SGOU
Higher Education for All

OPERATING SYSTEM DESIGN

COURSE CODE: M26CA01DC
Master of Computer Applications
Discipline Core Course
Self Learning Material



SGOU

SREENARAYANAGURU OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY
The State University for Education, Training and Research in Blended Format, Kerala

SREENARAYANAGURU OPEN UNIVERSITY

SGOU

Vision

To increase access of potential learners of all categories to higher education, research and training, and ensure equity through delivery of high quality processes and outcomes fostering inclusive educational empowerment for social advancement.

Mission

To be benchmarked as a model for conservation and dissemination of knowledge and skill on blended and virtual mode in education, training and research for normal, continuing, and adult learners.

Pathway

Access and Quality define Equity.

Operating System Design

Course Code: M26CA01DC

Semester - I

Discipline Core Course
Postgraduate Programme
Master of Computer Applications
Self Learning Material

SGOU



SREENARAYANAGURU
OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

SREENARAYANAGURU
OPEN UNIVERSITY

OPERATING SYSTEM DESIGN

Course Code: M26CA01DC
Semester- I
Discipline Core Course
Master of Computer Applications

Academic Committee

Prof. (Dr.) Sabu M.K.
Dr. G. Santhoshkumar T
Prof. (Dr.) Vinod Chandra S.S.
Dr. Vinod P.
Dr.Lajish V.L.
Sreekanth M.S.
Dr. Vivek P.
Dr. Arun K.S.
Dr. Abdul Jebbar P.

Development of the Content

Shamin S.
Greeshma P.P.
Sreerekha P.K.
Dr. Kanitha V.K.
Subi Priya Laxmi S.B.N.
Aswathy V.S.

Review and Edit

Dr. Sony P.

Scrutiny

Shamin S.
Greeshma P.P.
Sreerekha P.K.
Dr. Kanitha V.K.
Subi Priya Laxmi S.B.N.
Aswathy V.S.

Cover Design

Jobin J.

Co-ordination

Dr. Gopakumar C.
Head, School of ICS

SGOU

Scan this QR Code for reading the SLM on a digital device.

Edition
September 2026

Copyright
© Sreenarayanaguru Open University

All rights reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from Sreenarayanaguru Open University. Printed and published on behalf of Sreenarayanaguru Open University by Registrar, SGOU, Kollam.

www.sgou.ac.in

Visit and Subscribe our Social Media Platforms

MESSAGE FROM VICE CHANCELLOR

Dear Learner,

It gives me immense pleasure and a deep sense of pride to warmly welcome you to Sreenarayanaguru Open University—a vibrant and progressive institution committed to transforming lives through inclusive, flexible, and high-quality education.

Established in September 2020 as a forward-looking initiative of the Government of Kerala, the University stands as a beacon of opportunity for learners seeking to advance their academic and professional aspirations through the open and distance learning mode. Guided by our foundational principle that "access and quality define equity," we are steadfast in our mission to democratize education while upholding uncompromising academic standards.

Our university is inspired by the timeless vision and philosophy of Sree Narayana Guru, whose ideals of knowledge, equality, and social transformation continue to guide our academic journey. His enduring legacy instils in us the responsibility to create an educational environment that empowers individuals, nurtures critical thinking, and contributes meaningfully to society.

Understanding the dynamic needs of contemporary learners, we have adopted a robust and learner-centric blended learning model, seamlessly integrating Self-Learning Materials, Academic Counselling, and Advanced Digital Learning Platforms. This holistic approach ensures flexibility without sacrificing academic depth, enabling you to learn at your own pace while remaining meaningfully connected to a vibrant academic ecosystem.

The Master of Computer Applications (MCA) programme you are embarking upon is carefully designed to position you at the forefront of the digital revolution. It uniquely blends strong theoretical foundations with practical, industry-oriented competencies. The curriculum emphasizes algorithmic thinking, system design, programming, database management, networking, and emerging technologies such as artificial intelligence, data science, and cloud computing.
What sets this programme apart is its forward-thinking design, which offers:

- Opportunities for skill enhancement aligned with industry needs
- Multidisciplinary learning pathways for broader intellectual development
- Exposure to innovative and emerging technology domains
- Multiple specialization options tailored to evolving career landscapes
- A strong focus on employability, entrepreneurship, and global career readiness
-

Our Self-Learning Materials are meticulously developed by experts, enriched with contemporary case studies, real-world applications, and practical insights to ensure clarity, engagement, and relevance. We are committed to equipping you not just with knowledge, but with the confidence and competence to excel in a competitive global environment.

At Sreenarayanaguru Open University, you are never alone in your learning journey. Our dedicated learner support system is designed to provide continuous academic guidance, timely assistance, and effective grievance redressal. We encourage you to actively engage with us, share your concerns, and make the most of the resources and support available to you.

As you begin this important phase of your academic journey, I urge you to embrace learning with curiosity, discipline, and determination. The world of technology is ever-evolving, and your willingness to adapt, innovate, and grow will define your success.

Remember, this is not just a programme—it is a pathway to transforming your future.
I wish you a fulfilling learning experience and a successful, inspiring career ahead.

Warm regards,



Prof. (Dr.) Jagathy Raj V. P.
Vice Chancellor

SGOU

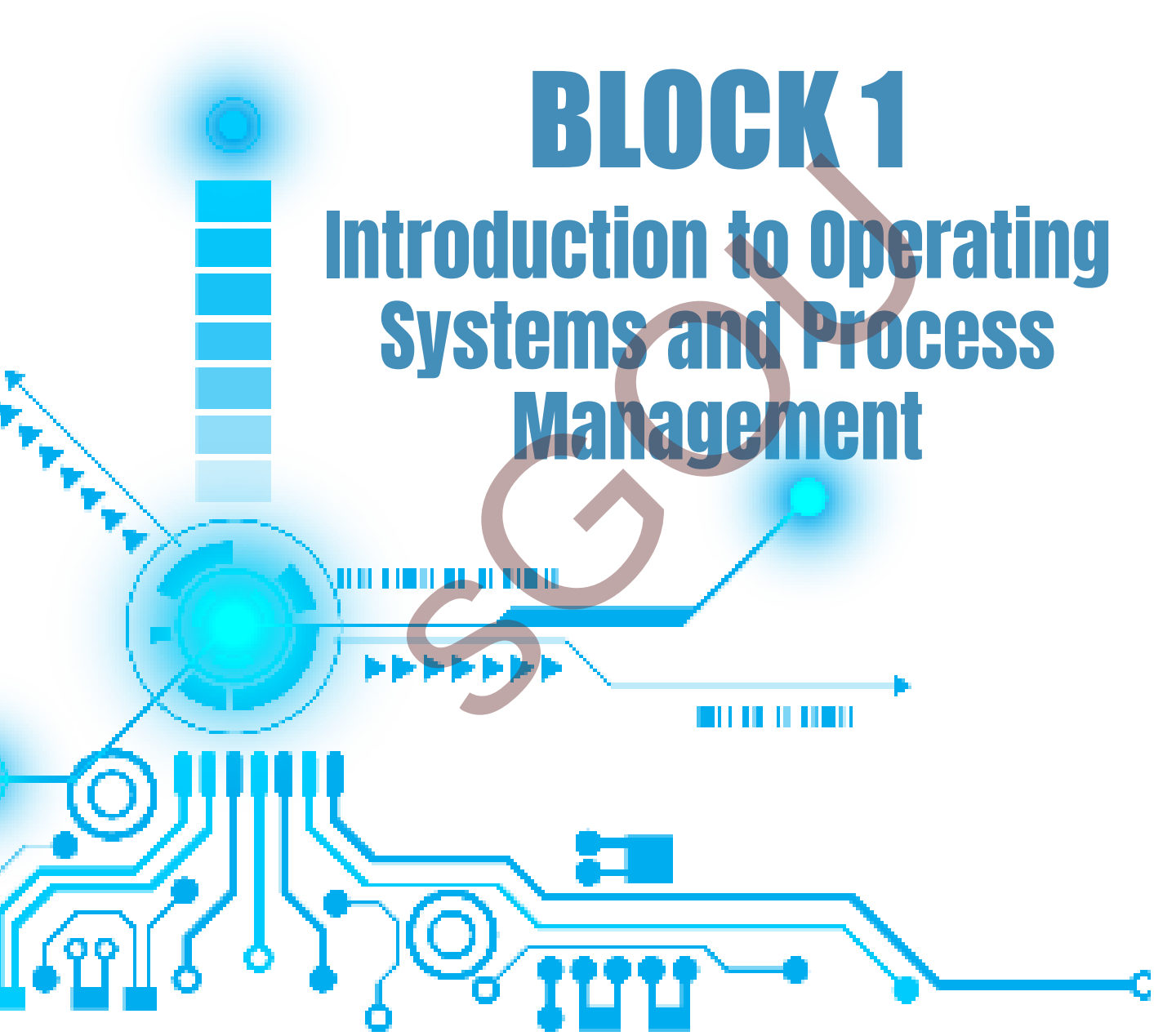
Contents

Block 01	**Introduction to Operating Systems and Process Management**	**1**
Unit 1	Introduction, Types, and Structures	2
Unit 2	Process Concept and Scheduling	37
Unit 3	Inter-process Communication and Synchronization	63
Unit 4	Deadlocks	85
Block 02	**Memory Management**	**98**
Unit 1	Memory Management Strategies	99
Unit 2	Virtual Memory Management	113
Unit 3	Page Replacement Algorithms	124
Unit 4	Segmentation and Paging	138
Block 03	**File Systems and I/O Management**	**151**
Unit 1	File System Interface	152
Unit 2	File System Implementation	181
Unit 3	I/O Systems	196
Unit 4	Disk Management	211
Block 04	**Distributed Systems, Security and Specialized OS Concepts**	**231**
Unit 1	Distributed Operating Systems	232
Unit 2	Operating System Security	248
Unit 3	Real-Time Operating System (Rtos)	265
Unit 4	Mobile Operating Systems	281
Model Question Paper Sets		**307**

SGOU

BLOCK 1

Introduction to Operating Systems and Process Management



SGOU

1 UNIT

Introduction, Types, and Structures

Learning Outcomes

After completion of this unit, the learner will be able to:

- define the term Operating System and list its main goals
- identify different types of operating systems such as batch, time-sharing, and real-time systems
- explain the basic functions of an operating system including memory management, process management, and file management
- describe the difference between user view and system view of an operating system
- summarize the basic structure of an operating system and its main components

SGOU

Background

An operating system (OS) is a fundamental system software that acts as an interface between the user and computer hardware. It manages hardware resources and provides a convenient environment for users to execute programs. The main goals of an operating system include making the computer system easy to use, ensuring efficient resource utilization, and enabling secure and reliable execution of applications. For example, when a user opens a word processor like Microsoft Word or a browser like Chrome, the OS handles memory allocation, input/output operations, and process scheduling behind the scenes.

Operating systems are classified into several types based on their working mechanisms and usage environments. In a batch operating system, jobs are collected and processed in groups without user interaction, such as payroll systems in large orga-

nizations. A multiprogramming OS allows multiple programs to reside in memory simultaneously, improving CPU utilization; for example, running a compiler, music player, and browser together. A time-sharing system enables multiple users to access a system concurrently, like in online examination systems or university lab computers. Other advanced types include distributed operating systems, which manage multiple connected computers as a single system (e.g., cloud computing platforms), real-time systems used in applications like air traffic control and medical devices, multiprocessor systems that use multiple CPUs for faster processing, and mobile operating systems like Android and iOS that are designed for smartphones and tablets. These systems provide services from both a user and system perspective and are structured in different ways such as monolithic, layered, or microkernel architectures to efficiently manage hardware and software resources.

Keywords

Batch, Multiprogramming, Time-sharing, Distributed, Real-time, Multiprocessor, and Mobile operating systems

Discussion

SGOU

An operating system is an essential system software that provides an interface between users and computer hardware while managing system resources effectively. It defines how hardware and software interact and ensures that computing tasks are performed efficiently and securely. The unit introduces the fundamental goals of an operating system, such as convenience for users, efficient utilization of resources, and smooth execution of programs. It also covers core functions like process handling, memory allocation, file organization, device control, and system security, which together ensure stable system performance.

This unit also explores various types of operating systems, including batch systems for processing jobs in groups, multiprogramming systems for executing multiple programs simultaneously, and time-sharing systems that allow multiple users to interact with a system concurrently. It further includes distributed operating systems that connect multiple computers, real-time systems used in critical applications, multiprocessor systems for high-speed processing, and mobile operating systems designed for portable devices. In addition, it explains operating system services from both user and system perspectives and describes how different OS structures organize and manage system components for better efficiency and reliability.

1.1.1 Introduction to Operating System

An operating system is a fundamental system software that manages computer hardware and software resources and provides a stable environment for running applications. It acts as a bridge between the user and the computer hardware, ensuring that users can interact with the system without needing to directly control complex hardware operations. The operating system continuously runs in the background and coordinates all activities of the computer, such as executing programs, managing memory, and controlling input/output devices.

Definition

An **operating system (OS)** is a system software that acts as an intermediary between computer hardware and the user. It prevents users from directly accessing hardware components and provides a controlled environment for interaction. The OS is a program that runs continuously on a computer and manages all other applications and system activities. It ensures that hardware resources such as memory, processor, and input/output devices are properly allocated to different tasks or processes (fig 1.1.1).

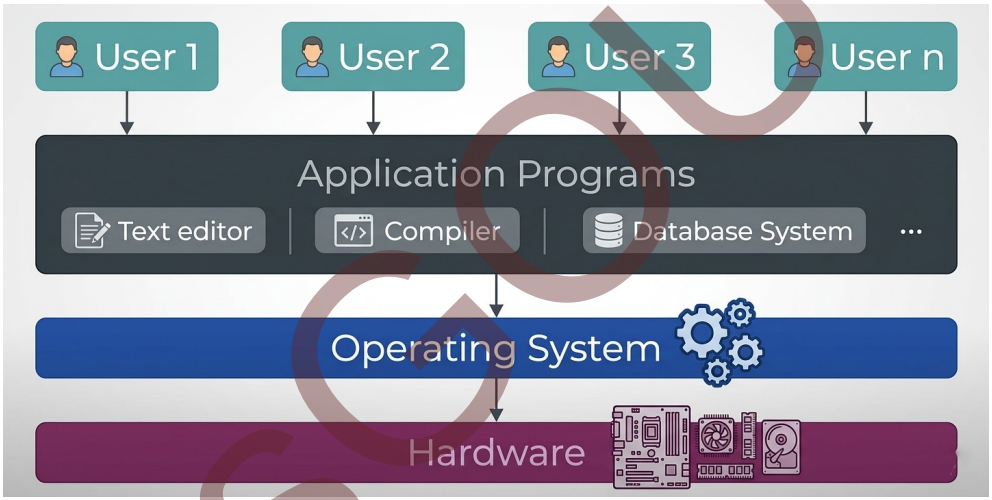


Fig 1.1.1 Concept of Operating System

An operating system can be understood as the core manager of a computer system that makes computing possible in a convenient and efficient way.

Example

When you use a smartphone with Android or an iPhone with iOS, the operating system manages all your activities such as opening apps, using the camera, or browsing the internet. For example, when you open YouTube on your mobile, the OS allocates memory, processes the request, and displays the video smoothly while managing background tasks.

Goals of Operating System

The goals of an operating system define the primary objectives that guide its design and functioning to ensure efficient resource management, user convenience, and smooth

SGOU

execution of computer programs. The main goals of an operating system are:

- **Convenience:** To provide an easy-to-use environment so that users can execute programs without dealing with hardware complexity.
- **Efficiency:** To utilize system resources like CPU, memory, and I/O devices in the best possible manner.
- **Resource Management:** To allocate and manage hardware resources among multiple processes effectively.
- **Fairness:** To ensure that all processes get a fair share of system resources.
- **Reliability and Security:** To protect system data and ensure stable system operation without unauthorized access.

1.1.1.1 Functions of Operating System

The operating system performs several important functions that ensure smooth system operation. It acts as a bridge between the user and hardware while controlling and coordinating all activities.

1. **Process Management:** The OS manages all running programs (processes). It decides which process gets the CPU and for how long. It handles process creation, scheduling, execution, and termination.
2. **Memory Management:** It assigns memory space to programs that are running. The OS keeps track of each memory location and ensures efficient use of RAM so that multiple programs can run simultaneously.
3. **Resource Allocation:** The operating system assigns system resources such as memory, processors, and input/output devices to processes that need them. It ensures that resources are shared properly among all running programs.
4. **File System Management:** The OS manages files stored on storage devices. It helps in creating, deleting, reading, writing, and organizing files in a structured manner.
5. **Device Management:** It controls input and output devices like keyboards, printers, and monitors using device drivers. The OS ensures smooth communication between hardware devices and software applications.
6. **Security and Protection:** The operating system protects data and system resources from unauthorized access. It ensures user authentication and controls access permissions.
7. **User Interface:** It provides an environment in which a user can execute programs conveniently and efficiently. This interface can be command-line or graphical (GUI), making interaction with the system easier.

SGOU

The operating system is a continuously running program that manages hardware resources and provides a user-friendly environment for executing applications efficiently and safely.

1.1.2 Types of Operating System

Operating systems are categorized into different types based on how they manage processing, memory, users, and system resources. Each type is designed to meet specific computing needs, ranging from simple batch processing systems to advanced real-time and mobile environments. These types of operating systems help improve system performance, support multiple users or tasks, and enable efficient use of hardware in various applications such as desktops, servers, distributed networks, and portable devices.

1.1.2.1 Batch Operating System

A **Batch Operating System** is designed to handle large groups of similar jobs efficiently. In this system, the computer does not interact directly with users during execution. Instead, jobs are grouped together by an operator and processed in batches. These jobs are placed in a queue and executed one after another automatically without any user interaction during processing. The main idea is to execute similar tasks in sequence to improve overall system efficiency.

Working of Batch Operating System

A batch operating system executes a group of similar jobs automatically in batches without user interaction. Users submit their jobs, and the system operator collects and groups them. The OS then processes each batch sequentially, ensuring continuous execution of tasks (fig 1.1.2).

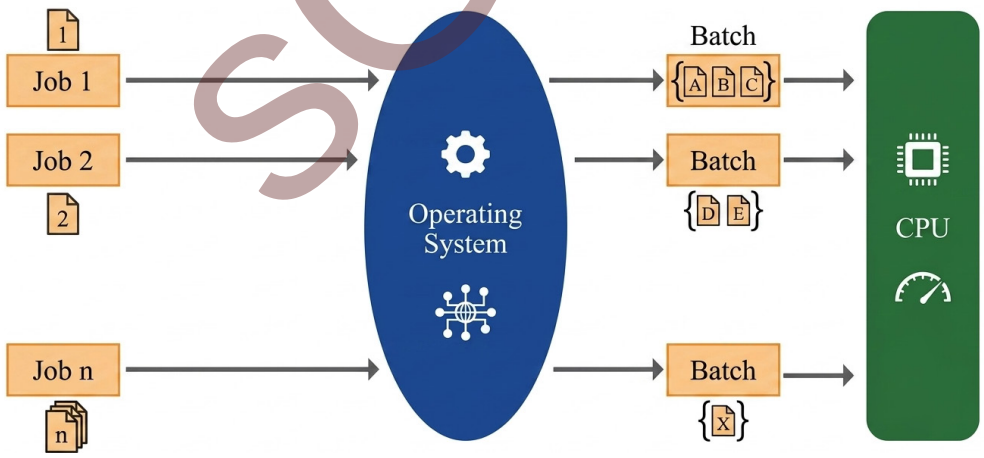


Fig 1.1.2 Working of Batch Operating System

Advantages

The advantages of a batch operating system highlight how it efficiently processes large volumes of similar jobs with minimal human intervention and improved system performance. Some key points are:

SGOU

- **Minimal Idle Time:** The system reduces CPU idle time by processing jobs continuously without human intervention.
- **Handling Repetitive Tasks:** It is suitable for repetitive tasks such as payroll processing, billing systems, and data processing with minimal manual effort.
- **Improved Throughput:** Batch systems can process a large number of jobs at once, which increases overall system throughput and efficiency.

Disadvantages

The disadvantages of a batch operating system arise mainly due to its lack of user interaction during execution and its sequential processing nature, which can lead to delays and inefficient resource utilization. Some key points are:

- **Inefficient CPU Utilization:** When a job is performing input/output (I/O) operations, the CPU may remain idle, leading to poor resource utilization.
- **Increased Response Time:** There is a long delay between job submission and output since jobs are processed sequentially.
- **Lack of Real-Time Feedback:** Users cannot interact with the system while jobs are being processed, making it unsuitable for interactive applications.

Examples

- Insurance claim processing systems
- Library book record management systems
- Stock market report generation systems
- Payroll processing in organizations

A batch operating system is efficient for processing large volumes of similar jobs automatically, but it lacks interactivity and real-time responsiveness, making it suitable mainly for offline data processing tasks.

1.1.2.2 Multiprogramming Operating System

As the name suggests, a **Multiprogramming Operating System** allows more than one program to be in memory and active at the same time. Earlier systems could load and execute only one program at a time, which resulted in poor CPU utilization because the processor often remained idle while waiting for input/output (I/O) operations. To overcome this limitation, multiprogramming was introduced so that the CPU can execute another process whenever the current process is waiting, thereby improving system efficiency and performance.

Concept and Working

In a multiprogramming system, multiple programs are stored in the main memory at the same time, and each program is treated as a process. The operating system manages

SGOU

all these processes and their states. When one process is waiting for I/O operations, the CPU is assigned to another ready process. This switching between processes happens very quickly, creating the illusion that multiple programs are running simultaneously on a single CPU (fig 1.1.3).

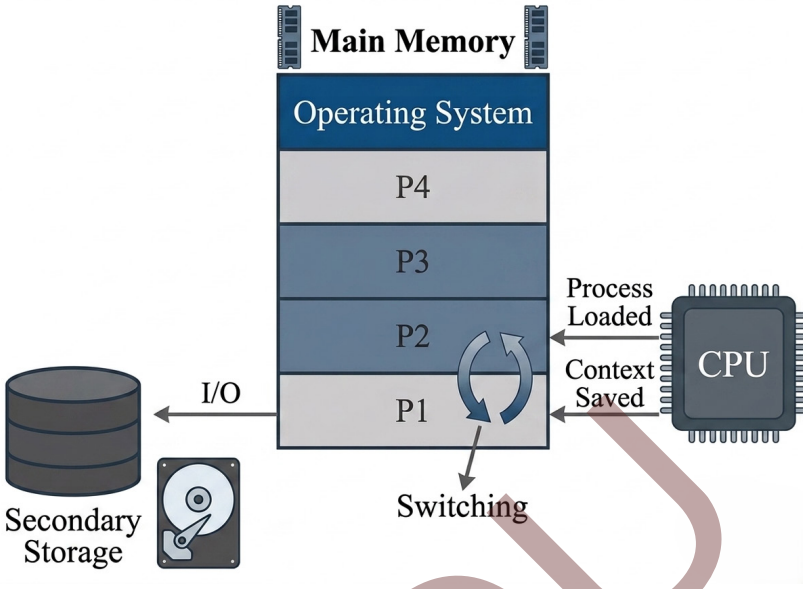


Fig 1.1.3 Working of Multiprogramming OS

For example, if a program is waiting for input from a keyboard or file, the CPU does not remain idle; instead, it immediately executes another process. This ensures better CPU utilization and improved system performance.

Features of Multiprogramming

The features of multiprogramming describe the key characteristics that enable an operating system to manage multiple programs simultaneously in memory and improve overall CPU utilization and system performance. Some key points are:

- Requires a single CPU for execution
- Multiple processes are kept in memory simultaneously
- Context switching occurs between processes
- CPU switches when a process enters a waiting state
- Reduces CPU idle time
- Ensures high resource utilization
- Improves overall system performance

Examples

- *UNIX*
- *Linux (Ubuntu, Red Hat, Debian)*

SGOU

- *macOS*

Advantages

The advantages of multiprogramming highlight the benefits of using multiple programs in memory simultaneously to improve CPU utilization, system efficiency, and overall performance.

- **Better CPU Utilization:** CPU is rarely idle as it executes another process during waiting time.
- **Increased Resource Utilization:** Efficient use of memory, CPU, and I/O devices.
- **Higher Throughput:** More programs are completed in a given time period.
- **Reduced Waiting Time:** Overall waiting time for execution is reduced.
- **Supports Multitasking:** Allows multiple applications to run seemingly at the same time.
- **Improved System Responsiveness:** System responds better to multiple processes.
- **Reliable System Performance:** Improves stability and efficiency of the system.

Disadvantages

The disadvantages of multiprogramming refer to the limitations and challenges that arise due to the complexity of managing multiple processes simultaneously in memory, which can affect system performance and resource management.

- **Complex Memory Management:** All processes must be stored and managed in main memory.
- **Long Waiting for Large Jobs:** Large or long processes may experience delays.
- **High Memory Requirement:** Requires efficient memory allocation techniques.
- **Thrashing Problem:** Excessive multiprogramming may lead to frequent swapping between memory and disk, reducing performance.
- **Heat and Resource Overload:** Running many processes may increase system load and heat.
- **Requires Scheduling Knowledge:** Proper scheduling algorithms are needed to decide process execution order.

SGOU

Types of Scheduling in Multiprogramming

The types of scheduling in multiprogramming refer to the different methods used by the operating system to decide which process should be executed next by the CPU, ensuring efficient process management and system performance.

- **Preemptive Scheduling:** A running process can be interrupted if a higher priority process arrives.
- **Non-Preemptive Scheduling:** A process continues execution until it completes or voluntarily releases the CPU.

Multiprogramming operating systems improve CPU efficiency by allowing multiple processes to reside in memory and switching between them when needed. This reduces idle time and increases system performance, making it a foundation for modern operating systems like **Windows and Linux**.

1.1.2.3 Time-Sharing Operating System

A **Time-Sharing Operating System** is an extension of multiprogramming in which multiple users or multiple processes share the CPU simultaneously. It allows the system to switch rapidly between tasks so that each task gets a small portion of CPU time. This creates the illusion that all tasks are running at the same time, making the system interactive and efficient for multi-user environments.

Concept and Working

A time-sharing operating system is a type of multiprogramming system where each process runs in a round-robin manner. Every task is assigned a fixed time interval called a time quantum. After the completion of this quantum, the operating system automatically switches to the next task in the queue. This continuous switching allows multiple tasks either from a single user or multiple users to run smoothly on a single system (fig 1.1.4).

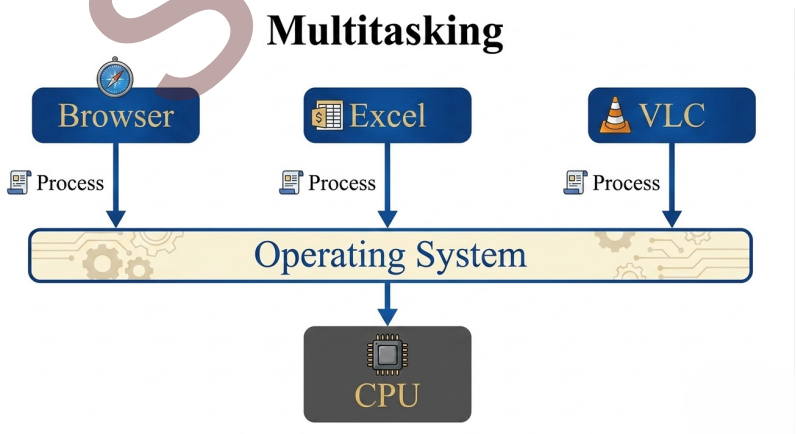


Fig 1.1.4 Working of Time-Sharing OS

For example, when several users are working on a shared computer system, each user's program gets a small time slice, and the CPU keeps rotating between them quickly, ensuring responsive performance.

SGOU

Features

The features of a time-sharing operating system describe the key characteristics that enable efficient sharing of CPU time among multiple users and processes to ensure smooth, interactive, and responsive system performance.

- Supports multiple users simultaneously
- Uses time slicing (quantum) for CPU allocation
- Implements fast context switching
- Provides interactive system response
- Ensures smooth sharing of system resources

Advantages

The advantages of a time-sharing operating system refer to the benefits gained from allowing multiple users and processes to share CPU time efficiently, resulting in better resource utilization, improved responsiveness, and fair access to system resources.

- **Equal CPU Access:** Each task gets a fair share of CPU time.
- **Reduced Software Duplication:** Multiple users can use the same software without installing separate copies.
- **Low CPU Idle Time:** Efficient scheduling ensures the CPU remains busy most of the time.
- **Better Responsiveness:** Users experience quick system response in interactive applications.

Disadvantages

The disadvantages of a time-sharing operating system refer to the limitations that arise due to continuous sharing of CPU time among multiple users, which can lead to security risks, performance overhead, and system reliability issues.

- **Lower Reliability:** If the system fails, it affects all connected users.
- **Security Concerns:** Multiple users increase risks related to data privacy and security.
- **Communication Issues:** Data sharing between users may lead to conflicts or inconsistencies.
- **High System Overhead:** Frequent context switching can reduce overall efficiency.

Examples

- IBM VM/CMS
- TSO (Time Sharing Option)

SGOU

- Windows Terminal Services

A time-sharing operating system allows multiple users or tasks to share CPU time efficiently using time slicing. It improves system interaction and resource utilization but requires careful management to handle security, reliability, and system overhead issues.

1.1.2.4 Distributed Operating System

A Distributed Operating System connects multiple independent computers through a shared communication network to work together as a single integrated system. Each computer in the network has its own CPU and memory, but they collaborate and share resources efficiently. One of the major advantages of this system is remote access, which allows users to access files, applications, and resources stored on other connected systems as if they were local (fig 1.1.5).

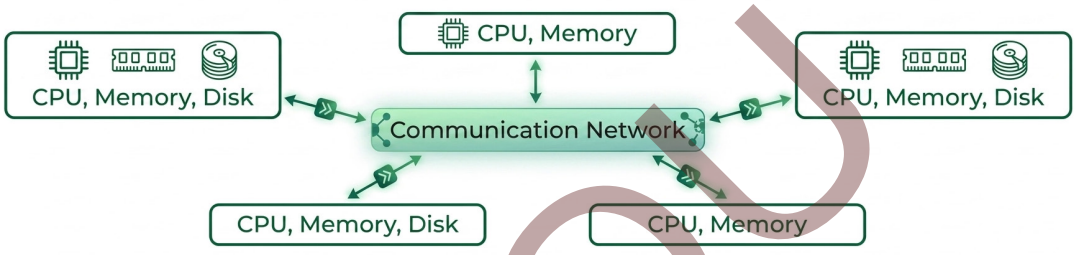


Fig 1.1.5 Architecture of Distributed OS

Definition

A distributed operating system is an OS that manages a group of independent computers and makes them appear to users as a single coherent system.

Working Concept

In a distributed OS, multiple computers (nodes) communicate over a network. Each node performs its own processing, but tasks can be shared among systems to improve performance and reliability. The OS coordinates communication, resource sharing, and task execution across all nodes.

Features

The features of a distributed operating system describe the key characteristics that enable multiple independent computers to work together as a single unified system while sharing resources, coordinating tasks, and improving overall system performance.

- Connects multiple independent computers
- Works as a single coordinated system
- Enables resource sharing across networked systems
- Supports remote access to files and applications

SGOU

- Provides load distribution among systems

Advantages

The **advantages of a distributed operating system** highlight the benefits of connecting multiple independent computers to work together, resulting in improved performance, reliability, scalability, and efficient resource sharing across the network.

- **Independent Systems:** Failure of one machine does not affect the entire system.
- **Easily Scalable:** New computers can be added to the network easily.
- **Faster Processing:** Tasks are distributed, reducing processing time.
- **Resource Sharing:** Files, software, and hardware can be shared efficiently.
- **Improved Reliability:** System continues functioning even if some nodes fail.
- **Better Performance:** Workload balancing improves overall efficiency.

Disadvantages

The disadvantages of a distributed operating system refer to the limitations and challenges that arise due to the complexity of managing multiple interconnected computers, including issues related to network dependency, security, cost, and system coordination.

Network Dependency: If the main network fails, communication between systems stops.

- **High Complexity:** Design and management of distributed systems are complicated.
- **High Cost:** Hardware, networking, and maintenance are expensive.
- **Lack of Standardization:** No universal model or language for implementation.
- **Security Risks:** Data transmitted over networks can be intercepted or tampered with.
- **Data Consistency Issues:** Maintaining consistent data across all nodes is difficult.
- **Communication Delay:** Network delays can affect system performance.

Examples

- LOCUS
- MICROS

SGOU

- Amoeba

A distributed operating system improves performance and reliability by connecting multiple independent computers to work as a unified system. It enables resource sharing and faster processing but introduces challenges such as complexity, security risks, and network dependency.

1.1.2.5 Real-Time Operating System

A **Real-Time Operating System (RTOS)** is designed to serve real-time systems where the response to inputs must occur within a very short and strictly defined time limit. The time taken by the system to process and respond to an input is called **response time**, and in RTOS this time is minimized to ensure immediate and predictable results. These systems are used in applications where timing is critical and delays can lead to serious consequences, such as missile systems, air traffic control systems, medical devices, and industrial robots (fig 1.1.6).

Definition

A real-time operating system is an OS that processes data and provides responses within strict time constraints to ensure timely and accurate system behavior .

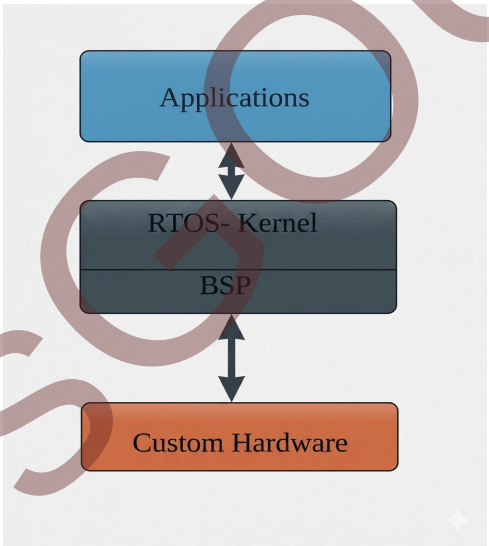


Fig 1.1.6 Real - Time OS

*BSP: Board Support Package

Types of Real-Time Operating System

The types of real-time operating systems are classified based on how strictly they handle timing constraints and ensure that tasks are completed within their required deadlines.

- **Hard Real-Time Operating System:** Used in systems where strict timing constraints must always be met, and any delay is unacceptable. Examples include airbags, nuclear systems, and automatic parachutes. These systems often avoid virtual memory to ensure fast and predictable response.

- **Soft Real -Time Operating System:** Used in systems where timing is important but small delays are acceptable. The system tries to provide fast responses but does not guarantee absolute timing precision. Examples include multimedia applications, video streaming, and online gaming.

Features

The features of a real-time operating system describe the key characteristics that enable it to process data and respond to inputs within strict time constraints while ensuring high reliability, predictability, and efficiency in critical applications.

- Ensures quick response to inputs
- Works under strict timing constraints
- Prioritizes time-sensitive tasks
- Provides predictable system behavior
- Suitable for critical control applications

Advantages

The advantages of a real-time operating system refer to the benefits of using an OS that ensures timely processing of tasks with high accuracy, reliability, and efficient resource utilization in time-critical applications.

- **Maximum Utilization:** Efficient use of system resources leads to higher output.
- **Error-Free Operation:** Designed to minimize errors in critical applications.
- **Efficient Memory Management:** Memory allocation is handled effectively for fast performance.
- **High Reliability:** Ensures consistent and predictable system responses.
- **Fast Processing:** Immediate response to real-time inputs improves system performance.

Disadvantages

The disadvantages of a real-time operating system refer to the limitations associated with its strict timing requirements, complex design, and restricted flexibility, which can make it difficult to implement and less suitable for general-purpose computing.

- **Limited Task Handling:** Focuses on specific tasks, with limited multitasking capability.
- **Complex Design:** Algorithms and system design are highly complex and difficult to implement.
- **Limited Flexibility:** Not suitable for general-purpose computing or frequent task switching.

SGOU

- **High Development Cost:** Requires specialized hardware and software design.
- **Priority Issues:** Managing thread priorities is challenging in real-time environments.

Examples

- Scientific experiments
- Medical imaging systems
- Industrial robots
- Air traffic control systems
- Missile guidance systems

A real-time operating system ensures that critical tasks are executed within strict time limits, making it essential for applications that require high accuracy, reliability, and immediate response.

1.1.2.6 Multiprocessor Operating System

A **Multiprocessor Operating System** is designed to manage and control a computer system that has two or more processors (CPUs) working together. These processors share common memory and resources and are coordinated by a single operating system. The main goal of this system is to increase processing speed, improve performance, and provide better reliability by distributing tasks among multiple processors (Fig 1.1.7).

Definition

A multiprocessor operating system is an OS that manages multiple CPUs in a single computer system, allowing them to work together to execute processes efficiently.

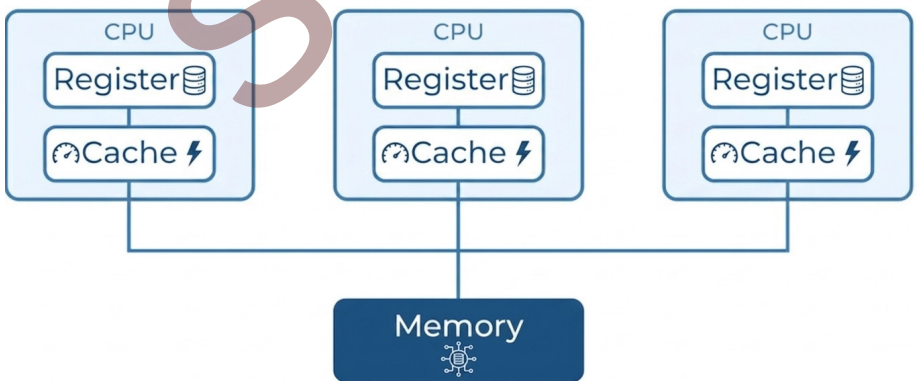


Fig 1.1.7 Multiprocessor OS

Concept and Working

In a multiprocessor system, multiple processors share the same memory and I/O devices. The operating system divides tasks into smaller parts and assigns them to different

processors. These processors execute tasks simultaneously, which improves system speed and efficiency.

For example, while one processor handles user applications, another can manage system tasks like memory management or input/output operations.

Types of Multiprocessor Systems

The types of multiprocessor systems refer to the different architectural designs used to organize and coordinate multiple processors within a computer system to efficiently execute tasks and manage system resources.

- **Symmetric Multiprocessing (SMP):** All processors are equal and share tasks equally under a single OS.
- **Asymmetric Multiprocessing (AMP):** One processor acts as the master and controls other slave processors.

Features

The features of a multiprocessor operating system describe the key characteristics that enable multiple processors to work together in a single system, improving processing speed, efficiency, and overall system performance through parallel execution and resource sharing.

- Multiple CPUs work in a single system
- Shared memory and resources
- Parallel execution of processes
- Single operating system controls all processors
- High performance and fast processing

Advantages

The advantages of a multiprocessor operating system refer to the benefits gained from using multiple processors in a single system, which leads to faster processing, improved performance, better resource utilization, and higher system reliability.

- **Increased Speed:** Multiple processors execute tasks simultaneously.
- **Better Performance:** Parallel processing improves system efficiency.
- **High Reliability:** Failure of one processor does not stop the system.
- **Resource Sharing:** Efficient use of memory and I/O devices.
- **Scalability:** More processors can be added to improve performance.

Disadvantages

The disadvantages of a multiprocessor operating system refer to the limitations and challenges that arise from using multiple processors in a single system, including higher

SGOU

cost, complex system design, and difficulties in coordination and synchronization between processors.

- **High Cost:** Hardware and system design are expensive.
- **Complex OS Design:** Difficult to manage multiple processors efficiently.
- **Synchronization Issues:** Coordination between processors is challenging.
- **Limited Software Support:** Not all applications are designed for parallel processing.

Examples

- High-performance servers
- Scientific simulations
- Weather forecasting systems
- Large database management systems
- Supercomputers

A multiprocessor operating system improves computing power by using multiple CPUs working together under a single OS. It enhances speed, performance, and reliability but requires complex management and higher cost compared to single-processor systems.

1.1.2.7 Mobile Operating System

A Mobile Operating System is designed specifically for mobile devices such as smartphones, tablets, and wearable devices. It manages both hardware and software resources of the device and provides a platform for running mobile applications. It also ensures smooth user interaction through touch-based interfaces and supports communication, multimedia, and internet services. Examples of mobile operating systems include Android and iOS (fig 1.1.8).

Definition

A mobile operating system is an OS developed to operate and control mobile devices by managing hardware resources and supporting application execution in a mobile environment.



Fig 1.1.8 Mobile OS

SGOU

Concept

Mobile operating systems are optimized for portable devices with limited resources like battery power, memory, and processing capacity. They focus on providing efficient performance, connectivity, and user-friendly interaction through touch screens and sensors.

Features

The features of a mobile operating system describe the key characteristics that enable smartphones and tablets to efficiently manage hardware resources, support applications, provide smooth user interaction, and ensure connectivity in a portable computing environment.

- Designed for smartphones and tablets
- Supports touch-based user interface
- Manages hardware, applications, and system resources
- Provides multitasking and background app support
- Enables wireless communication (Wi-Fi, Bluetooth, mobile data)

Advantages

The advantages of a mobile operating system refer to the benefits it provides in terms of ease of use, mobility, connectivity, and access to a wide range of applications on portable devices like smartphones and tablets.

- **User-Friendly Interfaces:** Easy to use with intuitive touch controls.
- **Extensive App Ecosystem:** Large number of apps for different purposes like education, entertainment, and business.
- **High Connectivity:** Supports internet, Bluetooth, GPS, and other communication features.
- **Portability:** Allows users to access information anytime and anywhere.

Disadvantages

The disadvantages of a mobile operating system refer to the limitations related to performance, security, and usability that arise due to hardware constraints, constant connectivity requirements, and the mobile environment.

- **Battery Life Constraints:** Heavy usage reduces battery performance.
- **Security Risks:** Vulnerable to malware, phishing, and data theft.
- **Fragmentation (Android):** Different devices and versions may cause compatibility issues for developers.

SGOU

- **Limited Processing Power:** Compared to desktops, mobile devices have lower performance capacity.

Examples

- Android
- iOS
- BlackBerry OS

Mobile operating systems are specialized OS designed for portable devices, providing easy usability, strong connectivity, and a rich app ecosystem, while also facing challenges like battery limitations and security risks.

1.1.2.8 Network Operating System

A **Network Operating System (NOS)** is an operating system that runs on a server and manages network resources, users, security, applications, and communication between connected computers. It enables multiple users to share files, printers, and other resources within a network, usually in a small private setup. In a network operating system, users can access and sometimes view the configuration and connections of other users, making it a tightly coupled system where all devices depend on central management.

Definition

A Network Operating System is an OS designed to manage and coordinate network resources by providing services such as data sharing, user management, security control, and application access over a network.

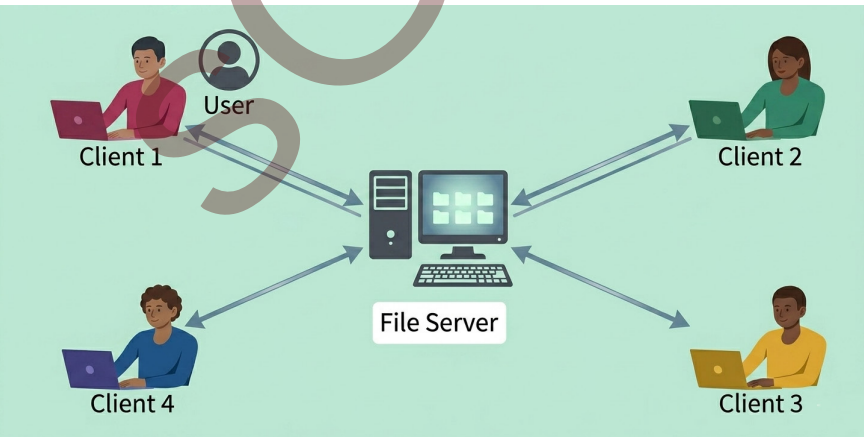


Fig 1.1.9 Network OS

Concept

The NOS works on a server-based model where a central server controls and manages all connected client computers. It ensures that users can communicate and share resources efficiently while maintaining security and proper access control.

SGOU

Features

The features of a network operating system describe the key characteristics that enable centralized management of network resources, user access, security, and communication between connected computers in an efficient and controlled manner.

- Runs on a central server
- Manages users, data, and network security
- Provides file and printer sharing
- Supports remote access to network resources
- Controls communication between connected systems

Advantages

The advantages of a network operating system refer to the benefits gained from centralized control of network resources, which improve resource sharing, security, accessibility, and system management across connected computers.

- **Centralized and Stable Servers:** Ensures reliable control and management of resources.
- **Easy Upgrades:** Hardware and software can be upgraded without affecting the entire network.
- **Remote Access:** Users can access resources from different locations and devices.
- **Efficient Resource Sharing:** Allows multiple users to share files, printers, and applications.
- **Better Security Control:** Centralized authentication improves system security.

Disadvantages

The disadvantages of a network operating system refer to the limitations that arise due to heavy reliance on a central server, higher cost, maintenance requirements, and potential network performance and security issues.

- **High Server Cost:** Installation and maintenance of servers are expensive.
- **Server Dependency:** Most operations depend on the central server; failure affects the entire network.
- **Regular Maintenance Required:** Frequent updates and technical support are needed.
- **Network Traffic Issues:** Heavy usage can slow down the system performance.

Examples

- Microsoft Windows Server 2003

SGOU

- UNIX
- Linux
- Mac OS X Server

A network operating system provides centralized control over network resources, enabling efficient sharing and communication between connected systems. It improves resource management and security but depends heavily on server performance and requires higher maintenance costs.

1.1.3 Operating System Services

An operating system is system software that acts as an intermediary between the user and computer hardware. It allows users to run various application programs and is the most essential software that continuously runs whenever the computer is on. Every computer system must have an operating system to ensure the smooth execution of all other programs and proper functioning of the device.

The operating system performs several important functions, such as coordinating the use of hardware and application programs for users. It provides a platform for running application software and consists of a set of special programs that help the computer operate efficiently. It also controls input–output devices, manages system operations, handles program execution, and takes care of file management for smooth system performance.

1. Program Execution

The Operating System is responsible for managing the execution of programs by first loading them into main memory and then controlling their execution through scheduling techniques such as FCFS and SJF. It ensures that both user and system programs run smoothly without interruption. The OS also prevents situations like deadlocks, where processes may get stuck waiting indefinitely for resources. In addition, it efficiently allocates system resources to maintain proper and effective system performance.

- Loads programs into memory so they can be executed
- Applies CPU scheduling methods like FCFS, SJF, etc., to determine execution order
- Detects and avoids deadlock situations during processing
- Allocates and manages system resources for smooth overall operation

2. Input/Output Operations

The Operating System is responsible for managing all input and output operations in a computer system. It acts as a bridge between users, applications, and hardware devices, ensuring smooth communication through device drivers. Device drivers serve as an interface between the OS and hardware components, enabling proper control of devices such as keyboards, printers, disks, and monitors. The OS ensures that input and

SGOU

output requests are handled efficiently, safely, and without conflicts between multiple programs.

- Manages all input and output operations in the system
- Communicates with hardware devices through device drivers
- Ensures proper coordination between hardware components and the OS
- Provides controlled and secure access to I/O devices for programs
- Handles buffering, caching, and spooling to improve performance
- Manages data transfer between memory and external devices
- Prevents conflicts when multiple programs request the same device

3. Communication Between Processes

The Operating System is responsible for managing communication between different processes running in the system. It enables smooth and efficient data exchange, ensuring that processes can share information without conflicts. The OS supports inter-process communication (IPC) mechanisms that allow processes within the same system as well as across networked systems to communicate effectively. This coordination improves system performance and ensures reliable data sharing.

- Manages inter-process communication (IPC) mechanisms
- Enables smooth transfer of data between different processes
- Supports communication within the same system as well as over networks
- Ensures secure exchange of information between processes
- Prevents data inconsistency during communication
- Coordinates synchronization between interacting processes
- Improves system efficiency through organized communication

4. File Management

The Operating System is responsible for managing all file-related activities in a computer system. It controls how files are created, stored, accessed, modified, and deleted. The OS also ensures proper access permissions such as read, write, or execute so that only authorized users and programs can use the files. It organizes data in a structured manner across different storage devices like hard disks, SSDs, and removable devices, ensuring efficient storage and retrieval.

- Grants file access permissions such as read-only, write, and execute
- Allows users to create, delete, rename, and organize files and folders
- Manages file storage across devices like hard disks, SSDs, and pen drives
- Maintains directory structure for easy file organization and access

SGOU

- Ensures data security by controlling unauthorized access

5. Memory Management

The Operating System manages computer memory efficiently by allocating and controlling memory space for different programs. It works like a cricket team manager who decides which players should be in the playing squad based on performance and requirements. Similarly, the OS checks whether a program qualifies for memory allocation and assigns the required amount of memory. It also ensures that memory is used efficiently without wastage or overlap between programs.

- Checks whether a program is eligible for memory allocation
- Determines the amount of memory required by each program
- Allocates memory space in suitable locations for execution
- Prevents unnecessary or excessive memory usage
- Frees memory when programs are no longer needed
- Avoids conflicts between multiple programs using memory

6. Process Management

The Operating System is responsible for managing multiple processes running in a system. It can be compared to a chef in a kitchen where the CPU is the stove and each program is a dish being prepared. The OS ensures that every process gets a fair share of CPU time so that no single process dominates the system. It schedules tasks properly and maintains smooth execution of all running programs.

- CPU acts as the execution unit similar to a kitchen stove
- OS acts as the chef who manages all running processes
- Ensures each process gets fair CPU time for execution
- Prevents any single process from occupying the CPU for too long
- Uses scheduling techniques to manage process execution order
- Handles creation, execution, and termination of processes

7. Security and Privacy

Security

The Operating System ensures the safety of the computer system by protecting it from unauthorized access and external threats. It adds multiple layers of protection to safeguard system resources and data. The OS defends against harmful activities such as viruses, malware, and hacking attempts using security tools like firewalls and antivirus programs. These mechanisms help maintain a secure computing environment.

- Blocks unauthorized users from accessing the system
- Protects the system from viruses, malware, and hacking attempts

SGOU

- Uses security tools such as firewalls and antivirus software
- Monitors system activities for suspicious behavior
- Ensures integrity of system files and programs

Privacy

The Operating System also ensures that user data and personal information remain private and secure. It controls access to sensitive files and ensures that only authorized users can view or modify them. Similar to a locked door that only the correct key can open, the OS protects confidentiality and prevents misuse of personal data.

- Keeps personal and sensitive data hidden from unauthorized users
- Provides access control mechanisms such as passwords and user accounts
- Restricts file and folder access based on user permissions
- Ensures only authorized users can view or modify information
- Protects confidential data from leakage or misuse

8. Resource Management

The Operating System is responsible for managing and distributing system resources effectively among all running processes. It ensures that hardware resources such as CPU, memory, and input/output devices are used in an organized and efficient manner. By deciding how and when each process gets access to resources, the OS prevents conflicts and improves overall system performance.

- Shares system resources like CPU, memory, and devices among multiple processes
- Allocates CPU time using scheduling algorithms such as FCFS, SJF, and Round Robin
- Manages memory allocation and deallocation efficiently
- Controls and coordinates input/output device usage
- Ensures fair distribution of resources to avoid starvation

9. User Interface

The Operating System provides a user interface that allows users to interact with the computer system in an easy and efficient way. It acts as a bridge between the user and hardware, enabling communication through different types of interfaces such as Command Line Interface (CLI) and Graphical User Interface (GUI). In CLI, users type commands that are interpreted and executed by the system, while GUI provides a more visual and user-friendly environment using windows, icons, and menus.

- Provides both CLI and GUI for user interaction
- CLI uses a command interpreter to execute user commands

SGOU

- GUI offers windows, icons, menus, and pointer-based navigation
- Makes system interaction simple and user-friendly
- Allows users to access applications and system functions easily

10. Networking

The Operating System supports networking services that enable communication between multiple devices connected through a network. It manages connections, controls data transmission, and ensures reliable exchange of information. This allows systems to connect to the internet, share resources, and communicate efficiently across local or wide-area networks.

- Manages connections between different devices in a network
- Supports internet access and data communication
- Sends and receives data packets across networks
- Ensures reliable and accurate data transfer

11. Error Handling

The Operating System is responsible for detecting, reporting, and correcting errors that occur during system operation. These errors may arise in the CPU, memory, or input/output devices. The OS ensures system stability by handling such issues efficiently and preventing system crashes or failures. It also protects the system from unexpected errors and external threats.

- Detects errors in CPU, memory, and input/output devices
- Handles and reports system and program errors
- Prevents system crashes and deadlocks
- Corrects errors during program execution whenever possible
- Ensures smooth and stable system performance

12. Time Management

The Operating System manages time-sharing between multiple processes to ensure smooth and efficient execution. It allocates CPU time to different tasks so that all programs get a fair chance to run. This can be compared to a traffic signal system where different lights control the flow of vehicles (processes), ensuring order and avoiding congestion.

- Reduces CPU idle time by efficient scheduling
- Allocates time slices to processes for execution
- Ensures fair sharing of CPU among all processes
- Red light represents waiting processes (simple queue)

SGOU

- Yellow light represents ready processes in queue
- Green light represents running processes
- Switches between processes at regular intervals

Operating System services play a vital role in ensuring the smooth and efficient functioning of a computer system. They manage important tasks such as program execution, memory management, process scheduling, file handling, input/output operations, security, and resource allocation. By acting as an interface between the user and hardware, the operating system ensures proper coordination of all system components. These services improve performance, provide security, support multitasking, and make the system user-friendly, reliable, and efficient for everyday computing tasks.

1.1.4 Views of Operating System

The operating system can be understood from two main perspectives: the **user view** and the **system view**. These two views help in understanding how an OS works differently depending on whether the focus is on user interaction or internal system functioning. The user view concentrates on how users interact with the system, while the system view focuses on how the OS manages hardware and resources efficiently (fig 1.1.10).

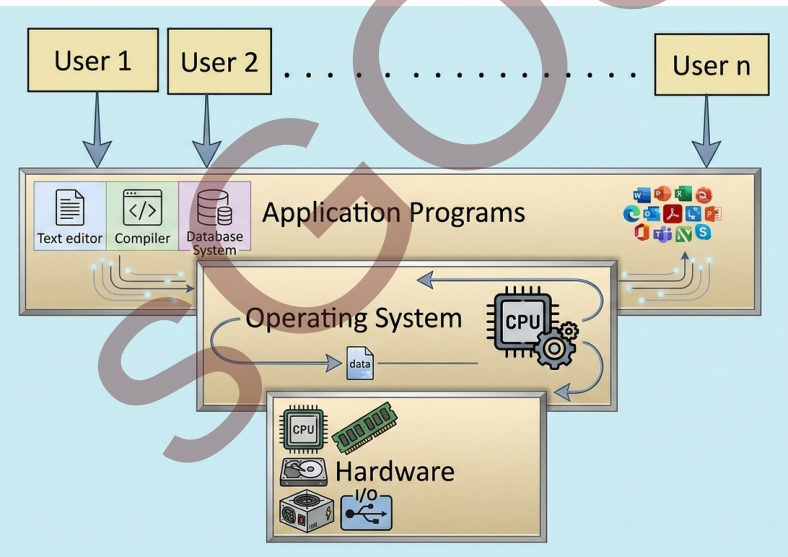


Fig 1.1.10 Views of Operating Syatem

1. User View

The user view of an operating system focuses on the interaction between the user and the computer system. It emphasizes ease of use, convenience, and accessibility through application programs and user interfaces rather than internal technical operations. The goal is to provide a smooth and user-friendly experience.

Types of User View

- **Single User Systems:** In systems like personal computers, the OS is designed

for one user at a time. The user can easily run applications such as browsers, games, and office tools, with a focus on simplicity and convenience.

- **Multi-User Systems:** These systems allow multiple users to access the system simultaneously, such as servers or mainframes. The OS ensures fair resource sharing while maintaining performance for all users.
- **Handheld Devices:** Smartphones and tablets provide a touch-based interface. The OS focuses on mobility, battery efficiency, and simple interaction for users on the go.
- **Embedded Systems:** Devices like smart TVs, washing machines, and cars use OS with minimal user interaction. The system performs specific tasks with limited or no direct user interface.

2. System View

The system view considers the operating system as a resource manager and control program. It focuses on how the OS manages hardware resources efficiently and ensures proper system functioning.

Key Functions in System View

- **Resource Allocation:** The OS manages CPU, memory, storage, and I/O devices efficiently among different processes.
- **Hardware Control:** It controls hardware components using device drivers and manages communication between hardware and software.
- **Process and Memory Management:** Handles scheduling of processes, memory allocation, paging, and virtual memory.
- **I/O Management:** Manages input/output devices and ensures smooth data transfer.
- **Interrupt Handling:** Responds to system and hardware interrupts for quick processing.
- **System Coordination:** Ensures all components of the system work together efficiently.

Key differences between User View and System View

The key differences between User View and System View highlight how an operating system is understood from two perspectives: one focusing on user experience and interaction, and the other focusing on internal system operations and resource management. (Table 1.1.1)

SGOU

Table 1.1.1 User view and System view

Aspect	User View	System View
Primary Focus	Ease of use and user experience	Resource management and system efficiency
Main Concern	Interface, applications, and interaction	Hardware control and system operations
Objective	Provide a simple and convenient computing experience	Ensure optimal utilization of system resources
Performance Priority	User satisfaction and responsiveness	System throughput and efficiency
Complexity	Hidden from the user, simple interface	Involves complex internal system operations
Interaction	Direct interaction through GUI or CLI	Indirect interaction through system processes
Example Systems	Personal computers, smartphones	Servers, mainframes, embedded systems
Functionality Focus	Running applications smoothly	Managing CPU, memory, I/O, and processes

The operating system serves two important roles through its dual perspectives. The user view ensures that the system is easy to use, interactive, and accessible, while the system view ensures efficient management of hardware resources and smooth system operation. Both views together help in achieving a balance between usability and performance, making the operating system effective for both users and system requirements.

1.1.5 Operating System Structure

An operating system is designed using different structural models that define how its components are organized and how they interact within the kernel. This structure acts as a framework or blueprint that guides the overall design and functioning of the OS. Since an operating system is complex, dividing it into smaller and manageable modules makes development, maintenance, and understanding much easier. The structure also determines how efficiently different components work together inside the system.

- OS structure defines how system components are organized and connected
- Acts as a blueprint for designing and building the operating system
- Breaks the system into smaller modules for easier development and maintenance
- Specifies how components interact within the kernel
- Improves system efficiency through organized design

SGOU

1. Simple Structure

Simple-structure operating systems are designed without a clear or well-organized architectural framework. They are generally small in size and provide only basic functionality. In this model, system components are not properly separated into layers, which reduces modularity and makes the system less reliable. Because there is little isolation between user programs and system functions, an error in one program can easily affect the entire system (fig 1.1.11).

- Lacks a well-defined architectural structure; simple and limited in functionality
- Does not clearly separate system components and interfaces

Example: MS-DOS

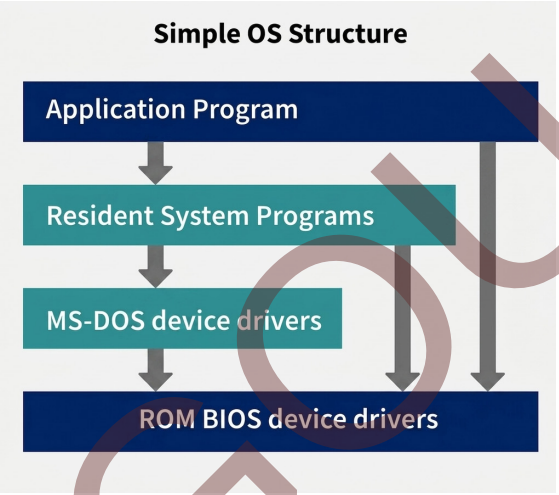


Fig 1.1.11 Simple OS Structure

2. Monolithic Structure

A monolithic structure is an operating system design in which the entire system functions as a single large kernel. In this approach, all core services are integrated into one unified codebase and run in kernel mode. Since all components are tightly coupled, the system executes quickly, but it can become complex and harder to manage as it grows. Services such as process management, memory management, file systems, and device drivers all operate within the same kernel space.

- The entire operating system is built as one large kernel
- All core services are included in a single unified codebase
- Includes process management, memory management, and file system management
- Device drivers operate directly within the kernel
- All components are tightly integrated and share the same memory space

Example: UNIX

SGOU

3. Layered Structure

In a layered structure operating system, the system is organized into multiple layers, where each layer performs specific functions and interacts only with the layer directly below it. The lowest layer contains the hardware, while the topmost layer provides the user interface. Each layer depends on the services provided by the lower layers, making the system more structured and easier to understand. This approach improves modularity and makes debugging simpler, as issues can be traced to a particular layer. However, it may introduce additional processing overhead because data must pass through several layers before reaching its destination (fig 1.1.12).

- OS is divided into multiple hierarchical layers
- Each layer uses services of the layer below it
- Hardware forms the lowest layer and user interface forms the highest
- Improves system organization and modular design
- Requires careful design and planning of layers
- Ensures better separation of functions and responsibilities

Example: UNIX operating system



Fig 1.1.12 Layered OS Structure

4. Microkernel Structure

A micro-kernel structure is an operating system design in which only the essential core functions are kept inside the kernel, while all non-essential services are moved to user space. The kernel is kept very small and is responsible only for basic operations such as communication between hardware and software. Other services like file systems, device drivers, and network management run as separate processes outside the kernel. This design improves flexibility, security, and reliability because system services can fail without affecting the entire operating system.

- Only essential services are included in the kernel
- Non-essential services run in user space as separate processes
- Kernel handles basic functions like communication and process management
- New services can be added without modifying the kernel
- Easier to maintain and update individual components
- Better system stability due to modular design
- Reduces risk of complete system failure

Example: macOS uses microkernel concepts within the hybrid XNU kernel

5. Modular Structure

A modular structure is an operating system design in which the kernel contains only the essential core functions, while additional features are implemented as separate modules. These modules can be loaded or removed dynamically during boot time or even while the system is running. Each module performs a specific task and interacts with the kernel through well-defined interfaces. This structure provides high flexibility and makes the system easier to extend and maintain compared to traditional designs (fig 1.1.13).

- Kernel includes only essential core components
- Additional services are implemented as loadable modules
- Modules can be added or removed during boot time or runtime
- Provides flexibility and easy system expansion
- Uses well-defined interfaces between kernel and modules
- Modules can communicate more freely compared to layered structure

Example: Solaris operating system

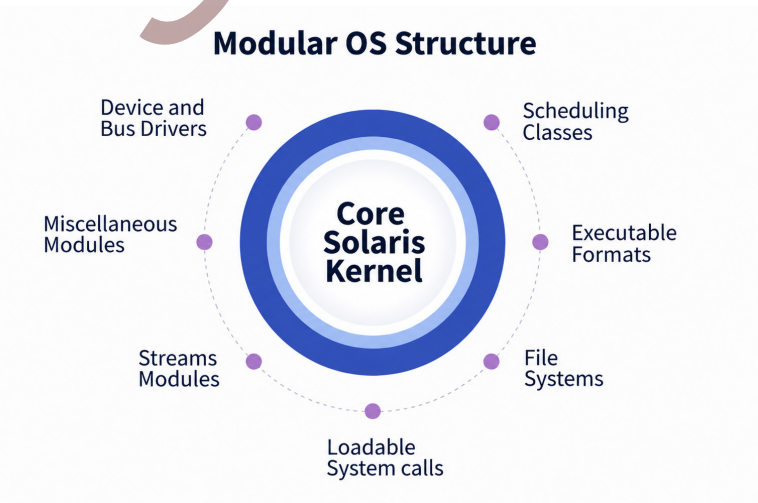


Fig 1.1.13 Modular OS Structure

SGOU

6. Hybrid Structure

A hybrid-kernel structure is an operating system design that combines the characteristics of both monolithic and microkernel architectures. It aims to achieve a balance between performance and modularity by integrating essential services within the kernel while keeping some services in user space. This approach allows the system to benefit from the fast execution of monolithic kernels and the flexibility and stability of microkernels, making it more practical for modern operating systems.

- Combines features of both monolithic and microkernel structures
- Includes essential services in the kernel for faster performance
- Keeps some services in user space for better modularity
- Provides high speed similar to monolithic kernel systems
- Ensures improved stability and reliability like microkernels
- Offers greater flexibility in system design and implementation
- Supports better resource management and system control

Example: Windows and macOS (XNU hybrid kernel)

In conclusion, an operating system is a fundamental software component that acts as an interface between users and computer hardware while managing all system resources efficiently. This unit has provided a clear understanding of the definition, goals, and functions of an operating system, along with its major types such as batch, multiprogramming, time-sharing, distributed, real-time, multiprocessor, and mobile operating systems. It also explains how operating system services support smooth functioning through user and system views, and how different OS structures are designed to organize system components effectively.

Overall, studying these concepts helps in understanding how operating systems ensure efficient program execution, resource management, security, and user interaction. Different OS types and structures are designed to meet varying computing needs, from simple personal systems to complex distributed environments, making the operating system an essential and powerful part of modern computing systems.

SGOU

Summarized Overview

This unit covers the fundamental concepts of operating systems, including their definition, goals, and core functions. It explains how an operating system acts as an interface between users and computer hardware while efficiently managing system resources such as CPU, memory, files, and input/output devices. It also introduces different types of operating systems, including batch, multiprogramming, time-sharing, distributed, real-time, multiprocessor, and mobile systems, each designed for specific computing environments. Additionally, the unit discusses operating system services from both user and system perspectives, highlighting how users interact with the system and how the OS internally manages resources and processes. It also explains various OS structures such as simple, layered, monolithic, microkernel, hybrid, and modular designs, which define how system components are organized. Overall, the unit provides a clear understanding of how operating systems ensure efficient, secure, and reliable computing.

Assignments

1. Define an operating system and explain its main goals and functions.
2. Describe different types of operating systems with suitable examples.
3. Explain operating system services and discuss user view and system view in detail.
4. Compare different OS structures such as simple, layered, monolithic, microkernel, hybrid, and modular.
5. Explain the process of program execution, memory management, and process management in an operating system.
6. Discuss how an operating system manages security, file systems, input/output operations, and networking services.

SGOU

Reference

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2023). *Operating system concepts* (10th ed.). Wiley.

2. Tanenbaum, A. S., & Bos, H. (2022). *Modern operating systems* (5th ed.). Pearson.

3. Stallings, W. (2023). *Operating systems: Internals and design principles* (10th ed.). Pearson.

4. Arpaci-Dusseau, R. H., & Arpaci-Dusseau, A. C. (2018). *Operating systems: Three easy pieces*. Arpaci-Dusseau Books.

5. Dhamdhere, D. M. (2017). *Operating systems: A concept-based approach* (3rd ed.). McGraw Hill Education.

Suggested Reading

1. GeeksforGeeks – Operating System Tutorial
   https://www.geeksforgeeks.org/operating-systems/

2. TutorialsPoint – Operating System Tutorial (PDF & Notes)
   https://www.tutorialspoint.com/operating_system/os_pdf_version.html

3. Guru99 – Operating System Tutorial
   https://www.guru99.com/os-tutorial.html

4. Great Learning – Operating System Introduction
   https://www.mygreatlearning.com/operating-system/tutorials

5. ResearchGate – OS Fundamentals PDF Notes
   https://www.researchgate.net/publication/371274506_Brief_Introduction_to_Operating_Systems

SGOU

Space for Learner Engagement for Objective Questions

Learners are encouraged to develop objective questions based on the content in the paragraph as a sign of their comprehension of the content. The Learners may reflect on the recap bullets and relate their understanding with the narrative in order to frame objective questions from the given text. The University expects that 1 - 2 questions are developed for each paragraph. The space given below can be used for listing the questions.

SGOU

2 UNIT

Process Concept and Scheduling

Learning Outcomes

After completion of this unit, the learner will be able to:

- define the concepts of program and process
- list the different states in a process lifecycle
- identify the components of a Process Control Block (PCB)
- recall the types of threads and their characteristics
- name various process scheduling algorithms and scheduling criteria

SGOU

Background

In everyday life, we perform many tasks such as opening apps on a mobile phone, browsing the internet, listening to music, or working on documents. Even though these activities appear to happen smoothly and sometimes at the same time, there is a system working behind the scenes to manage them efficiently. For example, when you switch between apps or run multiple applications together, your device ensures that each task gets the required attention and resources.

Similarly, a computer system also handles multiple tasks using a structured approach. You may already know that a computer follows instructions given in the form of programs. However, for these instructions to actually perform tasks, they must be executed, and this execution is managed by the operating system. This introduces important concepts such as processes, threads, and scheduling, which help the system run multiple tasks efficiently.

In this unit, you will learn how a program becomes a process, how processes are organized in memory, and how they move through different states during exe-

cution. You will also understand how the operating system keeps track of each process using structures like the Process Control Block (PCB). Further, the unit explains how multiple tasks are handled using threads and how the CPU decides which process to execute using scheduling algorithms. This knowledge will help you understand how computers manage multitasking and ensure smooth system performance.

Keywords

Process, Process Control Block (PCB), Process Scheduling, Ready Queue, Job Queue

Discussion

1.2.1 Process vs. Program

In an operating system, the concepts of program and process are fundamental to understanding how tasks are executed within a computer system. Although these terms are often used interchangeably, they have distinct meanings. A program represents a set of instructions, whereas a process represents the execution of those instructions. Understanding this distinction helps in analyzing how the operating system manages resources and performs multitasking.

A program is defined as a collection of instructions designed to perform a specific task, which is stored on a secondary storage device such as a hard disk. It is considered a passive entity, as it does not perform any operation by itself. A program does not possess any execution state and does not require system resources until it is loaded into main memory. It serves as a static representation of a task that is yet to be executed.

A process, on the other hand, is a program in execution and is regarded as an active entity within the system. When a program is loaded into main memory and executed by the CPU, it becomes a process. A process has a well-defined life cycle and transitions through various states such as ready, running, and waiting. It requires system resources including CPU time, memory, and input/output devices for its execution. Thus, a process represents the dynamic execution of a program and enables the actual performance of tasks within the system. In the upcoming section, we will discuss the concept of processes in detail.

1.2.2 Introduction to Process

A process is defined as a program in execution and is considered a fundamental unit of

SGOU

work in an operating system. It includes not only the program code but also the current activity of the program along with the resources allocated to it. Thus, a process represents the dynamic execution of a program and contains all the necessary information required for its proper functioning.

For example, when a user opens a web browser to access a website, the operating system creates a corresponding process. This involves loading the program from secondary storage into main memory and initiating its execution. During execution, the process enables the user to perform various activities such as browsing web pages, opening multiple tabs, streaming content, and downloading files. Each of these actions is carried out as part of the process.

While executing, a process does not remain in a single state; instead, it transitions through different states during its lifecycle. It enters the running state when actively executing instructions, moves to the waiting state when it is waiting for input/output operations or user interaction, and finally reaches the terminated state when the execution is completed. These states help the operating system efficiently manage multiple processes, and they will be discussed in detail in the subsequent section.

1.2.2.1 Memory Structure of a Process

A process is not limited to the execution of program code; rather, it is organized in memory into several distinct sections that support its execution. These sections together form the memory structure of a process, enabling efficient storage, access, and management of data during execution. The memory of a process is typically divided into four main sections: text section, data section, heap, and stack, as shown in Figure 1.2.1.

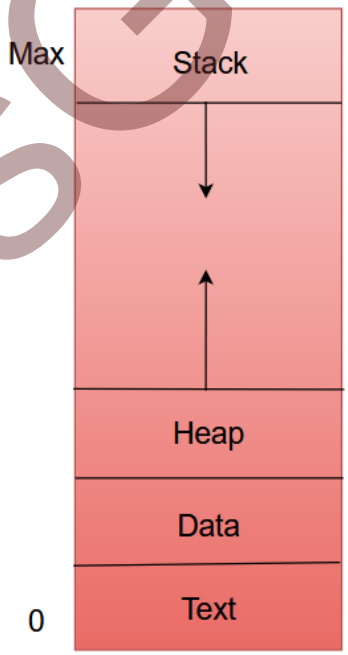


Fig 1.2.1 Memory Structure of Process

SGOU

The above figure illustrates the typical layout of a process in memory. At the lowest level is the text section, which contains the executable instructions of the program. Above it lies the data section, which stores global and static variables required during execution.

The heap is located above the data section and is used for dynamic memory allocation during runtime. It expands upward as the program requests more memory. The stack is positioned at the top of the memory layout and is used for managing function calls and local variables. It grows downward as new function calls are made.

The opposite growth directions of the heap and stack help in efficient utilization of memory. This structured organization allows the operating system to manage process execution effectively and ensures proper allocation and deallocation of memory resources.

1.2.3 Process State Diagram

In a multiprogramming environment, multiple processes reside in the system simultaneously, competing for CPU and other system resources. A process does not execute continuously from start to completion; instead, it undergoes a series of changes in its execution status. These changes are represented as process states, which describe the current condition of a process at any given time. The concept of process states enables the operating system to efficiently manage process execution, allocate resources, and ensure smooth multitasking. By monitoring and controlling these states, the operating system can switch between processes, handle input/output operations, and maintain overall system performance.

Different Process States

A process typically passes through the following states during its lifecycle:

1. New State:

The new state represents the stage where a process is being created. In this phase, the operating system performs necessary initializations such as assigning a process identifier (PID), allocating required resources, and setting up the process control structures. At this point, the process is not yet ready for execution and has not been admitted to the ready queue.

2. Ready State:

In the ready state, the process is prepared for execution and is waiting for CPU allocation. All necessary resources, except the CPU, have been assigned to the process. The processes in this state are maintained in a ready queue, and the scheduler selects one of them based on a scheduling algorithm for execution. The ready state ensures that the CPU is utilized efficiently by keeping processes available for execution at all times.

3. Running State

The running state indicates that the process is currently being executed by the CPU. In this state, the instructions of the process are actively processed. In a single-processor

SGOU

system, only one process can be in the running state at any given time. The process may continue in this state until it completes execution or is interrupted due to various reasons such as time slice expiration or I/O requests.

4. Waiting (Blocked) State

The waiting state, also called the blocked state, occurs when a process cannot continue its execution until a particular event happens. This usually happens when the process is waiting for input/output operations to finish or for some resource to become available. During this time, the process does not use the CPU. Once the required task is completed, the process moves back to the ready state and waits for its turn to execute again.

5. Terminated State

The terminated state signifies that the process has completed its execution or has been explicitly terminated by the operating system. In this state, all resources allocated to the process are released, and the process is removed from the system. This marks the end of the process lifecycle.

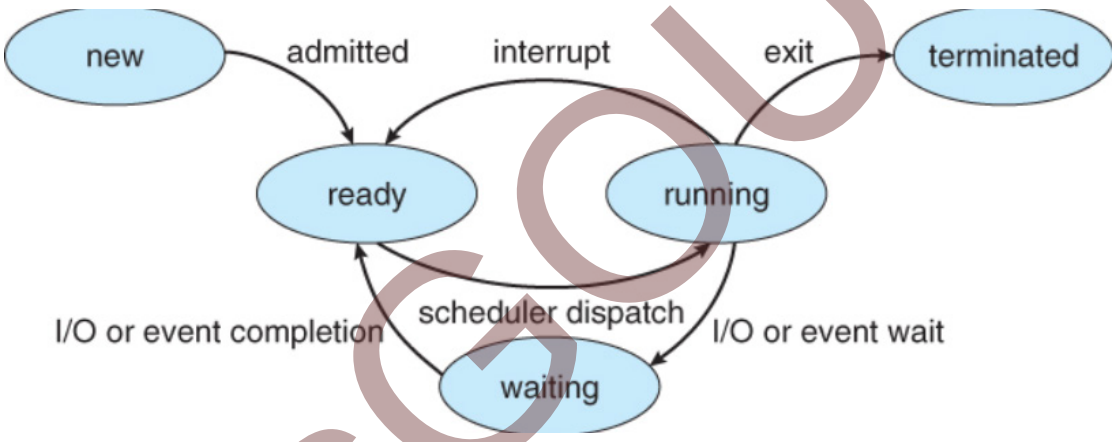


Fig 1.2.2 Process State Diagram

1.2.4 Process Control Block (PCB)

In a multiprogramming environment, the operating system manages multiple processes simultaneously. To keep track of each process and its execution details, the operating system maintains a special data structure known as the Process Control Block (PCB). It acts as a repository of all essential information related to a process and plays a crucial role in process management and scheduling.

Whenever a process is created, a corresponding PCB is also created by the operating system. The PCB is used to store and retrieve information about the process during its lifecycle. It is especially important during context switching, where the state of one process is saved and another process is loaded for execution.

Components of Process Control Block (PCB)

The Process Control Block (PCB) contains all the important information required by

SGOU

the operating system to manage a process. Each component of the PCB plays a specific role in tracking and controlling process execution. The various components of the PCB are illustrated in Figure 1.2.3.

Process Identifier
Process State
Program Counter
CPU Registers
Memory Management Information
I/O Status Information
Accounting Information
……

Fig 1.2.3 Components of Process Control Block (PCB)

1. Process Identifier (PID):

The Process Identifier (PID) is a unique number assigned to each process. It helps the operating system distinguish one process from another. No two processes will have the same PID at the same time.

2. Process State:

This field indicates the current state of the process. A process can be in states such as new, ready, running, waiting, or terminated. The operating system uses this information to manage process execution.

3. Program Counter:

The program counter stores the address of the next instruction that the process needs to execute. This is important when a process is interrupted, as it helps the system resume execution from the correct point.

4. CPU Registers:

This component stores the values of CPU registers used by the process. These include general-purpose registers, stack pointer, and other processor-specific registers. These values are saved during context switching and restored when the process resumes execution.

5. Memory Management Information:

This field contains details about the memory allocated to the process. It may include information such as base and limit registers, page tables, or segment tables. This helps the operating system manage memory efficiently.

SGOU

6. I/O Status Information:

This component stores information about input/output operations related to the process. It includes details such as allocated devices, list of open files, and pending I/O requests.

7. Accounting Information

The accounting information keeps track of resource usage by the process. It may include CPU time used, process priority, user ID, and time limits. This information is useful for scheduling and system monitoring.

To better understand the concept of a Process Control Block, consider a student record system in a university. Each student has a file that contains important details such as student ID, course information, attendance, marks, and current academic status. This record helps the university administration to track and manage each student effectively.

Similarly, in an operating system, each process has a Process Control Block (PCB) that stores all essential information such as process ID, current state, memory details, CPU usage, and input/output status. Just as the student record helps in monitoring and managing student activities, the PCB helps the operating system to manage and control process execution efficiently.

When the system needs to switch from one process to another, it uses the information stored in the PCB to save the current state and resume execution later. This ensures that no information is lost during process execution.

1.2.5 Threads

A thread is a single sequence of execution within a process. It is often referred to as a lightweight process because it requires fewer resources and is faster to create and manage compared to a full process. Threads enable a program to perform multiple tasks simultaneously, thereby improving overall efficiency and performance.

In a single-core system, threads create the illusion of parallel execution through rapid context switching, whereas in a multi-core system, multiple threads can execute truly in parallel on different cores. Each thread maintains its own program counter, register set, and stack, while sharing the code, data, and operating system resources of the parent process.

A simple real-world example can be observed in a word processing application. When a user is typing a document, one thread handles the typing input, another thread automatically saves the document in the background, and another may check spelling and grammar. Even though these tasks appear to happen at the same time, they are managed by different threads within the same process, ensuring smooth and efficient operation of the application.

Need for Threads

Threads play a crucial role in modern operating systems and applications due to the following reasons:

SGOU

- **Improved Performance:** Threads allow multiple tasks to execute concurrently, either in parallel or through interleaving, which enhances the overall speed of program execution.
- **Increased Responsiveness:** If one thread is engaged in a time-consuming operation, other threads can continue to handle user interactions, ensuring that the application remains responsive.
- **Support for Concurrency:** Threads enable multiple activities, such as data processing, file handling, and user input, to occur simultaneously within a single process.
- **Better CPU Utilization:** In multi-core systems, threads can be distributed across different processors, leading to efficient utilization of CPU resources.
- **Efficient Resource Sharing:** Since threads share the same memory space and resources of the process, communication between them is faster and involves less overhead.

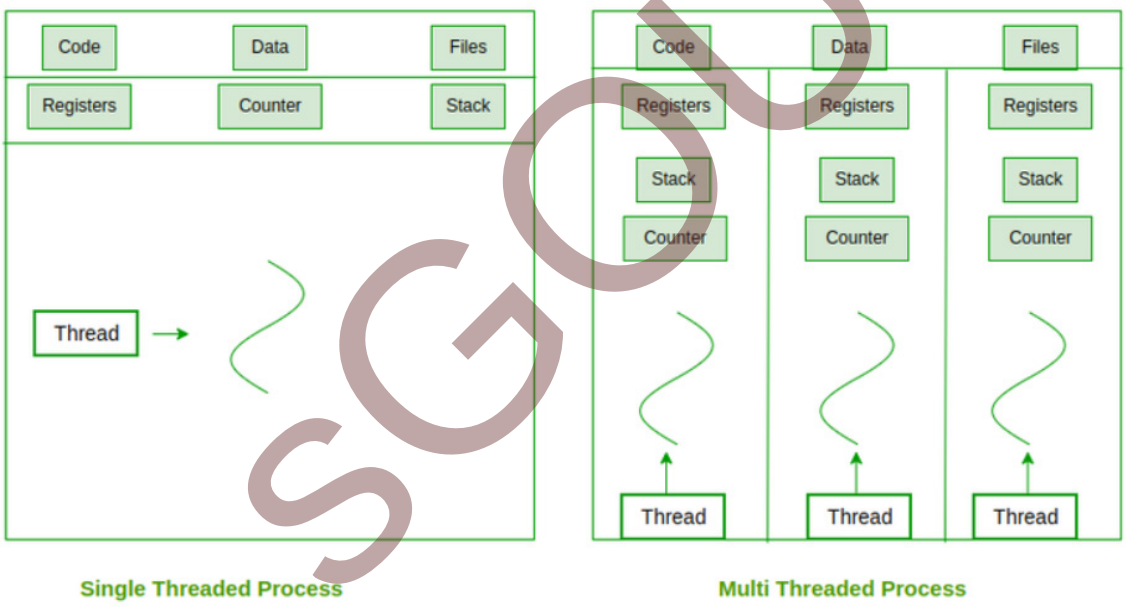


Fig 1.2.4 Single thread and Multiple thread process

Components of a Thread

Each thread consists of the following essential components:

- **Stack Space:** Stores temporary data such as local variables, function calls, and return addresses specific to the thread.
- **Register Set:** Holds intermediate data and values required during execution.
- **Program Counter:** Indicates the address of the next instruction to be executed by the thread.

SGOU

1.2.5.1 Types of Threads

1. User-Level Threads (ULTs)

User-level threads are managed by the application itself, and the operating system is not aware of them. All tasks such as creation and scheduling are handled within the program, making them faster and easier to manage. For example, consider a group of students working on a project where they divide the work among themselves without the teacher's involvement. This makes the work quick, but if one student stops or faces a problem, the entire group may be affected. Similarly, in user-level threads, if one thread is blocked, the whole process may stop, and they cannot fully use multiple processors.

2. Kernel-Level Threads (KLTs)

Kernel-level threads are managed by the operating system, which controls their execution. The OS schedules each thread separately, allowing better performance and efficient use of system resources. For example, in a bank, a manager assigns customers to different counters. If one counter is busy, the manager sends another customer to a free counter, ensuring smooth service. In the same way, if one kernel-level thread is blocked, the operating system can run another thread from the same process. This makes kernel-level threads more reliable and efficient, although they are slightly slower due to system involvement.

1.2.5.2 Process v/s Threads

Table 1.2.1 Comparison of Process and Threads

Feature	Process	Thread
Definition	A process is a program in execution	A thread is a unit of execution within a process
Nature	Heavyweight	Lightweight
Memory	Each process has its own separate memory space	Threads share the memory space of the process
Resource Usage	Requires more system resources	Requires fewer system resources
Creation Time	Process creation is time-consuming	Thread creation is faster
Independence	Processes are independent of each other	Threads are dependent on the parent process
Execution	Executes independently	Executes as part of a process

SGOU

1.2.6 Process Scheduling

In a multiprogramming operating system, multiple processes reside in memory at the same time, each competing for CPU access. Since only one process can use the CPU at a given moment (in a single-core system), there must be a mechanism to decide which process should be executed next. This mechanism is known as process scheduling.

Process scheduling refers to the method by which the operating system selects one process from the ready queue and allocates the CPU to it for execution. The ready queue contains all processes that are prepared to execute but are waiting for CPU availability. By selecting processes in an organized manner, process scheduling ensures efficient CPU utilization and smooth execution of multiple processes.

1.2.6.1 Process Scheduling Queues

To manage multiple processes efficiently, the operating system organizes processes into different queues based on their current state. These queues help the system keep track of processes and decide which process should be executed next. Understanding these queues is essential for clearly understanding process scheduling.

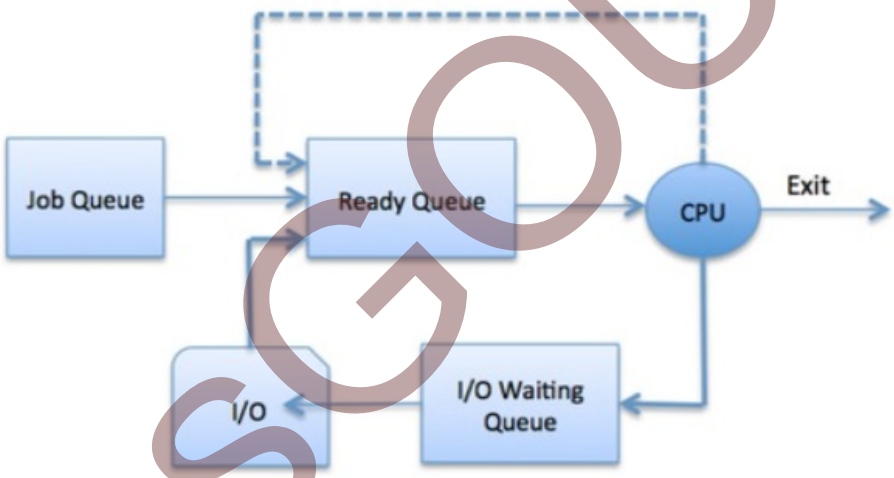


Fig 1.2.5 Process Scheduling Queues

Types of Process Scheduling Queues

1. Job Queue

The job queue contains all the processes that are present in the system. It includes every process that has been created, whether it is in memory or waiting to be loaded into memory. This queue represents the entire set of processes submitted to the operating system.

2. Ready Queue

The ready queue contains all processes that are loaded into main memory and are ready to execute but are waiting for CPU allocation. The operating system selects a process

SGOU

from this queue and assigns the CPU to it. The ready queue plays a central role in process scheduling.

3. Waiting Queue (Device Queue)

The waiting queue, also known as the device queue, contains processes that are waiting for input/output operations or some event to occur. For example, a process waiting for data from a disk or user input will be placed in this queue. Once the required operation is completed, the process moves back to the ready queue.

Working of Scheduling Queues

The working of scheduling queues describes how a process moves through different queues during its lifecycle under the control of the scheduler, which is the component of the operating system responsible for selecting processes for execution. Initially, when a process is created, it enters the job queue. Once it is loaded into memory and becomes ready for execution, it moves to the ready queue, where it waits for CPU allocation. The scheduler then selects one process from the ready queue and assigns the CPU to it, moving it to the running state. If the process requires an input/output operation or needs to wait for an event, it is moved to the waiting queue. After the completion of the required operation, the process returns to the ready queue and again waits for CPU allocation. This continuous movement of processes between different queues ensures efficient CPU utilization and smooth execution of multiple processes in the system.

1.2.7 Scheduler

Now that we have understood how the operating system performs process scheduling, it is important to identify the component responsible for this task. In an operating system, this responsibility is handled by the scheduler, which determines which process should be executed and when it should be given access to system resources.

To better understand this concept, consider the example of an Audio-Visual Room (AVR) in a university. A schedule is maintained to indicate the time slots during which the AVR is reserved or available. Certain departments or university authorities may be given priority on specific days, while others must wait for their turn. This organized allocation ensures effective utilization of the resource.

Similarly, in an operating system, the scheduler selects processes for execution and allocates resources based on predefined criteria. It works along with process queues to manage and control access to system resources efficiently.

1.2.7.1 Types of Schedulers

Now, let us discuss the different types of schedulers used in an operating system. There are three main types of schedulers:

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

SGOU

1. Long-Term Scheduler

The long-term scheduler, also known as the job scheduler, is responsible for selecting processes from the pool of jobs stored on secondary storage and loading them into main memory for execution. It operates less frequently compared to other schedulers. The primary function of the long-term scheduler is to control the degree of multiprogramming, that is, the number of processes present in memory at a given time. By carefully selecting processes, it ensures a balanced mix of CPU-bound and I/O-bound processes, thereby improving overall system performance.

2. Short-Term Scheduler

The short-term scheduler, also referred to as the CPU scheduler, is responsible for selecting one process from the ready queue and allocating the CPU to it for execution. It operates very frequently, as it is invoked whenever the CPU becomes available or requires a new process to execute. Due to its frequent operation, the short-term scheduler must be fast and efficient. It plays a critical role in determining the performance and responsiveness of the system.

3. Medium-Term Scheduler

The medium-term scheduler is responsible for process swapping. When a running process makes an I/O request, it may enter a suspended state and cannot continue execution. To free memory, such processes are temporarily moved from main memory to secondary storage; this is called swapping. The medium-term scheduler manages this process by removing and later bringing the process back into memory for execution. It operates between the long-term and short-term schedulers and helps in controlling system load and improving performance.

1.2.8 Process Scheduling Algorithms

At this point, it is important to understand how the scheduler actually performs process scheduling. The scheduler uses different scheduling algorithms to decide which process should be allocated to the CPU. These algorithms define the rules for selecting processes from the ready queue and determine the order of execution.

To understand this concept, consider a printer in a computer lab. Many students send print requests, but only one document can be printed at a time. The way in which these print jobs are selected and processed can be compared to scheduling algorithms used by the operating system.

Scheduling algorithms are broadly classified into two types:

1. Non-Preemptive Algorithms
2. Preemptive Algorithms

Non-Preemptive Scheduling Algorithms

In non-preemptive scheduling, once a process starts executing, it continues until it

SGOU

completes its execution or voluntarily leaves the CPU. The scheduler does not interrupt the process in between.

Using the printer example, once a document starts printing, it will continue until the entire document is printed, even if another urgent request arrives.

Examples of non-preemptive scheduling algorithms include:

- First Come First Serve (FCFS)
- Shortest Job First (SJF)

Preemptive Scheduling Algorithms

In preemptive scheduling, the operating system can interrupt a currently running process and allocate the CPU to another process, usually based on priority or urgency.

Examples of preemptive scheduling algorithms include:

- Shortest Remaining Time First (SRTF)
- Round Robin (RR)

1.2.8.1 Scheduling Criteria

In an operating system, different scheduling algorithms are used to decide which process should run first. However, all algorithms do not give the same results in every situation. So, we need some measures to check and compare how well these algorithms perform. These measures are called scheduling criteria. The main scheduling criteria are:

1. CPU Utilization
2. Throughput
3. Turnaround Time
4. Waiting Time
5. Response Time

1. CPU Utilization

CPU utilization refers to how much time the CPU is actively working. The goal is to keep the CPU busy as much as possible. For example, in a supermarket, if the billing counter is continuously serving customers without being idle, it shows good utilization. Similarly, higher CPU usage indicates better system performance.

2. Throughput

Throughput is the number of processes completed in a given period of time. It measures how much work the system is doing. For example, in a supermarket, if many customers are billed within an hour, the throughput is high. Higher throughput means the system is efficient.

SGOU

3. Turnaround Time (TAT)

Turnaround time is the total time taken from when a process arrives in the system until it is completed. For example, if a customer enters the billing queue at 10:00 AM and leaves at 10:20 AM, the turnaround time is 20 minutes. It includes waiting time as well as the actual processing time.

The formula is: **TAT = CT – AT**

4. Waiting Time (WT)

Waiting time is the amount of time a process spends waiting in the ready queue before execution. For example, if a customer arrives at 10:00 AM but starts billing at 10:15 AM, then the waiting time is 15 minutes. It does not include the actual processing time.

The formula is: **WT = TAT – BT**

5. Response Time (RT)

Response time is the time taken from when a process arrives until it starts responding for the first time. For example, if a customer arrives and the cashier starts scanning their items after 2 minutes, then the response time is 2 minutes. It is important in systems where quick response is needed.

1.2.9 Non-Preemptive Scheduling Algorithms

A non-preemptive scheduling algorithm is one in which, once a process starts executing, it continues to run until it completes its execution or voluntarily leaves the CPU. In other words, the process is not interrupted while it is running.

Examples of non-preemptive scheduling algorithms include:

- First Come First Serve (FCFS)
- Shortest Job First (SJF)

Let us now discuss each of these non-preemptive scheduling algorithms in detail.

1.2.9.1 First Come First Serve (FCFS)

The First Come First Serve (FCFS) scheduling algorithm is a non-preemptive scheduling method in which the process that arrives first in the ready queue is allocated the CPU first. Once a process starts execution, it continues until it completes its execution without any interruption.

In this algorithm, all processes are arranged in a queue based on their arrival time. The process at the front of the queue is selected and executed first, while the remaining processes wait for their turn. If a running process gets blocked, the next process in the queue is executed. When the blocked process becomes ready again, it is placed at the end of the queue.

A simple real-world example of FCFS is a queue at a bus stop, where passengers board

SGOU

the bus in the order they arrive. The first person in the queue gets the service first, followed by others in sequence.

Example:

We have the following processes with Arrival Time (AT) and Burst Time (BT):

Process(P)	AT	BT
P1	0	4
P2	2	5
P3	3	3
P4	4	2

Step 1: Order of Execution (FCFS Rule)

In FCFS scheduling, processes are executed in the order of their arrival time.

So the execution order will be:

P1 → P2 → P3 → P4

Step 2: Gantt Chart

A Gantt chart is a graphical representation used in CPU scheduling to show the execution order of processes along with time. It displays how processes are allocated to the CPU from the start time to the completion time.

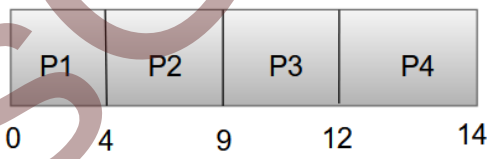


Fig 1.2.6 Gantt Chart

Let us understand how the above Gantt chart is obtained.

The Completion Time (CT) of a process is the time at which the process finishes execution. In FCFS scheduling, each process starts only after the previous process completes.

- In the case of P1, it arrives at time 0 and starts immediately. Its Burst Time (BT) is 4. Therefore, CT of P1 = 0 + 4 = 4
- In the case of P2, it starts only after P1 completes. So, its start time is the CT of P1, which is 4. Therefore, CT of P2 = 4 + 5 = 9

SGOU

- In the case of P3, it starts after P2 completes. So, its start time is the CT of P2, which is 9. Therefore, CT of P3 = 9 + 3 = 12

In the case of P4, it starts after P3 completes. So, its start time is the CT of P3, which is 12. Therefore, CT of P4 = 12 + 2 = 14

Table 1.2.3 Example of FCFS with CT

Process(P)	AT	BT	CT
P1	0	4	4
P2	2	5	9
P3	3	3	12
P4	4	2	14

Step 3: Calculation of Turnaround Time (TAT)

As discussed earlier, Turnaround Time (TAT) is calculated using the formula:

TAT = CT – AT

For example, in the case of P1, TAT = 4 – 0 = 4

in the case of P2, TAT = 9 – 2 = 7

in the case of P3, TAT = 12 – 3 = 9

in the case of P4, TAT = 14 – 4 = 10

Table 1.2.4 Example of FCFS with CT & TAT

Process(P)	AT	BT	CT	TAT
P1	0	4	4	4
P2	2	5	9	7
P3	3	3	12	9
P4	4	2	14	10

Step 4: Calculation of Waiting Time (WT)

As discussed earlier, Waiting Time (WT) is calculated using the formula:

WT = TAT – BT

For example, in the case of P1, WT = 4 – 4 = 0

SGOU

in the case of P2, WT = 7 – 5 = 2

in the case of P3, WT = 9 – 3 = 6

in the case of P4, WT = 10 – 2 = 8

Let us tabulate WT so that it will be easier to understand.

Table 1.2.5 Example of FCFS with CT, TAT and WT

Process(P)	AT	BT	CT	TAT	WT
P1	0	4	4	4	0
P2	2	5	9	7	2
P3	3	3	12	9	6
P4	4	2	14	10	8

Step 5: Calculation of Average TAT and Average WT

1. Average Turnaround Time (TAT) is calculated as:

$$\text{Average TAT} = \frac{Sum\ of\ all\ TAT}{Number\ of\ Process}$$

$$= \frac{4+7+9+10}{4}$$

$$= \frac{30}{4} = 7.5$$

2. Average Waiting Time (WT) is calculated as:

$$\text{Average WT} = \frac{Sum\ of\ all\ WT}{Number\ of\ Process}$$

$$= \frac{0+2+6+8}{4}$$

$$= \frac{16}{4} = 4$$

1.2.9.2 Shortest Job First (SJF)

Non-preemptive Shortest Job First (SJF) is a CPU scheduling algorithm in which the process with the smallest Burst Time (BT) among the available processes is selected for execution. Once a process starts execution, it continues until completion without interruption.

SGOU

Let us take a real-life example to understand this. SJF scheduling can be compared to a printing center, where multiple users submit documents for printing. The user with the least number of pages is given priority, followed by users with larger documents.

In the case of SJF, when several processes are waiting in the ready queue, the scheduler selects the process with the smallest Burst Time (BT) first. That is, the allocation of CPU to a process is based on the shortest next CPU burst. Let us consider the following example in Table 1.2.6.

Table 1.2.6 Example of SJF

Process (P)	AT	BT
P1	1	7
P2	2	5
P3	3	1
P4	4	2

Figure 1.2.7 shows the Gantt chart for table 1.2.6

Step 1: Order of Execution

In Non-Preemptive SJF, the process with the smallest Burst Time (BT) among the available processes is selected for execution. At time 1, only P1 is available, so it starts execution. Since the scheduling is non-preemptive, P1 runs until completion and finishes at time 8. After that, the available processes are P2, P3, and P4. Among them, the process with the smallest burst time is P3 (BT = 1), so it is executed next. After completing P3, the remaining processes are P2 and P4, among which P4 has the smaller burst time (BT = 2), so it is executed next.

Thus, the execution order is P1 → P3 → P4 → P2.

Step 2: Gantt chart

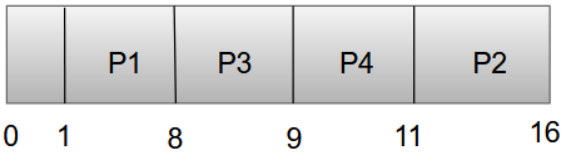


Fig 1.2.7 Gantt chart

It can be observed that at time 0, no process is present in the Gantt chart because no process has arrived at that time. At time 1, process P1 arrives, and hence it is included in the Gantt chart.

In SJF scheduling, the selection criterion is the Burst Time (BT). The scheduler always chooses the process with the smallest BT from the available processes.

SGOU

As seen, the BT of P1 is 7, and this is reflected in the Gantt chart. The Completion Time (CT) of P1 is obtained by adding its Arrival Time (AT) and Burst Time (BT), which gives: CT of P1 = 1 + 7 = 8.

Next, among the available processes, the one with the smallest BT is selected. Here, P3 has the least BT of 1, so it is scheduled next.

CT of P3 = 8 + 1 = 9.

After that, the process with the next smallest BT is P4, with a BT of 2. Hence, P4 is executed next.

CT of P4 = 9 + 2 = 11.

Then, P2 is selected as it has the next smallest BT of 5.

CT of P2 = 11 + 5 = 16.

Thus, the Completion Time (CT) for each process is determined based on the order of execution in the Gantt chart.

In the next stage, Turnaround Time (TAT) and Waiting Time (WT) are calculated as discussed earlier.

Table 1.2.7 Example of SJF with CT, TAT, WT

Process (P)	AT	BT	CT	TAT	WT
P1	1	7	8	7	0
P2	2	5	16	14	9
P3	3	1	9	6	5
P4	4	2	11	7	5

Average TAT = (7 + 14 + 6 + 7)/4 = 8.5

Average WT = (0 + 9 + 5 + 5)/4 = 4.75

1.2.10 Preemptive Scheduling Algorithms

A preemptive scheduling algorithm is a CPU scheduling technique in which a running process can be interrupted (preempted) and the CPU can be allocated to another process.

In this type of scheduling, the CPU is not assigned to a process until completion. Instead, the scheduler may switch the CPU to another process based on certain criteria such as priority or remaining execution time.

Preemptive scheduling improves response time, especially in systems where multiple processes need quick attention.

The commonly used preemptive scheduling algorithms are:

SGOU

- Shortest Remaining Time (SRT)
- Round Robin (RR)

1.2.10.1 Shortest Remaining Time First(SRT)

Shortest Remaining Time First (SRT) is the preemptive version of Shortest Job First (SJF). In this algorithm, the process with the smallest remaining burst time is selected for execution.

If a new process arrives with a shorter remaining time than the currently running process, the current process is preempted, and the new process is executed.

Let us consider the following example in Table 1.2.8

Table 1.2.8 Example of SRTF

Process(P)	AT	BT
P1	0	7
P2	1	5
P3	2	3
P4	3	1

In SRTF, the process with the smallest remaining time is selected at every instant.

At time 0, only P1 is available, so it starts execution.

At time 1, P2 arrives. Now compare remaining time:

P1 (6), P2 (5) → P2 is smaller, so P1 is preempted and P2 executes.

At time 2, P3 arrives. Compare:

P2 (4), P3 (3) → P3 is smaller, so P2 is preempted and P3 executes.

At time 3, P4 arrives. Compare:

P3 (2), P4 (1) → P4 is smaller, so P3 is preempted and P4 executes.

After P4 completes, the remaining processes are compared at each step and the smallest remaining time is selected.

Figure 1.2.8 shows the Gantt chart for Table 1.2.8

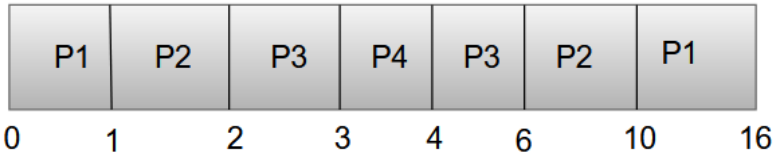


Fig. 1.2.8 Gantt chart

SGOU

Completion Time is taken when a process finishes its last execution.

- P4 completes at time 4
- P3 completes at time 6
- P2 completes at time 10
- P1 completes at time 16

This is represented in the following table 1.2.9.

Table 1.2.9 Example of SRTF with CT

Process(P)	AT	BT	CT
P1	0	7	16
P2	1	5	10
P3	2	3	6
P4	3	1	4

TAT is calculated as CT – AT.

WT is calculated as TAT – BT.

The following Table 1.2.10 shows the TAT and WT calculated.

Table 1.2.10 Example of SRTF with TAT and WT

Process(P)	AT	BT	CT	TAT	WT
P1	0	7	16	16	9
P2	1	5	10	9	4
P3	2	3	6	4	1
P4	3	1	4	1	0

The Average TAT is calculated as dividing the total TAT by no. of process Average

TAT = (19 + 9+ 4 + 1) / 4= 7.5

The Average WT is calculated as dividing the total WT by no. of process Average WT = (9+4+1+0) / 4= 3.5

1.2.10.2 Round Robin (RR)

The Round Robin scheduling algorithm derives its name from the principle of sharing resources equally in a cyclic manner, where each process gets an equal opportunity to

SGOU

use the CPU. It operates on the assumption that all processes are of equal importance.

In this scheduling technique, processes are arranged in a circular queue, and each process is executed for a fixed time interval known as the Time Quantum (TQ). After the time quantum expires, if the process is not completed, it is preempted and moved to the end of the queue, allowing the next process to execute.

The allocation of CPU time is thus shared among all processes in a cyclic order based on the time quantum.

Let us consider a real-life example of a computer lab system, where multiple students are given access to a system for a fixed duration, say 10 minutes (TQ). If a student finishes their work within this time, they leave the system. Otherwise, they must leave after 10 minutes and wait for their next turn. This process continues until all students complete their tasks.

In the following example, there are four processes named P1, P2, P3, P4 and their arrival time and burst time are shown in table 1.2.11. The time quantum of the system is 3 units.

Table 1.2.11 Example of Round Robin

Process(P)	AT	BT
P1	0	5
P2	1	4
P3	2	2
P4	3	1

Time Quantum (TQ) = 3 units

At time 0, P1 starts execution and runs for 3 units (TQ). Remaining BT of P1 = 2.

At time 3, the ready queue contains P2, P3, P4, and P1.

P2 executes next for 3 units. Remaining BT of P2 = 1.

At time 6, P3 executes. Since its BT (2) is less than TQ, it completes execution at time 8.

At time 8, P4 executes. Since its BT (1) is less than TQ, it completes at time 9.

At time 9, P1 executes its remaining 2 units and completes at time 11.

At time 11, P2 executes its remaining 1 unit and completes at time 12.

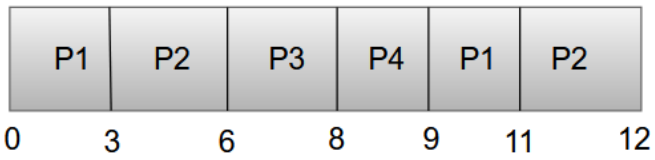


Fig 1.2.9 Gantt Chart

SGOU

Completion Time is taken when a process finishes its last execution.

P3 completes at time 8

P4 completes at time 9

P1 completes at time 11

P2 completes at time 12

Table 1.2.12 Example of Round Robin with CT

Process(P)	AT	BT	CT
P1	0	5	11
P2	1	4	12
P3	2	2	8
P4	3	1	9

Table 1.2.13 Example of Round Robin with CT, TAT and WT

Process(P)	AT	BT	CT	TAT	WT
P1	0	5	11	11	6
P2	1	4	12	11	7
P3	2	2	8	6	4
P4	3	1	9	6	5

$$Average\ TAT = \frac{11 + 11 + 6 + 6}{4} = \frac{34}{4} = 8.5$$

$$Average\ WT = \frac{6 + 7 + 4 + 5}{4} = \frac{22}{4} = 5.5$$

SGOU

Summarized Overview

Operating systems manage the execution of programs by converting them into processes and handling their activities efficiently. A process is an active entity with its own memory structure and passes through different states during execution. To manage each process, the system maintains detailed information using structures like the Process Control Block (PCB). For performing multiple tasks simultaneously, threads are used within a process, improving performance and responsiveness. The system also uses scheduling techniques to decide which process should use the CPU at a given time. This involves different queues, types of schedulers, and scheduling algorithms such as FCFS, SJF, SRTF, and Round Robin. Various criteria like CPU utilization, turnaround time, and waiting time are used to evaluate performance. Together, these concepts explain how multiple tasks are handled smoothly and efficiently in a computer system.

Assignments

1. Explain the difference between a program and a process with suitable examples.
2. Describe the lifecycle of a process and explain the different process states with a diagram.
3. Explain the memory structure of a process with neat illustration.
4. What is a Process Control Block (PCB)? Explain its components in detail.
5. Explain the concept of threads and discuss their need in modern operating systems.
6. Differentiate between user-level threads and kernel-level threads with examples.
7. Explain process scheduling and describe the different types of scheduling queues.
8. Describe the types of schedulers and explain their roles in process management.

SGOU

9. Explain FCFS scheduling algorithm with a suitable example and calculate TAT and WT.

10. Compare preemptive and non-preemptive scheduling algorithms with examples.

Reference

1. Arpaci-Dusseau, R. H., & Arpaci-Dusseau, A. C. (2018). *Operating systems: Three easy pieces*. Arpaci-Dusseau Books.

2. Stallings, W. (2018). *Operating systems: Internals and design principles* (9th ed.). Pearson.

3. Bovet, D. P., & Cesati, M. (2005). *Understanding the Linux kernel* (3rd ed.). O'Reilly Media.

Suggested Reading

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts* (10th ed.). Wiley.

2. Tanenbaum, A. S., & Bos, H. (2015). *Modern operating systems* (4th ed.). Pearson.

3. Dhamdhere, D. M. (2017). *Operating systems: A concept-based approach* (3rd ed.). McGraw Hill Education.

SGOU

Space for Learner Engagement for Objective Questions

Learners are encouraged to develop objective questions based on the content in the paragraph as a sign of their comprehension of the content. The Learners may reflect on the recap bullets and relate their understanding with the narrative in order to frame objective questions from the given text. The University expects that 1 - 2 questions are developed for each paragraph. The space given below can be used for listing the questions.

SGOU

3 UNIT Inter-process Communication and Synchronization

Learning Outcomes

After completion of this unit, the learner will be able to:

- define what Inter Process Communication (IPC) is and why it is important in operating systems.
- explain how IPC methods like shared memory and message passing work with simple examples.
- describe common IPC problems such as race conditions, deadlock, and starvation and their effects.
- apply basic synchronization methods to solve problems like Producer Consumer and Dining Philosophers.
- understand how tools like semaphores, monitors, and critical sections help manage shared resources.

SGOU

Background

Before studying this topic, a learner should have a basic understanding of operating system concepts, including processes and threads. Knowledge of basic programming (preferably in C or Java) is helpful to understand implementation aspects. Familiarity with memory management and fundamental algorithms will further support better understanding of synchronization problems and their solutions.

Studying Inter Process Communication (IPC) and synchronization is essential because modern operating systems run multiple processes simultaneously that must communicate and share resources efficiently. Without proper coordination, systems can face issues like data corruption, race conditions, or system crashes.

These topics help us understand how processes interact safely, especially in real-world systems such as banking applications, where multiple users may access the same account at the same time. For example, if two users withdraw money simultaneously without synchronization, the final balance may become incorrect. IPC mechanisms such as shared memory and message passing enable controlled data exchange between processes. Synchronization tools like semaphores and monitors ensure that only one process accesses critical resources at a time. Learning these concepts is also important for designing scalable and high-performance systems such as web servers, databases, and distributed applications.

Keywords

process synchronization, shared memory, message passing, race condition, deadlock, semaphore, critical section

Discussion

SGOU

1.3.1 Inter Process Communication (IPC)

Inter Process Communication, commonly known as IPC, refers to the mechanism that enables different processes to exchange information and coordinate their activities while executing. Since each process operates in its own separate memory space, IPC provides well-defined and controlled techniques for communication and data sharing. This ensures that processes can collaborate effectively and safely within an operating system.

IPC plays a crucial role in synchronizing process activities, facilitating data sharing, and preventing conflicts when multiple processes access shared resources.

There are two primary approaches to IPC: shared memory and message passing. Most operating systems support one or both of these communication methods.

Example: A typical example of IPC can be seen in a bank ATM system, where one process handles card and PIN verification, another process checks the account balance, and a third process manages cash dispensing. These processes interact and coordinate with each other to successfully complete the transaction.

1.3.1.1 Shared Memory

Interprocess communication through shared memory involves multiple processes accessing a common memory space. The effectiveness of this approach largely depends

on how the programmer designs and manages the shared variables. Using shared memory, a process can read data created by another process or write information that other processes can subsequently access.

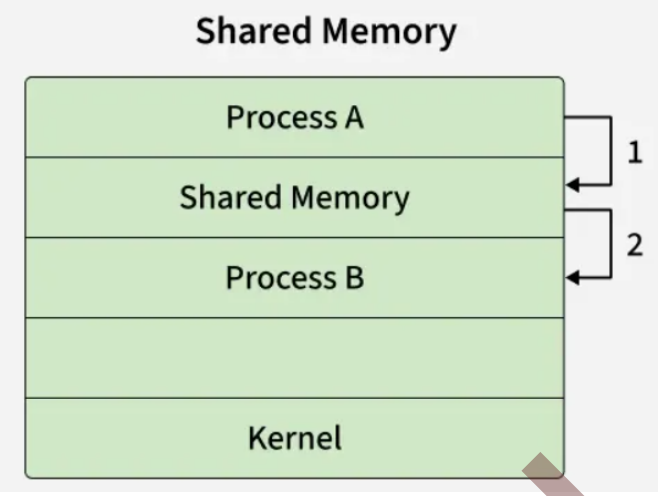


Fig. 1.3.1 Shared memory

In the shared memory model described above, the operating system kernel allocates a common memory region that can be accessed by multiple processes. Process A writes data into this shared memory area in the first step, and Process B subsequently reads the same data directly from that region in the second step. Because both processes operate on the same memory segment, this approach offers high speed and efficiency. However, it requires proper synchronization mechanisms, such as semaphores, to prevent conflicts when multiple processes attempt to read or write at the same time.

For example, it is similar to multiple users editing a shared Google document simultaneously, where all participants access and modify the same content in real time.

1.3.1.2 Message Passing

Message Passing is a communication mechanism in which processes exchange data by sending and receiving messages. In this approach, one process transmits a message while another process accepts it, enabling information sharing between them. This form of communication can be implemented using techniques such as sockets, message queues, or pipes.

In the message passing model, processes communicate by exchanging messages through the operating system kernel rather than sharing memory directly. In the first step, Process A sends a message to the kernel, and in the next step, the kernel forwards that message to Process B. Since there is no shared memory space between the processes, communication takes place using system calls such as send() and receive().

This approach is generally simpler and more secure because it avoids issues like data corruption or overwriting that can occur in shared memory. However, it introduces additional overhead due to the involvement of the kernel in every communication step.

For example, it is similar to participants sending messages in a group chat, where each

SGOU

message passes through a server before reaching others, rather than directly sharing information.

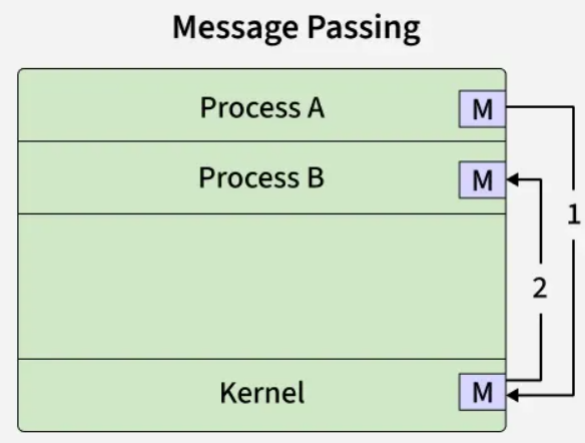


Fig. 1.3.2 Message passing

1.3.2 Problems in Inter-Process Communication (IPC)

Inter-Process Communication involves multiple processes interacting and often sharing system resources, which introduces several challenges. If synchronization is not handled properly, it can lead to **race conditions**, where processes compete to access shared data and produce unpredictable results. Issues such as **deadlock** may arise when processes wait indefinitely for resources held by each other, while **starvation** occurs when a process is continuously denied access to resources. Shared data can also suffer from **inconsistency** if updates are not properly controlled. In addition, IPC mechanisms introduce **communication overhead** due to context switching and kernel involvement, which can affect performance. There are also **security concerns**, as unauthorized processes might access sensitive data if proper controls are not in place. Furthermore, as the number of processes increases, managing communication becomes more complex, leading to **scalability issues**.

Some classical problems in IPC that illustrate these challenges are explained below:

1.3.2.1 Dining Philosophers Problem

The Dining Philosophers Problem is a classical problem in operating systems used to illustrate challenges in resource sharing, particularly deadlock and starvation. Imagine five philosophers seated around a circular table. Between each pair of philosophers lies a single fork, making a total of five forks. To eat, a philosopher must acquire two forks; one on the left and one on the right. However, each philosopher can pick up only one fork at a time.

A critical issue arises when all philosophers attempt to eat simultaneously. If each philosopher picks up the fork on their left at the same time, then every fork is held by one philosopher, and no one can acquire the second fork needed to eat. As a result, all philosophers remain stuck waiting indefinitely, this situation is known as deadlock.

SGOU

Another related issue is starvation, where a philosopher may repeatedly fail to acquire both forks because others keep taking them first. Even though the system is not completely blocked, some philosophers may never get a chance to eat.

Example Scenario

Consider philosophers P1, P2, P3, P4, and P5 sitting around the table:

- Each philosopher picks up the fork on their left simultaneously.
- Now, each philosopher is waiting for the fork on their right, but that fork is already held by their neighbor.
- Since no philosopher releases their fork voluntarily, the system enters a deadlock state, and no one can proceed.

Alternatively, suppose P1 frequently manages to acquire both forks and eat, while P3 is always delayed due to timing. In this case, P3 may starve, as it never gets access to both forks.

This problem highlights the importance of proper synchronization mechanisms in operating systems, such as:

- Avoiding circular wait conditions
- Using resource ordering
- Implementing algorithms like the waiter solution or semaphores

The Dining Philosophers Problem provides a simple yet powerful way to understand how improper resource allocation can lead to system inefficiencies and failures.

Solution:

This problem can be solved using synchronization techniques such as semaphores or monitors to regulate access to the forks. One approach is to allow only a limited number of philosophers (for example, four out of five) to attempt eating at the same time, ensuring that at least one philosopher can access both forks. Another method is to enforce an ordered strategy for picking up forks, such as always picking the lower-numbered fork first, which prevents circular waiting. These approaches help eliminate deadlock and reduce the possibility of starvation.

1.3.2.2 Producer–Consumer Problem

The Producer–Consumer problem is a classical example in operating systems that highlights the need for process synchronization and the efficient use of a shared buffer. In this model, one or more *producer* processes generate data items and place them into a common storage area (buffer), while one or more *consumer* processes retrieve and use those items. The buffer is typically of fixed size (bounded buffer), which introduces coordination challenges between producers and consumers.

The main issue arises when the buffer reaches its limits. If the buffer becomes full,

SGOU

producers must wait until space is available before adding more data. Similarly, if the buffer becomes empty, consumers must wait until new data is produced. Without proper synchronization mechanisms, this situation can lead to problems such as race conditions, data inconsistency, or even system crashes due to invalid memory access.

To handle this correctly, operating systems use synchronization tools like semaphores, mutex locks, or monitors to ensure that:

- Producers do not write into a full buffer
- Consumers do not read from an empty buffer
- Only one process accesses the shared buffer at a time (mutual exclusion)

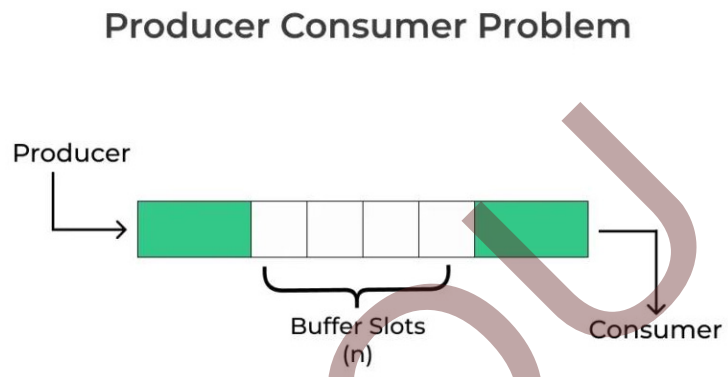


Fig. 1.3.3 Producer Consumer Problem

Example:

Consider a restaurant kitchen scenario:

- The chef acts as the *producer*, preparing dishes.
- The waiter acts as the *consumer*, picking up dishes and serving customers.
- The serving counter acts as the *buffer*, which can hold only a limited number of plates (say 5).

Situation:

- If the counter already has 5 dishes (buffer full), the chef must wait before placing more dishes.
- If there are no dishes on the counter (buffer empty), the waiter must wait until the chef prepares new ones.

This coordination ensures smooth operation without confusion, duplication, or errors.

Technical Example

Assume:

- Buffer size = 3

SGOU

- Producer generates items: A, B, C, D
- Consumer removes items one by one

Execution:

1. Producer adds A → Buffer: [A]
2. Producer adds B → Buffer: [A, B]
3. Producer adds C → Buffer: [A, B, C] (Full)
4. Producer tries to add D → ❌ Must wait
5. Consumer removes A → Buffer: [B, C]
6. Producer resumes and adds D → Buffer: [B, C, D]

The Producer–Consumer problem demonstrates how synchronization ensures correct and efficient cooperation between processes, preventing errors while maximizing resource utilization.

Solution:

To address this, a **mutex (mutual exclusion lock)** is used to ensure that only one process accesses the buffer at a time. Additionally, **counting semaphores** are employed to keep track of the number of empty and filled slots in the buffer. The producer process waits if the buffer is full, while the consumer process waits if the buffer is empty. This coordinated mechanism ensures proper synchronization, prevents data corruption, and maintains consistency in the shared buffer.

1.3.2.3 Readers–Writers Problem

The **Readers–Writers problem** is a classic synchronization problem in operating systems that focuses on **coordinating access to a shared resource** (such as a file, database, or memory). In this scenario, there are two types of processes: **readers**, who only read the data, and **writers**, who modify the data.

A key characteristic of this problem is that **multiple readers can access the resource simultaneously** without causing any inconsistency, since they do not alter the data. However, a **writer must have exclusive access**, meaning that when a writer is writing, no other writer or reader should access the resource at the same time. This ensures data integrity and prevents corruption.

The main challenge lies in designing a system that allows efficient access while maintaining fairness. If priority is always given to readers, writers may be **starved** (waiting indefinitely). On the other hand, if writers are always prioritized, readers may experience unnecessary delays. Therefore, the goal is to achieve a **balanced synchronization mechanism** that prevents both reader and writer starvation.

Example:

SGOU

Consider a **library reading room**:

- **Readers** are students who come to read books.
- **Writers** are staff members who update or rearrange books.

Situation:

- Many students can sit and read books at the same time (**multiple readers allowed**).
- But when a staff member needs to rearrange or update the books (**writer**), the room must be temporarily cleared (**exclusive access required**).

If students keep coming continuously, the staff may never get a chance to update the books (**writer starvation**). Similarly, if updates happen too frequently, students may not get enough time to read (**reader starvation**).

Technical Example

Assume a shared database:

- Readers: R1, R2, R3
- Writer: W1

Execution:

1. R1 and R2 access the database simultaneously → ☑ Allowed
2. R3 joins → ☑ Still allowed
3. W1 requests access → ✗ Must wait until all readers finish
4. After R1, R2, R3 exit → W1 gets exclusive access → ☑ Writes data
5. While W1 is writing → ✗ No readers or writers allowed
6. After W1 completes → Readers can access again

Variations of the Problem

- **Reader-priority solution** → Readers are favored, writers may starve
- **Writer-priority solution** → Writers are favored, readers may starve
- **Fair solution** → Balanced approach ensuring no starvation

The Readers–Writers problem demonstrates the importance of **controlled concurrent access**, ensuring **data consistency, fairness, and efficiency** when multiple processes interact with shared resources.

Solution:

This issue can be managed using **reader–writer locks** or semaphores. The system

SGOU

allows multiple readers to access the data at the same time as long as no writer is writing. When a writer needs access, it must wait until all readers have finished, and during writing, no readers are allowed to access the data. Priority mechanisms can also be implemented to ensure fairness, such as giving preference to writers or readers depending on system requirements. This prevents starvation and ensures efficient data access.

1.3.2.4 Sleeping Barber Problem

The Sleeping Barber problem is a classic synchronization problem in operating systems that demonstrates process coordination, resource allocation, and efficient handling of waiting queues. It models a barber shop where a single barber serves customers, and a limited number of chairs are available for waiting.

In this scenario, the barber sleeps when there are no customers. When a customer arrives, the behavior depends on the shop's state. If the barber is idle, the customer wakes the barber and gets service immediately. If the barber is busy, the customer checks for an available waiting chair. If a chair is free, the customer sits and waits; otherwise, if all chairs are occupied, the customer leaves without receiving service.

The main challenge is to synchronize the actions of the barber and customers so that:

- Customers are served efficiently without confusion
- The barber does not remain idle when customers are waiting
- No two customers are served at the same time
- The system avoids unnecessary waiting, conflicts, or resource wastage

This requires proper coordination mechanisms (like semaphores or locks) to manage access to shared resources such as chairs and the barber.

Example (Real-Life Scenario)

Consider a small barber shop with:

- 1 barber
- 3 waiting chairs

Situation:

1. No customers → Barber is sleeping
2. Customer C1 arrives → Wakes the barber and gets a haircut
3. While C1 is being served:
    - C2 arrives → Sits in waiting chair
    - C3 arrives → Sits in waiting chair

SGOU

- C4 arrives → Sits in last chair

4. C5 arrives → ❌ No chairs available → Leaves

After finishing C1:

- Barber calls the next waiting customer (C2), and the process continues

Technical Flow

- If no customers → Barber sleeps
- If customer arrives and barber is free → Immediate service
- If barber is busy and chairs available → Customer waits
- If chairs are full → Customer leaves

The Sleeping Barber problem highlights how proper synchronization ensures efficient use of limited resources, prevents idle time, and avoids overcrowding or conflicts in systems where demand and service must be carefully balanced.

Solution:

This problem is typically solved using semaphores to coordinate actions between the barber and customers. A semaphore is used to track the number of waiting customers, while another ensures mutual exclusion when accessing shared variables such as the number of available chairs. The barber goes to sleep when there are no customers and wakes up when a customer arrives. Customers either wait if chairs are available or leave if the waiting area is full. This solution ensures smooth coordination, prevents deadlock, and maintains proper synchronization between processes.

Overall, these classical problems highlight the importance of synchronization, proper resource allocation, and efficient coordination in IPC systems.

1.3.3 Critical Section in Synchronization

A critical section refers to a portion of a program where shared resources such as memory, files, or variables are accessed by multiple processes or threads. Since simultaneous access can lead to issues like race conditions and data inconsistency, it is essential to ensure that only one process or thread executes this section at any given time.

To achieve this, various **synchronization mechanisms** such as locks, semaphores, or monitors are used. These techniques enforce **mutual exclusion**, allowing safe and orderly access to shared resources. As a result, operations performed within the critical section remain **consistent, reliable, and predictable**, preventing conflicts between concurrent processes.

Key Points

- A critical section is the part of a program where **shared resources are accessed**.

SGOU

- **Only one process or thread** should execute it at a time.
- **Synchronization techniques** ensure safe and controlled access.

Example

Consider a shared bank account balance:

- Process A and Process B both try to **withdraw ₹100** from a balance of ₹1000 at the same time.
- Without synchronization, both may read the same balance and update incorrectly → ❌ wrong result
- With a critical section:
  - One process completes the transaction first
  - Then the other proceeds → ☑ correct balance maintained

The concept of a critical section is fundamental to **process synchronization**, ensuring that shared data is handled correctly even when multiple processes run concurrently.

1.3.3.1 Components of a Critical Section

1. Entry Section

This is the part where a process **requests permission to enter** the critical section. Synchronization mechanisms (such as mutex locks or semaphores) are used here to ensure that only one process can proceed at a time.

2. Critical Section

This is the **core part of the program** where shared resources (like variables, memory, or files) are accessed or modified.

Only one process is allowed to execute this section at any given moment.

3. Exit Section

In this section, the process **releases the lock or synchronization control**, allowing other waiting processes to enter the critical section.

4. Remainder Section

This includes the **rest of the program** that does not involve shared resources and can be executed freely without synchronization.

1.3.3.2 Shared Resources and Race Conditions

- **Shared resources** include memory, global variables, files, and databases that can be accessed by multiple processes.
- A **race condition** occurs when two or more processes try to modify shared data simultaneously, leading to unpredictable or incorrect results.

SGOU

Example:

Consider a bank account with a balance of ₹1000.

- Process A withdraws ₹200
- Process B withdraws ₹300 at the same time

Without synchronization, both may read the same initial balance and update it incorrectly, resulting in an **incorrect final balance**.

Step-by-step representation

```
do {
  // Entry Section
  acquire_lock();
  // Critical Section
  // Access or modify shared resource
  // Exit Section
  release_lock();
  // Remainder Section
  // Other operations
} while (true);
```

Requirements of a Good Solution

A proper critical section solution must satisfy:

- **Correctness** → Ensures shared data remains accurate and consistent
- **Efficiency** → Minimizes waiting time and improves CPU utilization
- **Fairness** → Guarantees that no process is indefinitely delayed (no starvation)

The structure of a critical section provides a **systematic approach to managing concurrent processes**, ensuring that shared resources are accessed safely, efficiently, and fairly.

1.3.3.3 Requirements of Critical Section Solutions

A correct solution to the critical section problem must satisfy three essential conditions to ensure safe and efficient access to shared resources:

1. Mutual Exclusion

At any given time, only one process or thread should be allowed to execute inside

SGOU

the critical section. This prevents conflicts by ensuring that shared resources are not accessed or modified simultaneously by multiple processes, thereby avoiding data inconsistency.

2. Progress

If no process is currently executing in the critical section and multiple processes are waiting to enter, the system must select one of the waiting processes within a finite time. The decision should not be indefinitely delayed, ensuring that the system continues to function smoothly without unnecessary blocking.

3. Bounded Waiting

There should be a limit on the number of times other processes can enter the critical section before a waiting process gets its turn. This condition prevents starvation, ensuring fairness so that every process eventually gets access.

1.3.3.4 Solution to the Critical Section Problem

A simple and widely used approach to handle critical sections is through locking mechanisms:

```
acquireLock();

// Critical Section (access shared resource)

releaseLock();
```

Before entering the critical section, a process must acquire a lock, which ensures exclusive access. Since only one process can hold the lock at a time, other processes must wait until it is released. Locks can be implemented using techniques such as mutexes, semaphores, or monitors.

1.3.3.5 Real-World Examples of Critical Sections

1. Banking Systems (ATM or Online Transactions)

- Critical Section: Updating account balance during deposits or withdrawals
- Without synchronization: Simultaneous transactions may lead to an incorrect balance due to race conditions

2. Ticket Booking Systems (Airlines, Movies, Trains)

- Critical Section: Allocating the last available seat
- Without synchronization: Multiple users may book the same seat, resulting in overbooking

3. Print Spooler in Networked Printers

- Critical Section: Managing the printer queue
- Without synchronization: Print jobs may overlap, get mixed, or be skipped

SGOU

4. Shared Document Editing (e.g., collaborative tools)

- Critical Section: Writing or saving changes to the document
- Without synchronization: Concurrent edits may cause conflicts or data loss

1.3.4 Semaphores in Process Synchronization

A **semaphore** is a fundamental synchronization mechanism used in operating systems to regulate access to **shared resources** in environments where multiple processes or threads execute concurrently. It is typically represented as an **integer variable** whose value is modified through atomic operations such as *wait()* and *signal()*. These operations ensure that changes occur safely without interference from other processes.

Semaphores play a crucial role in **preventing race conditions** and maintaining proper coordination among concurrent processes. By controlling how and when processes enter critical sections, they help ensure orderly execution and data consistency.

Key characteristics of semaphores include:

- They **control access to critical sections**, allowing only permitted processes to proceed.
- They maintain a **count of available resources**, which determines how many processes can access the resource simultaneously.
- They help enforce **mutual exclusion**, ensuring that shared data is not accessed concurrently in an unsafe manner.
- They have the ability to **block processes when resources are unavailable and wake them up when access becomes possible**.
- They are widely applied in **concurrent and parallel programming** for managing synchronization efficiently.

Semaphores provide a structured way to coordinate process execution while safeguarding shared resources.

```
wait(S):
    S = S - 1        // take resource
    if S < 0:        // none available
        wait         // go to sleep
```

1.3.4.1 Working of Semaphores

A semaphore operates through two essential atomic operations that control how processes

SGOU

access shared resources. These operations ensure that changes to the semaphore value happen safely without interruption.

wait(S) Operation

The *wait* operation reduces the value of the semaphore. If the value becomes negative or is already zero (depending on implementation), the requesting process is blocked and must wait until the resource becomes available. This operation is used when a process attempts to acquire a resource or enter a critical section.

```
wait(S):
    S = S - 1        // take resource
    if S < 0:        // none available
      wait           // go to sleep
```

signal(S) Operation

The *signal* operation increases the value of the semaphore. If there are processes waiting, one of them is woken up and allowed to proceed. This operation is used when a process releases a resource after completing its task.

```
signal(S):
    S = S - 1        // take resource
    if S <= 0:       // someone waiting
      wake up        // wake one process
```

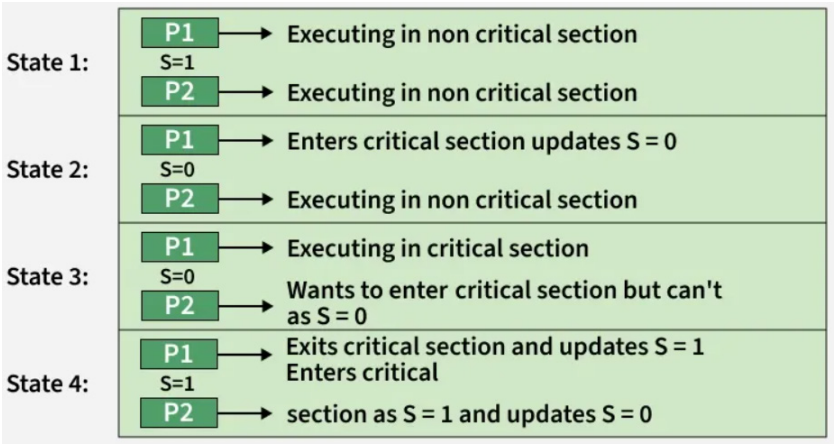
Example Scenario

Consider two processes, P1 and P2, sharing a semaphore S, initially set to 1 (binary semaphore):

- Initial State: Both processes are executing outside the critical section, and S = 1.
- Step 1: Process P1 executes *wait(S)* and enters the critical section. Now S = 0.
- Step 2: Process P2 attempts to enter but finds S = 0, so it is blocked and must wait.
- Step 3: After completing its task, P1 executes *signal(S)*, increasing S to 1.
- Step 4: P2 is awakened, performs *wait(S)*, and enters the critical section, setting S back to 0.

This mechanism ensures mutual exclusion, allowing only one process to access the shared resource at a time while properly coordinating waiting and execution among processes.

SGOU



1.3.4.2 Features and Types of Semaphores

Semaphores provide a flexible mechanism for controlling access to shared resources and coordinating multiple processes in concurrent systems. Their key capabilities include enforcing mutual exclusion, managing execution order among processes, and regulating the use of limited resources such as printers, files, or devices. They are also useful in solving classical synchronization problems like the Readers–Writers problem, where multiple readers can access data simultaneously while writers require exclusive access. Additionally, semaphores help reduce the chances of deadlocks by controlling how resources are allocated among processes.

Types of Semaphores

1. Counting Semaphore

A counting semaphore can take any non-negative integer value, making it suitable for managing multiple instances of a resource. It keeps track of how many units of a resource are available and allows multiple processes to access them as long as the count permits. For example, if there are 5 printers or 3 database connections, a counting semaphore can regulate access so that only the allowed number of processes use them simultaneously.

2. Binary Semaphore

A binary semaphore can have only two values: 0 or 1, and is primarily used to ensure mutual exclusion. It allows only one process to enter the critical section at a time, functioning similarly to a mutex lock.

This type is commonly used when managing access to a single shared resource.

1.3.4.3 Limitations of Semaphores

Despite their usefulness, semaphores have certain drawbacks:

- Priority Inversion: A lower-priority process holding a semaphore can delay a higher-priority process that needs the same resource.
- Deadlock Possibility: If processes wait on each other's semaphores in a circular manner, the system can become stuck indefinitely.

SGOU

- Management Complexity: Proper use requires careful handling of *wait* and *signal* operations; incorrect usage may lead to synchronization errors.
- Busy Waiting: In some implementations, processes repeatedly check the semaphore value, leading to unnecessary CPU usage.

Semaphores are powerful tools for synchronization and resource management, but they require careful design and disciplined use to avoid inefficiencies and system-level issues.

1.3.5 Monitors in Process Synchronization

Monitors provide a higher-level approach to process and thread synchronization, making it easier to manage shared resources in concurrent systems. They are built on top of basic locking mechanisms and are commonly used in multithreaded programming environments such as Java.

Unlike semaphores, where programmers must manually handle synchronization using operations like *wait()* and *signal()*, monitors encapsulate both the shared data and the operations performed on that data within a single structured unit. This design reduces complexity and minimizes the chances of programming errors, making synchronization more reliable.

A monitor can be thought of as similar to a class or module, where shared variables and the methods that access them are grouped together. One of its key features is that it automatically enforces mutual exclusion, allowing only one thread to execute within the monitor at any given time.

In Java, monitors are not represented by a separate keyword. Instead, their functionality is achieved using synchronized methods or blocks, which ensure that only one thread can access the protected section of code at a time. This built-in mechanism simplifies synchronization and makes it more secure compared to lower-level techniques.

1.3.5.1 How Monitors Are Implemented

Monitors are realized at the **programming language level** rather than being directly managed by the operating system. In languages like Java, monitor-like behavior is achieved using the **synchronized** construct, which ensures that only one thread can execute a protected section of code at a time.

A monitor combines the **shared resource (data)** with the **methods that operate on it**, keeping both within a single controlled structure. This design automatically enforces **mutual exclusion**, unlike semaphores where the programmer must explicitly manage synchronization using operations such as *wait()* and *signal()*.

Synchronization can be applied either through **synchronized methods** or **synchronized blocks**, depending on the level of control required. To manage coordination between threads, **condition variables** are used along with built-in methods like *wait()*, *notify()*, and *notifyAll()*, which allow threads to pause execution and resume when appropriate conditions are met.

SGOU

```
class Monitor{                    //Name of Monitor

   variables;
   condition variables;

   process p1 {....}              //synchronized method
   process p2 {....}              //synchronized method

}
```

1.3.5.2 Condition Variables in Monitors

Condition variables are used within monitors to **control the coordination of threads based on specific conditions**. They allow a thread to pause its execution until a required condition becomes true. These variables are always associated with a monitor, ensuring that synchronization is handled in a structured and safe manner.

There are three commonly used operations with condition variables:

- **wait()**: Causes the thread to temporarily release the monitor lock and enter a waiting state until it is notified.
- **signal()**: Wakes up one of the threads waiting on the condition, if any are present.
- **broadcast()** (available in some languages): Notifies all waiting threads so they can resume execution.

These operations help in managing situations where threads must wait for certain conditions, such as the availability of resources, before proceeding.

Example

```
class AccountUpdate {
   private int balance;
   condition sufficientFunds;   // condition variable
   synchronized void deposit(int amount) {
      balance = balance + amount;
      // notify waiting threads that funds are now available
      signal(sufficientFunds);
   }
   synchronized void withdraw(int amount) {
      if (balance < amount) {
```

SGOU

```
            wait(sufficientFunds);  // wait until sufficient balance
        } else {
            balance = balance - amount;
        }
    }
}
```

In this example, when a withdrawal request is made but the balance is insufficient, the thread waits until enough funds are deposited. Once a deposit occurs, a waiting thread is notified and can proceed. This ensures **safe and coordinated access to shared data** without conflicts.

Summarized Overview

Inter Process Communication (IPC) is a mechanism that allows processes to exchange data and coordinate their execution within an operating system. Since processes operate in separate memory spaces, IPC provides structured and controlled methods to ensure safe communication. It plays an important role in synchronization, data sharing, and preventing conflicts when multiple processes access shared resources. The two main approaches to IPC are shared memory and message passing; shared memory offers high-speed communication but requires proper synchronization, while message passing is more secure but involves additional overhead due to kernel involvement. IPC is widely used in real-world systems such as ATM operations, where multiple processes work together to complete a task. However, improper implementation can lead to problems like race conditions, deadlock, starvation, and data inconsistency. Classical problems such as Dining Philosophers, Producer–Consumer, Readers–Writers, and Sleeping Barber illustrate these challenges. To address them, synchronization mechanisms like semaphores, monitors, and locks are used to ensure efficient, safe, and coordinated process communication.

SGOU

Assignments

1. Explain the concept of Inter Process Communication (IPC). Discuss its importance in operating systems and describe the two main approaches used for communication between processes with suitable examples.

2. Describe the shared memory model of IPC in detail. Explain how it works, its advantages, and the need for synchronization mechanisms while using shared memory.

3. Explain the message passing mechanism in IPC. Compare it with shared memory in terms of performance, security, and complexity with relevant examples.

4. Discuss the major problems in Inter Process Communication, such as race conditions, deadlock, starvation, and scalability issues. Explain how these problems affect system performance.

5. Explain the Dining Philosophers Problem in detail. Describe how deadlock and starvation occur in this problem and discuss suitable solutions to overcome these issues.

6. Describe the Producer–Consumer problem with a suitable real-life and technical example. Explain how synchronization mechanisms such as semaphores or mutex locks help in solving this problem.

7. Explain the Readers–Writers problem and its different variations. Discuss how fairness can be achieved while ensuring efficient access to shared resources.

8. Describe the concept of a critical section. Explain its structure, requirements (mutual exclusion, progress, bounded waiting), and illustrate with a real-world example.

SGOU

Reference

1. Arpaci-Dusseau, R. H., & Arpaci-Dusseau, A. C. (2018). *Operating systems: Three easy pieces*. Arpaci-Dusseau Books.

2. Love, R. (2010). *Linux kernel development* (3rd ed.). Addison-Wesley.

3. Bovet, D. P., & Cesati, M. (2005). *Understanding the Linux kernel* (3rd ed.). O'Reilly Media.

Suggested Reading

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts* (10th ed.). Wiley.

2. Stallings, W. (2018). *Operating systems: Internals and design principles* (9th ed.). Pearson.

3. Tanenbaum, A. S., & Bos, H. (2015). *Modern operating systems* (4th ed.). Pearson.

4. Dhamdhere, D. M. (2017). *Operating systems: A concept-based approach* (3rd ed.). McGraw Hill Education.

SGOU

Space for Learner Engagement for Objective Questions

Learners are encouraged to develop objective questions based on the content in the paragraph as a sign of their comprehension of the content. The Learners may reflect on the recap bullets and relate their understanding with the narrative in order to frame objective questions from the given text. The University expects that 1 - 2 questions are developed for each paragraph. The space given below can be used for listing the questions.

SGOU

4 UNIT Deadlocks

Learning Outcomes

After completion of this unit, the learner will be able to:

- understand the concept and causes of deadlock in operating systems
- explain the four necessary conditions of deadlock with examples
- analyze different techniques such as prevention, avoidance, and detection
- apply deadlock handling strategies to improve system performance

SGOU

Background

Deadlocks arise in systems where multiple processes compete for limited resources. In early computing systems, improper resource allocation often led to system standstill situations, reducing efficiency and performance. As operating systems evolved, the need to manage resource sharing effectively became critical. Concepts such as mutual exclusion and process synchronization laid the foundation for understanding deadlocks. Over time, techniques like prevention, avoidance, and detection were developed to ensure smooth system operation and to avoid indefinite blocking of processes.

Keywords

Deadlock, Mutual Exclusion, Hold and Wait, No Preemption, Circular Wait, Banker's Algorithm, Safe State, Wait-for Graph

Discussion

1.4.1 Introduction

Deadlocks are a significant problem in operating systems and database systems that occur when multiple processes compete for limited resources. A deadlock arises when a group of processes becomes permanently blocked because each process is waiting for a resource held by another process in the group, creating a cycle of dependency. For a deadlock to occur, four necessary conditions must hold simultaneously: mutual exclusion, hold and wait, no preemption, and circular wait. These conditions make it impossible for the processes to proceed, leading to a complete system standstill and reduced efficiency.

To handle deadlocks, several techniques are used, including prevention, avoidance, and detection. Deadlock prevention aims to eliminate at least one of the necessary conditions, while deadlock avoidance ensures that the system always remains in a safe state by carefully allocating resources, often using algorithms like Banker's Algorithm. In contrast, deadlock detection allows deadlocks to occur and then identifies them using methods such as wait-for graphs, followed by recovery techniques like process termination or resource preemption. Together, these approaches help in managing deadlocks and maintaining system performance.

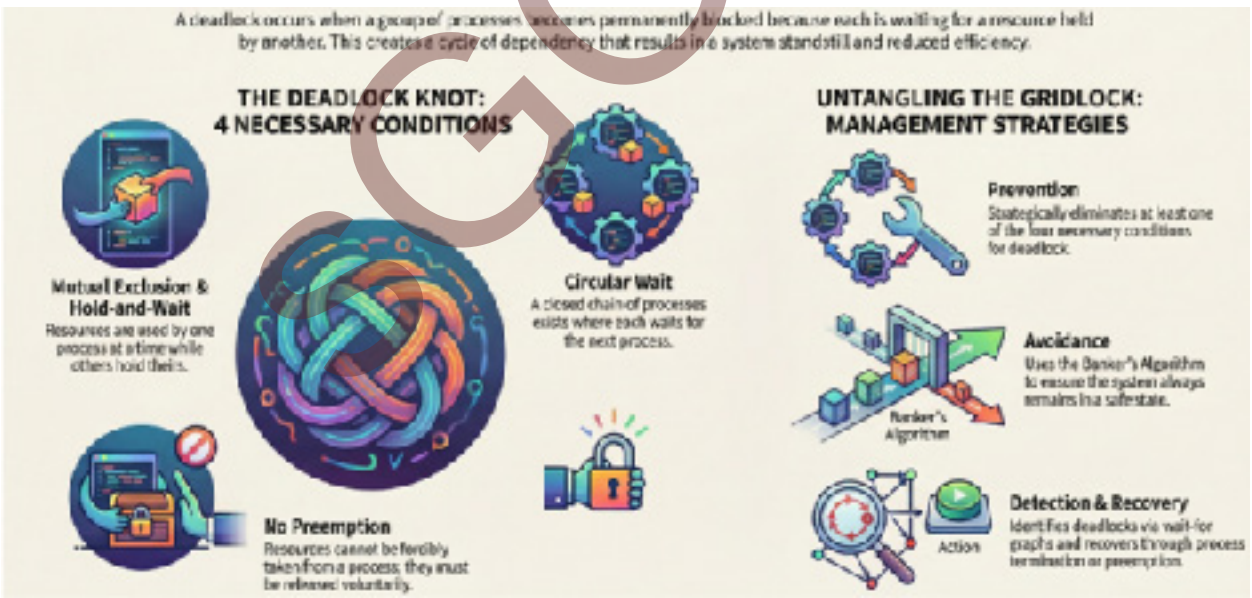


Fig 1.4.1 Deadlock : from standstill to solution

1.4.2 Deadlock

Deadlock is a situation in a computer system where two or more processes are unable to continue their execution because each process is waiting for a resource that is held by another process. In this state, none of the processes can proceed, resulting in a permanent

blocking situation. Deadlocks commonly occur in operating systems, database systems, and distributed systems where multiple processes compete for limited resources such as memory, files, or devices. Because the processes are stuck waiting indefinitely, system performance is affected and resources remain underutilized.

Deadlock happens due to improper allocation and management of resources among processes. When a process holds one resource and requests another resource that is currently being used by another process, a waiting situation arises. If this pattern continues among multiple processes, it creates a circular chain of dependencies. Since resources cannot be forcibly taken away and processes do not release their currently held resources while waiting, the system enters a deadlock state. This situation typically arises when all four necessary conditions: mutual exclusion, hold and wait, no preemption, and circular wait, occur at the same time.

Deadlocks usually occur in environments where resources are limited and shared among processes. For example, in a system with two processes and two resources, if Process 1 holds Resource A and waits for Resource B, while Process 2 holds Resource B and waits for Resource A, both processes will wait forever. This creates a circular waiting condition, and neither process can proceed. Such situations can also be seen in real life, like two vehicles on a narrow road waiting for each other to move, resulting in a complete standstill.

1.4.3 Necessary Conditions

Deadlock is a situation where processes are permanently blocked because each is waiting for resources held by others. For a deadlock to occur, four necessary conditions (Coffman conditions)(Fig 1.4.2) must exist simultaneously. These conditions explain why deadlocks happen and help in designing methods to prevent them.

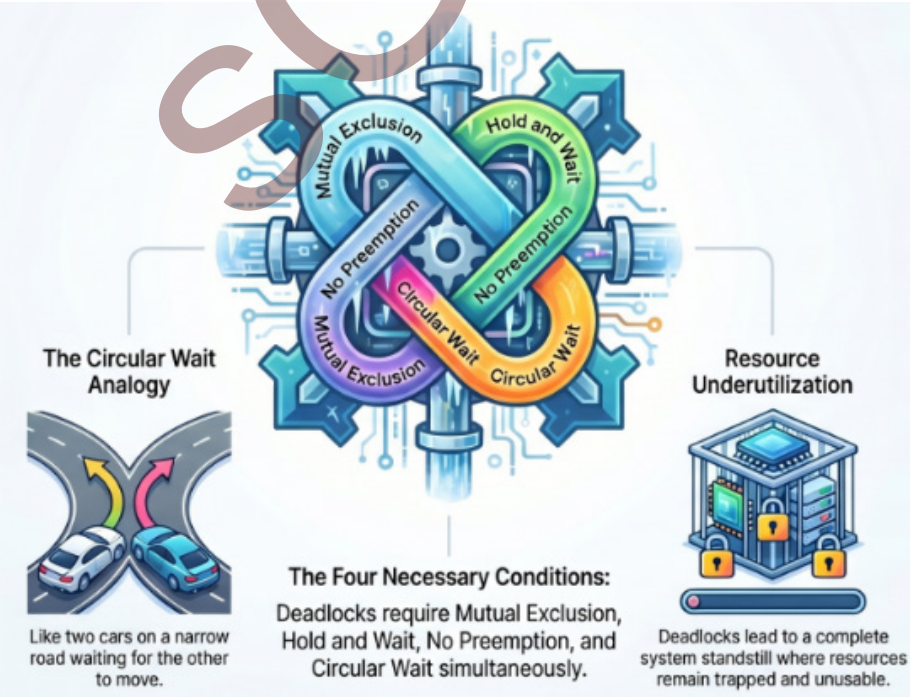


Fig 1.4.2 Four necessary conditions for deadlock.

SGOU

1.4.3.1 Mutual Exclusion

Mutual exclusion is a fundamental condition in operating systems where at least one resource is non-shareable, meaning only one process can access and use that resource at a time. When a process acquires such a resource, no other process is allowed to use it simultaneously. If another process requests the same resource, it is forced to wait until the current process releases it. This ensures that there is no conflict, inconsistency, or corruption of data while the resource is in use.

The key idea behind mutual exclusion is that certain resources, such as printers, files, or critical sections of memory, cannot be safely shared among multiple processes at the same time. Therefore, the system enforces exclusive access, allowing only one process to operate on the resource while others remain in a waiting state. This waiting mechanism is essential to maintain system stability and correctness.

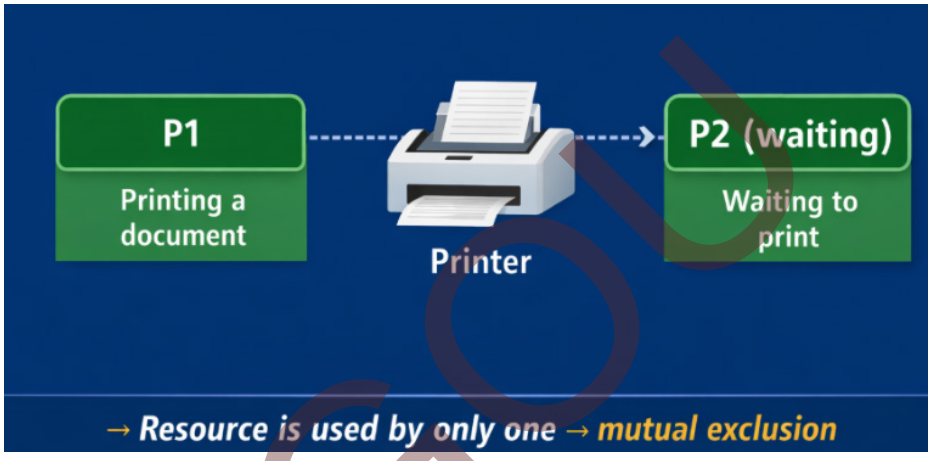


Fig 1.4.3 mutual exclusion

For example, consider a printer in a computer lab. When one process (P1) is printing a document, another process (P2) that wants to print must wait until P1 finishes and releases the printer. This situation can be represented in fig 1.4.3

Here, P1 is actively using the printer, and P2 is in a waiting state, demonstrating that the resource is allocated to only one process at a time.

1.4.3.2 Hold and Wait

Hold and wait is a condition in which a process is already holding one or more resources while simultaneously requesting additional resources that are currently held by other processes. In this situation, the process does not release the resources it already possesses and continues to wait until the required resources become available. This leads to a situation where multiple processes may be holding some resources while waiting for others, increasing the possibility of deadlock.

The key idea behind hold and wait is that processes retain their allocated resources while requesting new ones, instead of releasing them first. Because of this behavior,

other processes may also get stuck in a waiting state, as each process holds something the other needs. This creates a chain of dependencies that can eventually lead to a deadlock situation.

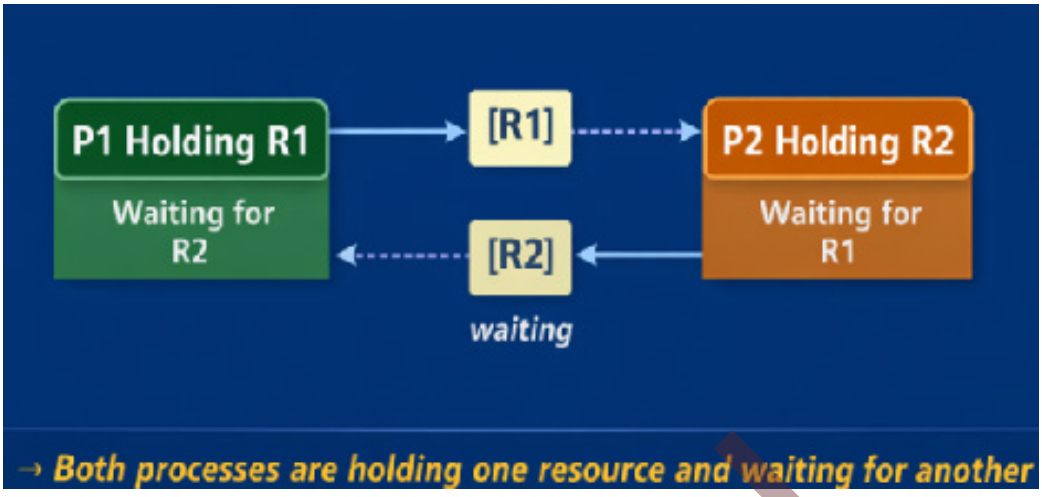


Fig 1.4.4 Hold and Wait

For example, consider two processes P1 and P2. Process P1 holds resource R1 and is waiting for resource R2, while process P2 holds R2 and is waiting for R1. This can be represented as: P1 → [R1] and waiting for R2, and P2 → [R2] and waiting for R1.

Both processes are holding one resource and waiting for another, so neither can proceed.

A simple real-life example is when one person holds a pen and waits for a notebook, while another person holds the notebook and waits for the pen. Since neither releases what they have, both remain stuck in a waiting state.

1.4.3.3 No Preemption

No preemption is a condition where resources allocated to a process cannot be forcibly taken away by the operating system. Once a process acquires a resource, it continues to hold it until it voluntarily releases it after completing its task. This means that if another process needs the same resource, it must wait until the current process finishes using it.

The key idea of no preemption is that there is no mechanism for the system to interrupt a process and reassign its resources to another process. This restriction can lead to situations where processes are forced to wait for long periods, especially if the resource-holding process is delayed or stuck.

For example, if process P1 is holding resource R1, the operating system cannot take R1 away from P1 to give it to another process P2. As a result, P2 must remain in a waiting state until P1 completes its work and releases the resource.

A real-life example is a person using a computer in a lab. You cannot forcibly take the computer away while they are working; you must wait until they finish and give it up voluntarily. This illustrates how the absence of preemption can contribute to deadlock conditions.



Fig 1.4.5 No Preemption

1.4.3.4 Circular Wait

Circular wait is a condition in which a set of processes are arranged in a circular chain, where each process is waiting for a resource that is held by the next process in the cycle. Because of this circular dependency, none of the processes can proceed, resulting in a deadlock.

The key idea behind circular wait is the formation of a closed loop of processes, where each process depends on another in the cycle. Since every process is waiting for a resource held by another process in the same loop, no process can release its resource, and the system becomes stuck.

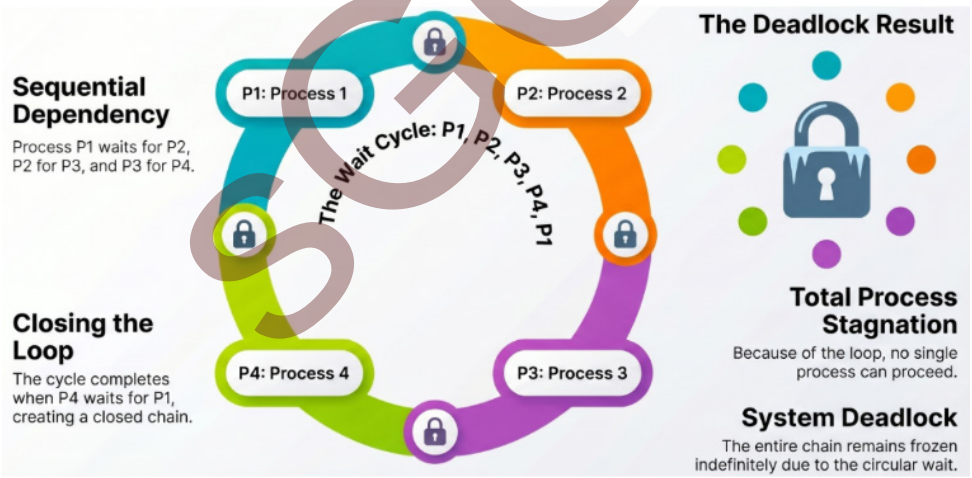


Fig 1.4.6 Circular Wait

For example, consider four processes P1, P2, P3, and P4 forming a cycle: P1 waits for P2, P2 waits for P3, P3 waits for P4, and P4 waits for P1. This creates a circular chain:

P1 → P2 → P3 → P4 → P1.

Because of this loop, none of the processes can proceed, leading to deadlock.

A common real-life example is a traffic intersection where four cars are stuck in a

circular wait. Car A waits for Car B, Car B waits for Car C, Car C waits for Car D, and Car D waits for Car A. Since no car can move forward, the entire system becomes blocked, illustrating a deadlock caused by circular wait.

1.4.4 Deadlock Prevention

Deadlock prevention is a strategy used in operating systems to ensure that a deadlock never occurs by eliminating at least one of the four necessary conditions: mutual exclusion, hold and wait, no preemption, and circular wait. Instead of allowing a deadlock to happen and then handling it later, prevention techniques proactively design the system in such a way that at least one of these conditions is always false.

1.4.4.1 Preventing Mutual Exclusion

Mutual exclusion cannot be completely eliminated because some resources are inherently non-shareable (like printers). However, in some cases, it can be reduced by making resources shareable wherever possible.

Method: Use shared resources (e.g., read-only files can be accessed by multiple processes simultaneously).

Example: Multiple processes can read the same file at the same time without conflict, reducing the need for exclusive access.

Limitation: Not all resources can be shared, so this method has limited applicability.

1.4.4.2 Preventing Hold and Wait

This condition can be prevented by ensuring that processes do not hold resources while waiting for others.

Methods:

1. **Request all resources at once:** A process must request all required resources before execution begins.
2. **Release before requesting new:** A process must release all held resources before requesting additional ones.

Example: A process needing a printer and scanner must request both at the same time instead of requesting them separately.

Advantages:

- Eliminates circular dependencies.

Disadvantages:

- Leads to low resource utilization.
- May cause starvation (processes wait indefinitely).

SGOU

1.4.4.3 Preventing No Preemption

This condition can be avoided by allowing the system to take resources away from processes when needed.

Method:

- If a process holding resources requests another unavailable resource:
  - It must release its currently held resources.
  - The released resources are allocated to other processes.

Example: If P1 holds a printer and requests a scanner (which is unavailable), the system may take the printer from P1 and allocate it to another process.

Advantages:

- Improves resource utilization.

Disadvantages:

- Not suitable for all resources (e.g., printers cannot be stopped midway).
- May lead to inconsistent states.

1.4.4.4 Preventing Circular Wait

This is one of the most effective methods for deadlock prevention.

Method:

- Assign a priority/order number to each resource.
- Processes must request resources in increasing order only.

Example:

If resources are ordered as R1 < R2 < R3:

- A process can request R1 → R2 → R3
- But cannot request R2 and then R1

Advantages:

- Completely eliminates circular wait.

Disadvantages:

- Requires strict ordering.
- May reduce flexibility in resource allocation.

SGOU

1.4.5 Deadlock Avoidance and Detection

1.4.5.1 Deadlock Avoidance

Deadlock avoidance is a dynamic approach. Every time a process requests a resource, the OS performs a "test-run" to see if granting it will leave the system in a Safe State.

1. The Banker's Algorithm

The Banker's Algorithm is the primary mechanism for avoidance in systems with multiple resource instances. It treats resources like a bank treats cash: it never lends so much that it cannot satisfy the needs of at least one remaining customer.

Key Mathematical Structures:

- **Max[n x m]**: The maximum demand of each process.
- **Allocation[n x m]**: What each process currently holds.
- **Need [n x m]:** The remaining requirement (Need = Max - Allocation).
- **Available [m]:** The current free resources in the system.

2. The Safety Algorithm (The "Check")

To determine if a state is safe, the OS looks for a Safe Sequence.

1. Initialize Work = Available and Finish[all] = False.

2. Find a process P_i where Finish[i] == False and $Need_i \leq Work$.

3. If found:

- Simulate P_i finishing: Work = Work + Allocation_i.
- Set Finish[i] = True.
- Repeat from Step 2.

4. If all Finish[i] == True, the system is in a Safe State

1.4.5.2 Deadlock Detection

In detection, the OS is more "optimistic." It grants requests immediately but periodically runs a diagnostic to find deadlocks that have already formed.

1. Wait-for Graph (WFG)

For systems where each resource has only one instance, we use a Wait-for Graph. This is a simplified version of a Resource Allocation Graph (RAG).

- Nodes represent Processes.
- A directed edge $P_i \rightarrow P_j$ exists if P_i is waiting for a resource held by P_j.

SGOU

- Rule: A cycle in the Wait-for Graph equals a deadlock.

2. Detection for Multiple Instances

When multiple instances exist (e.g., 3 identical CPUs), a cycle does not necessarily mean a deadlock. The OS uses a detection algorithm similar to the Banker's Safety Algorithm, but instead of using the Need matrix, it uses the Current Request matrix.

1.4.5.3 Recovery from Deadlock

Once the detection algorithm identifies a deadlock, the OS must intervene.

1. Process Termination

- **Abort All:** Simple but results in massive data loss.
- **Partial Abort:** Terminate processes one by one based on:
  - **Priority:** Kill lower-priority processes first.
  - **Progress:** Kill processes that have just started rather than those near completion.
  - **Resource Consumption:** Kill processes holding the most resources to "free up" the system quickly.

2. Resource Preemption

- **Victim Selection:** Determine which process can "lose" its resource with minimum cost.
- **Rollback:** Return the process to a previous safe checkpoint so it can restart later.
- **Starvation Prevention:** Ensure the same process isn't preempted repeatedly (usually by including a "rollback count" in the priority logic).

Table 1.4.1 Comparison of avoidance and detection

Feature	Avoidance (Proactive)	Detection (Reactive)
Philosophy	"Is it safe to give this?"	"Are we stuck yet?"
Data Required	Maximum future needs.	Current requests only.
System State	Never allowed to be unsafe.	Can be deadlocked until the next check.
Overhead	High (Calculated at every request).	Variable (Periodic check).
Typical Usage	High-reliability systems.	General-purpose OS (Windows/ Linux).

SGOU

Summarized Overview

Deadlock is a condition where processes are permanently blocked due to circular dependency on resources. It occurs when four conditions such as mutual exclusion, hold and wait, no preemption, and circular wait exist simultaneously. To handle deadlocks, systems use prevention (eliminating conditions), avoidance (ensuring safe state using Banker's Algorithm), and detection (identifying deadlocks using wait-for graphs). Recovery methods such as process termination and resource preemption help restore system functionality. Proper deadlock management improves system efficiency and reliability.

Assignments

1. Define deadlock and explain its causes with examples.
2. Explain the four necessary conditions for deadlock in detail.
3. Describe deadlock prevention techniques with advantages and disadvantages.
4. Explain Banker's Algorithm with suitable examples.
5. Differentiate between deadlock avoidance and detection.
6. Explain wait-for graph with diagram.
7. Describe methods for recovery from deadlock.

Reference

1. Godse, A. P., & Godse, D. A. (2017). *Operating systems* (2nd ed.). Technical Publications.

2. Flynn, I. M., & McHoes, A. M. (2011). *Understanding operating systems* (6th ed.). Cengage Learning.

SGOU

3. Deitel, H. M., Deitel, P. J., & Choffnes, D. R. (2003). *Operating systems* (3rd ed.). Prentice Hall.

4. Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of database systems* (7th ed.). Pearson.

Suggested Reading

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts* (10th ed.). Wiley.

2. Tanenbaum, A. S., & Bos, H. (2015). *Modern operating systems* (4th ed.). Pearson.

3. Stallings, W. (2018). *Operating systems: Internals and design principles* (9th ed.). Pearson.

4. Dhamdhere, D. M. (2017). *Operating systems: A concept-based approach* (3rd ed.). McGraw-Hill.

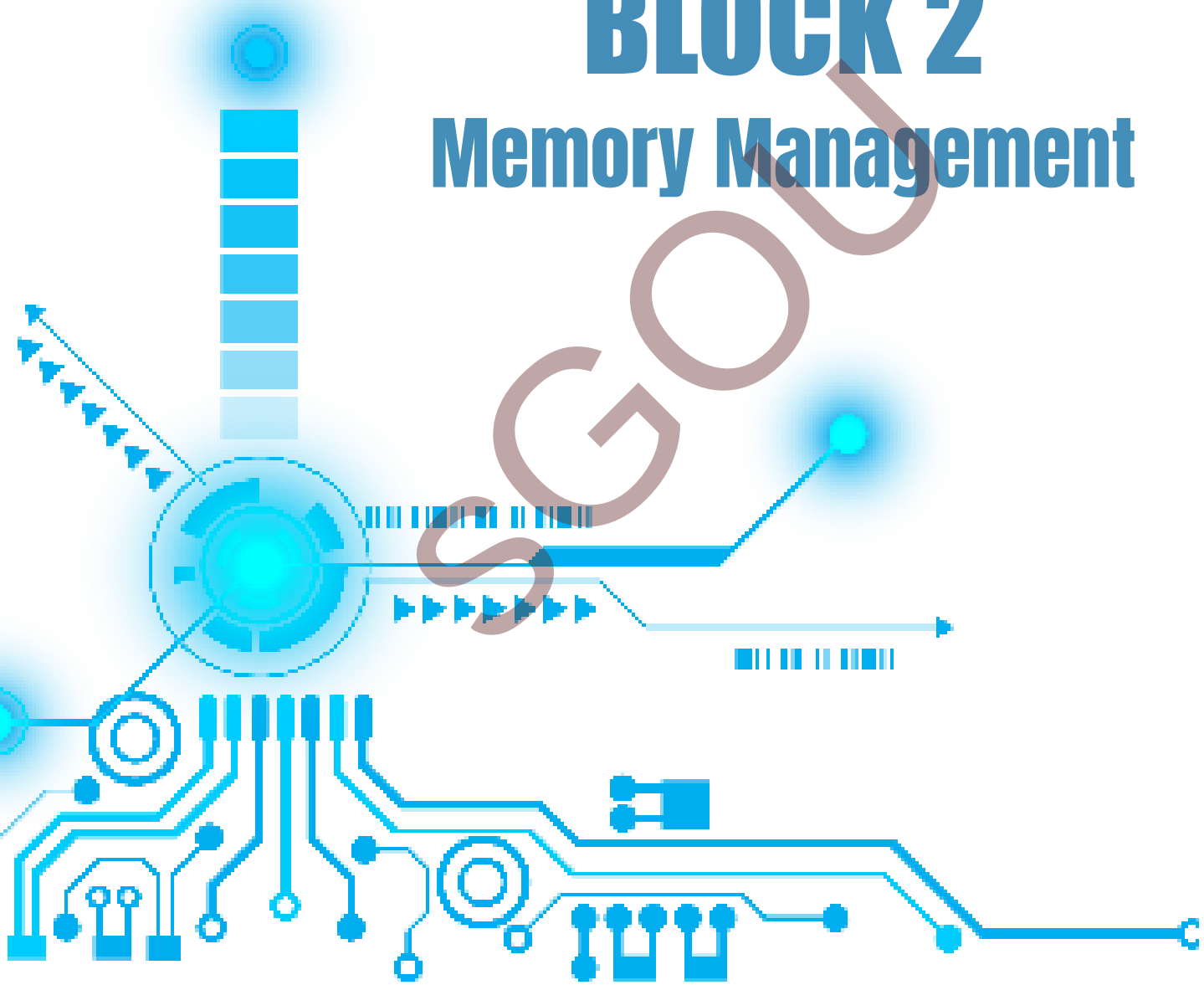
SGOU

Space for Learner Engagement for Objective Questions

Learners are encouraged to develop objective questions based on the content in the paragraph as a sign of their comprehension of the content. The Learners may reflect on the recap bullets and relate their understanding with the narrative in order to frame objective questions from the given text. The University expects that 1 - 2 questions are developed for each paragraph. The space given below can be used for listing the questions.

SGOU

BLOCK 2
Memory Management



SGOU

1 UNIT

Memory Management Strategies

Learning Outcomes

After completion of this unit, the learner will be able to:

- explain the role of memory management in operating systems
- describe the hardware components used for memory management
- differentiate between logical and physical addresses
- explain contiguous memory allocation, swapping, and partitioning techniques

Background

The growth of computer technology has greatly increased the demand for efficient memory utilization. Early computer systems could execute only one program at a time, so memory management was relatively simple. As technology advanced, computers began supporting multiprogramming and multitasking environments where several processes execute simultaneously. This created the need for effective memory management techniques to allocate memory properly, protect processes from one another, and improve overall system performance.

Modern operating systems such as Windows, Linux, and macOS continuously manage memory while multiple applications run together. Efficient memory management became essential to avoid problems such as memory conflicts, fragmentation, slow execution, and system crashes. To solve these issues, operating systems introduced concepts such as address binding, swapping, partitioning, logical and physical address mapping, and advanced hardware support including MMU, TLB, and cache memory.

SGOU

Over time, simple contiguous allocation methods were replaced by more flexible non-contiguous memory management techniques such as paging and segmentation. These methods improved memory utilization, supported virtual memory, and enabled modern systems to execute large programs efficiently. Today, memory management forms one of the core foundations of operating systems and plays a vital role in achieving fast, secure, and reliable computing environments.

Keywords

Base register, limit register, MMU, TLB, logical address, physical address, sap-in, swap-out

Discussion

2.1.1 Introduction

Modern computer systems run multiple applications at the same time, such as web browsers, music players, video editors, and games. Every application requires memory for storing instructions and data during execution. In a multitasking environment, several programs remain active simultaneously, so the operating system must manage memory carefully to ensure that programs do not interfere with one another. What happens if two programs attempt to use the same memory location? Why is efficient memory management necessary for system performance? These questions lead to the study of memory management in operating systems.

Memory management is one of the major responsibilities of an operating system. It is the process of allocating memory to programs, tracking used and free memory spaces, protecting processes from one another, and improving overall system performance. Whenever a program begins execution, the operating system allocates memory for storing instructions and data. After the program completes execution, the allocated memory is released and made available for other programs. Proper memory management ensures smooth execution of processes and efficient utilization of available memory resources.

A good memory management system should achieve several important objectives. It should provide protection so that one process cannot access another process's memory illegally. It should support controlled sharing of memory whenever necessary. It should improve memory utilization by minimizing wastage and making maximum use of available space. Memory management should also support multitasking by allowing several processes to reside in memory at the same time. Another important objective is relocation, which allows programs to move from one memory location to another

SGOU

during execution without affecting their functioning.

A real-life example of memory management can be compared to room allocation in a hostel. Each student is assigned a separate room, and no two students should occupy the same bed. Empty rooms are assigned to new students, and rooms become available again when students leave. In a similar manner, the operating system allocates memory spaces to different programs, prevents conflicts between them, and reuses free memory efficiently. Without proper memory management, programs may overwrite each other's data, the system may become unstable, or applications may stop responding.

Efficient memory management is essential for the smooth functioning of modern computer systems and for supporting multitasking environments effectively.

2.1.2 Basic hardware for Memory Management

Modern computer systems execute many programs at the same time, and each program requires memory space for storing instructions and data. But how does the operating system ensure that one program does not interfere with another? Can every program freely access all areas of memory? If that happens, important data may be overwritten, causing errors or system crashes. To avoid such problems, the operating system uses hardware support for memory management.

The CPU continuously accesses main memory during program execution. Whenever a program runs, the CPU generates addresses to fetch instructions and data from memory. At this stage, the system must ensure that the process accesses only its own memory area. This protection cannot be handled efficiently by software alone because memory operations happen very quickly. Therefore, special hardware mechanisms are used to control and monitor memory access.

1. Base Register

One important hardware component used in memory management is the base register. The base register stores the starting address of the memory region allocated to a process. Every process is assigned a separate starting address in memory, and the base register keeps track of that location.

2. Limit Register

Another important hardware component is the limit register. The limit register stores the size of the memory area assigned to a process. Now think about this: what should happen if a process tries to access memory beyond its assigned range? The system should immediately stop such access to protect other processes.

Working of Base and Limit Registers

Whenever a process generates an address, the system checks whether the address lies within the limit specified for that process. If the address is valid, the value stored in the base register is added to generate the actual physical memory address. If the address exceeds the limit, access is denied and an error occurs. In this way, hardware mechanisms help provide memory protection and safe execution of processes.

SGOU

Example

Suppose the base register contains the value 3000 and the limit register contains 1000. This means the process can access memory locations only within that range. If the process generates address 200, the system converts it into the physical address 3200. But if the process tries to access address 1500, the access is rejected because it exceeds the permitted range.

3. Memory Management Unit

The Memory Management Unit (MMU) is an important hardware component used in memory management. It helps in converting the logical addresses generated by the CPU into actual physical addresses in the main memory (RAM). Whenever a program runs, the CPU does not directly know the real physical location of data in memory. Instead, it generates logical addresses, and the MMU translates them into physical addresses before accessing memory.

Think about writing a friend's name in your mobile contacts instead of remembering their actual phone number. When you select the name, the phone automatically connects it to the correct number. Similarly, the CPU generates logical addresses, and the MMU automatically finds the correct physical addresses in memory.

The MMU is very useful because it allows programs to use memory as if it is continuous and unlimited, even though the actual physical memory may be divided into different locations. This makes program execution easier and more efficient. It also helps in memory protection and supports multitasking by ensuring that one program does not access another program's memory space.

4. Translation Look aside Buffer(TLB)

The Translation Lookaside Buffer (TLB) is a small and very fast memory unit present inside the CPU. It is used to improve the speed of address translation in paging. The TLB stores recently used page table entries so that the system can quickly find physical addresses without searching the page table repeatedly. You will learn in detail about paging in the 4th unit of this block.

Whenever the CPU generates a logical address, the system first checks the TLB to see whether the required page translation is already available. If the mapping is found in the TLB, the physical address can be obtained immediately. This is called a TLB hit. If the mapping is not found, the system searches the page table stored in memory. This is called a TLB miss.

Think about searching for a frequently used contact number in a mobile phone. Instead of searching the full contact list every time, recently called numbers appear in the recent calls section, making access much faster. Similarly, the TLB stores recently used address mappings so that the CPU can access memory quickly.

Without the TLB, the CPU would need to perform extra memory lookups every time it accesses data, which would slow down system performance. By storing recently used translations, the TLB reduces memory access time and improves the overall speed and efficiency of the computer system.

SGOU

5. Cache Memory : L1, L2, L3

The CPU performs millions of operations every second and constantly requires data and instructions for execution. However, the speed of the main memory (RAM) is lower than the speed of the CPU. If the processor waits for data from RAM every time, overall system performance decreases significantly. To overcome this speed difference, computers use cache memory.

Cache memory is a small but extremely fast memory located very close to the CPU. It temporarily stores frequently accessed data and instructions so that the processor can retrieve them quickly. Since the required information is readily available nearby, the CPU can continue execution without unnecessary delay.

Modern computer systems generally use three levels of cache memory: L1, L2, and L3 cache. The L1 cache is the fastest and is built directly inside the processor core. The L2 cache is larger than L1 and provides additional storage for frequently used data. The L3 cache is larger still and is often shared among multiple CPU cores. These cache levels work together to improve the efficiency of data access.

Whenever the CPU needs data, it first checks the cache memory. If the required data is available there, access becomes very fast.

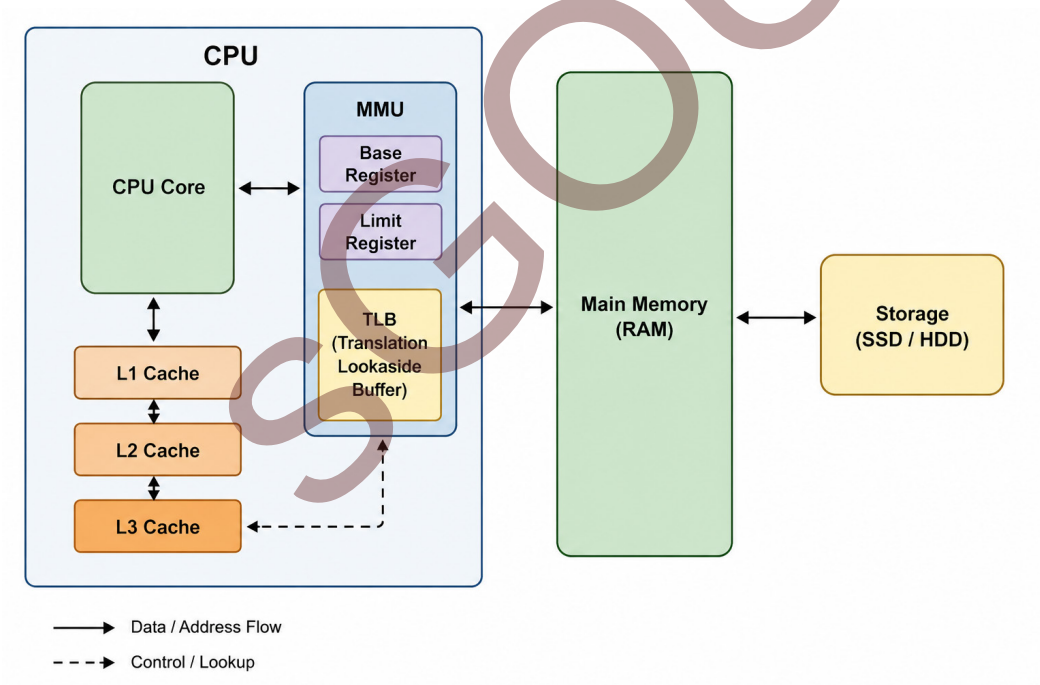


Fig. 2.2.1 . Hardware Components supporting Memory Management

2.1.3 Types of Memory Management Systems

Operating systems use different techniques to organize and allocate memory for programs. Earlier computer systems mainly used contiguous memory allocation because it was simple to implement. However, modern operating systems such as Windows, Linux,

and macOS mostly use non-contiguous memory management techniques because they provide better flexibility, protection, and memory utilization.

Contiguous Memory Management

In contiguous memory management, an entire program is loaded into one continuous block of memory. All instructions and data of the process remain together in adjacent memory locations. This method is simple because the operating system only needs to allocate one large block of free memory for the process.

For example, if a program requires 300 KB of memory, the operating system searches for a single free block of 300 KB. Since the process occupies one continuous region, memory access becomes easy and fast. However, as multiple processes enter and leave memory, free spaces become scattered, making it difficult to find large continuous memory blocks. This problem leads to memory wastage and fragmentation.

Due to these limitations, contiguous memory management is rarely used in modern general-purpose operating systems.

Non-Contiguous Memory Management

Modern operating systems mainly use non-contiguous memory management. In this method, a program is divided into smaller parts, and these parts are stored at different locations in memory instead of one continuous block. Even though the pieces are physically separated, the operating system manages them in such a way that the program appears continuous to the user.

This approach improves memory utilization because small free memory spaces can also be used effectively. It also supports multitasking, virtual memory, and efficient process management. Techniques such as paging and segmentation are examples of non-contiguous memory management.

2.1.4 Address Binding

Whenever a program executes, its instructions and data must be placed in memory. But at what stage should the memory addresses be assigned? This process of associating program instructions and data with memory addresses is called binding. The binding process can occur at different stages of program execution depending on the memory management technique used by the operating system.

Compile-Time Binding

In compile-time binding, the memory location of the program is known before execution begins. Therefore, the compiler generates absolute addresses directly in the compiled program. Since the addresses are fixed, the program must always be loaded into the same memory location.

For example, suppose a program is compiled to run starting from memory address 5000. All addresses generated by the compiler will be based on this fixed location. But what happens if the operating system later decides to load the program at a different location? In that case, the entire program must be recompiled to generate new addresses.

SGOU

Compile-time binding is simple, but it lacks flexibility because the program cannot easily move to another memory location after compilation.

Load-Time Binding

Sometimes, the exact memory location of a program is not known during compilation. In such situations, the compiler generates relocatable addresses instead of fixed absolute addresses. The final address binding is then performed when the program is loaded into memory.

Suppose a program is compiled without knowing where it will be placed in memory. During loading, the loader adjusts the addresses according to the starting memory location assigned to the program. If the starting location changes later, the program only needs to be reloaded instead of recompiled.

Load-time binding provides more flexibility compared to compile-time binding because programs can be loaded into different memory locations whenever required.

Execution-Time Binding

In execution-time binding, address binding is delayed until the program is actually running. This method allows a process to move from one memory location to another during execution. Modern operating systems mainly use this technique because it supports advanced memory management methods such as virtual memory and paging.

During execution, the CPU generates logical addresses, and special hardware such as the Memory Management Unit (MMU) converts them into physical addresses dynamically. Since the mapping occurs at run time, programs can be relocated while executing without affecting their operation.

Execution-time binding provides maximum flexibility and efficient memory utilization. It is widely used in modern operating systems because it supports multitasking, dynamic memory allocation, and virtual memory management effectively.

2.1.5 Logical and Physical Addresses

Whenever a program executes, the CPU continuously generates addresses to access instructions and data stored in memory. But are these addresses the actual locations present in the main memory? Not always. In memory management, two types of addresses are commonly used: logical addresses and physical addresses.

A logical address is the address generated by the CPU during program execution. It is also called a virtual address because it represents the address seen by the program. On the other hand, a physical address is the actual address present in the main memory (RAM). This is the real location where data and instructions are stored.

In compile-time binding and load-time binding, the logical and physical addresses are usually identical because the addresses are fixed before execution begins. However, in execution-time binding, the logical address generated by the CPU differs from the physical address stored in memory. In such cases, the operating system and hardware

SGOU

work together to convert logical addresses into physical addresses dynamically during program execution.

The collection of all logical addresses generated by a program is called the Logical Address Space. Similarly, the collection of all physical addresses corresponding to those logical addresses is known as the Physical Address Space. Modern operating systems mainly work with logical addresses because they provide flexibility, protection, and support for virtual memory.

Now think about this question: if the CPU generates logical addresses, who converts them into physical addresses? This task is performed by a hardware component called the Memory Management Unit (MMU). The MMU maps logical addresses to physical addresses during execution. This mapping process happens very quickly, so the user is not aware of it.

In address mapping, the base register is also called the relocation register. The value stored in this register is added to the logical address generated by the process to obtain the physical address. For example, suppose the relocation register contains the value 1000. If the program generates logical address 0, the MMU converts it into physical address 1000. Similarly, if the process generates logical address 346, it is mapped to physical address 1346.

This dynamic address translation allows programs to execute safely without knowing the actual physical memory locations. It also enables processes to move from one memory location to another during execution, which is an important feature of modern operating systems.

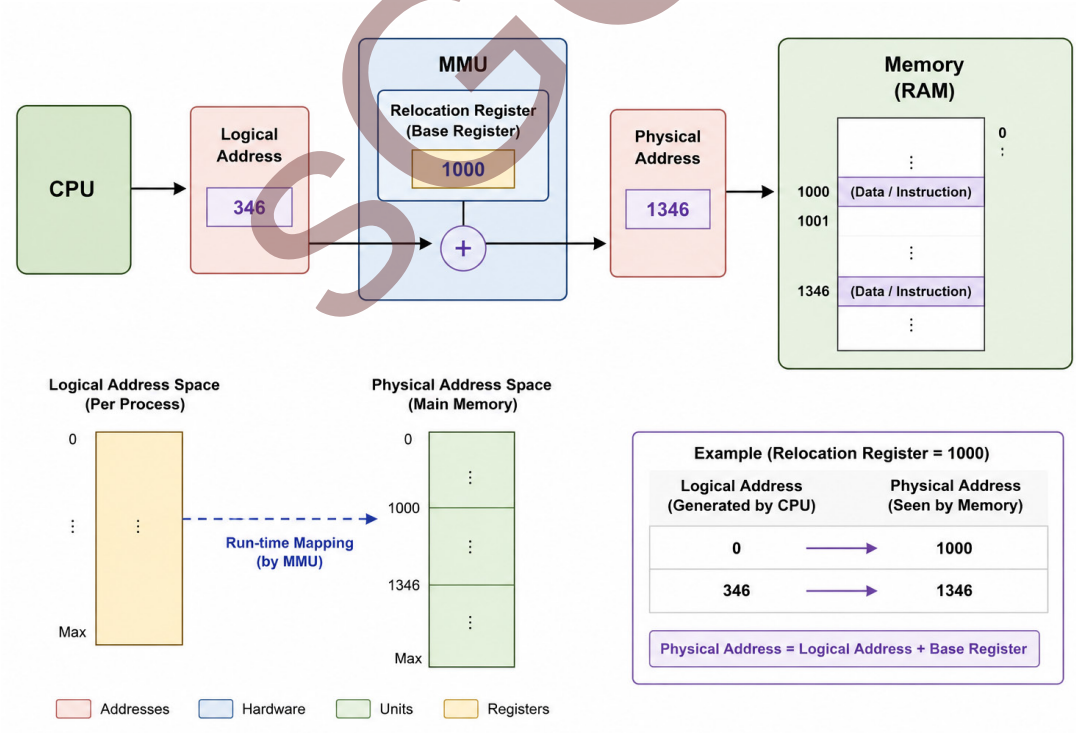


Fig. 2.1.2 Logical to Physical Address mapping

SGOU

2.1.6 Swapping

The concepts of logical addresses, physical addresses, and address mapping help processes execute correctly in memory. However, another important issue must also be handled by the operating system. Main memory has limited space, but many processes may need execution at the same time. So, what happens when the memory becomes full and a new process arrives? To manage such situations efficiently, the operating system uses a technique called swapping.

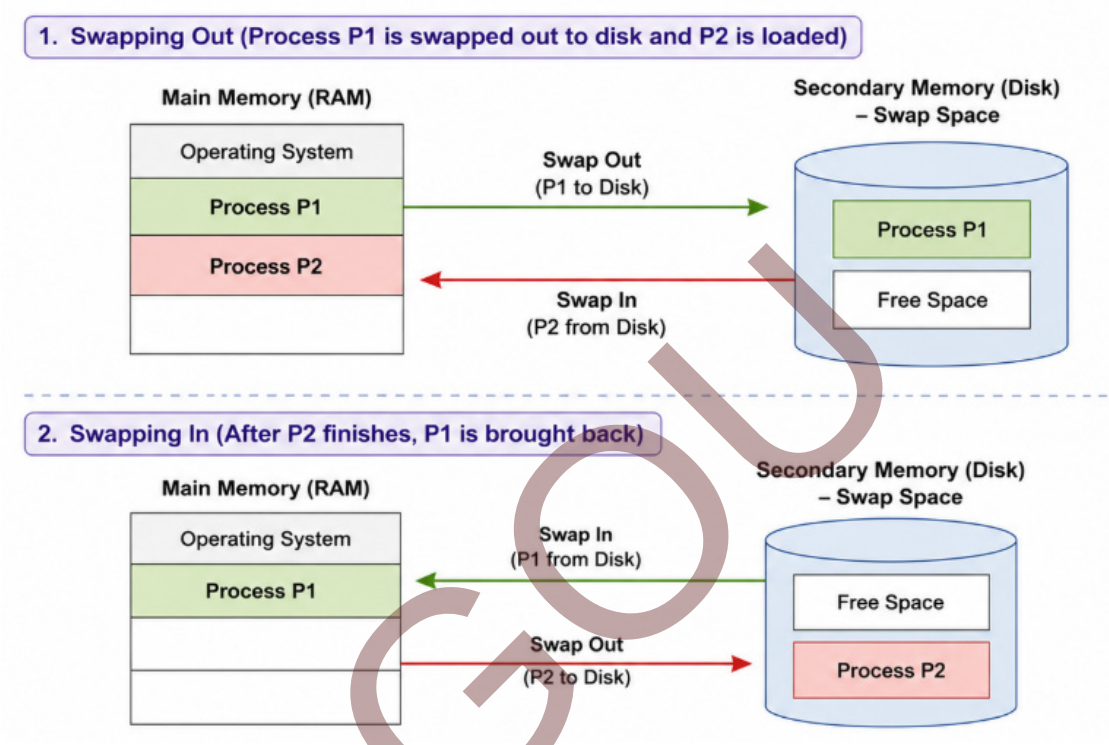


Fig.2.1.3 Swapping Technique in Memory management

Swapping is a memory management technique in which a process is temporarily transferred from the main memory (RAM) to secondary storage, usually a disk. The process moved out of memory is called a swapped-out process. When memory space becomes available again, the process is brought back into RAM and continues execution from the same point.

The area in the secondary storage used to store these temporarily removed processes is known as the swap space. This area acts like temporary storage for processes waiting to return to main memory.

Suppose several applications are running on a computer, and suddenly a large application needs memory for execution. Instead of stopping all running processes, the operating system may temporarily move one of the inactive or less important processes to disk. This creates free space in RAM for the new process. Later, the swapped-out process can be loaded back and resumed.

Swapping helps improve memory utilization because more processes can be managed

SGOU

even when physical memory is limited. It also supports multiprogramming by allowing several processes to share memory effectively.

In some systems, swapping is also used in priority-based scheduling. If a higher-priority process enters the system while memory is occupied by a lower-priority process, the operating system may temporarily swap out the lower-priority process and load the higher-priority process into memory. After the higher-priority process finishes execution, the lower-priority process is brought back and continues its execution.

This special form of swapping is called roll out, roll in. The process removed from memory is rolled out, and the process brought back into memory is rolled in. This technique helps important processes receive immediate attention without permanently terminating other processes.

Although swapping improves memory usage, excessive swapping may reduce system performance because transferring processes between RAM and disk takes time. Therefore, operating systems carefully control swapping activities to maintain efficient execution and better system performance.

2.1.7 Fixed Size and Variable size partitioning

In contiguous memory allocation, each process is placed in one continuous block of memory. But an important question arises here. How should the operating system divide memory for allocating processes? Should all memory blocks be of the same size, or should memory be allocated according to the exact requirement of each process? To solve this, operating systems use two approaches: fixed-size partitioning and variable-size partitioning.

2.1.7.1 Fixed-Size Partitioning

One method of contiguous allocation is fixed-size partitioning. In this method, the memory is divided into several fixed partitions. Each partition can contain only one process at a time. The partitions may be equal in size or different in size depending on the system design.

If a process is smaller than the partition allocated to it, some memory space inside the partition remains unused. This unused memory space is called internal fragmentation.

For example, suppose a partition size is 100 KB, but the process requires only 70 KB. The remaining 30 KB cannot be used by another process and is therefore wasted.

Fixed partitioning is simple to implement, but it limits the number of processes that can stay in memory because the number of partitions is fixed.

2.1.7.2 Variable-Size Partitioning

To reduce memory wastage, another method called variable-size partitioning is used. In this approach, memory is not divided into fixed partitions beforehand. Instead, the operating system allocates memory according to the exact size required by the process.

For example, if a process requires 120 KB, the operating system allocates exactly 120

SGOU

KB. The remaining free memory can later be reused for other processes. This method improves memory utilization compared to fixed partitioning.

However, after several processes are loaded and removed, free memory spaces may become scattered at different locations. This creates a problem called external fragmentation.

2.1.7.3 Fragmentation in Contiguous Allocation

The major disadvantage of contiguous memory allocation is fragmentation.

Fragmentation refers to memory wastage that occurs during memory allocation.

There are mainly two types of fragmentation:

Internal Fragmentation

Internal fragmentation occurs when allocated memory contains unused space inside it. This commonly happens in fixed-size partitioning.

External Fragmentation

External fragmentation occurs when free memory becomes scattered into small blocks at different locations. Even if enough total memory is available, a process may not get memory because a sufficiently large continuous block is unavailable.

Thus, although contiguous memory allocation is simple and easy to implement, fragmentation problems reduce memory utilization and efficiency in modern operating systems.

Although contiguous memory allocation methods are simple and easy to understand, they are not fully efficient for modern computing environments where many processes execute simultaneously. Problems such as internal fragmentation and external fragmentation lead to memory wastage and reduce system performance. This created the need for more advanced memory management techniques that could utilize memory more effectively.

To solve these limitations, operating systems introduced non-contiguous memory allocation methods such as paging and segmentation. Instead of storing an entire process in one continuous block, these methods divide programs into smaller parts and place them in different memory locations. Even though the parts are physically separated, the operating system manages them in such a way that the program appears continuous during execution.

These techniques greatly improve memory utilization, support multitasking efficiently, and form the basis of modern virtual memory systems. Because of their importance in operating systems, paging and segmentation will be explored in detail in Unit 4.

SGOU

Summarized Overview

Memory management is an essential function of an operating system that controls how memory is allocated, used, protected, and released for different processes. It ensures efficient utilization of memory resources and supports multitasking by allowing multiple programs to execute safely at the same time.

The topic begins with the basic hardware support required for memory management, including the base register, limit register, Memory Management Unit (MMU), Translation Lookaside Buffer (TLB), and cache memory. These hardware components help in address translation, memory protection, and improving system performance.

Different types of memory management systems are discussed, namely contiguous and non-contiguous memory allocation. The chapter explains how memory addresses are associated with programs through compile-time, load-time, and execution-time binding techniques. Concepts such as logical addresses, physical addresses, and address mapping are also introduced.

The topic further explains swapping, where processes are temporarily moved between main memory and secondary storage to improve memory utilization. Fixed-size and variable-size partitioning methods are discussed along with memory allocation strategies such as first-fit, best-fit, and worst-fit.

Finally, the problems of internal and external fragmentation are explained, highlighting the limitations of contiguous memory allocation and introducing the need for advanced techniques like paging and segmentation, which form the basis of modern memory management systems.

SGOU

Assignments

1. Explain the role of memory management in an operating system and discuss the importance of hardware support such as MMU, TLB, and cache memory.

2. Describe the different types of address binding with suitable examples. Explain how logical addresses are converted into physical addresses.

3. Explain fixed-size and variable-size partitioning in contiguous memory allocation. Discuss the causes of internal and external fragmentation.

4. A system has a base register value of 5000 and a limit register value of 1200.

Determine whether the following logical addresses are valid or invalid:

a) 250

b) 1300

Also calculate the physical address for the valid logical address.

5. Compare first-fit, best-fit, and worst-fit memory allocation strategies for the following free memory holes: 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB. Determine which hole will be allocated for a process requesting 212 KB using each strategy.

Reference

1. Operating System Concepts Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts* (10th ed.). Wiley.

2. Modern Operating Systems Tanenbaum, A. S., & Bos, H. (2015). *Modern operating systems* (4th ed.). Pearson.

3. Operating Systems Stallings, W. (2018). *Operating systems: Internals and design principles* (9th ed.). Pearson.

4. Operating System Principles Silberschatz, A., Peterson, J. L., & Galvin, P. B. (1991). *Operating system principles*. Wiley.

Suggested Reading

1. Operating Systems: A Concept-Based Approach Dhamdhere, D. M. (2017). *Operating systems: A concept-based approach* (3rd ed.). McGraw Hill.

2. Operating Systems Godbole, A. S. (2017). *Operating systems* (3rd ed.). McGraw Hill Education.

3. Principles of Operating Systems Chauhan, N. (2014). *Principles of operating systems*. Oxford University Press.

4. Operating Systems Deitel, H. M., Deitel, P. J., & Choffnes, D. R. (2003). *Operating systems* (3rd ed.). Pearson Education.

SGOU

Space for Learner Engagement for Objective Questions

Learners are encouraged to develop objective questions based on the content in the paragraph as a sign of their comprehension of the content. The Learners may reflect on the recap bullets and relate their understanding with the narrative in order to frame objective questions from the given text. The University expects that 1 - 2 questions are developed for each paragraph. The space given below can be used for listing the questions.

SGOU

2 UNIT Virtual Memory Management

Learning Outcomes

After completion of this unit, the learner will be able to:

- understand the concept and need for virtual memory
- explain demand paging and page fault handling
- analyze page replacement algorithms and their performance
- identify issues like thrashing and methods to prevent it

Background

As computer systems evolved, the limitation of physical memory became a major challenge. Early systems required programs to be fully loaded into memory, restricting execution of large applications. Virtual memory was introduced to overcome this limitation by allowing partial loading of programs. By combining main memory and secondary storage, virtual memory provides the illusion of large memory space. This development enabled efficient multitasking, better resource utilization, and improved system performance.

Keywords

Virtual Memory, Paging, Page Table, Demand Paging, Page Fault, Thrashing, Logical Address, Physical Address

SGOU

Discussion

2.2.1 Introduction

Virtual Memory is a fundamental concept in modern operating systems that enables efficient memory utilization by allowing programs to execute even when they are not completely loaded into main memory. It creates the illusion of a large, continuous memory space for processes, even though the actual physical memory (RAM) may be limited. This is achieved by using a combination of main memory and secondary storage (usually a hard disk). The operating system automatically transfers data between these two levels of memory as needed.

In earlier computing systems, a process had to be fully loaded into memory before execution could begin. This approach created several limitations, such as inability to run large programs, inefficient memory usage, and reduced system performance when multiple processes were involved. Virtual memory overcomes these issues by dividing programs into smaller units called pages and loading only the necessary pages into memory during execution. This improves flexibility, efficiency, and overall system performance.

2.2.2 Basic Concept of virtual memory

The basic idea behind virtual memory is to divide a program into fixed-size blocks known as pages. These pages are stored on secondary storage and are brought into main memory only when required. The operating system keeps track of which pages are currently in memory and which are on disk. When a program executes, only the actively used pages are loaded, while inactive pages remain on disk.

This approach significantly reduces memory wastage because unused parts of a program do not occupy valuable RAM space. It also allows multiple processes to share memory efficiently, enabling multitasking and better utilization of system resources.

In a virtual memory system, there are two types of addresses: logical and physical. The logical address is generated by the CPU during program execution and represents the address within the process's virtual address space. The physical address, on the other hand, refers to the actual location in the main memory (RAM).

The operating system uses a data structure called a page table to map logical addresses to physical addresses. This mapping is essential for translating program instructions into actual memory locations. By separating logical and physical memory, virtual memory provides abstraction and flexibility, allowing processes to operate independently without worrying about actual memory locations.

Fig 2.2.1 shows how only some pages of a process are loaded into physical memory, while others remain on disk. The mapping between pages and frames is handled by the page table.

SGOU

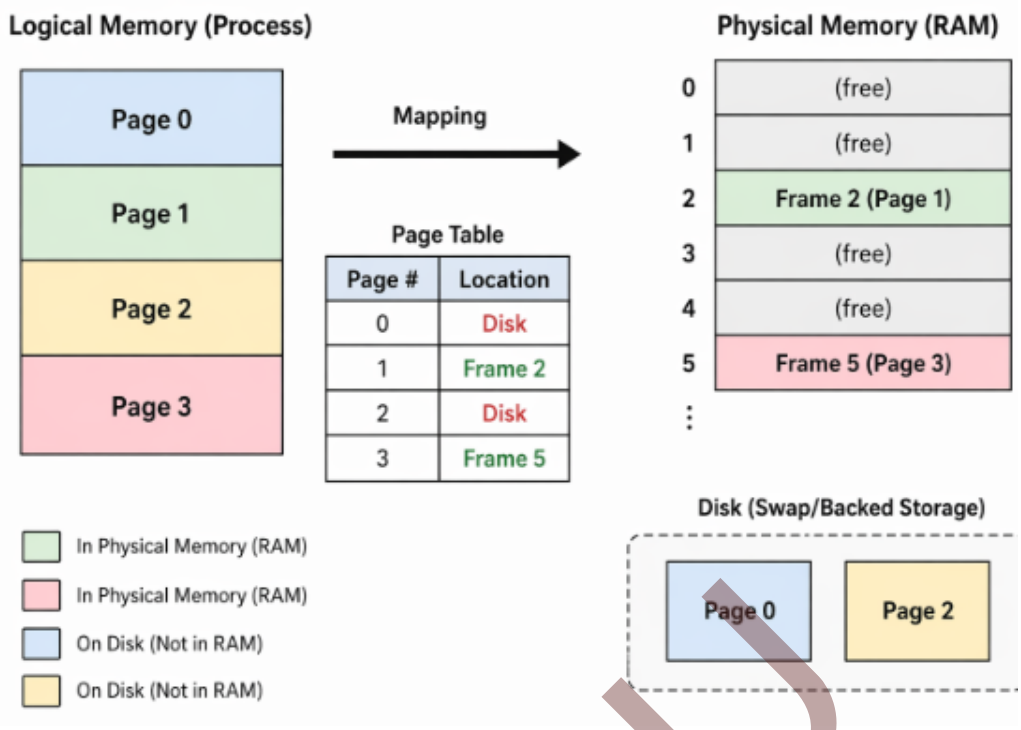


Fig 2.2.1 Virtual Memory Paging

Need for Virtual Memory

Virtual memory is required to overcome the limitations of physical memory and to improve system performance. It allows execution of programs larger than the available RAM by loading only necessary portions into memory. It also supports multiprogramming, where multiple processes can run simultaneously without requiring full memory allocation for each.

Additionally, virtual memory improves memory utilization by ensuring that only active data occupies RAM. It reduces I/O operations by avoiding unnecessary loading of unused program parts. Overall, it plays a crucial role in enhancing efficiency, flexibility, and scalability of modern computer systems.

2.2.3 Demand Paging

Demand Paging is a virtual memory technique in which pages are loaded into main memory only when they are actually required during execution. Instead of preloading all pages of a process, the operating system delays loading until a page is referenced by the CPU. This approach is also known as lazy loading because it avoids unnecessary work.

The central idea of demand paging is to minimize memory usage and improve efficiency by loading only essential pages. When a program starts execution, only a few pages are loaded initially. As execution continues, additional pages are brought into memory when needed. Pages that are not currently required remain stored on disk, saving memory space.

SGOU

This approach ensures optimal utilization of memory resources and allows more processes to reside in memory simultaneously.

Working of Demand Paging

The working of demand paging involves several steps. When a process begins execution, it requests access to a page. The operating system checks the page table to determine whether the page is present in memory. If the page is already available, execution continues without interruption. However, if the page is not present, a page fault occurs.

The operating system then loads the required page from disk into memory, updates the page table, and resumes execution. This dynamic loading mechanism ensures efficient use of memory but may introduce slight delays due to page faults. The fig 2.2.2 shows the working of demand paging.

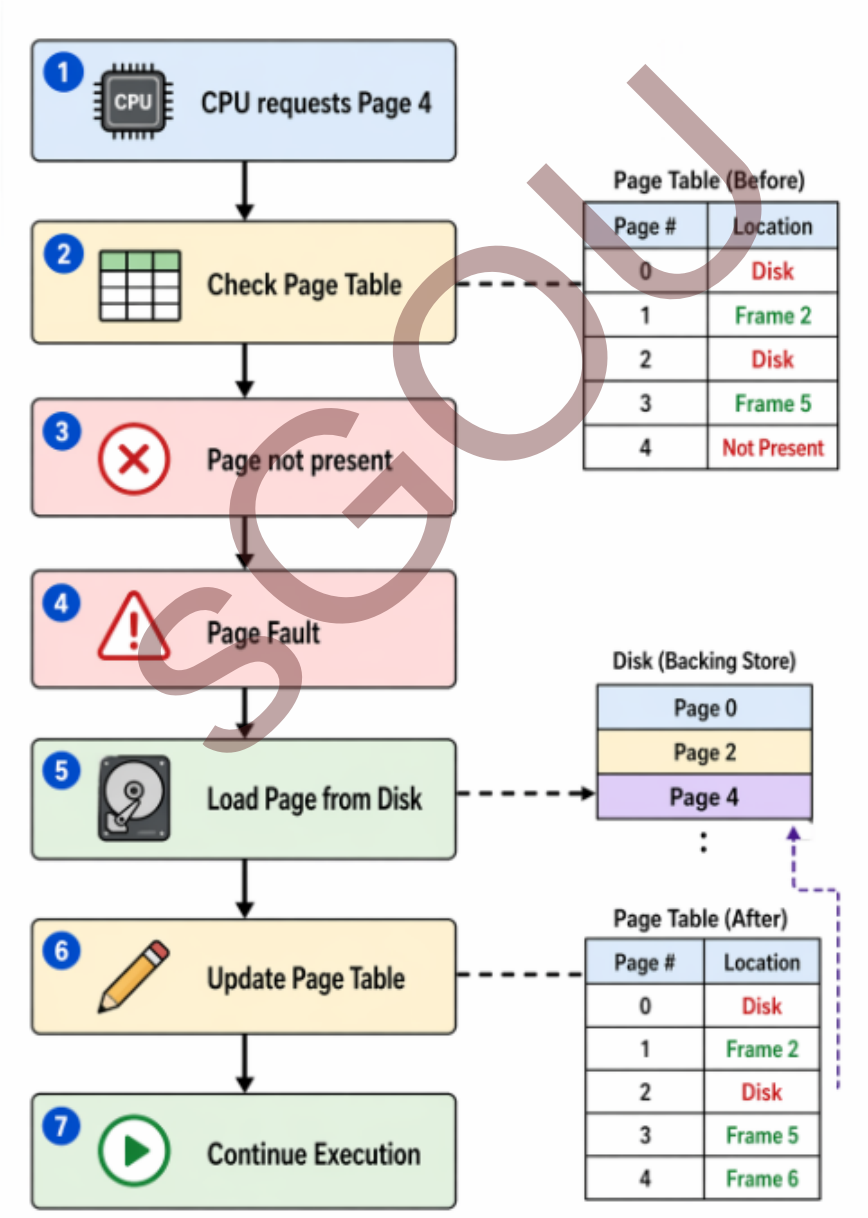


Fig 2.2.2 Page fault handling

SGOU

2.2.4 Page Table Structure

The page table is an essential component that keeps track of the status of each page. It contains entries for each page, including the frame number where the page is stored and a valid/invalid bit indicating whether the page is currently in memory.

A valid bit value of 1 indicates that the page is present in memory, while a value of 0 indicates that the page is stored on disk. This information helps the operating system quickly determine whether a page fault will occur.

Consider a process that consists of five pages (P1 to P5), but only two frames are available in memory. Initially, pages P1 and P2 are loaded. When the process tries to access page P3, which is not in memory, a page fault occurs. The operating system then replaces one of the existing pages (say P1) and loads P3 into memory.

This example illustrates how demand paging dynamically manages memory by loading and replacing pages as needed.

Advantages

Demand paging offers several advantages, including reduced memory usage, faster program startup, and efficient utilization of system resources. By loading only necessary pages, it minimizes memory wastage and allows more processes to run concurrently.

Disadvantages

Despite its benefits, demand paging has some drawbacks. Frequent page faults can slow down system performance, especially if the required pages are not readily available. It also increases system complexity due to the need for page management and replacement algorithms.

2.2.5 Benefits Of Virtual Memory

1. Efficient Memory Utilization

Virtual memory ensures that only the required portions of a program occupy main memory, thereby reducing wastage. This allows the system to make better use of available memory resources.

2. Execution of Large Programs

One of the most significant advantages of virtual memory is its ability to run programs larger than the physical memory. This is possible because only parts of the program are loaded at any given time.

3. Multiprogramming

Virtual memory supports multiprogramming by allowing multiple processes to share memory simultaneously. This increases system throughput and improves overall performance.

SGOU

4. CPU Utilization

Since programs do not need to be fully loaded before execution, the CPU can begin processing earlier. This reduces idle time and improves CPU efficiency.

5. Memory Protection

Each process operates in its own virtual address space, preventing unauthorized access to other processes' memory. This enhances system security and stability.

6. Reduced I/O Operations

By loading only required pages, virtual memory reduces unnecessary data transfers between disk and memory, improving system efficiency.

For example, consider a program of size 10MB running on a system with only 4MB RAM. Without virtual memory, the program cannot execute. However, with virtual memory, only the required 3MB portion is loaded into memory, while the rest remains on disk. This allows the program to run smoothly despite limited memory.

2.2.6 Page Fault Handling

A page fault occurs when a process attempts to access a page that is not currently available in main memory. It is a common event in systems using demand paging and is handled by the operating system.

Page faults can occur due to several reasons, such as accessing a page that has not yet been loaded, accessing a page that has been swapped out, or attempting to access an invalid memory location.

2.2.6.1 Steps in Page Fault Handling

When a page fault occurs, the operating system follows a sequence of steps. First, it traps the fault and checks whether the memory reference is valid. If invalid, the process is terminated. If valid, the OS locates the required page on disk and searches for a free frame in memory. If no free frame is available, a page replacement algorithm is used to free space.

The required page is then loaded into memory, the page table is updated, and the interrupted instruction is restarted. This ensures that the process continues execution correctly.

For instance, if a process tries to access page 5 and it is not in memory, a page fault occurs. The operating system loads page 5 from disk into memory, updates the page table, and resumes execution.

2.2.6.2 Page Replacement Algorithms

When memory is full, the operating system must replace an existing page. Common algorithms include FIFO (First In First Out), LRU (Least Recently Used), LFU (Least

SGOU

Frequently Used) and Optimal. These algorithms help minimize page faults and improve performance.

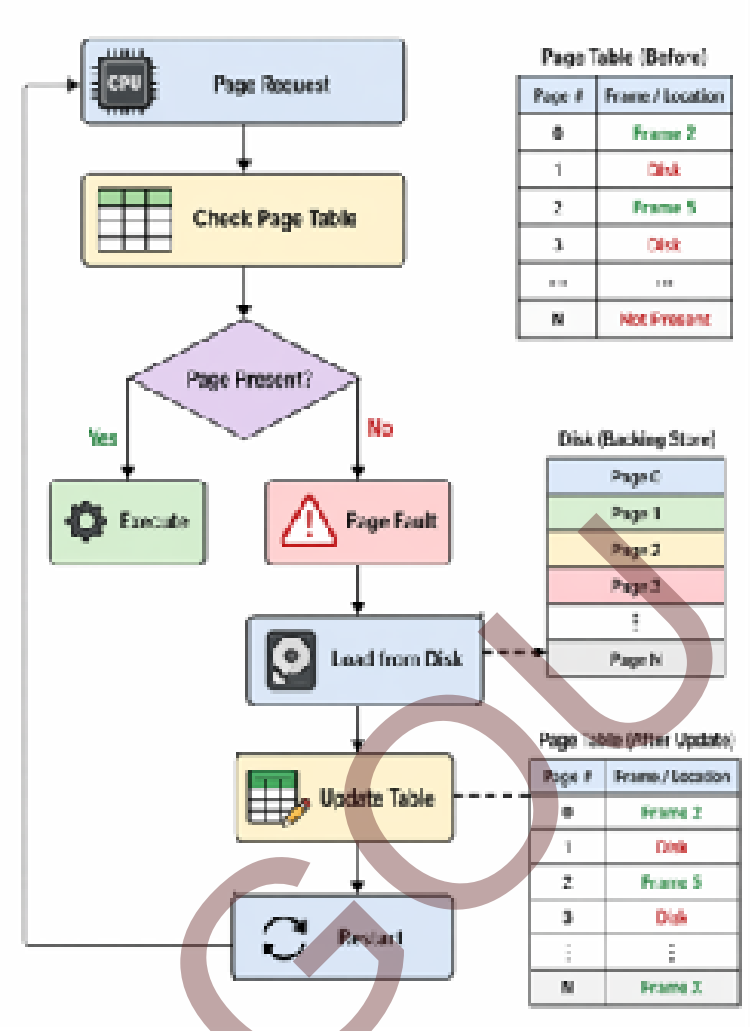


Fig 2.2.3 Memory paging process

Types of Page Replacement Algorithms

1. FIFO (First-In, First-Out)

- Replaces the page that entered memory first.
- Simple but may cause more page faults.

2. LRU (Least Recently Used)

- Replaces the page that has not been used for the longest time.
- Provides better performance than FIFO.

3. Optimal (OPT)

- Replaces the page that will not be used for the longest time in the future.
- Gives the minimum number of page faults but is not practical.

SGOU

4. LFU (Least Frequently Used)

- Replaces the page with the lowest usage count.

2.2.7 Thrashing

Thrashing is a condition where the system spends excessive time swapping pages between memory and disk instead of executing processes. It severely degrades system performance.

Thrashing occurs when there are too many processes in memory or when each process does not have enough frames to operate efficiently. This leads to frequent page faults.

In a thrashing state, processes continuously request pages that are not in memory. The operating system keeps replacing pages, but the required pages are quickly replaced again, leading to continuous swapping.

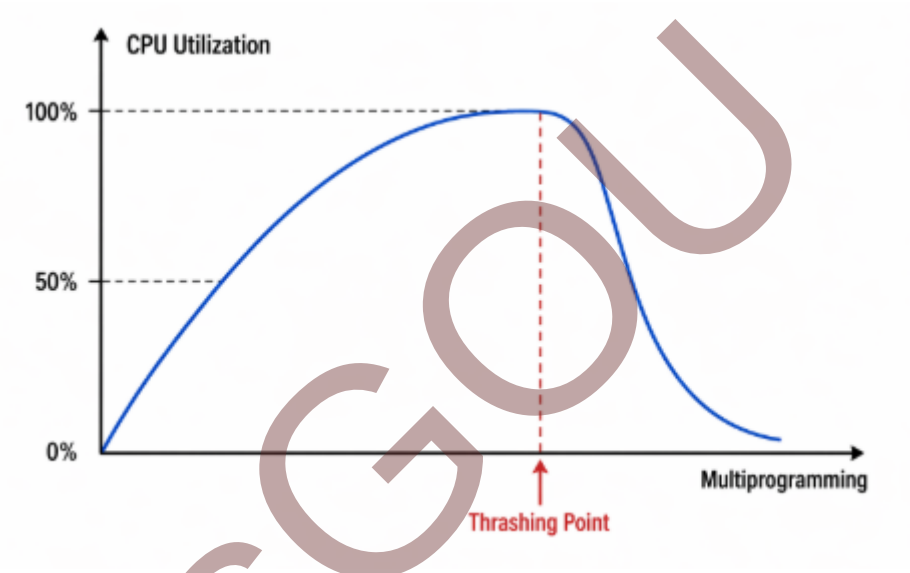


Fig 2.2.4 CPU utilization vs multiprogramming graph

The fig 2.2.4 shows that CPU utilization increases with multiprogramming up to an optimal level, but after the thrashing point, it drops sharply. This relates to thrashing, where excessive processes in memory lead to frequent page faults and continuous swapping between memory and disk. As a result, the system spends more time on paging instead of execution, causing low CPU utilization and reduced performance.

Symptoms: Thrashing is characterized by a high page fault rate, low CPU utilization, and excessive disk activity. The system becomes slow and unresponsive.

Example: Consider a process that requires six pages but is allocated only two frames. As it executes, it constantly replaces pages, resulting in continuous page faults and poor performance.

Prevention of Thrashing

Thrashing can be prevented by reducing the number of processes, allocating sufficient

SGOU

frames, using efficient page replacement algorithms, and applying the working set model.

The working set is the set of pages currently being used by a process. Keeping these pages in memory reduces page faults and improves performance.

Virtual Memory Management is a powerful technique that enhances system efficiency by allowing flexible and efficient use of memory. While it provides many benefits such as improved performance and multitasking, careful management is required to avoid issues like thrashing.

Summarized Overview

Virtual memory is a memory management technique that allows execution of large programs by loading only required pages into memory. It uses paging and page tables to map logical addresses to physical addresses. Demand paging loads pages only when needed, reducing memory usage. Page faults occur when required pages are not in memory, and they are handled by the operating system. However, excessive paging can lead to thrashing, reducing performance. Proper memory management techniques help maintain efficiency and stability.

Assignments

1. Explain the concept of virtual memory with a diagram.
2. Describe demand paging and its working.
3. What is a page fault? Explain its handling steps.
4. Explain page replacement algorithms (FIFO, LRU, Optimal).
5. Define thrashing and its causes.
6. Discuss advantages and disadvantages of virtual memory.
7. Explain the structure and function of a page table.

SGOU

Reference

1. Nutt, G. J. (2004). *Operating systems: A modern perspective* (2nd ed.). Addison-Wesley.

2. Milenkovic, M. (1992). *Operating systems: Concepts and design* (2nd ed.). McGraw-Hill.

3. Tanenbaum, A. S. (2001). *Operating systems: Design and implementation* (2nd ed.). Prentice Hall.

4. Hennessy, J. L., & Patterson, D. A. (2017). *Computer architecture: A quantitative approach* (6th ed.). Morgan Kaufmann.

Suggested Reading

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts*. Wiley.

2. Tanenbaum, A. S., & Bos, H. (2015). *Modern operating systems*. Pearson.

3. Stallings, W. (2018). *Operating systems: Internals and design principles*. Pearson.

4. Peterson, J. L., & Silberschatz, A. (1985). *Operating system concepts*. Addison-Wesley.

SGOU

Space for Learner Engagement for Objective Questions

Learners are encouraged to develop objective questions based on the content in the paragraph as a sign of their comprehension of the content. The Learners may reflect on the recap bullets and relate their understanding with the narrative in order to frame objective questions from the given text. The University expects that 1 - 2 questions are developed for each paragraph. The space given below can be used for listing the questions.

SGOU

3 UNIT

Page Replacement Algorithms

Learning Outcomes

After completion of this unit, the learner will be able to:

- define page replacement algorithm and page fault
- list different types of page replacement algorithms
- identify the steps involved in FIFO, Optimal, and LRU algorithms
- recall the concept of page frames and memory replacement
- state the purpose of page replacement in memory management

SGOU

Background

In daily life, we often manage limited space by deciding what to keep and what to remove. For example, when your phone storage is full, you may delete old photos or unused apps to make space for new ones. Usually, you might remove the least important or least recently used items. This simple decision-making process helps in efficiently using the available space.

Similarly, a computer system has limited main memory (RAM), but it needs to handle many programs and data at the same time. You may already know that not all data can be kept in memory at once, so the system must decide which data to keep and which to remove when new data needs to be loaded. This is where the concept of page replacement becomes important.

In this topic, you will learn how the operating system manages memory using page replacement algorithms. You will understand how decisions are made when memory is full, and how different techniques like FIFO, Optimal, and LRU help in selecting pages for replacement. This will help you understand how memory is used efficiently and how system performance is improved.

Keyconcepts

FIFO, Page Replacement, LRU, Optimal

Discussion

2.3.1 Page Replacement Algorithm

A page replacement algorithm is a fundamental component of memory management in operating systems that employ virtual memory. In such systems, processes are divided into fixed-size pages, and only a subset of these pages is loaded into main memory at any given time. This approach improves memory utilization but also introduces the possibility of page faults, which occur when a process attempts to access a page that is not currently present in memory.

When a page fault occurs and there are no free frames available in main memory, the operating system must remove one of the existing pages to make space for the required page. This process of selecting a page to be removed is known as page replacement, and the method used to make this decision is referred to as a page replacement algorithm.

The examples of page replacement algorithms are:

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

2.3.2 FIFO (First-In First-Out) Algorithm

The First-In First-Out (FIFO) Page Replacement Algorithm is one of the simplest page replacement techniques used in operating systems. This algorithm follows the principle that the page which has been in memory for the longest period of time is the first one to be removed when a new page needs to be loaded and no free frames are available.

In FIFO, pages are maintained in a queue structure. When a new page is loaded into memory, it is placed at the rear (end) of the queue. When a page replacement is required, the page at the front of the queue, which is the oldest page in memory, is selected as the page to be replaced and removed. The newly required page is then inserted into the free frame.

Example 1: Consider page reference string 1, 2, 3, 4, 1, 2, 2, 5 with 3-page frames. Find the number of page faults using FIFO Page Replacement Algorithm.

SGOU

Step 1: The memory is initially empty, page 1 is not present in memory, resulting in a page miss (page fault); therefore, page 1 is loaded into the first frame.

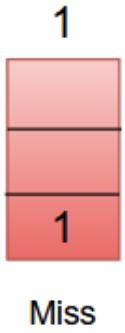


Fig. 2.3.1 Insertion of Page 1

Step 2: When page 2 is referenced, it is not present in memory, resulting in a page miss (page fault); hence, page 2 is loaded into the next available empty frame.

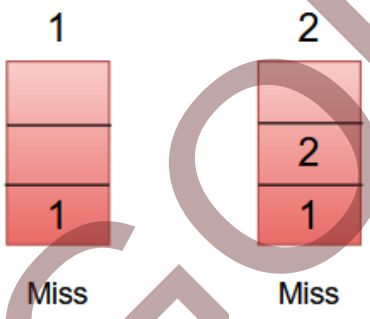


Fig. 2.3.2 Insertion of Page 2

Step 3: When page 3 is referenced, it is not present in memory, resulting in a page miss; therefore, page 3 is loaded into the remaining empty frame.

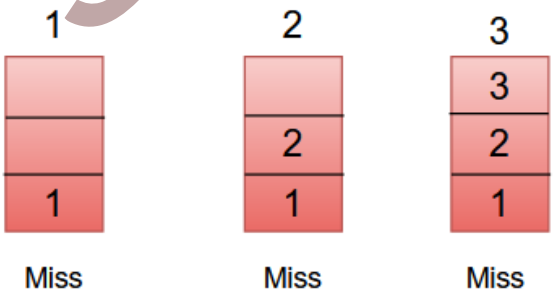


Fig. 2.3.3 Insertion of Page 3

Step 4: When page 4 is referenced, it is not present in memory, resulting in a page miss; since all frames are full, the oldest page (page 1) is selected as the page to be replaced according to the FIFO algorithm, and page 4 is loaded into memory.

SGOU

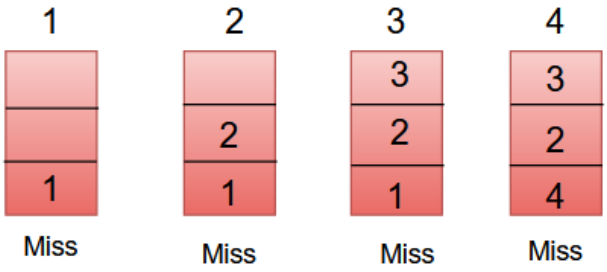


Fig. 2.3.4 Insertion of Page 4

Step 5: When page 1 is referenced again, it is not present in memory, resulting in a page miss; since all frames are full, the next oldest page (page 2) is selected as the page to be replaced according to the FIFO algorithm, and page 1 is loaded into memory.

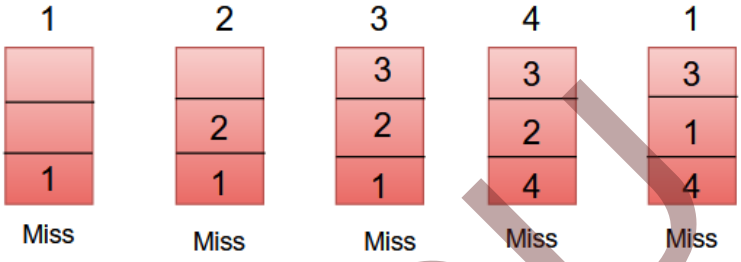


Fig. 2.3.5 Insertion of Page 1

Step 6: When page 2 is referenced, it is not present in memory, resulting in a page miss; since all frames are full, the next oldest page(page 3) is selected for replacement according to the FIFO algorithm, and page 2 is loaded into memory.

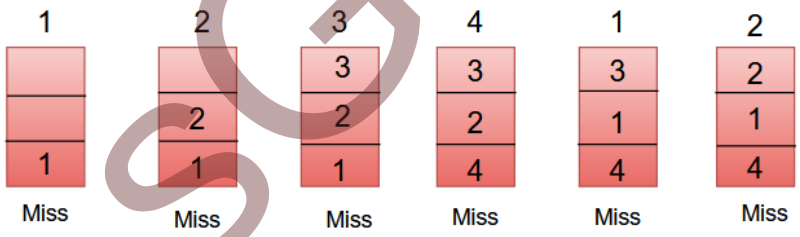


Fig. 2.3.6 Insertion of Page 2

Step 7: When page 2 is referenced again, it is already present in memory, resulting in a page hit; therefore, no page replacement is required and the contents of memory remain unchanged.



Fig. 2.3.7 Again Page 2

SGOU

Step 8: When page 5 is referenced, it is not present in memory, resulting in a page miss; since all frames are full, the next oldest page (page 4) is selected as the page to be replaced according to the FIFO algorithm, and page 5 is loaded into memory.

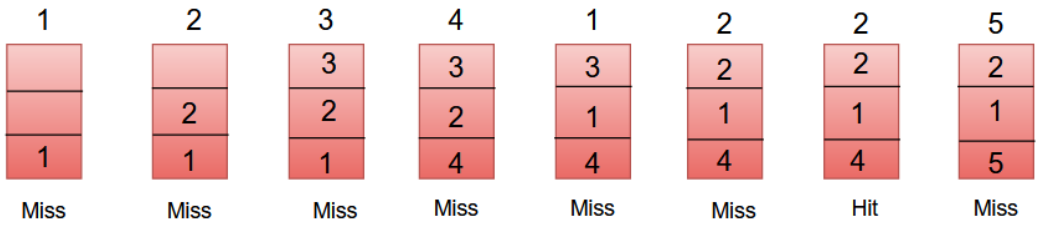


Fig. 2.3.8 Insertion of Page 5

The total number of page faults (page misses) that occurred during the execution of the FIFO page replacement algorithm is 7.

2.3.3 Optimal Page Replacement Algorithm

The Optimal Page Replacement Algorithm is a page replacement technique in which the page that will not be used for the longest period of time in the future is selected as the page to be replaced. This algorithm makes use of future knowledge of page references to make the best possible replacement decision.

When a page fault occurs and all frames in memory are occupied, the operating system examines the future sequence of page references and identifies the page whose next use is farthest in the future or may not be used again at all. This page is then replaced by the required page. By doing so, the algorithm ensures the minimum possible number of page faults.

Example: Consider the page references 1, 2, 3, 4, 1, 2, 5 with a 3-page frame. Find number of page faults using Optimal Page Replacement Algorithm.

Step 1: The first page, Page 1 is not present in memory, a page miss occurs; therefore, page 1 is loaded into the first frame.

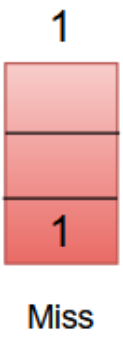


Fig. 2.3.9 Insertion of Page 1

SGOU

Step 2: When page 2 is referenced, it is not present in memory, resulting in a page miss (page fault); hence, page 2 is loaded into the next available frame.

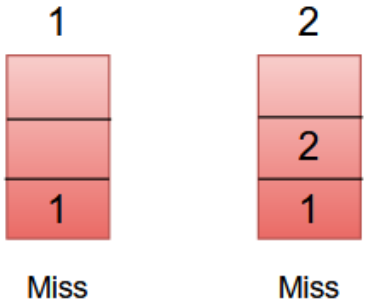


Fig. 2.3.10 Insertion of 2

Step 3: When page 3 is referenced, it is not present in memory, resulting in a page miss; therefore, page 3 is loaded into the remaining empty frame.

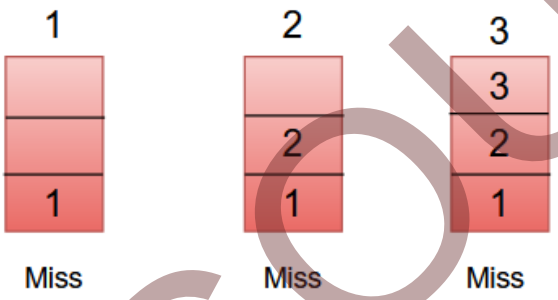


Fig. 2.3.11 Insertion of 3

Step 4: When page 4 is referenced, it is not present in memory, resulting in a page miss; since all frames are full, the operating system looks ahead in the reference string and determines that page 3 will not be used for the longest period in the future; hence, page 3 is selected as the page to be replaced, and page 4 is loaded into memory.

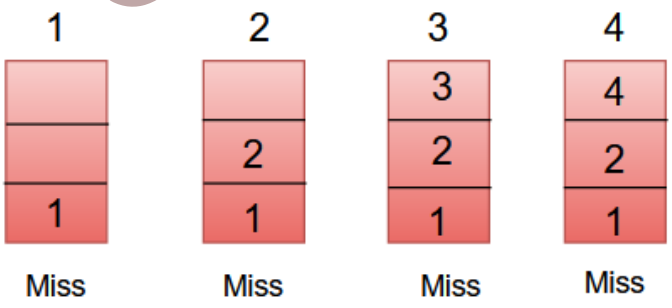


Fig. 2.3.12 Insertion of 4

Step 5: When page 1 is referenced, it is already present in memory, resulting in a page hit; therefore, no page replacement is required.

SGOU

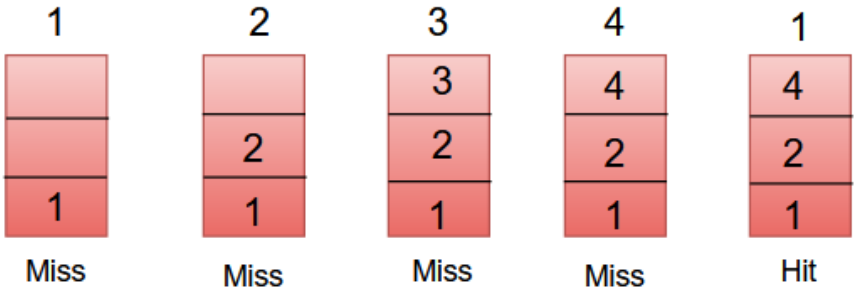


Fig. 2.3.13 Insertion of 1

Step 6: When page 2 is referenced, it is already present in memory, resulting in a page hit; hence, no change is made to the memory.

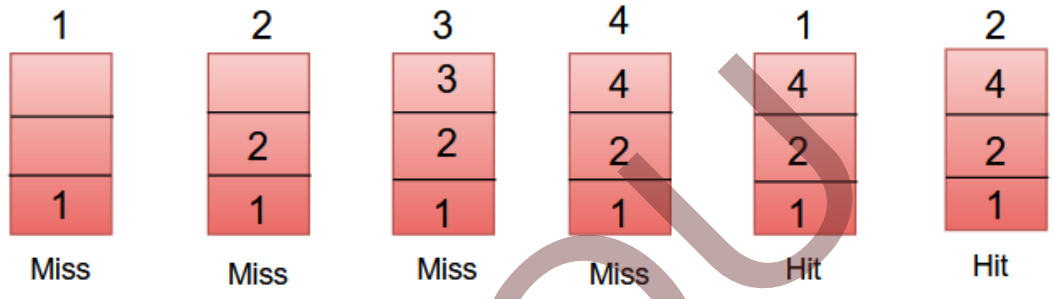


Fig. 2.3.14 Insertion of 2

Step 7: When page 5 is referenced, it is not present in memory, resulting in a page miss; since all frames are full, the operating system examines the future page references and observes that pages 1, 2, and 4 will not be used again. Therefore, any one of these pages can be selected as the page to be replaced. In this case, page 4 is chosen for replacement, and page 5 is loaded into memory.



Fig. 2.3.15 Insertion of 5

The total number of page faults (page misses) that occurred during the execution of the Optimal Page Replacement Algorithm is 5.

2.3.4 LRU (Least Recently Used) Algorithm

The Least Recently Used (LRU) Page Replacement Algorithm is a widely used technique in which the page that has not been used for the longest period of time in the

SGOU

past is selected as the page to be replaced. This algorithm is based on the assumption that pages that have been used recently are likely to be used again in the near future.

When a page fault occurs and all memory frames are occupied, the operating system examines the past usage of pages in memory and identifies the page that has not been accessed for the longest time. This page is then removed from memory, and the required page is loaded into the freed frame.

Example Consider the page reference string 1, 2, 3, 4, 1, 2, 5 with 3-page frames. Find number of page faults using LRU Page Replacement Algorithm.

Step 1: Page = 1

Since page 1 is not present in memory, a page miss (page fault) occurs; therefore, page 1 is loaded into the first frame.

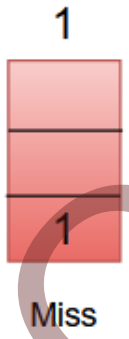


Fig. 2.3.16 Insertion of Page 1

Step 2: Page = 2

When page 2 is referenced, it is not present in memory, resulting in a page miss; hence, page 2 is loaded into the next available frame.

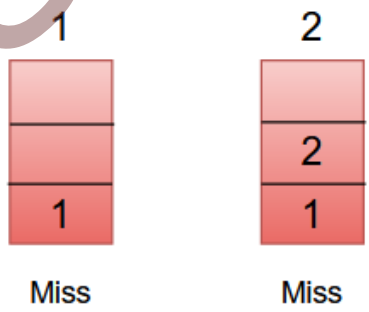


Fig. 2.3.17 Insertion of Page 2

Step 3: Page = 3

When page 3 is referenced, it is not present in memory, resulting in a page miss; therefore, page 3 is loaded into the remaining empty frame.

SGOU

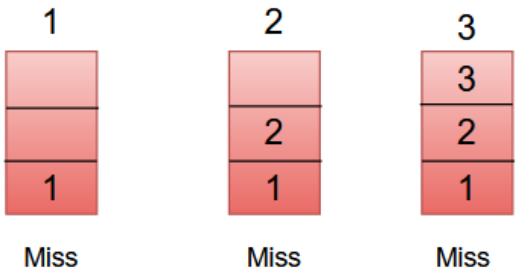


Fig. 2.3.18 Insertion of Page 3

Step 4: Page = 4

When page 4 is referenced, it is not present in memory, resulting in a page miss; since all frames are full, the least recently used page (page 1) is selected as the page to be replaced, and page 4 is loaded into memory.

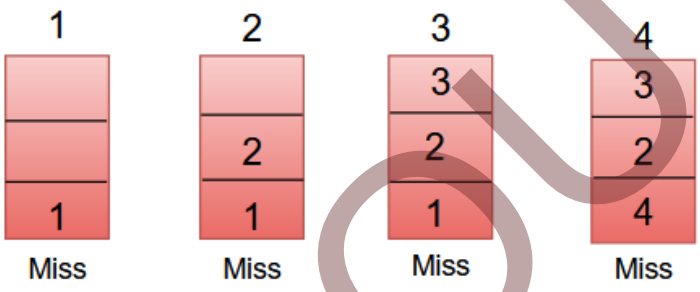


Fig. 2.3.19 Insertion of Page 4

Step 5: Page = 1

When page 1 is referenced, it is not present in memory, resulting in a page miss; since all frames are full, the least recently used page (page 2) is selected as the page to be replaced, and page 1 is loaded into memory.

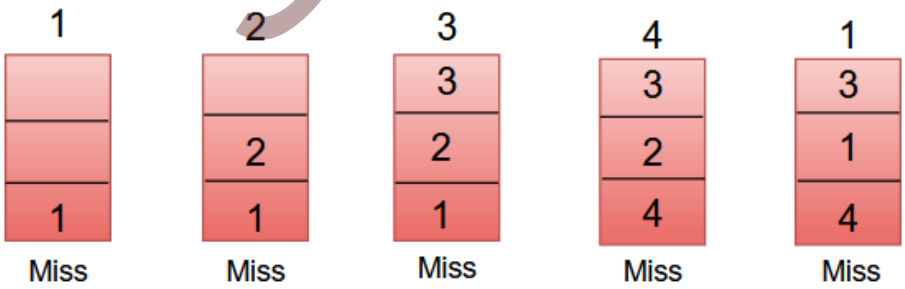


Fig. 2.3.20 Insertion of Page 1

Step 6: Page = 2

When page 2 is referenced, it is not present in memory, resulting in a page miss; since

SGOU

all frames are full, the least recently used page (page 3) is selected as the page to be replaced, and page 2 is loaded into memory.

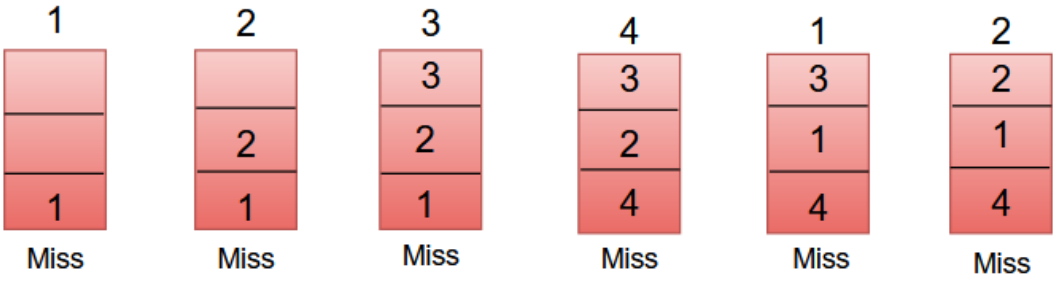


Fig. 2.3.21 Insertion of Page 2

Step 7: Page = 5

When page 5 is referenced, it is not present in memory, resulting in a page miss; since all frames are full, the least recently used page (page 4) is selected as the page to be replaced, and page 5 is loaded into memory.

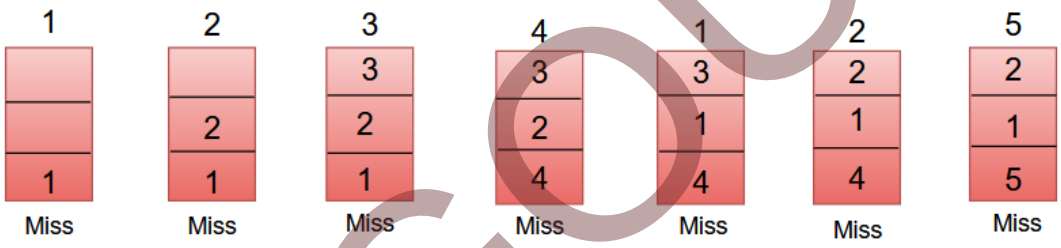


Fig. 2.3.22 Insertion of Page 5

The total number of page faults (page misses) is 7.

2.3.5 Belady's Anomaly

Belady's Anomaly is a phenomenon in which increasing the number of memory frames can sometimes result in an increase in the number of page faults. This is an unexpected behavior because, normally, providing more memory frames is expected to reduce the number of page faults.

Belady's Anomaly can occur in some page replacement algorithms, particularly the FIFO (First-In First-Out) algorithm. In FIFO, pages are replaced based on their arrival order without considering how frequently or recently they are being used. As a result, adding an extra frame may sometimes change the order of page replacement and cause more page faults.

Example of Belady's Anomaly

Consider the following page reference string:

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

SGOU

Using the FIFO page replacement algorithm:

- With 3 frames, the number of page faults is 9.
- With 4 frames, the number of page faults becomes 10.

Thus, although the number of frames is increased from 3 to 4, the number of page faults also increases from 9 to 10. This unexpected increase is called Belady's Anomaly.

Summarized Overview

Page replacement is a key concept in memory management that enables an operating system to utilize limited main memory efficiently. In systems that use virtual memory, programs are divided into pages, but only a few pages can be kept in memory at a time. When a process tries to access a page that is not currently in memory, a page fault occurs. If all memory frames are already occupied, the system must decide which existing page to remove to load the required page. This decision is made using page replacement algorithms.

Different algorithms follow different strategies to select the page for replacement. The First-In First-Out (FIFO) algorithm removes the page that has been in memory for the longest time. The Optimal algorithm selects the page that will not be used for the longest time in the future, thereby minimizing page faults. The Least Recently Used (LRU) algorithm replaces the page that has not been used for the longest time in the past.

These algorithms help in improving memory utilization and reducing the number of page faults. However, in algorithms such as FIFO, increasing the number of memory frames may sometimes increases the number of page faults, a phenomenon known as Belady's Anomaly.Understanding these techniques is important for analyzing system performance and designing efficient memory management systems.

SGOU

Assignments

1. Define page replacement algorithm and explain its role in memory management.
2. Explain the concept of page fault and its impact on system performance.

3. Describe FIFO page replacement algorithm with a suitable example.
4. Explain the working of Optimal page replacement algorithm with an example.
5. Describe LRU page replacement algorithm with a suitable example.
6. Compare FIFO, Optimal, and LRU algorithms based on their performance.
7. Explain how page replacement helps in efficient memory utilization.
8. Solve a problem using FIFO page replacement algorithm and calculate the number of page faults.
9. Solve a problem using LRU page replacement algorithm and determine page faults.
10. Explain the differences between page hit and page fault with examples.

Reference

1. Abraham Silberschatz, Peter B. Galvin, & Greg Gagne. (2018). *Operating system concepts* (10th ed.). Wiley.
2. Abraham Silberschatz, Peter B. Galvin, & Greg Gagne. (2012). *Operating system concepts* (9th ed.). Wiley.
3. Stallings, W. (2018). *Operating systems: Internals and design principles* (9th ed.). Pearson.
4. Tanenbaum, A. S., & Bos, H. (2015). *Modern operating systems* (4th ed.). Pearson.
5. Galvin, P. B., Silberschatz, A., & Gagne, G. (2004). *Operating system concepts* (7th ed.). Wiley.

SGOU

Suggested Reading

1. Tanenbaum, A. S. (2009). *Modern operating systems (3rd ed.).* Pearson Prentice Hall.

2. Dhamdhere, D. M. (2003). *Operating systems: A concept-based approach.* McGraw-Hill Education.

3. Deitel, H. M., Deitel, P. J., & Choffnes, D. R. (2004). *Operating systems (3rd ed.).* Pearson.

4. Nutt, G. J. (2004). *Operating systems (3rd ed.).* Addison-Wesley.

Space for Learner Engagement for Objective Questions

Learners are encouraged to develop objective questions based on the content in the paragraph as a sign of their comprehension of the content. The Learners may reflect on the recap bullets and relate their understanding with the narrative in order to frame objective questions from the given text. The University expects that 1 - 2 questions are developed for each paragraph. The space given below can be used for listing the questions.

SGOU

SGOU

4 UNIT Segmentation and Paging

Learning Outcomes

After completion of this unit, the learner will be able to:

- explain contiguous and non-contiguous memory allocation techniques
- describe different memory allocation strategies such as First-Fit, Best-Fit, and Worst-Fit
- explain the working of paging and segmentation in memory management
- compare paging and segmentation based on memory organization and fragmentation

SGOU

Background

In a computer system, many programs and processes execute simultaneously, and each process requires memory for its execution. Since the main memory of a computer is limited, the operating system must manage memory efficiently to ensure that multiple processes can run smoothly without interfering with one another. Early computer systems used simple memory allocation methods in which processes were stored in continuous blocks of memory. Although these methods were easy to implement, they often resulted in poor memory utilization and fragmentation problems.

To improve memory usage and support efficient execution of programs, operating systems introduced various memory allocation techniques and strategies. Methods such as First-Fit, Best-Fit, and Worst-Fit were developed to allocate available memory spaces effectively. Later, advanced techniques like paging and segmentation were introduced to overcome the limitations of contiguous memory allocation.

These techniques allow processes to be stored more flexibly in memory and improve overall system performance.

Understanding memory allocation methods is important because memory management is one of the core responsibilities of an operating system. Efficient memory management helps in improving system speed, increasing memory utilization, reducing fragmentation, and supporting multitasking in modern computer systems.

Keyconcepts

Contiguous and non-contiguous memory allocation, first-fit, best-fit, worst-fit, page offset, internal fragmentation

Discussion

2.4.1 Contiguous Memory Allocation

As you have already learned in the unit 1 of Block 2, in contiguous memory allocation, a process is stored in a single continuous block of memory. The entire memory space allocated to the process remains together at one location.

The main memory is generally divided into two sections:

- One section for the operating system
- Another section for user processes

Whenever a process requests memory, the operating system allocates one continuous block large enough to satisfy the request.

Contiguous memory allocation is simple and easy to implement. Since the entire process is located together, memory access becomes faster. This method is suitable for smaller computer systems. However, it may lead to memory wastage due to fragmentation.

Since fixed size and variable size partition allocation methods were already discussed earlier, the following section focuses on the important memory allocation strategies used in contiguous allocation.

2.4.1.1 Memory allocation Strategies

When free memory spaces called holes are available, the operating system must decide which hole should be allocated to a process. Different strategies are used for selecting suitable free memory blocks.

SGOU

The commonly used strategies are:

1. First-Fit
2. Best-Fit
3. Worst-Fit

First-Fit Strategy

In the First-Fit strategy, the operating system starts searching memory from the beginning and checks each free memory hole one by one. As soon as it finds the first hole that is large enough to hold the process, it immediately allocates that space to the process. The search then stops without checking the remaining holes.

For example, consider the following free memory holes:

- 100 KB
- 500 KB
- 200 KB
- 300 KB

Now suppose a process of size 180 KB arrives. The operating system first checks the 100 KB hole, but it is too small. Next, it checks the 500 KB hole. Since this hole is large enough, the process is allocated there immediately. The remaining holes are not checked further. See Fig.2.4.1

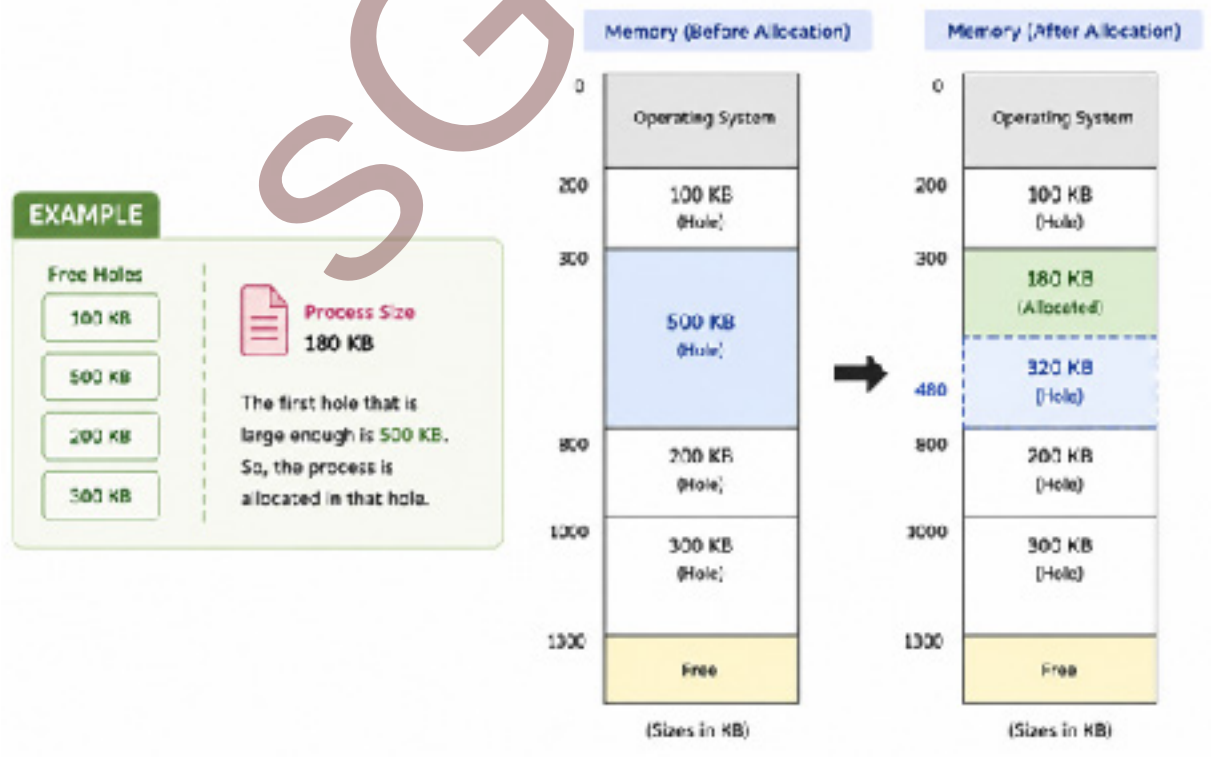


Fig.2.4.1 First-Fit strategy

The First-Fit method is simple and fast because the operating system does not need to search the entire memory before allocating space. This reduces searching time and improves speed.

However, over time, many small unused holes may get created near the beginning of memory. These scattered small holes may not be useful for future processes and can lead to external fragmentation.

Best-Fit Strategy

In the Best-Fit strategy, the operating system searches all the available free memory holes and selects the smallest hole that is large enough to hold the process. The aim is to reduce memory wastage by choosing the most suitable hole.

For example, consider the following free memory holes: 100 KB, 500 KB, 200 KB, 300 KB

Now suppose the process of size 180 KB arrives. The operating system checks all the holes and selects the 200 KB hole because it is the smallest hole that can accommodate the process.This is shown in the Fig.2.4.2

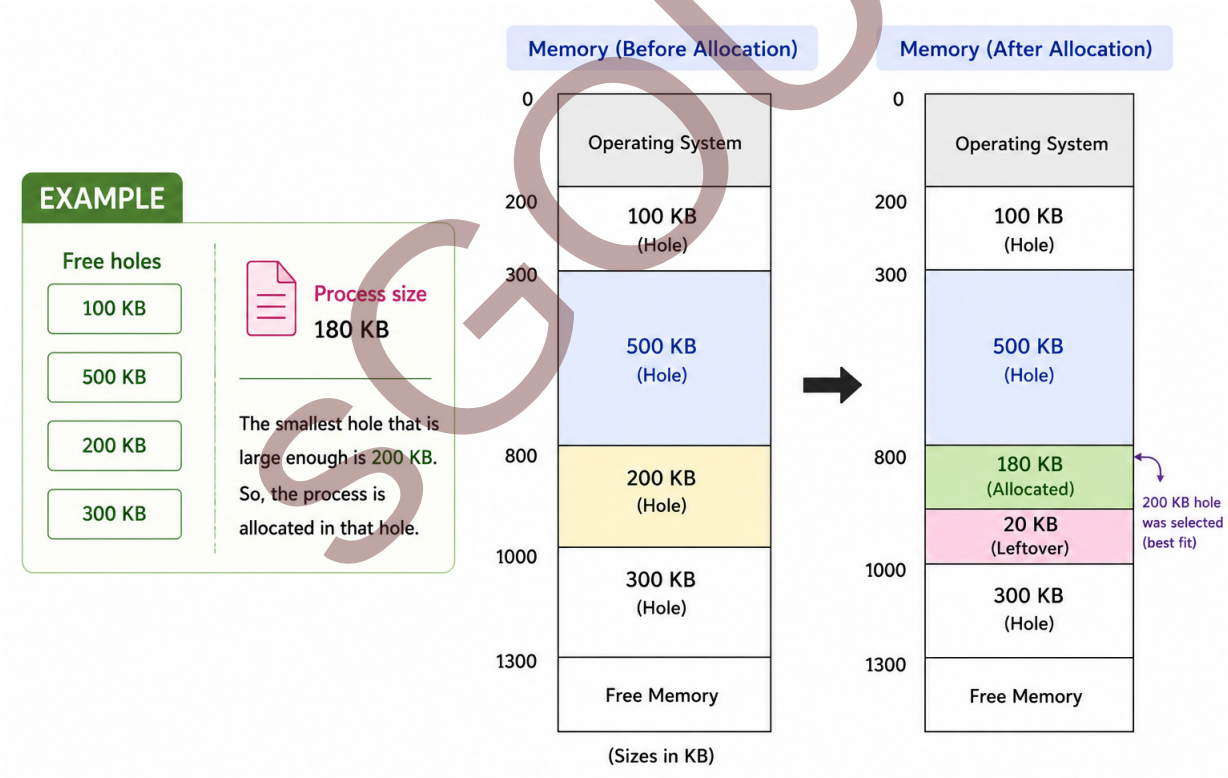


Fig.2.4.2 Best-Fit Strategy

The Best-Fit method tries to utilize memory more efficiently by reducing the amount of unused space left after allocation.

However, the operating system must search the entire memory list before making a decision, which increases searching time. Over time, many very small unused holes may also be created, leading to external fragmentation.

SGOU

Worst-Fit Strategy

In the Worst-Fit strategy, the operating system allocates the largest available memory hole to the process. Instead of selecting the first suitable hole or the smallest suitable hole, this method chooses the biggest free space available in memory.

Consider the following free memory holes:

- 120 KB
- 750 KB
- 250 KB
- 400 KB

Now suppose a process of size 200 KB arrives. The operating system checks all the available holes and selects the 750 KB hole because it is the largest free memory block.

After allocation, a large amount of free space still remains in that hole, which may be used for allocating future processes.

The Worst-Fit method attempts to avoid creating very small leftover holes. However, it often results in poor memory utilization. Since the operating system must search the entire memory list to find the largest hole, the searching time also becomes higher.

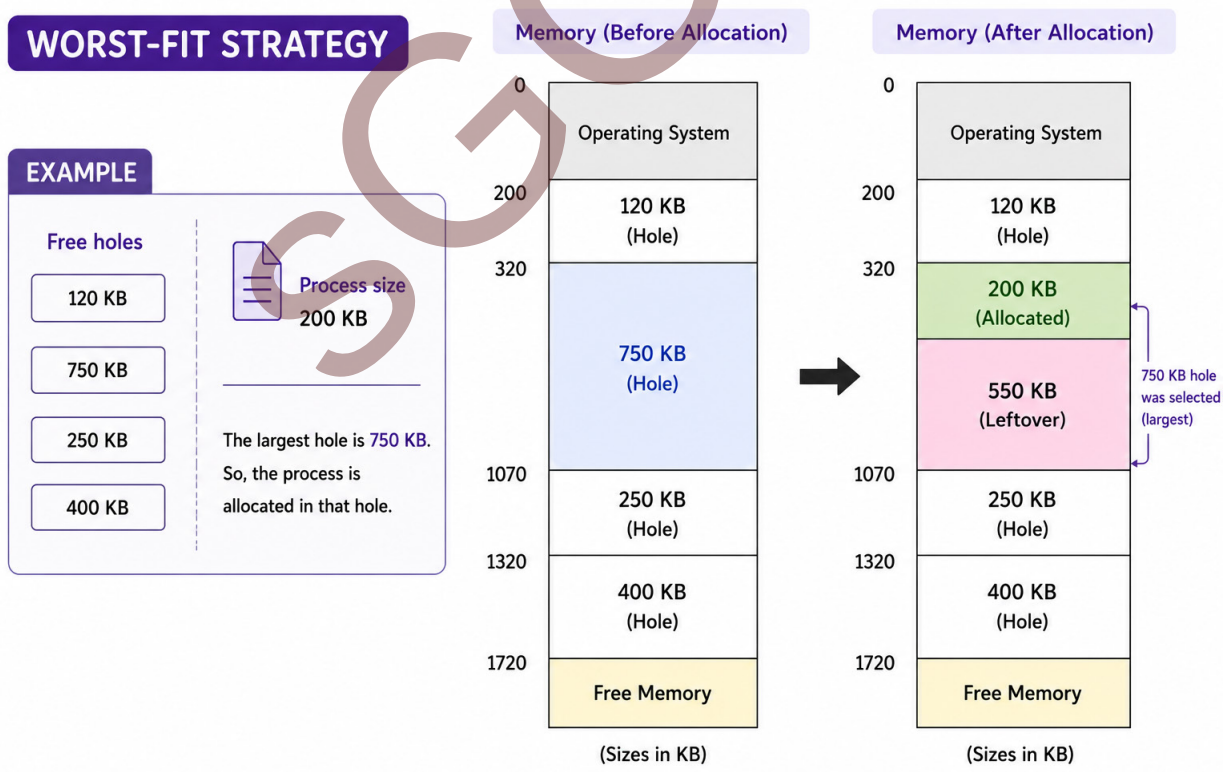


Fig.2.4.3 Worst-Fit Strategy

SGOU

2.4.2 Non-Contiguous Memory Allocation

In non-contiguous memory allocation, the memory space allocated to a process does not remain in one continuous block. Instead, different parts of the process are stored at different locations in memory according to the availability of free space.

In this method, the free memory spaces are scattered throughout the main memory. Whenever a process requires memory, the operating system allocates the available free spaces wherever they are found. As a result, a single process may be divided and placed in multiple memory locations.

This method helps in utilizing memory more efficiently because even small free spaces can be used for allocation. Therefore, memory wastage is reduced compared to contiguous memory allocation. Non-contiguous allocation also helps in reducing external fragmentation.

Although this method improves memory utilization, managing memory becomes more complex because the operating system must keep track of multiple memory locations belonging to the same process.

The commonly used types of non-contiguous memory allocation are:

- Paging
- Segmentation

2.4.2.1 Paging

Paging is one of the most important memory management techniques used in modern operating systems. It was introduced to overcome the limitations of contiguous memory allocation, particularly the problem of external fragmentation. In contiguous allocation, a process must be loaded into a single continuous block of memory, which becomes difficult when free memory spaces are scattered throughout the main memory. Paging solves this problem by allowing a process to be stored in different non-contiguous locations in memory.

In paging, both logical memory and physical memory are divided into fixed-size blocks. The logical memory of a process is divided into units called pages, while the physical memory is divided into units called frames. The size of a page is always equal to the size of a frame. Because of this equal sizing, any page can be placed into any available frame in physical memory.

When a process is executed, the operating system divides the process into pages and loads these pages into available memory frames. Since the pages can be stored in different locations, contiguous free memory is not required. This results in better utilization of memory and completely eliminates external fragmentation.

For example, consider a process consisting of four pages: Page 0, Page 1, Page 2, and Page 3. These pages may be loaded into Frame 5, Frame 2, Frame 9, and Frame 1 respectively. Thus, the pages of a single process are scattered across the main memory, but the operating system maintains their correct order logically.

SGOU

To manage this mapping between pages and frames, the operating system maintains a data structure called the page table. Each process has its own page table. The page table stores the frame number corresponding to each page number. Whenever the CPU generates a logical address, the operating system uses the page table to convert it into the corresponding physical address.

A logical address generated by the CPU is divided into two parts:

- Page Number (p)
- Page Offset (d)

The page number is used as an index into the page table to identify the frame where the page is stored. The page offset indicates the exact location within that page. By combining the frame number with the offset value, the operating system generates the physical memory address.

Suppose a logical address contains:

- Page number = 2
- Offset = 150

If the page table indicates that Page 2 is stored in Frame 9, then the physical address will point to offset 150 within Frame 9.

The size of a page is determined by the hardware architecture and is generally chosen as a power of 2. Common page sizes include 512 bytes, 1 KB, 4 KB, 8 KB, and even several megabytes in modern systems. Choosing a page size as a power of 2 simplifies address translation because the operating system can easily separate the page number and offset using bit operations.

If the logical address space contains 2^m addresses and the page size is 2^n bytes, then:

- The higher-order $m-n$ bits represent the page number
- The lower-order n bits represent the page offset

This structure makes paging highly efficient for hardware implementation.

Although paging eliminates external fragmentation, a small amount of internal fragmentation may still occur. This happens when the last page of a process does not completely fill an entire frame. The unused space inside the frame becomes wasted memory.

For example, suppose the page size is 4 KB and a process requires 10 KB of memory.

Since each page can store only 4 KB, the process will be divided into:

- Page 1 → 4 KB
- Page 2 → 4 KB
- Page 3 → 2 KB

SGOU

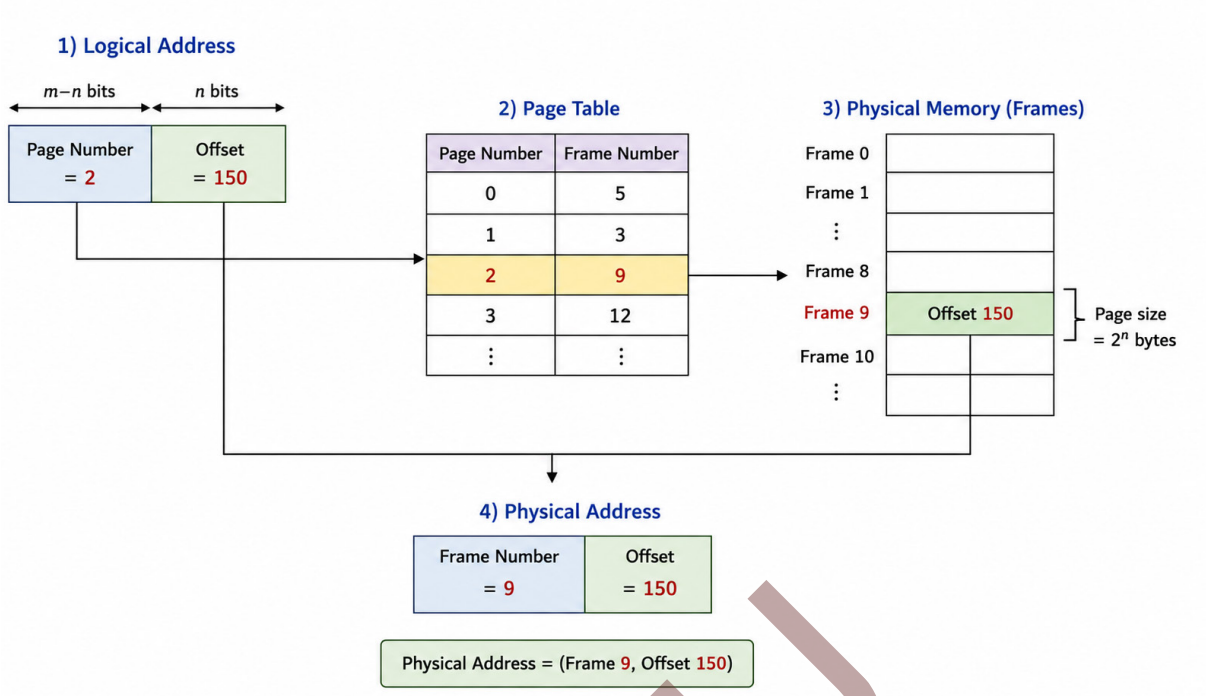


Fig.2.4.4. Paging

Thus, three frames are required to store the process. The first two frames are completely filled, but the third frame contains only 2 KB of data. The remaining 2 KB space inside that frame remains unused.

This unused space inside the allocated frame is called *internal fragmentation*.

Paging also introduces additional overhead because the operating system must maintain page tables for every process. Address translation requires extra processing, which may slightly increase memory access time. To overcome this issue, modern systems use hardware support such as the Translation Lookaside Buffer (TLB), which stores recently used page table entries and speeds up address translation.

2.4.2.2 Segmentation

Although paging is an effective memory management technique, it has certain limitations. One major issue is internal fragmentation, where a small portion of memory inside a frame may remain unused when the allocated page does not fully occupy it. Also, paging does not consider the logical structure of a program, since it divides memory into fixed-size blocks without distinguishing between code, data, or functions. To overcome these limitations, segmentation is used. Segmentation divides a program into logical, variable-sized units such as code, data, and stack, based on how the program is actually structured. This improves the programmer's view of memory and allows better protection and sharing of individual program components. Thus, segmentation provides a more meaningful and flexible approach to memory management compared to paging.

In operating systems, segmentation is a memory management technique in which

SGOU

memory is divided into variable-sized parts called **segments**. Each segment represents a logical part of a program, such as code, data, stack, or functions. Unlike paging, segmentation does not divide memory into fixed-size blocks. Instead, the size of each segment depends on the actual requirement of the program.

The information about all segments is stored in a segment table. This table mainly contains two important details for each segment:

- **Base** – the starting physical address of the segment in memory
- **Limit** – the size or length of the segment

When a process generates a logical address, it contains two parts:

- **s** – segment number
- **d** – offset within the segment

The operating system first checks whether the offset value d is within the limit of the segment. If the offset exceeds the segment size, an error or trap is generated because the process is trying to access memory outside the segment boundary. If the offset is valid, it is added to the base address of the segment to obtain the corresponding physical address in memory.

For example, consider a program divided into three segments:

Segment	Purpose	Base Address	Limit
0	Code Segment	1000	400
1	Data Segment	2000	300
2	Stack Segment	3000	200

Suppose the logical address generated is (1150). Here, 1 indicates the data segment and 150 is the offset. Since the offset 150 is less than the segment limit 300, it is valid. Therefore, the physical address is calculated as:

2000 + 150 = 2150

So, the required data will be accessed from physical address 2150.

However, if the logical address is (2250), the offset 250 exceeds the segment limit 200. In this case, the operating system generates an error because the process is attempting to access memory outside the stack segment.

Thus, segmentation organizes memory according to the logical structure of programs and provides better flexibility, protection, and sharing of memory.

SGOU

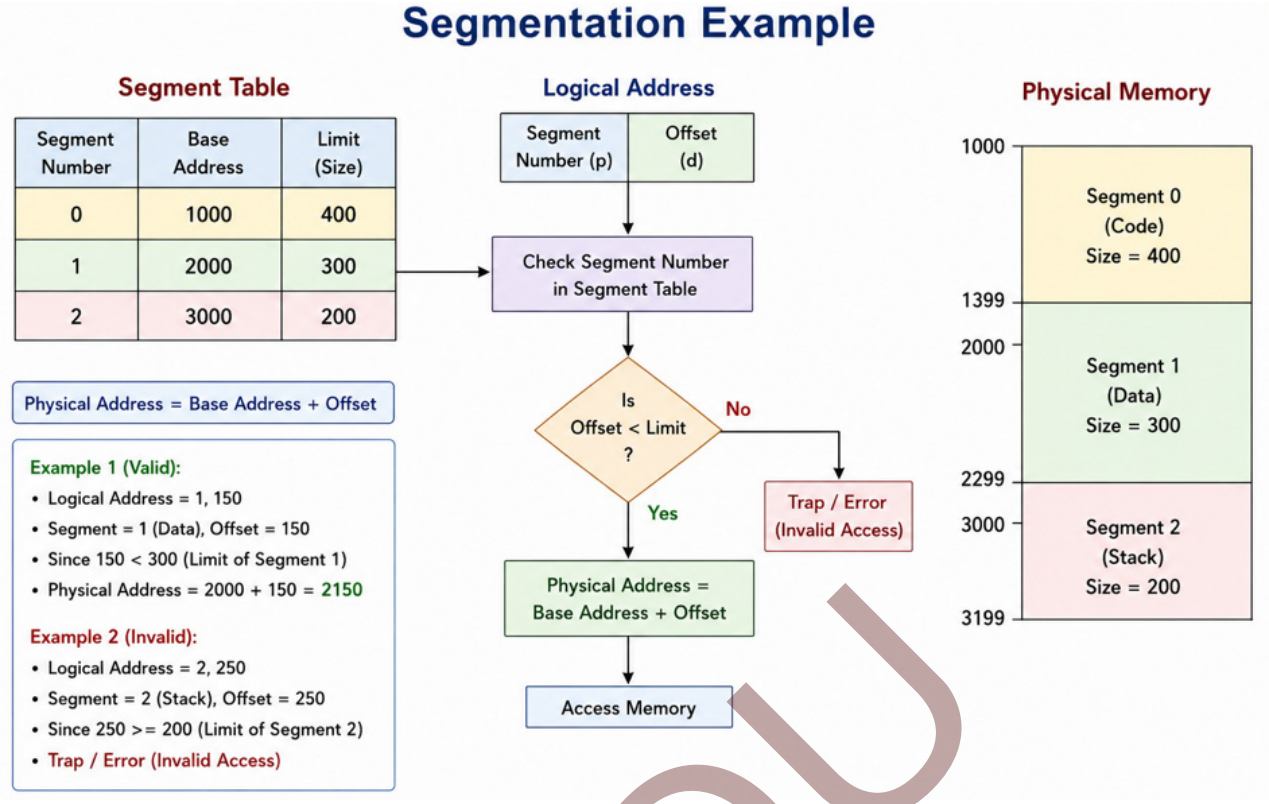


Fig 2.4.5 Segmentation

SGOU

Summarized Overview

Memory allocation is an important function of the operating system that manages how memory is assigned to processes. Memory allocation techniques are mainly classified into contiguous and non-contiguous allocation methods. In contiguous allocation, a process is stored in one continuous block of memory. Different strategies such as First-Fit, Best-Fit, and Worst-Fit are used to select suitable free memory spaces for processes. Although contiguous allocation is simple, it may lead to fragmentation problems.

To overcome these limitations, non-contiguous memory allocation techniques are used. In this method, a process can be stored in different locations in memory. Paging divides memory into fixed-size pages and frames, improving memory utilization and eliminating external fragmentation. Segmentation divides a program into logical variable-sized segments such as code, data, and stack, providing better logical organization and protection. Together, these techniques help operating systems manage memory more efficiently in modern computer systems.

Assignments

1. Explain contiguous memory allocation and discuss the working of First-Fit, Best-Fit, and Worst-Fit memory allocation strategies with suitable examples.

2. Describe the concept of paging in operating systems. Explain the role of pages, frames, and page tables in address translation.

3. What is segmentation? Explain the working of segmentation with the help of segment tables and logical address translation.

4. Consider the following free memory holes: 150KB, 400KB, 250KB, and 500KB. A process of size 220 KB arrives. Show how the process will be allocated using:

    - First-Fit
    - Best-Fit
    - Worst-Fit

SGOU

Also identify the amount of remaining free space in each case.

5. A system uses paging with a page size of 1 KB. A process of size 3500 bytes is to be loaded into memory.
   - Determine the number of pages required.
   - Calculate the amount of internal fragmentation produced.

Reference

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating system concepts (10th ed.). Wiley.
2. Stallings, W. (2018). Operating systems: Internals and design principles (9th ed.). Pearson.
3. Tanenbaum, A. S., & Bos, H. (2015). Modern operating systems (4th ed.). Pearson.
4. Dhamdhere, D. M. (2017). Operating systems: A concept-based approach (3rd ed.). McGraw Hill Education.

Suggested Reading

1. Godbole, A. S. (2017). *Operating systems* (3rd ed.). McGraw Hill Education.
2. Nutt, G. J. (2004). *Operating systems: A modern perspective* (3rd ed.). Pearson.
3. Deitel, H. M., Deitel, P. J., & Choffnes, D. R. (2003). *Operating systems* (3rd ed.). Pearson Education.
4. Harvey, M., & Paul, D. (2019). *Beginning operating systems design and implementation*. Apress.

SGOU

Space for Learner Engagement for Objective Questions

Learners are encouraged to develop objective questions based on the content in the paragraph as a sign of their comprehension of the content. The Learners may reflect on the recap bullets and relate their understanding with the narrative in order to frame objective questions from the given text. The University expects that 1 - 2 questions are developed for each paragraph. The space given below can be used for listing the questions.

SGOU

BLOCK 3
File Systems and I/O Management



SGOU

1 UNIT File System Interface

Learning Outcomes

After completion of this unit, the learner will be able to:

- recall the basic concepts of file system interface, including files, directories, and file operations
- identify different file attributes, file types, and basic file operations used in an operating system
- describe various file access methods such as sequential, direct, and indexed access
- explain different directory structures like single-level, tree-structured, and graph-based directories
- summarize the concepts of file system mounting and file protection mechanisms used to ensure data security

SGOU

Background

The file system interface is a crucial component of an operating system that provides a structured way to store, organize, retrieve, and manage data on storage devices. It defines how files are created, named, accessed, and protected, ensuring that users and applications can interact with data efficiently and securely. The file system acts as a bridge between the user and the physical storage hardware, making complex storage operations simple and user-friendly. Key concepts such as file concept, file attributes, and file operations form the foundation of this interface.

This unit also explores different methods used to access files, including sequential, direct, and indexed access methods, which determine how data is retrieved from storage. It further covers directory structures that help in organizing files in

a hierarchical or structured manner. Additionally, important topics like file system mounting explain how different file systems are integrated into a single directory tree, while file protection mechanisms ensure security through access control and permissions. Together, these concepts provide a complete understanding of how modern operating systems manage data effectively and securely.

Keywords

File Management, Access Methods, Directory Structure,File System Mounting, File Protection

Discussion

The File System Interface is a fundamental component of an operating system that defines how data is stored, organized, accessed, and protected on storage devices. It provides a structured approach for managing files and directories, enabling users and applications to perform operations such as creating, reading, writing, and deleting files efficiently. This unit also covers different access methods used to retrieve data, various directory structures that help organize files in a hierarchical manner, the process of mounting file systems to integrate multiple storage devices into a single directory tree, and protection mechanisms that ensure secure access and maintain data integrity within the system.

3.1.1 Overview of File System Interface

The File System Interface is a key component of an operating system that evolved from early data storage methods used before modern computing systems. In the past, information was stored using punch cards, magnetic tapes, and manual registers, where data was difficult to organize, retrieve, and update. As computers developed, the need for a structured system led to the creation of file systems that introduced files and directories to manage data efficiently in a logical and user-friendly manner.

Building on this foundation, the role of the file system in an operating system is to provide an organized and secure method for storing and accessing data, acting as a bridge between users and physical storage devices. It ensures proper allocation of disk space, maintains file structure, and supports multiple users and processes simultaneously. For example, when a user saves a file in a folder, the operating system uses the file system to decide where it is stored and how it can be retrieved, directly linking the concept of file organization to system-level data management.

Extending this functionality, file management functions are the operations that make

SGOU

the file system usable and efficient. These include creating, opening, reading, writing, closing, renaming, and deleting files, along with organizing them into directories and managing access permissions. For instance, when a file is created and saved, the file system assigns it storage space and records its attributes, which later allows the operating system to locate and modify it when needed. Thus, file management functions directly support the file system's role in ensuring structured and reliable data handling.

3.1.2 File Concept

The file concept is a fundamental idea in operating systems that explains how data is stored, organized, and managed in the form of files within a file system.

Definition of a File

A file is a named collection of related information or data stored on a storage device, such as a hard disk, SSD, or USB drive, and managed by the operating system. It acts as the basic unit of storage in a file system and allows data to be organized, saved, and retrieved efficiently. Each file is identified by a unique name and is associated with attributes such as size, type, location, and access permissions, which help the operating system manage it properly.

For example, a document created in a word processor like "report.docx" is a file that stores text, images, and formatting information in a structured form. Similarly, an image file like "photo.jpg" contains digital visual data, while a program file contains executable instructions. All these files are stored and managed by the file system, which ensures they can be accessed, modified, or deleted when required.

3.1.2.1 File Attributes

File attributes are the properties that describe and provide information about a file in a file system. They help the operating system identify, manage, secure, and organize files efficiently. Each attribute plays an important role in file handling and access control.

- **Name:** The name is the unique identifier of a file within a directory. For example, *report.docx* is used to distinguish one file from another.
- **Type:** The type indicates the kind of file, such as text file, image file, audio file, video file, or executable file. For example, *.jpg* represents an image file.
- **Size:** The size refers to the amount of storage space the file occupies on a storage device, usually measured in bytes, KB, MB, or GB.
- **Location:** The location specifies where the file is stored on the storage device, including its path within the directory structure (e.g., C:\Users\Documents).
- **Protection:** Protection defines the access rights of a file, such as read, write, execute, or delete permissions, ensuring security and controlled access.
- **Time Stamps:** Time stamps record important time-related information about a file, such as creation time, last modification time, and last access time, helping in tracking file usage and changes.

SGOU

Together, these attributes allow the operating system to efficiently manage files, maintain security, and provide quick access to stored data.

3.1.2.2 File Operations

File operations are the basic actions performed on files in a file system to manage and manipulate data efficiently. These operations are supported by the operating system and allow users and applications to interact with files in a structured manner.

- **Create:** This operation is used to generate a new file in the file system. When a file is created, the system allocates space and assigns attributes to it. For example, creating a file named *notes.txt*.
- **Open:** The open operation makes a file available for use by loading it into memory and establishing a connection between the file and the application. For example, opening a document in a text editor.
- **Read:** This operation is used to access and retrieve data stored in a file without modifying it. For example, reading the contents of a PDF file.
- **Write:** The write operation is used to add new data or modify existing data in a file. For example, saving new text in a document.
- **Reposition (Seek):** This operation moves the file pointer to a specific location within the file, allowing data to be read or written from a particular position instead of starting from the beginning.
- **Close:** The close operation terminates the connection between the file and the application, ensuring that all changes are saved and resources are released.
- **Delete:** This operation removes a file from the file system and frees the allocated storage space for reuse.

Together, these file operations enable efficient creation, access, modification, and management of data within an operating system.

3.1.2.3 File Types

File types classify files based on the kind of data they store and how that data is interpreted by the operating system and applications. This classification helps the system handle files appropriately during storage, processing, and execution.

- **Text Files:** Text files store data in human-readable form using characters such as letters, numbers, and symbols. They are easy to view and edit using simple text editors. For example, files like *.txt*, *.docx*, and source code files contain readable content that can be directly interpreted by users.
- **Binary Files:** Binary files store data in a non-readable format using binary (0s and 1s). These files are interpreted by specific applications rather than being directly readable by humans. For example, image files (*.jpg*), audio files (*.mp3*), and video files (*.mp4*) are binary files.

SGOU

- **Executable Files:** Executable files contain instructions that can be directly run by the operating system to perform specific tasks or programs. When executed, they initiate a sequence of operations defined in the program. For example, files like *.exe* in Windows or compiled program files in other systems are executable files.

Thus, different file types help the operating system manage data efficiently by determining how each file should be stored, accessed, and processed.

3.1.2.4 File Structure and Organization

File structure and organization refer to the way data is arranged, stored, and managed within a file in a file system. It defines how information inside a file is logically represented and how it can be accessed efficiently by the operating system and applications. A well-designed file structure improves data retrieval speed, storage efficiency, and ease of modification.

File structure can be organized in different ways such as byte stream structure, where a file is treated as a sequence of bytes without any internal structure, and record-based structure, where data is organized into fixed or variable-sized records. Some systems also use complex structured files, where data is stored in a hierarchical or indexed format for faster access. For example, a text file is usually a simple byte stream, while a database file may use structured records with fields and indexes. Proper file organization ensures that data is stored logically and can be accessed, updated, and managed efficiently by the operating system.

File System Architecture

The file system is organized into different layers, each performing specific functions:

- **User Application:** These are programs or software that request file operations such as read, write, or delete.
- **Logical File System:** It manages file names, directories, metadata, and access permissions.
- **Virtual File System (VFS):** Acts as an interface that allows different file systems to work under a single unified structure.
- **Physical File System:** Handles the actual storage of data blocks on the disk.
- **Disk Partitions (Partition 1, Partition 2, Partition 3):** These are divisions of storage devices where files are physically stored.

Popular File Systems

Different operating systems use different file systems based on their features:

- **FAT (File Allocation Table):** An older file system used in earlier Windows versions and portable devices.

SGOU

- **NTFS (New Technology File System):** Used in modern Windows systems; supports security, compression, and encryption.
- **ext (Extended File System):** Common in Linux and Unix-based systems.
- **HFS (Hierarchical File System):** Used in older macOS systems.
- **APFS (Apple File System):** A modern file system used in macOS and iOS devices with improved performance and security.

3.1.3 Access Methods

File access methods refer to the techniques used by an operating system to store, retrieve, and modify data within files in an efficient manner. They define the way information is organized and accessed from storage devices, which directly impacts system performance and data handling. The selection of an appropriate access method depends on the type of application and how frequently data needs to be read or updated. In operating systems, the commonly used file access methods include sequential access, direct access, and indexed sequential access, each providing different approaches to handling file data.

1. Sequential Access Method

Sequential access is a file access technique in which data is processed in a strict order, one record after another, starting from the beginning of the file. In this method, the file pointer automatically moves forward after each read or write operation, ensuring that data is accessed in a continuous sequence. It is commonly used in systems where data is stored on tape or where ordered processing is required.

In sequential access, each record is accessed one after the other in the same order in which it is stored. When a **read operation** is performed, the pointer moves to the next record automatically. Similarly, during a **write operation**, data is added and the pointer is moved to the end of the file. This method is well-suited for sequential storage devices like magnetic tapes, where random access is not possible.

Advantages of Sequential Access Method

- **Simple to implement:** Easy to design and use compared to other access methods.
- **Ordered processing:** Data is accessed in the exact sequence in which it is stored.
- **Reliable data handling:** Since data is written sequentially, the risk of corruption is reduced.
- **Efficient for batch processing:** Suitable for applications like payroll and log processing.
- **Works well with tape storage:** Ideal for sequential storage devices such as magnetic tapes.

SGOU

Disadvantages of Sequential Access Method

- **Slow search process:** Specific records take more time to locate as the file must be scanned from the beginning.
- **Poor random access support:** Cannot directly access a particular record without traversing earlier data.
- **Difficult updates:** Inserting or modifying records in the middle is complex and time-consuming.
- **Storage inefficiency:** Variable-length records may lead to wasted or fragmented storage space.
- **Limited flexibility:** Not suitable for applications requiring frequent data updates or fast retrieval.

2. Direct (Random) Access Method

Direct access method is a file access technique in which data can be accessed directly from any location in the file without following a sequential order. Each record in the file is assigned a unique address or index, allowing the system to quickly locate and retrieve specific data. The file pointer can move to any position in the file using this address, making access faster and more flexible. This method is commonly used in systems where quick retrieval and frequent updates are required, such as databases.

Advantages of Direct Access Method

- **Faster data access:** Files can be accessed immediately, reducing the average time required for retrieval.
- **No sequential traversal required:** There is no need to go through earlier records to access a specific block.
- **Efficient for large files:** Suitable for databases and systems handling large volumes of data.
- **Quick updates:** Data can be inserted, modified, or deleted without affecting other records significantly.
- **Improved performance:** Enhances overall system efficiency by supporting random retrieval of data.

Disadvantages of Direct Access Method

- **Complex implementation:** Requires advanced algorithms and data structures to manage record locations efficiently.
- **Higher storage overhead:** Additional memory is needed to store pointers, indexes, or address mapping information.
- **Risk of data inconsistency:** Improper management of addresses may lead to errors in data retrieval.

SGOU

- **More system overhead:** Managing random access adds extra processing burden on the system.
- **Not suitable for simple sequential tasks:** Less efficient for applications that process data in a fixed order.

3. Indexed Access Method

Indexed access method is a file access technique built on top of the sequential access method. In this approach, an index is created for the file, similar to an index in a book, which stores pointers to the actual data blocks. To access a specific record, the system first searches the index to find the address of the required block and then directly retrieves the data using that pointer. This combination of indexing and sequential storage allows both fast and organized access to file data.

Advantages of Indexed Access Method

- **Fast searching:** Indexing allows quick location of records without scanning the entire file.
- **Supports multiple access types:** Allows both sequential and direct (random) access to data.
- **Reduces access time:** Efficiently retrieves data, especially in large files.
- **Improves performance:** Enhances data retrieval speed compared to pure sequential access.
- **Better organization:** Helps manage large datasets in a structured manner.

Disadvantages of Indexed Access Method

- **Complex implementation:** More difficult to design and maintain compared to sequential access.
- **Extra storage requirement:** Additional memory is needed to store index tables.
- **Slower updates:** Insertions, deletions, and modifications require updating both data and index.
- **Maintenance overhead:** Indexes need frequent updates in dynamic file environments.
- **Increased system load:** Managing indexes adds extra processing overhead to the system.

File access methods play an important role in determining how efficiently data can be stored, retrieved, and managed within a file system. Different methods such as sequential, direct (random), and indexed access are designed to suit different types of applications and data handling requirements. Sequential access is simple and suitable for ordered processing, direct access provides fast retrieval for large datasets, and

SGOU

indexed access combines the advantages of both by enabling quick and flexible data access. Overall, the choice of file access method directly impacts system performance, storage efficiency, and data management effectiveness in an operating system.

3.1.4 Directory Structure

A directory structure is a systematic way of organizing files within a file system to make data storage and retrieval more efficient and manageable. It defines how files are grouped, named, and arranged in a hierarchical or logical manner. By using directories (or folders), the operating system provides a clear structure that helps users locate files easily and maintain order in storage devices. Different directory structures such as single-level, two-level, tree-structured, and graph-based directories are used depending on the system requirements.

The main purpose of directory systems is to provide an organized method for storing and accessing files efficiently. It helps in managing large volumes of data by grouping related files together and supporting easy navigation through the file system. Directory systems also ensure proper file naming, avoid duplication of file names, and improve security by controlling access to files. Additionally, they enhance search efficiency, reduce confusion, and support better file management in multi-user environments.

A. Single-Level Directory Structure

A single-level directory structure is the simplest form of file organization in which all files are stored in a single directory without any subdirectories. In this structure, every file is located in the same directory and is identified by a unique name. The operating system maintains only one directory for all users, and there is no hierarchical arrangement of files. This type of structure was commonly used in early operating systems due to its simplicity.

In this structure, file management is easy because all files are in one place, making creation, deletion, and searching straightforward for small systems. However, as the number of files increases, it becomes difficult to manage them efficiently. It also leads to issues such as file name duplication, lack of organization, and poor scalability in multi-user environments. Therefore, while simple and easy to implement, it is not suitable for modern complex systems.

Working

In a single-level directory system, when a user creates a file, it is directly stored in the only available directory. To access a file, the system searches through the single directory and matches the file name. Since all files are in one place, file operations like create, delete, open, and search are performed directly within this single directory without navigating through subdirectories.

Advantages

- **Simple structure:** Easy to understand and implement.
- **Easy file access:** All files are located in one place, making searching straightforward.

SGOU

- **Quick operations:** File creation, deletion, and access are fast for small systems.
- **No hierarchy complexity:** No need to manage multiple directory levels.

Disadvantages

- **File name conflicts:** Different users cannot use the same file name.
- **Poor organization:** All files are stored together, making management difficult.
- **Not scalable:** Becomes inefficient as the number of files increases.
- **No user separation:** Difficult to support multiple users in a secure way.
- **Slow searching in large systems:** Finding files becomes time-consuming when many files exist.

B. Two-Level Directory Structure

A two-level directory structure is a file organization method in which the file system is divided into two levels: a master file directory (MFD) and user file directories (UFDs). Each user is given a separate directory under the MFD, allowing them to store and manage their own files independently. This structure was developed to overcome the limitations of the single-level directory system, especially in multi-user environments.

Working

In this structure, the top-level directory (MFD) contains entries for each user, and each entry points to a separate user directory (UFD). When a user creates a file, it is stored in their own UFD. To access a file, the system first identifies the user's directory through the MFD and then searches within that specific UFD. This ensures that file names can be repeated across different users without conflict.

Advantages

- **User isolation:** Each user has a separate directory, improving privacy and organization.
- **No file name conflict:** Different users can use the same file name without issues.
- **Better organization:** Files are grouped under individual user directories.
- **Improved security:** Access is restricted to a user's own directory.
- **Efficient management:** Easier to handle files in multi-user systems compared to single-level structure.

Disadvantages

- **Limited sharing:** Difficult to share files between users.

SGOU

- **No subdirectory support:** Cannot create nested folders within a user directory.
- **Scalability issues:** Becomes less efficient in complex systems requiring deeper organization.
- **Rigid structure:** Less flexible compared to tree-structured directories.

C. Tree-Structured Directory

A tree-structured directory is a hierarchical file organization method in which files are arranged in a tree-like structure with a single root directory at the top and multiple levels of subdirectories below it. This structure allows users to organize files in a logical and systematic way, grouping related files together in different folders and subfolders. It is widely used in modern operating systems because of its flexibility and efficiency in managing large amounts of data.

Working

In a tree-structured directory, the root directory contains subdirectories, and each subdirectory can further contain files or additional subdirectories. Users navigate through the directory path using absolute or relative path names to access files. The system starts from the root directory and moves through different levels until it reaches the required file. This hierarchical approach makes file searching and organization more structured and efficient.

Advantages

- **Better organization:** Files can be grouped logically into directories and subdirectories.
- **Efficient searching:** Easier to locate files using hierarchical paths.
- **Scalability:** Suitable for large systems with many files and users.
- **No naming conflicts:** Same file names can exist in different directories.
- **Improved access control:** Permissions can be assigned at different directory levels.

Disadvantages

- **Complex structure:** More difficult to design and manage compared to simple directory systems.
- **Navigation overhead:** Users may need to traverse multiple levels to access files.
- **Path dependency:** File access depends on correct path specification.
- **Increased system overhead:** Managing hierarchical levels requires more processing.

SGOU

D. Acyclic Graph Directory

An acyclic graph directory is an advanced file organization structure in which directories are arranged in a graph format, allowing files and subdirectories to be shared among multiple parent directories. Unlike a tree structure, it permits links (shared references) to the same file or directory, but it does not allow cycles, meaning a directory cannot form a loop back to itself. This structure is useful in systems where file sharing is required without duplicating data.

Working

In an acyclic graph directory, a file or subdirectory can have multiple parent directories through links. These links act as references pointing to the same physical file location. When a file is accessed, the system follows the link to reach the actual data. The file system ensures that no circular paths are created, maintaining an acyclic (loop-free) structure. This allows efficient sharing of files while maintaining a structured organization.

Advantages

- **Efficient file sharing:** Multiple users or directories can access the same file without duplication.
- **Space saving:** Reduces storage usage by avoiding repeated copies of files.
- **Flexible structure:** Supports more complex relationships between files and directories.
- **Better organization:** Allows logical grouping and sharing of resources.
- **Improved collaboration:** Useful in multi-user systems where file sharing is required.

Disadvantages

- **Complex management:** Harder to implement and maintain compared to tree structures.
- **Risk of inconsistency:** Changes in shared files may affect multiple directories.
- **Difficult deletion handling:** Removing a file requires checking all references.
- **Increased overhead:** File system must manage links and references carefully.
- **No cycles allowed:** Restriction adds complexity in maintaining structure integrity.

E. General Graph Directory Structure

A general graph directory structure is a file organization model in which directories

SGOU

are arranged in a graph format that allows multiple links between files and directories, including the possibility of cycles. Unlike tree and acyclic graph structures, this model provides maximum flexibility by allowing shared files and subdirectories without strict hierarchical restrictions. However, because cycles are possible, special mechanisms are required to manage traversal and avoid infinite loops.

Working

In a general graph directory structure, files and directories can have multiple parent directories, and links can be created freely between them, even forming circular paths. The file system uses techniques such as reference counting or garbage collection to manage shared files and ensure proper deletion. When accessing files, the system must carefully track visited nodes to prevent infinite looping due to cycles in the structure.

Advantages

- **Maximum flexibility:** Allows any file or directory to be linked anywhere in the system.
- **Efficient file sharing:** Multiple users can access the same file without duplication.
- **Space optimization:** Reduces storage usage by sharing files instead of copying them.
- **Dynamic structure:** Supports complex relationships between files and directories.
- **Improved collaboration:** Useful in environments requiring extensive resource sharing.

Disadvantages

- **Very complex implementation:** Difficult to design and manage compared to other structures.
- **Cycle problem:** Possibility of loops makes navigation and processing complicated.
- **Risk of infinite traversal:** Special checks are needed to avoid endless loops.
- **Difficult deletion:** Removing files requires handling multiple references carefully.
- **High system overhead:** Requires additional processing for tracking and managing links.

3.1.4.1 Directory Operations

Directory operations are the basic functions provided by an operating system to manage files and directories efficiently. These operations help users organize, access, and maintain the file system in a structured manner. Each operation plays an important role in handling directories and their contents.

SGOU

- **Search:** This operation is used to locate a specific file or directory within the file system. The system scans the directory structure to find the required item based on its name or path.
- **Create:** This operation is used to make a new directory in the file system. It helps in organizing files into separate folders for better structure and management.
- **Delete:** This operation removes an existing directory from the file system. When a directory is deleted, all files and subdirectories inside it are also removed.
- **List:** This operation displays all files and subdirectories present inside a directory. It helps users view and manage the contents of a folder easily.
- **Rename:** This operation is used to change the name of an existing directory without affecting its contents or structure.

These directory operations ensure efficient file organization, easy navigation, and better management of data within the operating system.

3.1.5 File System Mounting

File system mounting is a fundamental concept in operating systems that enables the integration of storage devices into the system's directory structure. It is the process through which the operating system makes the data stored on a device such as a hard disk, USB drive, or network storage accessible to users by attaching its file system to a specific directory known as a mount point. Once mounted, the files and folders on the device can be accessed and managed just like any other part of the system.

Mounting provides a unified and organized way to handle multiple storage devices within a single system. It ensures that different file systems, whether internal or external, are seamlessly connected, allowing efficient data access, sharing, and management. This process plays a vital role in maintaining system flexibility, usability, and effective storage management.

Meaning of Mounting

Mounting is the process by which an operating system makes the files and directories of a storage device accessible to users by attaching its file system to the main directory structure. In simple terms, it allows the OS to read and use data stored on devices such as hard disks, USB drives, or network storage.

The file system of the storage device is attached to an empty directory so that users can access its contents through the system file manager. Storage devices can include internal hard disks, external drives, USB flash drives, SSDs, memory cards, CDs/DVDs, and even remote or network-based storage systems.

Mount Point Concept

A mount point is an empty directory in the existing file system where a new file system

SGOU

is attached. Once mounted, the contents of the storage device appear as part of that directory.

For example, if a USB drive is mounted at */mnt/mydisk*, all its files can be accessed through that folder.

Terminologies used in File System Mounting

- **File System**: A method used by the operating system to organize and manage data on a storage device, enabling efficient storage and retrieval of files and directories.
- **Device Name**: A unique identifier assigned to a storage device or partition. For example, in Windows, drives are labeled as *C:, D:*, etc., while in Linux, devices may appear as */dev/sdb1*.
- **Mount Point**: The directory where the file system is attached during mounting.

3.1.5.1 Mounting Process in Operating System

The mounting process in an operating system is the procedure by which a storage device's file system is attached to the system's directory structure, making its data accessible to users. This process ensures that the operating system can recognize the device, interpret its file system, and allow interaction with its files and directories. Steps in the mounting process are:

1. **Device Detection:** The operating system detects the presence of a storage device such as a hard disk, USB drive, or network storage when it is connected to the system.
2. **File System Identification:** The OS identifies the type of file system on the device (e.g., FAT32, NTFS, ext4) so that it can properly read and manage the stored data.
3. **Selection of Mount Point:** A mount point (an empty directory) is selected or created in the existing file system where the new file system will be attached.
4. **Attachment of File System:** The OS links the device's file system to the chosen mount point. After this step, the device becomes part of the system's directory hierarchy.
5. **Access and Usage:** Users can now access, read, write, and manage files on the mounted device through the mount point or file manager.
6. **Permission and Security Handling:** The operating system applies appropriate permissions and access controls to ensure secure usage of the mounted file system.

Example

In a Linux system,

SGOU

- Device: */dev/sdb1*
- Mount Point: */mnt/mydisk*

Command used: ***sudo mount /dev/sdb1 /mnt/mydisk***

After execution, all files in */dev/sdb1* become accessible through */mnt/mydisk.*

The mounting process is essential for enabling communication between the operating system and storage devices. It ensures that data from different devices is seamlessly integrated into a single, organized file system structure for easy access and management.

3.1.5.2 Types of File System Mounting

File system mounting can be classified into different types based on how and when the mounting process is performed. These types help the operating system manage storage devices efficiently and provide flexibility to users.

1. Automatic Mounting (Auto Mounting)

Automatic mounting occurs when the operating system mounts a storage device without user intervention.

- Triggered when a device is connected or the system starts.
- Common in user-friendly systems like Windows and macOS.
- The OS detects the file system and assigns a location (e.g., drive letter or directory).
- Example: A USB drive automatically appearing as *E:* in Windows.

Advantages

- Easy and convenient for users
- No need for manual commands
- Quick access to devices

Disadvantages

- Less control over mount location and options
- May mount unwanted or insecure devices automatically

2. Manual Mounting

Manual mounting is performed by the user or system administrator using specific commands.

- Common in Linux/Unix systems.
- Requires specifying the device name and mount point.

SGOU

- Gives full control over mounting options.

Example:

sudo mount /dev/sdb1 /mnt/mydisk

Advantages

- Greater control and flexibility
- Can define permissions and mount options
- More secure for advanced users

Disadvantages

- Requires technical knowledge
- Time-consuming compared to automatic mounting

3. Network Mounting

Network mounting allows file systems located on remote systems to be accessed over a network.

- Uses protocols like NFS (Network File System) or SMB (Server Message Block).
- Makes remote storage appear as part of the local file system.

Advantages

- Enables file sharing across systems
- Centralized data storage
- Useful in distributed environments

Disadvantages

- Depends on network availability
- May have security and performance issues

4. Temporary Mounting

Temporary mounting is done for short-term use and is removed after system shutdown or unmounting.

- Common for removable devices like USB drives or CDs.
- Data is accessible only during the session.

5. Permanent Mounting

Permanent mounting is configured to occur automatically every time the system boots.

SGOU

- Defined in system configuration files (e.g., */etc/fstab* in Linux).
- Used for frequently accessed storage devices.

Advantages

- Saves time by avoiding repeated mounting
- Ensures consistent availability

Disadvantages

- Incorrect configuration may cause system boot issues

Different types of file system mounting provide flexibility in managing storage devices. While automatic mounting offers ease of use, manual and network mounting provide greater control and functionality, especially in advanced and multi-user environments.

3.1.5.3 Mounting in Different Operating Systems

Mounting in different operating systems refers to the way various systems such as Linux, Windows, and macOS handle the process of attaching and accessing file systems from storage devices, either automatically or through user-controlled methods.

1. Linux/Unix-Based Systems

Mounting is usually done manually using commands.

Example:

sudo mount /dev/sdb1 /mnt/mydisk

Unmounting: *sudo umount /mnt/mydisk*

Here,

- */dev/sdb1* is the device name
- */mnt/mydisk* is the mount point

2. Windows Operating System

Mounting is automatic and user-friendly.

Steps:

1. Connect the external storage device.
2. Windows detects the file system (e.g., FAT32, NTFS).
3. It assigns a drive letter such as D: or E:.
4. Access via This PC → File Explorer → Drive Letter.

SGOU

3. Mac Operating System

Mounting is also automatic but can be done manually.

Method 1 (Automatic):

- Connect the device.
- It appears in Finder sidebar.
- Access the files directly.

Method 2 (Using Terminal): *diskutil mount /dev/disk2s1*

Unmounting: *diskutil unmount /dev/disk2s1*

3.1.5.4 Unmounting of File Systems

Unmounting is the process of safely detaching a mounted file system from the operating system. It ensures that all pending read and write operations are completed before the storage device is removed or disconnected. This prevents data corruption, data loss, and system errors.

When a file system is unmounted, it is no longer accessible through the directory structure, and the operating system removes its association with the mount point.

Need for Unmounting

Unmounting is essential for the following reasons:

- Ensures all data is properly written to the storage device
- Prevents file corruption and loss of information
- Maintains system stability and consistency
- Safely removes external storage devices

Unmounting Process

The unmounting process is the procedure of safely detaching a file system from the operating system so that the storage device can be removed without causing data loss or corruption. It ensures that all ongoing operations are completed and the device is properly released. Steps in the unmounting process as follows:

1. **Close All Open Files:** Any files or applications using the device must be closed.
2. **Flush Pending Data:** The operating system writes any cached data to the storage device.
3. **Detach File System:** The OS removes the link between the file system and the mount point.
4. **Release Device:** The device is marked safe for removal.

SGOU

3.1.5.5 Unmounting in Different Operating Systems

Unmounting in different operating systems refers to the methods used by systems such as Linux, Windows, and macOS to safely detach and remove file systems from storage devices, ensuring data integrity and preventing corruption.

1. Linux/Unix Systems

Unmounting is done using a command: *sudo umount /mnt/mydisk*

where,

- */mnt/mydisk* is the mount point
- The device must not be in use

2. Windows Operating System

Unmounting is performed using the **"Safely Remove Hardware"** option.

Steps:

1. Click on the "Safely Remove Hardware" icon in the system tray
2. Select the device
3. Wait for the confirmation message
4. Remove the device safely

3. Mac Operating System

Unmounting can be done using Finder or Terminal.

Using Finder: Click the eject icon next to the device

Using Terminal: d*iskutil unmount /dev/disk2s1*

Consequences of improper Unmounting

Improper unmounting occurs when a storage device is removed or disconnected without safely detaching it from the operating system. This can lead to several serious issues affecting both the data and the system. Major consequences are:

- **Data Loss:** Files that are being written or transferred may be lost if the device is removed suddenly.
- **File System Corruption:** The structure of the file system may become damaged, making files inaccessible or causing errors when accessing the device.
- **Incomplete File Transfers:** Ongoing operations may be interrupted, resulting in partially saved or unusable files.
- **System Errors:** The operating system may encounter errors or crashes due to sudden disconnection of active storage devices.

SGOU

Unmounting is a critical operation that ensures the safe removal of storage devices and maintains data integrity. Proper unmounting practices help prevent errors and protect both the system and stored information.

3.1.5.6 Importance of Mounting in System Integration

Mounting plays a vital role in system integration by enabling the operating system to combine multiple storage devices and file systems into a single, unified structure. It allows different types of storage such as internal disks, external drives, and network storage to function together seamlessly within the system. Key importance are:

- **Unified File System Structure:** Mounting integrates various storage devices into one hierarchical directory system, making data access simple and organized.
- **Easy Access to Data:** Users can access files from different devices through a common interface without needing to manage each device separately.
- **Support for Multiple Storage Devices:** It allows the system to handle multiple devices simultaneously, including USB drives, hard disks, and network storage.
- **Facilitates Data Sharing:** Mounting enables sharing of files across different systems, especially in network environments through remote file systems.
- **Improves System Flexibility:** Users can dynamically add or remove storage devices without affecting the overall system structure.
- **Enhances Resource Management:** The operating system can efficiently manage storage resources, permissions, and access control.
- **Supports Heterogeneous Environments:** Different file systems (e.g., FAT, NTFS, ext) can coexist and be accessed within the same system.

Mounting is an essential function of operating systems that allows users to access and manage data from various storage devices. By attaching file systems to specific mount points, the OS provides a unified and organized way to handle data, ensuring efficiency, flexibility, and system integration.

3.1.6 File Protection

File protection refers to the techniques and mechanisms used by an operating system to safeguard files from unauthorized access, modification, deletion, or damage. It ensures that only authorized users or processes can perform specific operations on files, such as reading, writing, or executing them.

Operating systems implement file protection using methods like access permissions, user authentication, encryption, and access control policies. For example, permissions may define who can read, write, or execute a file, thereby maintaining security and privacy of data.

SGOU

- **Need for File Protection**: File protection is essential for maintaining the security, integrity, and reliability of data in a computer system. The key reasons for file protection include:
- **Prevent Unauthorized Access**: Ensures that only authorized users can access sensitive or confidential data.
- **Maintain Data Integrity**: Protects files from accidental or intentional modification or deletion.
- **Ensure Privacy and Confidentiality**: Safeguards personal, organizational, or sensitive information from misuse.
- **Support Multi-user Environments**: In systems with multiple users, file protection ensures that users do not interfere with each other's data.
- **Prevent Malicious Activities**: Protects files from viruses, malware, and unauthorized tampering.

File protection is a fundamental aspect of operating systems that ensures data security and controlled access. It plays a crucial role in maintaining confidentiality, integrity, and proper functioning of both user and system files.

3.1.6.1 Types of File Access (Read, Write, Execute, Append, Delete)

File access types define the different kinds of operations that users can perform on files in a system. These access controls are an important part of file protection, as they restrict how files are used and help maintain data security and integrity.

The files that are directly accessible to multiple users require protection to prevent misuse. On the other hand, files that are not shared or accessed by others may not need strict protection. Protection mechanisms provide controlled access by limiting the types of operations that can be performed on a file. Whether access is granted or denied depends on factors such as the user's role and the type of access required.

Common Types of File Access

- **Read**: Allows a user to open and read the contents of a file without making any changes.
- **Write:** Permits creating, modifying, or overwriting the contents of a file.
- **Execute**: Allows a file (usually a program or script) to be loaded into memory and executed.
- **Append**: Enables adding new data to the end of an existing file without altering the existing content.
- **Delete**: Allows removal of a file from the system, freeing up storage space for other data.

SGOU

- **List**: Permits viewing the file name and its attributes (such as size, type, and permissions) without accessing its contents.

Other Controlled Operations

In addition to the above, some systems also control:

- Renaming files
- Editing existing content
- Copying files

Different file access types provide a structured way to control how users interact with files. By limiting operations such as reading, writing, or deleting, protection mechanisms ensure data security while allowing appropriate access based on user needs. Each protection mechanism has its own advantages and limitations, and the choice depends on the specific requirements of the system.

3.1.6.2 Access Control Methods

Access control methods are techniques used by operating systems to regulate how users access files and directories. These methods ensure that only authorized users can perform specific operations on files, thereby maintaining data security and integrity.

There are different methods used by systems to control file access. A common approach is to associate identity-dependent permissions with each file and directory using a structure called an **Access Control List (ACL)**. An ACL specifies the names of users and the types of access (such as read, write, or execute) allowed for each user.

1. Access Control List (ACL)

An **Access Control List (ACL)** is a list attached to a file that defines:

- which users can access the file
- what type of access each user has

Example:

User A → Read, Write

User B → Read only

Problems with ACL

- **Length of the List**: If access must be given to many users, the list becomes very long.
- **Difficult to Manage**: Creating and maintaining such lists can be tedious, especially when users are unknown in advance.
- **Space Management Issues**: Earlier, directory entries were of fixed size, but ACLs require variable sizes, making storage management more complex.

SGOU

2. Classification-Based Access Control (Owner, Group, Universe)

To overcome the limitations of long ACLs, many systems use a simplified and condensed approach by classifying users into three categories:

Owner: The user who creates the file and has full control over it.

Group: A set of users who share similar access needs and permissions.

Universe (Others): All other users in the system who are not part of the owner or group.

This method reduces complexity by assigning permissions to categories instead of individual users.

3. Combined Approach (ACL + Classification)

Modern operating systems often combine both methods:

- Use **Owner, Group, and Universe** as the default access control
- Add **ACLs** for more detailed and fine-grained permissions when needed

For example, systems like **Solaris** follow this hybrid approach, allowing flexibility and better control.

4. Password-Based Protection

Another method of access control is the use of passwords to restrict file access.

- Access is granted only if the correct password is provided
- Frequently changing and strong passwords improve security

Disadvantages of Password Protection

- Difficult to remember multiple passwords
- If one password is used for all files, security becomes weak (all-or-none access)
- Once a password is compromised, all protected files may become accessible

Access control methods are essential for protecting files in a multi-user environment. While ACLs provide detailed control, classification methods simplify management, and password protection adds an extra layer of security. A combination of these approaches is often used to achieve efficient and flexible file protection.

3.1.6.3 Security Mechanisms in File System

Security mechanisms in a file system are the methods and techniques used by an operating system to protect files and directories from unauthorized access, misuse, and damage. These mechanisms ensure that data remains secure, reliable, and accessible only to authorized users.

SGOU

1. Authentication

Authentication verifies the identity of a user before granting access to the system.

- Users must provide credentials such as usernames and passwords
- Advanced methods may include biometric verification or multi-factor authentication
- Ensures that only legitimate users can access files

2. Access Control

Access control regulates what actions a user can perform on a file.

- Implemented using permissions like read, write, execute
- Uses methods such as Access Control Lists (ACLs) and owner-group-others model
- Restricts unauthorized operations

3. Encryption

Encryption converts data into an unreadable format to protect it from unauthorized access.

- Data is encoded using encryption algorithms
- Only authorized users with a decryption key can read the file
- Provides strong protection for sensitive data

4. File Permissions

File permissions define the level of access granted to users.

- Includes permissions such as read, write, and execute
- Assigned to different user categories (owner, group, others)
- Helps in controlling file usage

5. Backup and Recovery

Backup and recovery mechanisms protect against data loss.

- Regular backups ensure data can be restored in case of failure
- Recovery tools help restore corrupted or deleted files
- Essential for system reliability

SGOU

6. Auditing and Logging

Auditing keeps track of file access and operations.

- Records who accessed a file and what actions were performed
- Helps in detecting unauthorized activities
- Useful for security analysis and troubleshooting

7. File Integrity Checking

Ensures that files are not altered without authorization.

- Uses techniques like checksums or hash values
- Detects any unauthorized modification of data

8. Protection Against Malware

File systems include mechanisms to detect and prevent malicious software.

- Antivirus and anti-malware tools scan files
- Prevents damage or unauthorized access caused by harmful programs

Security mechanisms in a file system are essential for protecting data from threats and unauthorized access. By combining authentication, access control, encryption, and other techniques, operating systems ensure data confidentiality, integrity, and availability.

SGOU

Summarized Overview

The File System Interface is an essential part of an operating system that provides a structured way for users and programs to interact with files and directories. A file is a collection of related data stored on a storage device, and it is managed by the operating system using attributes such as name, type, size, and permissions. To access file contents, different file access methods are used, including sequential access, direct access, and indexed access, each suitable for different types of applications. Files are organized using directory structures such as single-level, two-level, tree-structured, and graph-based directories, which help in efficient storage, retrieval, and management of data. Another important aspect of the file system interface is file system mounting, which allows storage devices to be integrated into the system by attaching their file systems to a specific directory called a mount point. This enables users to access data from various devices in a unified manner, while unmounting ensures safe removal without data loss. Additionally, file protection mechanisms play a key role in securing data by controlling access through permissions, access control methods like ACL and owner-group-others model, and security techniques such as authentication and encryption. Together, these components ensure that the file system interface provides efficient, organized, and secure management of data within an operating system.

SGOU

Assignments

1. Explain the file concept in an operating system. Describe file attributes and operations performed on files.

2. Discuss different file access methods such as sequential, direct, and indexed access with suitable examples.

3. Describe various directory structures used in operating systems. Compare their advantages and disadvantages.

4. Explain the concept of file system mounting and unmounting. How does mounting differ across operating systems?

5. Discuss file protection mechanisms and access control methods. Explain how they ensure data security in a multi-user environment.

Reference

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts* (10th ed.). Wiley.

2. Tanenbaum, A. S., & Bos, H. (2015). *Modern operating systems* (4th ed.). Pearson.

3. Arpaci-Dusseau, R. H., & Arpaci-Dusseau, A. C. (2018). *Operating systems: Three easy pieces*. Arpaci-Dusseau Books. https://pages.cs.wisc.edu/~remzi/OSTEP/

4. Tanenbaum, A. S., & Woodhull, A. S. (2006). *Operating systems: Design and implementation* (3rd ed.). Pearson.

5. McHoes, A. M., & Flynn, I. M. (2017). *Understanding operating systems* (8th ed.). Cengage Learning.

Suggested Reading

1. **File Systems in Operating System – GeeksforGeeks** https://www.geeksforgeeks.org/operating-systems/file-systems-in-operating-system/

2. **File-System Interface Notes – University of Illinois (UIC)** https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/11_FileSystemInterface.html

3. **Operating System – File System Interface (PDF)** https://www.bbau.ac.in/dept/dit/TM/File%20System.pdf

4. **Operating Systems Lecture Notes – File System Interface (PDF)** https://www.uomustansiriyah.edu.iq/media/lectures/5/5_2020_02_09%2111_00_52_PM.pdf

5. **Operating System Unit Notes – File System Mounting & Protection (PDF)** https://www.pvpsiddhartha.ac.in/dep_cse/lect_note/22/OS/23CS3401-Unit-5.pdf

SGOU

Space for Learner Engagement for Objective Questions

Learners are encouraged to develop objective questions based on the content in the paragraph as a sign of their comprehension of the content. The Learners may reflect on the recap bullets and relate their understanding with the narrative in order to frame objective questions from the given text. The University expects that 1 - 2 questions are developed for each paragraph. The space given below can be used for listing the questions.

SGOU

2 UNIT File System Implementation

Learning Outcomes

After completion of this unit, the learner will be able to:

- explain the concept, purpose, and structure of a file system in an operating system
- describe the layered architecture and components of a file system with suitable examples
- apply different directory implementation methods to identify suitable approaches for file organization
- demonstrate the use of file allocation methods in managing disk storage efficiently
- analyze the efficiency and performance of a file system by comparing different allocation and directory techniques

SGOU

Background

The rapid growth of computing systems and data-intensive applications has made efficient data storage and management a critical aspect of operating system design. A file system serves as an abstraction layer that provides a logical view of data while hiding the complexities of physical storage. It enables structured organization, efficient retrieval, and secure access to data stored on secondary storage devices. As modern applications generate large volumes of heterogeneous data, the role of the file system has expanded beyond basic storage to include performance optimization, reliability, and scalability. Concepts such as file organization, metadata management, and directory structures form the foundation for effective data handling in contemporary systems.

With advancements in storage technologies and system architectures, file systems have evolved to incorporate sophisticated mechanisms for allocation, indexing, and caching. Techniques such as contiguous, linked, and indexed allocation address trade-offs between access speed, space utilization, and fragmentation. Similarly, directory implementation methods and disk scheduling strategies significantly influence system performance. Modern file systems also integrate mechanisms for fault tolerance, concurrency control, and efficient resource utilization. This unit provides a comprehensive understanding of these advanced concepts, focusing on the design principles, efficiency considerations, and performance optimization techniques essential for developing robust and scalable storage systems.

Keywords

File System, File, Directory, Metadata, File Attributes, Layered Architecture, Logical File System. File Organization, Block, Linear List, Hash Table, Contiguous Allocation, Linked Allocation, Indexed Allocation, Fragmentation, Disk Scheduling

Discussion

3.2.1 Introduction to file system

A File System is a fundamental component of an Operating System (OS) that controls how data is stored, organized, managed, retrieved, and protected on secondary storage devices such as hard disks, solid-state drives (SSDs), pen drives, and memory cards. In simple terms, the file system acts as a bridge between the user and the physical storage device. Users interact with files using names and folders, while the operating system internally manages complex operations like disk allocation, indexing, and data retrieval. A file system ensures that data is not stored randomly on disk. Instead, it is stored in a structured and systematic manner, so that it can be easily accessed whenever needed.

3.2.1.1 Main purpose of file system

1. A file system helps to organize data into files on a computer.
2. It stores files in a proper and structured way on storage devices.
3. It helps users to easily save and open files when needed.
4. It manages the use of disk space efficiently.

SGOU

5. It assigns space for new files and frees space when files are deleted.
6. It helps in fast and easy retrieval of stored data.
7. It provides security by controlling who can access files.
8. It gives permissions like read, write, and execute to users.
9. It organizes files into folders and directories for easy management.
10. It acts as an interface between the user and the storage device.

3.2.2 File system structure

The file system structure defines how files are organized, stored, and managed inside a storage device such as a hard disk or SSD. It explains how the operating system handles files in a systematic way so that users can easily access and manage data without knowing the internal hardware details. The file system ensures that all files are stored in an organized manner and can be retrieved efficiently whenever required.

3.2.2.1 Layered structure of file system

A file system is designed using a layered architecture. In this architecture, different layers perform different tasks, and each layer depends on the layer below it. This structure makes file management easier, more organized, and more efficient.

1. Application layer

The application layer is the topmost layer of the file system. It consists of user applications such as MS Word, Excel, web browsers, and other software programs. Users interact with this layer directly when they perform file operations like creating, opening, saving, or deleting files. Whenever a user performs any file-related action, the request is generated in this layer and sent to the operating system for further processing.

2. Logical file system

The logical file system is responsible for managing the logical view of files. It handles important information such as file names, directory structure, and file metadata. It organizes files into folders and subfolders so that users can easily locate them. It also manages file permissions and ensures proper access control. This layer does not deal with physical storage; instead, it focuses only on how files are represented to the user.

3. File organization module

The file organization module is responsible for deciding how files are stored on the disk. It manages different file allocation methods such as contiguous allocation, linked allocation, and indexed allocation. It also handles free space management by keeping track of available disk blocks. This layer ensures that files are stored efficiently and helps in reducing fragmentation, which improves system performance.

4. Basic file system

SGOU

The basic file system is responsible for transferring data between memory and storage devices. It converts logical block addresses into physical block addresses so that data can be correctly accessed on the disk. It also handles buffering and caching of data to improve speed. This layer communicates with device drivers and ensures that file data is properly read from or written to the storage device.

5. I/O control layer

The I/O control layer manages communication between the operating system and hardware devices. It includes device drivers that control storage devices like hard disks and SSDs. This layer is responsible for performing low-level input and output operations such as reading and writing data. It converts high-level operating system commands into hardware-specific instructions so that the storage device can understand them.

6. Physical storage layer

The physical storage layer is the lowest layer of the file system structure. It consists of actual storage devices such as hard disks, solid-state drives, and USB drives. In this layer, data is stored physically in the form of binary digits (0s and 1s). The storage device organizes data into sectors and blocks, which are the smallest units of storage. This is where data is permanently stored.

3.2.2.2 File system components

The file system consists of several important components that help in managing data efficiently.

1. Files

Files are collections of related data stored in the system. A file can contain text, images, audio, video, or program instructions. Each file is treated as a separate unit and is identified by a unique name. Files are the basic storage units of a file system.

2. Directories

Directories are used to organize files in a structured manner. A directory can contain files as well as subdirectories. This helps in grouping related files together and makes it easier for users to search and manage data. Directories form a hierarchical structure similar to a tree.

3. Metadata

Metadata refers to the information about a file rather than the actual content of the file. It includes details such as file size, file type, location on disk, permissions, and creation or modification time. Metadata helps the operating system manage and control files effectively.

4. Blocks

Blocks are the smallest unit of storage in a disk. When a file is stored, it is divided into

SGOU

small blocks and stored across the disk. The file system keeps track of these blocks so that the file can be reconstructed when needed.

3.2.2.3 File Attributes

Every file in the system has certain attributes that describe its properties and control its behavior. A file has a **name**, which is used to identify it. It also has an **identifier**, which is a unique number used internally by the operating system. The file has a **type**, which defines whether it is a text file, image file, or executable file. It also stores the **location**, which indicates where the file is stored on the disk. The **size** attribute shows how much space the file occupies. The **protection** attribute defines access permissions such as read, write, or execute. Finally, **time stamps** store information such as creation time, modification time, and last access time. These attributes help the operating system manage files efficiently and securely.

3.2.3 Directory Implementation

A directory is a special file in the operating system that stores information about files such as file names, locations, and metadata. Directory implementation defines how these directory entries are stored inside memory and how efficiently files can be searched, accessed, modified, or deleted.

A good directory implementation improves system performance by reducing file search time and organizing files in a structured manner. It plays an important role in file system efficiency, especially when the number of files is very large.

The two main methods of directory implementation are:

- Linear list
- Hash table

3.2.3.1 Linear List Implementation

In linear list implementation, all directory entries are stored in a **sequential order** in a list. Each entry contains important details such as file name, file location, file size, and file attributes like permissions and timestamps. When a file needs to be accessed, the operating system performs a **sequential search**, meaning it checks each entry one by one from the beginning until the required file is found. This method is simple but time-consuming for large directories. This method is easy to understand and is commonly used in small or simple file systems where the number of files is limited. Example: Simple operating systems and small embedded systems use this method due to its simplicity.

Advantages:

- It is very simple to design and implement.
- New files can be easily added at the end of the list.
- It requires very little memory overhead compared to advanced methods.

SGOU

- It is suitable for small-scale systems where file count is low.

Disadvantages:

- File search is slow because each entry must be checked one by one.
- The time complexity is O(n), which means performance decreases as the number of files increases.
- Deleting or updating files may require shifting elements in the list.
- It is not suitable for large or complex file systems.
- Frequent searching makes the system inefficient for heavy workloads.

3.2.3.2 Hash Table Implementation

In hash table implementation, a **hash function** is used to convert the file name into a numeric index. This index directly points to the location of the file entry in the directory structure.

Instead of searching sequentially, the system directly jumps to the required location using the hash value. This significantly improves file access speed. Hash table implementation is widely used in modern operating systems where fast file access is required. Example: Modern operating systems like Linux and Windows use hash-based directory techniques for faster file access.

Advantages:

- File search is extremely fast (average case O(1)).
- It is highly efficient for large directories containing many files.
- It reduces the time required for file lookup operations.
- It improves overall system performance in large-scale systems.
- It supports quick insertion and retrieval of files.

Disadvantages:

- Collision may occur when two file names generate the same hash value.
- Collision handling techniques such as chaining or probing are required.
- It requires more memory compared to linear list implementation.
- The design of a good hash function is complex and critical for performance.
- If not properly designed, performance may degrade.

SGOU

Table 3.2.1 Comparison of Directory implementation methods

Feature	Linear List	Hash Table
Search Speed	Slow because sequential search is used	Very fast due to direct indexing
Complexity	Simple and easy to implement	More complex due to hashing logic
Memory Usage	Low memory requirement	Higher memory usage for hash table
Efficiency	Poor for large directories	Highly efficient for large directories
File Search Method	Sequential scanning	Direct access using hash function
Performance	Decreases as file count increases	Remains fast even with large data

3.2.4 File Allocation Methods

File allocation methods define how files are **stored on disk blocks** in a storage system. When a file is created, it is divided into smaller units called **blocks**, and the operating system decides how these blocks will be arranged on the disk. The main goal of file allocation is to use disk space efficiently and allow fast file access.

The three main file allocation methods are:

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

1. Contiguous Allocation

In contiguous allocation, all blocks of a file are stored in **continuous (adjacent) memory locations** on the disk. This means that once a starting address is known, all other blocks can be accessed easily by adding an offset.

For example, if a file starts at block 10 and has size 5 blocks, then it will occupy blocks 10, 11, 12, 13, and 14.

This method is simple and provides very fast access because the entire file is stored in one continuous sequence. It is used in systems where file size does not change frequently, such as CD-ROMs.

Advantages:

- It provides very fast access to files because blocks are stored sequentially.
- It is simple to implement and manage.
- It is efficient for sequential as well as direct access.

SGOU

- Less overhead is required for storing pointers or indexes.

Disadvantages:

- It suffers from **external fragmentation**, where free space is scattered.
- File size must be known in advance, which is not always possible.
- Expanding a file is difficult if adjacent space is not available.
- It may lead to inefficient disk utilization over time.

2. Linked Allocation

In linked allocation, each file is stored in **non-contiguous blocks**, and each block contains a pointer to the next block. This means the blocks of a file can be scattered anywhere on the disk. The first block contains a pointer to the second block, the second points to the third, and so on until the last block. File Allocation Table (FAT) uses a variation of linked allocation.

Advantages:

- There is no external fragmentation because blocks can be placed anywhere.
- File size can be easily increased or decreased.
- It uses disk space more efficiently.
- It is suitable for dynamic file systems.

Disadvantages:

- Random access is slow because blocks must be accessed sequentially.
- Pointer overhead increases storage usage.
- If a pointer is lost, the rest of the file may become inaccessible.
- Not efficient for applications requiring fast direct access.

3. Indexed Allocation

In indexed allocation, an **index block** is used to store all the addresses (pointers) of the file's data blocks. Each file has its own index block, which contains the list of all block locations.

Instead of storing pointers inside data blocks, all pointers are stored separately in one index structure. UNIX file systems use indexed allocation techniques.

For example, if a file uses blocks 5, 9, 12, and 20, then the index block will store these addresses.

Advantages:

- It supports direct access to file blocks.

SGOU

- There is no external fragmentation.
- It is flexible and supports dynamic file growth.
- Efficient for both sequential and random access.

Disadvantages:

- Additional storage is required for index blocks.
- Small files may waste space in index blocks.
- Index block failure may lead to file access issues.
- More complex compared to other methods.

3.2.2 Comparison of File allocation methods

Feature	Contiguous Allocation	Linked Allocation	Indexed Allocation
Storage Method	Continuous blocks	Non-contiguous blocks with pointers	Index block stores addresses
Access Speed	Very fast	Slow for random access	Fast for direct access
Fragmentation	External fragmentation	No external fragmentation	No external fragmentation
File Growth	Difficult	Easy	Easy
Space Efficiency	Low due to fragmentation	High	High
Complexity	Simple	Medium	Complex
Overhead	Low	Pointer overhead	Index block overhead
Best Use Case	Static files	Dynamic files	Large and random-access files

3.2.5 Efficiency and Performance of File System

Efficiency and performance are very important aspects of a file system in an operating system. The file system is responsible for storing, organizing, and retrieving data from storage devices. A good file system should ensure that file operations are performed **quickly, correctly, and with minimum use of system resources**.

Efficiency refers to how well the file system uses disk space and system resources, while performance refers to how fast file operations such as reading, writing, and searching are completed.

SGOU

3.2.5.1 Efficiency of file system

File system efficiency means how effectively the operating system uses disk space, memory, and processing power to manage files. It ensures that storage resources are utilized properly with minimum wastage and that file operations are performed smoothly and quickly. An efficient file system improves overall system performance by reducing delays in accessing and storing data. Factors affecting the efficiency of a file system are:

1. File Allocation Method

The file allocation method greatly affects the efficiency of a file system. It determines how files are stored on disk and how easily they can be accessed. Contiguous allocation uses disk space efficiently but may lead to fragmentation problems. Linked allocation avoids fragmentation but requires extra space for pointers, which reduces storage efficiency. Indexed allocation provides a balanced approach, but it introduces additional overhead due to index blocks.

2. Directory Structure

The structure of directories also affects file system efficiency because it controls how files are searched and accessed. A simple linear list structure is easy to implement but slow in searching files. In contrast, a hash table structure provides faster file search and improves efficiency, especially in large systems with many files.

3. Disk Space Utilization

Disk space utilization plays an important role in efficiency because it determines how well storage space is managed. An efficient file system ensures proper allocation and reuse of disk space when files are created and deleted. If disk space is not managed properly, it leads to fragmentation and wasted storage, which reduces system efficiency.

4. Caching and Buffering

Caching and buffering techniques improve file system efficiency by storing frequently used data in memory temporarily. This reduces repeated access to the disk, which is slower compared to memory access. As a result, file operations become faster and system performance is improved significantly.

3.2.5.2 Performance of file system

Performance refers to the **speed and responsiveness** of file operations. A high-performance file system completes operations like file creation, reading, writing, and deletion quickly. The factors affecting the performance of file system are:

1. Access Time

Access time is the time taken to locate and retrieve a file from storage. A good file system minimizes access time using indexing and caching techniques.

SGOU

2. Response Time

Response time is the time taken to complete a file request. It depends on disk speed, file organization, and system load.

3. Throughput

Throughput refers to the amount of data processed by the file system in a given time. Higher throughput indicates better performance.

4. Disk Scheduling

Disk scheduling algorithms improve performance by reducing seek time. Common algorithms include:

- FCFS (First Come First Serve)
- SSTF (Shortest Seek Time First)
- SCAN
- C-SCAN

These algorithms help in faster access to disk blocks.

3.2.5.3 Techniques to improve efficiency & performance

1. Caching

Frequently used files are stored in memory to reduce repeated disk access. This improves speed significantly.

2. Buffering

Data is temporarily stored in buffers before being written to disk, which reduces the number of disk operations.

3. Indexing

Indexing helps in faster file search by directly pointing to file locations instead of scanning all entries.

4. Efficient Allocation Methods

Using indexed or linked allocation improves flexibility and reduces fragmentation problems.

5. Proper Disk Management

Good disk space management avoids fragmentation and improves storage utilization.

SGOU

Summarized Overview

A **File System** is a vital component of an Operating System that organizes, stores, and retrieves data from storage devices such as hard disks and SSDs. It provides a convenient way for users and applications to access files and directories without dealing directly with physical storage structures.

Layered Architecture of File System

The file system is organized into multiple layers:

- **Application Layer** – Provides user interaction through applications.
- **Logical File System** – Manages file names, directories, and permissions.
- **File Organization Module** – Handles file allocation and free space management.
- **Basic File System** – Manages block mapping and data transfer.
- **I/O Control Layer** – Communicates with hardware using device drivers.
- **Physical Storage Layer** – Stores actual data on storage devices.

File Attributes

Every file contains important information such as:

- File name and identifier
- File type and location
- File size
- Access permissions
- Creation, modification, and access timestamps

Directory Implementation Methods

- **Linear List:** Stores files sequentially; simple but slower for large directories.
- **Hash Table:** Uses hash functions for quick file access; faster but more complex.

File Allocation Methods

- **Contiguous Allocation:** Stores files in consecutive blocks; offers fast access but may cause fragmentation.

SGOU

- **Linked Allocation:** Connects file blocks using pointers; eliminates fragmentation but slows random access.
- **Indexed Allocation:** Uses an index block to store addresses of file blocks; provides efficient and flexible access.

Efficiency and Performance

File system efficiency depends on effective allocation methods, directory structures, and disk space utilization. Performance is measured by:

- Access Time
- Response Time
- Throughput
- Disk Scheduling Techniques (FCFS, SSTF, SCAN, C-SCAN)

Techniques to Improve Efficiency and Performance

- **Caching:** Stores frequently accessed data in memory.
- **Buffering:** Temporarily holds data before writing to disk.
- **Indexing:** Speeds up file searching.
- **Efficient Allocation:** Minimizes fragmentation.
- **Proper Disk Management:** Optimizes storage utilization.

SGOU

Assignments

1. Explain the concept of a File System in an Operating System.
2. Discuss the layered architecture of a file system. Explain each layer with its functions and importance.
3. Explain different directory implementation methods.
4. Describe the various file allocation methods in detail.
5. Explain the efficiency and performance of a file system.

Reference

1. Tanenbaum, A. S., & Bos, H. (2024). *Modern operating systems* (5th ed.). Pearson.
2. Shukla, V. (2023). *Operating system* (Reprint ed.). S. K. Kataria & Sons.
3. https://www.tutorialspoint.com
4. https://www.geeksforgeeks.org
5. Stallings, W. (2018). *Operating systems: Internals and design principles* (9th ed.). Pearson.

Suggested Reading

1. Kaur, I., & Gupta, S. (2020). *Operating systems: A concept-based approach*. Dreamtech Press.
2. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts* (10th ed.). Wiley.
3. Dhamdhere, D. M. (2017). *Operating systems: A concept-based approach* (3rd ed.). McGraw-Hill Education.
4. Tanenbaum, A. S., & Bos, H. (2015). *Modern operating systems* (4th ed.). Pearson.

SGOU

Space for Learner Engagement for Objective Questions

Learners are encouraged to develop objective questions based on the content in the paragraph as a sign of their comprehension of the content. The Learners may reflect on the recap bullets and relate their understanding with the narrative in order to frame objective questions from the given text. The University expects that 1 - 2 questions are developed for each paragraph. The space given below can be used for listing the questions.

SGOU

3 UNIT I/O Systems

Learning Outcomes

After completion of this unit, the learner will be able to:

- identify basic I/O hardware components like controllers, buses, and DMA.
- explain how I/O works using polling, interrupts, and DMA with examples.
- describe different types of I/O devices based on data transfer, access, and speed.
- apply methods like buffering, caching, and spooling to improve performance.
- understand the role of the kernel I/O subsystem.

SGOU

Background

Understanding I/O hardware in operating systems requires a clear knowledge of how computers interact with external devices and manage data transfer efficiently. Modern computing systems rely heavily on input and output operations to communicate with devices such as disks, keyboards, printers, and network interfaces. The operating system acts as an intermediary, coordinating communication between the CPU and these peripherals through structured mechanisms like controllers, buses, and interfaces. Concepts such as polling, interrupts, and Direct Memory Access (DMA) evolved to improve the efficiency of data transfer and reduce CPU workload. As systems became more complex, advanced techniques like buffering, caching, and spooling were introduced to handle speed differences and optimize performance. Studying this topic is important because it helps in understanding

how data moves within a computer system and how hardware and software are tightly integrated. It is especially relevant in real-world applications such as database systems, file management, and high-speed data processing, where efficient I/O operations are critical. Knowledge of I/O hardware also enables better system design, troubleshooting, and performance optimization. Before studying this topic, learners should have a basic understanding of computer organization, including CPU, memory, and buses. Familiarity with operating system fundamentals such as processes, memory management, and basic programming concepts will further support a deeper understanding of I/O mechanisms and their practical applications.

Keywords

Device drivers, system bus communication, interrupt handling, DMA controller, buffering techniques, caching mechanism, spooling process

Discussion

3.3.1 I/O Hardware in Operating Systems

Input/Output (I/O) hardware consists of **specialized devices and components** that enable an operating system to interact with external peripherals such as disk drives, printers, and other input/output devices. These components are typically integrated into or connected through the motherboard and communicate with the processor via a **system bus**. Many I/O devices include dedicated **controllers**, which allow them to efficiently process requests from the operating system or even respond directly to application-level commands. Key concepts related to I/O hardware include structures such as daisy chaining, expansion buses, controllers, host adapters, memory-mapped I/O (MMIO), and Direct Memory Access (DMA).

Key Components and Mechanisms

Daisy Chain

This is a method of connecting multiple I/O devices in a sequential manner through a single connection point. Each device is linked to the next, forming a chain, and communication passes along this sequence. It simplifies wiring but may introduce delays as signals pass through multiple devices.

Expansion Bus

An expansion bus allows multiple peripherals to be connected **in parallel**, enabling simultaneous communication with the processor. This approach reduces the need for

SGOU

separate cables for each device and supports the addition of multiple peripherals while maintaining compatibility with existing systems.

Controller

A controller acts as an **intermediary between software and hardware**. It manages incoming data from devices and sends commands from the system to the hardware. For example, disk controllers and network controllers handle communication between the operating system and their respective devices.

Host Adapter

A host adapter serves as a **bridge between the system bus and expansion bus**, allowing multiple expansion cards to be connected. It often provides additional functionalities such as support for **DMA (Direct Memory Access)** and **MMIO (Memory-Mapped I/O)**, enhancing data transfer efficiency.

I/O hardware plays a crucial role in ensuring smooth communication between the operating system and peripheral devices. By using structured connection methods and dedicated controllers, the system can manage data transfer efficiently while supporting multiple devices simultaneously.

3.3.1.1 Polling

Polling is a **software-based method** used by the operating system to repeatedly check the status of an input/output device. The device may be a disk drive or any other peripheral, and the program continuously queries it to determine whether it is ready—for example, whether data is available or an operation has been completed.

This approach is generally considered **inefficient and slow**, because the processor must keep checking the device at regular intervals instead of performing other useful tasks. The CPU essentially "waits" by repeatedly asking the device for its status, which can lead to wasted processing time.

However, polling can be useful in certain situations. For instance, when multiple devices are being monitored and only one changes its state at a time, polling allows the system to detect updates in a controlled manner. It is also helpful for verifying whether a device is **active or offline**. If a device is found to be unavailable, the system can take appropriate actions, such as pausing or suspending tasks that depend on that device (for example, halting a backup process).

3.3.1.2 Interrupts

Interrupts are a mechanism through which the **CPU is signaled by hardware or software** when immediate attention is required. Various devices, including I/O hardware and peripherals, generate interrupts to notify the processor about events such as the completion of an operation or the need for data transfer.

Although multiple devices may request attention simultaneously, the CPU handles **one interrupt at a time**, processing them in an organized manner. Interrupts may also arise due to hardware or software issues—for example, a failure in a disk drive can halt

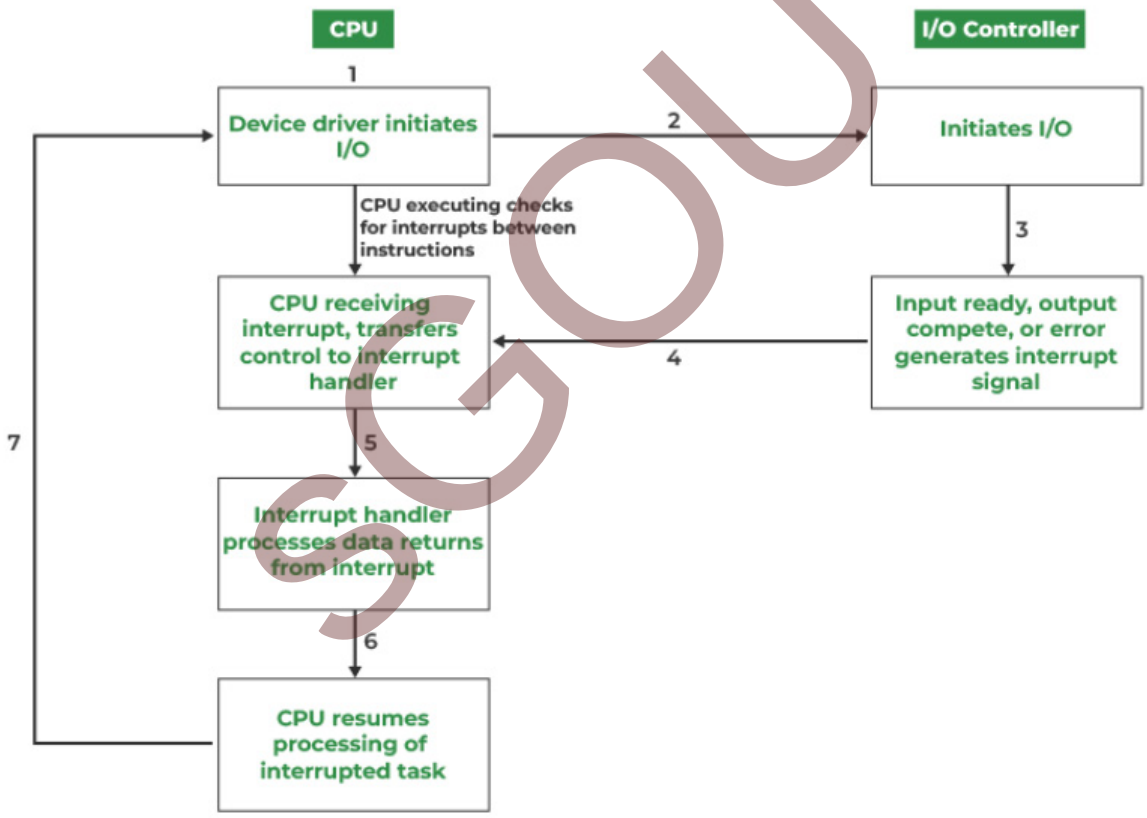
SGOU

ongoing data transfers until the issue is resolved, or a device may request access to memory resources.

In addition to hardware-generated interrupts, the operating system also manages **software-related interrupts** triggered by processes. For instance, when a user application performs an operation like writing data to a file and then terminates, the operating system completes the disk write operation before returning control. This capability supports **multitasking**, allowing multiple programs to run concurrently.

The operating system is responsible for allocating CPU time among different processes. It achieves this using a special routine known as an **Interrupt Service Routine (ISR)** or interrupt handler. When an interrupt occurs, the ISR is executed to handle the event, after which control is returned to the previously running program.

It is important to understand that while the CPU execution flow is temporarily redirected to handle interrupts, the operating system itself remains in control of managing and coordinating these interrupt-driven activities.



Interrupt-driven I/O cycle

SGOU

3.3.1.3 Direct Memory Access (DMA)

Direct Memory Access (DMA) is a technique that enables data to be transferred directly between memory and I/O devices without requiring continuous involvement of the CPU. Instead of the processor handling every step of data movement, a dedicated

DMA controller takes over the transfer, allowing the CPU to perform other tasks simultaneously and improving overall system efficiency.

When a program needs to read from or write to an I/O device (such as a disk drive), it first requests the operating system to initiate the operation. The operating system then configures the DMA controller by specifying details such as the memory location, direction of transfer, and amount of data. Once configured, the DMA controller manages the data transfer independently.

During this process, interrupts play an important role. After completing the transfer, the DMA controller sends an interrupt to notify the CPU that the operation has finished. This allows the system to coordinate tasks efficiently without constant CPU supervision.

Working Overview

1. The process requests data transfer through the operating system.
2. The OS sets up the DMA controller with necessary parameters.
3. The DMA controller transfers data directly between memory and the I/O device.
4. Once the transfer is complete, an interrupt is generated to inform the CPU.

I/O hardware refers to a collection of components and devices that enable the CPU and other processors in a computer system to interact with external devices. These components act as a bridge between the internal system and the outside world, supporting communication with peripherals such as keyboards, printers, and storage devices.

Some key elements include:

- Peripheral controller cards (I/O adapters): These are connected through expansion buses such as PCI or ISA and are responsible for managing the transfer of data between various parts of the computer system.
- Interrupt request lines (IRQs) and communication ports: These allow multiple peripheral devices-like keyboards, mice, and printers-to communicate efficiently without requiring separate dedicated communication channels.
- Device drivers: These are software programs that inform the operating system about the installed hardware, enabling proper interaction. They also manage dynamic aspects such as memory usage or storage availability, which may change over time.

I/O hardware is a fundamental part of an operating system, as it facilitates communication between the computer and external devices. Over time, I/O technology has advanced significantly, leading to a wide variety of interfaces and communication methods. Modern systems can choose from these options based on the specific requirements of applications, ensuring efficient and flexible device interaction.

SGOU

3.3.2 Applications of Input/Output (I/O) Interface

An Input/Output (I/O) interface is required whenever the CPU needs to communicate with peripheral devices. It acts as a communication layer that interprets the addresses and commands generated by the CPU and translates them into a form that I/O devices can understand. In this way, the interface enables smooth data exchange between the processor and external devices.

Applications and Uses of I/O Interface

- The I/O interface allows users and applications to access files and devices without needing detailed knowledge about their internal structure or operation. This abstraction simplifies system usage.
- It supports the integration of new hardware devices into a computer system without causing disruption to the operating system, making systems more flexible and scalable.
- The interface helps hide the differences between various I/O devices by categorizing them into general types. Each category can then be accessed through a standardized set of functions, ensuring consistency in operations.
- Different operating systems use their own device driver models to interact with hardware. As a result, a single hardware device may come with multiple drivers compatible with systems such as Windows, Linux, AIX, or macOS.

The I/O interface plays a vital role in simplifying communication, ensuring compatibility, and providing flexibility, enabling efficient interaction between the CPU and a wide range of devices without exposing their internal complexity.

S.No.	Basis	Alteration	Example
1.	Mode of Data-transfer	character or block	terminal disk
2.	Method of Accessing data	sequential or random	modem, CD-ROM
3.	Transfer schedule	synchronous or asynchronous	tape, keyboard
4.	Sharing methods	dedicated or sharable	tape, keyboard
5.	Speed of device	latency, seek time, transfer rate, delay between operations	
6.	I/O Interface	read only, write only, read-write	CD-ROM graphics controller disk

SGOU

3.3.2.1 Detailed Classification of I/O Devices

I/O devices differ widely in how they transfer data, allow access, coordinate with the CPU, and support operations. Understanding these characteristics helps in designing efficient operating systems and choosing the right device for a specific application.

1. Character Stream vs Block Devices

Both types deal with data in the form of bytes, but their handling approach is different:

- **Character Stream Devices**

These devices transmit data one byte at a time in a continuous flow. There is no grouping or buffering into large chunks, and data is processed immediately as it arrives.
Examples: Keyboard, mouse, serial communication ports
When you press a key, the system receives it instantly as a single character. These devices are typically used for real-time input, where immediate processing is more important than efficiency.

- **Block Devices**

These devices transfer data in fixed-size blocks, meaning multiple bytes are grouped together and processed as a unit.

Examples: Hard disks, SSDs, USB drives

Data is read or written in chunks (e.g., 512 bytes or 4 KB), which improves efficiency and performance. Operating systems often use buffering and caching with block devices to speed up access.

2. Sequential vs Random Access

This classification describes how data is retrieved:

- **Sequential Access Devices**

Data must be accessed in a fixed order, one piece after another.

Example: Magnetic tape storage

If you want to access data in the middle, the system must pass through earlier data first.

This makes access slower but is useful for backup systems and archival storage, where data is processed in order.

- **Random Access Devices**

These allow direct access to any data location without following a sequence.

Examples: Hard drives, SSDs, RAM

The system can jump directly to the required location, making it ideal for applications requiring fast retrieval, such as databases or operating system files.

SGOU

3. Synchronous vs Asynchronous Devices

This classification is based on timing and coordination:

- **Synchronous Devices**

These operate in coordination with system timing, often following a clock or predictable schedule.

Example: Real-time sensors, high-speed data acquisition systems

Data transfer occurs at fixed intervals, which is important in systems where timing accuracy is critical, such as embedded systems or industrial automation.

- **Asynchronous Devices**

These operate independently of system timing, with unpredictable response times.

Examples: Keyboard, mouse, network input

The system cannot predict when input will occur, so it relies on interrupts to handle events as they happen.

4. Sharable vs Dedicated Devices

This depends on how devices are used by processes:

- **Sharable Devices**

Multiple processes can use the device simultaneously without conflict.

Example: Hard disk, network interface

The operating system manages access using techniques like scheduling and buffering, allowing multiple users or programs to share resources efficiently.

- **Dedicated Devices**

Only one process can use the device at a time.

Example: Printer, tape drive

If another process requests the device, it must wait until the current task is completed.

This may require spooling (queueing jobs) to improve efficiency.

5. Speed of Operation

I/O devices vary greatly in speed depending on their function:

- Low-Speed Devices: Transfer small amounts of data slowly

Example: Keyboard, mouse

- Medium-Speed Devices: Moderate data transfer rates

SGOU

Example: Printers, scanners

- High-Speed Devices: Capable of transferring large volumes of data quickly

Example: SSDs, network interfaces, GPUs

Because of this variation, operating systems use techniques like buffering, caching, and DMA (Direct Memory Access) to optimize performance and reduce CPU load.

6. Read-Write, Read-Only, Write-Only Devices

This classification depends on the direction of data flow:

- Read-Write Devices

Support both input and output operations

Examples: Hard disks, USB drives

These devices are versatile and commonly used for general-purpose storage.

- Read-Only Devices

Allow only data retrieval

Example: CD-ROM

Data cannot be modified, making them suitable for secure distribution of information.

- Write-Only Devices

Allow only output operations

Example: Printer

These devices do not send data back to the system, but simply produce output.

These classifications highlight how different I/O devices behave and interact with the system. By understanding these characteristics, operating systems can apply appropriate techniques such as scheduling, buffering, caching, and synchronization to ensure efficient and reliable device management.

3.3.3 Kernel I/O Subsystem in Operating Systems

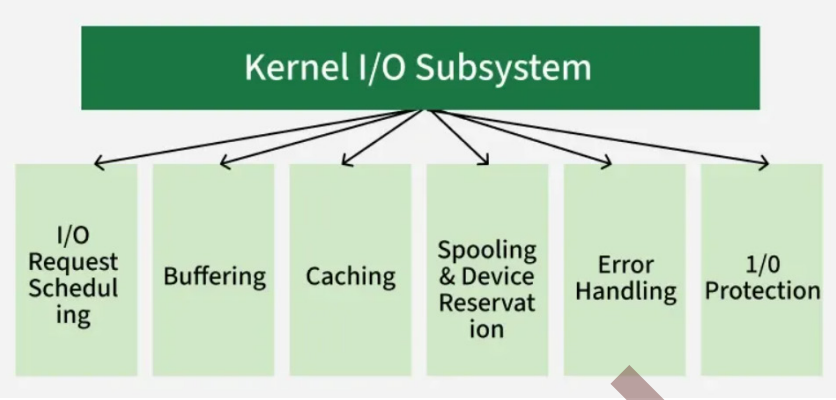
The Kernel I/O Subsystem is a crucial part of the operating system kernel that manages the interaction between the CPU and input/output devices such as disks, printers, keyboards, and network interfaces. It acts as an intermediary layer, converting high-level I/O requests from applications into low-level instructions that hardware devices can understand and execute.

In addition to facilitating communication, the kernel I/O subsystem is responsible for several important functions, including handling concurrent access to devices, ensuring proper synchronization, managing errors, and maintaining system security. It also

SGOU

improves performance through techniques such as buffering, caching, and spooling, while enforcing protection mechanisms to ensure safe and controlled device usage.

Overall, it plays a vital role in ensuring that data transfer between software and hardware is efficient, reliable, and well-coordinated.



3.3.3.1 Functions Offered by the Kernel I/O Subsystem

1. I/O Scheduling

I/O scheduling is the process of deciding the most efficient order in which input output requests should be executed. Its main objective is to improve system performance by reducing waiting time and turnaround time while ensuring fair access to devices for all processes. The operating system maintains waiting queues for each device where incoming requests are stored, and the scheduler rearranges these requests using suitable algorithms to achieve better efficiency and resource utilization. For example, when multiple processes issue blocking I/O requests such as disk operations, their requests are placed in a queue and reordered by the scheduler instead of being handled strictly in the order of arrival, which helps minimize delays and improve overall throughput.

2. Buffering

Buffering is a technique in which a temporary memory region is used to hold data while it is being transferred between devices or between a device and an application. This intermediate storage helps smooth out differences in data flow and improves overall efficiency.

Buffering is mainly used to handle situations where there is a difference in speed between the data producer and the consumer, ensuring that faster components do not have to wait for slower ones. It also helps in adjusting differences in data transfer sizes, allowing data to be processed in suitable chunks. In addition, buffering supports copy operations, where data from an application is first stored in a buffer before being written to a device such as a disk, ensuring safe and reliable transfer.

3. Caching

Caching is a technique in which **frequently used data is stored in faster memory** so that it can be accessed more quickly when needed again. By keeping a copy of commonly

SGOU

accessed information closer to the CPU, the system reduces the need to retrieve data from slower storage devices, thereby improving performance. For instance, frequently executed instructions or data may be stored in the CPU cache or main memory to avoid repeated access from slower disks.

Caching is different from buffering. A **buffer** temporarily holds data during transfer between devices or processes, whereas a **cache** retains a copy of data specifically to speed up future access.

4. Spooling and Device Reservation

Spooling is a technique in which output data, such as print jobs, is **temporarily stored on disk** before being sent to the device. This prevents outputs from different processes from getting mixed. The jobs are placed in a **queue, typically processed in first come first serve order**, and executed one after another. This approach allows multiple processes to submit tasks simultaneously while ensuring that the device handles them in an organized manner without conflicts.

5. Error Handling

The operating system incorporates reliable error handling mechanisms to manage both temporary and permanent I/O issues. It includes methods to detect errors, report them to the system and applications, and attempt recovery whenever possible without interrupting overall system operation. Additionally, users or administrators are informed about such issues through notifications or alerts, ensuring timely awareness and corrective action.

6. I/O Protection

I/O protection ensures that unauthorized or incorrect access to devices is prevented. Since I/O operations can directly affect system stability, they are treated as privileged instructions. Only the operating system kernel is allowed to execute these instructions, thereby safeguarding the system from errors or malicious actions by user-level processes.

3.3.3.2 Functions and Services Offered by the Kernel

The kernel delivers a wide range of services that support system operations, particularly in managing processes, memory, input output, and security.

- Process Management:

The kernel is responsible for handling processes by saving the state of interrupted programs, selecting and dispatching processes for execution, and maintaining scheduling queues to ensure efficient CPU utilization.

- Process Communication:

It enables interaction between processes by allowing them to send and receive messages, supporting coordination and data exchange in multitasking environments.

- Memory Management:

The kernel controls memory usage by enforcing protection mechanisms, handling page

SGOU

faults, and managing operations such as swapping data between main memory and secondary storage.

- I O Management:

It manages input output operations by initiating device requests, handling completion interrupts, and dealing with errors that may occur during data transfer.

- File Management:

The kernel provides mechanisms to open files, read and write data, and manage file-related operations within the system.

- Security and Protection:

It ensures system safety by managing authentication details for users and maintaining access control information for files and other resources.

- Network Management:

The kernel supports communication over networks by sending and receiving data through message based mechanisms.

3.3.3.3 Importance of the Kernel I/O Subsystem

The kernel I/O subsystem plays a vital role in the operating system by offering a uniform and consistent interface through which applications can interact with various devices. It hides the underlying hardware details, thereby simplifying device access and ensuring device independence.

It also handles situations where multiple processes attempt to use the same device by providing proper concurrency control and synchronization mechanisms, ensuring smooth and conflict free operation. Additionally, it maintains data integrity and enforces protection measures to prevent unauthorized or improper I/O access, thereby enhancing overall system security and reliability.

SGOU

Summarized Overview

I/O hardware in operating systems consists of specialized components that enable communication between the CPU and external devices such as disks, printers, and input devices. These components are connected through system buses and often include controllers and host adapters that manage data transfer efficiently. Techniques like daisy chaining and expansion buses help organize device connectivity, allowing multiple peripherals to function together. Polling and interrupts are two methods used to manage I/O operations, where polling repeatedly checks device status while interrupts notify the CPU only when attention is needed. Direct Memory Access (DMA) further improves performance by allowing data transfer between memory and devices without constant CPU involvement. Device drivers and I/O interfaces act as a bridge between hardware and the operating system, simplifying communication and ensuring compatibility. I/O devices are classified based on factors such as data transfer mode, access method, timing, sharing, and speed, which helps in efficient system design. The kernel I/O subsystem plays a central role by managing communication, synchronization, and error handling between software and hardware. It also improves performance through techniques like buffering, caching, and spooling. Overall, I/O hardware and its supporting mechanisms ensure efficient, reliable, and secure interaction between the system and external devices.

SGOU

Assignments

1. Explain the concept of I/O hardware in operating systems. Discuss its role in enabling communication between the CPU and peripheral devices with suitable examples.

2. Describe the key components of I/O hardware such as daisy chaining, expansion bus, controller, and host adapter. Explain how each contributes to efficient data transfer.

3. Compare and contrast polling and interrupts as I/O handling mechanisms. Discuss their advantages, disadvantages, and suitable use cases.

4. Explain the concept of Direct Memory Access (DMA). Describe its working process and discuss how it improves system performance compared to CPU-driven data transfer.

5. Discuss the role of device drivers and I/O interfaces in an operating system. Explain how they help in achieving device independence and system flexibility.

6. Classify I/O devices based on different criteria such as data transfer mode, access method, timing, sharing, and speed. Provide relevant examples for each category.

7. Explain the functions of the Kernel I/O Subsystem in detail. Discuss concepts such as I/O scheduling, buffering, caching, spooling, and error handling with examples.

8. Discuss the importance of the Kernel I/O Subsystem in modern operating systems. Explain how it ensures efficiency, protection, and coordination in handling multiple I/O requests.

Reference

1. Arpaci-Dusseau, R. H., & Arpaci-Dusseau, A. C. (2018). *Operating systems: Three easy pieces*. Arpaci-Dusseau Books.

2. Love, R. (2010). *Linux kernel development* (3rd ed.). Addison-Wesley.

3. Bovet, D. P., & Cesati, M. (2005). *Understanding the Linux kernel* (3rd ed.). O'Reilly Media.

Suggested Reading

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts* (10th ed.). Wiley.

2. Stallings, W. (2018). *Operating systems: Internals and design principles* (9th ed.). Pearson.

3. Tanenbaum, A. S., & Bos, H. (2015). *Modern operating systems* (4th ed.). Pearson.

4. Dhamdhere, D. M. (2017). *Operating systems: A concept-based approach* (3rd ed.). McGraw Hill Education.

SGOU

Space for Learner Engagement for Objective Questions

Learners are encouraged to develop objective questions based on the content in the paragraph as a sign of their comprehension of the content. The Learners may reflect on the recap bullets and relate their understanding with the narrative in order to frame objective questions from the given text. The University expects that 1 - 2 questions are developed for each paragraph. The space given below can be used for listing the questions.

SGOU

4 UNIT Disk Management

Learning Outcomes

After completion of this unit, the learner will be able to:

- explain the concept of disk management and disk structure
- describe different disk scheduling algorithms
- calculate seek time using various disk scheduling algorithms
- compare different disk scheduling algorithms based on performance
- analyze the working and importance of swap-space management

Background

In modern computer systems, efficient management of storage devices is essential for overall system performance. Secondary storage devices such as hard disks and solid-state drives store large volumes of data, but they are significantly slower than main memory. As multiple processes generate input/output requests simultaneously, the operating system must manage how data is accessed and transferred between memory and disk. This requirement led to the development of disk management techniques, which include organizing data on disk structures, controlling disk access, and optimizing read/write operations. Without proper management, the disk head may move randomly across tracks, increasing access time and reducing system efficiency.

To address these challenges, operating systems use disk scheduling algorithms to decide the order in which disk requests are serviced. These algorithms aim to minimize seek time, reduce rotational delay, and improve throughput. Techniques such as FCFS, SSTF, SCAN, C-SCAN, LOOK, and C-LOOK provide different strategies to balance efficiency and fairness. In addition, swap-space management plays

SGOU

a vital role in extending memory capacity by using disk space as virtual memory. When RAM becomes full, less-used processes are temporarily moved to disk, allowing the system to continue running multiple applications smoothly. Together, disk scheduling and swap-space management ensure better resource utilization, improved performance, and stable operation of modern computing systems.

Keywords

Disk Management, Disk Structure, Seek Time, Rotational Latency,Transfer Time,Disk Scheduling, FCFS Algorithm, SSTF Algorithm,SCAN Algorithm,C-SCAN Algorithm,LOOK Algorithm,C-LOOK Algorithm,Head Movement, Swap Space

Discussion

3.4.1 Introduction

Disk management is an important function of the operating system that deals with the organization, storage, allocation, and access of data on secondary storage devices such as hard disks and SSDs. Since disks are slower compared to main memory, efficient disk management is necessary to improve system performance. The operating system performs various tasks such as managing disk structure, handling disk scheduling, and maintaining swap space to ensure fast and efficient data access.

3.4.2 Disk structure

Disk structure refers to the physical and logical organization of data on a disk. A disk consists of circular plates called platters where data is stored magnetically. Each platter has two surfaces, and both sides can store data, which increases the storage capacity of the disk. The platters rotate at high speed, allowing the system to read and write data efficiently.

3.4.2.1 Components of disk structure

- **Platter:** A circular disk coated with magnetic material used to store data. Multiple platters are stacked together to increase storage capacity.
- **Track:** A circular path on the platter where data is stored in an organized manner. Each platter contains many tracks.
- **Sector:** The smallest unit of storage on a track, where actual data is stored. Each track is divided into several sectors.

SGOU

- **Cylinder:** A group of tracks located at the same position on different platters. Accessing data within the same cylinder is faster because the head does not need to move.
- **Head:** A device that reads and writes data from and to the disk. Each platter surface has its own read/write head.

When data is stored, it is written into sectors on the disk. The disk head moves to the correct track, and the platter rotates until the required sector comes under the head. The read/write head then performs the operation of reading or writing data.

Disk access time mainly depends on seek time, which is the time taken to move the head to the required track, rotational delay, which is the time taken for the desired sector to rotate under the head, and transfer time, which is the time required to transfer data.

The overall performance of the disk is affected by how quickly these operations are completed. Efficient disk management techniques aim to minimize seek time and rotational delay to improve speed. Additionally, modern disks use techniques such as caching and scheduling algorithms to optimize data access and enhance performance.

3.4.3 Disk attachment

Disk attachment refers to how storage devices are connected to a computer system. It determines how data is accessed, shared, and managed between the computer and storage devices. The method of disk attachment affects system performance, data transfer speed, and the ability to share data among multiple users.

3.4.3.1 Types of disk attachment

1. Direct attached storage (DAS)

In this method, the disk is directly connected to the computer using cables such as SATA or USB. The storage device is dedicated to a single system, and data access is controlled by that computer only. It is simple to set up and provides fast data transfer because there is no network delay.

Direct attached storage is commonly used in personal computers and small systems where data sharing is not required. However, it is not suitable for environments where multiple users need to access the same data.

2. Network attached storage (NAS)

In this method, storage devices are connected through a network, allowing multiple users and systems to access the data. NAS devices act as centralized storage systems and are usually connected via Ethernet.

It is suitable for organizations where data sharing is important. NAS systems are easy to manage and provide features such as remote access, data backup, and file sharing. However, performance may be affected by network speed and traffic.

SGOU

3. Storage area network (SAN)

This is a high-speed network used to connect storage devices to multiple servers. SAN provides block-level storage access and is designed for high performance and large-scale environments.

It is mainly used in large organizations and data centers where high speed, reliability, and scalability are required. SAN systems support large volumes of data and provide faster access compared to NAS. However, they are expensive and complex to set up and maintain.

3.4.4 Disk scheduling algorithms

Disk scheduling algorithms are used in an operating system to manage how data is read from and written to a computer's hard disk. These algorithms determine the order in which multiple disk read and write requests are processed, ensuring efficient utilization of disk resources. Since many processes may generate I/O requests simultaneously, proper scheduling is essential to maintain system performance.

Disk scheduling is also referred to as I/O Scheduling, as it deals with input/output operations involving storage devices. The primary objective of disk scheduling is to optimize disk performance, minimize access time, and improve overall system efficiency. By selecting an appropriate sequence of requests, the operating system can reduce unnecessary disk head movement and speed up data access. Common disk scheduling techniques include First-Come First-Served (FCFS), Shortest Seek Time First (SSTF), SCAN, C-SCAN, LOOK, and C-LOOK, each with its own advantages and limitations. In a multitasking environment, multiple processes generate I/O requests that must be handled by the disk controller. However, only one request can be processed at a time, while the remaining requests are placed in a waiting queue. Without proper scheduling, the disk head may move randomly across tracks, resulting in increased seek time and reduced performance.

Additionally, when requests are located far apart on the disk, excessive movement of the disk arm may occur, leading to higher access time. Since hard disks are among the slowest components in a computer system, efficient scheduling becomes necessary to ensure faster data access, reduced waiting time, and improved system throughput.

- **Key Terms Associated with Disk Scheduling**
  - **Seek Time**: The time required for the disk arm to move the read/write head to the desired track where data is stored. It is the total distance travelled by the disk head while servicing all the requests.

    $$\text{Seek Time} = \sum |\text{Current Position} - \text{Next Request}|$$

  - **Rotational Latency**: The time taken for the disk to rotate so that the required sector comes under the read/write head.
  - **Transfer Time**: The time required to transfer data between the disk and memory, depending on disk speed and data size.

SGOU

3.4.5 Types of disk scheduling algorithms

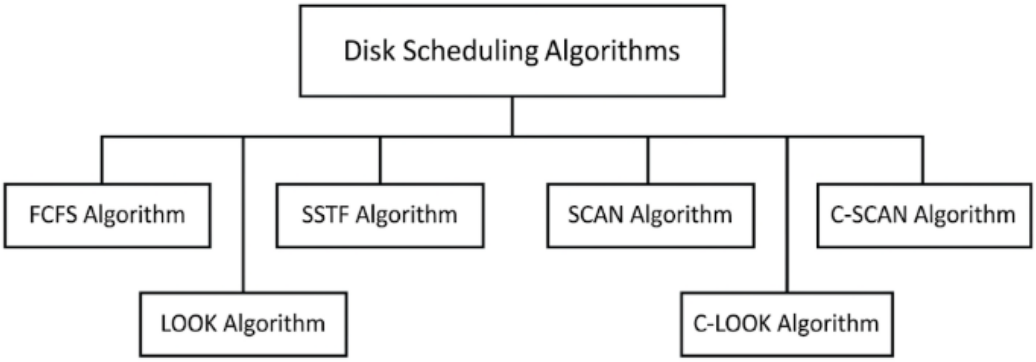


Fig 3.4.1: Disk scheduling algorithms

3.4.5.1 First-Come, First-Served (FCFS) Disk Scheduling

FCFS (First-Come, First-Served) disk scheduling algorithm is the simplest and most straightforward method in which disk I/O requests are serviced strictly in the order they arrive in the queue, following a FIFO (First-In, First-Out) principle without any reordering or optimization. The disk head moves from its current position to each requested track sequentially, which makes the algorithm easy to implement and ensures fairness, as every request gets a chance without starvation. However, FCFS does not consider the distance between tracks, so it often results in large and inefficient head movements, leading to increased seek time and poor overall performance, especially when requests are scattered across the disk. This can also cause a problem known as high variance in response time, where some requests experience significant delays. Despite its inefficiency, FCFS is useful in systems where simplicity and fairness are more important than performance optimization, and it often serves as a baseline for comparing other advanced disk scheduling algorithms.

Example: Suppose a disk having 200 tracks (0-199) The request sequence (82,170,43,140,24,16,190) of the disk is shown as in the given figure and the head start is at request 50.

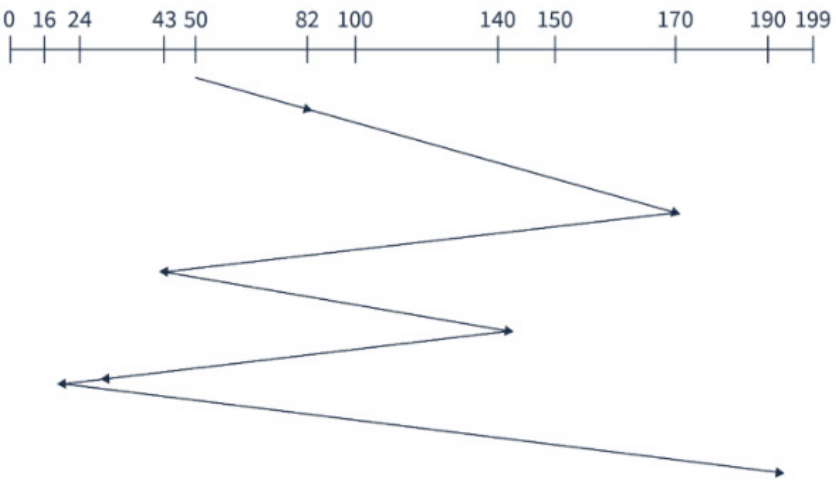


Fig 3.4.2: FCFS Disk Scheduling Head Movement Diagram

SGOU

In the given example, the disk head initially starts at position 50 and moves to the first request at 82. After servicing this request, the disk arm proceeds sequentially to the next requests in the queue, moving from 82 to 170, then to 43, followed by 140, 24, 16, and finally 190. In the FCFS (First-Come, First-Served) algorithm, the disk arm services all requests strictly in the order of their arrival, without any reordering or optimization. Thus, each request is handled one by one until all pending requests are completed.

The total seek time is calculated by summing the absolute differences between consecutive head movements:

$$\text{Seek Time} = |82 - 50| + |170 - 82| + |43 - 170| + |140 - 43| + |24 - 140| + |16 - 24| + |190 - 16| = 642$$

Thus, the total seek time required to service all the requests using the FCFS algorithm is 642 tracks.

Advantages:

- The FCFS algorithm is easy to implement and understand.
- It ensures fairness, as there is no chance of starvation.

Disadvantages:

- The seek time increases because the disk head movement is not optimized.
- It is not efficient for systems with heavy disk workloads.

3.4.5.2 SSTF disk scheduling algorithm

Shortest Seek Time First (SSTF) disk scheduling algorithm selects and services the disk I/O request that is closest to the current position of the disk head, thereby minimizing the immediate seek distance. At each step, the operating system computes the distance between the current head location and all pending requests, chooses the one with the minimum seek time, services it, and then repeats the process from the new head position. This greedy approach significantly reduces total head movement and average seek time compared to FCFS, leading to better throughput and faster response in many cases. However, SSTF has a major drawback: it can cause starvation, where requests that are far from the current head position may wait indefinitely if closer requests keep arriving. It may also lead to non-uniform response times and can be slightly more complex to implement due to continuous distance calculations. Despite these limitations, SSTF is widely used as a performance-improving strategy and often serves as a basis for more advanced algorithms like SCAN and LOOK.

Example : Suppose a disk has 200 tracks (0-199). The request sequenc (82,170,43,140,24,16,190) is shown in the given figure and the head position is at 50.

In the SSTF (Shortest Seek Time First) algorithm, the disk arm selects the request that

SGOU

has the minimum seek time from the current head position, i.e., the nearest request. Initially, the head is at position 50, and the closest request is 43. After servicing 43, the next nearest request is 24, followed by 16. From 16, the closest request becomes 82, then 140, 170, and finally 190. Thus, the disk head always moves to the nearest pending request at each step.

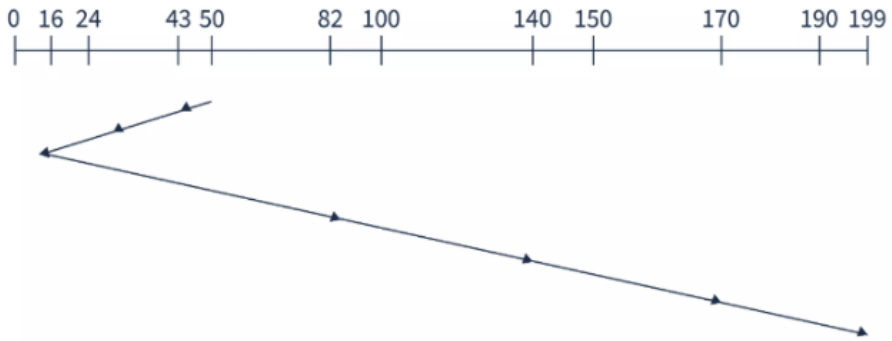


Fig 3.4.3 : SSTF Disk Scheduling Head Movement Diagram

Seek Time Calculation:

The total seek time is calculated by summing the absolute differences between consecutive head movements:

$$|50-43|+|43-24|+|24-16|+|82-16|+|140-82|+|170-140|+|190-170|=208$$

Therefore, the total seek time using the SSTF algorithm is 208 tracks.

Advantages:

- The SSTF algorithm reduces disk response time by selecting the nearest request.
- It is more efficient than the FCFS algorithm in terms of seek time and overall performance.
- It significantly reduces total head movement, leading to better throughput.

Disadvantages:

- The execution of the algorithm is slower due to continuous calculation of the nearest request.
- Starvation may occur, as requests that are far from the current head position may not be served for a long time.
- It may lead to non-uniform response time, making the system less predictable.

SGOU

3.4.5.3 SCAN disk scheduling algorithm:

The SCAN disk scheduling algorithm, also known as the *elevator algorithm*, is a disk scheduling technique in which the disk arm moves in a fixed direction (either inward toward the center or outward toward the end of the disk) and services all pending requests along its path. Once it reaches the last track in that direction, it reverses its movement and continues servicing requests in the opposite direction. This method ensures that the disk head moves smoothly across the disk, reducing unnecessary back-and-forth movement compared to simpler methods like FCFS. By processing requests in an ordered sequence based on their position, SCAN improves overall system efficiency and provides a more predictable service pattern.

One of the major advantages of the SCAN algorithm is that it offers better performance and lower average seek time than many basic scheduling techniques, especially when there are multiple requests distributed across the disk. It also ensures that every request will eventually be served, thereby avoiding indefinite postponement (starvation). However, the algorithm may still cause longer waiting times for requests that arrive just after the disk head has passed their position, as they must wait for the head to complete its full traversal and return. Despite this limitation, SCAN remains widely used in operating systems due to its balance between efficiency and fairness in handling disk I/O requests.

Example: Suppose a disk has 200 tracks (0-199) The request seque nce(82,170,43,140,24,16,190) is shown in the given figure and the head position is at 50. The 'disk arm' will first move to the larger values.

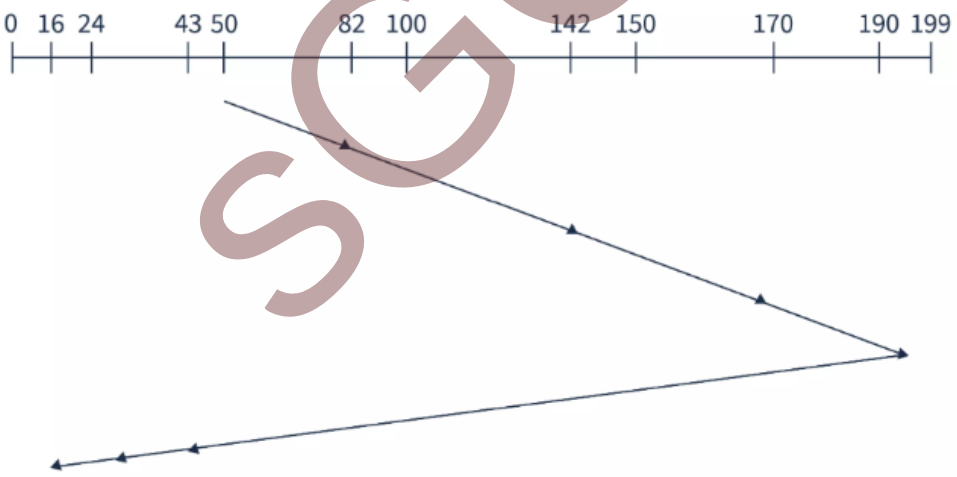


Fig 3.4.4 : Scan Disk Scheduling Head Movement Diagram

Based on the diagram, the disk arm begins at position 50 and moves in one direction (towards the higher track numbers) until it reaches the end of the disk at position 199. After reaching this end, the disk arm reverses its direction and starts servicing the remaining requests while moving towards the lower track numbers, eventually reaching position 16. This back-and-forth movement continues until all requests are completed.

SGOU

Therefore, the total seek time is calculated as the sum of the movement from 50 to 199 and then from 199 to 16:

Seek Time = (199 − 50) + (199 − 16) = 332

Advantages:

- Implementation is simple and easy to understand.
- Requests do not have to wait for a long time in a queue.
- Provides better performance compared to FCFS due to reduced head movement.
- Ensures more uniform waiting time for requests.

Disadvantages:

- The disk head moves to the end even if there are no requests in that direction, causing unnecessary movement.
- It may increase seek time compared to optimized algorithms like LOOK.
- Slightly more complex than FCFS in terms of movement handling.

3.4.5.4 C-SCAN disk scheduling algorithm

C-SCAN (Circular SCAN) disk scheduling algorithm is an improved version of the SCAN (elevator) algorithm designed to provide more uniform wait time. In this method, the disk arm moves in only one direction (for example, from the lowest track to the highest track) and services all the requests along its path. Once it reaches the end of the disk, instead of reversing direction, the head quickly returns to the beginning of the disk without servicing any requests during the return. After reaching the starting point, it again moves in the same direction servicing pending requests. This circular movement ensures that all requests are treated more equally, avoiding the bias that may occur in SCAN.

One of the main advantages of C-SCAN is that it provides a more uniform waiting time compared to SCAN, as each request waits for at most one full cycle of the disk. It also reduces the chances of starvation and offers better response time for requests located at different positions on the disk. However, a drawback is that the head has to travel a long distance without servicing any requests when it jumps from the end back to the beginning, which may increase the overall seek time. Despite this, C-SCAN is widely used in systems where fairness and uniform response time are important.

Example: Suppose a disk having 200 tracks (0-199). The request seque nce(82,170,43,140,24,16,190) is shown in the given figure and the head position is at 50.

In the given figure, the disk arm initially starts at position 50 and moves in one direction towards the end of the disk at position 199, servicing all the requests along its path.

SGOU

After reaching the end, instead of continuing in the same direction, the head jumps back to the beginning of the disk (position 0) without servicing any requests during this return movement.

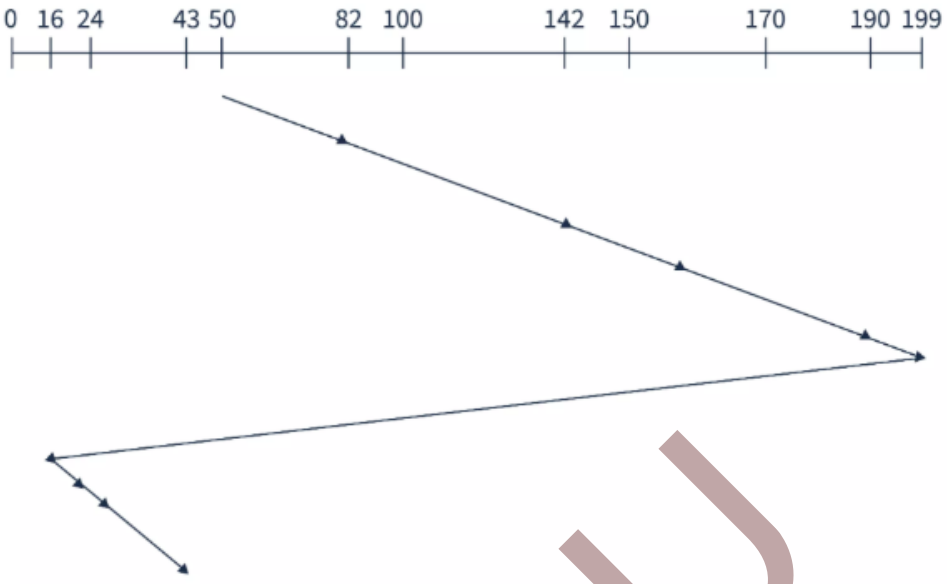


Fig 3.4.5 : C-SCAN disk scheduling Head Movement Diagram

Once it reaches position 0, the disk arm again starts moving in the same direction (towards higher track numbers) and services the remaining requests, beginning from 43 and continuing further. This circular movement continues until all requests are completed.

Hence, the total seek time is calculated as:

Hence, Seek Time will be =(199−50)+(199−0)+(43−0)=391

Advantages of C-SCAN Disk Scheduling Algorithm

- The waiting time is uniformly distributed among all requests.
- It provides better response time compared to SCAN.
- It treats all requests equally, giving fair service.
- It avoids starvation of requests.

Disadvantages of C-SCAN Disk Scheduling Algorithm

- The time taken by the disk arm to locate a track is increased.
- The head moves to the end of the disk even if no request is there.
- After reaching the end, it jumps back to the beginning without servicing requests.

SGOU

- This jump causes additional delay and extra head movement.

3.4.5.5 LOOK disk scheduling algorithm

The LOOK disk scheduling algorithm is an improved version of the SCAN (elevator) algorithm designed to reduce unnecessary disk head movement. In this method, the disk arm moves in a particular direction (either inward or outward) and services all pending requests in that path. However, unlike SCAN, the head does not go to the extreme end of the disk unless there is a request there. Instead, it "looks" ahead and only travels as far as the last request in that direction, then reverses its movement. This intelligent decision-making helps in minimizing seek time and improves efficiency. For example, if the current head position is 50 and the requests are up to 150 in one direction, the head will stop at 150 instead of going to the maximum disk limit (like 199), thereby saving time and movement.

The LOOK algorithm provides better performance compared to FCFS and SCAN by reducing total head movement and improving response time. It ensures that all requests are serviced in an orderly manner without unnecessary traversal, making it more efficient for real-time systems where performance matters. Additionally, it offers a more balanced waiting time for requests compared to SSTF, reducing the chances of starvation. However, like other scheduling techniques, LOOK may still have slight delays for requests that are just missed in the current direction, as they must wait for the head to reverse. Overall, the LOOK algorithm strikes a good balance between efficiency and fairness, making it a widely preferred disk scheduling technique in operating systems.

Example The request sequence(82,170,43,140,24,16,190) are shown in the given figure and the head position is at 50.

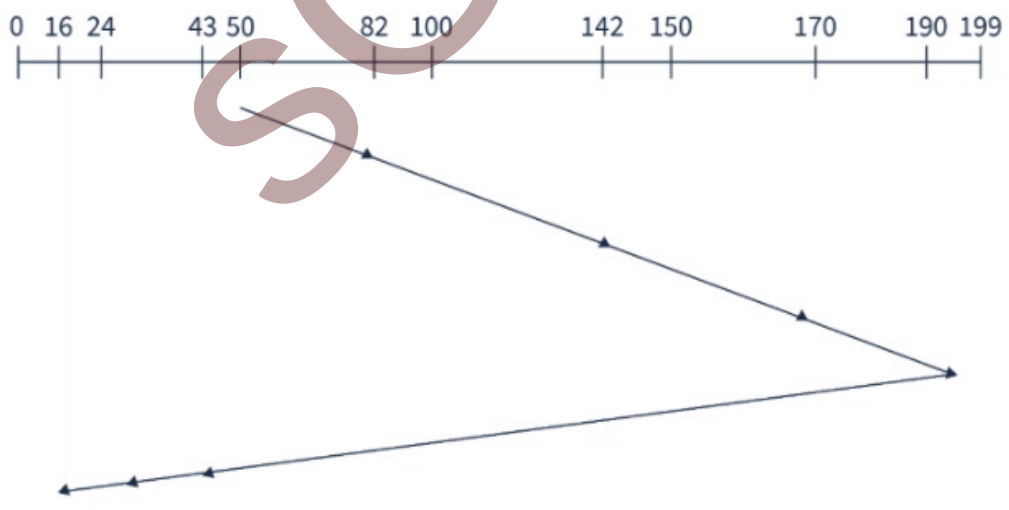


Fig 3.4.6 : LOOK disk scheduling Head Movement Diagram

The disk arm starts from position 50 and moves in one direction, servicing all the requests along the path. Instead of going to the end of the disk, it only goes up to the

SGOU

last request in that direction, which is 190. After reaching 190, the head does not move back step by step; instead, it directly jumps to the lowest request on the other side, which is 16. From there, it again continues servicing the remaining requests in the same direction.

Thus, the total seek time is calculated as the movement from 50 to 190 and then from 190 to 16. Therefore, the total seek time = (190 − 50) + (190 − 16) = 314.

Advantages

- Starvation does not occur, so every request gets a chance to be served.
- The head does not move to the end of the disk unnecessarily, so time is saved.
- This reduces total head movement and improves efficiency.
- It provides better performance compared to SCAN in many cases.

Disadvantages

- The disk arm has to identify the last request in each direction before moving.
- This makes the algorithm slightly more complex to implement.
- There may be a small delay in servicing some requests.

3.4.5.6 C-LOOK disk scheduling algorithm

The C-LOOK disk scheduling algorithm is an improved version of the LOOK algorithm that provides more uniform waiting time for disk requests. In this method, the disk head moves in only one direction and services all the requests along its path. Instead of going to the end of the disk, it only travels up to the last request in that direction. After reaching the final request, the head does not reverse its direction; instead, it directly jumps to the first request at the other end of the disk and continues servicing in the same direction. This circular movement helps in reducing unnecessary head movement and ensures that all requests are treated more fairly.

C-LOOK improves system performance by minimizing seek time and avoiding traversal to unused disk areas. It provides better response time compared to SCAN and LOOK, especially in systems with heavy disk load. Since the head always moves in a single direction, it offers a more uniform waiting time and reduces the chances of starvation. However, the jump from the last request to the first request may cause a slight delay, and the algorithm is a bit more complex to implement compared to simpler methods like FCFS.

Example: Suppose a disk having 200 tracks (0-199). The request seque nce(82,170,43,140,24,16,190) are shown in the given figure and the head position is at 50.

The disk arm starts from position 50 and moves in one direction, servicing all the

SGOU

requests along its path. Instead of going to the end of the disk, it goes only up to the last request in that direction, which is 190. After reaching 190, the head does not service requests while returning; instead, it directly jumps to the lowest request on the other end of the disk, which is 16. From there, it again starts moving in the same direction and continues servicing the remaining requests such as 43.

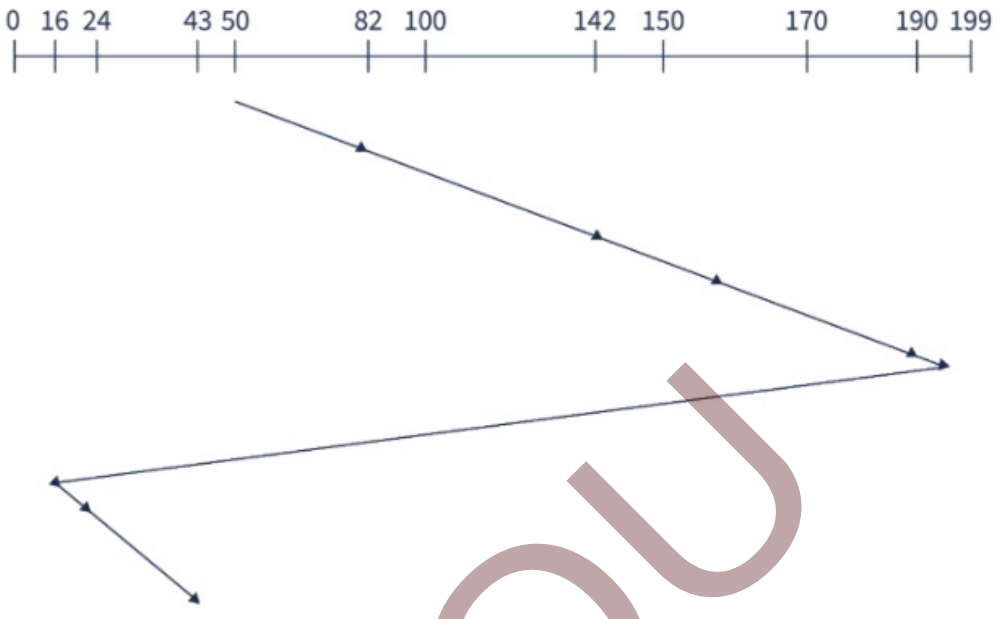


Fig 3.4.7 : C- LOOK disk scheduling Head Movement Diagram

Hence, the total seek time is calculated as the movement from 50 to 190, then from 190 to 16 (jump), and finally from 16 to 43.

Therefore, Seek Time = (190 − 50) + (190 − 16) + (43 − 16) = 341.

Advantages

- The waiting time is decreased, so requests are served faster.
- If there are no requests in the current direction, the head immediately reverses its direction.
- Starvation does not occur, and all requests get a chance to be served.
- The time taken by the disk arm to find the desired track is less, which improves performance.

Disadvantages

- The disk arm has to identify the last request in each direction before moving.
- This makes the algorithm slightly more complex to manage and implement.

3.4.6 Swap-Space Management

Swapping is a memory management technique used in multiprogramming systems

SGOU

to increase CPU utilization. In this method, a process is temporarily removed from the main memory (RAM) and stored in secondary memory (disk), and later brought back into RAM for continued execution. The process of moving a program from main memory to disk is called Swap Out, and bringing it back from disk to main memory is called Swap In. This technique helps the system run more processes even when physical memory is limited and ensures better utilization of system resources.

Swap space is a portion of secondary storage (usually a hard disk or SSD) used by the operating system to temporarily store data that cannot fit into main memory. It is an important part of virtual memory management, which allows the system to behave as if it has more memory than actually available. When RAM becomes full, the operating system moves inactive or less frequently used pages from RAM to swap space. This process is called paging out, and when those pages are required again, they are brought back into RAM, known as paging in. This process ensures smooth execution of programs without interruption.

Types of Swap Space

1. Swap Partition

- A dedicated partition on the disk reserved only for swap space.
- It provides faster access because it does not involve file system overhead.
- Commonly used in traditional Linux systems.
- More efficient but less flexible to resize.

2. Swap File

- A regular file in the file system used as swap space.
- Easier to create, resize, and manage.
- Slightly slower than swap partitions due to file system involvement.
- More flexible and commonly used in modern systems.

Working of Swap-Space Management

Swap-space management is used by the operating system to efficiently utilize memory and improve system performance. When the memory is full, the OS identifies less-used processes or pages and moves them to disk. This creates free space in RAM for active processes, which require faster access. The operating system uses page replacement algorithms (like FIFO, LRU) to decide which pages should be swapped out. This dynamic process allows the system to handle multiple applications efficiently.

Consider a computer system with 4 GB RAM. A user opens multiple applications such as a web browser, video player, and a programming IDE. As more applications are opened, the RAM becomes full and cannot hold all the running processes.

At this point, the operating system uses swap-space management. It identifies programs

SGOU

that are not currently active (for example, a minimized browser tab) and moves their data from RAM to the swap space on the disk. This process is called swap out. Now, free space is available in RAM for active programs to run smoothly.

When the user returns to the minimized browser, the operating system brings its data back from the disk to RAM. This process is called swap in. Thus, even with limited RAM, the system continues to run multiple applications efficiently, although accessing swapped data may be slightly slower.

Advantages

- Swap space increases the effective memory capacity of the system.
- It improves performance by freeing RAM for important and active processes.
- It provides flexibility by allowing dynamic memory allocation.
- It helps in running multiple applications simultaneously without crashing.
- It improves system stability during heavy workloads.

Disadvantages

- Accessing data from disk is much slower than RAM, which may reduce performance.
- Excessive swapping can lead to thrashing, where the system becomes very slow.
- It uses disk space, reducing storage available for other data.
- There is a risk of data loss if the swap space is corrupted.
- Frequent swapping may reduce the lifespan of SSDs due to continuous read/write operations.

SGOU

Summarized Overview

1. Disk Structure & Performance

- **Physical Components**:
  - **Platters**: Circular disks coated with magnetic material.
  - **Tracks**: Concentric circular paths on each platter.
  - **Sectors**: The smallest addressable unit on a track where data is stored.
  - **Cylinders**: A set of tracks at the same position across all platter surfaces.
  - **Heads**: The components that read and write data.
- **Performance Metrics**:
  - **Seek Time**: Time to move the head to the correct track.
  - **Rotational Latency**: Time for the sector to rotate under the head.
  - **Transfer Time**: Time to move data between disk and memory.

2. Disk Attachment Methods

- **Direct Attached Storage (DAS)**: Connected directly via cables (USB/SATA); fast but limited to one system.
- **Network Attached Storage (NAS)**: Centralized storage accessed via a network (Ethernet); great for file sharing.
- **Storage Area Network (SAN)**: High-speed, block-level storage network for large-scale data centers and servers.

3. Disk Scheduling Algorithms

These determine the order of I/O requests to minimize head movement (Seek Time).

Algorithm	Mechanism	Key Characteristics
FCFS	First-Come, First-Served	Simple and fair; high seek time (inefficient).
SSTF	Shortest Seek Time First	Services the closest request; risks **starvation**.
SCAN	Elevator Algorithm	Moves to the end of the disk, then reverses.

SGOU

C-SCAN	Circular SCAN	Moves in one direction, jumps back to start without servicing.
LOOK	"Looks" ahead	Like SCAN, but reverses at the **last request** (not the end).
C-LOOK	Circular LOOK	Like C-SCAN, but jumps from the **last to the first request**.

4. Swap-Space Management

- **Purpose**: Extends physical RAM by using secondary storage to hold inactive processes or pages.
- **Operations**:
  - **Swap Out**: Moving a process from RAM to disk.
  - **Swap In**: Bringing a process from disk back to RAM.
- **Implementation**:
  - **Swap Partition**: A dedicated, faster disk area with no file system overhead.
  - **Swap File**: A regular file within the file system; more flexible but slightly slower.
- **Risks**: Excessive swapping leads to thrashing, where the system spends more time moving data than executing processes.

SGOU

1. Describe the physical and logical organization of a hard disk drive.
2. Explain different types of disk attachment methods.
3. Describe FCFS and SSTF disk scheduling algorithms. Explain their working with advantages and disadvantages.

4. Explain the SCAN and C-SCAN disk scheduling algorithms. Compare their performance and efficiency.

5. Discuss the LOOK and C-LOOK disk scheduling algorithms. Explain their working with advantages and disadvantages.

6. Compare all disk scheduling algorithms (FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK). Highlight their differences and performance.

7. A disk has 200 tracks numbered from 0 to 199.

8. The request queue for I/O operations is given as:

   45, 78, 90, 10, 150, 38, 184. The initial position of the read/write head is at track 50. Draw the head movement diagram and calculate the total seek time using the following disk scheduling algorithms:

   - FCFS
   - SSTF
   - SCAN
   - C-SCAN
   - LOOK
   - C-LOOK

SGOU

Reference

1. https://www.scaler.com/topics/disk-scheduling-algorithms/

2. GeeksforGeeks. (n.d.). *Disk scheduling algorithms*. Retrieved from https://www.geeksforgeeks.org/

3. Arpaci-Dusseau, R. H., & Arpaci-Dusseau, A. C. (2018). *Operating systems: Three easy pieces*. Arpaci-Dusseau Books.

4. D. M. Dhamdhere. (2012). *Operating systems: A concept-based approach* (3rd ed.). McGraw-Hill Education.

Suggested Reading

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts* (10th ed.). Wiley.

2. Stallings, W. (2017). *Operating systems: Internals and design principles* (9th ed.). Pearson.

3. Tanenbaum, A. S., & Bos, H. (2014). *Modern operating systems* (4th ed.). Pearson.

4. Parker White. (2024). *Operating system*. Oryson Press.

Space for Learner Engagement for Objective Questions

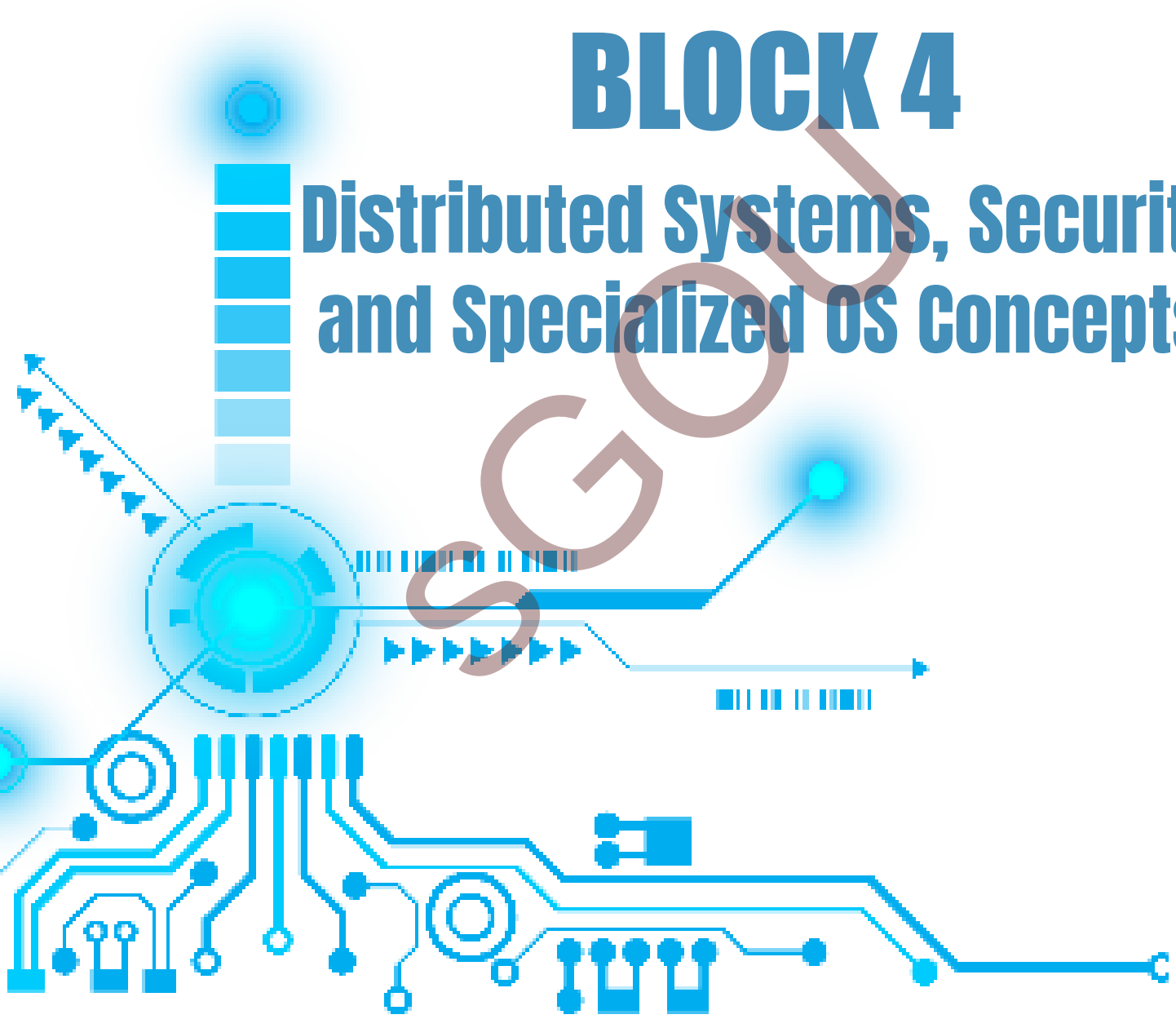
Learners are encouraged to develop objective questions based on the content in the paragraph as a sign of their comprehension of the content. The Learners may reflect on the recap bullets and relate their understanding with the narrative in order to frame objective questions from the given text. The University expects that 1 - 2 questions are developed for each paragraph. The space given below can be used for listing the questions.

SGOU

SGOU

BLOCK 4

Distributed Systems, Security and Specialized OS Concepts



SGOU

1 UNIT Distributed Operating Systems

Learning Outcomes

After completion of this unit, the learner will be able to:

- discuss the concept and features of distributed systems and distributed operating systems
- discuss different types of distributed operating systems and network topologies
- explain the major design issues in distributed operating systems
- describe the concept, components, and advantages of distributed file systems

Background

Distributed systems have become an essential part of modern computing environments. In earlier days, most computing tasks were performed using a single computer system. As the number of users, applications, and data increased, single systems became insufficient to handle large workloads efficiently. Organizations such as banks, universities, online shopping platforms, and cloud service providers required multiple computers to work together in order to provide faster, reliable, and continuous services.

This need led to the development of distributed systems, where multiple interconnected computers cooperate and share resources through communication networks. Distributed operating systems were introduced to manage these interconnected computers and make them function as a single unified system. Such systems support resource sharing, parallel processing, fault tolerance, scalability, and improved performance.

SGOU

The growth of computer networks, internet technologies, and cloud computing has further increased the importance of distributed systems. Today, distributed systems are widely used in cloud platforms, distributed databases, banking networks, online reservation systems, content delivery networks, and large-scale web services. Understanding distributed systems and distributed operating systems helps learners gain knowledge about modern computing infrastructures and the techniques used to manage communication, processes, data, synchronization, and distributed file systems effectively.

Keyconcepts

Client-server OS, Peer-to-peer OS, Middleware based OS, Cloud- based OS, Topology

Discussion

4.1.1 Introduction

Imagine a situation where a single computer is not powerful enough to handle all tasks efficiently. Large organizations, banks, online shopping platforms, universities, and cloud service providers often connect multiple computers together so that they can work as a single system. Such interconnected systems form the foundation of distributed systems and distributed operating systems.

Consider an online shopping website. When thousands of users place orders simultaneously, a single computer may become overloaded. To solve this problem, multiple computers share the workload. One computer may handle user login, another may manage payment processing, while another stores product information. Although many computers work behind the scenes, users experience the system as a single unified service.

This concept leads to the development of distributed operating systems. A distributed operating system manages multiple interconnected computers and allows them to work together efficiently. It provides resource sharing, communication, reliability, and improved performance.

In this chapter, learners will explore distributed systems, network structures, network topologies, design issues in distributed systems, and distributed file systems.

SGOU

4.1.2 Meaning of Distributed Systems

A distributed system is a collection of independent computers connected through a communication network that work together as a single system.

Each computer in the distributed system is called a node. These nodes communicate with each other and share resources such as files, printers, processors, databases, and applications.

Although the computers are physically separate, users often feel as if they are using one unified system.

Example

Consider a university campus network.

- One server stores student records.
- Another server manages library data.
- Another server handles online examinations.

All these computers communicate through a network and together provide services to students and staff.

Thus, the entire system behaves like one integrated computing environment.

Common examples include Cloud computing systems, Banking networks, Airline reservation systems, Distributed databases, Multiplayer online games, Content delivery networks, Google search systems etc.

4.1.3 Types of Distributed Operating Systems

Table. 4.1.1 Features of various Distributed Operating Systems

Type of Distributed Operating System	Description	Key Features	Example Applications	Examples
Client-Server Distributed OS	One or more servers provide services, and client systems request those services over a network.	Centralized management, resource sharing, easy maintenance	Web services, database systems, email systems	Windows Server, UNIX
Peer-to-Peer Distributed OS	All computers act as equals and can both request and provide services.	Decentralization, direct resource shar-ing, fault tolerance	File sharing, collaborative networks	Plan 9

SGOU

Middleware-Based Distributed OS	Middleware software connects distributed systems and enables communication between applications.	Platform independence, interoperability, scalability	Cloud computing, distributed applications	CORBA
Cloud-Based Distributed OS	Distributed resources are managed over cloud infrastructure to provide scalable services.	Virtualization, scalability, on-demand resources	Cloud platforms, online storage, SaaS	Google Cloud, Amazon Web Services

4.1.4 Network Structure and Topology in Distributed Systems

Network topology refers to the physical or logical arrangement of nodes and communication links in a distributed system. The topology influences how efficiently nodes exchange information and continue operation during failures. The common topologies in distributed systems are as follows.

1. Bus Topology

In a bus topology, all the computers or nodes in the network are connected using a single main communication cable known as the bus. This cable acts as a shared pathway through which data travels from one node to another. Whenever a node sends data, the signal is transmitted across the entire bus, so all connected nodes receive it. However, only the node for which the data is intended accepts and processes the message, while the others ignore it.

In distributed systems, bus topology provides a simple way for multiple nodes to communicate using a common communication channel. Since all devices share the same cable, the network structure becomes easy to set up and manage for small environments. It also allows devices to exchange data without requiring complex networking hardware. See Fig.4.1.1 for the pictorial representation of Bus topology.

One of the major advantages of bus topology is its simplicity. The installation process is easy because only a single main cable is needed to connect all the nodes. This reduces the amount of cabling required and lowers the implementation cost. Due to its simple design, bus topology became popular in early computer networks and small distributed systems.

Bus topology also has several limitations. Since all communication depends on the main cable, any damage or failure in that cable can stop communication across the entire network. Another problem is that network performance decreases when many nodes try to send data at the same time, leading to heavy traffic and delays. In addition, identifying faults in the network can be difficult because all devices are connected through the same communication line.

Because of these limitations, bus topology is generally suitable only for small distributed systems. As the number of connected nodes increases, reliability and performance issues

SGOU

become more significant, making the topology less effective for large-scale networks.

2. Star Topology

In a star topology, all the nodes in the network are connected to a central device such as a hub or switch. Instead of communicating directly with each other, every node sends and receives data through this central device. When one node wants to send information to another, the data first reaches the hub or switch, which then forwards it to the destination node.

In distributed systems, star topology is widely used because it makes communication management easier and more organized. The central hub controls the communication process, which helps in monitoring and managing the network efficiently. This topology is commonly seen in modern local area networks and client-server distributed systems. See Fig.4.1.1 for the pictorial representation of Star topology.

One important advantage of star topology is that fault detection becomes easy. Since each node has a separate connection to the central device, problems in one connection can be identified quickly. Another advantage is that failure of a single node does not affect the functioning of other nodes in the network. Star topology also provides better performance because data communication is managed through the central hub or switch, reducing communication conflicts.

However, star topology also has some limitations. The entire network depends heavily on the central hub or switch. If this central device fails, communication across the whole network stops. In addition, star topology requires more cabling compared to some other topologies because every node must be connected individually to the central device.

Due to its reliability, easy management, and better performance, star topology is widely used in modern client-server distributed systems and organizational networks.

3. Ring Topology

In a ring topology, each node in the network is connected to two neighboring nodes, forming a circular communication path. The data travels from one node to the next around the ring until it reaches the intended destination. Communication usually takes place in a single direction, although some advanced ring networks support communication in both directions to improve reliability.

In distributed systems, ring topology provides an organized method of communication because data follows a fixed path through the network. Each node participates in forwarding the data, which helps maintain orderly communication among all connected devices. See Fig.4.1.1 for the pictorial representation of Ring topology.

One of the main advantages of ring topology is the reduction of data collisions. Since data moves in an organized sequence around the ring, multiple nodes do not usually transmit at the same time, reducing communication conflicts. Another advantage is that all nodes get equal opportunities to communicate, making the network fair and balanced for data transmission.

In Ring topology, since every node forms part of the communication path, failure of

SGOU

a single node or a break in the connection can affect the entire network and interrupt communication. In addition, maintenance and troubleshooting can be difficult because identifying the exact point of failure in the ring may take time.

Due to these reliability and maintenance issues, ring topology is less commonly used in modern distributed operating systems and large distributed networks.

4. Mesh Topology

In a mesh topology, every node in the network is directly connected to every other node. This creates multiple communication paths between devices, allowing data to travel through different routes to reach its destination. Because many paths are available, communication can continue even if one connection fails.

In distributed systems, mesh topology is highly suitable for environments that require high reliability and strong fault tolerance. Since nodes are interconnected through several paths, the network can maintain communication without interruption even during failures. This makes the topology dependable for systems where continuous operation is very important.

One of the major advantages of mesh topology is its reliable communication. If one communication link becomes unavailable, data can be transmitted through an alternative path. This greatly improves fault tolerance and ensures network stability. The presence of multiple paths also reduces the chances of complete network failure.

However, mesh topology has certain limitations. Establishing direct connections between all nodes requires a large amount of cabling and networking hardware, making the implementation very expensive. In addition, installation, configuration, and maintenance become complex as the number of nodes increases. See Fig.4.1.1 for the pictorial representation of Mesh topology.

Due to its high reliability and fault tolerance, mesh topology is commonly used in critical distributed systems such as military communication networks, cloud infrastructures, and other systems where uninterrupted communication is essential.

5. Tree Topology

In a tree topology, the network structure combines the features of both bus and star topologies. The nodes are arranged in a hierarchical manner, similar to the branches of a tree. A main backbone cable connects different groups of nodes, and within each group, devices are usually connected through a central hub or switch. This creates multiple levels in the network structure.

In distributed systems, tree topology is useful for organizing large networks into different sections or levels. It allows departments, branches, or groups within an organization to communicate efficiently while still remaining connected to the main network. Because of its hierarchical arrangement, network management becomes more systematic and organized.

Tree topology provides good scalability because new nodes and groups can be added

SGOU

easily without disturbing the existing network structure. It also supports structured management, as different sections of the network can be monitored and controlled separately. Due to these features, tree topology is suitable for large organizations, educational institutions, and enterprise networks.

At the same time, tree topology has certain limitations. Since many nodes depend on the main backbone connections, failure in the backbone cable or higher-level connection can affect communication for a large part of the network. In addition, managing and maintaining the network may become complex as the hierarchy grows larger.

Because of its organized structure and scalability, tree topology is widely used in large distributed systems and organizational networks.

6. Hybrid Topology

In a hybrid topology, two or more different network topologies are combined within the same distributed system. This approach allows organizations to use the advantages of multiple topologies together according to their requirements. For example, a company may use star topology within individual departments while connecting different departments using a bus topology.

In distributed systems, hybrid topology provides flexibility in network design and helps organizations build networks that match their size, structure, and communication needs. Different parts of the network can use different topologies based on performance, reliability, or cost considerations.

Hybrid topology offers several advantages. It provides flexibility because network designers can combine suitable topologies for different sections of the system. The topology also supports improved scalability, making it easier to expand the network when new nodes or departments are added. In addition, combining multiple topologies can improve reliability and overall network performance.

At the same time, hybrid topology has certain limitations. Managing and maintaining a hybrid network can become complex because different topologies and devices must work together. The implementation cost is also higher due to the need for additional networking hardware and more complicated configurations.

Because of its flexibility, scalability, and reliability, hybrid topology is widely used in large distributed operating systems and modern organizational networks.

4.1.5 Design issues in Distributed Operating Systems

Designing a distributed operating system is a difficult task because many independent computers connected through a network must work together as a single unified system. These computers may differ in hardware, software, communication methods, and geographical location. Since users expect smooth and reliable services from distributed systems, several important design issues must be addressed while developing such systems.

SGOU

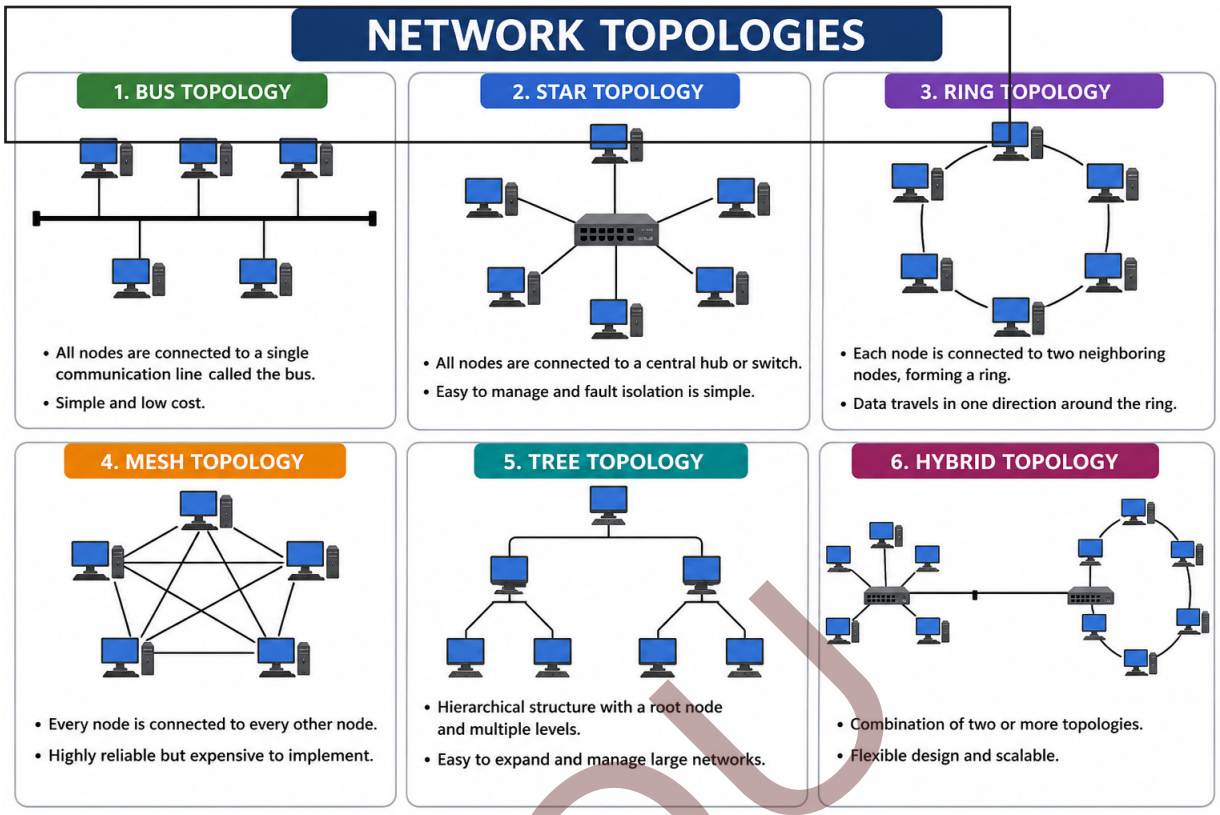


Fig. 4.1.1 Topologies in Distributed system

4.1.5.1 Communication issues

Communication is one of the major design issues in distributed operating systems because different components of the system are located on separate computers and connected through a network. These components must continuously exchange information to coordinate their activities and share resources properly. However, communication through networks is not always reliable. Problems such as network delays, message loss, and connection failures may affect system performance and efficiency.

One common communication method used in distributed systems is message passing. In this method, processes communicate with each other by sending and receiving messages through the network. Since processes are located on different nodes, message passing becomes essential for coordinating tasks and exchanging data.

Another important issue is communication latency and bandwidth. Latency refers to the time required for a message to travel from one node to another, while bandwidth refers to the amount of data that can be transmitted through the network within a given time. High latency and low bandwidth can reduce the speed and efficiency of the distributed system.

Communication protocols also play an important role in distributed operating systems. These protocols define the rules and procedures for exchanging data between nodes. Proper communication protocols help ensure reliable, accurate, and secure transmission of information across the distributed environment.

SGOU

4.1.5.2 Process Management

Process management is another important design issue in distributed operating systems. Since multiple computers work together in a distributed environment, processes may execute on different nodes simultaneously. The operating system must therefore manage the creation, execution, coordination, scheduling, and termination of processes efficiently across the entire system.

One major challenge in process management is process coordination. Processes executing on different nodes often need to communicate and synchronize with each other while performing tasks. Proper coordination is necessary to ensure correct execution and to avoid problems such as data inconsistency and resource conflicts.

Another important concept is process migration. In distributed systems, workload may not be equally distributed among all nodes. Some computers may become overloaded while others remain idle. To improve performance and balance the workload, the operating system may move a running process from one node to another. This technique is known as process migration. However, transferring a running process along with its execution state and resources is a complex task.

Thread management is also an important part of process management. A process may contain multiple threads that execute concurrently. The distributed operating system must manage the execution, scheduling, synchronization, and termination of these threads efficiently. Proper thread management helps improve system performance and supports parallel execution of tasks in distributed environments.

4.1.5.3 Data Management

Data management is another major design issue in distributed operating systems because data is stored and accessed from multiple nodes located at different places. The operating system must ensure that data is stored efficiently, accessed quickly, and maintained correctly throughout the distributed environment.

One important aspect of data management is data storage. In distributed systems, data is often distributed across multiple computers rather than stored in a single location. The operating system must manage this distributed storage efficiently to provide better scalability, faster access, and improved reliability. Data may also be stored redundantly on multiple nodes to prevent data loss during failures.

Data access is another important concern in distributed systems. Since many users and processes may access data simultaneously from different locations, efficient access mechanisms are necessary. The operating system must ensure that users can retrieve required information quickly even when the system contains very large amounts of data and numerous active users.

Consistency and replication also play an important role in distributed data management. Replication means maintaining multiple copies of the same data on different nodes. This improves fault tolerance and increases data availability because if one node fails, data can still be accessed from another node. However, maintaining consistency becomes

SGOU

challenging because all copies of the data must remain identical and updated whenever changes occur.

Another important issue is data integrity. Data integrity ensures that data remains accurate, complete, and consistent throughout its lifecycle. The operating system must prevent data corruption, unauthorized modification, and accidental loss of information. Proper integrity mechanisms help maintain the correctness and reliability of data in distributed environments.

4.1.5.4 Synchronization and Coordination

Synchronization and coordination are important design issues in distributed operating systems because multiple processes execute simultaneously on different nodes. These processes often share resources and exchange information while performing tasks. Therefore, proper coordination is necessary to ensure correct execution order and to avoid conflicts among processes.

One important aspect of synchronization is clock synchronization. Since each node in a distributed system maintains its own clock, differences in system time may occur between nodes. These time differences can create problems while coordinating tasks, recording events, or performing time-sensitive operations. Therefore, the distributed operating system must synchronize the clocks of all nodes to maintain a consistent system time across the network.

Leader election is another important coordination issue in distributed systems. In many situations, one node must act as a coordinator to manage shared resources and control system activities. The process of selecting such a coordinating node is called leader election. The selected leader helps maintain consistency and prevents conflicts among distributed processes. If the leader node fails, the system must automatically select another suitable node as the new leader.

Mutual exclusion is also necessary in distributed operating systems. Multiple processes may attempt to access the same shared resource simultaneously, which can lead to inconsistency and incorrect results. Mutual exclusion ensures that only one process can access a shared resource at a time while other processes wait for their turn. This helps prevent conflicts and maintains the correctness of operations in distributed environments.

4.1.5.5 Transparency

Transparency is an important design issue in distributed operating systems because users should be able to use the distributed system without experiencing its internal complexity. Although resources and processes may be distributed across multiple computers connected through a network, the entire system should appear as a single unified system to the users.

One important type of transparency is access transparency. In distributed systems, different nodes may use different methods for storing and representing data. Access transparency hides these differences and allows users to access resources in a uniform manner regardless of where the resources are located or how they are stored. As a

SGOU

result, users can interact with distributed resources just as they would interact with local resources.

Another important form is location transparency. In distributed operating systems, resources such as files, printers, and databases may be stored on different nodes across the network. Location transparency hides the physical location of these resources from users. Therefore, users can access resources without needing to know where they are actually stored.

Replication transparency is also important in distributed environments. To improve reliability and availability, distributed systems often maintain multiple copies of the same data on different nodes. Replication transparency hides the existence of these multiple copies from users. Users access the data as if only a single copy exists, while the operating system manages all replication activities internally.

Thus, transparency simplifies the use of distributed systems and improves user convenience by hiding the complexity of the distributed environment.

4.1.6 Distributed File System

A Distributed File System (DFS) is a file system that allows users to access and manage files stored on multiple computers through a network as if the files were stored on a single local system. In a distributed environment, data is not stored in one central location. Instead, files are distributed across several machines connected through a network. Even though the files are physically stored in different locations, users can access them in a simple and unified manner.

Distributed file systems improve data sharing, reliability, scalability, and resource utilization. Since files are stored on multiple nodes, the system can continue functioning even if one server fails. DFS also supports parallel access to files, which improves performance in large-scale systems with many users.

Another important advantage of DFS is that it hides the complexity of the underlying hardware and storage locations. Users need not know where a file is physically stored because the distributed file system manages file access automatically in the background. This makes data management easier and provides a seamless experience to users.

4.1.6.1 Components of DFS

A Distributed File System (DFS) allows users and applications to store and access files located on multiple computers through a network as if the files were stored on a local system. The architecture of a DFS consists of several important components that work together to provide efficient, reliable, and transparent file access.

1. Clients

Function

Clients are the end-user systems or applications that request access to files stored in the distributed file system.

SGOU

Role

Clients send read and write requests to the DFS. They interact with the system in such a way that accessing remote files appears similar to accessing local files. Clients depend on other DFS components for tasks such as locating files, managing communication, caching frequently used data, and maintaining security.

2. Metadata Server (NameNode)

Function

The Metadata Server acts as the control center or "brain" of the distributed file system.

Role

It manages the file system structure, including directories, file names, file attributes, permissions, and the locations where file data is stored. When a client requests a file, the metadata server provides information about the storage nodes that contain the required data blocks.

3. Storage Nodes (DataNodes)

Function

Storage Nodes are responsible for storing the actual file data.

Role

Files are divided into blocks and stored across multiple storage nodes in the network. These nodes handle data read and write operations requested by clients. Storage nodes also communicate with one another to create duplicate copies of data for reliability and fault tolerance.

4. Communication and Transport Infrastructure

Function

This component provides the networking support required for communication between all DFS components.

Role

It enables data transfer and message exchange among clients, metadata servers, and storage nodes. Technologies such as Remote Procedure Calls (RPCs) are commonly used to support communication across different machines in a transparent manner.

5. Management and Protocol Layer

Function

The Management and Protocol Layer controls the overall operation and coordination of the distributed file system.

SGOU

Role

It manages important system activities such as:

- **Consistency Management:** Ensures that all copies of replicated data remain updated and synchronized.
- **Caching:** Stores recently accessed data locally to improve performance and reduce network traffic.
- **Security and Authentication:** Protects data and allows access only to authorized users and applications.
- **System Coordination:** Maintains smooth interaction among different DFS components.

Summarized Overview

Distributed systems consist of multiple interconnected computers that work together as a single system to provide efficient, reliable, and scalable services. Distributed operating systems manage communication, resource sharing, process execution, data management, synchronization, and transparency among different nodes in the network. Various network topologies such as bus, star, ring, mesh, tree, and hybrid topologies are used to organize communication between nodes based on system requirements.

Designing distributed operating systems involves several challenges including communication management, process coordination, data consistency, synchronization, and transparency. Distributed File Systems (DFS) further enhance distributed environments by allowing files stored on multiple computers to be accessed as if they were located on a single local system. Components such as clients, metadata servers, storage nodes, communication infrastructure, and management protocols work together to provide reliable and efficient file access. Overall, distributed systems form the foundation of modern computing environments such as cloud computing, banking systems, online services, and large-scale networks.

SGOU

Assignments

1. Explain the concept of distributed systems and discuss the advantages of distributed operating systems.

2. Describe different network topologies used in distributed systems with their advantages and limitations.

3. Explain the major design issues in distributed operating systems such as communication, process management, synchronization, and transparency.

4. A university plans to connect different departments, laboratories, and libraries through a distributed system. Analyze which network topology would be most suitable for the university and justify your answer.

5. A cloud service provider stores copies of data on multiple servers located in different regions. Analyze how replication and transparency improve reliability and availability in distributed file systems.

Reference

1. Distributed Systems: Principles and Paradigms Tanenbaum, A. S., & Van Steen, M. (2017). *Distributed systems: Principles and paradigms* (3rd ed.). Pearson.

2. Distributed Operating Systems Tanenbaum, A. S. (2015). *Distributed operating systems*. Pearson Education.

3. Distributed Systems: Concepts and Design Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2012). *Distributed systems: Concepts and design* (5th ed.). Addison-Wesley.

SGOU

Suggested Reading

1. Computer Networks Tanenbaum, A. S., & Wetherall, D. J. (2013). Computer networks (5th ed.). Pearson.

2. Operating System Concepts Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating system concepts (10th ed.). Wiley.

3. Modern Operating Systems Tanenbaum, A. S., & Bos, H. (2014). Modern operating systems (4th ed.). Pearson.

Space for Learner Engagement for Objective Questions

Learners are encouraged to develop objective questions based on the content in the paragraph as a sign of their comprehension of the content. The Learners may reflect on the recap bullets and relate their understanding with the narrative in order to frame objective questions from the given text. The University expects that 1 - 2 questions are developed for each paragraph. The space given below can be used for listing the questions.

SGOU

SGOU

2 UNIT Operating System Security

Learning Outcomes

After completion of this unit, the learner will be able to:

- define the basic security goals of an operating system
- explain how system protection keeps data and resources safe
- list different security threats and violations in a system
- explain the basic ideas of encryption and authentication
- describe what a firewall is and how it protects a system

SGOU

Background

Operating System Security is an essential area that focuses on protecting system resources, data, and processes from unauthorized access and misuse. As computers and networks become more interconnected, the risk of attacks, data breaches, and system failures has increased. To address these challenges, operating systems are designed with specific security goals such as confidentiality, integrity, and availability. These goals ensure that information is kept private, remains accurate, and is accessible when needed. The concept of system protection further strengthens security by controlling access to files, memory, and other resources based on user permissions.

In addition to protection mechanisms, it is important to understand common security violations and threats such as malware, unauthorized access, and data theft. To safeguard information, basic techniques like encryption and authentication are used. Encryption converts data into a secure format, while authentication verifies the identity of users. Another important concept is the firewall, which acts as a barrier between a trusted system and external networks, preventing unauthorized access. Together, these concepts form the foundation of operating system security and help in building safe and reliable computing environments.

Keywords

Security Goals, System Protection, Security Threats, Encryption, Authentication, Firewall

Discussion

4.2.1 Introduction to Operating System Security

Operating System Security refers to the methods and techniques used to protect system resources such as data, programs, memory, and files from unauthorized access or misuse. It ensures that only authorized users can access and use system resources in a proper way. Security measures mainly protect the system from external threats, while protection focuses on controlling internal access within the system.

Protection is a mechanism that controls how programs, processes, and users access system resources. It is especially important in multiprogramming systems, where many users share common resources like files and directories safely. For example, the owner of a file can decide who can read, write, or modify it, and the operating system enforces these permissions.

Operating system security can be attacked in several ways such as authorization misuse, browsing, trap doors, invalid parameters, line tapping, electronic data capture, lost lines, improper access control, waste recovery, and rogue software. Common security threats also include Trojan horses, viruses, worms, and buffer overflows. To prevent such attacks, systems use tools like passwords, encryption, and access control mechanisms.

Need for Security

Security is needed to protect computer systems from unauthorized access, data theft, and system damage. As systems are connected through networks and used by multiple users, the risk of attacks increases. Without proper security, sensitive data can be stolen, modified, or destroyed.

Security also ensures safe sharing of data among users. For example, users may need to share files, but only with specific permissions. Security mechanisms like passwords and encryption help prevent unauthorized users from accessing the system. Therefore, security is essential to maintain trust, privacy, and proper system functioning.

Importance of OS Security

Operating System Security is important because it protects both the system and user data from internal and external threats. It ensures that resources are used only in an authorized manner and prevents misuse by users or malicious programs.

SGOU

It also helps in maintaining system stability and reliability by preventing attacks such as malware or unauthorized access. Proper security measures ensure smooth operation, protect sensitive information, and allow secure sharing of resources. Overall, OS security is essential for maintaining a safe and efficient computing environment.

4.2.2 Security Goals

Security goals define the basic objectives that an operating system must achieve to protect its data, resources, and services. These goals ensure that the system works safely, prevents unauthorized access, and allows only legitimate users to use resources properly. The main goal of a security system is to protect information and maintain system reliability.

The key goals of a security system include Confidentiality (Secrecy), Integrity, Availability, Authentication, and Authorization.

1. Confidentiality (Secrecy)

Confidentiality means that only authorized users are allowed to access system data and resources. Not everyone should have access to system files. This prevents sensitive information from being disclosed to unauthorized users. Techniques like passwords and encryption help maintain confidentiality.

2. Integrity

Integrity ensures that system data and resources are accurate and not altered by unauthorized users. Users with insufficient privileges should not modify important system files or data. It protects the system from accidental or malicious changes and maintains data correctness.

3. Availability

Availability ensures that system resources are accessible to all authorized users whenever needed. No single user or process should consume all resources, as this may lead to denial of service. Malware or attacks can limit system resources and prevent legitimate users from accessing them, which must be avoided.

4. Authentication and Authorization

Authentication is the process of verifying the identity of a user (for example, using passwords, OTPs, or biometrics). Authorization determines what actions a verified user is allowed to perform. Even if a user is authenticated, they can only access resources for which they have permission. These mechanisms help in controlling access and ensuring system security.

Overall, these security goals work together to create a safe and reliable operating system environment by preventing unauthorized access, protecting data, and ensuring proper resource usage.

4.2.3 System Protection

System protection refers to the mechanisms used by an operating system to control

sGOU

access to its resources such as files, memory, and CPU. It ensures that only authorized users and processes can access or modify system resources. Protection is especially important in multi-user systems where many users share the same resources. It helps maintain system stability, security, and proper functioning by preventing misuse or accidental damage.

4.2.3.1 Access Control

Access control is a security mechanism used in operating systems to regulate who can access system resources and how they can use them. It ensures that only authorized users or processes can access files, memory, programs, and other resources. The main goal of access control is to prevent unauthorized access and protect sensitive data from misuse.

Access control works by identifying users (through authentication) and then checking their permissions (authorization). It defines what actions a user can perform, such as read, write, or execute. There are different types of access control methods, such as Discretionary Access Control (DAC), where the resource owner decides access rights; Mandatory Access Control (MAC), where access is based on system policies; and Role-Based Access Control (RBAC), where permissions are assigned based on user roles. These methods help maintain security, privacy, and proper resource management in a computer system.

4.2.3.2 User Permissions

User permissions define the level of access that a user has over system resources such as files, programs, and directories. They specify what actions a user is allowed to perform, helping to protect data and prevent unauthorized use. By assigning appropriate permissions, the operating system ensures that users can only access or modify resources that they are authorized to handle.

The common types of permissions include read, write, and execute. *Read* allows a user to view the contents of a file, *write* allows modification or deletion, and *execute* allows running a program. Permissions are usually assigned based on user roles or ownership, ensuring proper control and security. This helps in maintaining data integrity, preventing accidental changes, and supporting safe sharing of resources among multiple users.

4.2.3.3 Protection Mechanisms

Protection mechanisms are techniques used by the operating system to safeguard important system resources such as memory, files, and CPU from unauthorized access or misuse. These mechanisms ensure that multiple users and processes can safely share system resources without interfering with each other. They help maintain system stability, security, and efficient operation.

Memory Protection ensures that one process cannot access or modify the memory space of another process. Each process is given its own memory area, and the operating system strictly controls access to prevent errors and security breaches.

SGOU

File Protection controls access to files by using permissions and ownership rules. Users are allowed to read, write, or execute files based on the permissions assigned to them, ensuring that sensitive data is not accessed or modified by unauthorized users.

CPU Protection ensures fair allocation of CPU time among processes. The operating system uses scheduling techniques to prevent any single process from monopolizing the CPU, ensuring that all processes get a chance to execute and the system remains responsive.

These protection mechanisms work together to provide a secure and well-managed computing environment.

4.2.3.4 Role-Based Access Control

Role-Based Access Control (RBAC) is a method of managing access to system resources based on the roles assigned to users. Instead of giving permissions to each individual user, permissions are assigned to roles such as administrator, manager, or user, and users are then assigned to these roles. This makes access control simpler and more organized.

In RBAC, each role has a predefined set of permissions. For example, an administrator may have full access to all system resources, while a regular user may only have limited access such as reading or editing certain files. This approach ensures that users only have access to what is necessary for their job, improving security and reducing the risk of misuse. RBAC also makes it easier to manage permissions in large systems, as changes can be made at the role level instead of updating each user individually.

4.2.4 Security Violations and Threats

Security violations and threats refer to actions or events that compromise the safety and proper functioning of a computer system. A security violation occurs when someone gains unauthorized access to system resources, while a threat is a potential danger that can exploit system weaknesses. These can affect data, programs, and system performance, leading to serious problems such as data loss, corruption, or system failure.

Threats may come from both internal sources (such as misuse by authorized users) and external sources (such as hackers or malware). Common examples include viruses, worms, Trojan horses, unauthorized access, and denial-of-service attacks. Understanding these violations and threats is important for designing effective security measures and protecting the system from potential attacks.

4.2.4.1 Types of Security Violations

Security violations occur when the rules of system security are broken, leading to unauthorized access or misuse of system resources. The main types of security violations are:

1. Breach of Confidentiality: This occurs when unauthorized users access private or sensitive information. It results in loss of data secrecy.

SGOU

2. Breach of Integrity: This happens when data is modified, deleted, or tampered with by unauthorized users, affecting its accuracy and reliability.

3. Breach of Availability: This occurs when system resources or services are not available to authorized users, often due to attacks like denial-of-service.

4. Theft of Service: Unauthorized use of system resources such as CPU time, storage, or network services without permission.

5. Elevation of Privilege: This happens when a user gains higher access rights than permitted, allowing them to perform restricted actions.

These violations can seriously affect system security, making it important to implement proper protection and security mechanisms.

4.2.4.2 Types of Threats

Threats are potential dangers that can harm a computer system, steal data, or disrupt normal operations. Many of these threats come in the form of **malware** (malicious software), which is designed to damage or gain unauthorized access to a system.

- **Malware:** A general term for harmful software that includes viruses, worms, Trojan horses, spyware, and ransomware. It can damage files, steal information, or slow down the system.
- **Viruses:** Programs that attach themselves to files or programs and spread when those files are shared. They can corrupt data and affect system performance.
- **Worms:** Self-replicating programs that spread automatically through networks without needing a host file. They can consume system resources and slow down networks.
- **Trojan Horses:** Malicious programs disguised as legitimate software. Once installed, they can steal data or give attackers control over the system.
- **Spyware:** Software that secretly monitors user activities and collects sensitive information like passwords or browsing habits.
- **Ransomware:** A type of malware that locks or encrypts data and demands payment to restore access.

Understanding these threats helps in taking proper security measures such as using antivirus software, firewalls, and safe browsing practices.

4.2.4.3 Internal vs External Threats

Security threats to a computer system can come from both inside and outside the organization. These are classified as internal threats and external threats.

Internal Threats

Internal threats originate from within the organization. These may be caused by

SGOU

employees, users, or anyone who has authorized access to the system. Sometimes, users may misuse their access intentionally or accidentally, leading to data loss or system damage.

Examples include:

- Misuse of privileges
- Accidental deletion of important files
- Sharing confidential data without permission

Internal threats are dangerous because insiders already have access to the system, making them harder to detect.

External Threats

External threats come from outside the organization, usually from hackers or malicious programs. These attackers try to gain unauthorized access to the system through the internet or networks.

Examples include:

- Hacking attempts
- Malware attacks (viruses, worms, ransomware)
- Phishing and data theft

External threats are more common and often target system vulnerabilities to break into the system.

Both internal and external threats must be managed properly using security measures like authentication, access control, encryption, and firewalls to ensure system safety.

4.2.4.4 Methods of Attack

Methods of attack are the different techniques used by attackers to gain unauthorized access, steal data, or damage a computer system. Understanding these methods helps in designing better security measures.

- **Phishing:** Attackers trick users into revealing sensitive information (like passwords or bank details) by sending fake emails or messages that appear legitimate.
- **Password Attacks:** Hackers try to guess or crack passwords using techniques like brute force (trying many combinations) or dictionary attacks.
- **Malware Attacks:** Malicious software such as viruses, worms, and ransomware is used to damage systems, steal data, or control devices.
- **Man-in-the-Middle (MitM) Attack:** An attacker secretly intercepts communication between two parties to steal or alter data.

SGOU

- **Denial-of-Service (DoS) Attack:** The attacker floods a system or network with excessive requests, making it unavailable to legitimate users.
- **SQL Injection:** Attackers insert malicious code into database queries to access or manipulate data.
- **Social Engineering:** Attackers manipulate people into giving confidential information by exploiting trust rather than technical weaknesses.
- **Eavesdropping (Sniffing):** Unauthorized interception of data as it travels over a network.

These attack methods highlight the importance of strong security practices such as using secure passwords, encryption, firewalls, and user awareness.

4.2.5 Basic Concepts of Encryption

Encryption is a technique used to protect data by converting it into a secret form that cannot be easily understood by unauthorized users. It plays an important role in maintaining confidentiality and security of information in computer systems and networks. When data is encrypted, even if it is intercepted, it cannot be read without the proper key.

Encryption is widely used in areas such as online banking, email communication, and data storage to ensure that sensitive information remains safe. It works using algorithms and keys, where the original data is transformed into an unreadable format and can only be converted back by authorized users.

Definition of Encryption

Encryption is the process of converting plain, readable data (plaintext) into an unreadable format (ciphertext) using an algorithm and a key. Only users who have the correct key can decrypt the data and convert it back into its original form.

In simple terms, encryption ensures that information remains secure and accessible only to authorized users.

4.2.5.1 Types of Encryption

Encryption can be broadly classified into two main types based on how keys are used: symmetric encryption and asymmetric encryption.

1. Symmetric Encryption

In symmetric encryption, the same key is used for both encryption and decryption. The sender and receiver must share this key securely before communication.

- **Features:**
  - Faster and efficient
  - Suitable for large amounts of data

SGOU

- Requires secure key sharing

- **Example:** AES, DES
- **Limitation:** If the key is exposed, the data can be easily decrypted by unauthorized users.

2. Asymmetric Encryption

In asymmetric encryption, two different keys are used: a public key and a private key.

The public key is used to encrypt data, while the private key is used to decrypt it.

- **Features:**
  - More secure than symmetric encryption
  - No need to share private key
  - Used in secure communication (e.g., online transactions)
- **Example:** RSA
- **Limitation:** Slower compared to symmetric encryption.

Both types of encryption are often used together to provide strong security in modern systems.

4.2.5.2 Encryption Process

The encryption process is the method of converting readable data into a secure, unreadable form to protect it from unauthorized access. It ensures that even if data is intercepted, it cannot be understood without the correct key.

The process begins with plaintext, which is the original readable data. This data is passed through an encryption algorithm along with a key, which is a secret value used to transform the data. The output of this process is called ciphertext, which appears as random or meaningless data. When the intended receiver wants to read the data, they use a decryption algorithm and the correct key to convert the ciphertext back into plaintext.

In simple terms, the encryption process can be understood as:

Plaintext → Encryption Algorithm + Key → Ciphertext → Decryption → Original Data

This process is widely used to secure sensitive information such as passwords, emails, and online transactions, ensuring data confidentiality and protection.

4.2.5.3 Uses of Encryption

Encryption is widely used to protect data and ensure secure communication in computer systems and networks. It helps maintain confidentiality and prevents unauthorized access to sensitive information.

SGOU

- **Data Protection:** Encryption secures important data stored in computers, databases, and storage devices so that only authorized users can access it.
- **Secure Communication:** It protects data during transmission over networks, such as emails, messaging, and online transactions, preventing interception by attackers.
- **Password Security:** Passwords are stored in encrypted form, making it difficult for attackers to read or misuse them.
- **Online Banking and E-commerce:** Encryption ensures safe financial transactions by protecting user details like account numbers and payment information.
- **Privacy Protection:** It helps maintain user privacy by keeping personal and sensitive information confidential.
- **Authentication Support:** Encryption is used in verifying user identity through secure login systems and digital signatures.

Overall, encryption plays a vital role in maintaining data security, privacy, and trust in digital systems.

4.2.6 Authentication Concepts

Authentication is a security process used to verify the identity of a user, system, or process before allowing access to resources. It ensures that only legitimate users can enter the system and use its services. Authentication is an important part of operating system security, as it prevents unauthorized users from accessing sensitive data.

Common authentication methods include passwords, PINs, biometrics (fingerprint, face recognition), and one-time passwords (OTP). Strong authentication mechanisms help protect systems from misuse and improve overall security.

Definition of Authentication

Authentication is the process of confirming the identity of a user or system before granting access. It checks whether the user is genuine by verifying credentials such as a username and password, biometric data, or security tokens.

In simple terms, authentication answers the question: "Who are you?" and ensures that only authorized users can access the system.

4.2.6.1 Types of Authentication

Authentication methods are used to verify the identity of a user before granting access to a system. The common types include:

1. Password Authentication

This is the most common method where a user enters a username and password to log in.

SGOU

- Simple and easy to use
- Can be weak if passwords are easy to guess or shared
- Requires strong password practices for better security

2. Biometric Authentication

This method uses physical or behavioral characteristics of a person for verification.

- Examples: fingerprint, face recognition, iris scan
- More secure because it is unique to each individual
- Cannot be easily shared or forgotten

3. One-Time Password (OTP)

OTP is a temporary password sent to the user's mobile or email for verification.

- Valid only for a short time
- Provides an extra layer of security
- Commonly used in banking and online transactions

These authentication methods help ensure that only authorized users can access system resources and improve overall system security.

4.2.6.2 Authentication vs Authorization

Authentication and authorization are two important security concepts in an operating system. Though they are related, they serve different purposes.

Authentication

Authentication is the process of verifying the identity of a user. It checks whether the user is genuine before allowing access to the system.

- Answers the question: "Who are you?"
- Examples: password, OTP, fingerprint, face recognition
- Occurs first in the security process

Authorization

Authorization is the process of deciding what a user is allowed to do after they are authenticated.

- Answers the question: "What can you do?"
- Defines permissions like read, write, execute
- Occurs after authentication

SGOU

Table 4.1.1 Difference Between Authentication and Authorization

Authentication	Authorization
Verifies user identity	Grants access rights
Comes first	Comes after authentication
Uses credentials (password, biometrics)	Uses permissions and roles
Example: Login process	Example: Accessing files

In simple terms, authentication confirms who you are, and authorization determines what you can access.

4.2.6.3 Multi-Factor Authentication

Multi-Factor Authentication (MFA) is a security method that requires users to provide two or more verification factors to access a system. Instead of relying only on a password, MFA adds extra layers of security, making it harder for unauthorized users to gain access.

MFA combines different types of authentication factors, such as:

- **Something you know** – Password or PIN
- **Something you have** – OTP sent to mobile, smart card
- **Something you are** – Biometric data like fingerprint or face recognition

For example, when logging into an online account, a user may enter a password and then receive an OTP on their phone. Even if the password is stolen, the attacker cannot access the account without the second factor.

MFA greatly improves security by reducing the risk of unauthorized access and protecting sensitive data from attacks.

4.2.7 Firewall Concepts

A firewall is an important security mechanism used to protect a computer system or network from unauthorized access. It acts as a barrier between a trusted internal network and untrusted external networks, such as the internet. Firewalls monitor and control incoming and outgoing network traffic based on predefined security rules.

Firewalls help prevent attacks, block malicious traffic, and allow only safe and authorized communication. They can be implemented as hardware, software, or a combination of both, and are widely used in personal computers, organizations, and networks to enhance security.

Definition of Firewall

A firewall is a network security system that filters and controls data traffic between

SGOU

different networks to prevent unauthorized access and protect system resources.

In simple terms, a firewall acts like a security gate that allows safe data to pass through while blocking harmful or suspicious data.

4.2.7.1 Types of Firewalls

Firewalls are mainly classified into hardware firewalls and software firewalls based on how they are implemented. Both types are used to protect systems and networks from unauthorized access and cyber threats.

1. Hardware Firewall

A hardware firewall is a physical device that is placed between a computer network and the internet. It filters network traffic before it reaches the internal system.

- Installed in routers or separate security devices
- Protects an entire network
- More secure for organizations and large networks
- Works independently of the computer system

Example: Firewall built into a network router

2. Software Firewall

A software firewall is a program installed on a computer or server that monitors and controls network traffic.

- Installed on individual devices
- Protects a single system
- Easy to configure and manage
- Uses system resources like memory and CPU

Example: Windows Defender Firewall

Both hardware and software firewalls are important for providing strong security. Hardware firewalls protect the network as a whole, while software firewalls provide protection at the individual device level.

4.2.7.2 Working of Firewall

A firewall works as a security system that monitors, filters, and controls network traffic between a trusted internal network and an untrusted external network (such as the internet). Its main role is to allow safe communication while blocking harmful or unauthorized access.

The firewall works based on a set of predefined security rules. When data packets try to enter or leave a system, the firewall checks them against these rules. If the data is safe

SGOU

and authorized, it is allowed to pass. If it is suspicious or unauthorized, it is blocked or dropped.

Firewalls inspect different details of network traffic such as source address, destination address, port numbers, and type of data. They can also maintain logs of traffic for monitoring and analysis. Advanced firewalls may also detect malicious behavior and prevent attacks like hacking, malware transmission, and denial-of-service attacks.

In simple terms, a firewall acts like a security guard that controls who can enter or leave a network, ensuring system safety and preventing cyber threats.

4.2.7.3 Advantages and Limitations

Advantages of Firewall

- **Prevents unauthorized access:** Firewalls block hackers and unauthorized users from entering the system or network.
- **Improves network security:** It filters incoming and outgoing traffic, protecting against malware and cyber attacks.
- **Monitors network traffic:** Firewalls track and control data flow, helping in detecting suspicious activities.
- **Protects sensitive data:** It safeguards important information from being stolen or misused.
- **Supports safe internet usage:** Allows only trusted and safe communication between networks.

Limitations of Firewall

- **Cannot stop all attacks:** Firewalls cannot fully protect against internal threats or advanced attacks like social engineering.
- **Does not protect against viruses already inside the system:** If malware enters through other ways (like USB or downloads), firewall may not detect it.
- **Requires proper configuration:** Incorrect settings can reduce security or block legitimate traffic.
- **Cannot protect against user mistakes:** If users share passwords or download unsafe files, firewall cannot prevent it.

In summary, firewalls are an important security tool, but they must be used along with other security measures like antivirus software, encryption, and user awareness.

SGOU

Summarized Overview

RTOS is a specialized operating system designed for time-critical applications where tasks must meet strict deadlines. It ensures deterministic behavior, low latency, and high reliability. RTOS supports scheduling techniques like Rate Monotonic Scheduling (RMS) and Earliest Deadline First (EDF). It also provides efficient resource management, synchronization, and deadlock handling. RTOS is widely used in embedded systems, industrial automation, medical devices, automotive systems, and telecommunications.

Assignments

1. Define RTOS and explain its characteristics.
2. Differentiate between hard and soft real-time systems.
3. Explain Rate Monotonic Scheduling (RMS) with example.
4. Describe Earliest Deadline First (EDF) scheduling.
5. Explain resource management in RTOS.
6. Discuss synchronization techniques used in RTOS.
7. Explain applications of RTOS with examples.

SGOU

Reference

1. Qing, L., & Yifeng, Z. (2012). *Embedded real-time operating systems*. Springer.
2. Wang, K. C. (2017). *Embedded and real-time operating systems*. Springer.
3. Vahid, F., & Givargis, T. (2010). *Embedded system design: A unified hardware/software introduction*. Wiley.
4. Noergaard, T. (2005). *Embedded systems architecture: A comprehensive guide for engineers and programmers*. Elsevier.

Suggested Reading

1. Laplante, P. A. (2017). *Real-time systems design and analysis*. Wiley.
2. Liu, J. W. S. (2000). *Real-time systems*. Prentice Hall.
3. Buttazzo, G. C. (2011). *Hard real-time computing systems*. Springer.
4. Krishna, C. M., & Shin, K. G. (1997). *Real-time systems*. McGraw-Hill.

Space for Learner Engagement for Objective Questions

Learners are encouraged to develop objective questions based on the content in the paragraph as a sign of their comprehension of the content. The Learners may reflect on the recap bullets and relate their understanding with the narrative in order to frame objective questions from the given text. The University expects that 1 - 2 questions are developed for each paragraph. The space given below can be used for listing the questions.

SGOU

SGOU

3 UNIT Real-Time Operating System (Rtos)

Learning Outcomes

After completion of this unit, the learner will be able to:

- understand the concept and characteristics of RTOS
- differentiate between hard and soft real-time systems
- explain scheduling techniques such as RMS and EDF
- analyze resource management and applications of RTOS

Background

With the advancement of embedded and time-critical systems, traditional operating systems were not sufficient to handle strict timing requirements. This led to the development of Real-Time Operating Systems (RTOS), which focus on predictable and timely execution of tasks. RTOS is widely used in applications where delays are unacceptable, such as medical devices, automotive systems, and industrial automation. Its design emphasizes reliability, low latency, and efficient resource management.

Keywords

RTOS, Deterministic Behavior, Latency, Scheduling, RMS, EDF, Priority Scheduling, Synchronization

SGOU

Discussion

4.3.1 Introduction

A Real-Time Operating System (RTOS) is a specialized operating system designed to manage hardware resources and execute tasks within strict timing constraints. Unlike general-purpose operating systems (such as Windows or Linux), where performance is measured mainly by throughput and user convenience, an RTOS focuses on timeliness and predictability of task execution.

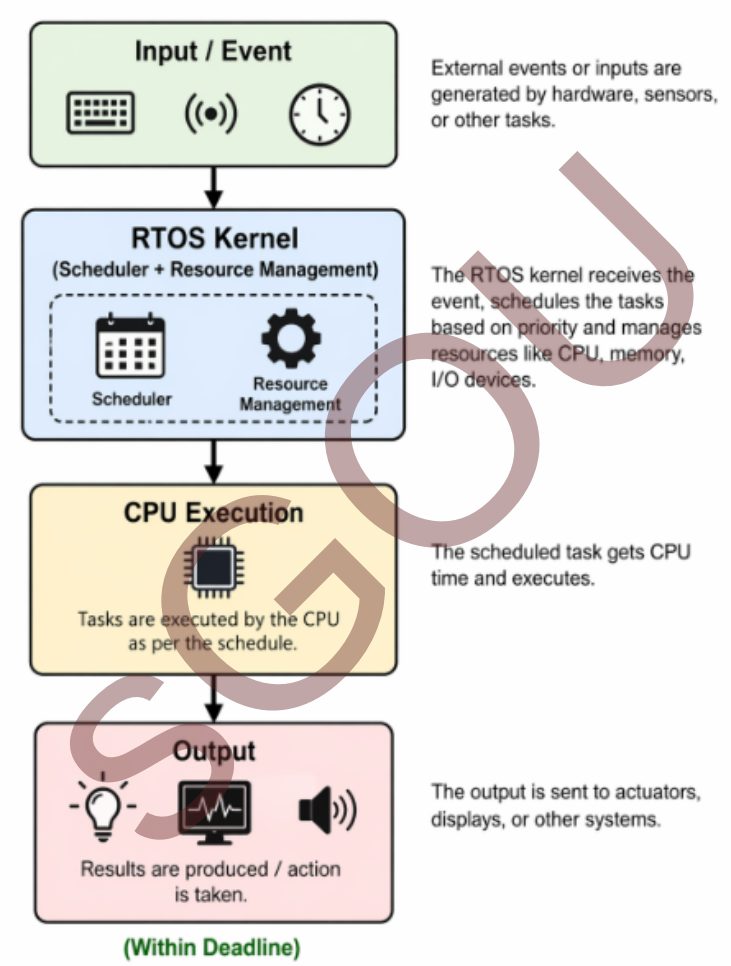


Fig 4.3.1 Basic Working of RTOS

In an RTOS, the correctness of a system is determined by two important factors:

1. **Logical correctness** – The output of the system must be correct.
2. **Time correctness** – The output must be produced within a specified deadline.

This means that even if the system produces the correct result, it is considered incorrect if the result is delayed. Hence, timing is as important as computation in real-time systems.

SGOU

RTOS is widely used in environments where delay cannot be tolerated, such as embedded systems, industrial control systems, medical devices, and aerospace applications. These systems require immediate and predictable responses to external events. To achieve this, RTOS efficiently manages CPU scheduling, memory allocation, task prioritization, and resource sharing, ensuring that all critical tasks meet their deadlines.

4.3.1.1 Key Characteristics of RTOS

1. Deterministic Behavior (Predictable Response Time)

Deterministic behavior means that the system responds to inputs in a predictable and fixed amount of time. Every operation in an RTOS has a known worst-case execution time (WCET).

- The same input will always produce the same output within the same time limit
- No unexpected delays
- Essential for safety-critical systems

Example: In an airbag system, the airbag must deploy within milliseconds during a collision, no variation is allowed.

2. Minimal Latency

Latency refers to the delay between an event occurrence and the system response.

RTOS minimizes two types of latency:

- **Interrupt latency** – Time taken to respond to an interrupt
- **Scheduling latency** – Time taken to start executing a task

Low latency ensures:

- Faster reaction to real-world events
- Immediate processing of high-priority tasks

Example: In a heart monitor, delay in detecting abnormal signals can be dangerous.

3. High Reliability

RTOS systems are designed to operate continuously without failure, especially in critical environments.

- Fault-tolerant design
- Stable under heavy load
- Ensures consistent performance

Example: Nuclear power plant systems must run continuously without crashes.

SGOU

4. Fast Context Switching

Context switching is the process of saving the state of one task and switching to another.

In RTOS:

- Context switching is very fast
- Enables efficient multitasking
- Ensures high-priority tasks get CPU immediately

Example: If a high-priority emergency task arrives, the system quickly switches from a low-priority task.

5. Efficient Interrupt Handling

Interrupts are signals generated by hardware or software to indicate an event.

RTOS handles interrupts efficiently by:

- Giving priority to critical interrupts
- Quickly transferring control to the required task
- Reducing delay in response

Example: In an automotive braking system, an interrupt from the brake sensor must be handled instantly.

A Real-Time Operating System ensures that tasks are executed correctly and within deadlines, making it essential for time-critical applications. Its key features such as deterministic behavior, low latency, high reliability, fast context switching, and efficient interrupt handling enable it to provide predictable and dependable system performance.

Table 4.3.1 Features of RTOS

Feature	Description	Key Points	Example	Importance
Deterministic Behavior	Produces the same output for a given input within a fixed time.	Predictable execution, no random delays, time-bound tasks	Airbag system deploying within fixed milliseconds after crash detection	Ensures timing accuracy and system predictability
Low Latency	Provides minimal delay between event occurrence and response.	Fast interrupt handling, quick response, reduced delay	Robot detecting obstacle and stopping immediately	Important for real-time control and safety systems

SGOU

Multitasking	Allows multiple tasks to execute concurrently.	Task scheduling, priority-based execution, efficient CPU use	Smartphone handling display, touch input, and network simultaneously	Supports complex applications and improves performance
Reliability	Ensures the system works correctly without failure over time.	Fault tolerance, error handling, system stability	Medical life-support system operating continuously without failure	Critical for mission-critical and safety systems
Fast Context Switching	Quickly switches CPU from one task to another.	Saves/restores task state, minimal switching time	Interrupt-driven system switching to emergency task instantly	Improves responsiveness and meets deadlines

4.3.2 Hard vs. Soft Real-time systems

4.3.2.1 Hard Real-Time Systems

A Hard Real-Time System is a system in which all tasks must be completed within strict deadlines. In such systems, time is the most critical factor, and missing even a single deadline is considered a complete system failure.

These systems are typically used in safety-critical applications, where a delay can lead to serious damage, loss of life, or system breakdown.

1. Tasks are Time-Critical

In hard real-time systems, every task has a fixed deadline, and execution must be completed within that exact time.

- Tasks are designed with strict timing constraints
- Each task has a defined Worst Case Execution Time (WCET)
- No flexibility in execution time

Example: In an aircraft system, control signals must be processed instantly to maintain stability.

2. Missing a Deadline (System Failure)

Unlike other systems, failure is not acceptable.

- Even a small delay leads to failure
- System may crash or cause dangerous outcomes
- No recovery mechanism for missed deadlines

Example: If a nuclear reactor control system fails to respond on time, it may cause a

SGOU

disaster.

3. No Tolerance for Delay

Hard RTOS systems do not allow any delay.

- Tasks must finish before deadline
- No buffering or postponing
- Strict time enforcement

This ensures maximum safety and reliability.

4. Requires Highly Predictable Scheduling

Scheduling must be deterministic and predictable.

- Uses algorithms like:
  - Rate Monotonic Scheduling (RMS)
  - Earliest Deadline First (EDF)
- System behavior must be known in advance

Predictability ensures that deadlines are always met.

Examples of Hard Real-Time Systems

- Aircraft flight control systems
- Nuclear reactor control
- Medical life-support systems
- Airbag control systems

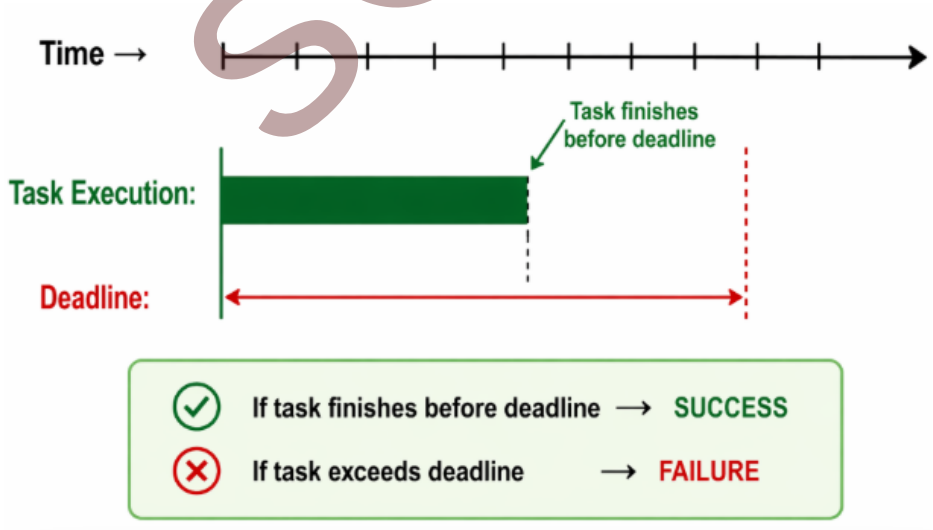


Fig 4.3.2 Hard Real-Time System

SGOU

The fig 4.3.2 shows a Hard Real-Time System where a task must complete execution before the specified deadline. The task execution bar finishes within the deadline limit, indicating successful execution. In such systems, timing is strictly enforced, if the task exceeds the deadline, it results in system failure. Hence, the figure highlights that meeting deadlines is mandatory and there is no tolerance for delay.

4.3.2.2 Soft Real-Time Systems

A Soft Real-Time System is a system where deadlines are important but not strictly enforced. Missing a deadline does not cause system failure, but it may reduce the system's performance or quality.

These systems are used in applications where user experience matters more than strict timing.

1. Deadlines are Important but Not Strict

- Tasks have deadlines, but they are flexible
- Late execution is acceptable occasionally
- System still produces useful output

Example: A video may buffer but continues playing.

2. Occasional Delay is Acceptable

- System tolerates small delays
- No catastrophic failure
- Performance degradation may occur

Example: Lag in online gaming affects experience but not system operation.

3. System Continues Functioning

- System does not stop even if deadlines are missed
- Maintains continuous operation
- Output quality may reduce

Example: Audio/video synchronization may be slightly off.

4. Focus on Performance Rather than Strict Timing

- Optimized for throughput and responsiveness
- Not strictly deterministic
- Best-effort scheduling

Examples of Soft Real-Time Systems

SGOU

- Video streaming platforms
- Online gaming systems
- Multimedia applications
- Web servers

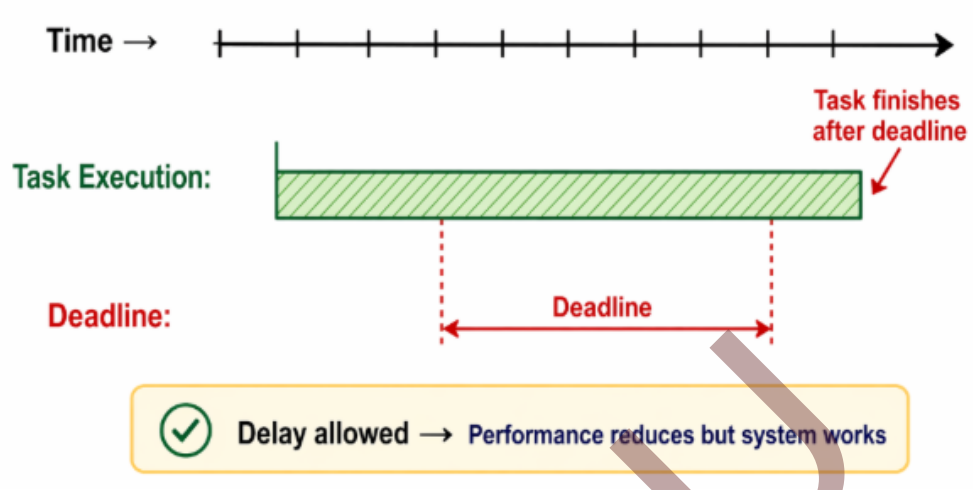


Fig 4.3.3 Soft Real-Time System

The fig 4.3.3 shows that a task may continue execution even after the deadline. The task execution bar extends beyond the deadline line, indicating a delay. Unlike hard real-time systems, this delay does not cause system failure. Instead, the system continues to work, but the performance or quality may decrease. This illustrates that soft real-time systems allow flexibility in timing with acceptable delays.

Table 4.3.2 Comparison: Hard Vs Soft Rtos

Feature	Hard RTOS	Soft RTOS
Deadline	Strict and must be met	Flexible and can be missed
Failure	Missing deadline = system failure	Performance degradation only
Tolerance	No delay allowed	Some delay acceptable
Predictability	Highly predictable	Less predictable
Scheduling	Deterministic	Best-effort
Usage	Critical systems	Multimedia systems

4.3.3 Scheduling In Rtos

Scheduling in a Real-Time Operating System (RTOS) is the process of selecting which task should execute at a given moment so that all tasks meet their deadlines. Unlike

SGOU

general-purpose systems, RTOS focuses on timeliness, predictability, and fast response. The scheduler ensures that high-priority and time-critical tasks are executed without delay. If a task misses its deadline, the system performance may degrade or even fail in critical applications.

4.3.3.1 Rate Monotonic Scheduling (Rms)

Rate Monotonic Scheduling (RMS) is a fixed-priority scheduling algorithm where priorities are assigned based on the task period (frequency). Tasks with shorter periods are given higher priority, and these priorities remain unchanged throughout execution. This makes RMS simple and predictable.

In RMS, the system mainly handles periodic tasks. The CPU always executes the highest-priority task that is ready. If a higher-priority task becomes ready while a lower-priority task is running, the lower-priority task is interrupted and the higher-priority task is executed immediately.

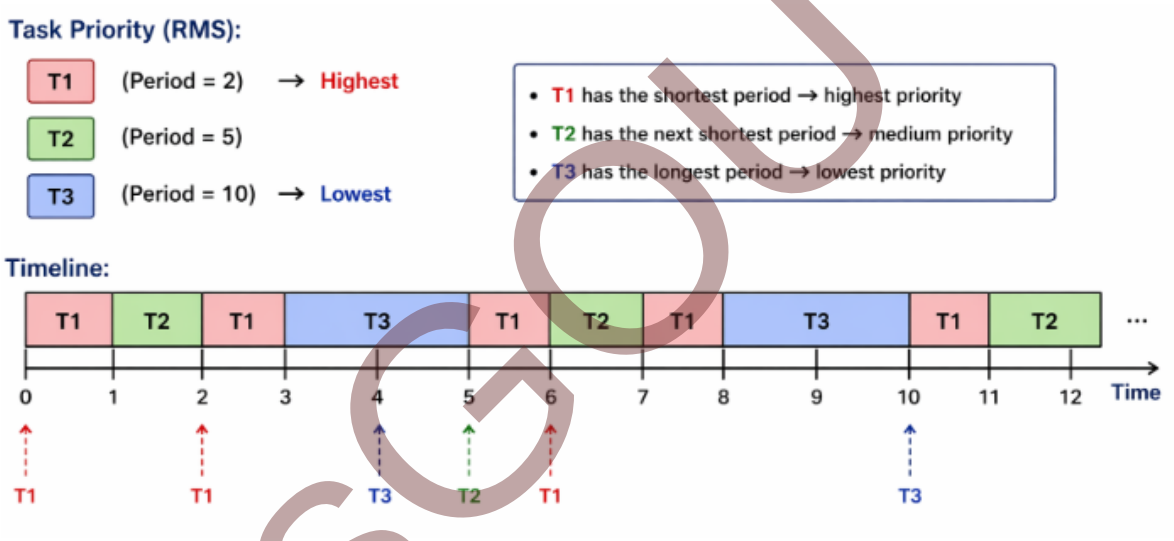


Fig 4.3 4 RMS Scheduling

The fig 4.3.4 shows that T1 executes most frequently because it has the shortest period. Whenever T1 becomes ready, it preempts other tasks. T2 and T3 are executed only when T1 is not ready. This ensures that high-frequency tasks meet their deadlines, but lower-priority tasks may experience delays.

4.3.3.2 Earliest Deadline First (Edf)

Earliest Deadline First (EDF) is a **dynamic scheduling algorithm** where priorities are assigned based on deadlines. The task with the nearest deadline is given the highest priority. Unlike RMS, priorities in EDF change continuously during execution.

The scheduler checks all ready tasks and selects the one with the earliest deadline. After execution or arrival of a new task, priorities are recalculated. This allows EDF to adapt to changing workloads and achieve better CPU utilization.

SGOU

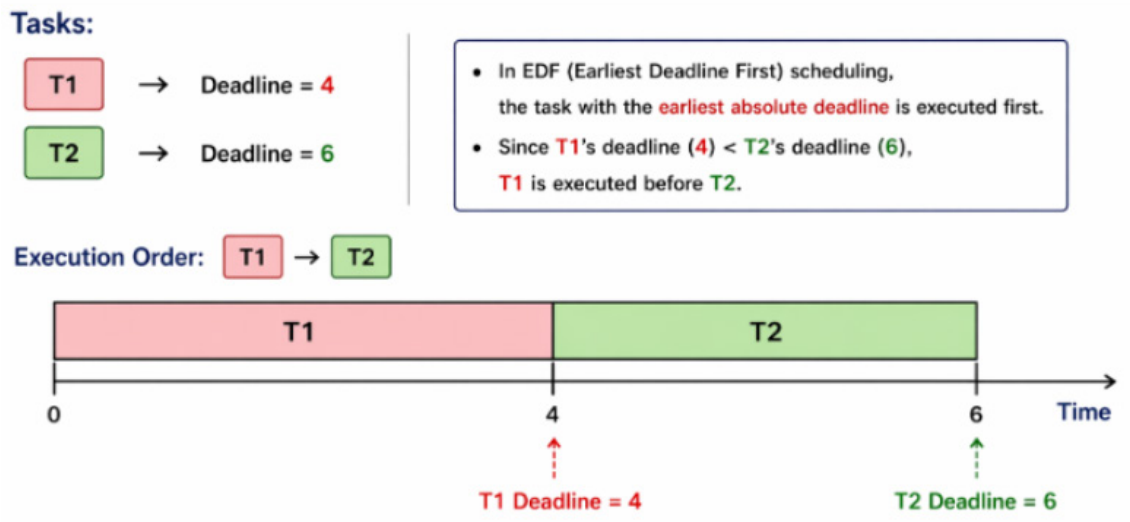


Fig 4.3.5 EDF Scheduling

The fig 4.3.5 shows that T1 executes first because it has an earlier deadline than T2. After T1 completes, T2 is executed. If a new task with an earlier deadline arrives, it will be executed before others, making EDF highly flexible.

4.3.3.3 Priority-Based Scheduling

In Priority-Based Scheduling, each task is assigned a priority, and the CPU executes the task with the highest priority first. This method is simple and widely used in real-time systems.

Types of Priority Scheduling

- **Preemptive Scheduling:** A high-priority task can interrupt a low-priority task to ensure immediate execution.
- **Non-Preemptive Scheduling:** A task continues execution until completion without interruption.

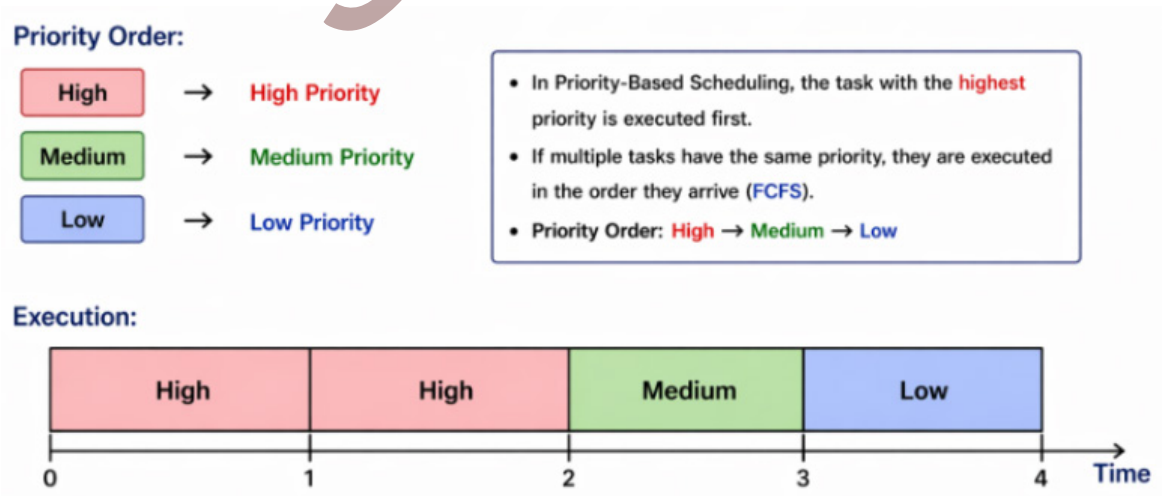


Fig 4.3.6 Priority-Based Scheduling

SGOU

The fig 4.3.6 shows that high-priority tasks are executed first, followed by medium and low-priority tasks. This ensures that critical tasks are handled immediately, but lower-priority tasks may have to wait, which can sometimes lead to starvation.

Scheduling in RTOS is essential for ensuring that tasks meet their deadlines. RMS provides a predictable approach using fixed priorities based on task periods. EDF offers flexibility and better CPU utilization by dynamically assigning priorities based on deadlines. Priority-based scheduling ensures that critical tasks are executed first, making it suitable for many real-time applications. The choice of scheduling method depends on system requirements, task characteristics, and the need for predictability or flexibility.

4.3.4 Resource Management In Rtos

Efficient resource management ensures that all system resources such as memory, CPU, and shared data are utilized effectively. This guarantees predictable timing, reliability, and system stability, which are critical in real-time applications like embedded systems, robotics, and industrial automation.

4.3.4.1 Memory Management

Memory management in RTOS is designed to be predictable and efficient. Unlike general-purpose operating systems, RTOS avoids dynamic memory allocation during runtime because it can cause unpredictable delays and memory fragmentation. Instead, it uses pre-allocated fixed-size memory blocks, commonly organized as memory pools. This allows allocation and deallocation to occur in constant time, ensuring faster execution and improved system reliability. Memory partitioning further enhances predictability by dividing memory into dedicated sections for different tasks.

For example, in an embedded washing machine system, separate memory blocks are pre-allocated for tasks such as water control, timer operation, and display management. When a task needs memory, it directly obtains a block from the pool without delay, ensuring smooth and real-time performance.

4.3.4.2 CPU Management

In RTOS, multiple tasks often need to access shared resources such as memory, files, or sensors. Without proper control, this can lead to data inconsistency and system errors. Synchronization mechanisms are used to ensure that only one task accesses a shared resource at a time.

Semaphores are used as signaling tools for communication and coordination between tasks, while mutex locks ensure mutual exclusion by allowing only one task to access a resource at a time. These techniques help maintain data integrity and ensure correct system behavior.

4.3.4.3 Synchronization

In RTOS, multiple tasks often need to access shared resources such as memory, files, or sensors. Without proper control, this can lead to data inconsistency and system errors.

SGOU

Synchronization mechanisms are used to ensure that only one task accesses a shared resource at a time.

Semaphores are used as signaling tools for communication and coordination between tasks, while mutex locks ensure mutual exclusion by allowing only one task to access a resource at a time. These techniques help maintain data integrity and ensure correct system behavior.

4.3.4.4 Deadlock Handling

A deadlock is a situation where two or more tasks are stuck waiting indefinitely for resources held by each other, causing the system to halt. This typically occurs due to conditions such as mutual exclusion, hold and wait, no preemption, and circular waiting.

RTOS prevents deadlocks using techniques like resource ordering, where resources are requested in a fixed sequence, and timeout mechanisms, where tasks wait only for a limited time before retrying or exiting.

For example, if Task A holds Resource 1 and waits for Resource 2, while Task B holds Resource 2 and waits for Resource 1, both tasks will be blocked forever. By enforcing a fixed resource request order, such situations can be avoided.

4.3.4.5 Priority Inheritance

Priority inheritance is a technique used to solve priority inversion, where a high-priority task is blocked by a low-priority task holding a required resource, while a medium-priority task delays execution.

In this mechanism, the low-priority task temporarily inherits the higher priority of the blocked task. This allows it to complete its execution faster and release the resource. Once the resource is released, the task returns to its original priority.

For example, in an automotive system, a brake control task (high priority) may be blocked by a sensor update task (low priority) holding a resource. Without priority inheritance, a medium-priority task like a music system could delay execution. With priority inheritance, the sensor task is boosted to high priority, quickly releases the resource, and allows the brake control task to execute immediately.

4.3.5 Applications Of Rtos

Real-Time Operating Systems (RTOS) are widely used in systems where timely and predictable responses are essential. These systems must perform tasks within strict deadlines to ensure proper functioning and safety. RTOS is commonly applied in embedded devices, industrial environments, healthcare systems, automotive technologies, and communication networks.

4.3.5.1 Embedded Systems

Embedded systems are small, dedicated computing devices designed to perform specific functions. RTOS plays a crucial role in controlling these systems by ensuring that tasks

SGOU

are executed efficiently and within precise time limits. Since embedded systems often have limited memory and processing power, RTOS helps manage resources effectively while maintaining reliability.

For example, in a washing machine, RTOS controls operations such as water level management, washing cycles, and timing functions. Similarly, in a microwave oven, it manages cooking time, temperature control, and user input. In both cases, the system must respond instantly to user commands and sensor inputs, making RTOS essential.

4.3.5.2 Industrial Automation

In industrial automation, RTOS is used to control complex processes and machinery in factories and production units. These systems require accurate timing and coordination to ensure smooth and efficient operations. RTOS helps in managing multiple tasks simultaneously, such as monitoring sensors, controlling machines, and maintaining safety standards.

For instance, in robotics, RTOS ensures precise movement and coordination of robotic arms used in manufacturing. In assembly lines, it controls the sequence of operations, ensuring that each step is completed at the right time without delays. This improves productivity and reduces errors in industrial processes.

4.3.5.3 Medical Systems

Medical systems rely heavily on RTOS because they require high precision and reliability to ensure patient safety. Any delay or error in these systems can have serious consequences. RTOS ensures that critical tasks are executed immediately and without interruption.

For example, heart monitors continuously track a patient's heartbeat and must respond instantly if any abnormality is detected. In ICU systems, RTOS manages multiple devices such as ventilators, infusion pumps, and monitoring equipment, ensuring they operate accurately and in coordination. This real-time response is vital for saving lives.

4.3.5.4 Automotive Systems

In modern vehicles, RTOS is used to control various safety-critical and performance-related systems. These systems must react instantly to changing conditions to ensure passenger safety and vehicle stability. RTOS provides fast and reliable execution of tasks, especially in emergency situations.

For example, airbag systems rely on RTOS to detect collisions and deploy airbags within milliseconds. Anti-lock braking systems (ABS) use RTOS to prevent wheel locking during sudden braking, maintaining control of the vehicle. These applications highlight the importance of real-time response in automotive safety.

4.3.5.5 Telecommunications

RTOS is widely used in telecommunications to manage real-time data transmission and communication processes. These systems handle large volumes of data and require

SGOU

fast processing to maintain seamless connectivity. RTOS ensures that data packets are transmitted, received, and processed without delay.

For instance, routers use RTOS to direct data packets efficiently across networks. Network switching systems rely on RTOS to manage connections and ensure smooth communication between devices. This helps maintain high-speed and reliable communication in modern networks.

Summarized Overview

RTOS is a specialized operating system designed for time-critical applications where tasks must meet strict deadlines. It ensures deterministic behavior, low latency, and high reliability. RTOS supports scheduling techniques like Rate Monotonic Scheduling (RMS) and Earliest Deadline First (EDF). It also provides efficient resource management, synchronization, and deadlock handling. RTOS is widely used in embedded systems, industrial automation, medical devices, automotive systems, and telecommunications.

Assignments

1. Define RTOS and explain its characteristics.
2. Differentiate between hard and soft real-time systems.
3. Explain Rate Monotonic Scheduling (RMS) with example.
4. Describe Earliest Deadline First (EDF) scheduling.
5. Explain resource management in RTOS.
6. Discuss synchronization techniques used in RTOS.
7. Explain applications of RTOS with examples.

SGOU

Reference

1. Qing, L., & Yifeng, Z. (2012). *Embedded real-time operating systems*. Springer.
2. Wang, K. C. (2017). *Embedded and real-time operating systems*. Springer.
3. Vahid, F., & Givargis, T. (2010). *Embedded system design: A unified hardware/software introduction*. Wiley.
4. Noergaard, T. (2005). *Embedded systems architecture: A comprehensive guide for engineers and programmers*. Elsevier.

Suggested Reading

1. Laplante, P. A. (2017). *Real-time systems design and analysis*. Wiley.
2. Liu, J. W. S. (2000). *Real-time systems*. Prentice Hall.
3. Buttazzo, G. C. (2011). *Hard real-time computing systems*. Springer.
4. Krishna, C. M., & Shin, K. G. (1997). *Real-time systems*. McGraw-Hill.

SGOU

Space for Learner Engagement for Objective Questions

Learners are encouraged to develop objective questions based on the content in the paragraph as a sign of their comprehension of the content. The Learners may reflect on the recap bullets and relate their understanding with the narrative in order to frame objective questions from the given text. The University expects that 1 - 2 questions are developed for each paragraph. The space given below can be used for listing the questions.

SGOU

4 UNIT

Mobile Operating Systems

Learning Outcomes

After completion of this unit, the learner will be able to:

- define a Mobile Operating System and the characteristics of Mobile Operating Systems.
- explain the architecture of a Mobile Operating System
- describe process management in Mobile Operating Systems
- explain memory management in Mobile Operating Systems
- discuss Security, Privacy features in Mobile Operating Systems

SGOU

Background

Every day, millions of people use smartphones, tablets, and wearable devices for communication, learning, banking, shopping, entertainment, and navigation. These devices perform a wide range of tasks smoothly and efficiently, but have you ever wondered what makes them work? Behind every mobile device is a Mobile Operating System that manages hardware resources, runs applications, provides security, and enables seamless user interaction. Understanding how mobile operating systems function helps learners appreciate the technology that powers modern mobile computing.

In this unit, you will explore the fundamental concepts of Mobile Operating Systems, their key characteristics, architecture, process and memory management techniques, and the security and privacy features that protect user data. This knowledge will help you understand how mobile devices deliver efficient, secure, and user-friendly services in today's connected world.

Keywords

Mobile OS Architecture, Process Management, Memory Management, Security and Privacy

Discussion

4.4.1 Introduction to Mobile Operating Systems

The widespread use of smartphones, tablets, smartwatches, and other portable devices has made mobile computing an integral part of everyday life. These devices enable users to perform a variety of activities such as communication, internet browsing, online learning, multimedia streaming, navigation, and mobile banking. The efficient functioning of these devices is made possible by a specialized type of operating system known as a Mobile Operating System (Mobile OS).

A Mobile Operating System is a system software specifically designed to operate mobile devices such as smartphones, tablets, wearable devices, and other portable computing systems. It acts as an interface between the hardware components of the device and the application software, providing a platform for executing mobile applications and managing system resources.

In simple terms, a Mobile Operating System controls the overall operation of a mobile device. It manages essential functions such as process execution, memory allocation, file management, device communication, network connectivity, and user interaction. It also provides the environment required for installing and running various mobile applications.

Unlike traditional desktop operating systems, mobile operating systems are designed to meet the unique requirements of mobile devices. They are optimized for touch-based interaction, wireless communication, mobility, sensor integration, and efficient battery utilization. Modern mobile operating systems also support advanced features such as multitasking, cloud connectivity, biometric authentication, and artificial intelligence-based services.

Over the years, several mobile operating systems have been developed for different types of mobile devices. While some have become widely adopted, others are used in specific devices or application domains.

Examples of Mobile Operating Systems

Some of the commonly used mobile operating systems are listed below:

SGOU

Table 4.4.1 Example of Mobile Operating System

Mobile Operating System	Developer	Commonly Used In
Android	Google	Smartphones and Tablets
iOS	Apple	iPhone and iPad
HarmonyOS	Huawei	Smartphones and IoT Devices
KaiOS	KaiOS Technologies	Feature Phones
Wear OS	Google	Smartwatches
watchOS	Apple	Apple Smartwatches

Among these, Android and iOS are currently the most widely used mobile operating systems across the world.

4.4.2 Characteristics of Mobile Operating Systems

Mobile Operating Systems are designed specifically for portable devices such as smartphones, tablets, and wearable devices. Unlike traditional desktop operating systems, mobile operating systems must operate efficiently within the limitations of battery power, memory, processing capability, and wireless connectivity. To meet these requirements, mobile operating systems possess several unique characteristics that enable them to provide a responsive, secure, and user-friendly computing environment.

The following sections discuss the major characteristics of modern mobile operating systems. The various characteristics of Mobile OS is shown in figure 4.4.1.



Fig. 4.4.1 Characteristics of Mobile OS

1. Mobility and Portability

One of the most significant characteristics of a Mobile Operating System is its support for mobility. Mobile devices are designed to be carried and used in different locations, allowing users to access information and services while on the move.

Mobile operating systems enable users to remain connected through various wireless communication technologies and ensure uninterrupted access to applications and services regardless of location.

Features

- Supports mobile and wireless communication.
- Provides access to services from different locations.
- Enables continuous connectivity while moving between networks.
- Supports location-based services.

Example

A user can access email, attend online classes, or navigate using GPS while travelling from one location to another.

2. Touch-Based User Interface

Mobile operating systems are primarily designed for touchscreen devices. Instead of relying on a keyboard and mouse, users interact directly with the screen through touch gestures.

Modern mobile operating systems provide intuitive graphical user interfaces that simplify device interaction.

Common Touch Gestures

Table 4.4.2 Example of Common Touch Gestures

Gesture	Function
Tap	Select an item
Double Tap	Zoom in or open content
Swipe	Move between screens
Pinch	Zoom in or out
Long Press	Access additional options

3. Energy Efficiency

Mobile devices depend on battery power for their operation. Therefore, efficient energy management is an essential characteristic of mobile operating systems.

SGOU

The operating system continuously monitors system activities and optimizes resource usage to reduce power consumption.

Energy Management Techniques

- Adaptive battery management
- Power-saving modes
- Background process control
- CPU frequency scaling
- Application hibernation

Example:

When the battery level becomes low, the operating system may reduce screen brightness, limit background activities, and disable unnecessary services to conserve energy.

4. Wireless Connectivity

Mobile operating systems support various wireless communication technologies that allow devices to connect to networks and other devices without physical cables.

Supported Technologies

Modern mobile operating systems support various wireless communication technologies that enable seamless connectivity and communication between devices and networks.

- **Wi-Fi**: Enables devices to connect to the internet through wireless local area networks.
- **Bluetooth**: Facilitates short-range wireless communication between devices such as smartphones, earphones, and smartwatches.
- **NFC (Near Field Communication)**: Allows data exchange and contactless transactions between devices placed close to each other.
- **GPS (Global Positioning System)**: Provides location and navigation services by determining the geographical position of a device.
- **4G LTE**: Offers high-speed mobile broadband connectivity for internet access and multimedia applications.
- **5G Networks**: Provide faster data transfer rates, lower latency, and improved support for advanced mobile applications and IoT devices.

Example:

A smartphone user may connect to the internet using Wi-Fi, listen to music through Bluetooth earphones, make a contactless payment using NFC, and simultaneously use GPS-based navigation, demonstrating the versatile connectivity features supported by modern mobile operating systems.

SGOU

5. Sensor Integration

Modern mobile devices contain multiple sensors that collect information about the device and its surroundings. Mobile operating systems manage these sensors and provide secure access to sensor data for applications.

Common Sensors in Mobile Devices

Table 4.4.3 Example of common

Sensor	Purpose
Accelerometer	Detects movement
Gyroscope	Measures orientation
GPS Sensor	Determines location
Proximity Sensor	Detects nearby objects
Ambient Light Sensor	Measures surrounding light
Fingerprint Sensor	User authentication

For example, when a user rotates a smartphone, the operating system receives data from the accelerometer and automatically changes the screen orientation.

6. Multitasking Support

Multitasking refers to the ability of an operating system to execute multiple applications or tasks simultaneously.

Mobile operating systems efficiently allocate processor time and memory resources among running applications to provide a smooth and responsive user experience. Although multiple applications appear to run at the same time, the operating system rapidly switches between them, ensuring that each application receives the resources it requires.

Examples

A user can:

- Listen to music,
- Browse the internet,
- Receive messages, and
- Download files

simultaneously on the same device.

Consider a student attending an online class on a smartphone. While participating in

SGOU

the class, the student can simultaneously receive messages, download study materials, and listen to audio notifications. The Mobile Operating System manages these activities efficiently without significantly affecting device performance.

7. Application Management

Mobile operating systems provide an environment for installing, executing, updating, and removing applications.

Applications are generally distributed through official application stores.

Examples

- Google Play Store (Android)
- Apple App Store (iOS)

Functions of Application Management

- Application installation
- Application updates
- Permission management
- Application lifecycle management

The operating system ensures that applications operate efficiently without interfering with one another.

8. Cloud Integration

Modern mobile operating systems support integration with cloud-based services. Cloud services allow users to store data on remote servers instead of storing everything on the device itself.

Cloud integration enables users to access, synchronize, and manage their data across multiple devices connected to the internet. This ensures that important information remains available even if the device is lost, damaged, or replaced.

Examples

- **Google Drive**: Provides cloud storage for documents, images, videos, and other files.
- **Google Photos**: Automatically backs up and organizes photos and videos in the cloud.
- **iCloud Drive**: Allows Apple users to store and synchronize files across iPhones, iPads, and Macs.
- **OneDrive**: Microsoft's cloud storage service for storing and sharing files online.

SGOU

Benefits

- **Data Backup**: Important files and data can be automatically stored in the cloud, reducing the risk of data loss.
- **File Synchronization**: Changes made on one device are automatically updated on other connected devices.
- **Remote Access to Information**: Users can access their files from any location using an internet connection.
- **Easy Device Migration**: Data can be easily transferred to a new device by signing in to the same cloud account.

For example, suppose a user takes photographs on a smartphone and enables Google Photos backup. The images are automatically uploaded to the cloud and can later be viewed from a tablet, laptop, or another smartphone using the same account.

9. Security and Privacy Support

Mobile devices store large amounts of personal and sensitive information, including contacts, messages, photographs, banking details, and location data. Therefore, security and privacy have become fundamental characteristics of modern mobile operating systems.

Mobile operating systems incorporate various mechanisms to protect user data, prevent unauthorized access, and give users greater control over how applications access their information.

Security Features

- **User Authentication**: Verifies the identity of the user before granting access to the device.
- **Fingerprint Recognition**: Allows users to unlock devices securely using their fingerprints.
- **Face Recognition**: Uses facial features to authenticate users and enhance device security.
- **Data Encryption**: Converts data into a secure format to prevent unauthorized access.
- **Secure Application Installation**: Helps ensure that applications are obtained from trusted sources and checked for security threats.
- **Permission-Based Access Control**: Requires applications to obtain user permission before accessing sensitive resources such as contacts, camera, or location.

Privacy Features

- **Location Permission Control**: Allows users to decide whether an application can access their location information.

SGOU

- **Camera and Microphone Access Control**: Enables users to control which applications can use the camera and microphone.
- **Privacy Dashboards**: Provide information about how applications access sensitive resources and data.
- **Application Tracking Restrictions**: Limit the ability of applications to track user activities across different apps and websites.

For example, when a mobile application requests access to the device's location or camera, the operating system displays a permission request. The user can choose to allow or deny access, helping to protect personal information and maintain privacy.

10. Context Awareness

Context awareness refers to the ability of a mobile operating system to adapt its behavior based on information obtained from the user and the surrounding environment.

The operating system gathers contextual information through sensors, location services, network status, and user activities.

Applications

- Automatic screen brightness adjustment
- Location-based recommendations
- Activity tracking
- Smart notifications

For example, a smartphone may automatically switch to silent mode during a scheduled meeting based on calendar information.

4.4.3 Mobile OS Architecture

A Mobile Operating System is a complex software system that manages the hardware and software resources of a mobile device. To perform various functions efficiently, a Mobile OS is organized into multiple layers, where each layer performs specific tasks and interacts with the layers above and below it.

This layered approach simplifies system design, improves maintainability, and allows developers to build applications without directly interacting with the hardware components of the device.

The architecture of a Mobile Operating System provides a structured framework for managing applications, memory, processes, communication, security, and hardware resources.

The architecture of a typical Mobile Operating System consists of the following layers as shown in figure 4.4.2.

SGOU

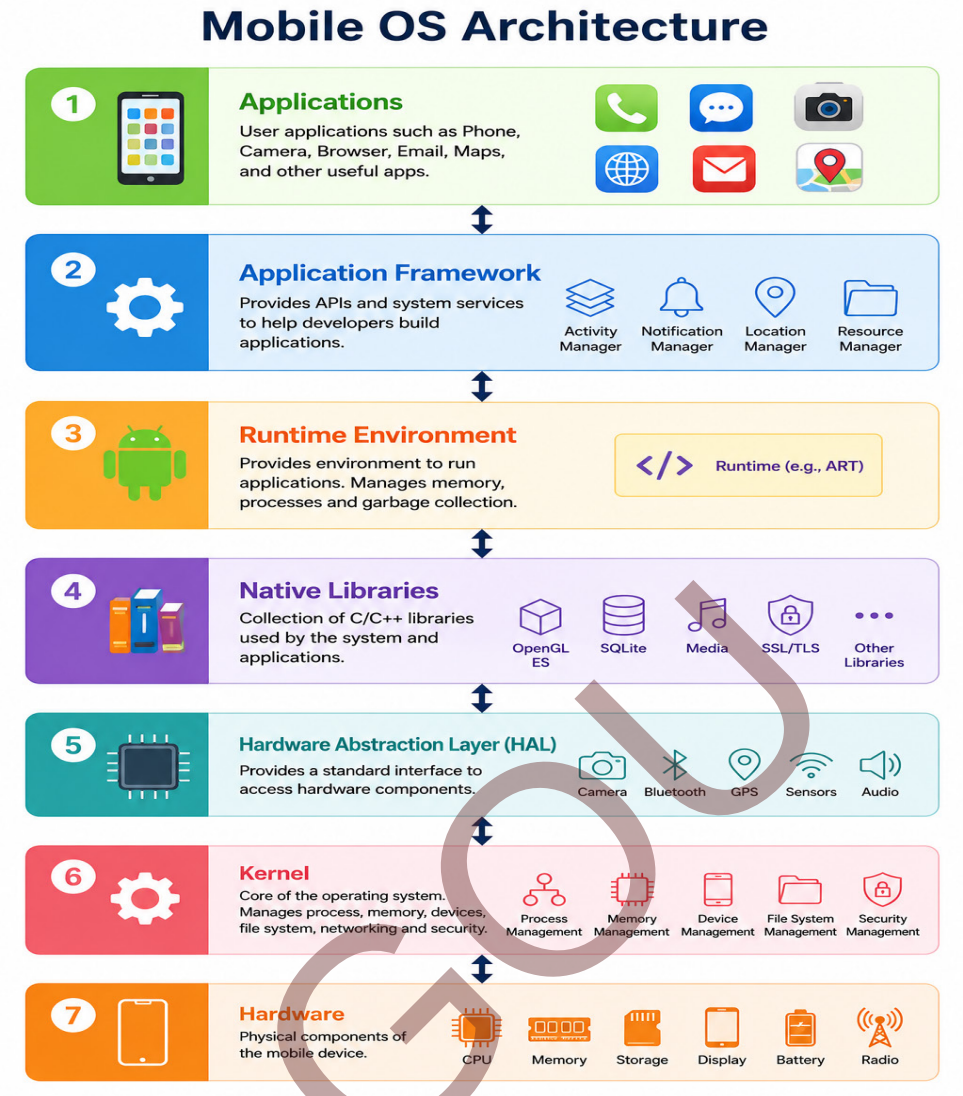


Fig. 4.4.2 Mobile OS Architecture

SGOU

1. Applications Layer

The Applications Layer is the topmost layer of the Mobile OS Architecture. It contains all the applications that users interact with directly. These applications provide various services such as communication, entertainment, education, productivity, navigation, and online transactions. Both system applications and user-installed applications reside in this layer.

Functions:

a. Provides Services to Users:

This layer enables users to perform various activities such as making calls, sending messages, browsing the internet, watching videos, and accessing online services through mobile applications.

b. Facilitates User Interaction:

Applications act as the interface between users and the mobile device, allowing users to

access device features in a simple and convenient manner.

c. Supports Diverse Activities:

The layer supports a wide range of activities including communication, learning, entertainment, banking, shopping, and navigation through specialized applications.

d. Utilizes System Resources:

Applications make use of services provided by lower layers of the operating system to access hardware resources such as cameras, sensors, storage, and network connections.

Example

Applications such as Phone, Camera, WhatsApp, Gmail, Google Maps, and YouTube belong to the Applications Layer.

2. Application Framework Layer

The Application Framework Layer provides a collection of reusable services and Application Programming Interfaces (APIs) that support the development and execution of mobile applications. It acts as an intermediary between applications and the lower layers of the operating system. By providing standardized services, the framework simplifies application development and ensures consistent interaction with system resources.

To perform these responsibilities effectively, the Application Framework Layer offers several important functions that support both application developers and end users.

Functions

a. Provides APIs for Application Development:

The framework offers a set of predefined APIs that enable developers to build applications efficiently without directly interacting with hardware components.

b. Manages Application Lifecycle:

It controls the creation, execution, suspension, and termination of applications, ensuring efficient utilization of system resources.

c. Handles Notifications and Alerts:

The framework provides notification services that allow applications to inform users about messages, reminders, updates, and other important events.

d. Provides Location and Resource Services:

It enables applications to access location information, device resources, and other system services through standardized interfaces.

Example

When a weather application retrieves the user's location to display local weather

SGOU

information, it uses location services provided by the Application Framework Layer.

3. Kernel Layer

The Kernel Layer is the core component of the Mobile Operating System and serves as the foundation upon which all other layers operate. It directly interacts with the hardware and manages critical system resources such as processor time, memory, storage, and communication devices. The kernel ensures that applications and system services can function efficiently while maintaining system stability and security.

In order to coordinate and manage these resources effectively, the Kernel Layer performs several essential functions.

a. Manages Processes:

The kernel creates, schedules, and terminates processes, allowing multiple applications to execute efficiently.

b. Performs Memory Management:

It allocates memory to applications and system processes while preventing conflicts among them.

c. Controls Hardware Devices:

The kernel communicates with hardware components through device drivers and coordinates their operation.

d. Handles File and Storage Management:

It manages the storage, organization, retrieval, and protection of files within the device.

e. Enforces Security Policies:

The kernel controls access permissions and protects system resources from unauthorized access.

Example

When multiple applications such as a music player, browser, and messaging application run simultaneously, the kernel allocates processor time and memory resources to ensure smooth operation.

4. Runtime Environment

The Runtime Environment is the layer responsible for executing mobile applications and providing the necessary environment for their operation. While the Application Framework offers services and APIs for application development, the Runtime Environment ensures that applications can run efficiently on the mobile device. It acts as an execution platform that manages application code, memory resources, and various runtime operations required during application execution.

In modern mobile operating systems, the Runtime Environment plays a crucial role in improving application performance, optimizing resource utilization, and maintaining

SGOU

system stability. For example, Android uses the Android Runtime (ART), which provides efficient execution of Android applications through advanced optimization techniques.

To support the smooth execution of applications, the Runtime Environment performs several important functions.

a. Executes Application Code:

The Runtime Environment loads and executes application programs, enabling them to perform their intended tasks. It translates application instructions into a form that can be processed by the device.

b. Manages Memory Allocation:

It allocates memory to running applications and ensures that sufficient memory resources are available for their execution. Efficient memory management helps improve application performance.

c. Performs Garbage Collection:

The Runtime Environment automatically identifies and removes unused objects from memory. This process, known as garbage collection, helps free memory resources and prevents memory leaks.

d. Optimizes Application Performance:

It uses various optimization techniques to improve application startup time, execution speed, and overall responsiveness.

e. Provides Runtime Services:

The Runtime Environment offers essential services required during application execution, such as exception handling, thread management, and resource management.

5. Native Libraries Layer

The Native Libraries Layer consists of a collection of software libraries that provide essential functionalities required by applications and system services. These libraries are typically written in high-performance programming languages such as C and C++ and are optimized to perform complex operations efficiently. Rather than developing these functionalities from scratch, applications can utilize the services provided by these libraries through the Runtime Environment and Application Framework.

Native libraries play an important role in enhancing the performance and capabilities of mobile applications. They support various functions such as multimedia processing, graphics rendering, database management, web content display, and secure communication. By providing ready-to-use functionalities, this layer simplifies application development and improves overall system efficiency.

To support various operations within the Mobile Operating System, the Native Libraries Layer performs several important functions.

SGOU

a. Supports Multimedia Processing:

Native libraries provide functionalities for processing audio, video, and image files. They enable applications to play music, stream videos, and display multimedia content efficiently.

b. Provides Database Management Services:

Database libraries help applications store, retrieve, and manage structured data. These services are widely used in applications that maintain user information, settings, and records.

c. Facilitates Graphics Rendering:

Graphics libraries support the creation and display of visual elements, animations, and graphical user interfaces. They help deliver smooth and interactive user experiences.

d. Enables Web Content Rendering:

Certain libraries assist in displaying and processing web pages within mobile browsers and web-based applications.

e. Supports Secure Communication:

Security libraries provide encryption and authentication mechanisms that help protect data transmitted over networks.

f. Promotes Code Reusability:

Since multiple applications can use the same libraries, developers can reuse existing functionalities instead of developing them repeatedly, reducing development effort and improving efficiency.

Table 4.4.4 Common Native Libraries

Library	Purpose
SQLite	Database Management
OpenGL ES	Graphics rendering
Media Framework	Audio and video processing
WebKit	Web page rendering
SSL/TLS Libraries	Secure communication

When a user watches a video on a smartphone, the Media Framework libraries process and decode the video content, enabling smooth playback. Similarly, applications that store data locally often use the SQLite library for database management.

6. Hardware Abstraction Layer (HAL)

The Hardware Abstraction Layer (HAL) acts as an intermediary between the software

SGOU

layers of the Mobile Operating System and the hardware components of the mobile device. It provides a standardized interface through which software components can access hardware resources without needing to understand the specific details of the underlying hardware implementation.

Mobile devices are manufactured with different hardware configurations, including cameras, sensors, processors, GPS modules, and communication devices from various vendors. The HAL hides these hardware-specific complexities and presents a uniform interface to the upper layers of the operating system. As a result, applications and system services can interact with hardware devices in a consistent manner, regardless of the hardware manufacturer.

By abstracting hardware details, the HAL improves portability, simplifies hardware integration, and reduces the complexity of software development.

To facilitate communication between software and hardware components, the Hardware Abstraction Layer performs several important functions.

a. Provides a Standardized Interface:

HAL offers a common interface through which software components can access hardware resources. This standardization allows applications and system services to operate independently of specific hardware implementations.

b. Facilitates Communication with Hardware Devices:

The layer enables software modules to communicate with hardware components such as cameras, sensors, GPS modules, Bluetooth adapters, and audio devices without directly interacting with them.

c. Hides Hardware Complexity:

HAL abstracts the technical details of hardware devices and presents simplified interfaces to higher software layers. This reduces the complexity involved in application development and system maintenance.

d. Supports Hardware Independence:

Since software components interact with standardized interfaces rather than specific hardware implementations, the operating system can support a wide range of hardware configurations with minimal modifications.

e. Simplifies Device Integration:

When new hardware components are introduced, developers can implement the required hardware-specific functions within the HAL without affecting higher software layers.

Hardware Components Accessed Through HAL

- Camera
- GPS Module
- Bluetooth Adapter

SGOU

- Wi-Fi Module
- Audio Devices
- Accelerometer
- Gyroscope
- Proximity Sensor
- Fingerprint Sensor

Consider a camera application that needs to capture a photograph. Instead of communicating directly with the camera hardware, the application sends its request through the operating system. The HAL receives this request and translates it into commands that the specific camera hardware can understand. Once the image is captured, the HAL transfers the data back to the upper layers for processing and display.

7. Kernel Layer

The Kernel Layer is the core and most fundamental component of the Mobile Operating System. It serves as the bridge between the software and hardware of the mobile device and is responsible for managing system resources efficiently. All higher layers of the Mobile OS depend on the services provided by the kernel to perform their operations.

The kernel directly interacts with the device hardware and controls essential functions such as process execution, memory allocation, device communication, file management, networking, and security. Since it operates at the lowest software level, the kernel plays a crucial role in ensuring the stability, reliability, and performance of the entire operating system.

Most modern mobile operating systems, including Android, are built on a modified version of the Linux kernel due to its robustness, security, and efficient resource management capabilities.

To manage the overall operation of the mobile device, the Kernel Layer performs several important functions.

a. Process Management:

The kernel manages the creation, scheduling, execution, and termination of processes. It ensures that multiple applications can run simultaneously by allocating processor time efficiently among them.

b. Memory Management:

The kernel allocates memory resources to applications and system processes based on their requirements. It also monitors memory usage and prevents conflicts between running applications.

c. Device Management:

The kernel controls communication between the operating system and hardware devices

SGOU

through device drivers. It manages devices such as cameras, displays, keyboards, sensors, speakers, and communication modules.

d. File System Management:

The kernel organizes and manages files stored on the device. It handles file creation, storage, retrieval, modification, and deletion while maintaining data integrity.

e. Network Management:

The kernel manages network communication and supports connectivity through technologies such as Wi-Fi, Bluetooth, mobile data networks, and internet protocols.

f. Security Management:

The kernel enforces security policies and controls access to system resources. It helps prevent unauthorized access and protects sensitive data stored on the device.

g. Power Management:

The kernel monitors power consumption and implements mechanisms to optimize battery usage. It manages power allocation to hardware components and background processes to improve battery life.

8. Hardware Layer

The Hardware Layer is the lowest layer of the Mobile OS Architecture and consists of all the physical components of a mobile device. These hardware components perform the actual computing, storage, communication, and input-output operations required for the functioning of the device.

The hardware layer provides the resources upon which the entire operating system and its applications depend. All software requests eventually reach the hardware layer, where the required operations are executed. The Mobile Operating System manages and coordinates these hardware components through the Kernel Layer and Hardware Abstraction Layer.

Modern mobile devices incorporate a wide range of hardware components that work together to provide computing capabilities, connectivity, multimedia support, and intelligent features.

To support the operation of mobile devices, the Hardware Layer performs several important functions.

a. Data Processing:

The processor (CPU) executes instructions and performs calculations required by applications and the operating system. It serves as the primary computing component of the device.

b. Data Storage:

Memory and storage devices store applications, operating system files, multimedia

SGOU

content, and user data. RAM provides temporary storage, while internal storage retains data permanently.

c. Input and Output Operations:

Hardware components such as touchscreens, microphones, cameras, speakers, and displays facilitate interaction between users and the device.

d. Wireless Communication:

Communication modules enable connectivity through Wi-Fi, Bluetooth, GPS, NFC, 4G LTE, and 5G networks, allowing users to access internet services and communicate with other devices.

e. Sensor-Based Data Collection:

Various sensors collect information about the device and its environment. This information is used by applications and system services to provide intelligent functionalities.

f. Power Supply and Management:

The battery supplies power to the device, enabling portable operation. Hardware power management components help regulate energy consumption and charging processes.

Major Hardware Components

- Processor (CPU)
- Random Access Memory (RAM)
- Internal Storage
- Touchscreen Display
- Camera
- Microphone
- Speaker
- Battery
- Wi-Fi Module
- Bluetooth Module
- GPS Receiver
- NFC Module
- Accelerometer
- Gyroscope
- Proximity Sensor

SGOU

- Ambient Light Sensor
- Fingerprint Sensor

When a user takes a photograph, the camera hardware captures the image, the processor processes the image data, the storage device saves the photograph, and the display presents the captured image to the user.

4.4.4 Process and Memory Management in Mobile Operating Systems

Modern mobile devices are expected to perform multiple tasks simultaneously while operating within limited hardware resources and battery power. Users often switch between applications such as messaging, web browsing, music streaming, video playback, and navigation without experiencing significant delays. To support such functionality, mobile operating systems must efficiently manage both processes and memory.

Process Management ensures that multiple applications and system services execute smoothly by allocating processor time and coordinating their activities. **Memory Management**, on the other hand, ensures that available memory resources are utilized efficiently so that applications can run without performance degradation or system instability.

Effective process and memory management are essential for maintaining responsiveness, optimizing resource utilization, reducing power consumption, and improving the overall user experience in mobile devices.

4.4.4.1 Process Management in Mobile Operating Systems

A **process** is an instance of a program that is currently being executed by the operating system. Whenever a user launches an application such as a web browser, messaging application, or media player, the operating system creates a process to execute that application.

In mobile operating systems, multiple processes often run simultaneously. For example, a user may be listening to music, receiving notifications, browsing the internet, and downloading files at the same time. Process management ensures that these activities are coordinated efficiently without causing conflicts or excessive resource consumption.

Since mobile devices have limited processing power and battery capacity compared to desktop computers, process management techniques are specifically designed to balance performance and energy efficiency.

To manage application execution effectively, mobile operating systems perform several process management functions.

Functions of Process Management

1. Process Creation

The operating system creates a new process whenever a user launches an application or

SGOU

when a system service needs to execute a task. Each process is assigned the necessary resources for its operation.

2. Process Scheduling

The operating system determines which process should receive processor time and for how long. Scheduling ensures that all active processes receive fair access to the CPU while maintaining system responsiveness.

3. Multitasking Support

Process management enables multiple applications to run simultaneously. The operating system rapidly switches between processes, creating the illusion that all applications are running at the same time.

4. Process Synchronization

When multiple processes need to access shared resources, synchronization mechanisms help prevent conflicts and ensure data consistency.

5. Process Termination

The operating system terminates processes that are no longer required and releases the resources allocated to them.

6.Background Process Management

Mobile operating systems manage background processes carefully to reduce battery consumption and conserve system resources. Applications running in the background may be restricted or suspended when necessary.

4.4.4.2 Memory Management in Mobile Operating Systems

Memory management refers to the process of allocating, monitoring, and controlling memory resources used by applications and system processes. Mobile devices have limited memory compared to desktop computers, making efficient memory management essential for maintaining system performance and stability.

The operating system continuously monitors memory usage and allocates memory to applications based on their requirements. It also reclaims memory from inactive or terminated applications when necessary.

Proper memory management prevents application crashes, improves multitasking performance, and ensures efficient utilization of available resources.

To achieve these objectives, mobile operating systems perform several memory management functions.

1. Memory Allocation

The operating system allocates memory to applications and system processes whenever they require resources for execution.

SGOU

2. Memory Deallocation

When applications are closed or no longer need memory, the operating system releases the allocated memory for reuse.

3. Virtual Memory Management

Mobile operating systems use virtual memory techniques to create the illusion of larger memory resources than are physically available.

4. Garbage Collection

Unused objects and memory blocks are automatically identified and removed to free memory resources and improve efficiency.

5. Memory Protection

The operating system prevents applications from accessing the memory space of other applications, thereby improving system security and stability.

6. Memory Optimization

Various optimization techniques are used to improve memory utilization and ensure smooth application performance.

4.4.5 Security and Privacy in Mobile Operating Systems

Mobile devices are widely used for communication, banking, shopping, education, healthcare, and entertainment. As a result, they store a significant amount of personal and sensitive information such as contacts, messages, photographs, passwords, and financial data.

With the increasing use of mobile applications and internet services, the risk of security threats and privacy breaches has also grown. Therefore, modern mobile operating systems provide various security and privacy mechanisms to protect user data and ensure safe device usage.

Security focuses on protecting devices and data from unauthorized access and cyber threats, while privacy focuses on controlling how personal information is collected, used, and shared. Together, they help create a secure and trustworthy mobile computing environment.

4.4.5.1 Security in Mobile Operating Systems

Security in a Mobile Operating System refers to the protection of the device, applications, and user data from unauthorized access, malicious software, cyberattacks, and other security threats. Mobile operating systems implement various security mechanisms to safeguard system resources and ensure that only authorized users and applications can access sensitive information.

To achieve these objectives, mobile operating systems provide several security features.

1. User Authentication

SGOU

User authentication is the process of verifying the identity of a user before granting access to the device. Authentication mechanisms help prevent unauthorized individuals from accessing sensitive information stored on the device.

Common authentication methods include:

- PINs
- Passwords
- Pattern Locks
- Biometric Authentication

2. Fingerprint Recognition

Fingerprint recognition is a biometric authentication technique that uses the unique fingerprint patterns of a user for identity verification. This method provides a convenient and secure way to unlock devices and authorize transactions.

3. Face Recognition

Face recognition uses facial features to identify and authenticate users. The device compares the user's facial characteristics with previously stored biometric data. This technology offers a contactless and user-friendly authentication mechanism.

4. Data Encryption

Data encryption converts information into an unreadable format that can only be accessed using the appropriate decryption key.

Even if unauthorized individuals gain access to stored data, encrypted information remains protected.

5. Secure Application Installation

Mobile operating systems implement mechanisms to ensure that applications are obtained from trusted sources and do not contain malicious code. Applications are often scanned and verified before installation.

6. Permission-Based Access Control

Permission-based access control requires applications to obtain user consent before accessing sensitive resources such as contacts, cameras, microphones, storage, or location information. This approach limits unauthorized access to personal data.

4.4.5.2 Privacy in Mobile Operating Systems

Privacy refers to the ability of users to control how their personal information is collected, used, stored, and shared. Mobile operating systems provide various privacy features that enable users to manage access to sensitive data and monitor application behavior.

As mobile applications increasingly collect user information, privacy protection has

SGOU

become a critical component of modern mobile operating systems.

To help users maintain control over their personal information, mobile operating systems offer several privacy features.

1. Location Permission Control

Location permission control allows users to decide whether an application can access their geographical location. Users may grant permanent access, temporary access, or deny access altogether.

2. Camera and Microphone Access Control

Mobile operating systems allow users to control which applications can access the device's camera and microphone. This helps prevent unauthorized recording of audio or video.

3. Privacy Dashboards

Privacy dashboards provide users with detailed information about how applications access sensitive resources such as location, camera, microphone, and contacts.

These dashboards improve transparency and help users identify potentially suspicious application behavior.

4. Application Tracking Restrictions

Many applications attempt to track user activities across different applications and websites for advertising and analytics purposes.

Mobile operating systems provide tracking restrictions that allow users to control or limit such tracking activities.

SGOU

Summarized Overview

Mobile Operating Systems (Mobile OS) are specialized software designed to manage and control mobile devices such as smartphones, tablets, and smartwatches. They act as an interface between the device hardware and applications, enabling users to perform tasks such as communication, internet browsing, online learning, navigation, banking, and entertainment. Popular mobile operating systems include Android, iOS, HarmonyOS, KaiOS, Wear OS, and watchOS. Unlike traditional desktop operating systems, mobile operating systems are optimized for portability, touch-based interaction, wireless communication, and efficient battery usage.

Modern mobile operating systems provide several important features, including mobility, multitasking, wireless connectivity, cloud integration, sensor support, security, and

privacy. They support technologies such as Wi-Fi, Bluetooth, GPS, NFC, 4G, and 5G, allowing users to stay connected and access services from anywhere. Mobile devices also use sensors like accelerometers, gyroscopes, proximity sensors, and fingerprint scanners to enhance functionality and user experience. The architecture of a Mobile OS consists of multiple layers, including applications, application framework, runtime environment, native libraries, hardware abstraction layer, kernel, and hardware.

Efficient process and memory management enable multiple applications to run smoothly while optimizing device resources. Security and privacy are essential aspects of modern Mobile OSs. Features such as passwords, fingerprint and face recognition, data encryption, secure application installation, permission-based access control, and privacy settings help protect user data from unauthorized access. These capabilities ensure a secure, reliable, and user-friendly mobile computing environment.

Assignments

1. Explain the concept of a Mobile Operating System. Discuss its major functions and compare it with a traditional desktop operating system.

2. Describe the key characteristics of modern Mobile Operating Systems, including mobility, touch-based interface, energy efficiency, and wireless connectivity.

3. Discuss the architecture of a Mobile Operating System. Explain the functions of its major layers with suitable examples.

4. Explain the role of process management in Mobile Operating Systems. Discuss process creation, scheduling, multitasking, and background process management.

5. What is memory management in Mobile Operating Systems? Explain the various memory management functions and their importance in improving device performance.

6. Discuss the security features provided by Mobile Operating Systems. Explain how authentication, encryption, secure application installation, and permission-based access control enhance security.

7. Explain the privacy mechanisms available in modern Mobile Operating Systems. Discuss location permission control, camera and microphone access control, privacy dashboards, and application tracking restrictions.

SGOU

Reference

1. Tanenbaum, Andrew S., and Herbert Bos. *Modern Operating Systems*. 4th ed., Pearson, 2015.

2. Coulouris, George, Jean Dollimore, Tim Kindberg, and Gordon Blair. *Distributed Systems: Concepts and Design*. 5th ed., Pearson, 2012.

3. Laudon, Kenneth C., and Jane P. Laudon. *Management Information Systems: Managing the Digital Firm*. 16th ed., Pearson, 2020.

4. Deitel, Paul, Harvey Deitel, Abbey Deitel, and Michael Morgano. *Android for Programmers: An App-Driven Approach*. 3rd ed., Pearson, 2016.

Suggested Reading

1. Stallings, William. *Operating Systems: Internals and Design Principles*. 10th ed., Pearson, 2018.

2. Silberschatz, Abraham, Peter B. Galvin, and Greg Gagne. *Operating System Concepts*. 10th ed., Wiley, 2018.

3. Yaghmour, Karim, et al. *Embedded Android: Porting, Extending, and Customizing*. O'Reilly Media, 2013.

4. Meier, Reto, and Ian Lake. *Professional Android*. 4th ed., Wrox Press, 2018.

5. Burnette, Ed. *Hello, Android: Introducing Google's Mobile Development Platform*. 4th ed., Pragmatic Bookshelf, 2015.

SGOU

Space for Learner Engagement for Objective Questions

Learners are encouraged to develop objective questions based on the content in the paragraph as a sign of their comprehension of the content. The Learners may reflect on the recap bullets and relate their understanding with the narrative in order to frame objective questions from the given text. The University expects that 1 - 2 questions are developed for each paragraph. The space given below can be used for listing the questions.

SGOU

SREENARAYANAGURU OPEN UNIVERSITY

QP CODE: …… Reg. No : .....

Name : …….....

Model Question Paper- set-I

End Semester Examination

MASTER OF COMPUTER APPLICATIONS

M26CA01DC: Operating Systems Design

(CBCS - PG)

2026-27 - Admission Onwards

Time: 3 Hours Max Marks: 70

Section A

Answer any ten of the following questions. Each answer should be in two or three valid points. Each question carries 2 marks. (10×2= 20)

1. List some Real Time Operating Systems.
2. What is multiprogramming?
3. What is a Process?
4. Write about Short-Term Scheduler.
5. What is the role of the Memory Management Unit (MMU)?
6. What is virtual memory?
7. What is Belady's Anomaly?
8. What is First-Fit memory allocation strategy?
9. Explain different types of file attributes.
10. What is Polling?
11. Explain the disk attachment?
12. Explain the importance of the Kernel I/O Subsystem in an operating system.

SGOU

13. Define a distributed operating system. Mention any one of its characteristics.

14. What are the two major goals of operating system security?

15. What is a firewall? Mention its primary purpose.

Section B

Answer any six of the following questions. Provide adequate valid points with clear explanations, proportionate to the marks allotted. Each question carries 5 marks.

(5x6=30)

16. Write about Semaphores in Process Synchronization.

17. What is a Batch Operating System? Write its advantages and disadvantages.

18. Explain logical and physical addresses and their differences

19. Explain the LRU page replacement algorithm with a suitable example

20. Explain the different types of access methods in the file system.

21. Explain the different types of directory implementation.

22. Explain the security goals and system protection mechanisms in operating systems.

23. Explain resource management and applications of real-time operating systems.

24. Explain the major characteristics and architecture of mobile operating systems.

Section C

Answer any two of the following questions. Provide adequate valid points with clear explanations, proportionate to the marks allotted. Each question carries 10 marks.

(10X2=20)

25. Explain about the necessary conditions for Deadlock in operating systems.

26. Explain contiguous memory allocation in detail. Discuss First-Fit, Best-Fit, and Worst-Fit memory allocation strategies with suitable examples

27. Explain the Kernel I/O Subsystem in an Operating System. Discuss its functions, services offered by the kernel, and its importance.

28. Explain the architecture and working of a distributed operating system with suitable examples.

SGOU

SREENARAYANAGURU OPEN UNIVERSITY

QP CODE: …… Reg. No : .....

Name : …….....

Model Question Paper- set-II

End Semester Examination

MASTER OF COMPUTER APPLICATIONS

M26CA01DC: Operating Systems Design

(CBCS - PG)

2026-27 - Admission Onwards

Time: 3 Hours Max Marks: 70

Section A

Answer any ten of the following questions. Each answer should be in two or three valid points. Each question carries 2 marks. (10x2=20)

1. Write about Turnaround Time (TAT) and waiting time.
2. What is a Batch Operating System?
3. What are non-premptive scheduling algorithms? Write any 2 examples.
4. What is a Process Control Block (PCB)?
5. What is swapping?
6. Name any two page replacement algorithms.
7. What is demand paging?
8. What is segmentation?
9. Explain the different file operations.
10. Explain Direct Memory Access (DMA).
11. Explain the different components of disk structure in an operating system.
12. Explain the applications of the Input/Output (I/O) interface in an operating system.

SGOU

13. What is resource management in an RTOS?

14. What is network topology? Name any two types of network topology.

15. Differentiate between authentication and encryption.

Section B

Answer any six of the following questions. Provide adequate valid points with clear explanations, proportionate to the marks allotted. Each question carries 5 marks.

(5x6=30)

16. What is a Multiprogramming Operating System? Explain its advantages and disadvantages.

17. Explain about Process state diagram.

18. Explain internal fragmentation and external fragmentation with suitable examples

19. Explain First-Fit, Best-Fit, and Worst-Fit memory allocation strategies with suitable examples.

20. Explain the different types of directory structures.

21. Compare the different file allocation methods.

22. Explain the major design issues in distributed systems.

23. Explain distributed file systems and their role in distributed operating systems.

24. Explain encryption and authentication as security mechanisms.

Section C

Answer any two of the following questions. Provide adequate valid points with clear explanations, proportionate to the marks allotted. Each question carries 10 marks.

(10X2=20)

25. Explain in detail about Operating system Services.

26. Explain FIFO, and Optimal page replacement algorithms and compare their working.

27. Explain the different disk scheduling methods used in an operating system.

28. Explain real-time operating systems with emphasis on scheduling and resource management.

SGOU

സർവ്വകലാശാലാഗീതം

വിദ്യയാൽ സ്വതന്ത്രരാകണം
വിശ്വപൗരരായി മാറണം
ഗ്രഹപ്രസാദമായ് വിളങ്ങണം
ഗുരുപ്രകാശമേ നയിക്കണേ

കൂരിരുട്ടിൽ നിന്നു ഞങ്ങളെ
സൂര്യവീഥിയിൽ തെളിക്കണം
സ്നേഹദീപ്തിയായ് വിളങ്ങണം
നീതിവൈജയന്തി പാറണം

ശാസ്ത്രവ്യാപ്തിയെന്നുമേകണം
ജാതിഭേദമാകെ മാറണം
ബോധരശ്മിയിൽ തിളങ്ങുവാൻ
ജ്ഞാനകേന്ദ്രമേ ജ്വലിക്കണേ

കുരീപ്പുഴ ശ്രീകുമാർ

SREENARAYANAGURU OPEN UNIVERSITY

SGOU

Regional Centres

Kozhikode
Govt. Arts and Science College
Meenchantha, Kozhikode,
Kerala, Pin: 673002
Ph: 04952920228
email: rckdirector@sgou.ac.in

Thalassery
Govt. Brennen College
Dharmadam, Thalassery,
Kannur, Pin: 670106
Ph: 04902990494
email: rctdirector@sgou.ac.in

Tripunithura
Govt. College
Tripunithura, Ernakulam,
Kerala, Pin: 682301
Ph: 04842927436
email: rcedirector@sgou.ac.in

Pattambi
Sree Neelakanta Govt. Sanskrit College
Pattambi, Palakkad,
Kerala, Pin: 679303
Ph: 04662912009
email: rcpdirector@sgou.ac.in

SAY NO TO RAGGING

BEFORE YOU EVEN THINK OF RAGGING



Humiliation

Blacklisting

Ruined Career

Download

ANTI RAGGING

App

Don't just stand and watch. Stop Ragging! Show Character

Visit UGC Website i.e. www.ugc.Ac.in & www antinaging.in to UGC Anti Ragging regulations.

Ministry of Education
Government of India

Are You Being Ragged ?

विश्वविद्यालय अनुदान आयोग
University Grants Commission
quality higher education for all

DON'T LET IT BE TOO LATE

SAY NO TO DRUGS

LOVE YOURSELF AND ALWAYS BE HEALTHY



SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

SGOU

Operating System Design

COURSE CODE: M26CA01DC

SGOU

Sreenarayanaguru Open University
Kollam, Kerala Pin- 691601, email: info@sgou.ac.in, www.sgou.ac.in Ph: +91 474 2966841