MODULE 1

Introduction to Operating Systems

Comprehensive Study Material

Course Code: DCSE250042

B.Sc. Computer Science | Semester 3

Table of Contents

1.1 What is an Operating System? 3
1.2 Operating System Functions 4
1.3 Multiprogramming Operating Systems 5
1.4 Time-Sharing Operating Systems 7
1.5 Real-Time Operating Systems (RTOS) 8
1.6 Multiprocessor Operating Systems 10
1.7 Multicore Operating Systems 11
1.8 Various Computing Environments 12
1.9 Comparison of OS Types 13
1.10 Quick Revision Notes 14
1.11 Practice Questions 15

1.1 What is an Operating System?

1.1.1 Definition

> An Operating System (OS) is system software that acts as an intermediary between the user and computer hardware. It manages hardware resources and provides an environment for application programs to execute.

1.1.2 Goals of an Operating System

Goal	Description
Convenience	Make the computer easy to use for humans
Efficiency	Use hardware resources efficiently
Ability to Evolve	Allow development and updates
Resource Management	Fairly allocate resources among processes
Protection & Security	Prevent unauthorized access

1.1.3 Two Main Views of an OS

View 1 — Resource Manager: The OS manages all hardware resources (CPU, Memory, I/O Devices, Files, Network, Security). It allocates, tracks, deallocates resources and resolves conflicts.

View 2 — Control Program: The OS controls execution of user programs to prevent errors and unauthorized access, acting as a supervisor that decides when each program runs and what it can access.

1.1.4 Why Do We Need an OS?

Without an OS, developers must write low-level code to initialise hardware, handle interrupts, manage memory, and control devices — taking months and working only on specific hardware. With an OS, developers write simple calls (e.g. printf("Hello")) and the OS handles everything — resulting in stable, secure programs that work across hardware.

1.1.5 Computer System Components

Layer	Description
Hardware	CPU, Memory, I/O Devices
Operating System	Provides basic services to applications, manages hardware
Application Programs	Word, Excel, Browser, Games, Compilers, Databases
Users	People who use the computer

> Key Insight: The OS is the ONLY software that interacts directly with hardware. All other software must go through the OS.

1.1.6 Kernel and System Programs

Kernel: The core of the OS that runs in privileged (kernel) mode. Handles process scheduling, memory management, file systems, device drivers, networking, and interrupt handling.

System Programs: Utilities that use kernel services, running in user mode — e.g. shell, compilers, editors, databases, web servers.

System Calls: The interface between user-mode programs and the kernel.

1.2 Operating System Functions

1.2.1 Main Functions (P-M-F-I-S-N)

Function	Description	Example
Process Management	Creating, scheduling, terminating processes	Running multiple programs
Memory Management	Allocating and tracking memory	Loading programs into RAM
File Management	Organising and controlling file access	Creating, reading, writing files
I/O Management	Controlling input/output devices	Printing, reading from keyboard
Security & Protection	Preventing unauthorised access	User accounts, permissions
Networking	Managing network communications	Internet access, file sharing

1.2.2 Process Management

The Process Manager loads programs from disk, creates a Process Control Block (PCB), allocates resources, schedules CPU time, manages process states (Ready / Running / Waiting), and handles termination.

Example: When you open Chrome, the OS loads chrome.exe, creates a new process with a unique PID, allocates memory, and schedules it to run on the CPU.

1.2.3 Memory Management

- Track which parts of memory are in use
- Allocate memory to processes when needed
- Deallocate memory when processes finish
- Manage virtual memory (using disk as extension)

- Protect memory of one process from another
- Handle memory shortages (swap to disk)

1.2.4 File Management

- Create and delete files and directories
- Support file operations (read, write, seek)
- Map files to disk blocks; manage free space
- Provide file protection (permissions) and implement file sharing

1.2.5 I/O Management

- Device drivers for each device
- Buffering (temporary storage during transfer)
- Caching (keep frequently used data in faster memory)
- Spooling (queue output for slow devices, e.g. printers)
- Error handling and device scheduling

1.3 Multiprogramming Operating Systems

1.3.1 Definition

> Multiprogramming: Multiple programs are kept in memory simultaneously. When one program waits for I/O, the CPU switches to another ready program — keeping the CPU busy at all times.

1.3.2 Key Insight

In single programming, the CPU sits idle during I/O operations (which are 100-1000x slower than CPU speed). Multiprogramming eliminates this wasted time by running other programs during those I/O waits.

1.3.3 Worked Example

Three programs — A (CPU 5ms, I/O 20ms, CPU 5ms), B (CPU 3ms, I/O 10ms, CPU 7ms), C (CPU 2ms, I/O 15ms, CPU 4ms):

Metric	Without Multiprogramming	With Multiprogramming
Total Time	71 ms	~36 ms
CPU Idle Time	27 ms (38% wasted)	0 ms
CPU Utilisation	62%	100%

1.3.4 Advantages and Disadvantages

Advantages	Disadvantages
Better CPU utilisation	Complex OS design
More efficient use of memory	Need memory protection
Increased throughput	May cause starvation
Higher performance	Overhead of context switching
Batch processing possible	Need scheduling algorithms

1.3.5 Related Concepts

Concept	Key Feature
Multiprogramming	Multiple programs in memory; CPU switches on I/O
Multitasking	Same as multiprogramming + time sharing
Multiprocessing	Multiple CPUs
Multithreading	Multiple threads within one process

1.4 Time-Sharing Operating Systems

1.4.1 Definition

> Time-Sharing (Multitasking): An extension of multiprogramming where the CPU switches between programs so quickly (every 10–100ms quantum) that each user feels they have a dedicated computer.

1.4.2 How It Works

A hardware timer fires every quantum (e.g. 50ms). The interrupt handler saves the current process state, selects the next process (scheduling algorithm), loads its state, and resumes it. This cycle repeats continuously.

1.4.3 Time-Sharing vs. Multiprogramming

Feature	Multiprogramming	Time-Sharing
Primary goal	Maximise CPU utilisation	Minimise response time
Switching trigger	I/O wait	Timer interrupt (quantum expiry)
User interaction	None (batch)	Interactive

Feature	Multiprogramming	Time-Sharing
Response time	Seconds to minutes	Milliseconds to seconds
Fairness	Not important	Very important
Typical environment	Mainframes, batch systems	Personal computers, servers

1.4.4 Advantages and Disadvantages

Advantages	Disadvantages
Interactive computing	More complex OS
Fast response time	More overhead (timer interrupts)
Fair resource sharing	Need memory protection
Many simultaneous users	Potential for thrashing
Better user experience	Security concerns

1.5 Real-Time Operating Systems (RTOS)

1.5.1 Definition

> An RTOS is designed to serve applications that process data as it arrives with guaranteed timing constraints. The key requirement is determinism — operations must complete within specified deadlines, every time.

1.5.2 Hard RTOS vs. Soft RTOS

Feature	Hard RTOS	Soft RTOS
Missed deadline	System failure (catastrophic)	Reduced quality (tolerable)
Timing guarantee	Absolute — must be guaranteed by design	Best-effort
Applications	Airbags, pacemakers, aircraft controls, nuclear reactors	Video streaming, VoIP, online gaming
Examples	VxWorks, QNX	Multimedia OS, Linux with PREEMPT-RT

1.5.3 Hard RTOS Applications

- Automotive airbag: must deploy within 10–15ms of crash detection — late = injury or death
- Medical pacemaker: must detect arrhythmia and respond within milliseconds — late = heart attack

- Aircraft fly-by-wire: must process sensor data within 1ms — late = loss of control
- Industrial robot: must stop within 0.5ms of emergency signal — late = worker injury

1.5.4 RTOS Requirements

Requirement	Description	Why Important
Deterministic	Same input → same output time	Predictable behaviour
Low latency	Fast interrupt handling	Meet deadlines
Priority scheduling	Higher priority tasks run first	Critical tasks get CPU
Priority inversion handling	Prevent low-priority blocking high-priority	Avoid missed deadlines
Minimal jitter	Consistent response times	Reliable operation
Resource locking	Control access to shared resources	Prevent conflicts

1.5.5 Priority Inversion Problem

Priority inversion occurs when a medium-priority task (M) effectively blocks a high-priority task (H) because H is waiting for a resource held by a low-priority task (L), which is itself preempted by M.

Step	Event
1	Task L acquires shared resource R
2	Task H wakes up, tries to acquire R — BLOCKED (waiting for L)
3	Task M wakes up (higher priority than L) — preempts L
4	Task H still blocked (L can't run because M is running)
Result	Medium-priority task runs before high-priority task!

Solutions: Priority Inheritance Protocol (L temporarily inherits H's priority) or Priority Ceiling Protocol (resource has a ceiling priority).

1.6 Multiprocessor Operating Systems

1.6.1 Definition

> A Multiprocessor OS manages a system with two or more physical CPUs that share the same memory.

1.6.2 SMP vs. AMP

Feature	SMP (Symmetric)	AMP (Asymmetric)
CPU roles	All equal	Master–Slave
OS location	Can run on any CPU	Master CPU only
Load balancing	Automatic	Manual
Reliability	Higher (no single point of failure)	Master failure = system down
Complexity	Higher	Lower
Scalability	Limited (memory bus bottleneck)	Better
Common use	Desktop, servers	Embedded systems

1.6.3 Advantages

- Increased throughput — more CPUs = more work done per unit time
- Cost effective — cheaper than multiple separate computers
- Increased reliability — one CPU fails, others continue (graceful degradation)
- Load sharing — work distributed among CPUs
- Scalability — can add more CPUs (up to physical limit)

1.6.4 Challenges

- Cache coherency — multiple CPU caches may hold different values of the same variable
- Memory bus contention — all CPUs share same bus; becomes bottleneck at high core count
- Scheduling complexity — deciding which CPU should run which process (cache affinity)
- Synchronisation — more locking needed when multiple CPUs access shared data

1.7 Multicore Operating Systems

1.7.1 Definition

> A multicore processor has multiple processing cores on a single chip. Unlike multiprocessor systems, cores share an on-chip L3 cache, enabling faster inter-core communication.

1.7.2 Cache Hierarchy

Cache Level	Scope	Size	Speed
L1	Per core (private)	32 KB	Fastest
L2	Per core (private)	256 KB	Fast

Cache Level	Scope	Size	Speed
L3	Shared by all cores	8–16 MB	Slower than L1/L2 but much faster than RAM
Main Memory (RAM)	System-wide	GBs	Slowest

1.7.3 Multicore vs. Multiprocessor

Aspect	Multicore	Multiprocessor
Physical packaging	Single chip	Multiple chips
Communication speed	Very fast (on-chip bus)	Slower (through motherboard)
Cache sharing	Shared L3 cache	Separate caches per chip
Power consumption	Lower	Higher
Cost	Lower	Higher
Scalability	Limited (heat, power)	Better (add more chips)
Example	Intel Core i7 (12 cores)	Dual Xeon server (2 chips × N cores)

1.8 Various Computing Environments

Environment	OS Type	Key Feature	Examples
Traditional	General-purpose	Interactive, GUI-based	Windows, macOS, Linux
Mobile	Mobile OS	Power-efficient, touch, sensors	Android, iOS
Distributed	Network OS	Resource sharing across network	Cluster, grid computing
Cloud	Virtualised	Scalable, pay-as-you-go	AWS, Azure, GCP
Embedded	RTOS or specialised	Deterministic, resource-constrained	VxWorks, FreeRTOS

1.8.1 Mobile Computing

- Power-efficient OS with app sandboxing for security
- Touch interface with gesture recognition
- Limited resources: battery, memory, storage
- Always connected (cellular, WiFi); push notifications; cloud sync
- Sensors: GPS, accelerometer, camera, fingerprint reader

1.8.2 Cloud Computing Service Models

Model	Full Name	What's Provided	Example
IaaS	Infrastructure as a Service	Virtual machines, storage, networking	AWS EC2, Azure VMs
PaaS	Platform as a Service	Runtime, middleware, development tools	Google App Engine, Heroku
SaaS	Software as a Service	Ready-to-use applications over internet	Gmail, Office 365, Salesforce

1.8.3 Embedded Computing Characteristics

- Resource-constrained: limited memory and CPU power
- Often battery-powered; optimised for low energy
- Real-time requirements (frequently uses RTOS)
- Single-purpose by design; no or minimal user interface
- Long uptime expected — years without rebooting

1.9 Comparison of OS Types

Feature	Batch	Multiprogramming	Time-Sharing	Hard RTOS	Soft RTOS
User interaction	No	No	Yes	Minimal	Yes
Response time	Hours	Minutes	Seconds	Microseconds	Milliseconds
CPU utilisation	Medium	High	High	Critical	High
Fairness	Low	Low	High	N/A	Medium
Complexity	Low	Medium	High	Very High	High
Deadline guarantee	No	No	No	Yes	No
Typical use	Old mainframes	Batch processing	Desktops, servers	Medical, automotive	Multimedia

---

1.10 Quick Revision Notes

Key Definitions

Term	Definition
Operating System	Software that manages hardware and provides services to applications
Kernel	Core of the OS that runs in privileged mode

Term	Definition
Multiprogramming	Multiple programs in memory; CPU switches on I/O wait
Time-Sharing	Multiple users interactively; CPU switches on timer interrupt
RTOS	OS with guaranteed timing constraints
SMP	Symmetric Multiprocessing — all CPUs equal
Multicore	Multiple cores on a single chip
Throughput	Amount of work completed per unit time
Response time	Time between request and first response

OS Functions Mnemonic: P-M-F-I-S-N

- P — Process Management
- M — Memory Management
- F — File Management
- I — I/O Management
- S — Security & Protection
- N — Networking

7 Critical Points to Remember

- OS is the ONLY software that directly accesses hardware — everything else goes through the OS.
- Multiprogramming ≠ Time-Sharing — multiprogramming switches on I/O wait; time-sharing switches on a timer.
- Hard RTOS guarantees deadlines — used in safety-critical systems where missing a deadline means failure.
- Multicore ≠ Multiprocessor — multicore has cores on the same chip; multiprocessor has separate chips.
- Kernel mode vs User mode — kernel can execute privileged instructions; user programs cannot.
- Context switching has overhead — saving/restoring process state wastes some CPU time.
- Interrupts are key to time-sharing — hardware timer interrupts enable preemptive multitasking.

Key Formulae

Metric	Formula
CPU Utilisation (approx.)	1 − (I/O wait time / Total time)
Throughput	Number of processes completed / Total time
Response time	Time from request to first response

1.11 Practice Questions

Short Answer (2–3 marks)

1. Define operating system. What are the main goals of an OS?
2. Differentiate between multiprogramming and time-sharing systems.
3. What is a Real-Time OS? Give two examples of hard real-time applications.
4. What is the difference between SMP and AMP?
5. What is a multicore processor? How is it different from a multiprocessor?
6. List any five functions of an operating system.
7. Why is time-sharing called 'interactive' computing?
8. What is throughput? How does multiprogramming affect it?
9. Give three examples of embedded computing environments.
10. What is the difference between batch processing and time-sharing?

Medium Answer (5–7 marks)

11. Explain with diagrams how multiprogramming improves CPU utilisation.
12. Compare batch, multiprogramming, and time-sharing operating systems.
13. Describe hard real-time and soft real-time systems with examples.
14. Explain symmetric and asymmetric multiprocessing architectures.
15. What are the different computing environments where an OS operates?

Long Answer (10–15 marks)

16. Draw and explain the layered structure of a computer system showing hardware, OS, applications, and users.
17. A system has three programs: A (CPU: 4ms, I/O: 15ms, CPU: 3ms), B (CPU: 2ms, I/O: 8ms, CPU: 2ms), C (CPU: 1ms, I/O: 12ms, CPU: 2ms). Calculate total execution time and CPU utilisation with and without multiprogramming.
18. Explain the concept of time-sharing in detail. Include: time quantum, timer interrupts, context switching, and response time considerations.
19. Compare and contrast multiprocessor and multicore systems. Include advantages, disadvantages, and architectural differences.
20. Describe real-time operating systems in detail. Include: hard vs. soft RTOS, priority scheduling, priority inversion problem, and applications.

Multiple Choice Questions

1. Which of the following is NOT an OS function?

a) Process management
b) Memory management
c) Database management
d) I/O management

2. In multiprogramming, the CPU switches to another process when the current process:

a) Completes
b) Performs I/O
c) Uses CPU
d) Starts

3. Time-sharing systems use _____ to switch between processes.

a) I/O interrupts
b) Timer interrupts
c) System calls
d) User commands

4. Which type of RTOS has the strictest timing requirements?

a) Soft RTOS
b) Firm RTOS
c) Hard RTOS
d) All are same

5. In SMP (Symmetric Multiprocessing):

a) One master CPU controls others
b) All CPUs are equal
c) CPUs have different roles
d) Only one CPU runs OS

6. A multicore processor has:

a) Multiple CPU chips
b) Multiple cores on one chip
c) Multiple computers
d) Multiple operating systems

7. Which computing environment typically uses battery-powered devices?

a) Traditional
b) Mobile
c) Distributed
d) Embedded

8. The core of the operating system is called:

a) Shell
b) Kernel
c) BIOS
d) Firmware

> Answers: 1-c, 2-b, 3-b, 4-c, 5-b, 6-b, 7-b, 8-b

References & Further Study

- Silberschatz, Galvin & Gagne — Operating System Concepts (Chapters 1–2)
- Previous year question papers for DCSE250042
- Online OS simulation tools for visualising scheduling and memory management