

UNIT 1

Introduction to Operating System

(2 Hrs) — Operating System (CAC251)

1.1 Introduction to Operating System

An **Operating System (OS)** is system software that manages the computer hardware and provides a convenient, efficient environment in which application programs can run. It acts as an **intermediary between the user and the hardware**. Without an operating system, every program would have to contain its own code to drive the disk, the keyboard, the screen and the memory — which is impractical and error-prone.

Examples of operating systems include Windows, Linux, macOS, Android and iOS. A useful informal definition is: *“the one program that is always running on the computer” — the kernel*. Everything else is either a system program (shipped with the OS) or an application program.

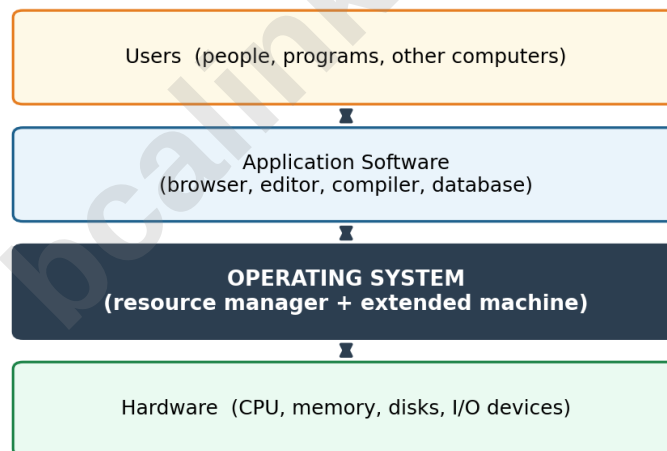


Figure 1.1 — Position of the operating system in a computer system

Components of a computer system

Component	Role	Examples
Hardware	Provides basic computing resources	CPU, memory, disks, I/O devices
Operating system	Controls and coordinates use of hardware among applications and users	Windows, Linux, macOS
Application programs	Solve the users' computing problems	Browsers, word processors, compilers

Component	Role	Examples
Users	People, machines or other computers that use the system	Students, administrators

1.2 Objectives of an Operating System

The operating system can be viewed from two complementary points of view.

(a) Operating System as a Resource Manager (top-down vs bottom-up view)

A computer has many resources — CPU time, memory space, file storage, I/O devices. Several programs compete for these resources. The OS acts as a **resource manager** (or resource allocator): it keeps track of who is using which resource, grants requests, resolves conflicting requests fairly and efficiently, and protects programs from one another.

- **Time multiplexing** — resources shared in time (e.g., CPU is given to one process after another).
- **Space multiplexing** — resources shared in space (e.g., memory is divided among many processes; a disk holds many users' files).

(b) Operating System as an Extended Machine (virtual machine)

Raw hardware is ugly and hard to program. Programming a disk controller directly means dealing with registers, sectors, seek commands and error codes. The OS hides these details and presents the programmer with a clean, simple abstraction such as a **file**. Thus the OS **extends** the machine: it turns the ugly bare hardware into a beautiful, easy-to-use virtual machine.

Example — Reading from a disk

Without an OS a programmer must issue low-level commands: set the cylinder, head and sector, start the motor, wait for the interrupt, check for errors.

With an OS the same task is simply: **read(file, buffer, size)**. The OS handles everything underneath.

Resource Manager view	Extended Machine view
Bottom-up view — manages hardware	Top-down view — serves the programmer
Concerned with efficiency and fairness	Concerned with convenience and simplicity
Allocates CPU, memory, I/O among competing programs	Hides hardware complexity behind abstractions like files and processes

1.3 History and Generations of Operating Systems

Operating systems have evolved along with computer hardware. The history is conventionally divided into generations.

Generation	Period	Hardware	Software / OS characteristics
First	1945 – 1955	Vacuum tubes, plugboards	No operating system. Programs entered in machine language by operators; one user at a time; very slow set-up.
Second	1955 – 1965	Transistors, mainframes	Batch systems: jobs punched on cards, grouped in batches and run one after another by a resident monitor. Languages: FORTRAN, assembly.
Third	1965 – 1980	Integrated circuits (ICs)	Multiprogramming, time-sharing , spooling. Examples: IBM OS/360, MULTICS, early UNIX.
Fourth	1980 – present	LSI/VLSI, personal computers	Personal computers, GUIs, networked and distributed OS. Examples: MS-DOS, Windows, Linux, macOS.
Fifth (modern)	2000s onward	Multi-core, mobile, cloud	Mobile OS (Android, iOS), virtualization, cloud and real-time embedded systems.

Note: Multiprogramming keeps several jobs in memory at once so that the CPU is never idle while one job waits for I/O. Time-sharing extends this by switching between jobs so rapidly that each user feels they have the machine to themselves.

1.4 Types of Operating Systems

1. Batch Operating System

Similar jobs are collected into a **batch** and executed one after another with no user interaction while the batch runs. *Advantage:* good CPU utilisation for large repetitive jobs. *Disadvantages:* no interaction, long turnaround time, CPU idle during I/O. Example: payroll and bank statement processing.

2. Multiprogramming Operating System

Several jobs reside in main memory at the same time. When the running job waits for I/O the CPU switches to another job, so the CPU stays busy. The degree of multiprogramming is the number of jobs in memory.

3. Time-Sharing (Multitasking) Operating System

The CPU is given to each user or task for a short **time slice (quantum)** in turn. Response time is short, so the system feels interactive. Examples: UNIX, Linux, Windows.

4. Multiprocessor (Parallel) Operating System

Two or more CPUs share memory, the bus and peripherals under a single OS. Advantages: increased throughput, economy of scale and higher reliability (graceful degradation). May be *symmetric* (SMP — all processors equal) or *asymmetric* (master–slave).

5. Distributed Operating System

A collection of independent computers connected by a network that appears to its users as a single coherent system. Resources and computation are shared across machines. Example: Amoeba, LOCUS; also cluster systems.

6. Network Operating System

Runs on a server and provides shared file, printer and user-account services to client machines. Users are aware of the multiple machines. Examples: Windows Server, Novell NetWare, UNIX/Linux servers.

7. Real-Time Operating System (RTOS)

Correct operation depends not only on the logical result but also on the **time** at which it is produced. A *hard real-time* system guarantees deadlines absolutely (air-traffic control, pacemakers); a *soft real-time* system tolerates occasional misses (video streaming, multimedia). Examples: VxWorks, QNX, FreeRTOS.

8. Embedded and Mobile Operating Systems

Small, specialised operating systems running inside devices such as phones, routers, ATMs, TVs and cars. Examples: Android, iOS, Embedded Linux.

Type	Key feature	Example
Batch	No interaction; jobs run in groups	IBM mainframe batch jobs
Time-sharing	Short response time, many users	UNIX, Linux
Real-time	Strict deadlines	VxWorks, QNX
Distributed	Many computers act as one	Amoeba, clusters
Network	Server provides shared services	Windows Server

Type	Key feature	Example
Mobile / embedded	Limited resources, power-aware	Android, iOS

1.5 Functions of an Operating System

Function	Description
Process management	Creating, scheduling, suspending, synchronising and terminating processes.
Memory management	Tracking memory use, allocating and de-allocating memory, virtual memory.
File management	Creating, deleting, reading, writing files and directories; access control.
Device (I/O) management	Device drivers, buffering, spooling, caching and scheduling of I/O requests.
Secondary storage management	Free-space management, allocation, disk scheduling.
Protection and security	Controlling access to resources; authentication of users.
Networking	Communication between computers; distributed computing support.
User interface (Command interpreter / GUI)	Lets the user communicate with the system via commands or graphics.
Error detection and accounting	Detecting hardware/software errors; recording resource usage per user.

System calls — interface to OS services

Programs request services from the OS using **system calls**, which switch the CPU from *user mode* to *kernel mode*. Common categories are process control, file management, device management, information maintenance and communication.

Key Takeaways

- ✓ The OS is system software that manages hardware and provides a platform for applications.
- ✓ Two views: **resource manager** (allocates CPU, memory, I/O) and **extended machine** (hides hardware complexity).
- ✓ Generations: no OS → batch → multiprogramming/time-sharing → PC and network OS → mobile/cloud.
- ✓ Types: batch, multiprogramming, time-sharing, multiprocessor, distributed, network, real-time, embedded.
- ✓ Main functions: process, memory, file, I/O and storage management, protection, and user interface.

— *End of Unit 1* —

bcalink.com