, trojan horses, and other harmful software.
b. Typically short code snippets are capable of corrupting files, deleting data, replicating to spread further, and even causing system crashes.
c. Malware often operates silently, allowing criminals to extract important data unnoticed by the victim user.

2. **Network Intrusion:**

a. Involves unauthorized access to a system by masqueraders, misfeasors, and rogue users.
b. Masqueraders gain access using another user's account.
c. Misfeasors misuse programs, data, or resources after gaining legitimate access.

d. Rogue users attempt to evade access constraints and audit collection by assuming supervisory authority.

3. **Buffer Overflow:**

a. Also known as buffer overrun, it's a common and dangerous security issue in operating systems.
b. Occurs when more input is placed into a buffer than its allotted capacity, potentially overwriting other information.
c. Attackers exploit this vulnerability to crash a system or insert specially crafted malware, allowing them to take control of the system.

5.8 Security Tools in Operating Systems

1. **Antivirus Software:**

a. Detects, prevents, and removes malware infections on the operating system.
b. Scans files, directories, and memory for known malware signatures and suspicious behaviour.
c. Provides real-time protection by monitoring system activities and blocking malicious processes.

2. **Firewalls:**

a. Controls incoming and outgoing network traffic based on predetermined security rules.
b. Prevents unauthorized access to the operating system by filtering network packets.
c. Can be software-based or hardware-based, providing an additional layer of defense against network-based attacks.

3. **Intrusion Detection Systems (IDS):**

a. Monitors system and network activities for signs of malicious behaviour or policy violations.
b. Analyzes network traffic, system logs, and audit trails to detect potential security incidents.
c. Alerts administrators or takes automated actions to respond to detected threats or suspicious activities.

4. **Encryption Tools:**

a. Encrypts data at rest and in transit to protect confidentiality and prevent unauthorized access.

b. It employs cryptographic algorithms to transform plaintext data into ciphertext, which can only be reverted to plaintext using the correct keys.
c. Provides secure storage and transmission of sensitive information, such as passwords, personal data, and confidential documents.

5. **Patch Management Software:**
   a. Manages the installation and deployment of security patches, updates, and fixes for the operating system and installed software.
   b. Ensuring that the operating system and software applications are regularly updated with the latest security patches is essential to address known vulnerabilities and exploits effectively.
   c. Helps mitigate the risk of security breaches and exploits by promptly applying patches to vulnerable systems.
6. **Access Control Tools:**
   a. Enforce access control policies to regulate user access to system resources and data.
   b. Implement mechanisms like access control lists (ACLs), role-based access control (RBAC), and mandatory access control (MAC) to define and enforce access permissions effectively.
   c. Restrict access to sensitive files, directories, and system settings based on user roles, groups, or individual user permissions.
7. **Security Information and Event Management (SIEM):**
   a. Collects, correlates, and analyzes security-related events and logs from various sources within the operating system.
   b. Provides centralized monitoring, real-time threat detection, and incident response capabilities.
   c. Helps identify security incidents, investigate security breaches, and facilitate compliance with security policies and regulations.

These security tools play a crucial role in protecting operating systems from various security threats and vulnerabilities, ensuring the confidentiality, integrity, and availability of system resources and data.

Certainly! Here are notes on the classification of operating systems:

5.9 Classification of Operating Systems

1. **Single-User vs. Multi-User:**
   a. Single-User: Designed for a single user at a time, typical for personal computers and workstations.
   b. Multi-User: Supports multiple users simultaneously, allowing concurrent access to system resources, commonly found in servers and mainframes.
2. **Single-Tasking vs. Multi-Tasking:**
   a. Single-Tasking: A single-tasking operating system can execute only one task or program at a time, necessitating users to wait for completion before initiating another task.
   b. Multi-Tasking: Capable of executing multiple tasks or programs concurrently, enabling users to switch between tasks seamlessly, common in modern desktop and server operating systems.
3. **Batch Processing vs. Interactive Systems:**
   a. Batch Processing: Executes predefined sequences of tasks without user interaction, suitable for processing large volumes of data in batches, often used in mainframe environments.
   b. Interactive Systems: Allows users to interact with the operating system in real-time, providing a graphical user interface (GUI) or command-line interface (CLI) for user input and system feedback, common in personal computers and workstations.
4. **Real-Time Systems:**
   a. Hard Real-Time: Guarantees strict deadlines for processing tasks, critical for applications where timing is crucial, such as industrial control systems and embedded devices.
   b. Soft Real-Time: Provides best-effort scheduling to meet deadlines most of the time, used in applications where occasional delays are tolerable, such as multimedia systems and online gaming.
5. **Distributed Systems:**
   a. Network Operating Systems (NOS): Coordinates and manages resources across multiple interconnected computers or nodes, facilitating communication and resource sharing, commonly used in client-server environments.
6. **Clustered Systems:**

   Groups multiple independent systems into a single, highly available and scalable computing resource, used for high-performance computing (HPC) and fault- tolerant applications.

7. **Embedded Systems:**
    a. Designed for specific purposes, these embedded systems are integrated into larger systems or devices, including consumer electronics, automotive systems, medical devices, and industrial machinery.
    b. They are often optimized for low-power consumption, real-time operation, and specific functionalities, typically running on specialized hardware platforms with limited resources.
8. **Time-Sharing Systems:**
    a. Shares computing resources, such as CPU time and memory, among multiple users or processes concurrently, providing the illusion of simultaneous execution.
    b. Allows users to interact with the system through terminals or remote connections, commonly used in multi-user environments, such as cloud computing and virtualized environments.

Understanding the classification of operating systems helps in identifying their key characteristics, functionalities, and suitability for different computing environments and applications.

- **Knowledge Check 1**

  Fill in the Blanks

  1. The purpose of protection mechanisms in operating systems is to regulate access to system resources and ensure _____. (**integrity, confidentiality, availability**/security, usability, efficiency)
  2. One key aspect of protection in operating systems is user authentication, which verifies the identity of users attempting to access the system or its resources, ensuring only _____access. (**authorized** / unrestricted)
  3. Domains of protection define sets of objects and subjects and the _____between them. (**access rights**/control mechanisms)
  4. Security in operating systems encompasses various strategies, mechanisms, and policies designed to ensure confidentiality, integrity, availability, and _____. (**authenticity**/redundancy)
  5. One of the primary purposes of protection is to prevent _____ breaches by enforcing access controls. (**malicious**/unintentional)

- **Outcome-Based Activity 1**

  Develop a concept map that visually represents the interconnected concepts of protection and security in operating systems.

5.10 Self-Assessment Questions:

1. What is meant by "protection" in the context of computer systems and operating systems?
2. Discuss the goals of protection mechanisms in operating systems. How do they contribute to system security?
3. Explain the concept of "domains of protection" and how they are implemented in operating systems.
4. Describe the purpose and structure of an "access matrix" in the context of access control mechanisms.
5. What is "security" in the context of computer systems? How does it relate to protection mechanisms?

Unit 6: Distributed System

Learning Outcomes:

- Students will be able to understand Distributed System.
- Students will be able to recognize and address types of Network Based OS.
- Students will be able to select and implement Communication structure and protocol.
- Students will be able to manage distributed file access, replication, and synchronization efficiently.

Structure:

6.1 Distributed Systems

In plain language, a distributed system is a network of computers that collaborate to accomplish a shared objective. Instead of depending on a single central computer, tasks and data are distributed across several interconnected computers. This setup enhances performance, fault tolerance, and scalability.

For example, consider a file-sharing system like BitTorrent. In BitTorrent, when you download a file, you're not downloading it from a single central server. Instead, the file is divided into small pieces, and each piece is hosted on different computers (called peers) connected to the BitTorrent network. When you download the file, your BitTorrent client connects to multiple peers simultaneously and downloads different pieces of the file from each peer. This distributed approach allows for faster download speeds because you're downloading from multiple sources at once. Additionally, if one peer goes offline, your download can still continue because you can download missing pieces from other peers.

In this example, the BitTorrent network is a distributed system because it distributes the file-sharing workload across multiple computers (peers) connected by a network, rather than relying on a single central server.

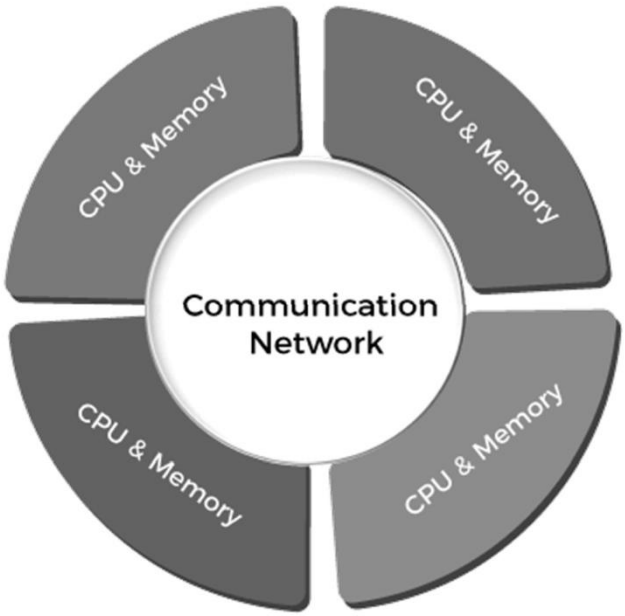


Figure 6.1 : Types of network-based OS

6.2 Types of Network-based operating systems:

Network-based operating systems can be categorized into different types depending on their architectural design, intended use, and operational capabilities. Here are several typical

classifications of network-based operating systems:

1. **Client-Server Operating Systems:**
   a. In client-server operating systems, computers are classified into two main roles: clients and servers. Clients initiate requests for services or resources, while servers fulfill these requests by providing the requested services over the network.
   b. Examples include Microsoft Windows Server, Linux distributions with server editions (e.g., CentOS, Ubuntu Server), and Novell NetWare.
2. **Peer-to-Peer Operating Systems:**
   a. Peer-to-peer (P2P) operating systems enable computers (peers) to communicate and share resources directly with each other without the need for a central server.
   b. Examples include various Linux distributions (e.g., Ubuntu Desktop, Fedora) and decentralized file-sharing networks like BitTorrent.
3. **Distributed Operating Systems:**
   a. Distributed operating systems distribute computation and data processing tasks across multiple interconnected computers to improve performance, fault tolerance, and scalability.
   b. Examples include distributed versions of Unix-like operating systems (e.g., Plan 9 from Bell Labs, MINIX 3) and research-oriented distributed operating systems like Amoeba andSprite.
4. **Network-Attached Storage (NAS) Operating Systems:**
   a. NAS operating systems are specialized operating systems designed to manage and provide storage resources over a network.
   b. Examples include FreeNAS, OpenMediaVault, and proprietary NAS solutions like Synology DiskStation Manager and QNAP QTS.
5. **Router Operating Systems:**
   a. Router operating systems are embedded operating systems designed to run on network routers and manage network traffic routing and forwarding.
   b. Examples include Cisco IOS (Internetwork Operating System) for Cisco routers, Juniper Junos for Juniper routers, and OpenWrt for open-source router firmware.
6. **Embedded Network Operating Systems:**
   a. Embedded network operating systems are lightweight operating systems designed to run on embedded devices and IoT (Internet of Things) devices connected to a network.
   b. Examples include Embedded Linux distributions (e.g., OpenWrt, Buildroot), RTOS (Real-

Time Operating Systems) like FreeRTOS and Contiki, and proprietary embedded OS solutions.

Certainly! Here's information about "4.3.2 Network Structure and Topologies":

6.3 Network Structure and Topologies

In the context of distributed systems, network structure and topologies play a crucial role in determining how data is transmitted, how nodes communicate with each other, and how fault tolerance and scalability are achieved. Here's a perspective on network structure and topologies concerning distributed systems:

Network Structure in Distributed Systems:

1. Physical vs. Logical Structure:
    a. Distributed systems can have both physical and logical network structures. The physical structure refers to the actual physical connections between nodes, while the logical structure defines the logical relationships and communication paths between nodes.
2. Node Placement and Connectivity:
    a. The placement and connectivity of nodes in a distributed system depend on factors such as geographic location, network infrastructure, and communication requirements. Nodes may be distributed across different geographical locations, connected through various network technologies such as LANs, WANs, or the Internet.
3. Redundancy and Fault Tolerance:
    a. Distributed systems frequently integrate redundancy and fault tolerance mechanisms to uphold system reliability and availability. These include redundant connections and backup nodes, which mitigate the effects of node failures or network outages.

Topologies in Distributed Systems:

1. **Mesh Topology:**

a. In a distributed system, a mesh topology can be employed where each node is connected to every other node in the network. This provides multiple communication paths between nodes, improving fault tolerance and ensuring reliable data transmission.

2. **Star Topology:**
    a. A star topology in a distributed system involves a central node (e.g., a server) to which all other nodes are connected. This central node facilitates communication and coordination

among distributed nodes, making it easier to manage and monitor the network.

3. **Hybrid Topology:**

   a. Distributed systems often utilize a hybrid topology that combines multiple basic topologies to meet specific requirements. For example, distributed system may have combination of star and mesh topologies, allowing for centralized communication along with redundant paths for fault tolerance.

4. **Overlay Networks:**

   a. Overlay networks are virtual networks built on top of existing physical networks, enabling nodes in a distributed system to communicate and collaborate efficiently. Examples include peer-to-peer (P2P) overlay networks used in file-sharing systems and content delivery networks (CDNs) for distributing web content.

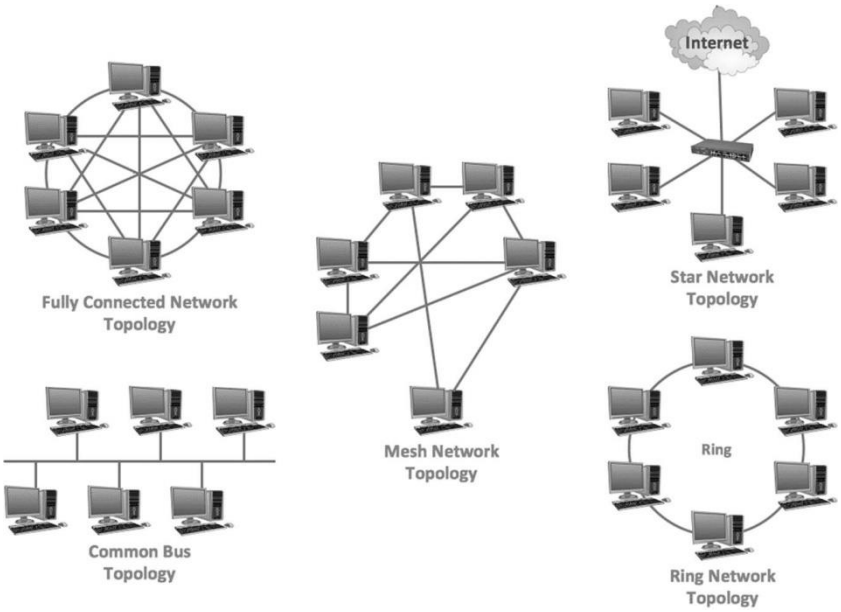


Figure 6.2: Network Topologies

Understanding the network structure and topologies in distributed systems is essential for designing resilient, scalable, and efficient distributed architectures that can effectively handle the complexities of distributed computing environments.

6.4 Communication Structure & Protocol

1. **Communication Structure:**

   a. In distributed systems, communication structure refers to the arrangement and organization of communication channels and protocols used for data exchange between nodes.

   b. Nodes in a distributed system communicate over a network, which can be physical or

virtual, and may span multiple geographical locations.

c. Communication structures vary based on factors such as network topology, scalability requirements, and communication patterns between nodes.

2. Point-to-Point vs. Broadcast Communication:

a. Point-to-point communication involves direct communication between two nodes in a distributed system. Messages are sent from a sender node to a specific receiver node, typically over a dedicated communication channel.

b. Broadcast communication involves sending messages from a sender node to multiple receiver nodes simultaneously. Messages are broadcasted over a shared communication channel, allowing multiple nodes to receive the same message.

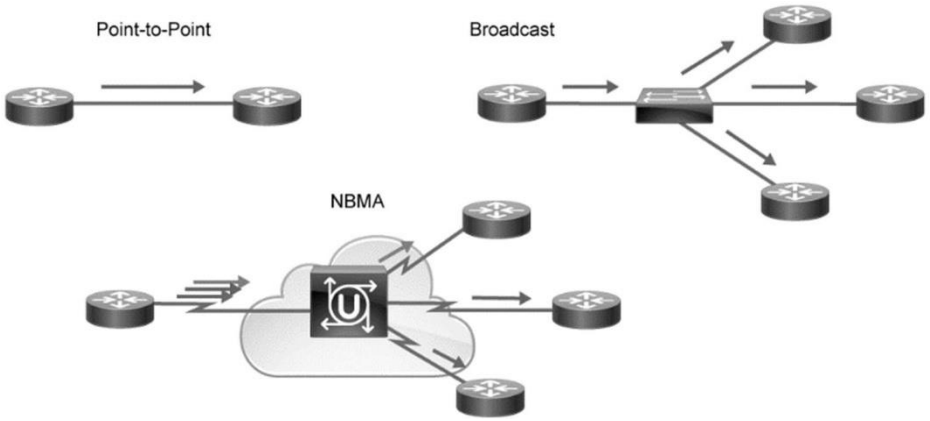


Figure 6.3: Point-to-Point and Broadcast Communication

3. Client-Server vs. Peer-to-Peer Communication:

a. In client-server communication, a client node initiates by requesting services or resources from a server node. The client sends a request to the server, which then processes the request and responds back to the client accordingly.

b. Peer-to-peer (P2P) communication entails direct interaction among peer nodes without relying on a central server. Peers within the network can function as both clients and servers, enabling them to share resources and collaborate directly with one another.

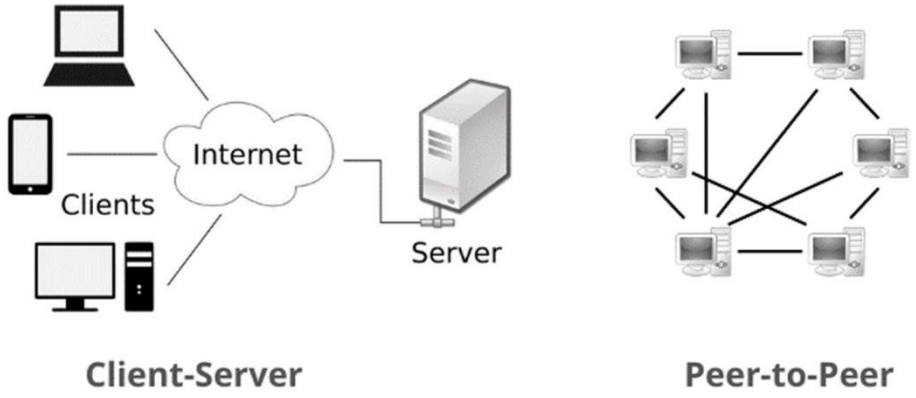


Figure 6.4: Client-Server and Peer-to-Peer Communication

4. Communication Protocols:

a. Communication protocols define rules and standards for exchanging data and coordinating communication between nodes in a distributed system.
b. Protocols specify message formats, data encoding schemes, error detection and correction mechanisms, and rules for message transmission, routing, and delivery.
c. Various communication protocols are utilized in distributed systems, such as TCP/IP (Transmission Control Protocol/Internet Protocol) for reliable and connection-oriented communication, UDP (User Datagram Protocol) for lightweight and connectionless communication, HTTP (Hypertext Transfer Protocol) for web-based interactions, and MQTT (Message Queuing Telemetry Transport) for efficient publish-subscribe messaging.

5. Message-Oriented Middleware (MOM):

a. Message-oriented middleware is a software infrastructure that facilitates communication between distributed applications by providing messaging services and implementing communication protocols.
b. MOM systems decouple communication participants by allowing them to send and receive messages asynchronously, enabling reliable and scalable communication in distributed systems.

Understanding communication structure and protocols is essential for designing and implementing efficient and reliable communication mechanisms in distributed systems, ensuring seamless data exchange and collaboration between distributed nodes.

6.5 Design Issues

1. **Scalability:**
    a. Scalability refers to the ability of a distributed system to handle increasing workloads and accommodate growing numbers of users or data volumes without sacrificing performance or reliability.
    b. Designing for scalability involves considerations such as distributed data storage, load balancing, and parallel processing to ensure that the system can scale horizontally by adding more resources or nodes as needed.
2. **Fault Tolerance:**
    a. Fault tolerance is ability of distributed system to continue operating and providing services in presence of component failures or network disruptions.
    b. Designing for fault tolerance involves implementing redundancy, replication, and error detection mechanisms to detect and recover from failures automatically, ensuring uninterrupted service availability and data integrity.
3. **Consistency and Replication:**
    a. Consistency refers to the agreement of data across distributed nodes in a system. Maintaining consistency is challenging in distributed systems due to factors like network latency and node failures.
    b. Replication involves creating multiple copies of data across distributed nodes to improve availability, fault tolerance, and performance. Designing a replication strategy involves balancing consistency, availability, and partition tolerance (CAP theorem).
4. **Concurrency Control:**
    a. Concurrency control ensures that multiple concurrent operations on shared resources in a distributed system do not lead to data inconsistency or conflicts.
    b. Designing concurrency control mechanisms involves implementing techniques such as locking, timestamps, and distributed transactions to coordinate access to shared resources and maintain data integrity.
5. **Security:**
    a. Security is essential in distributed systems to protect sensitive data, prevent unauthorized access, and ensure the confidentiality, integrity, and availability of system resources.
    b. Designing for security involves implementing encryption, authentication, access control,

and auditing mechanisms to mitigate security threats such as data breaches, unauthorized access, and denial-of-service attacks.

6. **Communication Protocols and Middleware:**
    a. Designing effective communication protocols and middleware is crucial for enabling seamless communication and coordination between distributed nodes in a system.
    b. Choosing appropriate communication protocols and middleware involves considering factors such as message delivery guarantees, latency, reliability, and compatibility with existing systems and standards.

7. **Resource Management:**
    a. Resource management involves efficiently allocating and managing distributed system resources such as CPU, memory, storage, and network bandwidth to optimize performance and utilization.
    b. Designing resource management mechanisms involves implementing scheduling algorithms, resource monitoring, and dynamic resource allocation strategies to ensure fair resource sharing and maximize system efficiency.

Addressing these design issues is essential for building robust, reliable, and efficient distributed systems that can meet the demands of modern applications and support complex distributed computing environments.

6.6 Distributed File System

Introduction to "Distributed File System (DFS)":

A "distributed file system (DFS)" is network-based file storage system that spans various file servers and locations. It permits programs to access and manage data like local files, enabling users to access files from any system connected to the network. DFS facilitates regulated and permitted sharing of information and files among network users while ensuring that servers maintain complete control over the data and provide access control.

Components of DFS:

DFS consists of two main components:

- Local Transparency: Achieved through the namespace component, which ensures that clients do not need to be aware of number or location of file servers and storage devices.

- Redundancy: Achieved through a file replication component, which creates multiple copies of files distributed across different storage devices or servers. This redundancy ensures increased data availability and fault tolerance by allowing data from multiple sources to be logically combined under single folder known as "DFS root."

Features of DFS:

DFS offers various features, including:

- Transparency: DFS ensures four types of transparency: structure, naming, access, and replication, allowing seamless access to files without revealing their location or copy status.
- Scalability: Designed to scale rapidly as the number of nodes and users in the distributedsystem increases.
- Data Integrity: Ensures the integrity of transferred data through effective concurrency control methods, providing users with atomic transactions for data integrity.
- High Reliability: Limits the risk of data loss by backing up key files and ensuring systemreliability through stable storage.
- High Availability: Ensures system functionality in case of partial failures, such as node failures, storage device crashes, or link failures.
- Ease of Use: Provides a simple user interface and minimal commands for ease of use inmultiprogramming environments.
- Performance: Assesses performance based on the average time to persuade a client, aimingto perform similarly to a centralized file system.

Distributed File System Replication (DFSR):

By just duplicating the modified portions of files, DFSR improves file replication in DFS while lowering network traffic through data compression. Additionally, it provides adaptable configuration choices for scheduling network traffic control.

History of DFS:

The DFS server component was initially introduced as an additional feature and later became a standard component of Windows operating systems. Linux kernels and versions of Mac OS X alsoinclude DFS-compatible components.

Working of Distributed File System:

DFS can be implemented in two methods: “standalone DFS namespace” and “domain-based DFS” namespace. “Standalone DFS” does not use Active Directory and permits DFS roots only on local system, while “domain-based DFS” stores configuration in Active Directory.

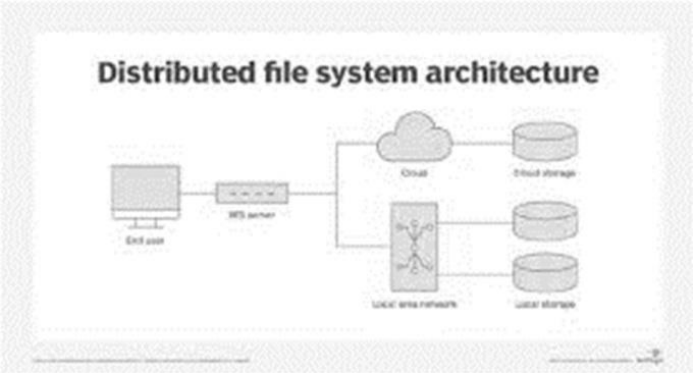


Figure 6.5: Distributed File System Architecture

Applications of DFS:

DFS finds applications in various systems, including Hadoop for distributed storage and management of large data sets, NFS for remote file access, SMB for file sharing, NetWare for computer network operating systems, and CIFS for Internet file systems.

Advantages and Disadvantages of DFS:

DFS offers advantages such as improved data access, network efficiency, and remote data sharing. However, it also poses challenges such as complicated database connections and handling, potential overloading during simultaneous data transfers, and the risk of missed messages and datain the network.

Understanding the features, components, and workings of DFS is crucial for effectively managing distributed file systems and optimizing data access, reliability, and performance in network environments.

6.7 Remote File Access

Remote file access refers to the ability to access and manipulate files stored on remote servers or storage devices over a network. Users and applications can interact with files as if they were storedlocally on their own systems. Remote file access facilitates collaboration, data sharing, and centralized file management across distributed environments.

Example: In a cloud-based storage system, users can access their files stored on remote servers via the internet using file-sharing protocols such as SMB (Server Message Block) or NFS

(Network File System). They can upload, download, modify, and delete files as needed, with changes reflected across all access points.

Types of Remote File Access:

- Read-Only Access: Users can only view or download files but cannot modify or delete them. This access mode is suitable for sharing documents or resources that should remain unchanged.
- Read-Write Access: Users can view, modify, and delete files, allowing for collaborative editing and updating of shared documents.
- Read-Write-Execute Access: Users have full control over files, including the ability to execute programs or scripts stored remotely. This access mode is common in application deployment scenarios.

6.8 File Replication

File replication involves creating multiple copies of files and distributing them across different storage devices or servers in a distributed file system. Replication improves data availability, fault tolerance, and performance by ensuring redundancy and enabling efficient access to files from multiple locations.

Example: In a distributed file system, files are replicated across multiple servers to prevent data loss in case of hardware failures or network outages. Users can access files from any replica, enhancing system reliability and availability.

Types of File Replication:

i. Full Replication: All files are replicated entirely across multiple storage devices or servers,ensuring that each replica contains a complete copy of the original file.
ii. Partial Replication: Only specific parts or blocks of files are replicated across replicas,reducing storage overhead while still improving data availability and performance.
iii. Dynamic Replication: Replicas are created or removed dynamically based on changing system conditions, such as workload patterns and storage capacity.
iv. Lazy Replication: Replicas are created on-demand when accessed by clients, reducing storage overhead by creating replicas only when necessary.
v. Eager Replication: Replicas are proactively created in advance of client requests, ensuringthat replicas are readily available to serve requests without delay.
vi. Static Replication: Replicas are manually configured based on predefined policies and remain static unless updated or removed manually.

File replication strategies are chosen based on factors such as system requirements, performance considerations, and resource utilization goals, ensuring optimal data management and access in distributed file systems.

6.9 Distributed synchronization

A distributed system comprises interconnected computers communicating via a high-speed network. Hardware and software components in such systems coordinate through message passing, allowing nodes to share resources. Efficient resource allocation is crucial for preserving resource states and coordinating processes, often resolved through synchronization.

Synchronization in distributed systems relies on clocks, typically adjusted using physical clocksto ensure nodes share a consistent time reference. Nodes exchange their local time, usually based on UTC (Universal Time Coordination), to synchronize. Clock synchronization methods include External and Internal Clock Synchronization.

- External Clock Synchronization: Utilizes an external reference clock for time adjustments across nodes.
- Internal Clock Synchronization: Nodes share their time, collectively adjusting their times.

Clock synchronization algorithms are classified into Centralized and Distributed types.

- Centralized Algorithms: Employ a single-time server as a reference, propagating time to all nodes. Examples include Berkeley Algorithm, Passive Time Server, and Active Time Server. However, centralized approaches are susceptible to single-point failures, impactingsystem reliability and scalability.
- Distributed Algorithms: Do not rely on a central time server. Nodes synchronize by averaging time differences, enhancing scalability and resilience against single-point failures. Examples include Global Averaging Algorithm, Localized Averaging Algorithm, and NTP (Network Time Protocol).

Centralized algorithms face significant drawbacks:

i. Single-Point Failure: Reliance on a single-time server makes the system vulnerable. Failure of the time server disrupts clock synchronization, undermining system reliability.
ii. Scalability Concerns: Centralized solutions strain the time-server, especially in large systems, posing scalability challenges and reducing system efficiency.

Distributed algorithms address these drawbacks by eliminating the need for a centralized time-server. Instead, each node synchronizes independently using real-time receivers. Multiple real-

time clocks, one per node, ensure decentralized clock synchronization, enhancing reliability and scalability in distributed systems.

During concurrent execution, processes often need to access critical sections of shared program segments, which can lead to inconsistencies if not synchronized properly. This phenomenon, known as a race condition, arises when multiple processes execute simultaneously, affecting the values stored in the critical section. Process synchronization is essential to eliminate race conditions and ensure orderly execution within critical sections.

6.10 Mutual exclusion

Mutual exclusion is a vital concept in process synchronization, dictating that only one process should access the critical section at any given time. This principle, coined by Dijkstra, is fundamental to preventing race conditions. Without mutual exclusion, achieving synchronization and preventing inconsistencies in shared resources becomes challenging.

Concurrency Contexts:

Concurrent execution scenarios where mutual exclusion is crucial include:

- "Interrupt handlers"
- "Interleaved, preemptively scheduled processes/threads"
- "Multiprocessor clusters with shared memory"
- "Distributed systems"

Conditions for Mutual Exclusion:

Four criteria define the applicability of mutual exclusion:

1. **Simultaneous Exclusion:** Concurrent processes cannot access shared resources simultaneously within their critical sections.
2. **Process Speed Independence:** Processes' relative speeds should not impact mutual exclusion assumptions.
3. **Non-Blocking Access:** Processes outside a critical section must not hinder others fromaccessing it.
4. **Finite Critical Section Accessibility:** Multiple processes must access critical sections within a finite time frame, avoiding indefinite wait loops.

Implementation Approaches:

Several methods are employed to implement mutual exclusion:

1. **Software Method:** Processes manage mutual exclusion internally, prone to errors and highoverheads.
2. **Hardware Method:** Special instructions facilitate shared resource access, offering speed butincomplete solutions susceptible to deadlock and starvation.
3. **Programming Language Method:** Operating systems or programming languages providesupport for mutual exclusion.

Requirements of Mutual Exclusion:

Key requirements for mutual exclusion include:

1. Single process access to critical sections at any time
2. Software-based implementation on the machine
3. Bounded time duration for processes within critical sections
4. Independence from asynchronous process speeds
5. No process blocking others indefinitely

Illustrative Example:

Consider a linked list where concurrent threads attempt to remove distinct nodes simultaneously. Without mutual exclusion, race conditions may occur, leading to incorrect list states. Mutual exclusion prevents simultaneous updates to ensure list integrity.

Real-world Analogy:

In a supermarket's clothes section, two individuals wish to use a changing room concurrently. Mutual exclusion ensures only one person occupies the room at a time, avoiding conflicts and maintaining orderly usage.

6.11 Concurrency control

Concurrency control mechanisms are essential for ensuring that transactions executed across different nodes in a distributed system adhere to the principles of ACID (Atomicity, Consistency, Isolation, Durability) or BASE (Basically Available, Soft state, Eventually consistent), depending on the database model used. Without proper concurrency control, inconsistencies and data

mix-ups can occur, compromising the integrity of the distributed system.

Transactions in distributed systems are executed in sets, with each set comprising multiple sub-transactions. It is crucial that these sub-transactions across all nodes are executed serially to upholddata integrity. Concurrency control mechanisms facilitate this serial execution.

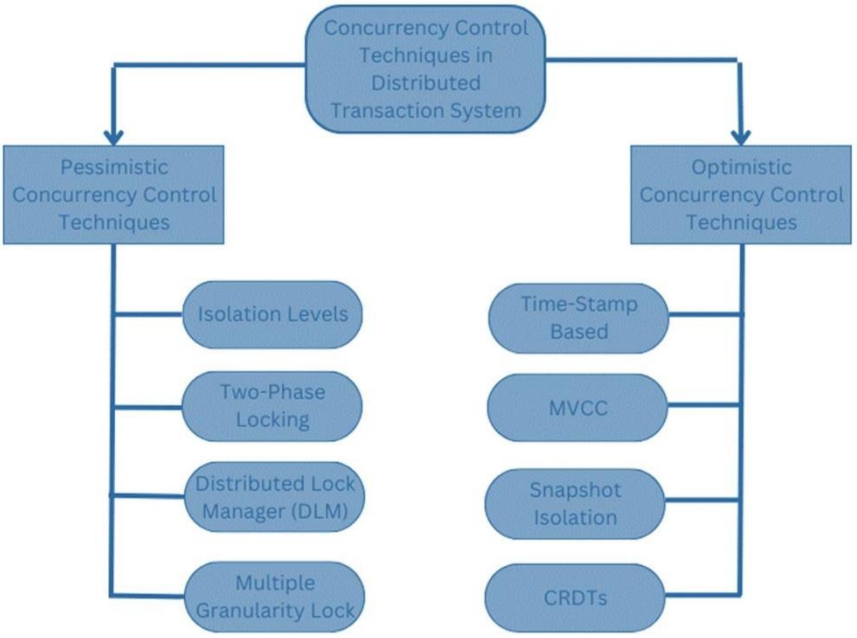


Figure 6.6: Concurrency Control in Distributed System

Types of Concurrency Control Mechanisms:

There are two main types of concurrency control mechanisms:

Pessimistic Concurrency Control (PCC) mechanisms are based on the assumption that multiple transactions may attempt simultaneous access to the same resource. These mechanisms aim to prevent concurrent access to shared resources by employing a locking system. Before any operation is executed, a lock is obtained on the data item to ensure exclusive access and maintain data integrity.

Optimistic Concurrency Control (OCC) techniques operate on the assumption that few or no transactions will concurrently access the same resource. In a fully optimistic system like FULLY OPTIMISTIC, locks are not employed, and conflicts are verified only at the time of commit.

OCC typically involves four operational phases:

1. **"Read Phase":** The transaction reads data while logging the timestamp of the data read for conflict validation during validation phase.
2. **"Execution Phase":** The transaction executes all its operations such as "create", "read", "update",or "delete".

3. "**Validation Phase**": Before committing the transaction, validation check is performed to ensure consistency by comparing the last updated timestamp with one recorded during the "read phase".
4. "**Commit Phase**": During the phase, transactions are either committed or aborted based onvalidation check performed in previous phase.

Pessimistic Concurrency Control Methods:

1. **Isolation Level**: Defines the degree to which data in the database must be isolated by transactions for modification to prevent unwanted inconsistency problems. Isolation levels include Read-Uncommitted, Read-Committed, Repeatable Read, and Serializable.
2. **Two-Phase Locking Protocol:** Involves two phases - Growing Phase, where locks are acquired on data items required for transaction execution, and Shrinking Phase, where acquired locks are released.
3. **Distributed Lock Manager**: Coordinates lock acquiring and releasing operations in distributed transactions to maintain data integrity.

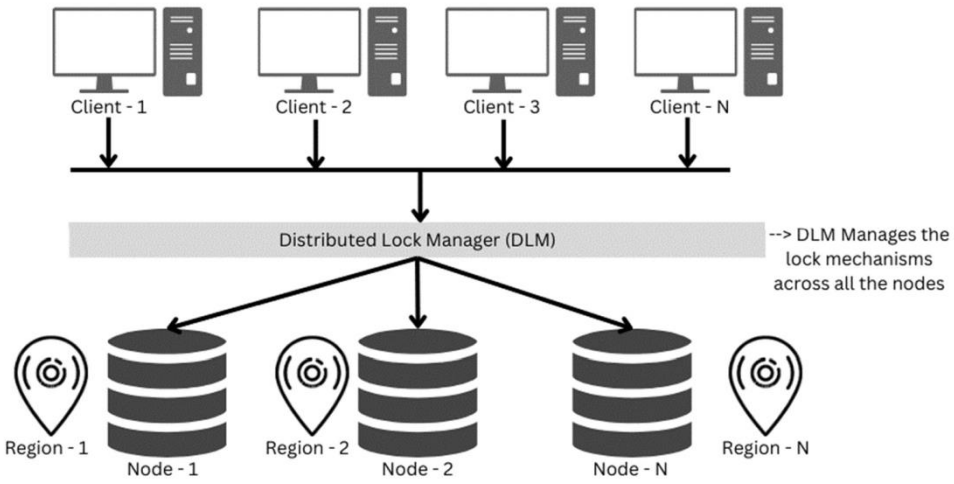


Figure 6.7: Distributed Lock Manager

4. **Multiple Granularity Lock:** Allows locks to be acquired at various granular levels such as "tablelevel, row/record level", or "page level" to manage concurrency effectively.

Optimistic Concurrency Control Methods:

1. **Timestamp Based (OCC):** Each transaction is assigned a unique timestamp, and conflicts are resolved during the commit phase based on timestamps. Transactions may be restarted or aborted if conflicts occur.

2. **Multi-Version Concurrency Control (MVCC)**: Involves creating multiple versions of data items, allowing concurrent operations without blocking each other.
3. **Snapshot Isolation**: Transactions read a snapshot of the database system at the beginning and ensure that data items are not changed while executing operations on them.
4. **Conflict-Free Replicated Data Types (CRDTs):** Data structure technique allowing transactions to perform operations and replicate data across nodes while ensuring conflict-free merging for eventual consistency.

6.12 Deadlock handling

In a distributed system, deadlocks cannot be entirely prevented due to the system's vastness and complexity, but only deadlock detection can be implemented. Deadlock detection techniques in distributed systems must satisfy two key requirements:

1. **"Progress**": Method should detect all deadlocks in system.

2. **"Safety**": The method should not falsely detect deadlocks (phantom deadlocks).

There are three main approaches to detect deadlocks in distributed systems:

1. Centralized Approach: In this approach, a single responsible resource is designated to detect deadlocks. It is simple to implement but has drawbacks such as excessive workload at one node and single-point failure. If this node fails, the entire system crashes, reducing reliability.
2. Distributed Approach: Different nodes collaborate to detect deadlocks in this approach. Workload is evenly distributed among all nodes, eliminating the risk of single-point failure. The speed of deadlock detection also improves compared to the centralized approach.
3. Hierarchical Approach: This approach combines elements of both centralized and distributed approaches. Selected nodes or clusters of nodes are responsible for deadlock detection, but they are controlled by a single node. This approach offers advantages such as improved reliability compared to the centralized approach and faster detection comparedto the distributed approach.

Deadlock Handling Strategies

In the distributed system context, various strategies are employed to handle deadlocks:

1. Deadlock Prevention, Avoidance, and Detection:

a. Deadlock prevention involves ensuring that processes can't enter a state where deadlock ispossible by carefully managing resource allocations.
b. Deadlock avoidance dynamically evaluates resource requests to ensure they won't lead todeadlock.
c. Deadlock detection focuses on identifying existing deadlocks.

2. **Complexity in Distributed Systems:** Handling deadlocks becomes more challenging in distributed systems due to limited knowledge of the system's global state and unpredictable inter-site communication latency.

3. **Deadlock Avoidance Method**: Operating system utilizes deadlock avoidance to determine if system is in safe state by considering resource requests and allocations. Processes must declare their maximum resource needs before execution.

4. **Prevention Techniques**: Deadlock prevention often involves acquiring all necessary resources before execution begins or preempting processes with acquired resources. However, these methodsare inefficient and impractical in distributed systems.

5. **Cyclical Wait Detection:** Detecting deadlocks in distributed systems requires examining the interactions between processes and resources to identify cyclical waits. Issues of Deadlock Detection in tackling deadlock detection in distributed systems, two main issues need addressing:

1. **Detecting Deadlocks**: This involves maintaining the Wait-for Graph (WFG) and searching for cycles within it. In distributed systems, cycles may span multiple sites, requiring a system-wide view of the WFG.

Resolution of Deadlock Detection

Various resolutions for deadlock detection in distributed systems include:

1. Braking Existing Wait-for Dependencies: Deadlock resolution involves breaking wait-for dependencies within the system's WFG.
2. Rolling Deadlocked Processes: Deadlocked processes can be rolled back, and their resourcesreallocated to blocked processes to allow them to resume execution.

Deadlock Detection Algorithms in Distributed Systems

Several deadlock detection algorithms are employed in distributed systems:

1. **“Path-Pushing Algorithms”:**

These algorithms maintain an explicit global WFG and exchange local WFGs between sitesto detect distributed deadlocks.

2. **“Edge-Chasing Algorithms”:**

Edge-chasing methods verify cycles in distributed graph structures by sending specialized messages along edges to cancel cycle formation.

3. **“Diffusing Computations Based Algorithms”:**

Deadlock detection computation is diffused over system's WFG using echo algorithms,with failure leading to deadlock global state detection.

4. “**Global State Detection Based Algorithms”:**

Algorithms based on global state detection utilize consistent snapshots of the distributedsystem to identify deadlocks while preserving stable properties.

- **Knowledge Check 2**

  State True or False

  1. Distributed systems can operate on any one network structure and topologies, including client-server, peer-to-peer, and hybrid architectures. (False)
  2. Distributed file systems enable remote file access, allowing users to access files stored on networked devices from anywhere with appropriate permissions. (True)
  3. File replication is a common technique used in distributed file systems to enhance data availability, reliability, and performance by storing multiple copies of files across different nodes. (True)
  4. Mutual exclusion is a fundamental concept in distributed systems that ensures only one process can access a shared resource at a time to prevent conflicts and maintain data integrity. (True)
  5. Concurrency control mechanisms are employed in distributed systems to manage access to shared resources concurrently accessed by multiple processes, ensuring consistency and preventing data corruption. (True)

- **Outcome-Based Activity 2**

  What skills are gained from studying Section Distributed Systems, covering “network-based OS types”, “structures”, “protocols”, “design issues”, “distributed file systems”, “remote access”, “replication”, “synchronization”, “mutual exclusion”, “concurrency control”, and “deadlock handling” for real- world application?

6.13 Summary

- Goals of protection involve ensuring the security and integrity of computer systems by implementing measures to prevent unauthorized access and data breaches.
- Domains of protection refer to the different areas within a system that require protection, such as user data, system files, and network connections.
- An access matrix is security model that defines access rights for “subjects” (users or processes)to “objects” (files, directories, etc.) based on a matrix-like structure.
- Security problem encompasses various challenges and vulnerabilities that can compromise the security of computer systems, including malware, unauthorized access, and data breaches.
- Threats are potential risks or dangers to the security and integrity of computer systems, including viruses, hackers, and data theft.
- Security tools are software or hardware solutions designed to enhance and maintain the securityof computer systems, such as antivirus programs, firewalls, and encryption tools.
- Classification categorizes security measures and protocols based on their purpose and functionality, helping to organize and prioritize security strategies.
- Types of network-based OS are operating systems specifically designed to support distributed computing, including network-centric, distributed, and cloud-based operating systems.
- Network structures and topologies refer to the layout and arrangement of interconnected devices and nodes in a distributed system, such as client-server, peer-to-peer, and hybrid architectures.
- Communication structures and protocols are essential components of distributed systems, facilitating data exchange and interaction between nodes using standardized communication protocols like TCP/IP and HTTP.
- Design issues in distributed systems encompass challenges and considerations related to the design and implementation of distributed computing environments, including scalability, fault tolerance, and data consistency.
- Remote file access allows users to access files stored on networked devices from remote locations, enabling seamless collaboration and data sharing in distributed systems.
- File replication involves creating and maintaining multiple copies of files across different nodes in a distributed file system to improve data availability and reliability.
- Mutual exclusion ensures that only one process at a time can access a shared resource, preventing conflicts and maintaining data integrity in distributed systems.

- Concurrency control mechanisms manage simultaneous access to shared resources by multiple processes, ensuring consistency and preventing data corruption in distributed systems.
- Deadlock handling techniques identify and resolve deadlock situations where processes are unable to proceed due to resource conflicts, ensuring system stability and performance in distributed systems.

6.14 Self-Assessment Questions

1. What are the primary goals of protection in computer systems, and how do they contribute toensuring system integrity and security?
2. Explain the concept of domains of protection within a computer system and provide examplesof different domains that require protection.
3. Describe the access matrix security model and discuss how it is used to manage access rightsbetween subjects and objects in a computer system.
4. Identify common security problems faced by computer systems and discuss their potentialimpact on system integrity and functionality.
5. Discuss various types of threats that can pose risks to the security of computer systems, andexplain how they can be mitigated using appropriate security measures and tools.

6.15 References / Reference Reading

- "Deitel H.M., Operating Systems, Pearson Education."
- "Stallings William, Operating System, PHI Learning."
- "Godbole A.S., Operating Systems, Tata McGraw-Hill, New Delhi."

Unit 7: Concurrent Process and Semaphores

Learning Outcomes:

- Students will be able to understand concurrent processes and their management challenges.
- Students will be able to recognize the critical section problem and its importance in resourcesynchronization.
- Students will be able to describe semaphores and their role in coordinating shared resourceaccess.
- Students will be able to analyze classical process coordination problems and solutions.
- Students will be able to evaluate deadlocks, including characterization, prevention, and recovery mechanisms.

Structure:

7.1 Concurrent Processes

Concurrent processing in computing employs multiple processors simultaneously to execute instructions concurrently, aiming to enhance efficiency. The term "concurrent" denotes activities occurring in tandem with other events. It involves dividing tasks into subtypes that are processed concurrently by several processors, rather than sequentially by a single processor. While "parallel" and "concurrent" processing are sometimes used interchangeably, in concurrent computing, "real and virtual concurrency" specifically denote:

Multiprogramming Environment: In a multiprogramming environment, a single processor handles multiple tasks concurrently. While the operating system can create a virtual illusion by allocating a processor to each task, the true virtualization occurs when each task is assigned its own processor. The diagram illustrates this layered ecosystem.

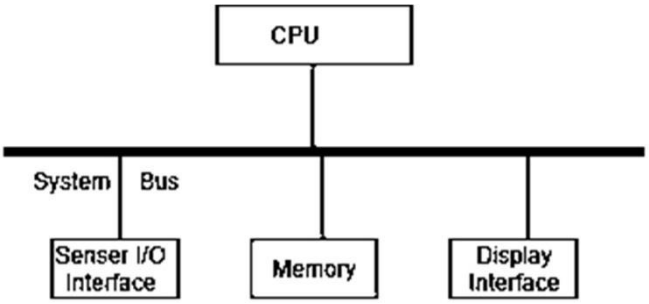


Figure 7.1: Multiprogramming (Single CPU) Environment

Multiprocessing Environment: In a multiprocessing environment, multiple CPUs utilize shared memory and a unified virtual address space. Each CPU accesses the same virtual address space, facilitating shared memory among all processors. Jobs are stored in this shared memory, and concurrent execution by multiple processors enables support for concurrency in this environment. The diagram illustrates this multiprocessing setup.

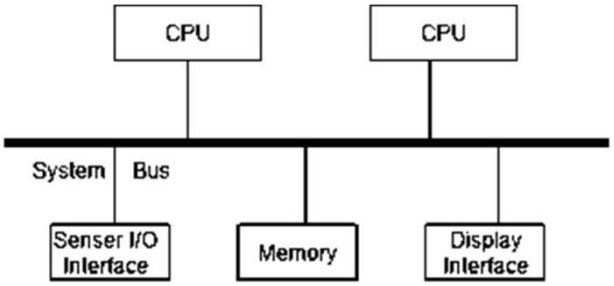


Figure 7.2: Multiprocessing Environment

Distributed Processing Environment: In "distributed processing environment", two or more computers are interconnected via a communication network or high-speed bus. Each computer has its own local memory, and there is no shared memory between the CPUs of different computers. Therefore, a distributed program consists of multiple concurrent tasks or jobs that communicate through messages sent across the network. The diagram illustrates this distributed processing environment.

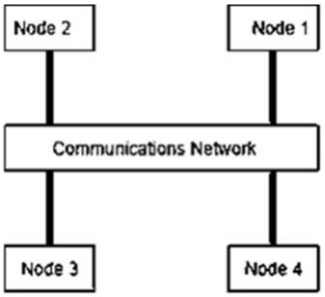


Figure 7.3: Distributed Processing Environment

7.2 Critical section problem

The critical section of a program refers to a segment where shared resources are accessed, and simultaneous execution can lead to conflicts or inconsistencies. Operating systems must provide synchronization mechanisms such as locks and semaphores to enforce mutual exclusion within these critical sections. These mechanisms prevent concurrent processes from interfering with each other, thereby maintaining the integrity of shared resources.

Understanding the Critical Section Problem in OS:

A race condition may arise when several processes access or alter a shared resource simultaneously, with the last process to finish determining the resource's final value.

Consideran example involving two processes, p1 and p2, with a shared variable "value" initially set to 3. If process p1 increments "value" by 3 and process p2 decrements it by 3, the expected value of "value" should be 6. However, due to concurrent execution, if process p2 interrupts p1 before its completion, "value" reverts to 3, indicating a synchronization issue.

Solutions to the Critical Section Problem:

To effectively resolve the Critical Section Problem, solutions must meet three fundamental requirements:

- **“Mutual Exclusion”:** Ensures that only one process can execute within its critical section at any given time, thereby preventing conflicts and data corruption.
- **“Progress”:** Enables processes that are waiting to enter their critical sections to eventually do so, preventing indefinite blocking and ensuring that execution can move forward.
- **“Bounded Waiting”:** Imposes limit on number of times process can execute in its critical section before another process's request is granted, ensuring fairness.

Various solutions exist to meet these requirements, primarily employing software-based locks for synchronization. Examples include Test-and-Set, Compare-and-Swap, Mutex Locks, Semaphores, and Condition Variables. These solutions guarantee exclusive access to critical sections while facilitating progress and preventing indefinite waiting.

Strategies for Avoiding Problems:

To manage critical sections effectively and avoid conflicts, several strategies can be employed:

1. **Fine-Grained Locking:** Break down resources into smaller units and apply locks selectively to improve concurrency.
2. **Lock Hierarchies:** Establish a specific order for lock acquisition to prevent deadlocks.

3. **Read-Write Locks:** Differentiate between read and write operations to allow concurrent reading and exclusive writing.
4. **Optimistic Concurrency Control (OCC):** Enable multiple processes to read data withoutlocks, verifying modifications before writes.
5. **Lock-Free and Wait-Free Data Structures:** Design data structures to operate withouttraditional locks, ensuring progress in concurrent access scenarios.

Impact of Critical Sections on Scalability:

Improper or excessive use of critical sections can impact system scalability by creating bottlenecks, reducing parallelism, and introducing locking overhead. While critical sections ensure data integrity, they also introduce contention among processes, affecting system scalability.

Advantages and Disadvantages of Critical Sections:

Critical sections offer advantages such as data integrity, simplicity, predictability, and compatibility with legacy code. However, they also pose challenges like potential deadlocks, reduced concurrency, the overhead of locking and unlocking, and complexity in debugging.

7.3 Semaphores

Semaphores are fundamental synchronization mechanisms used in operating systems to coordinate access to shared resources among concurrent processes or threads. They were introduced by Dutch computer scientist Edsger W. Dijkstra in the late 1960s as a solution to the critical section problem.

Definition of Semaphores:

A semaphore functions as a variable or abstract data type used for signaling to regulate access to shared resources. It holds a non-negative integer value and facilitates two key operations: wait (P) and signal (V).

Operations on Semaphores:

1. “Wait (P) Operation”:

- Decrements value of semaphore by 1.
- If semaphore's value becomes negative, process executing wait operation is blocked, and its execution is suspended until semaphore's value becomes non-negative.

2. “Signal (V) Operation”:

- Increments the value of semaphore by 1.
- If there are processes blocked on semaphore due to previous wait operations, one of them is unblocked, allowing it to proceed.

Types of Semaphores:

1. Binary Semaphore:

- Also known as mutex (mutual exclusion) semaphore.
- Can only take on the values 0 and 1.
- Used for binary synchronization, such as implementing mutual exclusion or signallingbetween two processes.

2. Counting Semaphore:

- Can take on integer values greater than or equal to 0.
- Used for counting and resource allocation, allowing multiple processes to access a finitenumber of resources.

Application of Semaphores:

1. Mutual Exclusion:

- Semaphores are commonly used to implement mutual exclusion, ensuring that only one process accesses a critical section at a time.
- Binary semaphores are used for this purpose, with their value indicating whether the criticalsection is currently occupied (1) or available (0).

Advantages of Semaphores:

1. Flexibility:

- Semaphores provide a flexible synchronization mechanism suitable for a wide range of concurrency control scenarios.
- They can be used to implement various synchronization patterns, including mutual exclusion, resource allocation, and coordination of producer-consumer and reader-writer scenarios.

2. Efficiency:

- Semaphores are efficient synchronization primitives, typically implemented using hardware-supported atomic operations or efficient software algorithms.
- They minimize overhead and ensure effective resource utilization in concurrent systems.

Disadvantages of Semaphores:

1. **Complexity:**
   - Semaphores require careful programming and an understanding of concurrency control concepts to avoid issues such as deadlock and starvation.
   - Incorrect usage of semaphores can lead to subtle synchronization bugs that are challenging to diagnose and resolve.
2. **Potential for Deadlock:**
   - Improper synchronization using semaphores can lead to deadlock situations where processes are unable to proceed due to circular dependencies on resources.

7.4 Classical process co-ordination problems and their solutions

Classical process coordination problems are fundamental challenges in operating systems and concurrent programming. These problems involve multiple processes or threads that must coordinate their activities to achieve specific objectives while avoiding issues such as race conditions, deadlocks, and starvation. Several well-known problems illustrate these coordination challenges, each with its own set of requirements and solutions.

1. Producer-Consumer Problem:

The Producer-Consumer Problem is classic synchronization problem often used to illustrate the challenges of coordinating access to a shared, bounded buffer between two types of processes: producers and consumers. It highlights the need for synchronization mechanisms like semaphores to ensure data integrity and prevent issues such as race conditions, deadlocks, and buffer overflows.

Problem Statement:

In the Producer-Consumer Problem, there are two types of processes:

1. **"Producers**": These processes generate data items and add them to shared buffer.

2. **“Consumers”:** These processes retrieve data items from the shared buffer and consume them. The shared buffer has a finite size, meaning it can only hold a limited number of data items at any given time. The challenge is to ensure that producers do not add items to the buffer when it is full and that consumers do not attempt to retrieve items from an empty buffer.

Key Requirements:

1. Mutual Exclusion: Only one process (either a producer or a consumer) should be allowedto access the shared buffer at a time to prevent data corruption or inconsistencies.
2. Empty and Full Buffer Conditions: Producers should be blocked when the buffer is full,and consumers should be blocked when the buffer is empty.
3. Buffer Size Limit: The buffer should not overflow or underflow, meaning it should not exceed its maximum capacity or become empty.

Solution Using Semaphores:

1. Semaphore Initialization:

- Two semaphores are typically used: one for controlling access to the buffer (mutex semaphore) and another for tracking the number of empty slots in the buffer (empty semaphore).

2. Producer Process:

- The producer waits on the empty semaphore to check if there is space available in the buffer.
- Once space is available, the producer waits on mutex semaphore to enter criticalsection (accessing buffer).
- The producer adds the produced item to buffer, updates the buffer state, and releasesthe mutex semaphore.
- Finally, the producer signals the full semaphore to indicate that a new item is added to thebuffer.

3. Consumer Process:

- The consumer waits on the full semaphore to check if there are items available in the buffer.
- Once an item is available, the consumer waits on mutex semaphore to enter criticalsection (accessing buffer).
- The consumer retrieves the consumed item from buffer, updates the buffer state, and releases the mutex semaphore.

- Finally, the consumer signals the empty semaphore to indicate that a slot in the buffer isnow available.

Synchronization Challenges:

1. Deadlocks: Deadlocks can occur if producers and consumers wait indefinitely for resources(e.g., waiting for empty or full slots in the buffer) without releasing them.
2. Starvation: Starvation can occur if one type of process (either producers or consumers) continuously acquires resources, leading to other processes being blocked indefinitely.
3. Race Conditions: Race conditions can occur if multiple processes attempt to access shared resources simultaneously without proper synchronization, leading to unpredictable behaviour.

2. Reader-Writer Problem:

The Reader-Writer Problem is another classic synchronization problem in computer science and operating systems, focusing on coordinating access to a shared resource between multiple readers and writers. It demonstrates the challenges of managing concurrent access to shared data while ensuring data consistency and performance. The problem highlights the need for synchronization mechanisms like semaphores or locks to address issues such as data corruption, race conditions, and starvation.

Problem Statement:

In the Reader-Writer Problem, multiple processes or threads can be divided into two categories:

1. Readers: These processes only read data from the shared resource but do not modify it.

2. Writers: These processes both read and modify the shared resource.

The challenge is to devise a solution that allows multiple readers to access the shared resource concurrently while ensuring exclusive access for writers. Additionally, the solution should prioritize readers to prevent writers from being starved, thereby maximizing system throughput.

Key Requirements:

1. Readers-Writers Mutual Exclusion: Writers should have exclusive access to the sharedresource while they are writing to prevent data inconsistency.
2. Readers Priority: Readers should be allowed to access the shared resource concurrently tomaximize throughput and efficiency.
3. Writer Starvation Avoidance: Writers should not be starved, meaning they should have afair chance to access the shared resource without being indefinitely blocked by readers.

Solution Strategies:

Several strategies can be employed to address the Reader-Writer Problem:

1. Reader-Writer Locks (RWLocks):

- RWLocks are synchronization primitives specifically designed for managing access toshared resources by multiple readers and writers.
- RWLocks maintain separate counters for readers and writers and use mechanisms likemutexes or semaphores to provide mutual exclusion between readers and writers.

2. Read-Preferring RWLocks:

- These locks prioritize readers over writers to prevent writers from being starved.
- When readers are accessing the resource, writers are blocked until all readers have finished.
- However, once a writer requests access, it is granted exclusive access, and subsequentreaders are blocked until the writer has finished.

3. Writer-Preferring RWLocks:

- These locks prioritize writers over readers to ensure data consistency and prevent readersfrom accessing the resource while it is being modified by a writer.
- When a writer requests access, it is granted exclusive access immediately, and subsequentreaders and writers are blocked until the writer has finished.

4. Semaphore-Based Solutions:

- Semaphores can also be used to implement solutions to the Reader-Writer Problem.
- Separate semaphores can be used to control access for readers and writers, with additionallogic to prioritize readers or writers based on the chosen strategy.

Synchronization Challenges:

1. **Writer Starvation**: If readers continuously access the resource, writers may be starved, leadingto degraded system performance or potential deadlock situations.

2. **Data Consistency:** Ensuring data consistency is critical, especially when allowing concurrent access by readers. Writers must have exclusive access to prevent data corruption.

3. Dining Philosopher's Problem

The Dining Philosophers Problem is a classic synchronization problem in computer science and concurrent programming, first introduced by Edsger Dijkstra in 1965. It illustrates the challenges of resource allocation and deadlock avoidance in a concurrent system where multiple processes

(Philosophers) contend for a finite set of resources (forks) to perform their tasks (eating).

Problem Statement:

The problem involves a dining table with a circular arrangement of philosophers, each of whom alternates between thinking and eating. There are five philosophers and five forks, with each philosopher requiring two forks to eat. The challenge is to devise a solution that prevents deadlocks, where each philosopher holds one fork and waits indefinitely for the second fork held by their neighbour, leading to a situation where no philosopher can proceed.

Key Requirements:

1. Mutual Exclusion: Philosophers must acquire exclusive access to the forks to avoid conflicts and ensure that only one philosopher can use a fork at a time.
2. Deadlock Avoidance: The solution must prevent situations where all philosophers hold one fork and are waiting for the second fork held by their neighbour, resulting in a deadlock.
3. Resource Utilization: The solution should maximize the utilization of resources (forks) by allowing as many philosophers as possible to eat concurrently without violating mutual exclusion or causing deadlocks.

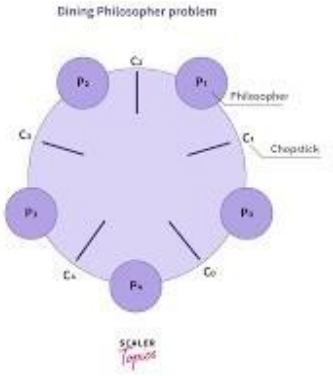


Figure 7.4: Dining Philosophers Problem

Solution Strategies:

Several strategies can be employed to solve the Dining Philosopher's Problem:

1. Chopstick Locks:

- Each fork (chopstick) is represented by a semaphore or mutex lock.
- Philosophers must acquire both the left and right forks (locks) to eat and release themafterwards.

- This solution ensures mutual exclusion and prevents deadlock by allowing philosophers towait for the availability of both forks before proceeding.

2. Asymmetric Fork Acquisition:

- To prevent deadlock, one philosopher can be designated to acquire the forks in a differentorder (e.g., always pick up the left fork first).
- This breaks the symmetry in fork acquisition and eliminates the potential for deadlock caused by circular wait conditions.

3. Resource Pooling:

- Introduce a global pool of resources (forks) managed by a semaphore or mutex lock.
- Philosophers request access to the pool and release resources when done, ensuring fairresource allocation and preventing deadlock by avoiding circular wait conditions.

Challenges:

1. Deadlock: The main challenge in solving the Dining Philosophers Problem is avoiding deadlock, where each philosopher holds one fork and waits indefinitely for the second forkheld by their neighbour, leading to a situation where no philosopher can proceed.
2. Resource Utilization: Another challenge is maximizing the utilization of resources (forks) by allowing as many philosophers as possible to eat concurrently without violating mutual exclusion or causing deadlocks.

7.5 Deadlock Prevention and Avoidance:

- **Resource Allocation Strategies**: Resource allocation strategies are crucial in preventing deadlocks by ensuring that the system remains in a deadlock-free state. These strategies involve carefully managing the allocation of resources to processes to avoid situations where processes are indefinitely waiting for resources held by others, thereby preventing the occurrence of “deadlocks”.
- **“Banker's Algorithm”:** “The Banker's Algorithm” is resource allocation and deadlock avoidance algorithm used in operating systems to ensure that the system remains in a safe state, where deadlock cannot occur. It was developed by E.W. Dijkstra in 1965 and is widely implemented in various operating systems to manage resource allocation among competing processes.

Principles:

  - Resource Allocation: The Banker's Algorithm considers the total number of

available resources in the system, as well as the maximum resources each process may need to completeits execution. Resources can include various system resources such as CPU cycles, memory, and input/output devices.

- Safety Criteria: The algorithm ensures that system remains in safe state, meaning that there is at least one sequence of resource allocations that allows all processes to complete their execution without encountering a deadlock. A deadlock occurs when processes are stuck in a circular dependency, waiting for resources held by other processes.

Operation:

1. Initialization:

- The system initializes several data structures to keep track of resource allocation and availability.
- It maintains a matrix `max[n][m]`, where `n` represents the number of processes and `m' represents the number of resource types. This matrix indicates the maximum resourceseach process may need to complete its execution.
- It also maintains matrices `allocation[n][m]` and `available[m]`, where `allocation[i][j]` represents the number of resources of type `j` allocated to process ` i', and `available[j]` represents the number of available resources of type `j` in the system.

2. Resource Request:

- When a process requests additional resources, the Banker's Algorithm checks if grantingthe request will lead to a safe state by simulating the allocation of resources.
- It analyzes whether the system has enough available resources to satisfy the request withoutrisking deadlock.

3. Granting Resources:

- If granting the resource request maintains system safety, the resources are granted to theprocess.
- Otherwise, the process must wait until it can safely allocate resources without risking deadlock.

Example:

Consider a simplified example with three processes (P1, P2, P3) and three resource types (A, B, C). The following matrices represent the current state of the system:

- Max Resources (“max”):

	A	B	C
P1	7	5	3
P2	3	2	2
P3	9	0	2

- Allocated Resources (“allocation”):

	A	B	C
P1	0	1	0
P2	2	0	0
P3	3	0	2

- Available Resources (“available”):

A	B	C
3	3	2

Now, suppose Process “P2” requests one additional unit of resource “B”. The “Banker's Algorithm”checks if granting this request will lead to a safe state:

1. Check Safety:

- If resource B is allocated to P2, the available resources become `(3, 2, 1)`.
- Then, the system can satisfy the maximum resource needs of P1 and P3, leaving `(0, 2, 1)`resources available.
- Since there is still a safe sequence `(P1, P3)` to complete execution, the request for oneunit of resource B by P2 can be granted.

2. Grant Resources:

- The system grants one unit of resource B to Process P2, updating the allocation matrixaccordingly.

This example demonstrates how Banker's Algorithm ensures that resource requests are granted in a manner that prevents deadlock formation and maintains system safety. By carefully analyzing resource allocation and availability, the algorithm effectively manages resource allocation in operating systems, enhancing system reliability and stability.

 Dynamic Resource Allocation:

Dynamic resource allocation is a strategy used in operating systems to efficiently manage resources among competing processes in real time. Unlike static resource allocation, which predefines resource allocations, dynamic allocation adjusts resource distribution based on the immediate needs of processes and the current state of the system. This approach aims to optimize resource utilization while ensuring fair and efficient allocation, thereby enhancing system performance and responsiveness.

Principles:

1. Real-Time Resource Allocation: Dynamic resource allocation involves allocating resources to processes in real time based on their current demands and the availability of resources in the system. This allows for flexible resource management, adapting to changing process requirements and system conditions dynamically.
2. Resource Utilization: The primary goal of dynamic resource allocation is to maximize resource utilization by efficiently distributing resources among processes. By dynamically adjusting resource allocations based on process demands and system availability, the system can utilize resources more effectively, preventing underutilization or overutilizationof resources.
3. System Monitoring: Continuous monitoring of system resources and process states is essential for dynamic resource allocation. System components such as the resource manager or scheduler continuously monitor resource usage, process requests, and systemconditions to make informed decisions regarding resource allocation.

Operation:

1. Resource Monitoring:

- The system continuously monitors the usage of system resources, including CPU, memory,I/O devices, and others.
- Resource managers collect data on resource utilization, availability, and process requirements to inform resource allocation decisions.

2. Dynamic Adjustment:

- Based on real-time resource demands and availability, the system dynamically adjusts resource allocations to processes.
- Resource managers analyze process requests and system conditions to determine the optimal allocation of resources to maximize resource utilization while avoiding resource contention or over-commitment.

3. Preventative Measures:

- Proactive measures are taken to prevent processes from entering deadlock-prone states or resource contention situations.
- Dynamic resource allocation algorithms prioritize resource allocations to prevent processesfrom waiting indefinitely for resources, thereby minimizing the risk of deadlock formation.

Advantages:

1. **Optimized Resource Utilization**: Dynamic resource allocation optimizes resource utilization by adapting resource allocations to match process demands and system conditions in real-time. This ensures that resources are efficiently utilized, maximizing system performance and responsiveness.

- Adaptability: The flexibility of dynamic resource allocation allows the system to adapt to changing workload patterns and resource demands. Resources can be dynamically reallocated to meet shifting priorities and optimize system performance.
- Deadlock Prevention: By dynamically adjusting resource allocations based on process demands and system conditions, dynamic resource allocation helps prevent processes from entering deadlock-prone states. Proactive resource management minimizes the risk ofresource contention and deadlock formation.

2. Deadlock Avoidance:

- Resource Scheduling Algorithms: Use resource scheduling algorithms to dynamically analyze resource requests and allocations to prevent deadlock formation.
- Safe State Detection: Determine safe states where deadlock cannot occur and scheduleresource allocations accordingly to avoid deadlock.

7.6 Self-Assessment Questions:

1. Explain what concurrent processes are and provide an example of their importance in modern computing.
2. Define the critical section problem and discuss why it is essential to manage critical sections in concurrent programming.
3. Describe how semaphores are used to synchronize access to shared resources among concurrent processes.
4. Identify and explain one classical process coordination problem and discuss a viable solution to mitigate it.

Unit 8: Deadlocks

Learning Outcomes:

- Students will be able to understand the Monitors and Deadlocks.
- Students will be able to identify Deadlock Characterization.
- Students will be able to demonstrate the Deadlock handling.
- Students will be able to analyse the Deadlock detection and recovery.

Structure:

8.1 Monitors

Monitors are a synchronization construct introduced by C.A.R. Hoare in 1974 as a high-level abstraction for managing shared resources in concurrent programming. Monitors encapsulate shared data and the procedures (methods) that operate on them, providing a structured approach to concurrent programming while ensuring mutual exclusion and data integrity.

Key Concepts:

1. Encapsulation: Monitors encapsulate shared data and procedures (methods) that operate on them. This encapsulation hides the implementation details of data and synchronization mechanisms, allowing safe concurrent access to shared resources.
2. Condition Variables: Monitors include condition variables, which are used to manage the execution of threads waiting for specific conditions to be satisfied. Threads can wait on condition variables until signaled by other threads when the desired condition becomes true.
3. Mutual Exclusion: Monitors provide built-in mutual exclusion mechanisms to ensure that only one thread can execute a monitor procedure (method) at a time. This prevents concurrent access to shared resources, avoiding race conditions and data corruption.

Operations on Monitors:

1. Initialization: Monitors are initialized before use, typically by creating an instance of the monitor data structure and initializing any shared data and condition variables.
2. Procedure Invocation: Threads invoke monitor procedures (methods) to access shared resources. Only one thread can execute a monitor procedure at a time, ensuring mutual exclusion.
3. Conditional Wait: Threads can wait on condition variables inside monitor procedures, suspending their execution until signaled by other threads. This allows threads to coordinate their activities based on specific conditions.
4. Signal and Broadcast: Threads can signal condition variables to wake up waiting threads when certain conditions are met. The signal operation wakes up one waiting thread, while the broadcast operation wakes up all waiting threads.

Synchronization Mechanisms:

1. Mutex Locks: Monitors use mutex locks internally to achieve mutual exclusion. When a thread enters a monitor procedure, it automatically acquires the mutex lock associated with the monitor, preventing other threads from entering the monitor concurrently.
2. Condition Variables: Condition variables are used to coordinate thread execution inside monitors. Threads wait on condition variables using atomic wait operations and other threads signal or broadcast to wake up waiting threads when necessary.

Benefits of Monitors:

1. Simplicity: Monitors provide a high-level abstraction for managing shared resources, hiding the complexity of low-level synchronization mechanisms such as mutex locks and condition variables.
2. Encapsulation: Monitors encapsulate shared data and synchronization mechanisms, promoting modular and structured concurrent programming.
3. Ease of Use: Monitors simplify the development of concurrent programs by providing a clear and structured approach to synchronization, reducing the likelihood of synchronization bugs and race conditions.

Limitations of Monitors:

1. Blocking Operations: Threads waiting on condition variables inside monitors are blocked until signalled, which may lead to potential performance issues in scenarios where threads need to wait for prolonged periods.
2. Limited Expressiveness: Monitors have limited expressiveness compared to other synchronization constructs like semaphores, which can be used to implement a wider range of synchronization patterns and algorithms.

- **Knowledge Check1 Fill in the Blanks**

1. In the critical section problem, a process is not allowed to be executed by more than _____Process (es) simultaneously. (**one**/Two)
2. Semaphores are synchronization primitives used to coordinate access to_____ resources.
3. **(shared**/ exclusive)
4. One of the key requirements of classical process coordination problems is ensuring between processes. (**mutual exclusion**/race conditions)
5. Monitors encapsulate shared data and procedures, providing a structured approach to concurrent programming while ensuring and data integrity. (**mutual exclusion**/resource utilization)
6. In the dining philosophers problem, each philosopher alternates between and eating.
7. (thinking/sleeping)

- **Outcome-BasedActivity1**

  Explain how Semaphores are synchronization constructs used to control access to shared resources in concurrent systems. In the context of traffic management at an intersection, semaphores can simulate the operation of traffic lights effectively.

8.2 Deadlocks

In computing, processes often require access to various resources to complete their tasks. However, resource allocation typically occurs sequentially, where processes request resources, utilize them, and then release them upon completion.
Here's a simplified breakdown of this process:

1. Resource Request: When a process requires a resource, it sends a request to the operating system (OS) to allocate the resource.
2. Resource Allocation: If the requested resource is available, the OS grants it to the process. If the resource is unavailable (due to being held by another process), the requesting process is placed in a waiting state until the resource becomes available.
3. Resource Utilization: The process uses the allocated resources to carry out its tasks.
4. Resource Release: Upon completing its tasks, the process releases the resource back to the OS, making it available for other processes.

Deadlock Scenario :

A deadlock occurs when multiple processes are waiting for resources that are held by other processes, creating a cycle of dependencies that prevents any process from progressing. Consider the following scenario involving three processes (P1,P2,P3) and three resources (R1, R2, R3):

1. Initially, R1 is allocated to P1, R2to P2, and R3toP3.
2. At a later point, P1 requests R1, which is currently held by P2, causing P1 to wait for R1.
3. Similarly, P2 requestsR3, held by P3, leading toP2waiting for R3.
4. Additionally, P3 requests R1, held by P1, resulting in P3waiting for R1.

This situation forms a cycle among the processes, where each process is waiting for a resource held by another process. Consequently, none of the processes can progress further, and the system becomes unresponsive as all processes are blocked.

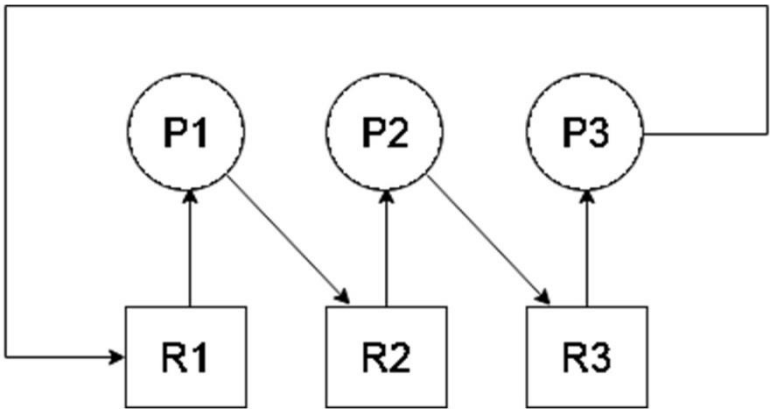


Figure8.1: Deadlock

8.3 Deadlock characterization

Deadlock is a critical issue in operating systems and concurrent programming, where processes become stuck in a state of waiting for resources that are held by other processes, ultimately leading to a system-wide halt. Characterizing deadlock involves identifying the conditionsunder which deadlocks can occur and understanding their implications for system design and management.

Conditions for Deadlocks:

1. Mutual Exclusion: Resources can only be accessed in a mutually exclusive manner, meaning that if one process is using a resource, it cannot be simultaneously accessed by another process. This condition ensures that conflicts and inconsistencies do not arise from concurrent access to shared resources.
2. Hold and Wait: Processes may hold on to resources while waiting to acquire additional resources. This scenario can lead to deadlock if each process is waiting for a resource held by another process, creating a situation where no process can proceed until the resource it needs is released by another process.
3. No Preemption: Once a process acquires a resource, it retains control of that resource until it completes its execution. The operating system does not forcibly preempt resources from processes, meaning that a process cannot be interrupted or have its resources forcibly taken away by the scheduler while it is still executing.
4. Circular Wait: Deadlocks occur when processes form a circular chain of dependencies, with each process waiting for a resource held by another process in the chain. This circular dependency prevents any process from releasing its resources and perpetuates the deadlock situation.

8.4 Deadlock Handling

Deadlock handling is a critical aspect of operating system design aimed at preventing, detecting, and recovering from deadlocks to ensure system reliability and performance. Various strategies and techniques are employed to address deadlocks and mitigate their impact on system operation. Strategies for Deadlock Handling are Deadlock Prevention, Dynamic Resource Allocation, and Deadlock avoidance.

8.5 Deadlock detection and recovery

- Resource-Allocation Graph: Employ a resource-allocation graph to detect deadlocks by identifying cycles in the graph, indicating potential deadlock situations.
- Deadlock Detection Algorithms: Implement algorithms to periodically check for deadlocks in the system by analyzing resource allocation and process states.
- Deadlock Recovery Mechanisms: Upon detecting a deadlock, employ recovery mechanisms such as process termination, resource preemption, or rollback of process states to resolve the deadlock and restore system operation.

Implementation of Deadlock Handling:

1. System Design Considerations:

   - Incorporate deadlock handling mechanisms into the design of the operating system, including resource management modules and scheduling algorithms.
   - Ensure that deadlock prevention, avoidance, detection, and recovery mechanisms are integrated seamlessly into the overall system architecture.

2. Resource Management:

   - Implement resource allocation policies and algorithms that prioritize preventingdeadlock formation while maximizing resource utilization.
   - Define clear rules for resource acquisition and release to minimize the likelihood of deadlocks.

3. Process and Thread Management:

   - Manage processes and threads efficiently to minimize contention for shared resources.
   - Implement mechanisms for process termination, priority-based scheduling, andresource preemption to mitigate deadlock risks.

Impact of Deadlock Handling:

1. System Reliability and Performance:

- Effective deadlock handling mechanisms contribute to system reliability by preventingand resolving deadlocks promptly.
- Proper deadlock management ensures optimal system performance by minimizing the impact of deadlocks on system operation and resource utilization.

2. Complexity and Overhead:

- Deadlock handling adds complexity and overhead to the system design and operation.
- Implementing sophisticated deadlock prevention, detection, and recovery mechanisms requires careful consideration of system resources and performance implications.

- **Knowledge Check 2**

  State True or False

  1. Deadlocks occur when processes wait indefinitely for resources held by otherprocesses. (True)
  2. Deadlock characterization identifies conditions like mutual exclusion and hold-and-wait leading to deadlock. (True)
  3. Deadlock handling includes prevention, detection, and recovery strategies for system reliability. (True)
  4. Deadlock prevention eliminates conditions causing deadlock, while avoidance dynamically manages resources. (True)
  5. Deadlock detection identifies deadlocks, and recovery resolves them via measures like process termination. (True)

- **Outcome-Based Activity 2**

Describe a scenario where deadlock characterization principles can be applied. Identify andexplain the necessary conditions for deadlock to occur in this scenario and propose potential deadlock handling strategies to mitigate the risk of deadlock formation.

8.6 Summary

- Concurrency involves the simultaneous execution of multiple processes, which share resources and may encounter synchronization issues.
- The critical section problem addresses the need for mutual exclusion, ensuring that only one process accesses a shared resource at a time, along with considerations for progress and

bounded waiting.

- Semaphores are synchronization primitives used to control access to shared resources by signaling among concurrent processes.
- Classical process coordination problems, such as producer-consumer, readers-writers, and dining philosophers, are solved using techniques like semaphores and monitors.
- Deadlocks occur when processes are stuck in a circular waiting pattern for resources held by each other, leading to system stagnation.
- Deadlock characterization identifies necessary conditions for deadlock occurrence, including mutual exclusion, hold and wait, no preemption, and circular wait.
- Deadlock handling involves prevention, avoidance, detection, and recovery strategies to ensure system reliability.
- Deadlock prevention and avoidance employ strategies and algorithms to eliminate or dynamically manage resource allocation to prevent deadlock formation.
- Deadlock detection and recovery involve identifying the presence of deadlocks and implementing measures like process termination or resource preemption to resolve them.

8.7 Self-Assessment Questions

1. How does the critical section problem arise in concurrent programming, and why is it crucial to address?
2. Discuss the concept of semaphores and their role in managing synchronization among concurrent processes.
3. What are some classical process coordination problems, and what are their solutions in terms of synchronization mechanisms like semaphores or monitors?
4. Explain the concept of deadlocks in concurrent systems and discuss the conditions that contribute to their occurrence.
5. Describe the strategies for handling deadlocks in operating systems, including prevention, avoidance, detection, and recovery mechanisms.

8.8 References

- Silberschatz A., Galvin P. B., Gagne G., Operating System Concepts, Wiley India Pvt. Ltd.
- Chauhan Naresh, Principles of Operating Systems, Oxford University Press.
- Tanenbaum A.S., Operating System- Design and Implementation, PHI Learning.