KL (DEEMED TO BE UNIVERSITY) | CDOE CENTER FOR DISTANCE AND ONLINE EDUCATION

MCA

OPERATING SYSTEMS

How to Use Self-Learning Material?

The pedagogy used to design this course is to enable the student to assimilate the concepts with ease. The course is divided into modules. Each module is categorically divided into units or chapters. Each unit has the following elements:

Table of Contents: Each unit has a well-defined table of contents. *For example: "1.1.1. (a)" should be read as "Module 1. Unit 1. Topic 1. (Sub-topic a)" and 1.2.3. (iii) should be read as "Module 1. Unit 2. Topic 3. (Sub-topic iii).*

Aim: It refers to the overall goal that can be achieved by going through the unit.

Instructional Objectives: These are behavioural objectives that describe intended learning and define what the unit intends to deliver.

Learning Outcomes: These are demonstrations of the learner's skills and experience sequences in learning, and refer to what you will be able to accomplish after going through the unit.

Self-Assessment Questions: These include a set of multiple-choice questions to be answered at the end of each topic.

Did You Know?: You will learn some interesting facts about a topic that will help you improve your knowledge. A unit can also contain Quiz, Case Study, Critical Learning Exercises, etc., as metacognitive scaffold for learning.

Summary: This includes brief statements or restatements of the main points of unit and summing up of the knowledge chunks in the unit.

Activity: It actively involves you through various assignments related to direct application of the knowledge gained from the unit. Activities can be both online and offline.

Bibliography: This is a list of books and articles written by a particular author on a particular subject referring to the unit's content.

e-References: This is a list of online resources, including academic e-Books and journal articles that provide reliable and accurate information on any topic.

Video Links: It has links to online videos that help you understand concepts from a variety of online resources.

LEADERSHIP KLEF

President
Er. Koneru Satyanarayana

Vice Chancellor
Dr. G. Pardha Saradhi Varma

Pro-Vice Chancellor
Dr. N. Venkatram

Registrar
Dr. K. Subbarao

CREDITS

Author
Ch. Phanikanth
Swathi Voddi

Director CDOE
C. Shanath Kumar

Instructional Designer
Nabina Das

Content Writer
K. Hima Bindu

Graphic Designer
B. Suchitra

First Edition, 2023.

Author's Profile

Ch. Phanikanth

Ch. Phanikanth is an accomplished Assistant Professor with a strong background in Computer Science & Engineering. With over 12 years of teaching experience, he specialises in various areas including Web Technologies, DBMS, Python, Computer Networks, and C Programming. Holding qualifications of B.E. and M.Tech, Ch. Phanikanth has been an active participant in workshops and faculty development programs, continuously enhancing his expertise. He has contributed significantly to the field, earning certifications in areas such as HTML5 Application development, Cyber Security, Cloud Computing, Big Data, and more. His research pursuits have resulted in notable publications, including research papers in esteemed journals.

Author's Profile

Swathi Voddi

Swathi Voddi is an Assistant Professor at KL University in the Department of CSE with 17-year long experience in academics and teaching. Her areas of specialisation include Machine Learning, Artificial Intelligence, Data Warehousing & Mining, which have earned her numerous accolades and recognition for her excellent delivery expertise. Currently pursuing a Ph.D. at JNTUK, Kakinada, her research focuses on Medical Data Analysis using Machine Learning. Her passion for research is evident through her 10 research publications in UGC, peer-reviewed, and ABDC journals. She actively participates in workshops, Faculty Development Programs (FDPs), and completes certification courses to stay updated with the latest developments in her field. With her expertise and dedication to advancing knowledge, she plays a crucial role in imparting valuable insights to her students and contributing to the academic community.

Operating Systems

Course Description

This course provides a comprehensive overview of operating system concepts and design, offering students an in-depth understanding of the key functionalities that drive modern computing. The course begins by exploring operating system types, core functionalities, and the architectural support for process management and virtualisation. Students will delve into process management, learning how processes are created, executed, and managed through APIs and direct execution.

The second module focuses on CPU scheduling techniques, including multi-level feedback, lottery scheduling, and multiprocessor scheduling. It also introduces the concept of concurrency, covering threads, common concurrency problems, and synchronisation mechanisms such as semaphores, condition variables, and event-based concurrency to manage parallel tasks efficiently.

In the third module, students will explore memory management, including memory virtualisation, address spaces, and translation techniques like paging and segmentation. They will also learn about free space management, advanced page table structures, and the intricacies of swapping mechanisms and policies to optimise system performance.

The final module covers persistence and file system implementation, focusing on how operating systems manage I/O devices, hard disk drives, and RAID systems. Topics such as file system structure, data integrity, protection, and distributed systems are introduced, providing a holistic view of data management in standalone and distributed environments.

By the end of the course, students will thoroughly understand how operating systems function, interact with hardware, and support software applications, preparing them for advanced studies or careers in system design and administration.

The course **'Operating Systems'** is divided into **four** modules.

MODULE 1 FUNDAMENTALS AND TYPES OF OPERATING SYSTEMS

What is an OS, Brief history, Functionalities of OS, Basics Computer System Architecture overview. Operating System Structures, Types of Different OS, Basic Oss: Batch, Multi-programmed batch, Timesharing, Real-Time OS (RTOS), Distributed OS.

MODULE 2 PROCESS MANAGEMENT AND CPU SCHEDULING

Processes: Definition, Process States, 5 state model, Process structure: PCB and components, Inter process Communication, Operations on Processes, Threads, CPU Scheduling: I/O burst cycle, Context Switching, Short Term, Long Term and Scheduling Criteria, Algorithms: First Come First Serve, Shortest Job First, Priority Scheduling, Round Robin.

MODULE 3 PROCESS SYNCHRONISATION, DEADLOCK CONTROL, AND I/O MANAGEMENT

Process Synchronisation: Critical Section Problem, Mutual Exclusion, Races, Semaphores, Classic Synchronisation Problems, Readers/Writers, Dining Philosophers. Deadlocks: Deadlocks and Starvation, System Model, Necessary Conditions for a deadlock, Mutual Exclusion, Hold and Wait, No Pre-emption, Circular wait, Resource Allocation Graphs, Handling Deadlocks, Prevention, Avoidance, Bankers Algorithm.. I/O Device Management, I/O Device Types and Characteristics.

MODULE 4 MEMORY MANAGEMENT, FRAGMENTATION, AND FILE SYSTEM ORGANISATION

Memory Management: Swapping, Multiple Partition-First Fit-Best Fit-Worst Fit, RAID and Data Redundancy. Fragmentation: Internal and External Fragmentation, Paging and Demand Paging, Page Replacement, Page Replacement Algorithms, Thrashing. File-System: File-System structure, Access Methods, Directory structure, File-System Implementation, Protection.

Table of Contents

OPERATING SYSTEMS

MODULE 1

Fundamentals and Types of Operating Systems

Module Description

This module introduces the fundamental concepts of Operating Systems (OS) and explores their key functions. It begins with an overview of what an OS is, a brief history, and its core functionalities. The module also covers the basics of computer system architecture and different types of OS structures.

Students will learn about various types of OS, including Batch, Multi-programmed, Time-sharing, Real-Time, and Distributed Systems.

Key topics include process management, where students will explore what processes are, process states, process control blocks (PCB), and how processes communicate. CPU scheduling methods like First-Come-First-Serve, Shortest Job First, Priority Scheduling, and Round Robin are explained, along with context switching and scheduling criteria.

The module also covers process synchronisation issues like the Critical Section Problem and solutions using semaphores. Students will learn about deadlocks, how they occur, and ways to prevent and avoid them, including the Banker's Algorithm.

Other topics include managing I/O devices, memory management techniques like partitioning and paging, handling fragmentation, and file system structures. This module provides a solid understanding of how operating systems manage resources and ensure smooth system operation.

The module consists of **two** units.

Unit 1.1: Introduction to Operating Systems and System Architecture

Unit 1.2: Types of Operating Systems

MODULE 1

Fundamentals and Types of Operating Systems

Unit 1

Introduction to Operating Systems and System Architecture

Unit Table of Contents

Unit 1.1 Introduction to Operating Systems and System Architecture

Aim

To provide foundations of operating systems, including their functions, historical evolution, and underlying system architecture, as well as the different structures used in modern OS design.

Instructional Objectives

This unit intends to:

- Define an Operating System (OS)
- List the key historical developments of Operating Systems
- Identify the primary functionalities of an Operating System
- Recall the basic components of Computer System Architecture
- Recognise different types of Operating System structures

Learning Outcomes

At the end of this unit, students are expected to:

- Explain the concept and role of an Operating System
- Describe the major historical milestones in the development of Operating Systems
- Summarise the key functionalities performed by an Operating System
- Discuss the basic components of Computer System Architecture and their relevance to Operating Systems
- Compare different types of Operating System structures and their applications

1.1.1 Introduction to Operating Systems

(a) What is an Operating System?

An Operating System (OS) is a software layer that acts as an intermediary between computer hardware and the user. It manages the computer's hardware resources, including the central processing unit (CPU), memory, storage, input/output devices, and network connections, and provides an environment for running software applications. Essentially, an OS ensures that the hardware and software can work together efficiently while making the computer usable by humans.

The main roles of OS include:

- **Resource Management**: The OS allocates system resources such as CPU time, memory, and storage to different applications and processes.
- **Process Management**: It manages running processes, assigns CPU time, and handles switching between processes.
- **Memory Management**: The OS allocates and tracks memory usage, ensuring efficient use of the system's RAM.
- **File Management**: It provides a system for storing, retrieving, and organising files on storage devices.
- **Device Management**: The OS communicates with external devices like printers, disk drives, and display screens through drivers.
- **Security and Access Control**: The OS enforces security policies and ensures that users and applications can only access resources they are authorised to use.

In simpler terms, an OS makes it possible for users to interact with a computer without needing to know how the hardware works internally. By managing both the hardware and software, the OS provides a user-friendly interface for executing commands, running applications, and interacting with files and devices.

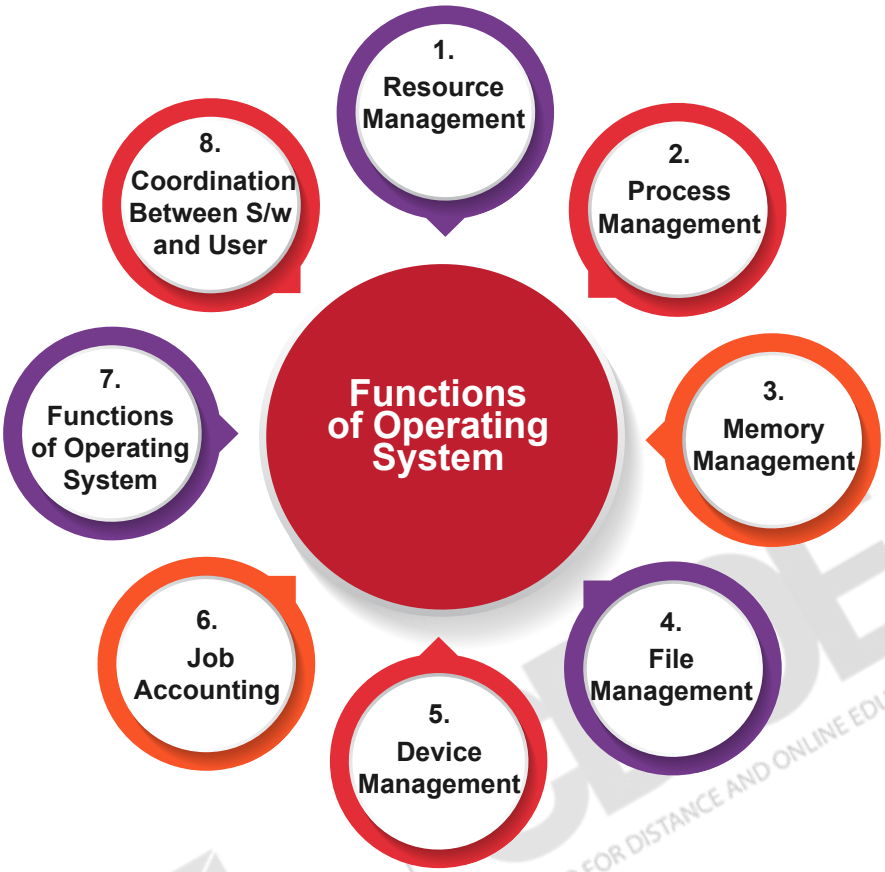


Fig. 1: Main roles of operating systems

Examples of popular operating systems include:

- **Windows**: Developed by Microsoft, Windows is one of the most widely used desktop and laptop operating systems.
- **Mac OS**: Apple's operating system for Mac computers, known for its user-friendly interface and seamless integration with Apple's ecosystem.
- **Linux**: An open-source OS commonly used in servers, embedded systems, and supercomputers.
- **Android and iOS**: These are the dominant mobile operating systems used in smart phones and tablets.

Fig. 2: Examples of operating systems

b) Brief history of operating systems

The development of operating systems has a rich history that spans over six decades, beginning in the early days of computing. Below is a brief timeline of major milestones in the evolution of operating systems:

1950s: Early Batch Systems

The earliest computers had no operating systems. Users had to interact with the hardware directly using switches and punched cards. As computers evolved, simple batch processing systems emerged. A batch system would run jobs (tasks) sequentially, one after another, with minimal human intervention.

1960s: Multiprogramming and Time-Sharing Systems

As computer hardware became more advanced, the concept of multiprogramming emerged, where multiple programs could be loaded into memory and run simultaneously. Time-sharing systems were also developed, allowing multiple users to access the system at the same time. This era saw the

development of UNIX, a powerful and influential operating system that laid the foundation for many modern systems.

1970s: Microcomputers and Personal Computing

The introduction of microprocessors in the 1970s paved the way for personal computers. Operating systems like CP/M (Control Program for Microcomputers) emerged, providing basic functionality for early personal computers. This period also saw the development of Microsoft's MS-DOS and Apple's Apple DOS.

1980s: Graphical User Interfaces (GUI)

In the 1980s, graphical user interfaces (GUIs) revolutionised how users interacted with computers. Instead of relying on command-line inputs, GUIs allowed users to interact with the system using visual elements like windows, icons, and menus. Apple's macOS and Microsoft's Windows were key players in the GUI revolution.

1990s: Networking and Internet Integration

The 1990s marked the rise of networked computers and the Internet. Operating systems began to integrate networking capabilities, allowing computers to connect to local area networks (LANs) and the global Internet. Microsoft Windows became the dominant OS for personal computers, while Linux gained popularity in server environments.

2000s and Beyond: Mobile and Cloud Computing

In the 21st century, the focus shifted towards mobile and cloud computing. Operating systems like Android and iOS dominated the mobile device market, while cloud-based systems and virtualisation became key trends in enterprise computing. Virtualisation allowed multiple operating systems to run on a single physical machine, revolutionising data centers and cloud services.

Today, operating systems continue to evolve with advancements in technology. They support a wide range of devices, from smart phones to supercomputers, and are integral to the functioning of modern society.

(c) Functionalities of an Operating System

An operating system performs several critical functions that are essential for the smooth operation of a computer system. The key functionalities include:

Process management

The OS manages processes, which are running instances of programs. It handles the creation, scheduling, and termination of processes, ensuring that each process gets appropriate CPU time.

It also manages inter-process communication and synchronisation to prevent conflicts between processes.

Memory management

The OS is responsible for managing the system's memory. It allocates memory to processes, tracks memory usage, and ensures that processes do not interfere with each other's memory spaces. It also handles virtual memory, allowing the system to use disk space as an extension of physical memory when RAM is insufficient.

File system management

Operating systems provide a hierarchical file system for storing and organising files. The OS handles the creation, deletion, and reading/writing of files, as well as managing access permissions to ensure data security.

Device management

The OS communicates with peripheral devices (such as printers, keyboards, and disk drives) through device drivers. It ensures that devices are correctly initialised and available for use by applications and users.

Security and access control

Modern operating systems enforce security policies to protect the system from unauthorised access. They control access to resources (such as files, devices, and memory) by implementing user authentication, permissions, and encryption mechanisms.

User interface

The OS provides a user interface that allows users to interact with the computer. This can be a command-line interface (CLI) or a graphical user interface (GUI), depending on the system.

Networking

The OS manages network connections, allowing the computer to communicate with other devices over a network. It handles protocols, routing, and data transmission, enabling the sharing of resources and information.

Error detection and handling

The OS monitors the system for errors, such as hardware failures or software crashes. It provides mechanisms for recovering from errors and logging error information for future troubleshooting.

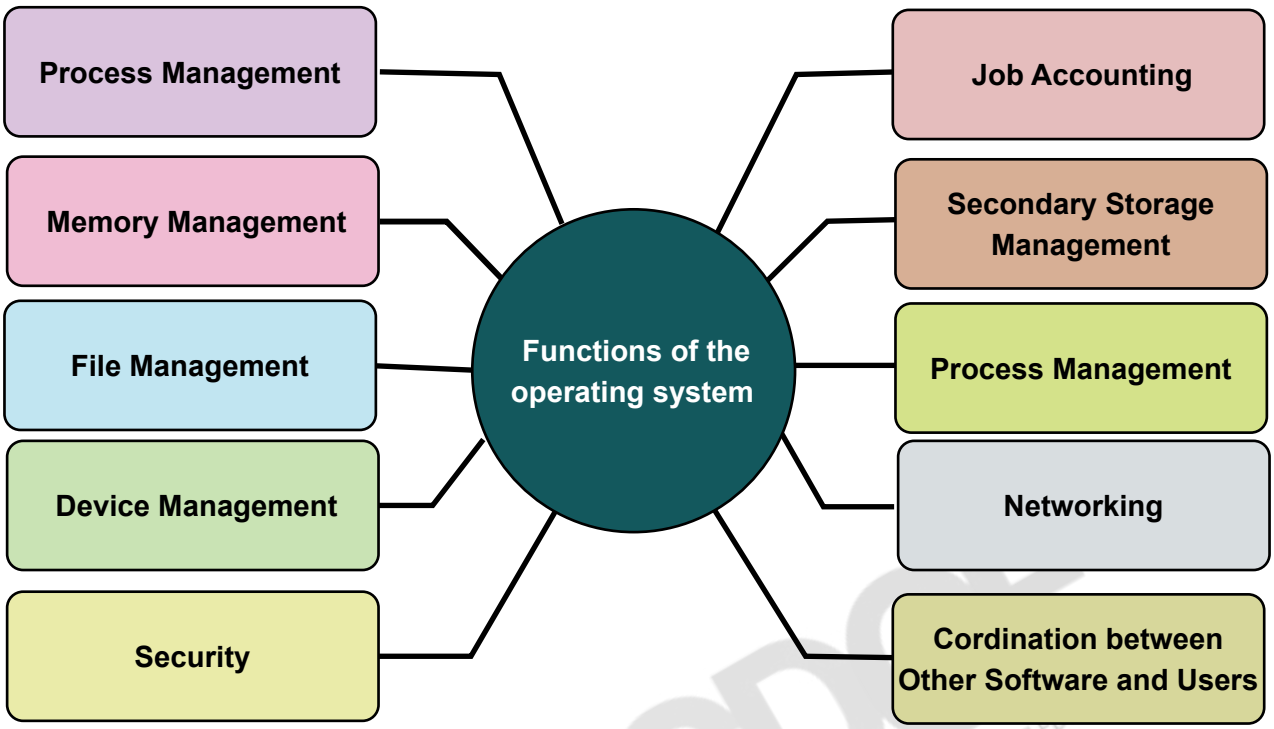


Fig. 3: Functionalities of an operating system

Self-Assessment Questions

1. What is the primary role of an operating system?

    a) To increase the speed of the CPU
    b) To manage hardware resources and provide a user interface
    c) To provide electricity to the computer
    d) To serve as a backup for the hard drive

2. Which of the following is NOT a function of an operating system?

    a) Process management
    b) Memory management
    c) Device management
    d) Hardware manufacturing

3. In the history of operating systems, which decade introduced graphical user interfaces (GUIs)?

    a) 1950s
    b) 1960s
    c) 1980s
    d) 1990s

4. What type of system allows multiple users to access a computer at the same time?

    a) Batch system
    b) Real-time system
    c) Time-sharing system
    d) Single-user system

5. Which of the following operating systems is open source and commonly used on servers?

    a) Windows
    b) Mac OS
    c) Linux
    d) iOS

1.1.2 Systems Architecture and OS Structures

(a) Basics of Computer System Architecture Overview

Computer system architecture refers to the design and organisation of the various hardware components of a computer system. It defines how these components interact and work together to execute programs and process data. The key components of computer architecture include:

Central Processing Unit (CPU)

The CPU is the brain of the computer, responsible for executing instructions and performing calculations. It consists of the control unit, arithmetic logic unit (ALU), and registers. The CPU fetches instructions from memory, decodes them, and executes them, cycling through these steps repeatedly.

Memory (RAM)

Random Access Memory (RAM) is used to store data and instructions that are actively being used by the CPU. It provides fast, temporary storage that the CPU can access quickly. However, RAM is volatile, meaning its contents are lost when the system is powered off.

nput/output (I/O) Devices

These devices allow the system to interact with the external world. Examples include keyboards, mice, printers, monitors, and disk drives. The OS manages these devices through drivers, ensuring that data can be transferred between the CPU and peripherals.

Storage

Storage devices, such as hard drives and solid-state drives, provide long-term data storage. They retain data even when the system is powered off. The OS organises storage using a file system, enabling users to save and retrieve files.

Buses

Buses are communication pathways that transfer data between the CPU, memory, and I/O devices. A system bus connects the CPU to the main memory, while peripheral buses connect the CPU to external devices. The OS plays a critical role in managing these components, ensuring that hardware resources are used efficiently and that applications can interact with the system without needing to know the details of how the hardware works.

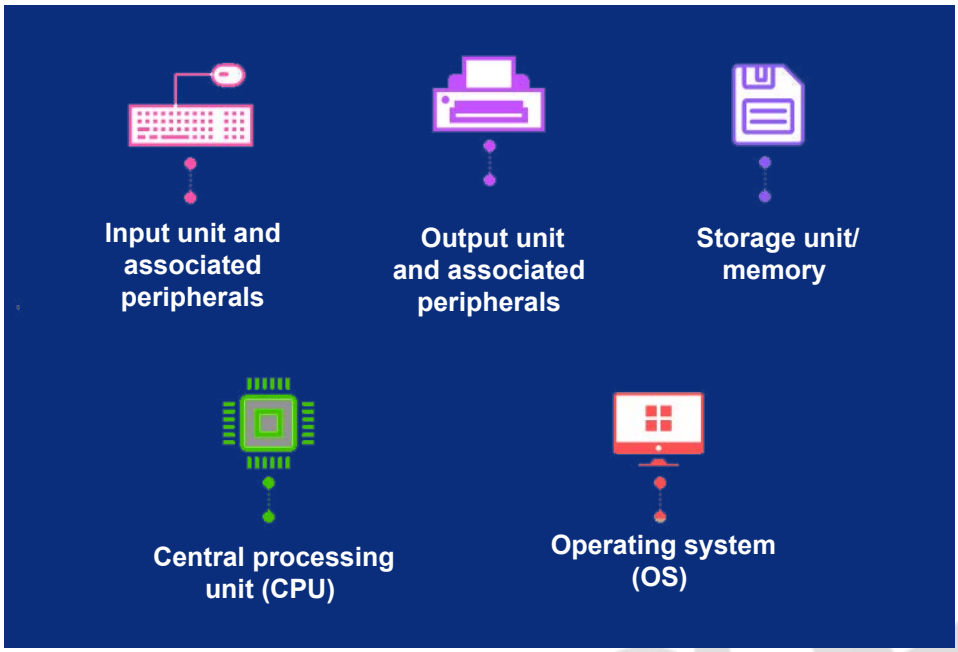


Fig. 4: Computer system architecture

(b) Operating System Structures

Operating system structures refer to the internal organisation and design of the OS. Different structures are used to organise the components of the OS, manage processes, and handle interactions with hardware.

Some common OS structures include:

(i) Monolithic systems

In monolithic operating systems, all the components of the OS are tightly integrated into a single layer. This structure offers high performance because all components are directly accessible. However, it can be difficult to maintain and debug, as changes to one component may affect the entire system. Examples of monolithic OSs include early versions of UNIX and MS-DOS.

(ii) Layered systems

In a layered OS structure, the system is divided into layers, each responsible for a specific function. The lower layers interact with hardware, while the upper layers interact with the user. This design allows for better organisation and modularity, making the system easier to maintain and extend. However, layered systems can be less efficient due to the overhead of communication between layers. The THE operating system is an example of a layered OS.

(iii) Micro kernels

In a microkernel OS, only the most essential services, such as process management and inter-process communication, are included in the kernel. Other services, such as device drivers and file systems, are run in user space as separate processes. This structure enhances security and stability, as failures in user-space services do not affect the kernel. Mach and MINIX are examples of microkernel-based systems.

(iv) Modular systems

Modular operating systems allow components to be loaded and unloaded dynamically as needed. This design provides flexibility and allows the OS to be tailored to specific requirements. Modern operating systems like Linux and Windows use a modular approach, where device drivers, file systems, and other components can be added or removed without restarting the system.

(v) Client-server systems

In a client-server structure, the OS is divided into client and server processes. Clients request services, and servers provide them. This structure is commonly used in distributed operating systems, where multiple computers interact over a network. The client-server model allows for scalability and the distribution of resources across multiple systems.

(vi) Virtual Machines (VM)

A virtual machine is a software emulation of a physical computer. The OS can be structured to support multiple virtual machines, each running its own operating system instance. Virtualisation allows multiple operating systems to run on a single physical machine, providing isolation between different environments. VMware and Hyper-V are popular examples of virtualisation platforms.

Self-Assessment Questions

6. What is the role of the Central Processing Unit (CPU) in a computer system?

    a) To store data temporarily
    b) To execute instructions and perform calculations
    c) To manage input/output devices
    d) To store long-term data

7. Which component of the computer system is responsible for providing fast, temporary storage for the CPU?

    a) Hard drive
    b) RAM
    c) I/O devices
    d) Registers

8. What is the main difference between RAM and storage devices such as hard drives?

    a) RAM is non-volatile, while hard drives are volatile
    b) Hard drives are used for temporary storage, while RAM is for long-term storage
    c) RAM is volatile, meaning it loses data when powered off, while hard drives retain data
    d) Hard drives can be accessed faster than RAM

9. Which type of operating system structure consists of tightly integrated components in a single layer?

    a) Microkernel
    b) Monolithic system
    c) Layered system
    d) Client-server system

10. What is a major advantage of a microkernel-based operating system?

    a) All components are loaded in a single layer
    b) It is highly efficient and fast
    c) Only essential services are included in the kernel, enhancing security and stability
    d) It avoids communication overhead between layers

Summary

- Operating System (OS) serves as a crucial interface between computer hardware and users, ensuring efficient management of resources like the CPU, memory, storage, and input/output (I/O) devices.
- It performs essential functions such as process management, memory management, file system management, and device management.
- The OS also enforces security and access control, provides a user interface, and manages networking and error detection.
- The evolution of operating systems spans over decades, from simple batch systems in the 1950s to modern systems that support mobile and cloud computing.
- Modern OS architectures can be structured in various ways, including monolithic systems, layered systems, microkernel architectures, modular systems, client-server systems, and virtual machines. Each structure offers different benefits in terms of performance, flexibility, and security.

Terminal Questions

1. Name three examples of operating systems.
2. What is the role of the CPU in a computer?
3. What is the function of RAM in a computer system?
4. What is the difference between input and output devices?
5. What is the purpose of file management in an OS?
6. How does the OS manage processes?

Answer Keys

Self-Assessment Questions	
Question No.	**Answer**
1	B
2	D
3	C
4	C
5	C
6	B
7	B
8	C
9	B
10	C

Activity

Activity Type: Offline **Duration: 1 day**

You are tasked with designing a simple monolithic operating system for a small computing device, such as an embedded system. In a monolithic OS, all the core functionalities (such as file management, process management, device drivers, and memory management) are integrated into a single layer.

Task:

1. List the key components and functionalities you would include in your monolithic OS design.
2. Explain the advantages and disadvantages of using a monolithic architecture for your OS in terms of performance, maintainability, and security.
3. Describe how you would handle errors or failures in one component, considering that the entire OS is tightly integrated.

Glossary

- **Process management:** OS function that manages the execution of programs by scheduling and allocating CPU time.
- **Memory management:** OS function that allocates and manages system memory (RAM) for processes.
- **File system management:** OS function that organises and controls access to files and storage devices.
- **Device management:** OS function that manages external devices via drivers.
- **Security and access control:** Mechanisms used by the OS to protect data and resources from unauthorised access.

Bibliography

Textbooks

- Stallings, W. (2013). *Operating systems* (9th ed.). Philadelphia, PA: Pearson Education.
- Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems* (4th ed.). Upper Saddle River, NJ: Pearson.

External Resources

- Russinovich, M. E., Solomon, D. A., & Ionescu, A. (2012). *Windows(r) Internals* (5th ed.). Redmond, WA: Microsoft Press.
- Love, R. (2012). *Linux Kernel Development* (3rd ed.). Boston, MA: Addison-Wesley Educational.

e-References

- **Introduction to operating systems:** https://www.geeksforgeeks.org/introduction-of-operating-system-set-1/
- **Systems architecture and OS structures:** https://www.prepbytes.com/blog/operating-system/operating-system-architecture/

Video Links

Video	Links
Introduction to operating systems	https://youtu.be/vBURTt97EkA
Systems architecture and OS structures	https://youtu.be/fvN98a_7AT4?list=PLBInK6fEyqRgRF-FUWec-0w4yWCurLy1t

Image Credits

- **Fig. 1:** Self-Made
- **Fig. 2:** Self-Made
- **Fig. 3:** Self-Made
- **Fig. 4:** Self-Made

Keywords

- Batch processing systems
- Time-sharing systems
- Graphical user interfaces
- Virtual memory
- Multiprogramming
- Punched cards

MODULE 1

Fundamentals and Types of Operating Systems

Unit 2

Types of Operating Systems

Unit Table of Contents

Unit 1.2 Types of Operating Systems

Aim

To explore foundational types of operating systems such as batch, multi-programmed, time-sharing, real-time, and distributed systems.

Instructional Objectives

This unit intends to:

- Define different types of operating systems, including single-user, multi-user, and network OS
- Identify the characteristics and functions of various types of operating systems
- List the advantages and disadvantages of each basic operating system type
- Recognise the historical significance and use cases of batch, time-sharing, and multi-programmed systems

Learning Outcomes

At the end of this unit, students are expected to:

- Explain the differences between single-user, multi-user, and network operating systems
- Discuss the unique features and functions of various types of operating systems
- Compare basic operating systems based on their performance and use cases
- Summarise the evolution and significance of foundational operating systems in modern computing

1.2.1 Types of Different Operating Systems

Operating systems (OS) are the backbone of any computer system, providing an interface between the hardware and the user or application software. Over time, various types of operating systems have been developed to serve different computational needs, ranging from personal computing to large-scale enterprise operations. This content explores some of the key types of operating systems and a few basic ones, such as batch systems, multi-programmed systems, time-sharing, real-time OS (RTOS), and distributed operating systems.

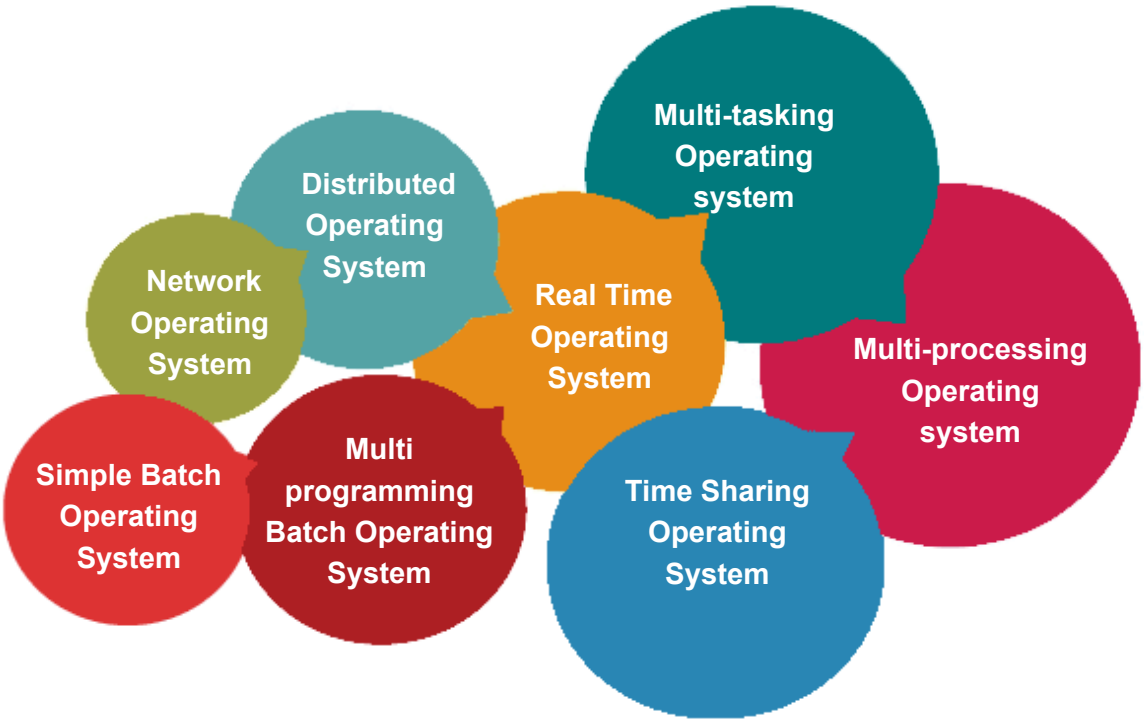


Fig. 1: Different operating systems

1. Single-user operating systems

A single-user operating system is designed to handle one task for one user at any given time. It focuses on maximising the efficiency of individual computing needs, primarily found in personal computing devices like desktops and laptops. This type of OS provides a graphical user interface

(GUI) that allows users to perform activities such as document editing, browsing the internet, and playing multimedia files.

Examples: Microsoft Windows, Mac OS, and Linux (for single-user configurations).

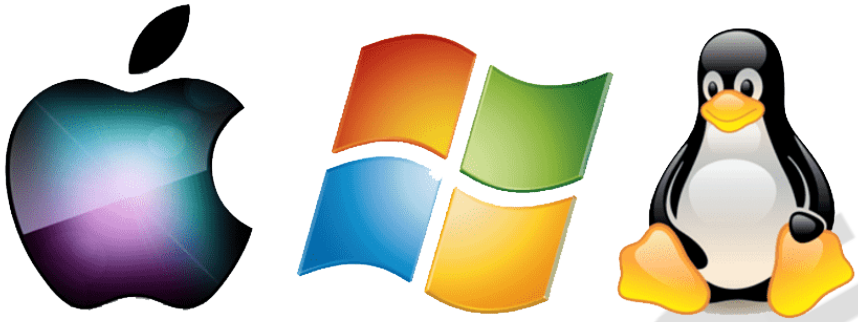


Fig. 2: Single-user operating systems

2. Multi-user operating systems

In contrast to single-user systems, multi-user operating systems allow multiple users to interact with the computer simultaneously. These systems are commonly used in large organisations where many people need to access the system at the same time without interfering with each other's work. Unix and Linux are well-known multi-user systems, often found in servers, where several users can log in remotely and perform tasks like file management, running programs, and utilising shared resources.

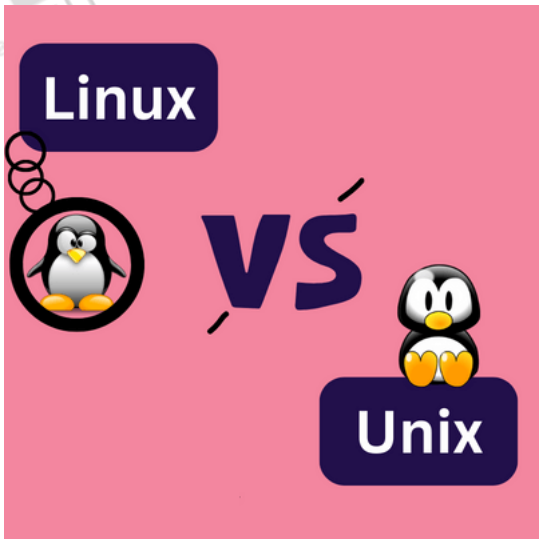


Fig. 3: Multi-user operating systems

3. Network Operating Systems (NOS)

A network operating system is designed to support networking functions. It allows computers to communicate and share resources such as files, applications, and hardware (like printers) over a local area network (LAN) or wide area network (WAN).

Examples: Novell NetWare, Windows Server, and UNIX. These OS types include advanced security features, network protocols, and resource management capabilities to maintain the integrity and security of shared data.

Fig. 4: Network Operating System (NOS)

4. Distributed operating systems

A distributed operating system manages a group of independent computers and makes them appear as if they are a single system to the user. These systems enable resource sharing, load balancing, and fault tolerance across a network of computers, ensuring efficient use of hardware and data redundancy in case of failure.

Examples: Amoeba, Plan 9, and the Google File System. Distributed OS is particularly useful for large organisations with extensive networked systems, such as cloud computing services.

5. Real-Time Operating Systems (RTOS)

A real-time operating system is designed to serve real-time applications that process data as it comes in, typically without buffer delays. These systems are characterised by a stringent requirement for response times, making them suitable for environments where precise timing is crucial.

Examples: VxWorks, QNX, and RTLinux. RTOS is commonly used in embedded systems, such as those found in robotics, medical devices, industrial control systems, and aviation, where predictable execution is essential for safety and performance.

Self-Assessment Questions

1. What is the primary focus of a single-user operating system?

    a) Allowing multiple users to perform tasks simultaneously
    b) Supporting networking functions for resource sharing
    c) Handling one task for one user at a time
    d) Managing a group of independent computers as a single system

2. Which of the following is an example of a single-user operating system?

    a) Windows Server
    b) Mac OS
    c) Unix
    d) Novell NetWare

3. What is the key characteristic of a multi-user operating system?

    a) It handles multiple users simultaneously.
    b) It allows resource sharing over a network.
    c) It processes data in real-time without delays.
    d) It is designed for personal computing.

4. Which of the following is an example of a multi-user operating system?

    a) Microsoft Windows
    b) Unix
    c) VxWorks
    d) QNX

5. A network operating system (NOS) is primarily designed to do what?

    a) Handle real-time tasks.
    b) Manage a group of independent computers as a single system.
    c) Support networking functions such as file and hardware sharing.
    d) Handle one task for one user at a time.

1.2.2 Basic Operating Systems

In addition to the broader categories of OS types discussed above, there are some basic and foundational operating systems that shaped the evolution of modern-day systems. These include batch systems, multi-programmed batch systems, time-sharing systems, real-time systems, and distributed systems, each offering unique functionalities to meet different computing needs.

1. Batch operating systems

Batch operating systems were one of the earliest types of operating systems. These systems are characterised by their ability to execute batches of jobs without manual intervention. In a batch system, jobs (which are tasks or processes) are collected in batches, and then executed sequentially by the OS. A user submits their job, often via punch cards or magnetic tape, and waits for the system to complete the task.

How batch OS works

- Jobs are prepared and placed in a queue.
- The OS picks up a job from the queue, loads it into memory, executes it, and then outputs the result.
- Once the job is completed, the next job is taken from the queue and the process continues.

Batch systems were common in the 1960s and 1970s and are still used today in environments where multiple users submit tasks that don't require user interaction. These systems are efficient at executing repetitive tasks, though they lack the flexibility of modern OSs in terms of user interaction and multitasking.

Advantages

- High efficiency for repetitive tasks.
- Effective utilisation of system resources.

Disadvantages

- No real-time user interaction.
- Debugging is difficult as errors are detected after the completion of a job.

Example

- IBM's early batch operating systems used for processing payrolls and financial data.

2. Multi-programmed batch systems

A multi-programmed batch system evolved from the basic batch systems by adding the ability to handle more than one program in memory at the same time. This OS allows several jobs to reside in memory at once, effectively improving system utilisation by switching between programs when one is waiting for I/O operations (such as reading from disk or printing output).

How multi-programming works

- When a program waits for an I/O operation, the OS suspends the program and loads another job from memory to execute.
- This ensures that the CPU is always busy and does not sit idle waiting for I/O.

Multi-programming is the foundation of modern multitasking. It increases system efficiency by using CPU resources effectively and handling multiple jobs concurrently. However, it does not provide the interactive computing experience that modern systems offer.

Advantages

- Maximises CPU utilisation by ensuring the CPU always has a task to perform.

Disadvantages

- The complexity of managing multiple jobs.
- No guarantee of timely execution (unsuitable for real-time tasks).

Example

- The IBM System/360 was one of the first widely-used systems to implement multi-programmed batch processing.

3. Time-sharing operating systems

A time-sharing operating system allows multiple users to access a computer system concurrently by sharing the processing time of the CPU. In a time-sharing environment, the CPU is allocated to each user's job for a short period (known as a time slice or quantum), then it moves to the next job in a round-robin fashion. This creates the illusion that the users have dedicated access to the system.

How time-sharing works

- Each job gets a small fraction of the CPU's time.
- The system switches between jobs so quickly that users perceive the system as providing them exclusive access.

Time-sharing systems were critical in the evolution of interactive computing, allowing multiple users to work on the same machine remotely. These systems introduced features like virtual memory and preemptive multitasking, which are common in modern OSs.

Advantages

- Efficient use of system resources for multiple users.
- Enhanced user interaction and multitasking.

Disadvantages

- Complex to implement and manage.
- A slower response for some users due to sharing of CPU time.

Examples

- Unix is a widely-used time-sharing system.
- Mainframes that allowed terminal access to multiple users are also an example of time-sharing systems.

4. Real-Time Operating Systems (RTOS)

A real-time operating system provides immediate processing and response to external stimuli, which is crucial for time-sensitive applications. RTOS is designed to run applications where timing is critical, such as in systems that control machinery, medical devices, or other processes requiring immediate and predictable responses. These systems must meet strict deadlines, and any delay could result in failure or disaster.

How RTOS works

- Tasks are prioritised based on urgency, and high-priority tasks are executed immediately, while lower-priority tasks are deferred.
- RTOS can be classified as Hard RTOS (where missing a deadline could cause catastrophic failure) or Soft RTOS (where deadlines are important but not as strict).

Advantages

- Precise timing and predictable behaviour.
- Suitable for systems requiring immediate response.

Disadvantages

- Limited flexibility compared to general-purpose OS.
- Often requires specialised hardware.

Examples

- Embedded systems in automotive control units.
- Flight control systems in airplanes.

5. Distributed operating systems

A distributed operating system controls a group of separate computers, allowing them to operate as a single cohesive unit. These systems ensure resource sharing across multiple machines and distribute computing tasks for better efficiency and fault tolerance. Distributed OSs enable collaboration across multiple devices, which makes them highly useful in areas like cloud computing, where computational tasks are distributed among different servers.

Advantages

- Efficient resource sharing and load balancing.
- High fault tolerance due to the distributed nature of tasks.

Disadvantages

- High complexity in managing distributed resources.
- Security concerns due to resource sharing over networks.

Examples

- Google's cloud services are supported by a distributed OS.
- Hadoop Distributed File System (HDFS) is another example used for processing big data.

Fig. 5: Basic operating systems

Self-Assessment Questions

6. What is a key characteristic of a batch operating system?

a) Jobs are executed interactively.
b) Jobs are processed in batches without user interaction.
c) The OS focuses on real-time task management.
d) Users have dedicated time slices for processing tasks.

7. How does a batch operating system handle tasks?

a) It processes jobs interactively with real-time feedback.
b) It collects jobs in a queue and executes them sequentially.
c) It assigns each user a dedicated time slice.
d) It processes multiple jobs simultaneously.

8. What is a disadvantage of a batch operating system?

a) It offers high efficiency for repetitive tasks.
b) It allows no real-time user interaction.
c) It requires user interaction for every job submission.
d) It provides a complex user interface.

9. Which of the following was a typical use case for batch operating systems?

a) Embedded systems
b) Interactive gaming
c) Payroll processing
d) Web hosting

10. How does a multi-programmed batch system improve over a basic batch system?

a) By allowing multiple users to access the system simultaneously.
b) By handling several programs in memory concurrently.
c) By providing dedicated time slices for each job.
d) By guaranteeing real-time task execution.

Summary

- Operating systems (OS) are essential to the functionality of any computing system, serving as an intermediary between hardware and user applications.
- Various types of operating systems have been developed to address different computing environments and needs.
- Single-user OS, such as Windows and macOS, are commonly used in personal computers, while multi-user OS, like Unix and Linux, are designed for environments where multiple users need access simultaneously.
- Network operating systems (NOS) enable resource sharing across local and wide area networks, while distributed OS manage multiple interconnected computers to act as a unified system, enhancing efficiency and fault tolerance.
- Real-time operating systems (RTOS) provide immediate, predictable responses, critical for applications in medical devices, industrial control, and aviation.
- The concept of basic operating systems, including batch systems, multi-programmed batch systems, and time-sharing systems, highlights the historical evolution of modern OS.
- Batch systems focus on sequential job execution without user interaction, while multi-programmed batch systems introduced multi-tasking to improve CPU utilisation.
- Time-sharing OS enables concurrent access for multiple users by rapidly switching between tasks, creating the illusion of individual access.
- Real-time and distributed systems continue to advance OS functionality for specialised and large-scale computational environments.

Terminal Questions

1. What are the key differences between single-user and multi-user operating systems?
2. How do network operating systems (NOS) manage resource sharing across networks?
3. What are the main features of a distributed operating system, and where are they commonly used?
4. Why is a real-time operating system (RTOS) crucial in time-sensitive applications?
5. How did batch and multi-programmed batch systems contribute to the development of modern operating systems?
6. What are the advantages and limitations of time-sharing systems in managing multiple users?

Answer Keys

Self-Assessment Questions	
Question No.	**Answer**
1	C
2	B
3	A
4	B
5	C
6	B
7	B
8	B
9	C
10	B

Activity

Activity Type: Offline **Duration: 1 day**

Design of an RTOS for an autonomous vehicle requires modular architecture with components like task management, memory allocation, and I/O handling.

Glossary

- **Kernel –** The core component of an operating system that manages system resources and facilitates communication between hardware and software.
- **Multitasking –** The ability of an operating system to execute multiple tasks (processes) simultaneously by switching between them.
- **Thread –** The smallest unit of processing that can be scheduled by the operating system. A process may contain multiple threads.
- **Paging –** A method used by virtual memory systems to manage how data is transferred between the RAM and the disk.
- **Swapping –** A memory management technique where processes are swapped in and out of main memory to maximise the efficiency of RAM usage.

Bibliography

Textbooks

- Stallings, W. (2013). *Operating systems* (9th ed.). Philadelphia, PA: Pearson Education.
- Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems* (4th ed.). Upper Saddle River, NJ: Pearson.

External Resources

- Russinovich, M. E., Solomon, D. A., & Ionescu, A. (2012). *Windows(r) Internals* (5th ed.). Redmond, WA: Microsoft Press.
- Love, R. (2012). *Linux Kernel Development* (3rd ed.). Boston, MA: Addison-Wesley Educational.

e-References

- **Types of different operating systems:** https://www.geeksforgeeks.org/types-of-operating-systems/
- **Basic operating systems:** https://www.javatpoint.com/operating-system

Video Links

Video	Links
Types of different operating systems	https://youtu.be/s-tk8el8JoU

Image Credits

- **Fig. 1:** Self-Made
- **Fig. 2:** Self-Made
- **Fig. 3:** Self-Made
- **Fig. 4:** Self-Made
- **Fig. 5:** Self-Made

Keywords

- Scheduler
- Context switching
- Deadlock
- Semaphore
- Critical section

OPERATING SYSTEMS

MODULE 2

Process Management and CPU Scheduling

Module Description

This module provides a comprehensive introduction to process management and CPU scheduling in operating systems. It begins with the definition and core concepts of a process, covering its structure, states, and lifecycle within a system. The 5-state process model is explored, outlining key stages such as new, ready, running, blocked, and terminated, with emphasis on how the operating system transitions between these states.

Next, the module examines the Process Control Block (PCB) and its components, highlighting its role in process management and execution. Additionally, the fundamentals of Inter-Process Communication (IPC) are introduced, explaining how processes communicate and coordinate to ensure smooth system operation.

The module also covers essential operations on processes like creation, termination, and synchronisation, alongside the concept of threads, both user and kernel-level, to illustrate multitasking within processes.

A key focus of the module is CPU scheduling, where students learn about the I/O burst cycle, context switching, and various scheduling algorithms. Different types of scheduling are covered, including short-term and long-term scheduling, and criteria for optimal CPU scheduling are discussed. The module concludes by exploring common scheduling algorithms like First Come First Serve (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin, providing insights into how modern systems handle process execution and resource allocation efficiently.

The module consists of two units.

Unit 2.1: Process and Inter-Process Communication

Unit 2.2: CPU Scheduling

MODULE 2

Process Management and CPU Scheduling

Unit 1

Process and Inter-Process Communication

Unit Table of Contents

Unit 2.1 Process and Inter-Process Communication

Aim

To explore and implement efficient process management and inter-process communication mechanisms to enhance the performance and reliability of software systems.

Instructional Objectives

This unit intends to:

- Identify the key concepts and definitions related to processes and inter-process communication (IPC)
- Describe the various types of processes and their states in a computing environment
- Explain the principles of inter-process communication and the mechanisms used for IPC

Learning Outcomes

At the end of this unit, students are expected to:

- Analyse the roles and functionalities of processes and threads in a multi-tasking environment
- Evaluate the effectiveness of different inter-process communication methods, such as message passing and shared memory, in various scenarios
- Compare and contrast the performance implications of using threads versus processes in application development

2.1.1 Processes

Introduction

In modern computing systems, processes and inter-process communication (IPC) play crucial roles in resource management and execution. This unit delves into the fundamentals of processes, their lifecycle, and the structure that defines them, along with how processes communicate and coordinate. We will also explore threads, an essential component of concurrent execution in multitasking environments.

A process is a running instance of a program, encompassing the program's code, data, and its execution state. It is a fundamental concept in operating systems, as processes enable the execution of multiple programs on a system. The operating system manages all processes, allocating resources and managing execution to ensure that programs run efficiently and without conflicts.

(a) Definition of a process

A process is essentially an abstraction of a running program, comprising the program code, the current activity (represented by the value of the program counter and registers), the process stack, and the data section that includes global variables. It is more than just the program code (often called the text section); it includes everything required for its execution, such as the following:

1. **Text section**: The code of the program.
2. **Data section**: Global variables used in the program.
3. **Stack section**: Temporary data, including function parameters, return addresses, and local variables.
4. **Heap section**: Dynamically allocated memory used by the process during runtime.

Processes are fundamental to multitasking, allowing several processes to share the CPU and other system resources simultaneously. However, the processes themselves are isolated from each other and run in separate address spaces, ensuring that one process cannot interfere with another unless explicitly allowed through inter-process communication mechanisms.

(b) Process states

A process goes through several states during its lifecycle, which the operating system manages. These states help in organising the scheduling and execution of processes. The typical states of a process are:

1. **New**: The process is being created.
2. **Ready**: The process is waiting to be assigned to the processor.
3. **Running**: Instructions are being executed.

4. **Waiting/Blocked**: The process is waiting for some event (such as I/O completion) to occur.
5. **Terminated**: The process has finished execution.

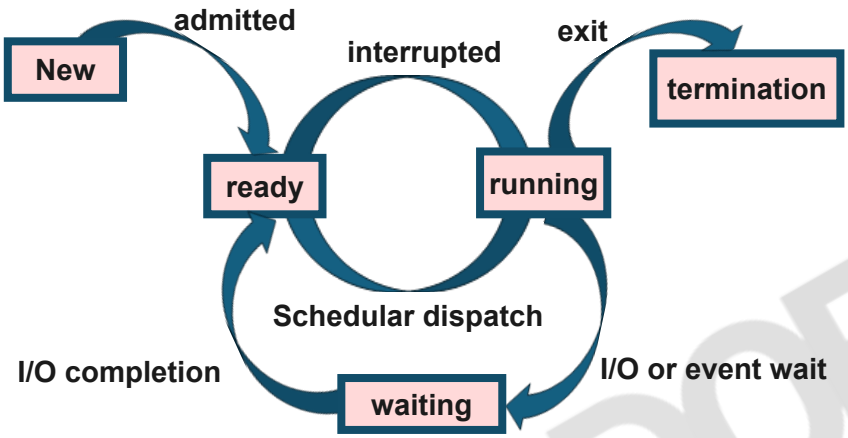


Fig. 1: Process states

(c) 5-State model

The 5-state process model is an extension of the simple 3-state model, providing more granularity and flexibility in the process management lifecycle. In addition to the basic states (New, Ready, Running, and Terminated), the 5-state model introduces a Blocked state.

Here is an overview of the five states in the model:

1. **New**: A proce**ss is being created and initialised, but is not yet ready to execute.**
2. **Ready:** Once the process is created, it enters the ready state, meaning it is prepared to execute, but is waiting for CPU allocation.
3. **Running:** The process is currently being executed by the CPU.
4. **Blocked (or Waiting):** If a process requires an external event, such as completing an I/O operation, it enters the blocked state.
5. **Terminated:** The process has completed its execution and is awaiting cleanup by the operating system.

The transition between these states occurs based on events like system interrupts, I/O operations, and scheduling decisions made by the operating system.

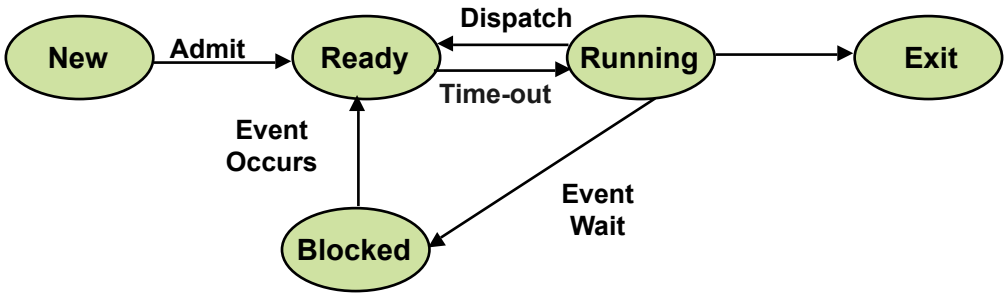


Fig. 2: 5-State model

(d) Process structure: PCB (Process Control Block) and components

The Process Control Block (PCB) is a critical data structure in operating systems that stores all the information about a process. Every process in the system has its own PCB, which the operating system uses to manage and track the process. The PCB contains information like:

1. **Process State**: The current state of the process (new, ready, running, etc.).
2. **Process ID (PID):** A unique identifier for each process.
3. **Program Counter:** The address of the next instruction to be executed.
4. **CPU Registers:** Contents of all process-related CPU registers.
5. **Memory Management Information:** Information about the memory used by the process, such as base and limit registers or page tables.
6. **I/O Status Information**: Information on the I/O devices allocated to the process.
7. **Accounting Information**: Includes process execution time, priority, and other related statistics.

The PCB allows the operating system to switch between processes efficiently by saving the state of a currently running process and restoring the state of another process during a context switch. It enables multitasking by storing and managing the execution state of multiple processes.

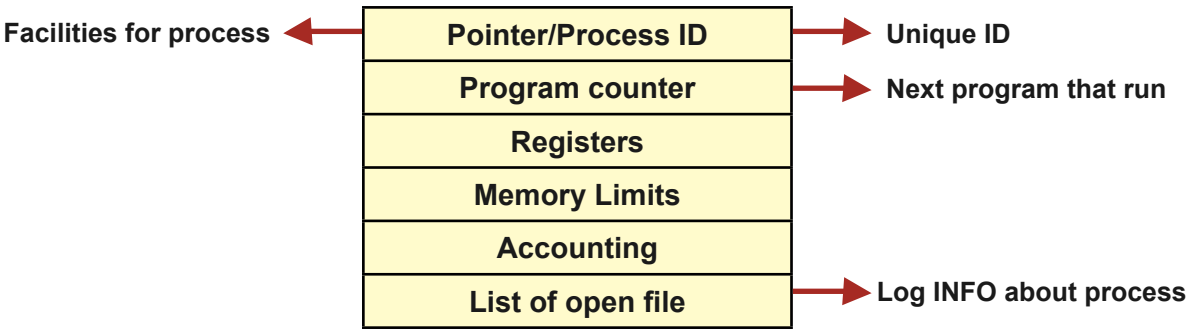


Fig. 3: PCB (Process Control Block)

(e) Inter-Process Communication (IPC)

In a multitasking environment, processes often need to communicate with each other. Inter-Process Communication (IPC) is the mechanism through which this communication occurs. IPC facilitates data sharing, synchronisation, and coordination between processes.

Processes can communicate in two main ways:

1. **Shared Memory**: Processes can share a portion of memory where they can exchange information. This is typically faster but requires synchronisation mechanisms like semaphores or mutexes to prevent race conditions.
2. **Message Passing**: Processes send and receive messages to communicate. This method does not require shared memory and is generally safer but may have higher overhead compared to shared memory.

The key aspects of IPC include:

1. **Synchronisation**: Processes must synchronise to ensure they access shared resources correctly. Semaphores and locks are common synchronisation tools.
2. **Communication**: Message queues, pipes, and sockets are common IPC mechanisms for sending and receiving messages between processes.

(f) Operations on processes

The operating system provides several operations that can be performed on processes to manage their lifecycle. These operations include

1. **Process Creation**: A new process is created using system calls like fork() in Unix/Linux. A parent process can create a child process.
2. **Process Termination**: A process can terminate normally after completing its task or be terminated prematurely using system calls like exit().
3. **Process Suspension/Resumption**: Processes can be suspended and resumed by the operating system to manage CPU usage efficiently.
4. **Process Synchronisation**: Processes must be synchronised when accessing shared resources to avoid conflicts and ensure data integrity.

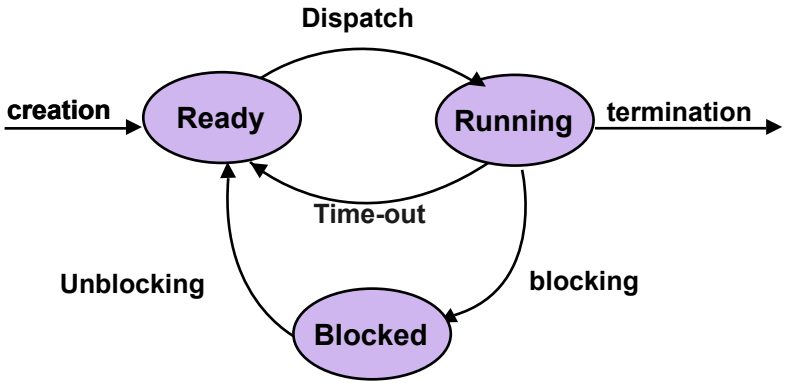


Fig. 4: Operations on processes

Process creation is a key aspect of multitasking. When a process creates another process, it is called the parent process, and the new one is called the child process. The child inherits certain attributes from the parent, such as environment variables and open files.

In Unix-like operating systems, the fork() system call creates a new process by duplicating the existing one. The child process can then execute the same or different code than the parent, depending on the program's logic.

Similarly, process termination can be initiated voluntarily or involuntarily. A process may finish execution and call the exit() system call to terminate. Alternatively, a process may be terminated by another process (e.g., by a parent terminating a child process) using system calls like kill().

Self-Assessment Questions

1. What is a process in an operating system?

    a) A section of memory
    b) A running instance of a program
    c) A type of file
    d) A hardware component

2. Which of the following is NOT part of a process structure?

    a) Text section
    b) Stack section
    c) Heap section
    d) File section

3. In the 5-state process model, which state represents a process that is waiting for an external event to occur?

    a) New
    b) Ready
    c) Running
    d) Blocked

4. What is the function of the Process Control Block (PCB)?

    a) Manages file systems
    b) Stores all information about a process
    c) Controls network connections
    d) Allocates memory to devices

5. Which of the following components is NOT stored in a Process Control Block (PCB)?

    a) Process ID
    b) CPU Registers
    c) Process Execution Time
    d) Network Configuration

2.1.2 Threads

A thread is the smallest unit of execution within a process. While a process may contain multiple threads, each with its own execution path, all threads within the same process share the same memory space. Threads are lightweight, allowing for more efficient use of resources and faster context switching compared to processes.

Definition of Threads

A thread represents a single sequence of instructions within a process. A thread, also known as a lightweight process, includes the thread ID, program counter, register set, and stack. However, threads share certain parts of the process, such as code, data, and files. This shared nature makes threads efficient for parallel execution.

Threads are useful when a task can be broken into smaller, independent subtasks that can run simultaneously. For instance, in a web browser, different threads can be responsible for rendering web pages, downloading files, and handling user input, all running concurrently within the same process.

Types of Threads

Threads can be classified into two main types:

1. **User-Level Threads (ULT)**: These threads are managed by user-level libraries rather than the operating system. The kernel is unaware of the existence of these threads. As a result, thread management tasks such as creation, scheduling, and synchronisation are handled by the user-level library.

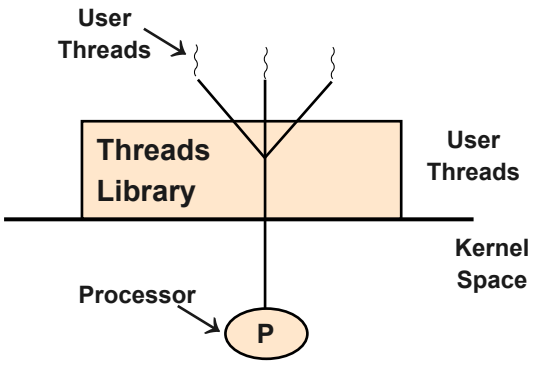


Fig. 5: User-Level Threads (ULT)

2. **Kernel-Level Threads (KLT)**: These threads are managed directly by the operating system's kernel. The kernel is responsible for scheduling and managing the threads. Since the kernel has complete control, kernel-level threads offer better performance in certain scenarios but are more resource-intensive.

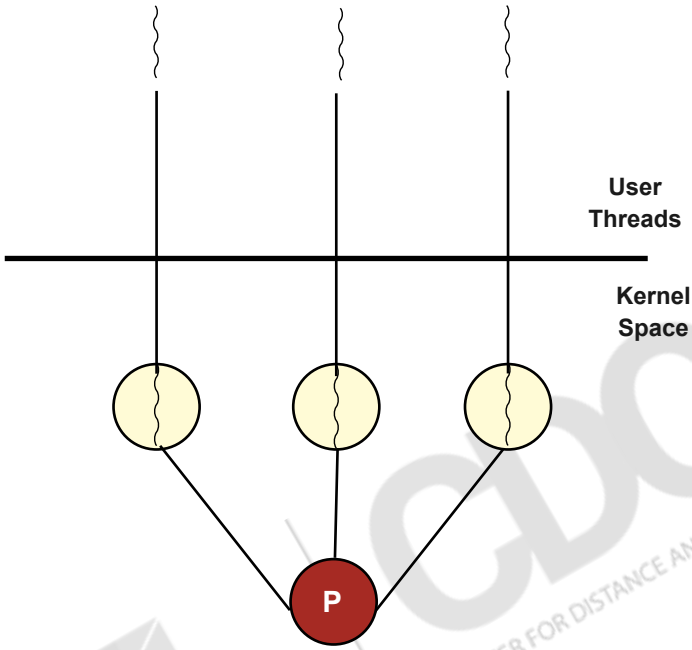


Fig. 6: Kernel-Level Threads (KLT)

Threads enable multithreading, where multiple threads run concurrently within a single process, improving the efficiency and performance of applications. Multithreading is particularly beneficial in situations where tasks can be parallelised, such as in server applications, where multiple clients may be served simultaneously.

The fundamental concepts of processes and inter-process communication (IPC). A process is a crucial element in operating systems, representing the execution of a program. The process control block (PCB) plays a vital role in managing the lifecycle of a process. Moreover, inter-process communication facilitates coordination between processes, enabling data sharing and synchronisation.

Threads, as lightweight processes, allow for parallel execution within a process, improving performance and responsiveness in multitasking systems. Understanding processes, threads, and IPC is essential for grasping how modern operating systems manage and execute multiple programs efficiently.

This foundational knowledge sets the stage for more advanced topics such as CPU scheduling, memory management, and file system organisation, which we will explore in subsequent units.

Self-Assessment Questions

6. What is a thread in the context of an operating system?

   a) A lightweight process that runs in isolation from others
   b) The smallest unit of execution within a process
   c) A method for communication between processes
   d) A type of memory management system

7. What parts of a process are shared by all threads within that process?

   a) Program counter and register set
   b) Stack and thread ID
   c) Code, data, and files
   d) Stack and heap

8. Which of the following is true about User-Level Threads (ULT)?

   a) They are managed by the operating system kernel
   b) The kernel is unaware of their existence
   c) They require more resources compared to Kernel-Level Threads (KLT)
   d) They offer better performance than Kernel-Level Threads (KLT) in all scenarios

9. Which of the following best describes Kernel-Level Threads (KLT)?

   a) They are managed entirely by user libraries
   b) They are invisible to the operating system
   c) They are more resource-efficient than User-Level Threads (ULT)
   d) They are managed and scheduled by the operating system kernel

10. What is one advantage of using threads over processes in a multitasking environment?

    a) Threads do not share memory, making them more secure
    b) Threads are lighter in terms of resource usage and allow faster context switching
    c) Threads are isolated from each other, which reduces errors
    d) Threads always run independently of each other

Summary

- The fundamental concepts of processes, inter-process communication (IPC), and threads in modern operating systems. It begins by defining a process as a running instance of a program, explaining the key components such as the text, data, stack, and heap sections.
- The process lifecycle is explained through states like new, ready, running, blocked, and terminated, with the 5-state model adding clarity to process transitions.
- A critical aspect of process management is the Process Control Block (PCB), which stores essential information about each process, such as its current state, process ID, program counter, CPU registers, and memory management details.
- The PCB enables smooth multitasking by allowing the operating system to switch between processes efficiently.
- Inter-Process Communication (IPC), which allows processes to communicate and coordinate for resource sharing and synchronisation.
- Two key methods of IPC are shared memory and message passing, both of which enable processes to exchange data securely and efficiently.
- Threads share the same memory space but have their own execution paths, making them more efficient for tasks that can be parallelised.
- User-level threads and kernel-level threads are distinguished by whether the operating system or a user-level library manages them.
- This foundational knowledge prepares learners to explore more advanced topics such as CPU scheduling, memory management, and file systems in later units.

Terminal Questions

1. What is a Process Control Block (PCB), and what information does it store?
2. Differentiate between shared memory and message passing in Inter-Process Communication (IPC).
3. Describe the difference between user-level threads and kernel-level threads.
4. Why is multithreading beneficial in modern operating systems? Provide an example of a situation where threads are useful.
5. Discuss how processes are created and terminated in Unix-like operating systems using system calls such as fork() and exit().

Answer Keys

Self-Assessment Questions	
Question No.	**Answer**
1	B
2	D
3	D
4	B
5	D
6	B
7	C
8	B
9	D
10	B

Activity

Activity Type: Offline **Duration: 1 day**

Analyse different mechanisms of Inter-Process Communication (IPC) and their use in coordinating processes in an operating system.

Task

1. Research three different IPC mechanisms: Pipes, Message Queues, and Shared Memory.
2. For each IPC mechanism, answer the following:
    - How does the mechanism work?
    - In what scenarios is this mechanism most useful?
    - What are the advantages and limitations of using this mechanism for communication between processes?

Glossary

- **Shared memory:** A method of IPC where processes share a portion of memory for communication, requiring synchronisation mechanisms.
- **Message passing:** A method of IPC where processes exchange messages to communicate without shared memory.
- **Fork():** A system call in Unix-like systems that creates a new process by duplicating an existing one.
- **Exit():** A system call used by a process to terminate execution in Unix-like systems.
- **Context switching:** The process of switching the CPU from executing one process to another by saving the state of the current process and restoring the state of the next process.
- **Multithreading:** The use of multiple threads within a single process to perform parallel tasks, improving performance and efficiency.

Bibliography

Textbooks

- Stallings, W. (2013). *Operating systems* (9th ed.). Philadelphia, PA: Pearson Education.
- Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems* (4th ed.). Upper Saddle River, NJ: Pearson.

External Resources

- Russinovich, M. E., Solomon, D. A., & Ionescu, A. (2012). *Windows(r) Internals* (5th ed.). Redmond, WA: Microsoft Press.
- Love, R. (2012). *Linux Kernel Development* (3rd ed.). Boston, MA: Addison-Wesley Educational.

e-References

- **Process and inter-process communication :** https://shorturl.at/KSwKf

Video Links

Video	Links
Process and inter-process communication	https://youtu.be/dJuYKfR8vec
Processes and threads	https://youtu.be/OrM7nZcxXZU

Image Credits

- **Fig. 1:** Self-Made
- **Fig. 2:** Self-Made
- **Fig. 3:** Self-Made
- **Fig. 4:** Self-Made
- **Fig. 5:** Self-Made
- **Fig. 6:** Self-Made

Keywords

- Synchronisation
- Lightweight process
- Concurrent execution
- Blocking state
- Ready state
- Terminated state
- Process creation

MODULE 2

Process Management and CPU Scheduling

Unit 2

CPU Scheduling

Unit Table of Contents

Unit 2.2 CPU Scheduling

Aim

To analyse the fundamentals and algorithms of CPU scheduling to optimise process execution and resource allocation in operating systems.

Instructional Objectives

This unit intends to:

- Describe the different types of CPU scheduling algorithms, including preemptive and non-preemptive scheduling
- List the advantages and disadvantages of various CPU scheduling algorithms such as FCFS, SJF, RR, and Priority scheduling
- Explain the criteria used to evaluate CPU scheduling algorithms, such as turnaround time, waiting time, and CPU utilisation

Learning Outcomes

At the end of this unit, students are expected to:

- Evaluate the impact of various CPU scheduling strategies on system efficiency and user experience in multi-tasking environments
- Compare and contrast the effectiveness of preemptive versus non-preemptive scheduling algorithms in managing process execution
- Assess the applicability of specific CPU scheduling algorithms in real-world scenarios, considering factors like workload characteristics and system architecture

2.2.1 Fundamentals of CPU Scheduling

In modern computing systems, efficient management of the central processing unit (CPU) is essential to ensure optimal performance, resource utilisation, and system responsiveness. CPU scheduling is a core concept in operating systems, responsible for determining which processes or threads should be executed by the CPU at any given time. It plays a critical role in managing the order of execution for processes in a multitasking environment, ensuring fair resource allocation, reducing wait times, and improving throughput.

As a multitasking system runs multiple processes concurrently, the CPU must switch between them efficiently. This switching process requires careful decision-making, where various scheduling algorithms are employed to determine the sequence of process execution. The performance and responsiveness of an operating system largely depend on these algorithms, which aim to balance factors such as CPU utilisation, response time, throughput, and waiting time.

Fundamentals of CPU scheduling, including the I/O burst cycle, context switching, different types of schedulers (short-term and long-term), scheduling criteria, and commonly used scheduling algorithms such as First Come First Serve (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin. Understanding these concepts is crucial for improving the efficiency and fairness of process execution in any operating system.

(a) I/O burst cycle

The I/O burst cycle is a key concept in CPU scheduling that represents the alternating phases of a process between CPU execution and I/O operations. In a multitasking environment, processes typically alternate between performing computations (CPU bursts) and waiting for I/O operations (I/O bursts). This behavior is cyclical: processes spend time executing instructions on the CPU and then request data from I/O devices (e.g., reading a file from disk), during which they cannot use the CPU.

A typical process might exhibit multiple bursts of CPU activity interleaved with periods of I/O activity. The length and frequency of these bursts depend on the nature of the task. CPU-bound processes spend most of their time doing computations, meaning they have long CPU bursts and fewer I/O bursts. In contrast, I/O-bound processes often wait on I/O devices, so their CPU bursts are short, and I/O bursts dominate their activity.

Understanding the I/O burst cycle is crucial for developing efficient CPU scheduling algorithms. The CPU scheduler attempts to maximise system performance by allocating CPU time between processes based on their I/O burst cycles, ensuring that the CPU is kept busy while I/O operations are completed in the background.

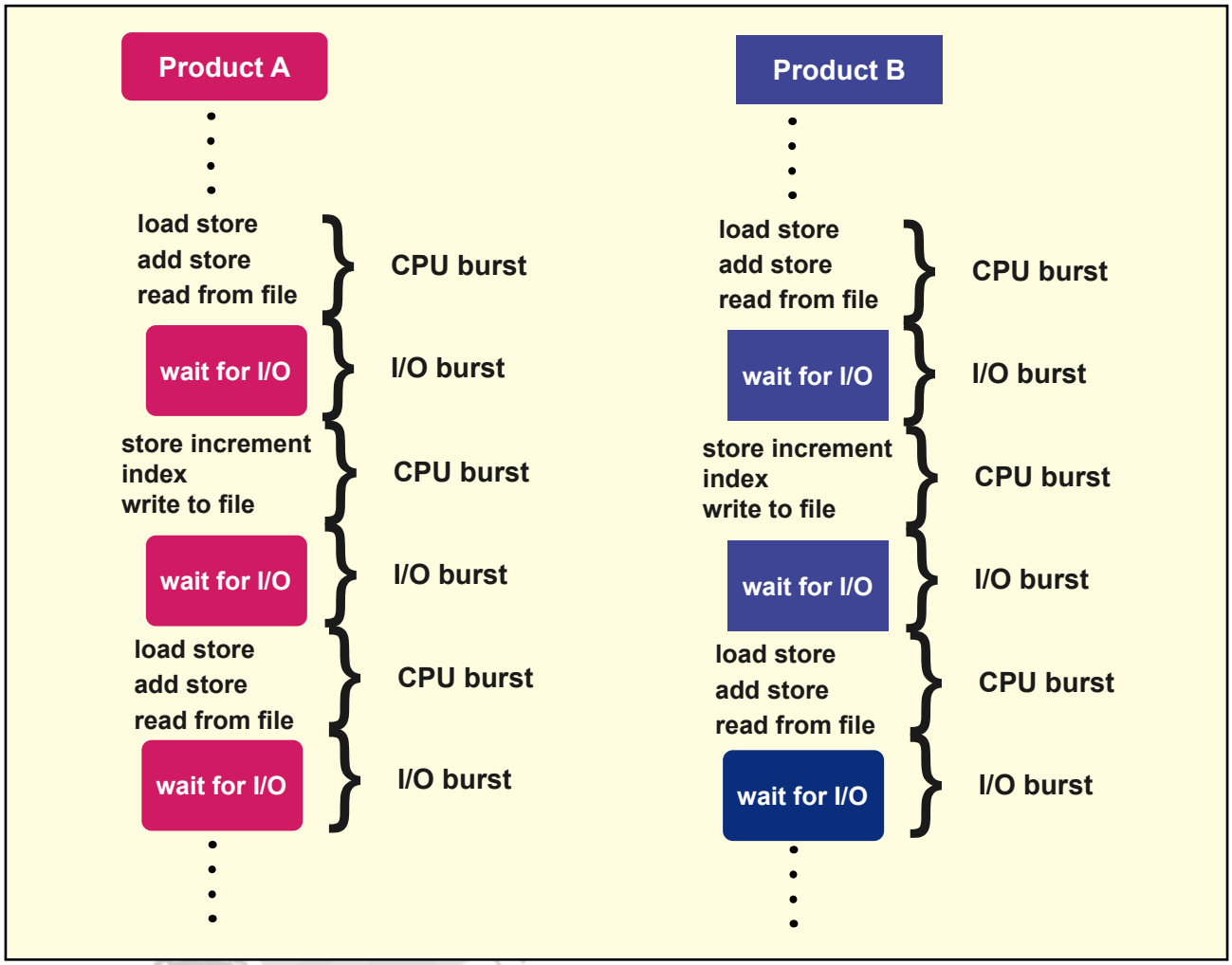


Fig. 1: I/O burst cycle

(b) Context switching

Context switching is the mechanism that allows an operating system to switch the CPU from executing one process to another. This is an essential aspect of multitasking systems, where multiple processes compete for the CPU. A context switch occurs when the operating system saves the state (context) of a currently running process and loads the saved state of the next process that is ready to run.

The process state includes the contents of CPU registers, the program counter (which indicates the next instruction to be executed), and memory management information. This data is stored in the process's Process Control Block (PCB). When a context switch occurs, the operating system stores the current process's state in its PCB and restores the state of the next process from its PCB.

Context switching incurs some overhead because saving and loading the process states take time, and no useful work is done during this process. Therefore, minimising unnecessary context switches is an important goal in designing scheduling algorithms. The more frequently the system switches between processes, the more time is spent switching, reducing overall efficiency.

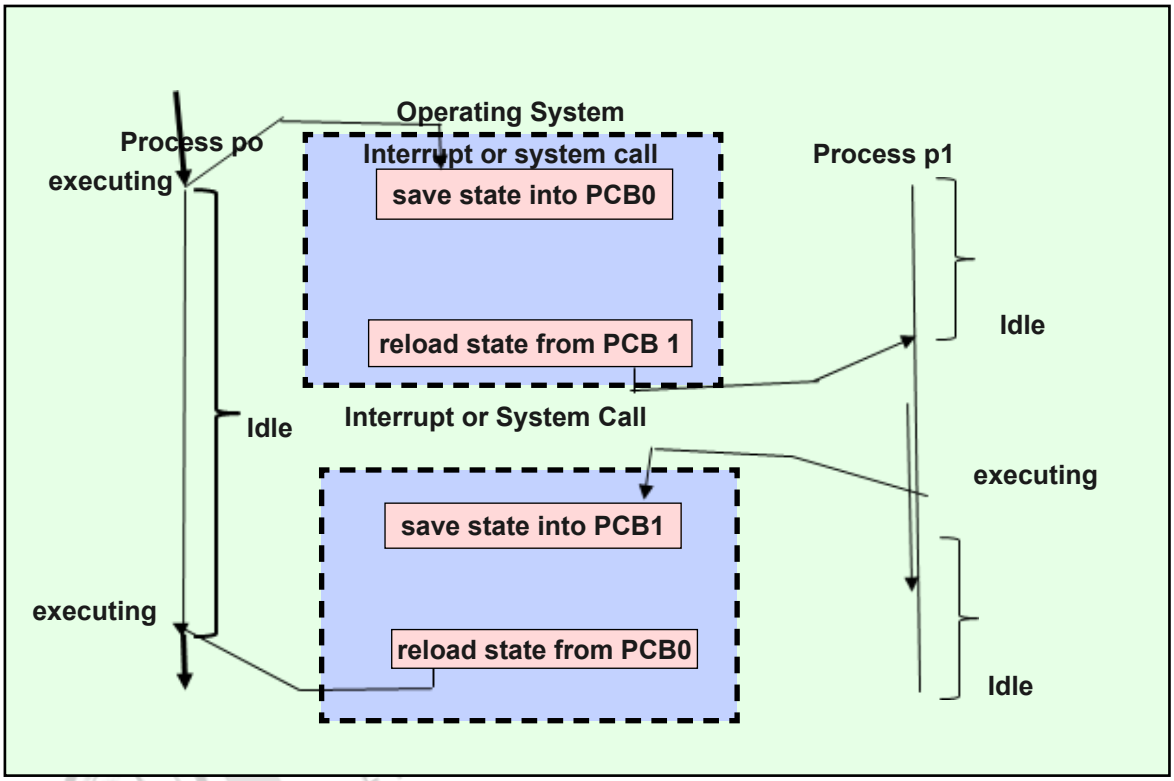


Fig. 2: Context switching

(c) Short-term scheduling

Short-term scheduling refers to the selection of a process from the ready queue to run on the CPU. This is also known as CPU scheduling and is the primary focus of most scheduling algorithms. The short-term scheduler, also called the CPU scheduler, makes decisions based on the current state of the system, such as which process is ready to execute and which is blocked waiting for I/O.

The short-term scheduler operates frequently and makes decisions in real-time, meaning its scheduling decisions directly impact system responsiveness. The goal of short-term scheduling is to keep the CPU busy by selecting processes in such a way that they maximise throughput (the

number of processes completed in a given time), minimise response time, and ensure fairness among processes.

(d) Long-term scheduling

Long-term scheduling determines which processes are admitted to the system for execution. It controls the degree of multiprogramming, or how many processes are loaded into memory and competing for CPU time. The long-term scheduler selects processes from the job pool (typically stored on a disk) and brings them into the ready queue, where the short-term scheduler can then choose them for execution.

Long-term scheduling happens less frequently than short-term scheduling and is responsible for deciding which jobs to load into memory based on system load. For example, if the system is heavily loaded, the long-term scheduler might delay admitting new jobs to avoid overloading the CPU and memory. Conversely, if the system is underutilised, the long-term scheduler will admit more processes.

Effective long-term scheduling ensures a balanced load between CPU-bound and I/O-bound processes. This balance is essential for keeping both the CPU and I/O devices busy and maximising overall system performance.

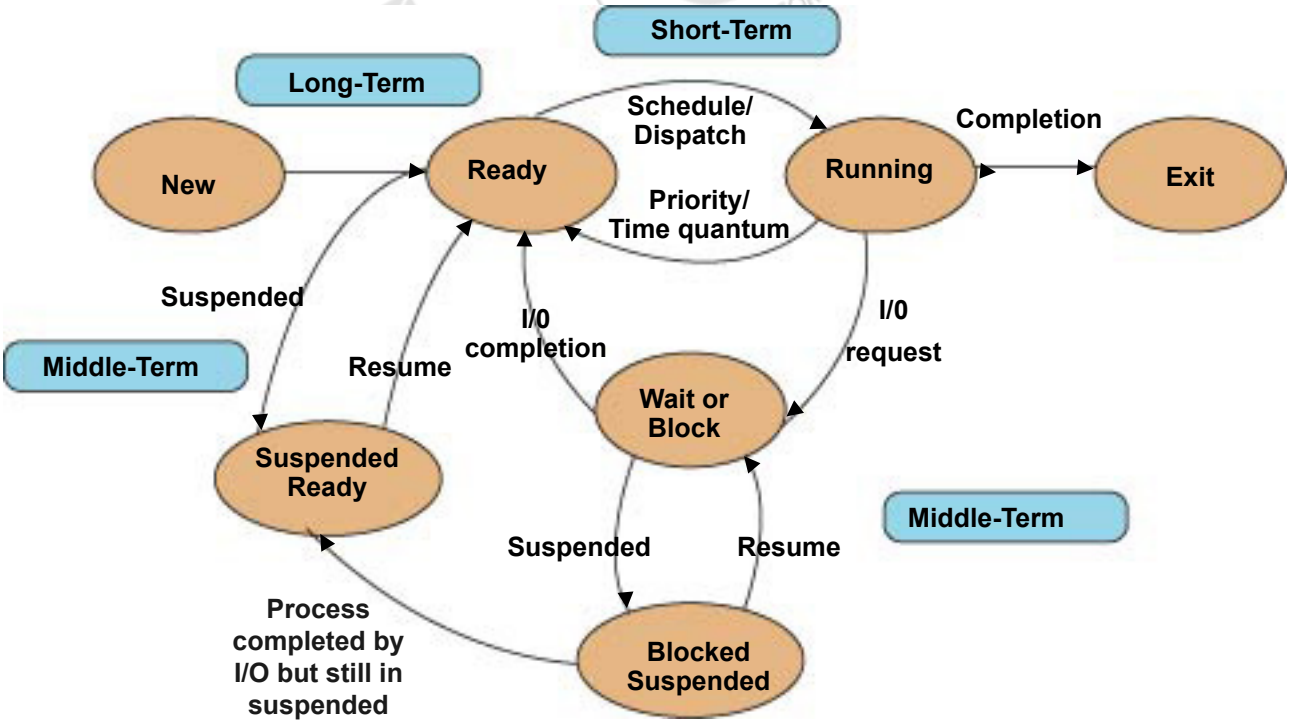


Fig. 3: Long-term and short-term scheduling

(e) Scheduling criteria

CPU scheduling algorithms aim to optimise system performance based on a variety of criteria, depending on the specific goals of the operating system. Common scheduling criteria include:

1. **CPU utilisation**: Keeping the CPU as busy as possible. Ideally, CPU utilisation should approach 100%, but this can vary depending on the workload.
2. **Throughput**: The number of processes completed per unit time. Higher throughput means the system can handle more tasks in a given period.
3. **Turnaround time**: The total time taken to execute a particular process, from submission to completion. Lower turnaround time is desirable.
4. **Waiting time**: The total time a process spends in the ready queue waiting for CPU allocation. Minimising waiting time improves system responsiveness.
5. **Response time**: The time between submitting a request and the first response being produced. This criterion is crucial for interactive systems, where users expect quick feedback.
6. **Fairness**: Ensuring that all processes receive a fair share of CPU time, preventing any single process from being starved of CPU resources.

The choice of scheduling algorithm directly impacts these criteria, and different algorithms may prioritise different goals depending on the use case.

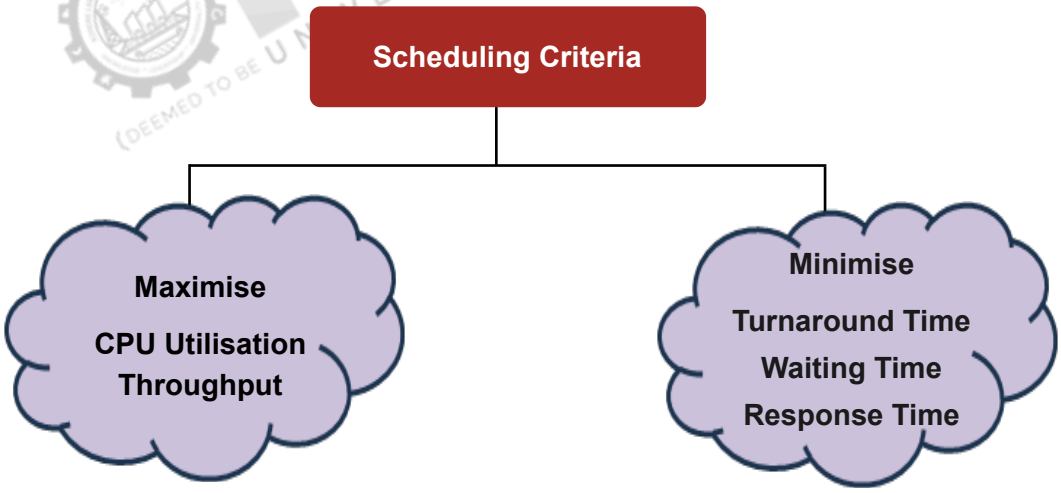


Fig . 4: Scheduling criteria

Self-Assessment Questions

1. What is the primary function of CPU scheduling in an operating system?

    a) Managing memory allocation
    b) Determining which process should execute next on the CPU
    c) Handling input/output operations
    d) Controlling access to files

2. In the context of CPU scheduling, what does the term “I/O burst” refer to?

    a) The time a process spends performing computations
    b) The time a process spends waiting for I/O operations
    c) The time a process spends switching between tasks
    d) The time a process spends waiting in the ready queue

3. Which of the following is a disadvantage of context switching in CPU scheduling?

    a) It increases the CPU’s efficiency
    b) It introduces overhead due to saving and restoring process states
    c) It reduces the need for multitasking
    d) It minimises the time spent in the ready queue

4. Which type of scheduler determines which processes are admitted to the system for execution and controls the degree of multiprogramming?

    a) Short-term scheduler
    b) Long-term scheduler
    c) Medium-term scheduler
    d) Real-time scheduler

5. Which of the following is NOT a common scheduling criterion in CPU scheduling?

    a) CPU utilisation
    b) Throughput
    c) Response time
    d) Disk access time

2.2.2 Algorithms of CPU Scheduling

CPU scheduling algorithms are critical for determining the order in which processes are executed by the CPU. These algorithms help allocate CPU time to various processes in a fair and efficient manner, ensuring optimal performance in multitasking environments. Different scheduling algorithms are designed to meet specific objectives such as minimising waiting time, maximising CPU utilisation, improving response time, and ensuring fairness among processes.

The most commonly used CPU scheduling algorithms include First Come First Serve (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin. Each of these algorithms has its own set of advantages, disadvantages, and applications depending on the nature of the processes and system requirements. By understanding these algorithms and their performance characteristics, system administrators and developers can optimise the overall efficiency of the operating system.

CPU scheduling algorithms provides an overview of the key algorithms, their working principles, and their impact on system performance, making it a foundation for learning how operating systems manage multiple processes effectively.

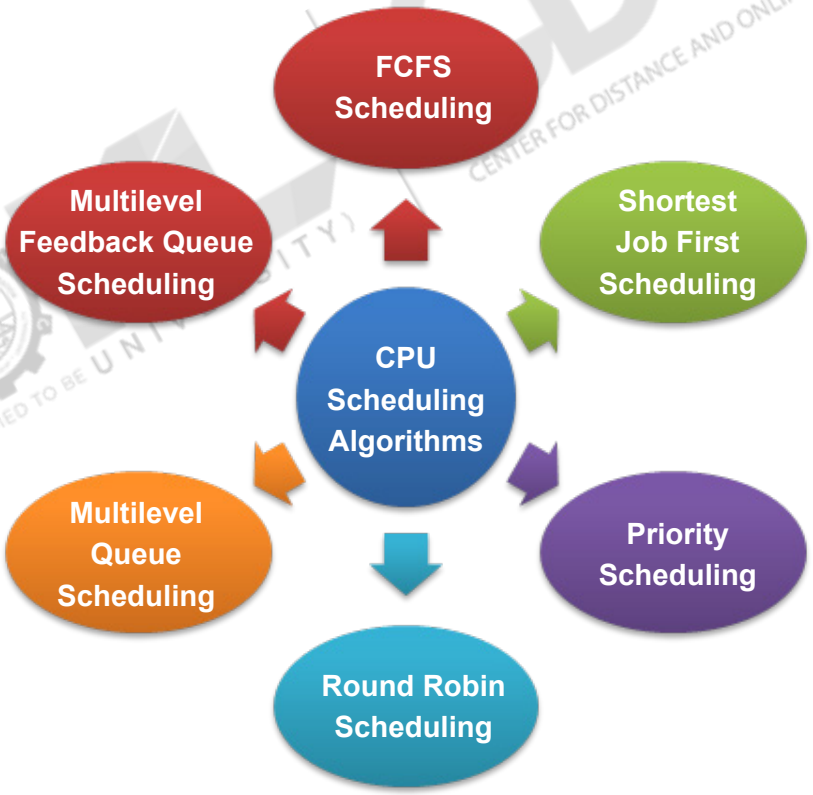


Fig. 5: Algorithms of CPU scheduling

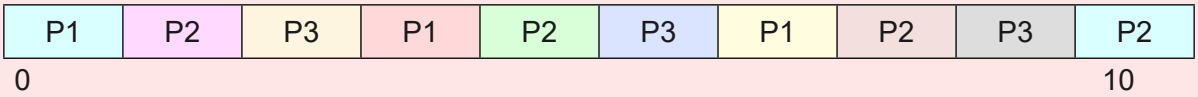
(a) First Come First Serve (FCFS) scheduling

The First Come First Serve (FCFS) scheduling algorithm is one of the simplest CPU scheduling techniques. In this algorithm, processes are assigned CPU time in the order in which they arrive in the ready queue. The process that arrives first is the one that gets executed first, and all other processes must wait their turn.

While FCFS is easy to implement and understand, it has several disadvantages, particularly in terms of efficiency. One significant drawback is the convoy effect, where short processes must wait for long processes to finish, leading to poor CPU utilisation and long waiting times for smaller tasks. This makes FCFS unsuitable for interactive or real-time systems but acceptable for batch systems where fairness and simplicity are prioritised over efficiency.

Process	Duration	Order	Arrival Time
P1	3	1	0
P2	4	2	0
P3	3	3	0

Suppose time quantum is 1 unit.



P1 waiting time: 4

P2 waiting time: 6

P3 waiting time: 6

The average waiting time(AWT): (4+6+6)/3=5.33

Fig. 6: First Come First Serve (FCFS)

(b) Shortest Job First (SJF) scheduling

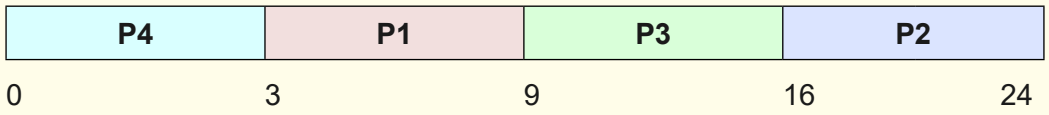
Shortest Job First (SJF) scheduling selects the process with the shortest execution time next. This algorithm is designed to minimise waiting time and maximise throughput, as shorter tasks are completed quickly, allowing more processes to be executed in a given time.

Example of SJF

Consider the following set of processes (in the ready queue) with the length of the CPU burst given in milliseconds:

Process	Burst Time (ms)
P1	6
P2	8
P3	7
P4	3

- **SJF scheduling chart**



Waiting time for P4 = 0;
Waiting time for P1 = 3;
Waiting time for P3 = 9;
Waiting time for P2= 16

Average waiting time = (3 + 16 +9 +0)/4 = 28/4 = 7 ms

Fig. 7: Shortest Job First (SJF) scheduling

There are two types of SJF scheduling

- **Non-preemptive SJF: Once** a process is selected for execution, it cannot be interrupted until it finishes.
- **Preemptive SJF (Shortest Remaining Time First)**: The scheduler can interrupt the current process if a new process arrives with a shorter remaining time.

SJF can be optimal in terms of minimising average waiting time, but it has limitations, such as the need to know or estimate the length of processes in advance. It can also lead to starvation of longer processes if short jobs keep arriving.

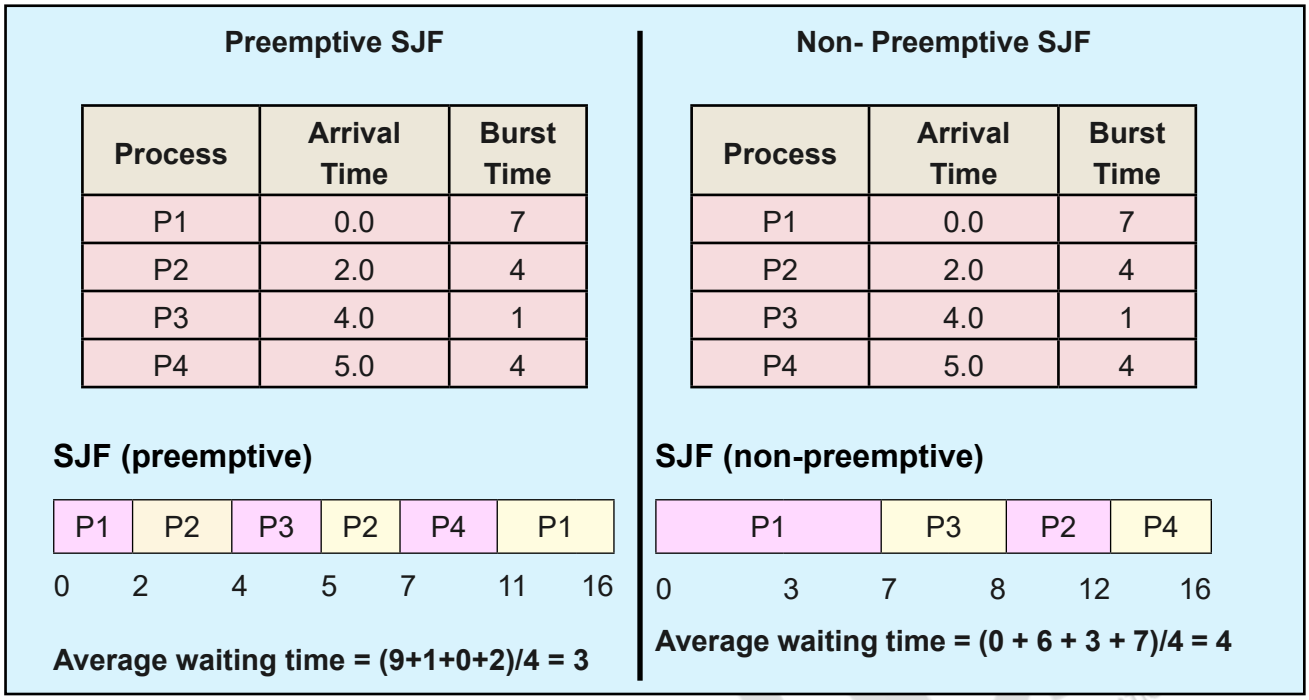


Fig. 8: Examples of SJF

(c) Priority scheduling

Priority Scheduling assigns a priority level to each process, and the CPU is allocated to the process with the highest priority. Priority can be based on various factors, such as the importance of the process, time constraints, or resource requirements. If multiple processes have the same priority, they can be scheduled using another algorithm, such as FCFS.

Priority scheduling can be either preemptive or non-preemptive. In preemptive priority scheduling, the currently running process can be interrupted if a new process with a higher priority arrives. In non-preemptive priority scheduling, the CPU will only switch to the higher priority process after the current one finishes.

One major issue with priority scheduling is starvation, where lower-priority processes may never get executed if higher-priority processes keep arriving. To address this, a technique called aging is used, where the priority of a process increases the longer it waits, ensuring that all processes eventually receive CPU time.

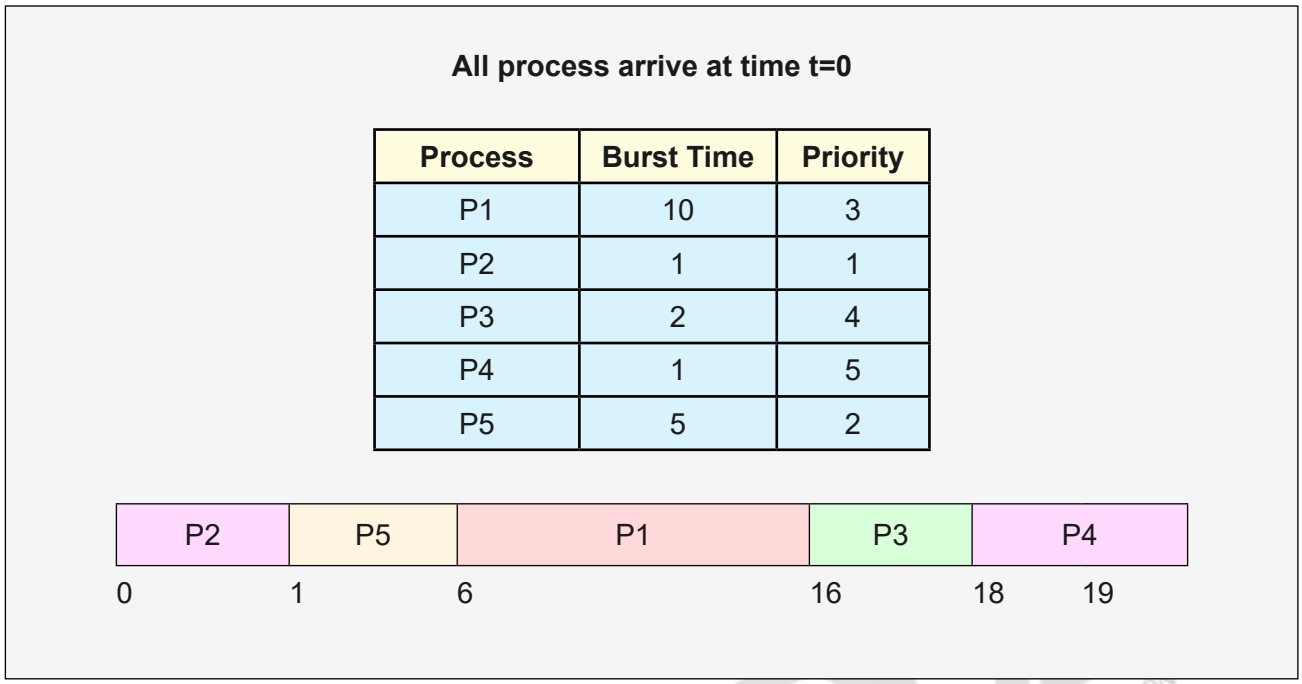


Fig. 9: Priority scheduling

(d) Round Robin (RR) scheduling

Round Robin (RR) scheduling is one of the most widely used algorithms, particularly for time-sharing systems. In RR, each process in the ready queue is assigned a small, fixed time slot called a quantum. The CPU scheduler assigns the CPU to each process in a cyclic order, executing for only one quantum at a time. If the process doesn't finish within its assigned quantum, it is preempted, and the next process in the queue is scheduled.

The primary advantage of RR scheduling is its simplicity and fairness, as each process gets an equal share of the CPU in turn. It is particularly well-suited for interactive systems where users expect quick response times. However, the choice of time quantum is critical: if the quantum is too small, there will be too many context switches, reducing system efficiency. If it's too large, the system will behave like FCFS, leading to longer wait times for short tasks.

RR ensures a balance between responsiveness and system throughput, making it an ideal choice for environments where processes need fair access to CPU time without significant overhead.

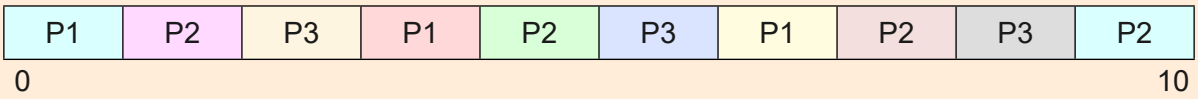
Efficient scheduling ensures that system resources are utilised effectively, maintaining a balance between CPU-bound and I/O-bound processes, minimising waiting time, and improving

responsiveness. Various algorithms, such as FCFS, SJF, Priority Scheduling, and Round Robin, offer different approaches depending on the specific requirements of the system and the workload.

Schedulers are designed to maximise CPU utilisation and throughput while minimising response and waiting times. The choice of scheduling algorithm can significantly impact system performance, and no single algorithm is best for all scenarios. Operating systems often use a combination of scheduling strategies to meet diverse needs.

Process	Duration	Order	Arrival Time
P1	3	1	0
P2	4	2	0
P3	3	3	0

Suppose time quantum is 1 unit.



P1 waiting time: 4

P2 waiting time: 6

P3 waiting time: 6

The average waiting time(AWT): (4+6+6)/3=5.33

Fig. 10: Round Robin (RR) scheduling

Self-Assessment Questions

6. Which of the following is a characteristic of the First Come First Serve (FCFS) scheduling algorithm?

    a) It is a priority-based scheduling algorithm.
    b) It assigns CPU time based on the length of the process.
    c) It executes processes in the order they arrive.
    d) It allows preemption of processes.

7. Which CPU scheduling algorithm can lead to the “convoy effect”?

    a) Round Robin (RR)
    b) Shortest Job First (SJF)
    c) Priority Scheduling
    d) First Come First Serve (FCFS)

8. What is the primary goal of the Shortest Job First (SJF) scheduling algorithm?

    a) Maximise CPU utilisation
    b) Minimise response time
    c) Minimise waiting time and maximise throughput
    d) Provide equal CPU time to all processes

9. In which type of Priority Scheduling can a running process be interrupted if a process with higher priority arrives?

    a) Non-preemptive Priority Scheduling
    b) Preemptive Priority Scheduling
    c) First Come First Serve Scheduling
    d) Shortest Job First Scheduling

10. In Round Robin (RR) scheduling, the system behaves like First Come First Serve (FCFS) if:

    a) The time quantum is too small.
    b) The time quantum is too large.
    c) Processes have the same priority.
    d) Only one process is running.

Summary

- Efficient CPU scheduling is essential for the performance of modern operating systems, determining which process gets CPU time and when.
- This process is crucial for multitasking environments, where multiple processes need to share the CPU in a way that balances throughput, CPU utilisation, and responsiveness.
- Key concepts include the I/O burst cycle, context switching, and scheduling algorithms like First Come First Serve (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin (RR).
- Each algorithm has distinct advantages and drawbacks depending on the system's requirements, such as fairness, efficiency, or real-time performance.
- These fundamentals is crucial to designing and optimising operating systems to ensure efficient resource use and improve system responsiveness.

Terminal Questions

1. What is CPU scheduling and why is it important in operating systems?
2. Explain the concept of the I/O burst cycle and its significance in CPU scheduling.
3. Describe the role of context switching in multitasking systems.
4. Compare and contrast First Come First Serve (FCFS) and Shortest Job First (SJF) scheduling algorithms.
5. How can the aging technique solve the issue of starvation in Priority Scheduling?
6. Why is it important to choose the correct time quantum in Round Robin scheduling?
7. What is the difference between short-term and long-term scheduling in operating systems?

Answer Keys

Self-Assessment Questions	
Question No.	**Answer**
1	B
2	B
3	B
4	B
5	D
6	C
7	D
8	C
9	B
10	B

Activity

Activity Type: Offline **Duration: 1 day**

You are given four processes with the following burst times and priorities

Process	Burst Time (ms)	Priority (1 = highest)
P1	8	2
P2	4	1
P3	9	3
P4	5	4

Using the following scheduling algorithms, calculate the waiting time and turnaround time for each process, then determine the average waiting and turnaround times

1. First Come First Serve (FCFS)
2. Shortest Job First (SJF)
3. Priority Scheduling (Non-preemptive)
4. Round Robin (RR) (with a time quantum of 4ms)

Questions

- Create a Gantt chart for each algorithm to visualise the order of execution.
- Calculate the total waiting time and turnaround time for each process.
- Compare the average waiting and turnaround times for each algorithm.

Glossary

- **Multitasking:** The ability of an operating system to execute multiple processes or tasks simultaneously by rapidly switching between them.
- **Pre-emptive scheduling:** A type of CPU scheduling where a running process can be interrupted and replaced by a higher-priority process.
- **Non-pre-emptive scheduling:** A type of scheduling where a process, once started, runs to completion without being interrupted.
- **Throughput:** The number of processes completed by the CPU in a given amount of time.
- **Turnaround time:** The total time taken from the submission of a process to its completion, including waiting, processing, and I/O time.
- **Waiting time:** The total time a process spends in the ready queue waiting for CPU allocation.

Bibliography

Textbooks

- Stallings, W. (2013). *Operating systems* (9th ed.). Philadelphia, PA: Pearson Education.
- Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems* (4th ed.). Upper Saddle River, NJ: Pearson.

External Resources

- Russinovich, M. E., Solomon, D. A., & Ionescu, A. (2012). *Windows(r) Internals* (5th ed.). Redmond, WA: Microsoft Press.
- Love, R. (2012). *Linux Kernel Development* (3rd ed.). Boston, MA: Addison-Wesley Educational.

e-References

- **Fundamentals of CPU scheduling:** https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/
- **Algorithms of CPU scheduling:** https://www.javatpoint.com/cpu-scheduling-algorithms-in-operating-systems

Video Links

Video	Links
Fundamentals of CPU scheduling	https://youtu.be/EWkQl0n0w5M
Algorithms of CPU scheduling	https://youtu.be/WYo1SpUh9FI

Image Credits

- **Fig. 1:** Self-Made
- **Fig. 2:** Self-Made
- **Fig. 3:** Self-Made
- **Fig. 4:** Self-Made
- **Fig. 5:** Self-Made
- **Fig. 6:** Self-Made
- **Fig. 7:** Self-Made
- **Fig. 8:** Self-Made
- **Fig. 9:** Self-Made
- **Fig. 10:** Self-Made

Keywords

- Quantum overhead
- Degree of multiprogramming
- Fairness
- Efficiency
- Context switch overhead
- Interactive system0

OPERATING SYSTEMS

MODULE 3

Process Synchronisation, Deadlock Control, and I/O Management

Module Description

This module explores key concepts in process synchronisation, deadlock control, and I/O device management within operating systems. The first section covers Process Synchronisation, focusing on mechanisms to coordinate the execution of multiple processes to prevent conflicts over shared resources. Students will explore foundational topics, including the Critical Section Problem, Mutual Exclusion techniques, and Race Conditions. This section also introduces Semaphores and classic synchronisation challenges, such as the Readers-Writers Problem and the Dining Philosophers Problem, which exemplify the complexity of maintaining data consistency and preventing concurrency issues.

Deadlocks, a critical area in process management in which system resources are held indefinitely due to cyclic dependencies. The module examines conditions that lead to deadlocks, such as Mutual Exclusion, Hold and Wait, No Pre-emption, and Circular Wait. It also presents Resource Allocation Graphs as a model for analysing deadlocks. Strategies for handling deadlocks, including Prevention, Avoidance, and applying the Banker's Algorithm, are discussed to provide students with robust approaches to managing system resources effectively.

I/O Device Management, covering input and output device types, characteristics, and operational requirements. This section prepares students to understand the significance of efficient device management in optimising overall system performance. By the end of this module, students will gain practical knowledge of synchronisation, deadlock resolution, and device management, which are essential for developing reliable and efficient operating systems.

The module consists of **two** units.

Unit 3.1: Process Synchronisation and Deadlock Control Management

Unit 3.2: I/O Device Management

MODULE 3

Process Management and CPU Scheduling

Unit 1

Process Synchronisation and Deadlock Control Management

Unit Table of Contents

Unit 3.1 Process Synchronisation and Deadlock Control Management

Aim

To apply effective techniques for managing process synchronisation and controlling deadlocks in operating systems.

Instructional Objectives

This unit intends to:

- Define the concept of process synchronisation in operating systems
- Identify various issues that arise from process synchronisation
- Describe different strategies for deadlock prevention, avoidance, and detection
- Recognise the importance of deadlock control management in system reliability

Learning Outcomes

At the end of this unit, students are expected to:

- Evaluate how process synchronisation impacts the efficiency and performance of an operating system
- Assess different methods used to handle process synchronisation issues
- Discuss scenarios where deadlock management is essential for system stability
- Compare the effectiveness of different deadlock control methods in preventing system resource conflicts

3.1.1 Process Synchronisation

Process Synchronisation is essential in concurrent systems where multiple processes need to access shared resources simultaneously. Proper synchronisation mechanisms are crucial to maintaining data consistency, preventing conflicts, and ensuring that processes operate smoothly without interfering with each other.

(a) Critical section problem

The critical section problem is central to process synchronisation and concurrent programming. A critical section is a portion of code where a process accesses shared resources, such as files, memory, or other data. To prevent conflicts, only one process should execute in its critical section at any given time.

Requirements for critical section problem

- **Mutual exclusion**: Only one process can be in the critical section at a time.
- **Progress**: If no process is in the critical section, waiting processes should be allowed to proceed.
- **Bounded waiting**: Each process should have a limited wait time to enter its critical section.

Solutions

- **Software solutions**: Algorithms like Peterson's Algorithm ensure mutual exclusion through shared variables.
- **Hardware solutions**: Atomic operations, such as test-and-set and compare-and-swap, support mutual exclusion at the hardware level.

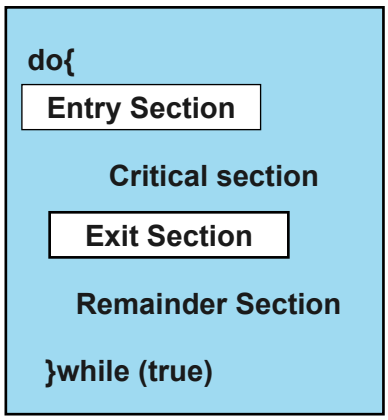


Fig. 1: Critical section problem

(b) Mutual exclusion

Mutual exclusion ensures that only one process at a time can enter a critical section, thus preventing simultaneous access to shared resources, which could lead to data inconsistencies.

Mechanisms for mutual exclusion

- **Locks**: Prevent concurrent access but must be managed to avoid deadlocks.
- **Disabling Interrupts**: Prevents context switching on a uni-processor by temporarily disabling interrupts.
- **Test-and-Set Instruction**: A low-level atomic hardware instruction providing mutual exclusion.

While crucial, mutual exclusion mechanisms alone may lead to problems such as starvation and deadlock if not carefully managed.

(c) Races

A Race Condition occurs when multiple processes access shared data simultaneously, leading to inconsistent or unpredictable results due to the sequence of operations.

Example: In a banking application, two withdrawals from a shared account balance without synchronisation can result in incorrect balances.

- **Prevention**: Race conditions are prevented using locks, semaphores, or other synchronisation tools to control access.

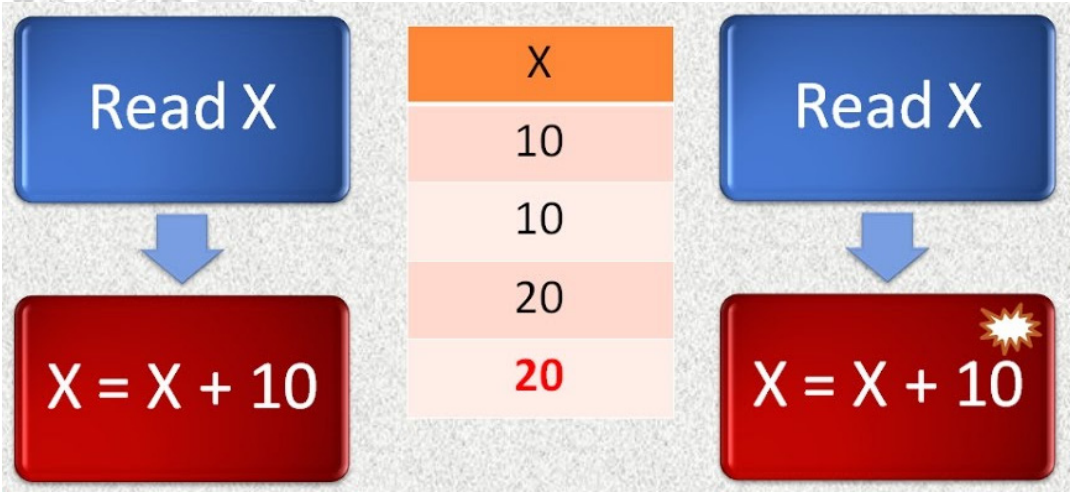


Fig. 2: Races

(d) Semaphores

A Semaphore is a synchronisation primitive used to control access to resources by processes.

Types

- **Binary semaphore (Mutex)**: Functions as a mutual exclusion lock.
- **Counting semaphore**: Manages multiple instances of a resource, allowing multiple processes access.
- **Operations**: The wait() and signal() functions are used to control access to the resource.

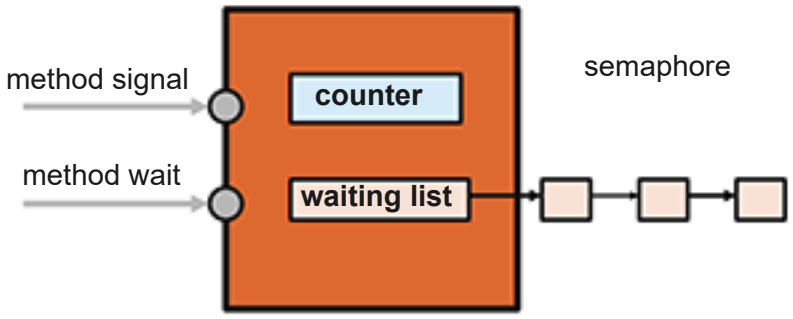


Fig. 3: Semaphores

(e) Classic synchronisation problems

Classic synchronisation problems illustrate common challenges in managing concurrent access.

- **Readers-writers problem:** Involves managing access to shared data by multiple readers and writers.
- **Dining philosophers problem:** Models processes competing for limited resources, focusing on preventing deadlock and starvation.

Self-Assessment Questions

1. Which of the following is NOT a requirement for solving the Critical Section Problem?

    a) Mutual Exclusion
    b) Progress
    c) Starvation
    d) Bounded Waiting

2. Which synchronisation mechanism is a low-level atomic hardware instruction that provides mutual exclusion?

    a) Locks
    b) Semaphore
    c) Test-and-Set Instruction
    d) Disabling Interrupts

3. In the context of Process Synchronisation, what does a Counting Semaphore do?

    a) Ensures only one process accesses a resource at a time
    b) Manages multiple instances of a resource, allowing multiple processes access
    c) Disables interrupts to prevent context switching
    d) Prevents deadlock and starvation among processes

4. The Readers-Writers Problem illustrates issues with managing concurrent access to shared resources. Which of the following statements about it is correct?

    a) It allows only one reader or writer at any given time
    b) It models processes competing for shared memory space
    c) It permits multiple readers or a single writer to access data at one time
    d) It uses test-and-set instruction for synchronisation

5. In a scenario where multiple processes access shared data at the same time without proper synchronisation, leading to inconsistent results, this situation is called a:

    a) Deadlock
    b) Race Condition
    c) Bounded Waiting
    d) Semaphore Operation

3.1.2 Deadlock Control Management

Deadlocks occur when processes are stuck in a circular wait pattern, each holding resources needed by another, leading to a halt in system functionality.

(a) Deadlocks and starvation

- **Deadlock**: A cyclic waiting pattern where processes are blocked indefinitely.
- **Starvation**: Processes wait indefinitely due to unequal resource allocation.

Deadlock involves a cyclic dependency, while starvation is due to priority or preemption issues without cyclic dependency.

(b) System model for deadlock

The System Model for deadlock management defines processes, resources, and rules for resource allocation and release. Deadlocks are modeled using resource allocation graphs where nodes represent processes or resources, and edges show resource holding or requests.

(c) Necessary conditions for deadlock

Deadlock can occur if the following four conditions hold simultaneously

- **Mutual exclusion**: Resources cannot be shared.
- **Hold and wait**: Processes holding resources can request additional resources.
- **No Pre-emption**: Resources cannot be forcibly removed.
- **Circular wait**: A circular chain exists where each process holds a resource needed by the next.

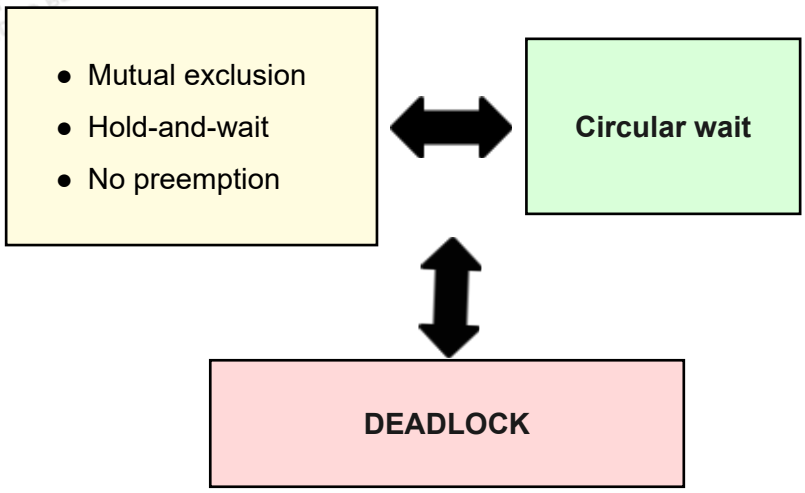


Fig. 4: Conditions for deadlock

(d) Resource allocation graphs

Resource allocation graphs are visual tools for deadlock detection, where

- **Cycle detection**: A cycle in a graph may indicate a potential deadlock.
- **Deadlock detection algorithms**: Some systems use graph-based cycle detection to identify deadlocks.

(e) Handling deadlocks

Various methods address deadlocks in systems

- **Deadlock prevention**: Ensures that at least one of the four necessary deadlock conditions does not occur.
    - **Examples**: Limit holding of resources while requesting new ones, allow preemption.
- **Deadlock avoidance**: Dynamically monitors resource allocation to ensure that the system remains in a safe state.

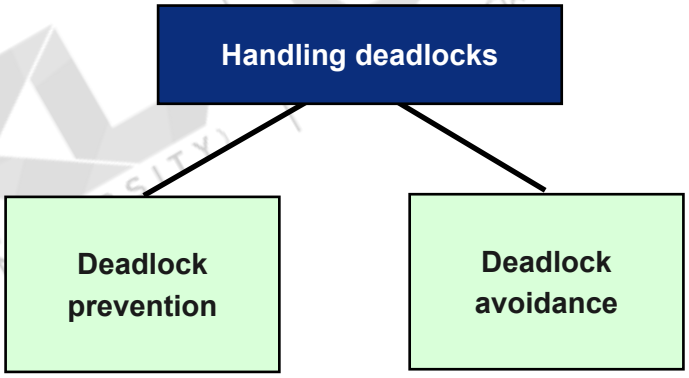


Fig. 5: Methods for handling deadlocks

(f) Banker's algorithm

A popular avoidance method that checks resource availability to prevent unsafe resource allocation.

Self-Assessment Questions

6. What is the primary difference between deadlock and starvation?

    a) Deadlock is a temporary state, while starvation is permanent.
    b) Deadlock involves a cyclic dependency, while starvation results from priority issues.
    c) Starvation involves cyclic dependency, while deadlock does not.
    d) There is no difference; both terms are interchangeable.

7. Which of the following is NOT one of the necessary conditions for a deadlock to occur?

    a) Mutual Exclusion
    b) Hold and Wait
    c) Resource Allocation
    d) Circular Wait

8. In the context of resource allocation graphs, what does a cycle indicate?

    a) A successful resource allocation
    b) A potential deadlock
    c) Safe resource allocation
    d) Resource request fulfillment

9. Which of the following strategies prevents deadlocks by ensuring at least one necessary condition does not occur?

    a) Deadlock Avoidance
    b) Deadlock Detection
    c) Deadlock Prevention
    d) Resource Allocation

10. The Banker's Algorithm is primarily used for which of the following purposes?

    a) Detecting deadlocks
    b) Preventing deadlocks
    c) Allocating resources to processes
    d) Managing process priorities

Summary

- Process synchronisation is essential in systems where multiple processes need to access shared resources. Without it, processes could interfere with each other, leading to data inconsistency or system errors.
- Synchronisation methods, like using locks or semaphores, help manage access to shared resources and maintain system stability.
- One of the key challenges in synchronisation is the Critical Section Problem, where only one process should access shared resources at a time.
- Solutions like software algorithms (e.g., Peterson's Algorithm) or hardware instructions (e.g., Test-and-Set) help maintain order in accessing these critical sections.
- Mutual Exclusion ensures that only one process accesses a resource at a time. However, improper handling of mutual exclusion can lead to issues like deadlock,
- Processes get stuck waiting on each other indefinitely, or starvation, where a process waits too long due to others taking priority.
- Deadlocks occur when processes hold resources while waiting for others held by other processes, leading to a cycle of dependency.

Terminal Questions

1. What are the key requirements for solving the Critical Section Problem?
2. Explain the differences between deadlock and starvation.
3. Describe the four necessary conditions for deadlock to occur.
4. How can race conditions be prevented in concurrent systems?
5. What are semaphores, and how do they function in process synchronisation?
6. Illustrate the Readers-Writers Problem and its challenges.

Answer Keys

Self-Assessment Questions	
Question No.	**Answer**
1	C
2	C
3	B
4	C
5	B
6	B
7	C
8	B
9	C
10	B

Activity

Activity Type: Offline **Duration: 1 day**

In a small office, there are four employees—A, B, C, and D—who each need to use a printer and a scanner. However, there is only one printer and one scanner available. If each employee has already accessed one resource and is now waiting for the other, what potential deadlock scenario could arise from this situation?

Glossary

- **Critical section:** A part of the code where a process accesses shared resources.
- **Mutual exclusion:** A condition where only one process can access a critical section at a time.
- **Process Synchronisation:** Process Synchronisation is a mechanism in operating systems that manages the order in which multiple processes access shared resources or data to avoid conflicts and ensure consistency.
- **Bounded waiting:** The requirement that each process must have a limited waiting time to enter its critical section.
- **Race condition:** A scenario where the outcome depends on the timing of uncontrollable events, leading to inconsistent results.

Bibliography

Textbooks

- Stallings, W. (2013). *Operating systems* (9th ed.). Philadelphia, PA: Pearson Education.
- Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems* (4th ed.). Upper Saddle River, NJ: Pearson.

External Resources

- Russinovich, M. E., Solomon, D. A., & Ionescu, A. (2012). *Windows(r) Internals* (5th ed.). Redmond, WA: Microsoft Press.
- Love, R. (2012). *Linux Kernel Development* (3rd ed.). Boston, MA: Addison-Wesley Educational.

e-References

- **Process synchronisation:** https://www.geeksforgeeks.org/introduction-of-process-synchronisation/
- **Deadlock control management:** https://www.javatpoint.com/deadlock-in-dbms

Video Links

Video	Links
Process synchronisation	https://youtu.be/ph2awKa8r5Y
Deadlock control management	https://youtu.be/ee-wg9q29f0

Image Credits

- **Fig. 1:** Self-Made
- **Fig. 2:** Self-Made
- **Fig. 3:** Self-Made
- **Fig. 4:** Self-Made
- **Fig. 5:** Self-Made

Keywords

- Concurrency
- Atomic operations
- Test-and-Set Instruction
- Compare-and-Swap
- Lock mechanism
- Inter-process Communication (IPC)

MODULE 3

Process Management and CPU Scheduling

Unit 2

I/O Device Management

Unit Table of Contents

Unit 3.2 I/O Device Management

Aim

To explore various I/O device types and their characteristics to enable effective integration and management in computing systems.

Instructional Objectives

This unit intends to:

- Identify various types of I/O devices commonly used in computing systems
- Define the primary functions of each type of I/O device
- List characteristics that differentiate input devices from output devices
- Recall the basic functions of I/O devices in data processing and system communication

Learning Outcomes

At the end of this unit, students are expected to:

- Describe the different types of I/O devices and their roles in a computer system
- Explain how the characteristics of I/O devices affect system performance and data flow
- Classify I/O devices into input, output, and storage categories based on their functions
- Compare various I/O devices based on speed, functionality, and compatibility

3.2.1 I/O Device Types

Introduction to I/O devices

I/O devices are crucial hardware components that facilitate communication between a computer and its external environment. They manage the flow of data input and output, enabling users to interact with the system effectively. I/O devices are typically categorised based on their primary functions, which can be broadly divided into four main types:

1. Input devices

Input devices allow users to provide data to the computer for processing. They are essential for user interaction with the software and enable the transmission of commands, data, or signals to the CPU.

Common examples

- **Keyboard**: A standard input device used for typing text and executing commands. It consists of keys that represent characters, numbers, and functions.
- **Mouse**: A pointing device that enables navigation and selection within a graphical user interface. It typically has buttons and a scroll wheel for interaction.
- **Scanner**: Converts physical documents and images into digital format, allowing for electronic storage and manipulation.
- **Microphone**: Captures audio input for purposes such as voice recognition, recording, and real-time communication.

2. Output devices

Output devices present processed data from the computer in a format that users can understand. They transform binary data into human-readable forms, allowing users to visualise or hear information generated by the system.

Common examples

- **Monitor**: Displays visual output, including graphical user interfaces, videos, and images. Monitors come in various sizes and resolutions.
- **Printer**: Produces hard copies of documents or images from digital formats. Different types of printers include inkjet, laser, and thermal.
- **Speakers**: Convert digital audio signals into audible sound, enabling playback of music, alerts, and other audio content.
- **Projector**: Projects visual output onto larger surfaces, suitable for presentations and educational purposes.

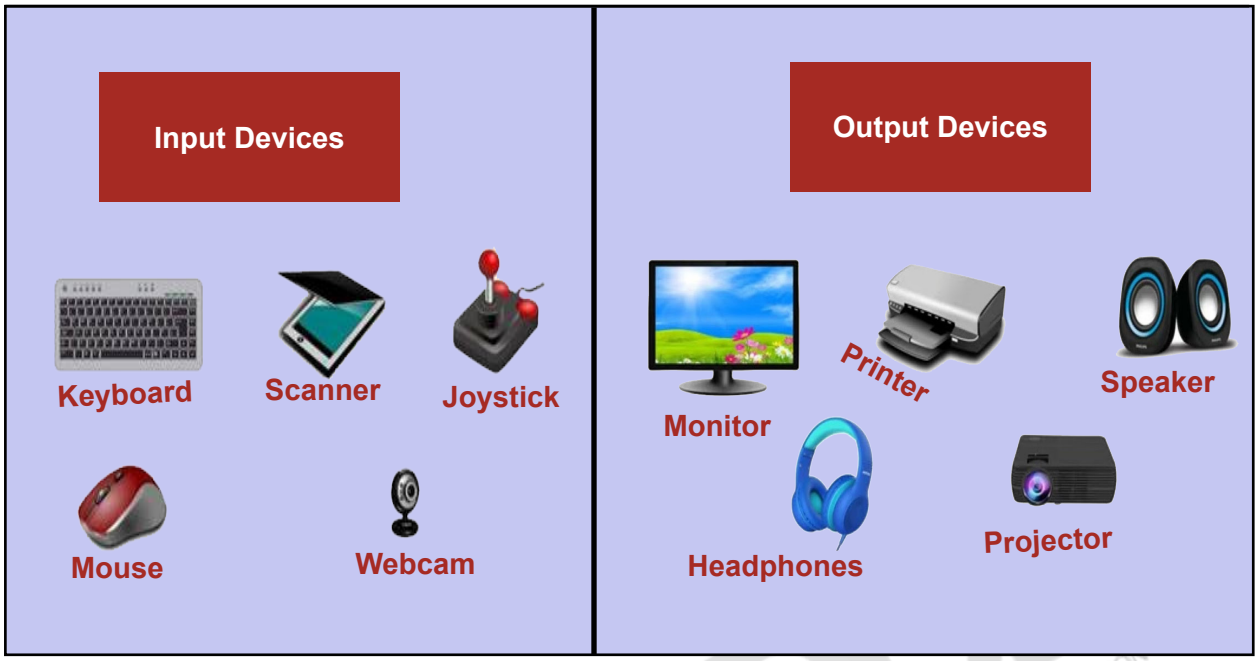


Fig. 1: Input and output devices

3. Storage devices

Storage devices function as input and output devices, enabling data storage and retrieval. They provide a medium for saving files, applications, and data persistently, ensuring that information remains accessible even when the computer is powered off.

Common examples

- **Hard Drive (HDD)**: A magnetic storage device used for long-term storage of operating systems, applications, and files. HDDs offer large storage capacities but are generally slower than SSDs.
- **Solid State Drive (SSD)**: A faster, non-volatile storage device that uses flash memory, resulting in quicker data access and improved performance compared to HDDs.
- **USB Flash Drive**: A portable storage medium that allows users to easily transfer and back up data between devices.
- **Optical Disk (CD/DVD)**: Used for data storage and distribution in an optical format, commonly used for media and software distribution.

Fig. 2: Storage devices

4. Hybrid devices

Hybrid devices combine the functionalities of input and output within the same hardware component, allowing for real-time interaction between users and the computer.

Common examples

- **Touch screen**: Acts as both an input (touch) and output (display) device, enabling users to interact directly with visual elements on the screen.

Fig. 3: Touch screen

- **Network Interface Card (NIC):** Facilitates network communication by managing both input and output data, enabling the sending and receiving of data packets over a network.

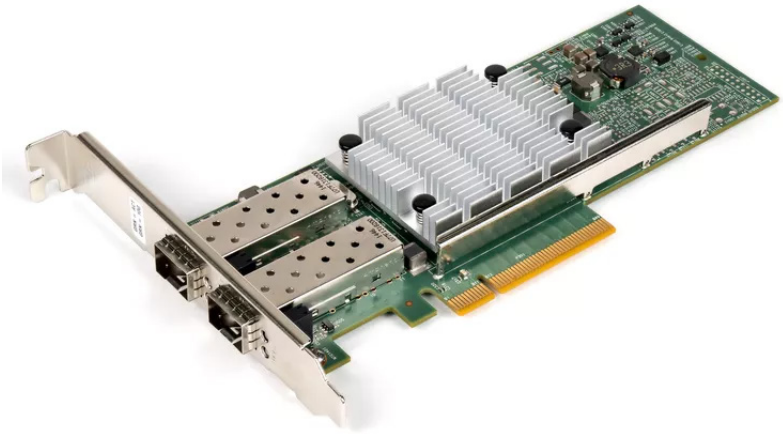


Fig. 4: Network Interface Card (NIC)

Self-Assessment Questions

1. Which of the following is primarily considered an input device?

    a) Monitor
    b) Printer
    c) Keyboard
    d) Projector

2. What is the main function of output devices?

    a) To store data for long-term use
    b) To capture audio signals
    c) To display or present processed data in a human-readable format
    d) To interact with network interfaces

3. Which type of I/O device can act as both an input and an output device?

    a) Mouse
    b) Scanner
    c) Touch screen
    d) Microphone

4. A hard drive (HDD) is categorised as which type of I/O device?

    a) Input Device
    b) Output Device
    c) Storage Device
    d) Hybrid Device

5. What is the primary role of a Network Interface Card (NIC) as an I/O device?

    a) To display data
    b) To connect to other input devices
    c) To manage network communication by handling data input and output
    d) To capture digital images

3.2.2 Characteristics of I/O Devices

Each I/O device type possesses unique characteristics that impact its functionality, performance, and compatibility with a computer system. Understanding these characteristics is crucial for selecting appropriate devices based on specific use cases. Key characteristics include:

1. Data transfer rate

The data transfer rate defines the speed at which data is transmitted between the I/O device and the computer system. It is measured in bytes per second (Bps) or bits per second (bps). Higher data transfer rates indicate better performance. For instance, SSDs typically exhibit much higher transfer rates than traditional HDDs, leading to quicker boot times and faster file access.

Fig. 5: Data transfer rate

2. Access method

I/O devices utilise different methods for accessing data, which can significantly affect performance:

- **Sequential access**: Data is accessed in a specific sequence, which is typical for devices like tape drives.
- **Direct access**: Allows data to be accessed directly at any location on the storage medium. This is common in HDDs and SSDs, making them more versatile and efficient.

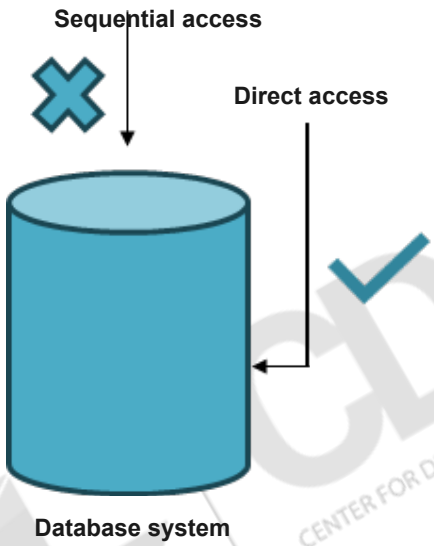


Fig. 6: Access method

3. Latency

Latency refers to the delay between issuing a command to the I/O device and the start of data transfer. Lower latency is crucial for enhancing the responsiveness of devices, particularly in real-time applications such as gaming, video streaming, and video conferencing.

4. Device compatibility

Compatibility is critical for ensuring that I/O devices can connect and communicate seamlessly with the computer system. This includes compatibility with the operating system and other hardware components. Some peripheral devices may require specific drivers or software updates to function correctly.

5. Form factor

The physical size and shape of an I/O device are referred to as its form factor. This is especially important for internal components like hard drives, which must fit into designated bays within a computer case. Form factors can affect installation and integration into the system.

6. Port type and interface

Different I/O devices use specific ports for data transfer. Common interfaces include:

- **USB**: Widely used for external devices like flash drives, mice, and printers.
- **HDMI**: Used for video output devices such as monitors and projectors, supporting high-definition video and audio transmission.
- **SATA**: Primarily used for connecting internal storage devices like HDDs and SSDs.
- **Ethernet**: Commonly used for networking devices, facilitating data transfer across local area networks (LANs).

Fig. 7: Port type and interface

7. Power consumption

The power requirements of I/O devices can vary significantly. Devices with lower power consumption are beneficial for reducing energy usage, especially in portable devices where battery life is a concern. For example, SSDs generally consume less power than HDDs.

8. Reliability and durability

Reliability refers to how consistently an I/O device performs over time, while durability indicates its ability to withstand physical wear and tear. Storage devices, in particular, require high reliability and durability, as they are often responsible for critical data storage. For instance, enterprise-grade SSDs are designed for higher endurance compared to consumer models.

9. Cost

Cost is an important factor when selecting I/O devices. Prices can vary widely based on device type, performance specifications, and additional features. Budget considerations often play a significant role in determining the right I/O device for specific applications.

Self-Assessment Questions

6. What does the data transfer rate of an I/O device indicate?

    a) The amount of data it can store
    b) The speed at which data is transmitted between the device and computer
    c) The size of the device's form factor
    d) The compatibility with the operating system

7. Which access method allows data to be retrieved directly from any location on the storage medium?

    a) Sequential Access
    b) Random Access
    c) Direct Access
    d) Delayed Access

8. Why is low latency important in I/O devices, particularly in applications like gaming and video streaming?

    a) It reduces power consumption
    b) It enhances device responsiveness
    c) It increases storage capacity
    d) It decreases device compatibility

9. Which of the following port types is primarily used for connecting internal storage devices?

    a) USB
    b) HDMI
    c) SATA
    d) Ethernet

10. In terms of I/O device characteristics, what does reliability refer to?

    a) The device's ability to access data directly
    b) The consistency of the device's performance over time
    c) The device's form factor and physical size
    d) The data transfer rate

Summary

- I/O (Input/output) devices are essential hardware components that enable a computer to communicate with its external environment, facilitating data input, processing, and output.
- These devices are categorised into four main types based on their functions: Input Devices (e.g., keyboard, mouse, scanner), Output Devices (e.g., monitor, printer, speakers), Storage Devices (e.g., hard drives, SSDs, USB flash drives), and Hybrid Devices (e.g., touch screen, NIC).
- Each type of I/O device possesses specific characteristics impacting functionality and performance, including data transfer rates, access methods, latency, compatibility, form factor, port types, power consumption, reliability, and cost.
- These factors help in determining the most suitable I/O device for particular applications, taking into account the unique requirements of the user and the computing environment.

Terminal Questions

1. Define I/O devices and explain their significance in computer systems.
2. List the four main types of I/O devices and give two examples of each type.
3. What is the difference between sequential access and direct access methods in I/O devices? Provide examples of devices for each method.
4. Why is device compatibility important for I/O devices?
5. Describe the importance of data transfer rate in I/O devices and how it affects performance.

Answer Keys

Self-Assessment Questions	
Question No.	**Answer**
1	C
2	C
3	C
4	C
5	C
6	B
7	C
8	B
9	C
10	B

Activity

Activity Type: Offline **Duration: 2 hours**

Imagine you need to store a large collection of photos, videos, and important documents safely for a long time. Which type of storage device would you choose and why? Consider factors such as storage capacity, speed, portability, and durability in your answer.

Glossary

- **Data Transfer Rate:** The speed at which data is transferred between the I/O device and the computer, typically measured in bytes or bits per second.
- **Latency:** The delay between issuing a command to an I/O device and the beginning of data transfer.
- **Sequential Access:** Access method where data is read in a specific order, commonly used in tape drives.
- **Direct Access:** Method allowing data to be accessed at any location, typical of HDDs and SSDs.
- **Device Compatibility:** The ability of an I/O device to work seamlessly with a computer system, including software and hardware components.

Bibliography

Textbooks

- Stallings, W. (2013). *Operating systems* (9th ed.). Philadelphia, PA: Pearson Education.
- Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems* (4th ed.). Upper Saddle River, NJ: Pearson.

External Resources

- Russinovich, M. E., Solomon, D. A., & Ionescu, A. (2012). *Windows(r) Internals* (5th ed.). Redmond, WA: Microsoft Press.
- Love, R. (2012). *Linux Kernel Development* (3rd ed.). Boston, MA: Addison-Wesley Educational.

e-References

- **I/O Device types:** https://www.geeksforgeeks.org/i-o-hardware-in-operating-system/
- **Characteristics of I/O devices:** https://www.tutorialspoint.com/operating_system/os_io_hardware.htm

Video Links

Video	Links
Input & Output devices	https://youtu.be/AeLjpyWp4bk
Storage devices	https://youtu.be/BHj81z5QQc4
Data transfer rate	https://youtu.be/bSIVCmOT3MM

Image Credits

- **Fig. 1:** Self-Made
- **Fig. 2:** Self-Made
- **Fig. 3:** https://www.angelpos.ca/assets/images/17%20monitor%20touch.png
- **Fig. 4:** https://www.lifewire.com/thmb/T7TLMsmAn2gBD_nqhjSdtR-ND7I=/750x0/filters:no_upscale():max_bytes(150000):strip_icc():format(webp)/QIe3442-cu_10gbe_nic-5aee673cff1b780036491a47.jpg
- **Fig. 5:** https://techhyme.com/wp-content/uploads/2023/11/Sending-Information-Techhyme.jpg
- **Fig. 6:** https://www.scaler.com/topics/file-access-methods-in-os/
- **Fig. 7:** https://www.etechnophiles.com/different-types-of-computer-ports/

Keywords

- Form factor
- Port type
- Power consumption
- Reliability
- Durability

OPERATING SYSTEMS

MODULE 4

Memory Management, Fragmentation, and File System Organisation

Module Description

This section provides an in-depth overview of core memory management techniques, storage solutions, and file-system structures essential to modern operating systems. It begins with Swapping, which involves moving processes in and out of memory to maximise CPU usage, followed by Multiple Partition Allocation strategies like First Fit, Best Fit, and Worst Fit to efficiently allocate memory blocks to processes. RAID (Redundant Array of Independent Disks) and Data Redundancy is then examined to explain data protection and storage optimisation methods.

Fragmentation is discussed, distinguishing between Internal Fragmentation (unused space within allocated memory blocks) and External Fragmentation (scattered free memory outside blocks), both affecting memory efficiency. Paging and Demand Paging are explored to clarify memory access optimisation, along with Page Replacement strategies and algorithms such as FIFO, LRU, and Optimal Replacement that reduce page faults. The concept of Thrashing, a performance-degrading situation due to excessive page swapping, highlights the importance of effective memory management.

In the file-system area, the structure of file systems and various Access Methods (e.g., sequential, direct) are introduced. This section also covers the Directory Structure, detailing file organisation, as well as File-System Implementation aspects such as storage allocation, metadata management, and error handling. File Protection techniques are explained to demonstrate how access permissions control and secure data.

The module consists of **two** units.

Unit 4.1 Memory Management and Fragmentation

Unit 4.2 File System Organisation and Management

MODULE 4

Memory Management, Fragmentation, and File System Organisation

Unit 1

Memory Management and Fragmentation

Unit Table of Contents

Unit 4.1 Memory Management and Fragmentation

Aim

The aim is to explain essential memory management techniques, including fragmentation and paging, for efficient resource allocation in operating systems.

Instructional Objectives

This unit intends to:

- Define key memory management techniques, including swapping and partition allocation
- List the types of memory partition allocation strategies (First Fit, Best Fit, Worst Fit)
- Describe paging and demand paging as memory access techniques
- Identify page replacement strategies and basic page replacement algorithms

Learning Outcomes

At the end of this unit, students are expected to:

- Explain how memory management techniques, such as swapping and partition allocation, optimise system performance
- Analyse the characteristics of First Fit, Best Fit, and Worst Fit allocation strategies and their effects on memory usage
- Differentiate between various page replacement strategies (e.g., FIFO, LRU, Optimal Replacement) based on their effectiveness in reducing page faults
- Discuss the concept of thrashing and how it affects overall system performance

4.1.1 Memory Management Techniques

Memory management is a critical function of an operating system, responsible for managing a computer's primary memory (RAM) effectively to support multiple processes. It ensures that processes have adequate memory to execute, protects processes from each other's data, and optimises the use of available memory to improve performance and prevent system crashes.

In a multitasking environment, several processes run concurrently, each requiring its own memory space. The operating system's memory management module oversees the allocation and de-allocation of memory, ensuring that each process has sufficient memory to function without interfering with other processes. Memory management aims to balance efficient memory usage, system performance, and user experience.

Fig. 1: Memory management

Key goals of memory management

1. **Efficient allocation**: Dynamically allocating memory to processes while maximising system memory use and minimising waste.
2. **Isolation and protection**: Ensuring processes are isolated from one another to prevent data corruption or unauthorised access.
3. **Multitasking support**: Allowing multiple processes to run simultaneously without interruption or resource contention.
4. **Data safety**: Preventing data loss or crashes due to memory errors or fragmentation.
5. **Optimised performance**: Using techniques like caching, swapping, and virtual memory to keep active processes ready to execute and reduce latency.

Types of memory managed by an OS

1. **Primary Memory (RAM)**: Fast and directly accessible by the CPU, where processes are loaded for execution.
2. **Secondary storage**: Used for data that doesn't need to reside in memory all the time, such as swap space on a hard drive.
3. **Cache memory**: High-speed memory used for storing frequently accessed data to reduce access time.

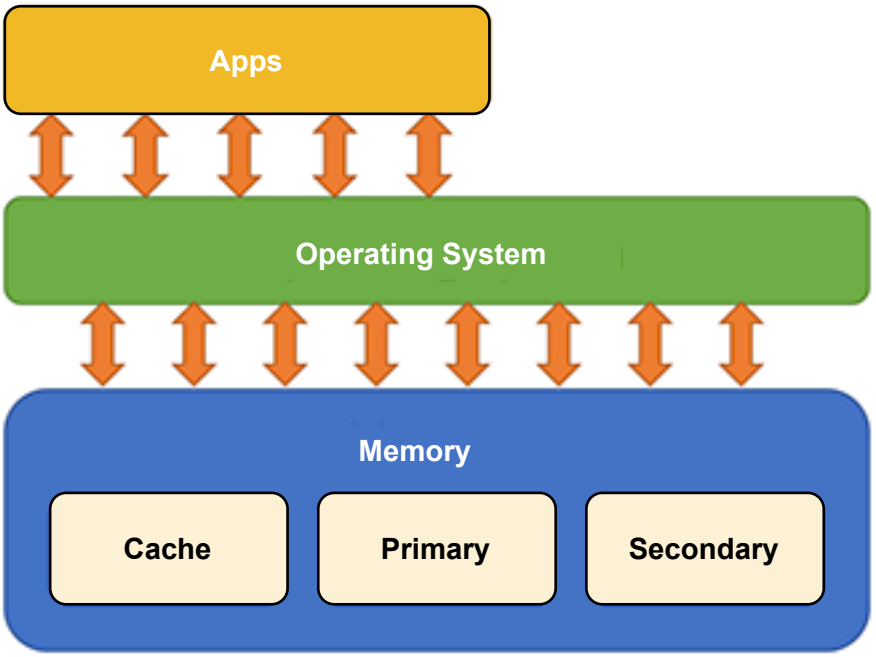


Fig. 2: Types of memory

Techniques in memory management

(a) Swapping

Swapping is a memory management technique that moves processes between the main memory and a secondary storage area (such as a hard disk) to ensure that memory is available for active processes. When there is insufficient physical memory, inactive processes are "swapped out" to disk to free up space, and later "swapped in" when needed. Swapping enables systems to multitask by allowing more processes to reside in memory in a controlled manner, though it can increase the latency for process access.

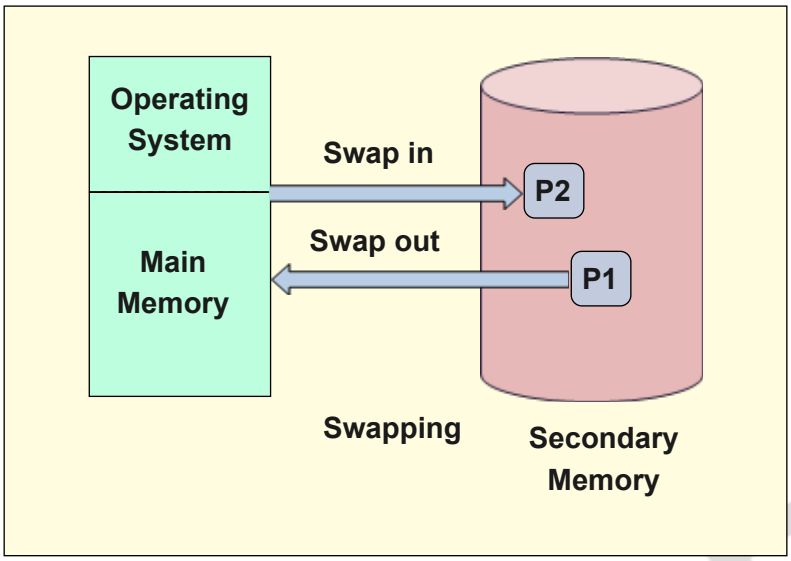


Fig. 3: Swapping

b) Multiple partition allocation strategies

In multiple partition allocation, memory is divided into various partitions that can be allocated to processes. This approach helps manage memory allocation more effectively by categorising partitions based on process needs. The key strategies include

- **First Fit**: This strategy places a process in the first memory partition large enough to accommodate it. It is a fast and straightforward approach but may lead to memory wastage as larger partitions get assigned to smaller processes.
- **Best Fit**: In this strategy, the system searches the memory for the smallest partition that is large enough to meet the process requirements, thereby minimising wasted space. Best Fit can reduce fragmentation but is slower as it requires scanning all partitions to find the best match.
- **Worst Fit**: This strategy allocates a process to the largest available partition. The goal is to leave larger spaces available for future allocations, but it can result in increased fragmentation.

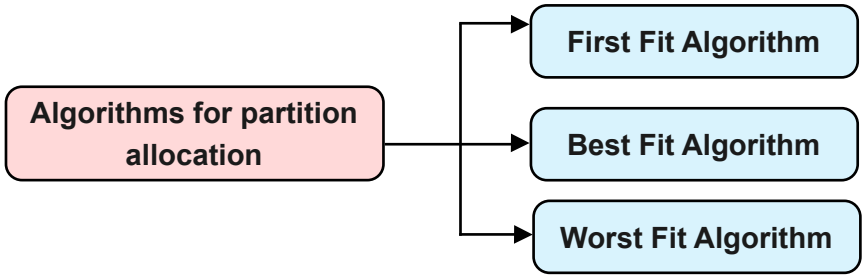


Fig. 4: Multiple partition allocation

(c) RAID and Data redundancy

RAID (Redundant Array of Independent Disks) is a data storage technology that combines multiple physical disk drives into a single logical unit to improve data redundancy, performance, or both. RAID configurations can offer different advantages

- **RAID 0**: Focuses on performance by striping data across multiple disks but lacks redundancy.
- **RAID 1**: Mirrors data across disks to create a backup in case of disk failure, enhancing data redundancy.
- RAID 5: Stripes data and parity information across multiple disks, allowing recovery from a single disk failure.

Data redundancy in RAID ensures data availability and durability, making RAID suitable for systems that require high uptime and reliability.

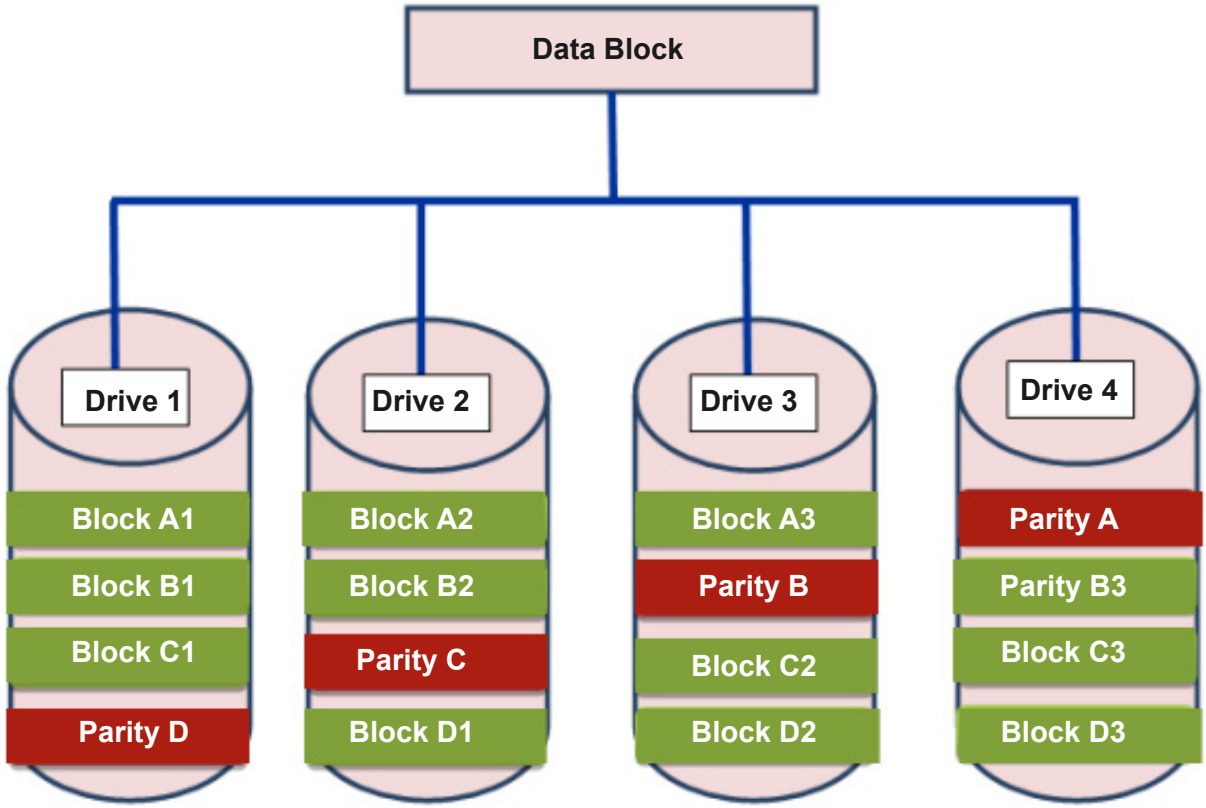


Fig. 5: RAID

Difference between RAID & Data Redundancy

Aspect	RAID (Redundant Array of Independent Disks)	Data Redundancy
Definition	A storage technology that combines multiple disks to improve performance and/or reliability.	The practice of storing multiple copies of data to ensure its availability.
Primary purpose	Enhances performance, reliability, and sometimes redundancy, depending on the RAID level.	Ensures data availability and protection against data loss.
Implementation	Uses various RAID levels (e.g., RAID 0, 1, 5, 6, 10) that distribute data and parity across disks.	Can be achieved through backups, replication, mirroring, or RAID setups.
Protection scope	Typically protects against disk failure within a single RAID array.	Protects against data loss or corruption, sometimes across different systems.
Performance impact	Some RAID levels (e.g., RAID 0, RAID 10) improve read/write speed along with redundancy.	No direct impact on performance; focused solely on data availability.
Cost	Can require multiple disks and specific configurations, increasing hardware costs.	Varies based on redundancy method; backup and replication can be costly, especially with remote storage.
Common use cases	Used in servers, data centers, and high-performance storage systems.	Used in disaster recovery, remote backups, and data archiving.
Example	RAID 1 mirrors data across two disks for redundancy.	Cloud storage services keep multiple data copies for reliability.

Table 1: RAID vs. Data Redundancy

Self-Assessment Questions

1. What is the primary purpose of memory management in an operating system?

    a) To enhance graphics performance
    b) To manage the allocation and de-allocation of memory for processes
    c) To increase the speed of the CPU
    d) To improve the quality of user interfaces

2. Which of the following memory management techniques involves moving processes between main memory and secondary storage?

    a) Paging
    b) Swapping
    c) Compaction
    d) Fragmentation

3. In the context of multiple partition allocation strategies, which method allocates a process to the first available partition that is large enough?

    a) Best Fit
    b) Worst Fit
    c) First Fit
    d) Random Fit

4. Which RAID configuration focuses on performance by striping data across multiple disks but does not provide redundancy?

    a) RAID 0
    b) RAID 1
    c) RAID 5
    d) RAID 10

5. What is the main difference between RAID and data redundancy?

    a) RAID is a type of backup, while data redundancy is a hardware solution.
    b) RAID combines multiple disks for performance and reliability; data redundancy involves storing multiple copies of data.
    c) Data redundancy only applies to cloud storage, while RAID applies to local disks.
    d) RAID increases performance without concern for data loss, while data redundancy improves data loss protection.

4.1.2 Fragmentation and Paging

Fragmentation

1. Fragmentation is the condition where memory is inefficiently used, resulting in wasted space.
2. Fragmentation occurs in two forms
    - **Internal fragmentation**: Occurs when allocated memory exceeds the memory required by the process, causing leftover space within a partition. Internal fragmentation is common in fixed-size partitioning schemes.

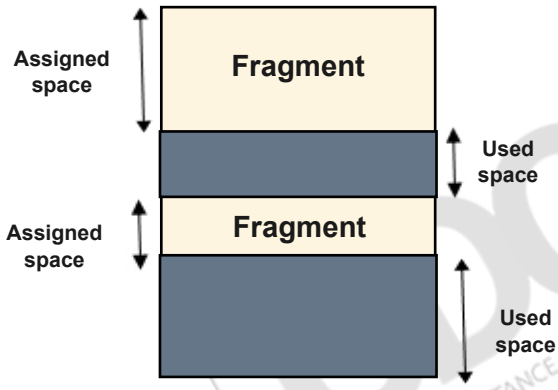


Fig. 6: Internal fragmentation

- **External fragmentation**: Occurs when free memory is scattered in small blocks between allocated segments, preventing large processes from being allocated due to lack of contiguous space. This problem can be mitigated by compaction or more sophisticated memory management techniques.

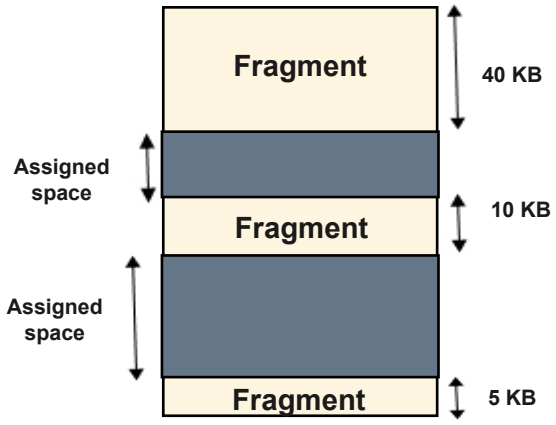


Fig. 7: External fragmentation

Paging and demand paging

1. Paging is a memory management scheme that eliminates the need for contiguous allocation by dividing memory and processes into fixed-size pages.
2. Each page is loaded into any available memory frame, facilitating easier memory allocation and preventing fragmentation.
3. Key components include

- **Demand paging**: In demand paging, pages are loaded into memory only when they are needed, rather than all at once. This approach conserves memory and speeds up processes by loading only the essential parts of a program, reducing unnecessary memory usage.
- **Page table**: A data structure that maintains the mapping between the virtual memory addresses of pages and their physical addresses in memory.

Page replacement

1. Page replacement is the process of swapping out pages from memory to bring in new ones when the memory is full. This occurs when a page fault happens (a page needed by a process is not in memory).
2. Several algorithms manage page replacement to optimise memory performance:

- **FIFO (First-In-First-Out)**: Removes the page that has been in memory the longest.
- **LRU (Least Recently Used)**: Replaces the page that has not been used for the longest time, aiming to keep frequently accessed pages in memory.
- **Optimal page replacement**: Removes the page that will not be needed for the longest period in the future, though this algorithm requires future knowledge and is mostly theoretical.

Example: Source (https://afteracademy.com/blog/what-are-demand-paging-and-pre-paging/)

Lets us understand this with the help of an example. Suppose we have to execute a process P having four pages as P0, P1, P2, and P3. Currently, in the page table, we have page P1 and P3.

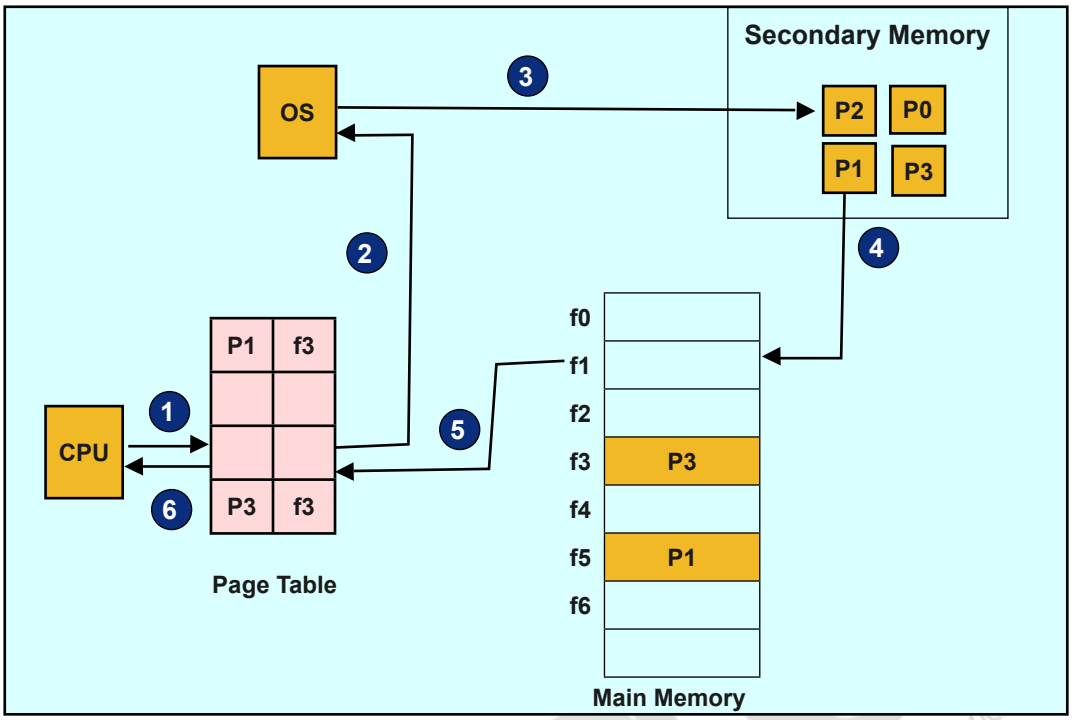


Fig. 8: Example of paging

1. Now, if the CPU wants to access page P2 of a process P, first it will search the page in the page table.
2. As the page table does not contain this page so it will be a trap or page fault . As soon as the trap is generated and context switching happens and the control goes to the operating system.
3. The OS system will put the process in a waiting/ blocked state. The OS system will now search that page in the backing store or secondary memory.
4. The OS will then read the page from the backing store and load it to the main memory.
5. Next, the OS system will update the page table entry accordingly.
6. Finally, the control is taken back from the OS and the execution of the process is resumed.

Thrashing

1. Thrashing is a performance degradation issue where excessive page swapping (paging in and out) occurs due to a lack of sufficient memory, leading to reduced CPU efficiency.
2. When the system spends more time managing memory and swapping pages than executing processes, it enters a thrashing state.

3. Thrashing can be mitigated by
    - **Working set model**: Adjusting the set of pages needed by a process in memory to maintain an optimal level.
    - **Load control**: Limiting the number of concurrent processes to prevent overloading the memory.

Self-Assessment Questions

6. What type of fragmentation occurs when allocated memory exceeds the memory required by the process, resulting in unused space within a partition?

    a) External fragmentation
    b) Internal fragmentation
    c) Memory leakage
    d) Segmentation fault

7. Which memory management technique eliminates the need for contiguous allocation by dividing memory and processes into fixed-size pages?

    a) Segmentation
    b) Paging
    c) Compaction
    d) Clustering

8. What is the purpose of the page table in a paging system?

    a) To manage CPU scheduling
    b) To maintain the mapping between virtual memory addresses and their physical addresses in memory
    c) To store the contents of all loaded pages
    d) To handle page faults

9. What is thrashing in the context of memory management?

    a) The process of allocating memory to a new process
    b) A state where excessive page swapping occurs, degrading performance
    c) The method of compressing data in memory
    d) A technique for optimising cache usage

10. Which of the following page replacement algorithms removes the page that has been in memory the longest?

    a) LRU (Least Recently Used)
    b) FIFO (First-In-First-Out)
    c) Optimal page replacement
    d) Random replacement

Summary

- Memory management techniques and file-system structures within an operating system. Memory management plays a critical role in efficiently allocating and protecting memory resources, supporting multitasking by managing primary memory (RAM) through processes like swapping and partition allocation (First Fit, Best Fit, Worst Fit).
- RAID technology and data redundancy, which protect data integrity and enhance storage performance.
- Fragmentation, common memory inefficiency, is explained with distinctions between internal and external fragmentation.
- Paging and demand paging optimise memory use by loading only necessary pages, reducing unnecessary memory load.
- Page replacement algorithms (FIFO, LRU, Optimal) are also discussed to illustrate how pages are managed when memory is full.
- The concept of thrashing highlights performance issues when the system struggles to handle memory load due to excessive page swapping, along with solutions like the working set model and load control.
- Memory allocation strategies, address fragmentation, and appreciate file-system organisation essentials in modern operating systems.

Terminal Questions

1. Explain the concept of memory management and its importance in an operating system.
2. Describe the functions and objectives of swapping in memory management.
3. Discuss the differences between First Fit, Best Fit, and Worst Fit allocation strategies.
4. Compare and contrast RAID 0, RAID 1, and RAID 5 in terms of redundancy and performance.
5. Define internal and external fragmentation, and discuss how each impacts memory efficiency.

Answer Keys

Self-Assessment Questions	
Question No.	**Answer**
1	B
2	B
3	C
4	A
5	B
6	B
7	B
8	B
9	B
10	B

Activity

Activity Type: Offline **Duration: 1 day**

Explain how swapping enables a multitasking operating system to manage limited physical memory effectively. Illustrate this by describing a scenario where a process is swapped out and another is brought into memory.

Glossary

- **Address binding:** The process of mapping program addresses to physical memory addresses during execution.
- **Allocation unit:** The smallest unit of memory that can be allocated, often a block or page.
- **Backing store:** Secondary storage used to hold pages that are swapped out of main memory.
- **Base register:** A hardware register containing the starting address of the process in memory, used in address calculations.
- **Stack segment:** Memory segment used to hold a program's local variables and manage function calls.
- **Swapping space:** Disk area reserved for holding pages swapped out of memory to manage virtual memory.

Bibliography

Textbooks

- Stallings, W. (2013). *Operating systems* (9th ed.). Philadelphia, PA: Pearson Education.
- Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems* (4th ed.). Upper Saddle River, NJ: Pearson.

External Resources

- Russinovich, M. E., Solomon, D. A., & Ionescu, A. (2012). *Windows(r) Internals* (5th ed.). Redmond, WA: Microsoft Press.
- Love, R. (2012). *Linux Kernel Development* (3rd ed.). Boston, MA: Addison-Wesley Educational.

e-References

- **Memory management techniques:** https://www.includehelp.com/operating-systems/memory-management-techniques-in-operating-system.aspx
- **Fragmentation and paging:** https://www.enterprisestorageforum.com/hardware/paging-and-segmentation/

Video Links

Video	Links
Memory management techniques	https://youtu.be/lxus-whQDnk
Fragmentation and paging	https://youtu.be/dz9Tk6KCMIQ

Image Credits

- **Fig. 1:** https://icon-library.com/images/memory-icon-png/memory-icon-png-12.jpg
- **Fig. 2:** Self-Made
- **Fig. 3:** Self-Made
- **Fig. 4:** Self-Made
- **Fig. 5:** Self-Made
- **Fig. 6:** Self-Made
- **Fig. 7:** Self-Made
- **Fig. 8:** Self-Made
- **Table 1:** Self-Made

Keywords

- Heap memory
- LRU algorithm
- Memory compaction
- Pre-emptive allocation
- Resident set
- Segmentation fault

MODULE 4

Memory Management, Fragmentation, and File System Organisation

Unit 2

File System Organisation and Management

Unit Table of Contents

Unit 4.2 File System Organisation and Management

Aim

The main aim of this unit is to detail file system organisation and management.

Instructional Objectives

This unit intends to:

- Define the core components of a file-system, including structure, access methods, directory organisation, implementation, and protection mechanisms
- Identify different access methods and directory structures within file systems and explain their purposes in data organisation
- List the various file-system implementation techniques and protection mechanisms used to ensure data integrity and security

Learning Outcomes

At the end of this unit, students are expected to:

- Describe how file-system structure, access methods, and directory organisation facilitate efficient data storage, retrieval, and management
- Explain the differences between file-system implementation techniques, such as contiguous, linked, and indexed allocation, and their impact on storage performance
- Discuss various file protection mechanisms, including access permissions and encryption, and how they contribute to data security and integrity in a file system

4.2.1 File-System Structure

The file-system structure is the foundational architecture that organises and manages files on storage devices. It is responsible for defining how data is stored, organised, and accessed by the operating system and users. At a high level, a file system provides a systematic way to store, retrieve, and organise files, enabling efficient use of storage media such as hard drives, SSDs, and external storage devices. The structure typically includes:

1. **Logical vs. Physical Structure**

   (i) The logical structure of a file system provides an abstracted view of files and folders that users interact with, hiding the complex underlying details.

   (ii) The physical structure refers to the actual arrangement of data blocks on storage devices. For example, sectors and tracks on a hard disk are part of its physical structure.

2. **Layers of a File-System**

   (i) The file-system can be divided into layers, each with specific tasks, from low-level storage to high-level file management. The main layers include:

   - **Logical file system**: Manages user access, permissions, and file naming.
   - **File organisation module**: Organises data into file structures.
   - **Basic file system**: Manages the reading and writing of data blocks.
   - **I/O control**: Handles device drivers and interactions with the hardware.

3. **File Control Block (FCB)**

   (i) The File Control Block is a data structure that contains information about a file, such as file permissions, size, creation date, and location on the disk. It plays a critical role in maintaining file integrity and providing access control.

(a) Access methods

Access methods refer to the ways data within a file can be accessed or retrieved. Different applications and use cases require different access methods, and file systems support multiple methods to cater to these needs:

1. **Sequential access**

   (i) Sequential access is the simplest method where data in a file is accessed in a linear fashion from beginning to end. It is typically used in applications where data is read or written in order, such as media streaming or text file processing.

(ii) This method is efficient for large files that require full reads but can be limiting for tasks requiring quick random access to data.

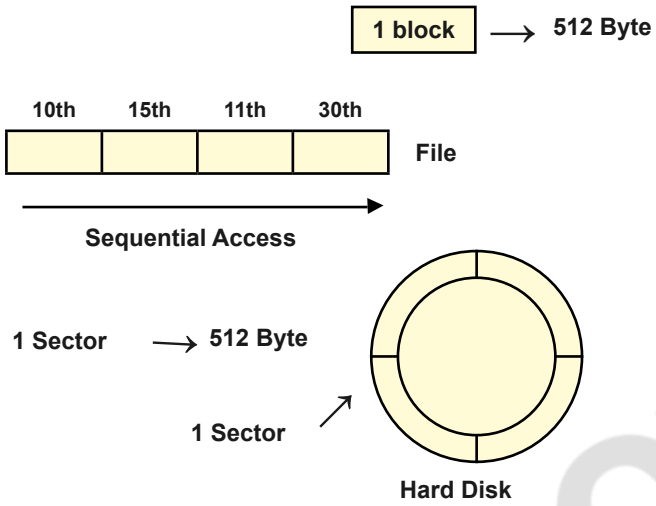


Fig. 1: Sequential access

2. Direct access (Random access)

(i) In contrast, direct access enables data retrieval from any point in the file without reading the preceding data, allowing for faster operations.

(ii) Used in database systems and applications that need to access records or blocks of data directly, direct access offers flexibility but may demand more complex file structures.

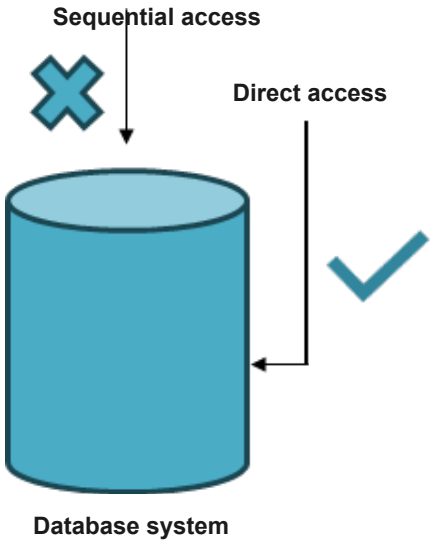


Fig. 2: Direct access

3. Indexed access

(i) Indexed access involves using an index table that maps file contents to specific locations on the disk, facilitating fast access to records.

(ii) Often used in large databases, indexed access allows efficient searches and random access, though it can increase storage requirements due to the index overhead.

4. Hashed access

(i) Hashing provides an access method based on hashed keys. In this method, file contents are mapped to physical locations using a hash function, making it possible to quickly access data.

(ii) This approach is particularly useful in large data systems where speed is critical, although managing hash collisions is a known challenge.

(b) Directory structure

The directory structure is an organisational scheme used by file systems to arrange files and directories (or folders) in a hierarchy. It helps users and applications locate files and maintain an orderly arrangement of data.

1. Single-level directory

(i) In a single-level directory, all files reside in a single directory. While this is simple and easy to implement, it lacks scalability, as a large number of files in a single directory can lead to name conflicts and make file management challenging.

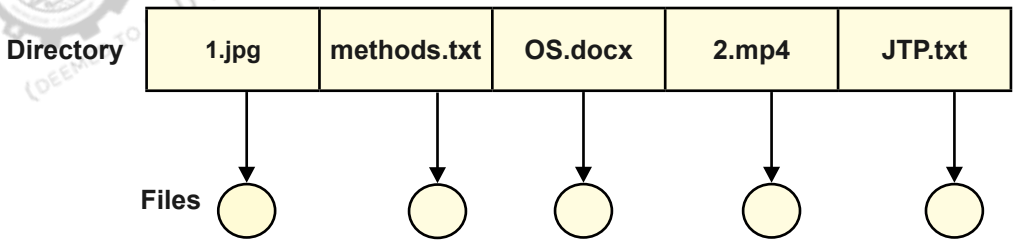


Fig. 3: Single-level directory

2. Two-level directory

(i) A two-level directory introduces a per-user directory structure, separating files based on user ownership. Each user has their own directory, reducing name conflicts.

(ii) This structure is useful in multi-user environments but can still be restrictive when managing complex data needs

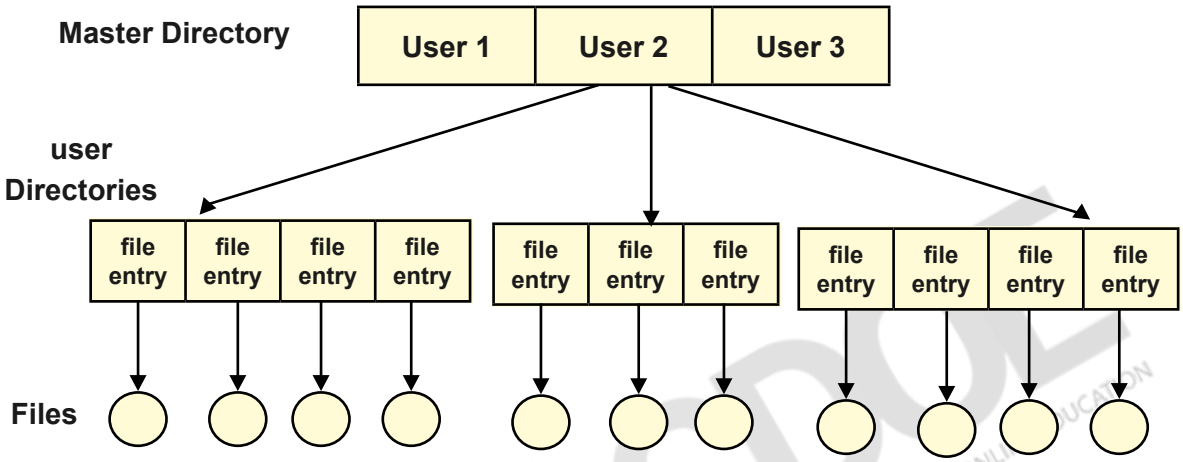


Fig. 4: Two-level directory

3. Tree-structured directory

(i) A tree-structured directory is a hierarchical model that allows files and subdirectories within directories, forming a tree. This structure provides flexibility, supports complex file organisation, and is widely used in modern operating systems.

(ii) Users can create nested folders, enabling a logical arrangement of files, but complexity increases with the depth of the hierarchy.

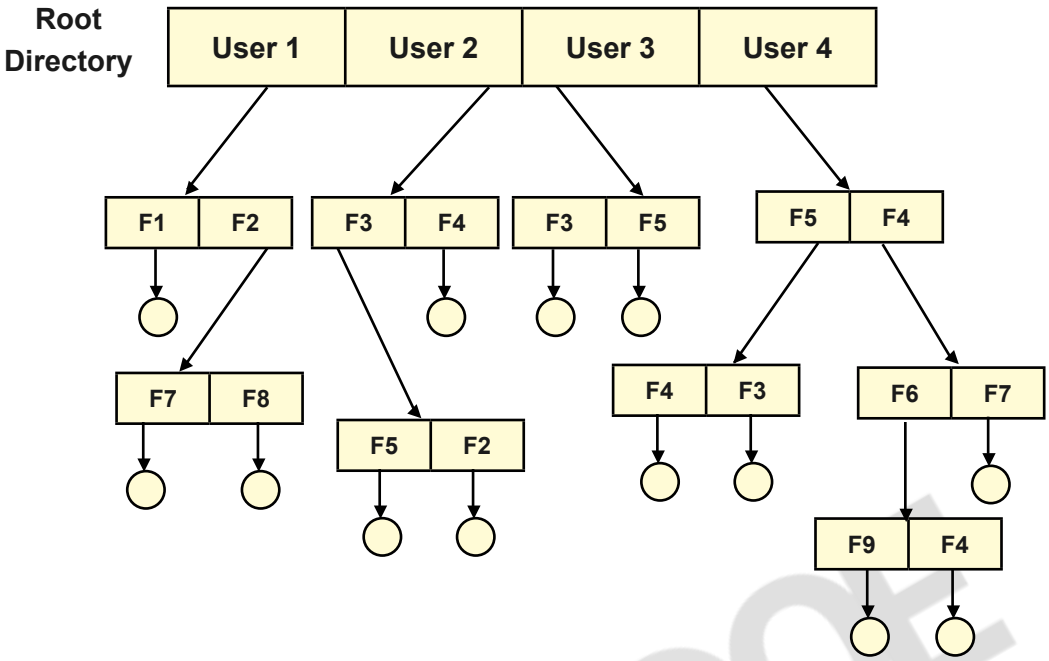


Fig. 5: Tree structured directory

4. Acyclic-graph directory

(i) In an acyclic-graph directory, a more complex structure allows files or directories to have multiple parent directories. This creates a directed acyclic graph where files can be shared among directories.

(ii) Although it provides flexibility, managing file permissions and handling cycles can be complex in this setup.

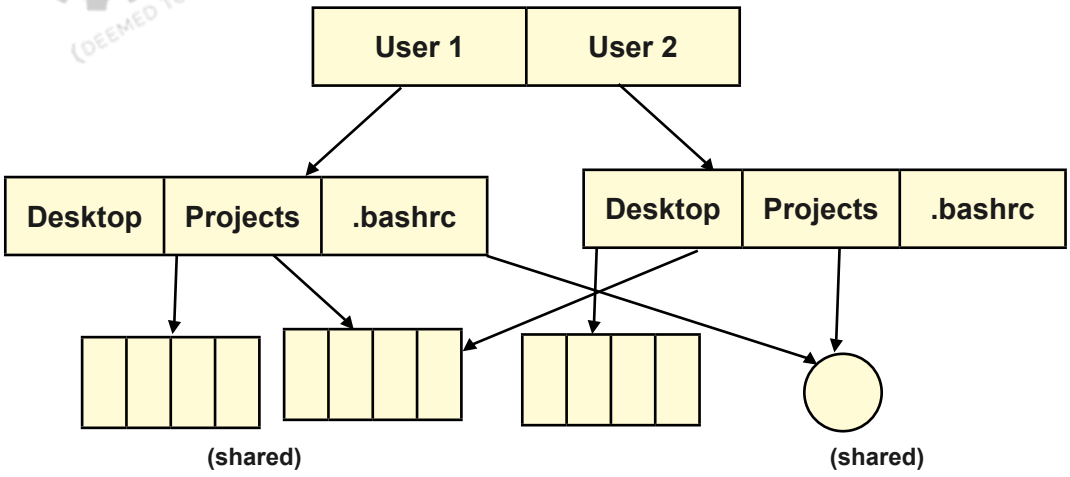


Fig. 6: Acyclic-graph directory

5. **General graph directory**

   (i) A general graph directory allows cycles in the directory structure, meaning directories can reference each other, creating loops.

   (ii) While flexible, cycles can lead to complications such as infinite loops in file traversal. Most systems prevent these cycles by using reference counters or garbage collection.

(c) File-system implementation

Implementing a file system involves various techniques to store, retrieve, and manage data on storage devices. It typically includes strategies for file allocation, free-space management, and system-level organisation to ensure efficient storage.

1. **File allocation methods**

   (i) **Contiguous allocation:** Files are stored in contiguous blocks on the disk, which simplifies access but can lead to fragmentation as files grow or shrink.

   (ii) **Linked allocation:** Each file is a linked list of disk blocks scattered across the disk. While this method avoids fragmentation, it may slow down access due to pointer traversal.

   (iii) **Indexed allocation:** An index block holds pointers to all blocks of a file, offering efficient random access. However, it may require additional storage for the index table.

2. **Free-space management**

   (i) File systems must manage free space to allocate blocks for new files and de-allocate space from deleted files. Techniques include:

   - **Bit vector (Bitmap)**: Tracks free blocks using bits (1 for free, 0 for occupied).
   - **Linked list**: Links free blocks in a chain, making it easier to allocate/de-allocate but potentially inefficient for large spaces.
   - **Grouping and counting**: Aggregates free blocks in groups for faster access, which can be more efficient in large storage systems.

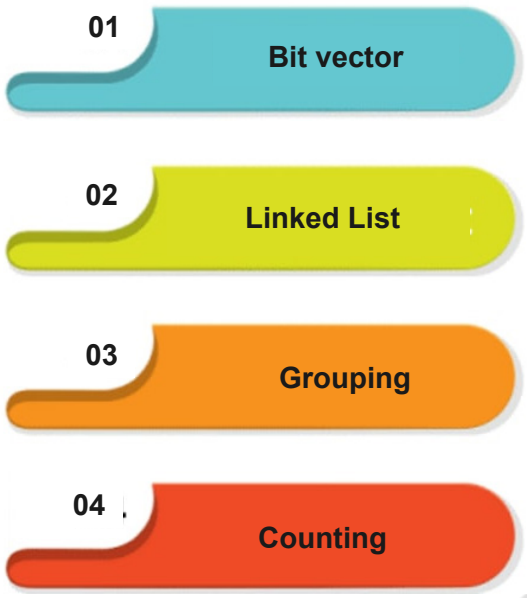


Fig. 7: Free-space management

3. File-system mounting

(i) Mounting is the process of making a file system accessible to users by integrating it into the directory structure. An un-mounted file system isn't available until mounted at a designated mount point.

(ii) The mount point acts as an access point, linking the system's root structure to the file system being mounted. In UNIX-based systems, for instance, a new file system might be mounted under another specified path.

4. File-system consistency and recovery

(i) Ensuring consistency is crucial to maintaining data integrity. File systems incorporate checks like journaling, where changes are logged before they are made, ensuring that they can be rolled back if interrupted.

(ii) Recovery mechanisms are in place to manage unexpected shutdowns or crashes. By checking for inconsistencies, the file system can repair itself or restore data from backup structures.

(d) Protection

File protection mechanisms are essential to restrict unauthorised access, ensure data privacy, and maintain system integrity. These mechanisms are particularly important in multi-user environments, where data security and integrity are critical.

1. **File access permissions**

   (i) Access permissions define what actions (read, write, execute) are permitted for different users. Common permission models include:

   - **Owner, Group, Others**: This model, seen in UNIX systems, assigns permissions based on file ownership.
   - **Role-Based Access Control (RBAC)**: Permissions are based on roles (e.g., admin, user), which are more flexible than basic ownership models.

2. **Access Control Lists (ACLs)**

   (i) ACLs provide fine-grained control by allowing permissions to be assigned to specific users or groups, enhancing security beyond basic permission schemes.

   (ii) Each file or directory can have an associated ACL specifying allowed actions for different entities, making it easy to manage permissions in complex environments.

3. **Authentication and authorisation**

   (i) Authentication verifies a user's identity, while authorisation grants or denies access based on predefined rules. Both are critical in file systems to ensure that only legitimate users can access specific files.

   (ii) Techniques include passwords, biometrics, and multi-factor authentication, ensuring secure access at different levels of the system.

4. **Encryption**

   (i) File systems often employ encryption to protect data at rest. Encryption algorithms encode data, making it accessible only to those with decryption keys.

   (ii) By encrypting sensitive data, file systems prevent unauthorised users from interpreting the contents, even if they bypass other security measures.

Self-Assessment Questions

1. What is the primary role of a file system in an operating system?

    a) Manage user interfaces
    b) Organise and manage files on storage devices
    c) Perform hardware diagnostics
    d) Execute programs

2. Which of the following best describes the logical structure of a file system?

    a) It represents the actual arrangement of data blocks on a storage device.
    b) It provides an abstract view of files and folders for user interaction.
    c) It controls hardware device drivers.
    d) It manages CPU scheduling.

3. In a file system, what is the purpose of the File Control Block (FCB)?

    a) To store user data
    b) To manage system memory
    c) To contain metadata about a file, such as permissions and location
    d) To monitor system performance

Self-Assessment Questions

4. Which access method is most commonly used for media streaming applications?

   a) Sequential access
   b) Direct access
   c) Indexed access
   d) Hashed access

5. What is a key characteristic of direct access in a file system?

   a) Files must be accessed in a specific order.
   b) Data can be accessed from any point within the file.
   c) Files are stored in contiguous memory blocks.
   d) Access requires reading the entire file.

6. Which directory structure is most commonly used in modern operating systems?

   a) Single-level directory
   b) Two-level directory
   c) Tree-structured directory
   d) Acyclic-graph directory

Summary

- File-system structure organises and manages data on storage devices, providing a structured way for users and the operating system to access and organise files.
- File systems are layered, with the logical file system handling permissions, file naming, and user interactions, while lower layers manage data organisation, block storage, and hardware communication.
- Key components, such as the File Control Block (FCB), store metadata critical for file access and integrity.
- File access methods include sequential, direct, indexed, and hashed access, each suited to specific application needs, from linear file reads to rapid data retrieval.
- Directory structures, ranging from single-level to general graphs, provide different ways of organising files, with each model offering trade-offs in complexity and flexibility.
- Implementation aspects like file allocation, free-space management, and mounting allow efficient storage, while consistency mechanisms and recovery protocols ensure data integrity during system failures.
- File protection features, such as permissions, Access Control Lists (ACLs), authentication, and encryption, secure the file system, especially in multi-user environments, maintaining data privacy and preventing unauthorised access.

Terminal Questions

1. What are the key differences between the logical and physical structures of a file system?
2. Describe the purpose of the File Control Block (FCB) and what metadata it stores.
3. Explain the difference between sequential access and direct access methods, and give examples of when each would be used.
4. List and describe the types of directory structures available in file systems. What are the pros and cons of each?
5. What is file-system mounting, and why is it essential for making a file system accessible?

Answer Keys

Self-Assessment Questions	
Question No.	**Answer**
1	B
2	B
3	C
4	A
5	B
6	C

Activity

Activity Type: Offline **Duration: 1 day**

Create a sample folder hierarchy that includes at least three levels of directories and subdirectories. Document the names and layout of these folders to illustrate how a tree-structured directory system is organised.

Glossary

- **Encryption:** The process of encoding data to protect it from unauthorised access, accessible only with a decryption key.
- **Reference Counter:** Tracks the number of references to a file, aiding in cycle prevention in directory structures.
- **Mount Point:** A directory location where a file system is made accessible in the OS directory hierarchy.
- **Tree traversal:** Navigating a directory structure to locate files, often recursively in a hierarchical system.
- **Hash collision:** When two inputs produce the same hash value, potentially complicating data retrieval.

Bibliography

Textbooks

- Stallings, W. (2013). *Operating systems* (9th ed.). Philadelphia, PA: Pearson Education.
- Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems* (4th ed.). Upper Saddle River, NJ: Pearson.

External Resources

- Russinovich, M. E., Solomon, D. A., & Ionescu, A. (2012). *Windows(r) Internals* (5th ed.). Redmond, WA: Microsoft Press.
- Love, R. (2012). *Linux Kernel Development* (3rd ed.). Boston, MA: Addison-Wesley Educational.

e-References

- **Access methods:** https://www.javatpoint.com/os-file-access-methods
- **Directory structure:** https://www.scaler.com/topics/directory-structure-in-os/
- **File-system implementation:** https://ebooks.inflibnet.ac.in/csp3/chapter/implementation-of-file-systems/

Video Links

Video	Links
Access methods	https://youtu.be/0vno0Ra0wEg
Directory structure	https://youtu.be/OVOYAFZcWdY
File-system implementation	https://youtu.be/2MlaqSmklhM?list=PLskQvPDUk0sKgGGReUz7nMwYkDTbevUC1

Image Credits

- **Fig. 1:** Self-Made
- **Fig. 2:** Self-Made
- **Fig. 3:** Self-Made
- **Fig. 4:** Self-Made
- **Fig. 5:** Self-Made
- **Fig. 6:** Self-Made
- **Fig. 7:** Self-Made

Keywords

- File allocation
- Contiguous allocation
- Linked allocation
- Indexed allocation
- Free-space management
- File-system mounting

OPERATING SYSTEMS

(DEEMED TO BE UNIVERSITY)
CDOE
CENTER FOR DISTANCE AND ONLINE EDUCATION

https://www.kluniversity.in/cdoe/

supportcdoe@kluniversity.in