1. OS

OS (Operating system) :

- IT work as intermediate interface between User and computer Hardware.
- It is a resource manager. (i.e CPU, Memory)
  If any app need any recourse it can't interact with Hardware , App say to OS then OS provide the resource to that App.

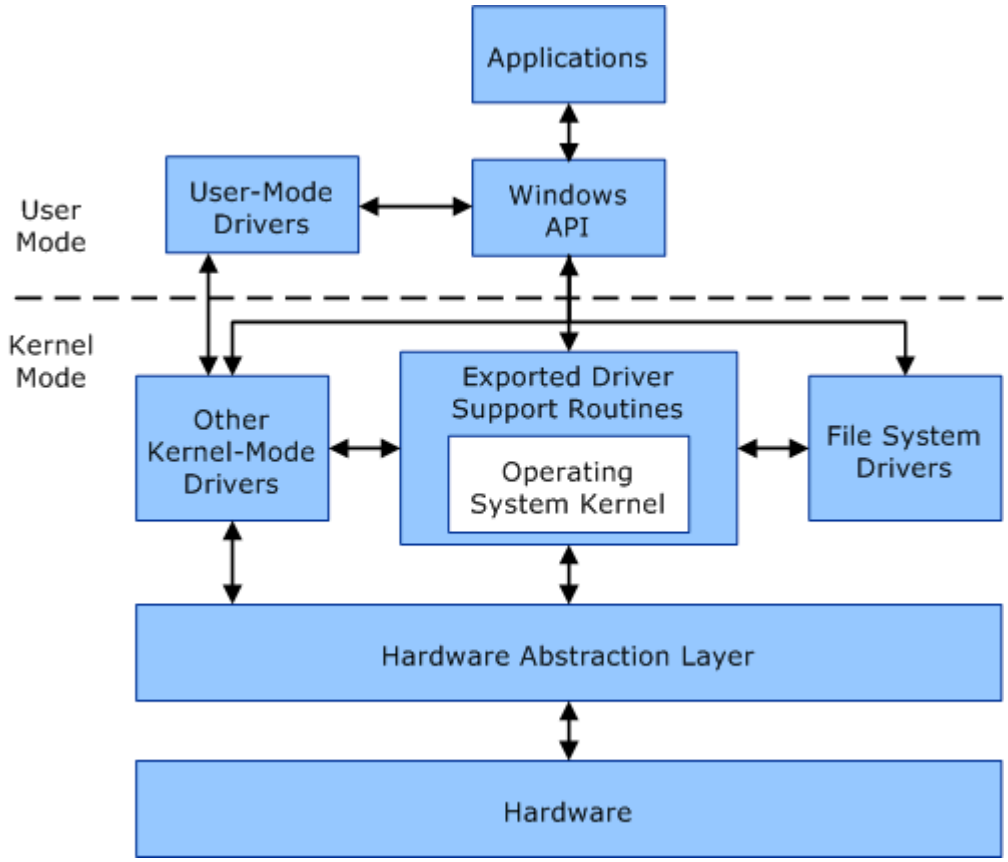
Why OS is needed?

- **Manages Resources**: Allocates CPU, memory, and storage for efficient use.
- **Provides User Interface**: Enables easy interaction with the computer.
- **Handles File Management**: Organizes and manages files on storage devices.
- **Manages Processes**: Facilitates multitasking and program execution.
- **Ensures Security**: Protects data and controls user access.
- **Manages Devices**: Coordinates communication with input/output devices.
- **Supports Networking**: Enables connectivity and communication between computers.

> Function of OS ? Points just above in Why OS is needed they are function as well of OS.

Kernel

It is a computer program that is core of OS, one should also know it is also the first program that loads after the bootloader. It then does all the talking between the hardware and the software or applications. So if you launch a program, the user interface sends a request to Kernel. The Kernel then sends a request to the CPU, Memory to assign processing power, memory, and other things so the application can run smoothly in the front end.



Kernel Security & Protection

The kernel also protects the hardware. If there is no protection, any program will be able to carry out any task on the computer, including crashing your computer, corrupting data, etc.

Secure Boot is a security standard designed to protect systems from malicious software by ensuring that only trusted applications run during startup. It verifies the signatures of boot software, including firmware and the operating system, before allowing the system to boot.

Trusted Boot enhances this process by using a Virtual Trusted Platform Module (VTPM) to verify the digital signature of the Windows 10 kernel and all other startup components. If any file is altered, the bootloader detects the change and prevents the system from loading it, establishing a chain of trust for the entire boot process.

types of Kernel

1. **Monolithic Kernel:** Here, the OS and Kernel both run in the same memory space and are suitable where security is not a significant concern. It results in faster access, but if there is a bug in the device driver, the entire system crashes.

> Linux work on Monolithic Kernel.
> Each app depend on Kernel for its all dependency.
> It is for heavy task as it use X86 or X64.

2. **Microkernel:** It's a stripped-down version of Monolithic Kernel where the Kernel itself can do most of the job, and there is no need of an extra GUI. They should be used where security and the crashing system isn't or will not happen.

> Android work on Micro Kernel.
> Each app have its dependency itself so it don't need to depend on kernel. as in result give security and no crash if any bug.
> Is is for lightweight task as it based on ARM processor used in mobiles.

3. **Hybrid Kernel:** This Kernel is what we see **most. Windows, Apple**'s macOS. They are a mix of Monolithic Kernel and Microkernel.
    - It moves out drivers but keeps system services inside the Kernel – similar to how drivers are loaded when Windows Starts the bootup process.
4. **Nano Kernel:** If you need to have a kernel, but its majority of function is set up outside, then this comes into the picture.
5. **Exo Kernel:** This kernel only offers process protection and resource handling. However, it is mostly used when testing an in-house project and you upgrade to a better Kernel type.

System call

A system call is a way for a program to request a service from the operating system's kernel.

- When a program makes a system call, it typically switches from user mode to kernel mode, where it has higher privileges to access hardware and system resources.
- This mechanism ensures that **user programs cannot directly** access critical system components, providing security and stability.

Common examples of system calls include:

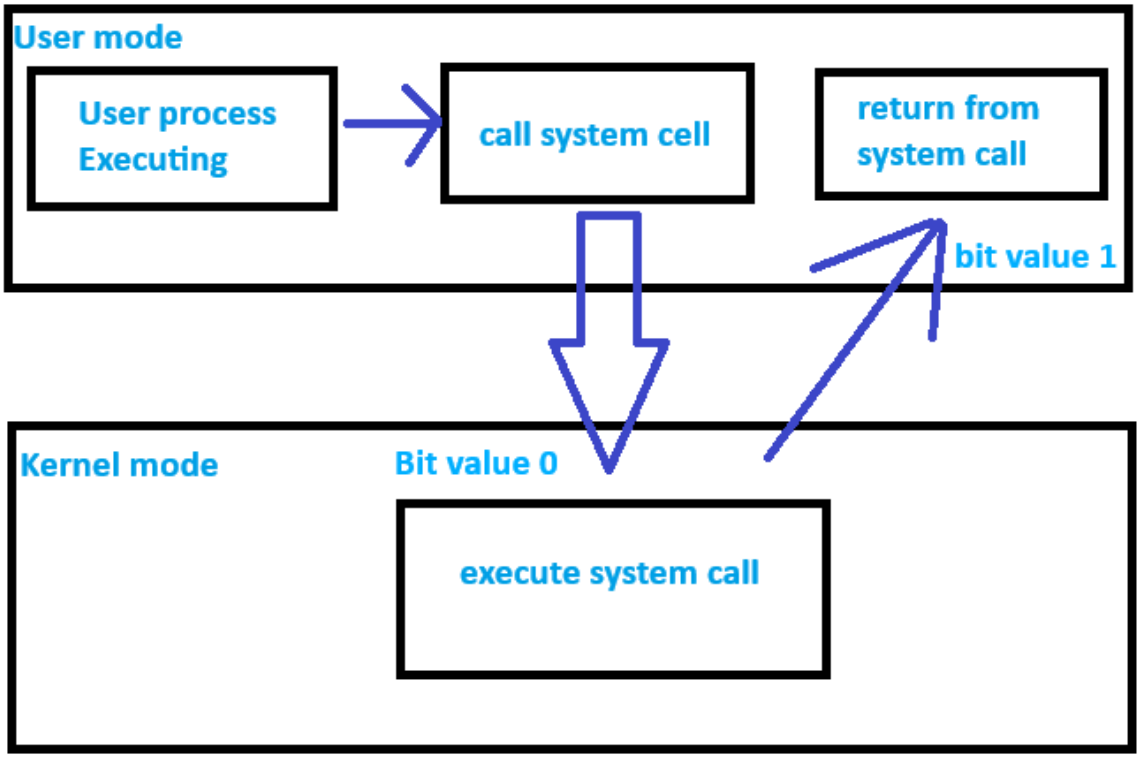
- **File operations**: `open`, `read`, `write`, `close`
- **Process management**: `fork`, `exec`, `wait`
- **Memory management**: `mmap`, `brk`
- **Networking**: `socket`, `connect`, `bind`

User mode / Kernel mode

- **User mode :** when a Application do a **system call directly** to access the resource is called User mode. When applications are running in user mode, this bit is **set to 1.**
- **Kernel mode :** When a Application do a system call thought kernel to access the resource is called Kernel mode. When the operating system is running in kernel mode, this **bit is set to 0.**

Ques : so how Hardware know who is doing system call User or kernel mode?
When a User mode do a system call its Mode bit is 1.
when a kernel do a system call its Mode bit is 0. so by monitoring mode bit hardware get to know who is doing system call.

> When user process execute it generate a system call user bit 1 convert first to kernel bit 0 then it execute the system call in kernel mode as the kernel execute the system call and done it send that system call to software by changing kernel bit 0 to 1.
>
> - when kernel execute the system if their is any resource present then it give to process or denied.



How does Kernel Mode work in Windows 11/10?

reference

All code that runs in kernel mode shares a single virtual address space. This means that a kernel-mode driver is not isolated from other drivers and the operating system itself. If a kernel-mode driver accidentally writes to the wrong virtual address, data that belongs to the operating system or another driver could be compromised. If a kernel-mode driver crashes, the entire operating system crashes.

2 How does User mode work in Windows 11/10?

When you start a user-mode application, Windows creates a process for the application. The process provides the application with a private virtual address space and a private handle table. Because an application's virtual address space is private, one application cannot alter data that belongs to another application. Each application runs in isolation, and if an application crashes, the crash is limited to that one application. Other applications and the operating system are not affected by the crash.

How does Windows 11 boot

Phase	Boot Process	BIOS	UEFI
1	PreBoot	MBR/PBR (Bootstrap Code)	UEFI Firmware
2	Windows Boot Manager	%SystemDrive%\bootmgr	\EFI\Microsoft\Boot\bootn
3	Windows OS Loader	%SystemRoot%\system32\winload.exe	%SystemRoot%\system32
4	Windows NT OS Kernel	%SystemRoot%\system32\ntoskrnl.exe	%SystemRoot%\system32

1] PreBoot: POST or Power-On Self-Test loads firmware settings. It checks for a valid disk system, and if the system is good to go for the next phase. If the computer has a valid MBR, i.e., Master Boot Record, the boot process moves further and loads Windows Boot Manager.

2] Windows Boot Manager: This step determines if you have multiple OS installed on your computer. If yes, then it offers a menu with the names of the OSs. When you select the OS, it will load the right program, i.e., Winload.exe to boot you into the correct OS.

3] Windows OS Loader: Like its name, *WinLoad.exe* loads important drivers to kick start the Windows Kernel. The kernel uses the drivers to talk to the hardware and do rest of the things required for the boot process to continue.

4] Windows NT OS Kernel: This is the last stage that picks up the Registry settings, additional drivers, etc. Once that has been read, the control is taken by the system manager process. It loads up the UI, the rest of the hardware and software. That's when you finally get to see your Windows 10 Login screen.

Secure boot

Secure Boot is a security feature implemented in the UEFI (Unified Extensible Firmware Interface) firmware that helps ensure that only trusted software can be executed during the boot process of a computer. Here's how it works and its key components:

How Secure Boot Works

1. **Firmware Verification:**

- When the computer powers on, the UEFI firmware performs a check to verify the integrity and authenticity of the firmware and bootloader. It uses digital signatures to ensure that only signed and trusted components can be loaded.

2. **Chain of Trust:**
    - Secure Boot establishes a "chain of trust." This means that each component in the boot process (firmware, bootloader, operating system kernel, and other critical software) must be signed with a valid digital signature. If any component is not signed or fails verification, the boot process may be halted or prevented from loading that component.
3. **Key Management:**
    - UEFI firmware includes a database of trusted keys (public keys) and certificates. It uses these to verify the signatures of the software components. If a new component needs to be added (like a driver), it must also be signed with a trusted key.
4. **Operating System Support:**
    - Secure Boot must be supported by the operating system. Major operating systems like Windows, Linux distributions, and others typically support Secure Boot. However, users can often configure Secure Boot settings or disable it if necessary.

computer system architecture

single processor system :

A **single processor system** is a computer architecture where a single central processing unit (CPU) executes instructions and manages all tasks.

Key Features:

- **Architecture:** Typically follows the Von Neumann or Harvard architecture.
- **Performance:** Limited to the processing power of one CPU. Performance can be impacted by the number of tasks and their complexity.
- **Memory Access:** The CPU accesses a single memory space for both instructions and data, which can lead to bottlenecks, especially in multitasking scenarios.
- **Cost-Effective:** Generally less expensive than multi-processor systems, making it suitable for personal computers and small servers.
- **Simplicity:** Easier to design and manage due to the absence of complex inter-processor communication.

2. Multi-Processor System :

A **multi-processor system** contains two or more CPUs that can work on different tasks simultaneously, improving overall performance and throughput.

Key Features:

- **Parallel Processing:** Multiple processors can execute multiple instructions simultaneously, enhancing performance for multi-threaded applications.
- **Scalability:** Easily scalable by adding more processors, which can increase computing power without significant architectural changes.
- **Improved Performance:** Better handling of resource-intensive applications and improved multitasking capabilities.
- **Complexity:** More complex design and management due to the need for synchronization and communication between processors.

3. Cluster System:

A **cluster system** consists of multiple independent computers (often referred to as nodes) that work together as a single system to provide higher performance, reliability, and availability. Clustering enables resource sharing, load balancing, and fault tolerance.

Key Features:

1. **Nodes:**
    - Each node is typically a standalone computer, which can be a server or a workstation, connected through a local area network (LAN).
2. **Interconnectivity:**
    - Nodes communicate over high-speed networks, which may include Ethernet or specialized cluster interconnects (like InfiniBand) for reduced latency and increased bandwidth.
3. **Resource Sharing:**
    - Nodes share resources such as processing power, memory, and storage, which allows for better performance than single-node systems.
4. **Scalability:**
    - Clusters can be easily scaled by adding more nodes, making it flexible for growing workloads.