Part 1: Basics

What is an Operating System (OS)? There is actually "no universally accepted definition". A common approximation is that an OS is just "everything a vendor ships when you order an operating system," though this varies wildly between companies. But an operating system is a program that acts as the bridge between you (the user) and the physical computer hardware. Its main goals are to execute your programs, make the computer convenient to use, and ensure the hardware is used efficiently.

To achieve this, the OS acts as two things:

- **A Resource Allocator:** It manages all the computer's resources (like memory and CPU power) and decides how to fairly distribute them when different programs request them at the same time.
- **A Control Program:** It controls how programs execute to prevent unexpected errors and stop any improper use of the computer.

Different Points of View: The OS acts differently depending on the type of computer.

- **Personal Computers:** Users just want convenience and ease of use, and don't care about how efficiently resources are used.
- **Mainframes/Shared Computers:** The OS must focus heavily on resource utilization to keep all shared users happy.
- **Handheld Computers:** The OS prioritizes usability and battery life because resources are poor.
- **Embedded Computers:** Systems like those in cars have little or no user interface.

The Kernel The core of the operating system is called the **kernel**. The kernel is the one essential program that is constantly running on your computer from the moment you turn it on. Everything else on your computer is either an application program or an extra system program.

The Four Components of a Computer System Every computer system can be broken down into four distinct layers:

1. **Users:** This can be people, machines, or other computers.
2. **Application Programs:** These are the tools used to solve problems, like word processors, web browsers, compilers, and video games.
3. **Operating System:** This layer controls and coordinates how the hardware is used by the various applications and users.
4. **Hardware:** The physical computing resources, such as the CPU, memory, and input/output (I/O) devices.

Computer Startup When you first power on or reboot your PC, it needs a simple way to start running. It uses an initial piece of code called a **bootstrap program**.

- This program is permanently stored in read-only memory (ROM or EEPROM), which is generally known as firmware which is specifically the programming written directly to read-only memory (ROM). You can also define **EPROM** as "Erasable Programmable Read-Only Memory," which is the specific type of chip that safely retains its data when the power supply is switched off. This ensures the startup code is not lost when the power is turned off.
- The bootstrap program wakes up and initializes all the physical parts of the system. Finally, it loads the OS kernel into the computer's memory so the system can actually start executing your commands.

Part 2: Interrupts

1. Computer System Organization & Operation

Inside your computer, one or more CPUs and various "device controllers" (like a USB controller or a disk controller) are all connected by a **common bus**.

- **Visualizing the Bus:** Think of the bus as a central highway. The CPU and all devices use this shared highway to access the shared main memory.
- **How they operate:** The CPU and devices work concurrently (at the same time), which means they actively compete for turns to use the memory highway.
- **Buffers:** Each device controller is in charge of a specific type of device and has a "local buffer" (a temporary waiting room for data). The CPU moves data between the main memory and these local buffers, while the device itself handles moving data in and out of the buffer.

2. Interrupts (How devices get the CPU's attention)

An operating system is entirely **interrupt-driven**, meaning it multitasks by responding to events. When a device finishes its task, it sends an "interrupt" signal to the CPU. There are two types of interrupts:

- A Hardware Interrupt (with Device-Status Table & ISR)

1. **The Setup:** When a program first requests asynchronous I/O, the OS records this in the **device-status table**. This table tracks the type, memory address, and current state of every single I/O device on the computer. The OS marks the device as "busy" and the CPU goes back to other work.
2. **The Trigger:** The physical I/O device finishes its background work and sends an electrical interrupt to the CPU.

3. **Preserving the State:** The Kernel immediately takes control by switching bit modes and preserves the exact state of the currently running user program. It does this by saving the CPU's registers and the "program counter" (which acts as a bookmark pointing to the very next instruction the user program was supposed to execute). It also temporarily disables other incoming interrupts so that another signal doesn't overwrite what the CPU is currently doing, preventing a "lost interrupt".
4. **Checking the Table:** The Kernel needs to figure out which specific device caused the interrupt. It does this either through polling (sending a signal out to every single device asking, "Did you make this request?") or by using a vectored interrupt system (where the device automatically identifies itself to the CPU).

   It then looks up that device in the **device-status table** to confirm its status and modifies the table entry to note that the interrupt has arrived by changing it to “idle”.

5. **The Interrupt Service Routine (ISR):** The OS needs to know how to process this specific device's data. It uses the **interrupt vector**, which is an array containing the unique memory addresses for all the different service routines. The CPU jumps to the correct address and executes the **ISR**—this is the actual, separate segment of OS code that handles the specific action required for that device’s requested interrupt.
6. **The Return:** The ISR finishes, the Kernel restores the saved registers and program counter, and the CPU seamlessly resumes the user program exactly where it was paused.

- A Software Interrupt (Trap or Exception)

1. **The Trigger:** A user program either causes an error (like dividing by zero) or requests an OS service via a system call.
2. **The Mode Switch:** The hardware switches the mode bit from User mode (1) to Kernel mode (0) to protect the system. Now in Kernel mode, the CPU is allowed to read the Kernel's privileged instructions to safely process the system call or handle the software error.
3. **The Interrupt Service Routine (ISR):** The CPU jumps to a specific segment of OS code designed to handle that exact system call or error.
4. **The Return:** Once the Kernel has finished its work, a return instruction resets the mode bit back to User mode (bit 1), and the CPU goes back to executing the user program.

3. I/O (Input/Output) Structure

There are two ways the computer handles I/O requests:

1. Synchronous I/O (Waiting)

- The user program makes an I/O request, switching the mode bit from 1 to 0 (Kernel mode).
- The Kernel takes control and tells the specific device to start working.

- **The Key Difference:** The Kernel *keeps* control and **does not** switch the mode bit back to User mode (1). Control returns to the user program *only* after the I/O is completely finished. The CPU simply sits idle using a "wait instruction" or a "wait loop" until the device sends an interrupt. Because of this waiting, at most one I/O request can be outstanding at a time.

2. Asynchronous I/O (Not Waiting)

- The user program makes the I/O request, switching the mode bit to 0 (Kernel mode).
- The Kernel tells the device to start and logs the device's state in the device-status table.
- **The Key Difference:** The Kernel immediately switches the mode bit back to 1 (User mode) and returns control to the user program *without* waiting for the I/O to finish. The CPU goes right back to executing your program's instructions while the device does its physical work in the background.
- Even in an Asynchronous setup, a user program can make a specific **"System call"** to ask the operating system to allow it to wait for the I/O completion anyway.

4. Direct Memory Access (DMA)

DMA is a special structure used for high-speed devices that transmit data at speeds close to memory speeds.

- Instead of the CPU moving data byte-by-byte from the device buffer to main memory, the device controller transfers entire blocks of data directly to main memory without the CPU's intervention.
- This is highly efficient because it generates only **one interrupt per block** of data, rather than one interrupt per byte, freeing up the CPU to do other work.

5. Storage Definitions (A quick review)

- The basic unit is a **bit** (0 or 1).
- A **byte** is 8 bits (the smallest chunk a computer easily moves).
- A **word** is the computer architecture's native unit of data (e.g., a 64-bit computer has an 8-byte word).
- Sizes scale up: Kilobyte (1,024 bytes), Megabyte ($1,024^2), Gigabyte ($1,024^3), Terabyte ($1,024^4), and Petabyte ($1,024^5).

Part 3: Storage Structure

Storage Structure Types

- **Main Memory (RAM):** This is the only large storage media the CPU can access directly. It provides random access but is **volatile**, meaning it completely loses its content when the power is switched off.
- **Secondary Storage:** This acts as an extension of main memory, providing a massive amount of **non-volatile** storage capacity that survives without power.

- **Magnetic Disks:** The most common secondary storage. These are rigid metal or glass platters covered with a magnetic recording material. The physical disk surface is logically divided into rings called **tracks**, which are then subdivided into **sectors**. A disk controller handles the logical interaction between the disk drive and the computer.
- **Tertiary Storage:** Even slower backup storage like optical disks or magnetic tape. This storage can vary between WORM (write-once, read-many-times) and RW (read-write) formats.

The Storage Hierarchy and Caching

Computer storage systems are organized into a strict hierarchy based on three factors: **speed, cost, and volatility**.

Here is exactly how the hierarchy scales from the fastest and smallest to the slowest and largest:

1. **Registers:** Under 1 KB in size, extremely fast, managed by the compiler.
2. **Cache:** Managed by the hardware.
3. **Main Memory:** Slower, managed by the operating system.
4. **Disk Storage:** Hundreds of Gigabytes, very slow, managed by the operating system.

Caching is the principle of temporarily copying information from a slower storage system into a faster storage system so the CPU can access it quicker next time.

- When the CPU needs data, it checks the faster cache first.
- If the data is there, it uses it directly. If it is not there, the data is copied into the cache and then used.
- Because a cache is always much smaller than the storage it is caching, the system must carefully manage the **cache size and replacement policy** (deciding which old data to delete to make room for new data).

Data Migration and Coherency

When the computer needs a piece of data (let's call it Integer A), that data must migrate step-by-step up the entire hierarchy: **Magnetic Disk -> Main Memory -> Cache -> Hardware Register**.

This creates a major challenge for the computer:

- **In Multitasking:** The system must be incredibly careful to always use the most recent value of Integer A, regardless of where it currently sits in that hierarchy.
- **In Multiprocessing:** Because multiple CPUs each have their own local caches, the hardware must provide **cache coherency** to ensure that if one CPU changes Integer A, all other CPUs instantly see the most recent value in their own caches.
- **In Distributed Environments:** This is even more complex, as several copies of the same data can exist across entirely different computers on a network.

Part 4: Computer-System Architecture

Most systems use a single general-purpose processor, but multiprocessor systems (also called parallel or tightly-coupled systems) are becoming much more common. They offer three major advantages: increased throughput (getting more work done in less time), economy of scale (costing less than buying multiple separate single-processor computers), and increased reliability (providing fault tolerance or "graceful degradation" if one processor fails).

There are two main types of multiprocessing:

- **Asymmetric Multiprocessing (AMP):** Processors are not treated equally. A master processor assigns specific tasks to the others. For example, the system might only allow one CPU to execute OS code or perform I/O operations. This results in moderate throughput.
- **Symmetric Multiprocessing (SMP):** This is the most common modern architecture. Two or more identical processors share the same main memory and have equal access to all I/O devices. The single OS treats them all equally, resulting in high throughput.

Other physical setups include:

- **Dual-Core Design:** Two complete execution units (cores), along with their caches, are combined onto a single integrated circuit (silicon chip).
- **Clustered Systems:** Multiple entirely separate computers (nodes) networked together, typically sharing storage over a Storage-Area Network (SAN). They provide high availability that survives hardware failures. This can be *Asymmetric* (one machine sits in "hot-standby" mode waiting to take over if the main one fails) or *Symmetric* (multiple nodes run applications while monitoring each other). Some clusters are built for High-Performance Computing (HPC), which requires applications to be written using parallelization.

Part 5: OS Operations and Process Management

The operating system is completely interrupt-driven. To protect itself and other system components from software errors (like dividing by zero) or malicious programs, it relies on strict hardware controls.

- **Dual-Mode Operation:** The hardware provides a "mode bit" to distinguish between running normal user code (User mode, bit 1) and OS code (Kernel mode, bit 0). Certain privileged instructions can only be executed in Kernel mode. When a system call is made, it changes the mode to Kernel, and returning from the call resets it back to User. Dual-mode operation protects the system against buggy processes that might try to modify each other or overwrite the OS itself.

- **Timers:** To prevent a program from getting stuck in an infinite loop and hogging the system's resources, the OS uses a timer. It sets an interrupt to trigger after a specific period and decrements a counter. When the counter reaches zero, it interrupts the program so the OS can regain control or terminate it.

Process Management: A program (like a file sitting on your disk) is completely passive, while a **process** is an active program currently executing. A process needs resources like CPU time, memory, files, and initialization data to do its work. When it finishes, the OS must reclaim any reusable resources.

- A single-threaded process has one program counter that executes instructions sequentially, one at a time.
- A multi-threaded process has one program counter per thread.
- The system creates concurrency by multiplexing (rapidly sharing) the CPUs among all these active processes and threads.

To keep all of this organized, the OS is fully responsible for 5 specific process management activities:

1. Creating and deleting both user and system processes.
2. Suspending and resuming processes.
3. Providing mechanisms for process synchronization.
4. Providing mechanisms for process communication.
5. Providing mechanisms for deadlock handling.

The OS hides the hardware's specific characteristics from the user. It specifically handles **buffering** (storing data temporarily during transfer), **caching** (storing parts of data in faster storage), and **spooling** (overlapping the output of one job with the input of another).

Part 6: Security and Distributed Environments

1. Protection and Security While they sound similar, these are two distinct responsibilities for the operating system:

- **Protection:** This is the internal mechanism the OS uses to control how processes and users access specific system resources.
- **Security:** This is the broader defense of the system against both internal and external attacks, such as viruses, denial-of-service attacks, and identity theft.

To enforce these rules, the OS needs to know exactly who is using the computer:

- **User IDs (Security IDs):** Every user is assigned a unique name and number. This ID is attached to every single file and process that the user creates, allowing the OS to strictly control their access.
- **Group IDs:** This allows the OS to define a set of users and apply the exact same access controls to all of them at once.
- **Privilege Escalation:** Sometimes a user needs extra permissions to complete a task. This mechanism allows a user to temporarily change to an effective ID with more rights.

2. Distributed Computing Instead of just one machine, a distributed system is a collection of entirely separate, potentially very different (heterogeneous) computer systems linked together via a network.

- **The Network Operating System:** This is the software that runs across these linked computers. It provides a communication scheme so they can exchange messages, creating the illusion for the user that they are simply using one single system.
- **Network Sizes:** Networks scale up logically: a **LAN** (Local Area Network) covers a room or building, a **MAN** (Metropolitan Area Network) links buildings within a city, and a **WAN** (Wide Area Network) stretches across cities or countries. Therefore, LAN < MAN < WAN.

3. Computing Environments The way systems interact across these networks generally falls into a few models:

- **Client-Server:** Smart PCs act as "clients" that send requests to central "servers". For example, a compute-server processes tasks like database queries, while a file-server strictly stores and retrieves files for the clients.
- **Peer-to-Peer (P2P):** This model completely removes the difference between clients and servers. All connected computers are considered "peers" and can act as a client, a server, or both. To join a P2P network, a node must either register with a central lookup service or broadcast its presence using a discovery protocol.
- **Web-Based:** Because web access is now ubiquitous, specialized devices called "load balancers" are used to manage heavy web traffic across similar servers.

4. Special-Purpose Systems & Open Source Finally, the slides note that not all operating systems run on standard PCs:

- **Real-time embedded systems:** These are actually the most prevalent form of computers, using limited-purpose operating systems.
- **Handheld systems:** Devices like smartphones have limited CPU, memory, and power, so they require a reduced-feature operating system.
- **Open-Source Operating Systems:** Unlike closed-source systems that only ship the binary code, open-source systems (like GNU/Linux) make their raw source code available. Started by the Free Software Foundation, this approach directly counters the Digital Rights Management (DRM) movement.

Missing information:

- **Part 3 (Storage Performance Specs):** Your notes omit the technical specifications of the storage hierarchy. For example, it notes that cache is built with CMOS SRAM (0.5 – 25 ns access time) and main memory uses CMOS DRAM (80–250 ns access time).
- **Part 4 (Von Neumann Architecture):** The slides specify that the basic interaction of instruction execution and data movement between the CPU and memory is known as a "von Neumann architecture".
- **Part 5 (Multiprogramming & Timesharing):** Your notes do not include that timesharing requires a response time of under 1 second. They also miss the definitions for **swapping** (moving processes in and out of memory to run) and **virtual memory** (allowing execution of processes not completely loaded into memory).
- **Part 5 (Memory & Storage Management):** You are missing the OS responsibilities for memory (tracking usage, deciding what to move, allocating space), file systems (organizing directories and mapping files), and mass-storage (free-space management and disk scheduling).
- **Part 6 (Multimedia & OS Evolution):** Your notes omit **Multimedia systems**, which require data streams to be delivered with strict time restrictions. They also leave out P2P examples like Napster and Gnutella, and the fact that older client-side systems like Windows 95 evolved into versatile client/server systems like Linux and Windows XP.