Introduction to Operating Systems (OS)

An operating system (OS) is the fundamental system software that manages computer hardware and software resources, providing common services for computer programs. Without an operating system, a computer would be unusable for most purposes.

Operating systems serve as the critical intermediary between hardware components and the applications we use daily. They provide the foundation upon which all other software runs, handling complex tasks like memory allocation, process scheduling, and device communication that would otherwise need to be programmed into every application.

Basic Functions of an Operating System

Resource Management

The OS efficiently allocates and manages system resources including CPU time, memory space, file storage, and input/output devices. It ensures that multiple programs can run simultaneously without interfering with each other, maximising system performance.

This management involves complex scheduling algorithms to determine which processes receive CPU attention and when, as well as sophisticated memory management techniques to optimise available RAM usage.

User Interface Provision

Whether through a command-line interface (CLI) or graphical user interface (GUI), the OS provides the means for users to interact with the computer system. This interface translates user commands into instructions the computer hardware can understand.

System Security Enforcement

The OS implements security measures that protect the system from unauthorised access, malicious software, and data corruption. It manages user authentication, file permissions, and system integrity checks.

System Software vs Application Software: Overview

System Software (OS)

System software, primarily the operating system, creates the platform that enables hardware-software interaction. It manages system resources and provides services to other software. The OS runs at the lowest level, directly communicating with hardware through drivers and firmware interfaces.

Without system software, the hardware components would remain disconnected islands of technology, unable to function as a cohesive computing system.

Application Software

Application software is designed to help users perform specific tasks. These programs rely on the services and resources provided by the operating system to function properly. Applications cannot communicate directly with hardware; they must make requests through the OS.

While system software is essential for the computer to operate, application software is what makes computers useful for specific purposes.

What is Application Software?

Application software refers to programs designed to perform specific functions directly for the user. Unlike operating systems that manage hardware resources, applications focus on helping users accomplish particular tasks. They exist at a higher level of abstraction, working through the services provided by the underlying operating system.

The distinguishing characteristic of application software is its purpose-driven nature. Each application is created with specific user needs in mind, whether that's writing documents, editing photos, or playing media files.

01

Task-Specific Design

Each application is built to serve a particular function or set of related functions, such as word processing, image editing, or web browsing.

02

Popular Examples

- Microsoft Word (document creation)
- Adobe Photoshop (image editing)
- VLC Media Player (multimedia playback)
- Google Chrome (web browsing)

03

OS Dependency

Applications cannot run independently; they require an operating system to provide hardware access, memory management, and other essential services.

Distinction: Application Software vs OS

Operating System

- Manages hardware resources and provides services
- Runs continuously from system boot until shutdown
- Controls access to CPU, memory, storage, and I/O devices
- Examples: Windows, macOS, Linux, Android

Application Software

- Performs specific user-oriented tasks
- Runs only when explicitly launched by user
- Relies on OS to access hardware resources
- Examples: Word processors, games, web browsers

The fundamental difference lies in their roles: the OS serves as the resource manager and platform provider, while applications are tools designed for specific user activities. Applications must request resources through the OS, which maintains control over all hardware access to ensure system stability and security. Without an OS, applications would need to implement their own hardware management code, making software development prohibitively complex.

This distinction becomes particularly evident when considering that you can replace applications without affecting the operating system, but replacing the OS requires all applications to be compatible with the new system.

Why OS is Essential

System Bootstrapping

The operating system is responsible for the bootstrap process that brings computer hardware to life during startup. When you power on a computer, the OS loads essential drivers and initializes hardware components, making them ready for use.

This complex process involves checking hardware, loading core system files into memory, and establishing the runtime environment necessary for software execution.

Application Enablement

All applications depend on the services provided by the operating system. The OS creates a stable platform that allows application developers to focus on functionality rather than hardware details.

Without an OS, each application would need its own drivers and resource management code, making software development impractical and inefficient.

Common Services Provider

The OS provides a consistent set of services that all applications can utilise, including file management, memory allocation, process scheduling, and device control.

These shared services create a standard environment that simplifies software development and ensures compatibility across different applications.

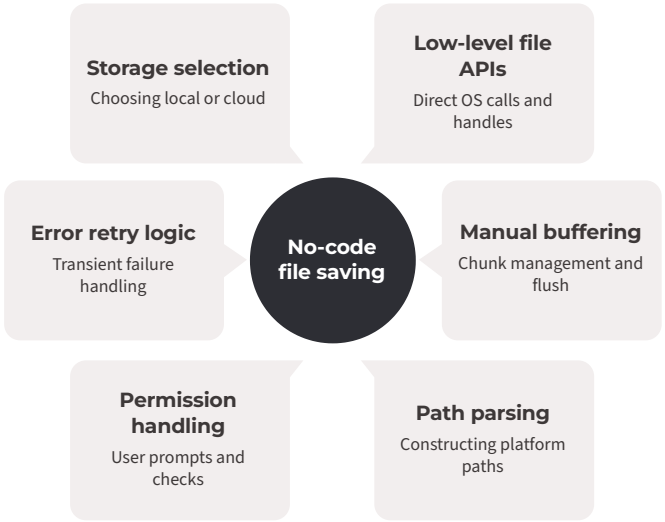
OS as the Hardware–Software Bridge

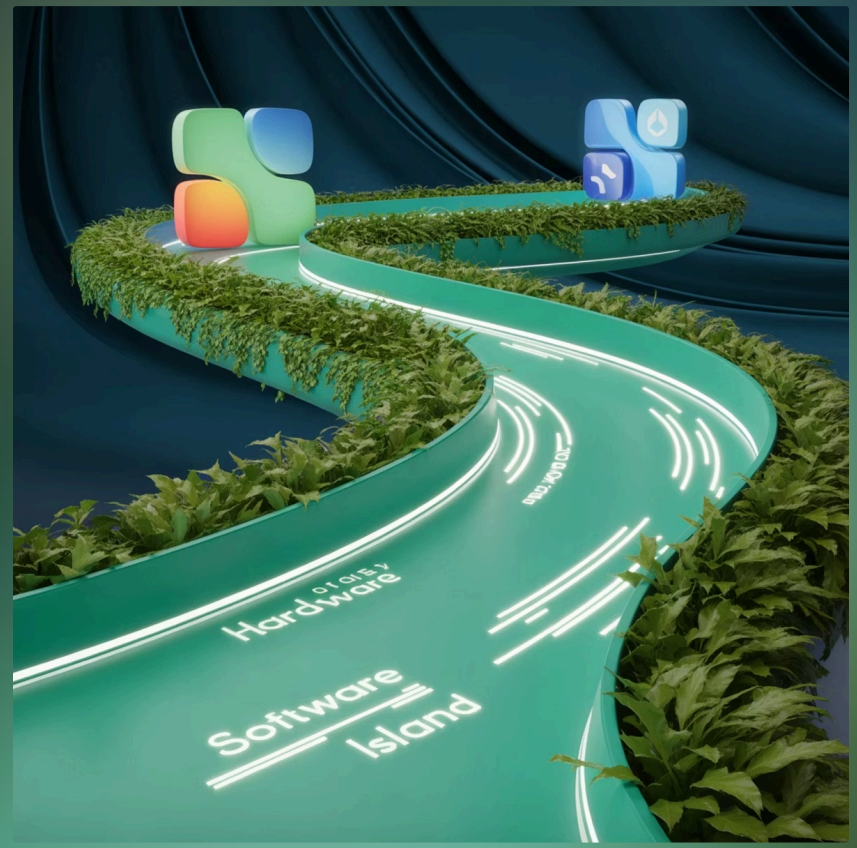
One of the most crucial functions of an operating system is to abstract hardware details away from application software. This abstraction creates a standardised interface that applications can use without needing to understand the underlying hardware specifics.

The OS serves as a translator between the high-level requests of applications and the low-level operations of hardware components. This translation process is bidirectional, with the OS both conveying application requests to hardware and returning hardware responses to applications.

Example: File Operations

When an application needs to save a file, it doesn't need code for:





Hardware Dependency: Core Concept

CPU Architecture Specificity

Operating systems must be designed for specific CPU architectures, as they need to use the processor's native instruction set. For example, Windows designed for x86/x64 processors cannot run directly on ARM-based systems without significant modification or emulation.

Hardware-Specific Optimizations

Operating systems include optimizations specific to certain hardware features, such as specialized instructions for multimedia processing or virtualization capabilities only available on certain CPU models.

Low-Level Communication

OS kernels contain machine code specific to the target hardware platform. This low-level code directly interacts with hardware registers, memory addresses, and device controllers using instructions that vary across different processor architectures.

Hardware "Language"

Each hardware platform has its own "language" (instruction set) that the OS must speak. This is why Windows was historically designed for Intel-compatible processors and couldn't run on PowerPC architectures that Apple previously used.

This hardware dependency is fundamental to operating system design and explains why you cannot simply install any OS on any hardware platform. The OS must be specifically built for the target architecture.

Hardware Dependency: Real-World Implications

Installation Failures

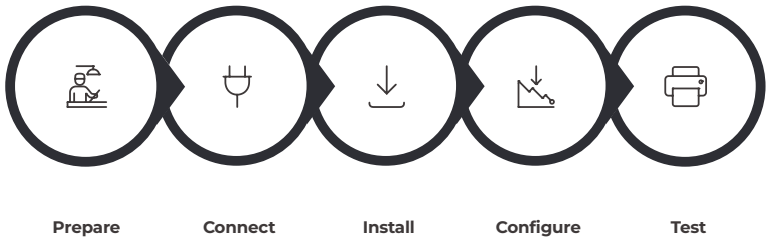
When users attempt to install an operating system on incompatible hardware, the installation typically fails completely or results in a non-functional system. This is because the OS contains code that can only execute on the specific processor architecture it was designed for.

Driver Requirements

Peripheral devices require drivers that are both OS-compatible and hardware-specific. A printer that works perfectly with Windows 10 might not function at all with Linux or macOS because the required driver translation layer is missing or incompatible.

Real-World Example: Printer Configuration

When connecting a printer to a computer system:



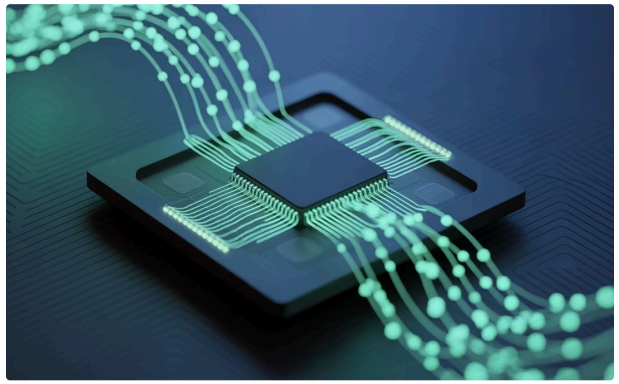
Handling Interrupts & I/O Devices

Keyboard Input

When a key is pressed, the keyboard controller generates a hardware interrupt. The OS's interrupt handler for that specific keyboard model must recognize the interrupt, read the scan code from the keyboard controller, and translate it to the appropriate character.

Storage Devices

Hard drives, SSDs, and other storage devices use different communication protocols (SATA, NVMe, etc.). The OS must implement specific interrupt handlers and device drivers for each protocol, managing data transfer and error handling appropriately.



Network Interfaces

Network cards use interrupts to signal when data packets arrive. The OS must handle these interrupts differently based on the network card model and interface type, processing incoming data according to the appropriate network protocols.

The way operating systems handle interrupts varies significantly across different hardware architectures. For example, ARM processors handle interrupts differently than x86 processors, requiring OS developers to implement architecture-specific interrupt handling code. This creates a tight coupling between the OS and the underlying hardware platform.

Memory Management and CPU Considerations

Memory Protection Mechanisms

Different CPU architectures implement memory protection in various ways. The OS must be designed to work with the specific protection mechanisms of the target processor:

- x86/x64 CPUs use segmentation and paging with privilege rings
- ARM processors use memory domains and security extensions
- RISC-V has a different privilege model with machine, supervisor, and user modes

The OS kernel must be written to utilize these hardware-specific features correctly to maintain system stability and security.

Instruction Set Architecture (ISA)

Operating systems contain low-level code that must match the CPU's instruction set. This is a core hardware dependency for the OS - it needs to be designed and optimized for the specific processor architecture it will run on.

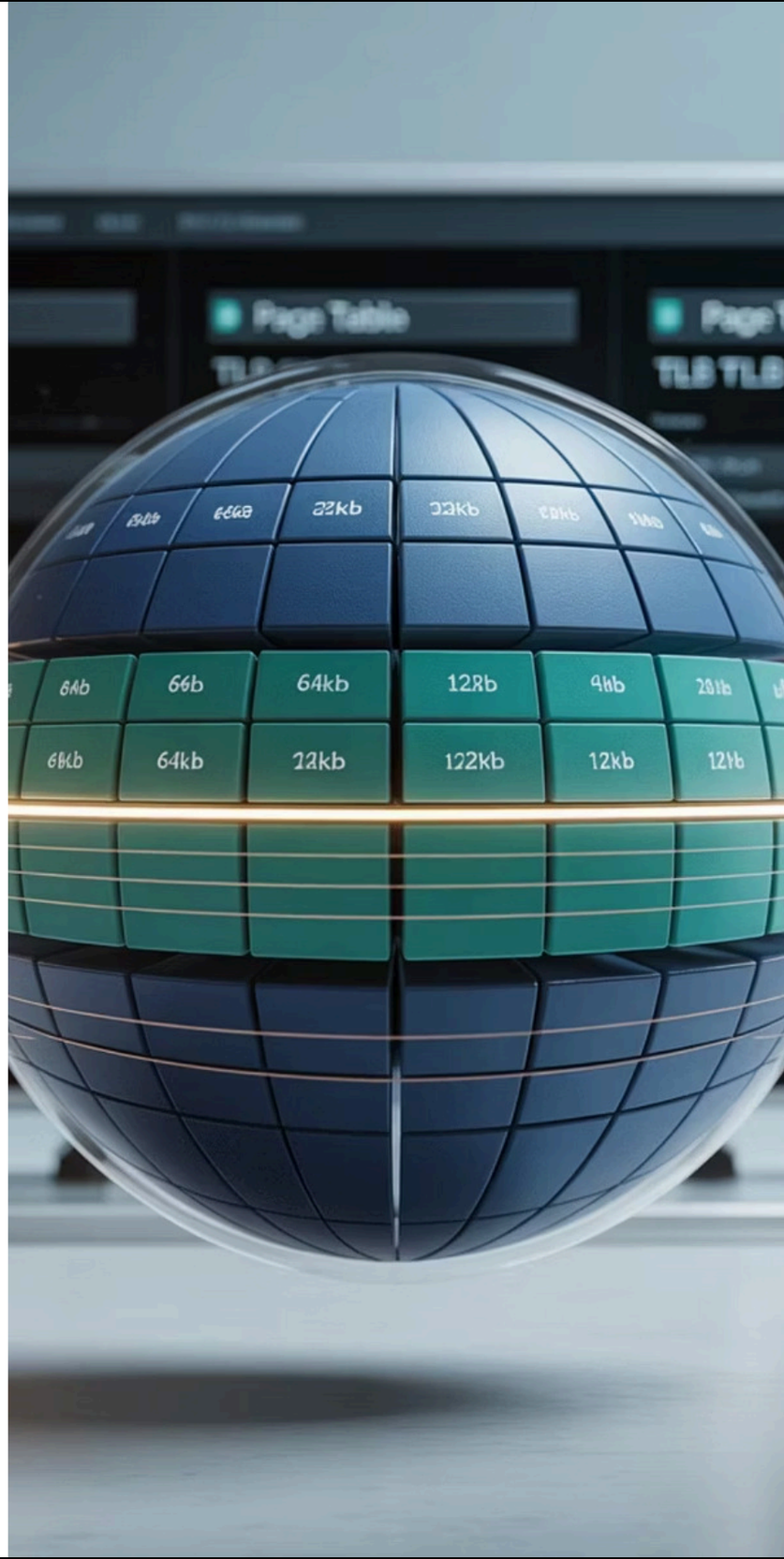
The operating system's kernel is the central component that interacts directly with the computer's hardware. It provides an abstraction layer between the application software and the physical components, allowing programs to execute without needing to know the intricate details of the underlying hardware.

This hardware-specific code within the OS kernel is responsible for tasks like:

1. **Executing machine instructions**
2. **Managing memory addresses**
3. **Handling interrupts and I/O**

The OS must be carefully designed to leverage the unique capabilities and features of the target CPU architecture. This hardware dependency extends to other system components like the file system, device drivers, and memory management, which all need to be tailored for the specific hardware platform.

Maintaining this tight integration between the OS and the hardware is essential for optimal performance, stability, and reliability. As new CPU designs emerge, operating systems must evolve to take advantage of the latest hardware advancements.



Key OS Components: Overview

Kernel

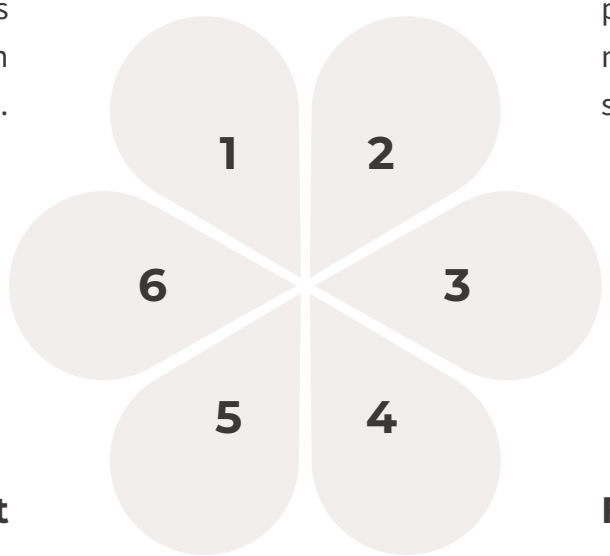
The core of the operating system that has complete control over everything in the system. It manages system resources and facilitates communication between hardware and software components.

Security & Protection

Implements authentication, authorization, and encryption. Prevents unauthorized access to system resources and protects against malicious software.

Device Management

Coordinates the use of I/O devices through appropriate drivers. Handles device interrupts and manages data transfer between devices and memory.



Process Management

Handles the creation, scheduling, and termination of processes. Manages multitasking and provides mechanisms for inter-process communication and synchronization.

Memory Management

Allocates and tracks memory usage. Implements virtual memory systems to optimize memory usage and provide process isolation.

File System Management

Organizes and controls access to files stored on various media. Manages directory structures and enforces file permissions.

These components work together to create a cohesive system that manages hardware resources, provides services to applications, and creates an environment where multiple processes can execute simultaneously. Each component is essential to the overall functioning of the operating system.

Component: Process Management

Process Creation and Termination

The OS is responsible for creating new processes when applications are launched and cleaning up resources when processes terminate. This involves:

- Allocating memory and resources for the new process
- Loading program code into memory
- Initializing runtime data structures
- Recovering resources when processes end

This process management ensures that system resources are used efficiently and not leaked over time.

Multitasking and Scheduling

Modern operating systems implement sophisticated scheduling algorithms to share CPU time among multiple processes:

Component: Memory Management

Physical Memory Allocation

The OS tracks which portions of RAM are in use and which are available. When a process needs memory, the OS allocates available blocks and maintains records of ownership to prevent processes from accessing each other's memory spaces.

This allocation system must be efficient to minimize wasted space while also being fast enough to handle frequent allocation requests.

Virtual Memory Implementation

Virtual memory creates the illusion of more RAM than physically exists by using disk space as an extension of memory. The OS manages:

- Page tables mapping virtual to physical addresses
- Page replacement algorithms when physical memory is full
- Swapping pages between RAM and disk storage

Memory Protection

The OS prevents processes from accessing memory they don't own by using hardware protection mechanisms. This isolation is critical for system stability and security, preventing:

- Accidental corruption of other processes' data
- Malicious attempts to access sensitive information
- System crashes due to memory violations

Memory management is particularly hardware-dependent as it must work directly with the CPU's memory management unit (MMU) and understand the physical memory architecture of the system.

Component: File System Management

File Organization

The operating system implements file systems that organize data on storage devices. These file systems determine how files are named, stored, and structured on the physical media. Different operating systems often use different file systems:

- Windows typically uses NTFS or FAT32
- Linux often uses ext4, XFS, or Btrfs
- macOS uses APFS or HFS+

The file system implementation must be optimized for the specific characteristics of the storage hardware, which creates another layer of hardware dependency.

Access Control

File systems implement a system of permissions that determine which users or processes can read, write, or execute specific files. This access control system is essential for several key reasons:

1 **Security**

2 **Privacy**

3 **Integrity**

By restricting access to sensitive files and directories, the file system permissions help protect the system from unauthorized access, data breaches, and malicious modifications. This security layer is crucial for safeguarding the confidentiality, integrity, and availability of data.

The permissions also enable different users to have tailored access levels, allowing some to only view files while others can edit or execute them. This granular control is essential for maintaining privacy and preventing accidental or malicious changes to important system files.

Overall, the file system's access control mechanisms are a fundamental component of an operating system's security and protection features, ensuring the reliable and trustworthy operation of the computer system.

Component: Device Management

Device Drivers

Device drivers are specialized software components that facilitate communication between the OS and specific hardware devices. Each type of hardware requires its own driver that understands:

- The device's communication protocols
- How to initialize and configure the device
- Methods for sending commands and receiving data
- Error handling procedures specific to the device

Interrupt Handling

When hardware devices need attention, they generate interrupts that temporarily pause the CPU's current work. The OS must:

- Recognize the interrupt source
- Save the current execution context
- Execute the appropriate interrupt handler
- Restore the original context when finished

I/O Scheduling

For devices like disk drives, the OS implements I/O scheduling algorithms that:

- Optimize the order of requests to minimize seek time
- Prioritize critical I/O operations
- Balance performance and fairness among competing processes
- Maximize throughput while maintaining reasonable response times

Device management is perhaps the most hardware-dependent component of an operating system, as each device type requires specific knowledge of its operational characteristics. This is why new hardware often requires updated drivers when moving to a new OS version.

Component: Security & Protection

Authentication & Authorization

The OS implements mechanisms to verify user identities and control access to resources:

- Password verification and management
- Biometric authentication (fingerprints, facial recognition)
- Multi-factor authentication systems
- Role-based access control for system resources

These systems ensure that only authorized users can access sensitive data and system functions.

Additional Security Features

Modern operating systems implement numerous security measures to protect the system and its data from threats:

1 Access Control

OS file systems use permissions to restrict which users and processes can access, modify, or execute specific files and directories.

2 Privilege Separation

The OS runs most programs in a limited user mode, only granting elevated "root" or "admin" privileges to trusted system processes.

3 Sandboxing

Application sandboxes isolate programs from the core OS and each other, preventing malware from spreading across the system.

These security features help protect the confidentiality, integrity, and availability of the system and its data. They defend against unauthorized access, data breaches, and malicious modifications.

The OS security model also enables different users to have tailored access levels, allowing some to only view files while others can edit or execute them. This granular control is essential for maintaining privacy and preventing accidental or malicious changes to important system files.

Overall, the operating system's robust security architecture is a fundamental component for safeguarding the computer system and the user's data. It provides the essential safeguards needed for reliable and trustworthy computing.

Other Important OS Features

Scalability

Modern operating systems must scale effectively across devices ranging from small embedded systems to massive server clusters. This requires flexible resource management that can adapt to:

- Different CPU architectures and core counts
- Varying memory configurations from megabytes to terabytes
- Storage systems ranging from simple flash drives to complex RAID arrays
- Network capabilities from basic connectivity to high-performance clustering

Compatibility

Operating systems must maintain compatibility with a diverse ecosystem of hardware and software:

- Supporting legacy hardware through appropriate drivers
- Providing backward compatibility for older applications
- Implementing standard interfaces and protocols
- Supporting cross-platform technologies when possible

Customisability

Different users have different needs, so operating systems must be adaptable:

- Configurable user interfaces to match user preferences
- Modular designs allowing features to be added or removed
- Tunable performance parameters for different workloads
- Accessibility options for users with different abilities

These features represent the balance operating systems must strike between power and usability. A good OS must be both technically sophisticated enough to leverage hardware capabilities efficiently and user-friendly enough to be accessible to its target audience. This balance varies across different operating systems, with some prioritizing advanced features and others focusing on ease of use.

Common OS Examples

Microsoft Windows

Dominates the desktop and laptop market, particularly in business environments. Windows is designed primarily for x86/x64 architecture processors from Intel and AMD. Recent versions include Windows 10 and Windows 11.

Windows has evolved to support ARM architecture for some devices, though with compatibility limitations for legacy software.

macOS

Apple's desktop operating system exclusively designed for their hardware. Originally developed for PowerPC processors, Apple transitioned macOS to Intel x86/x64 architecture in 2006, and more recently to their custom ARM-based Apple Silicon.

This transition demonstrates how deeply an OS is tied to hardware architecture, requiring significant engineering effort to change platforms.

Linux

An open-source OS kernel that forms the foundation for numerous distributions (Ubuntu, Fedora, Debian). Linux is remarkably adaptable and runs on more processor architectures than any other OS, including x86/x64, ARM, RISC-V, POWER, and many others.

This adaptability makes Linux the dominant OS for servers, embedded systems, and supercomputers.

Android

Google's mobile operating system based on a modified Linux kernel. Android is primarily designed for ARM processors, which dominate the mobile device market due to their energy efficiency.

Android's hardware dependency is particularly evident in the device-specific customizations required for each smartphone or tablet model.

Conclusion: The OS in Everyday Computing

Essential System Foundation

The operating system serves as the indispensable foundation of modern computing, enabling all other software to function. Without an OS, our devices would be collections of disconnected components rather than useful tools.

By providing standardized interfaces, resource management, and security services, the OS creates the platform that allows application developers to focus on solving specific problems rather than dealing with hardware complexity.

Hardware-OS Relationship

The hardware dependency of operating systems is not a limitation but a necessity born from the OS's role as the intermediary between hardware and software. This dependency allows the OS to fully leverage hardware capabilities while providing appropriate abstractions for applications.

Integrated Components

The various components of an operating system—process management, memory management, file systems, device management, and security—work together as an integrated whole. Each component depends on the others, creating a cohesive system that: