Operating System

BCSE303L

Module 1

Dr. Naveenkumar Jayakumar

Associate Professor

Department of Computational Intelligence

PRP 217-4

naveenkumar.jk@vit.ac.in

Operating System - Definition

- An Operating System (OS) is system software that acts as an **intermediary between computer hardware and the user**.
- It manages hardware resources.
- It provides an environment in which application programs can run efficiently.

The OS makes the computer usable by controlling the hardware and providing services for programs.

Operating System - Definition

Aspect 1- Resource Management

The OS allocates and controls the use of hardware components like memory, storage, and processors.

Aspect 2 - Providing Services

The OS offers common functionalities like file management, security, and a user interface (UI) for applications to run smoothly.

Operating System - Definition

It is the first program loaded by a computer at startup (boot loader loads the OS kernel).

It provides an interface between users and computer hardware.

It acts as a control program that prevents misuse of hardware resources.

The OS is a resource allocator, deciding which program gets which resource and for how long.

Operating System - Goals

Convenience

- Make interaction with the computer user-friendly.
- Provide GUIs, virtual desktops, file explorers.

Efficiency

- Optimize system performance — maximize CPU, disk, and I/O utilization.
- Minimize response time, turnaround time, waiting time.

Ability to Evolve

- Modular design so parts can be upgraded.
- Support new hardware easily.

Operating System - Goals

Fairness	• Ensure fair resource allocation among multiple users/programs.
Robustness & Reliability	• Keep the system stable even under unexpected failures.
Scalability	• Handle growth in number of users, tasks, or resources.

Operating System - Functionality

Process Management	Memory Management	File System Management
Device Management	**Security and Protection**	Networking
Command Interpretation	**Error Detection and Recovery**	User Interface

Operating System - Functionality

Process Management

- Create, schedule, suspend, and terminate processes.
- Context switching between processes.
- Synchronization and communication between processes (e.g., semaphores, pipes).

Memory Management

- Keeps track of each byte of memory.
- Allocates memory to processes when they need it and deallocates when done.
- Implements techniques like paging, segmentation, and virtual memory.
- Handles swapping between main memory and disk.

Operating System - Functionality

File System Management

- Creates, deletes, reads, writes files/directories.
- Manages permissions and access control lists (ACLs).
- Maintains file metadata (size, timestamps, permissions).

Device Management

- Manages input/output devices using device drivers.
- Handles buffering, spooling, and caching of data.
- Provides uniform interface to devices (device independence).

Operating System - Functionality

Security and Protection

- User authentication (login, passwords, biometrics).
- Authorization and access control.
- Protects user data and system resources from malware and unauthorized access.

Networking

- Manages network connections (TCP/IP stack).
- Supports distributed computing and network file systems.
- Provides protocols for secure data exchange.

Operating System - Functionality

Command Interpretation

- Shell interprets user commands and executes them.
- Provides scripting capabilities for automation.

Error Detection and Recovery

- Monitors hardware for failures.
- Reports errors and takes corrective actions.
- Logs system events for diagnostics.

Operating System – What it Does?

Process Management	Memory Management	File System Management	Device Management
• Process Scheduling • Process Creation and Termination • Concurrency and Synchronization	• Allocation and Deallocation • Paging and Segmentation • Virtual Memory	• File Organization • Access Control • File Operations	• Device Drivers • I/O Operations

Operating System – What it Does?

Security and Access Control

- Authentication
- Authorization
- Encryption

User Interface

- Graphical user interface (GUI)
- Command Line Interface (CLI)

Networking

- Network Communication
- Resource Sharing

Operating System – What it Does?

System Performance and Monitoring

- Performance Optimization
- Monitoring Tools

Error Handling and Recovery

- Error Detection
- Recovery Mechanisms

Multitasking and Multithreading

- Multitasking
- Multithreading

Operating System – Design Issues

An operating system's design is a complex process that involves addressing a multitude of challenges and making critical trade-offs to ensure a computer system is both usable and efficient.

The OS must decide which of the many running processes gets to use the CPU and for how long

The operating system is responsible for allocating and deallocating memory space to processes. It must keep track of which parts of memory are currently being used and by whom, and it needs to ensure that processes do not interfere with each other's memory.

The OS manages communication with hardware devices such as disk drives, printers, and network adapters. It provides a consistent interface for applications to interact with these devices, hiding the complexities of the underlying hardware.

This involves using device drivers to translate high-level requests into low-level device-specific commands.

When multiple processes need to access shared resources, the OS must ensure that their operations are coordinated to avoid conflicts and maintain data consistency.

Operating System – Design Issues

An operating system's design is a complex process that involves addressing a multitude of challenges and making critical trade-offs to ensure a computer system is both usable and efficient.

A deadlock is a situation where two or more processes are blocked forever, each waiting for a resource held by another. The OS must have strategies to prevent, detect, and recover from deadlocks.

The OS must define how files are stored on secondary storage devices, including their structure and naming conventions

The OS needs to enforce permissions to control which users and processes can access which files and what operations they can perform (read, write, execute).

The file system must ensure that data is not corrupted due to system crashes or hardware failures.

Operating System – Design Issues

An operating system's design is a complex process that involves addressing a multitude of challenges and making critical trade-offs to ensure a computer system is both usable and efficient.

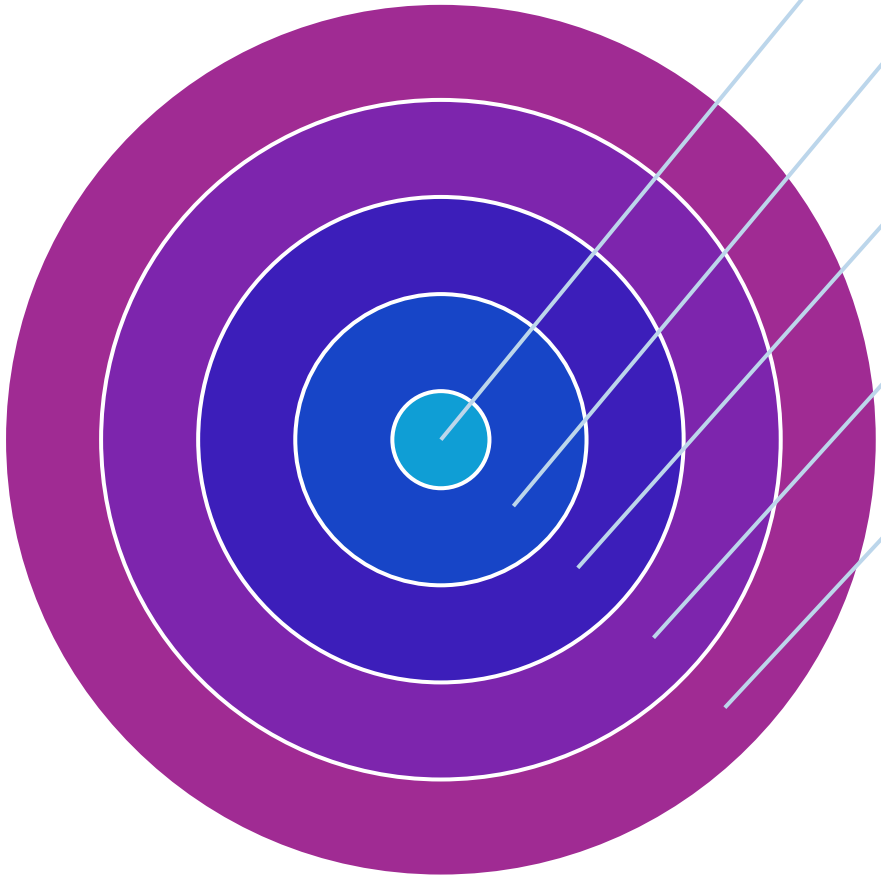
Verifying the identity of users before granting them access to the system.

Restricting the access of processes and users to system resources based on a defined policy.

Implementing mechanisms to defend against viruses, malware, and other security threats.

An operating system should be designed to be efficient and provide good performance. This involves minimizing the overhead of the OS itself and making optimal use of the available hardware resources.

Operating System – Abstract View



Operating System – Abstract View

Hardware Layer:

- CPU
- Memory (RAM)
- Storage (HDD/SSD)
- I/O Devices

Kernel Layer:

- Process Management
- Memory Management
- Device Drivers
- System Calls

System Libraries and Utilities Layer:

- Standard Libraries (e.g., C Standard Library)
- Utility Programs (e.g., file managers, system monitors)

User Applications Layer:

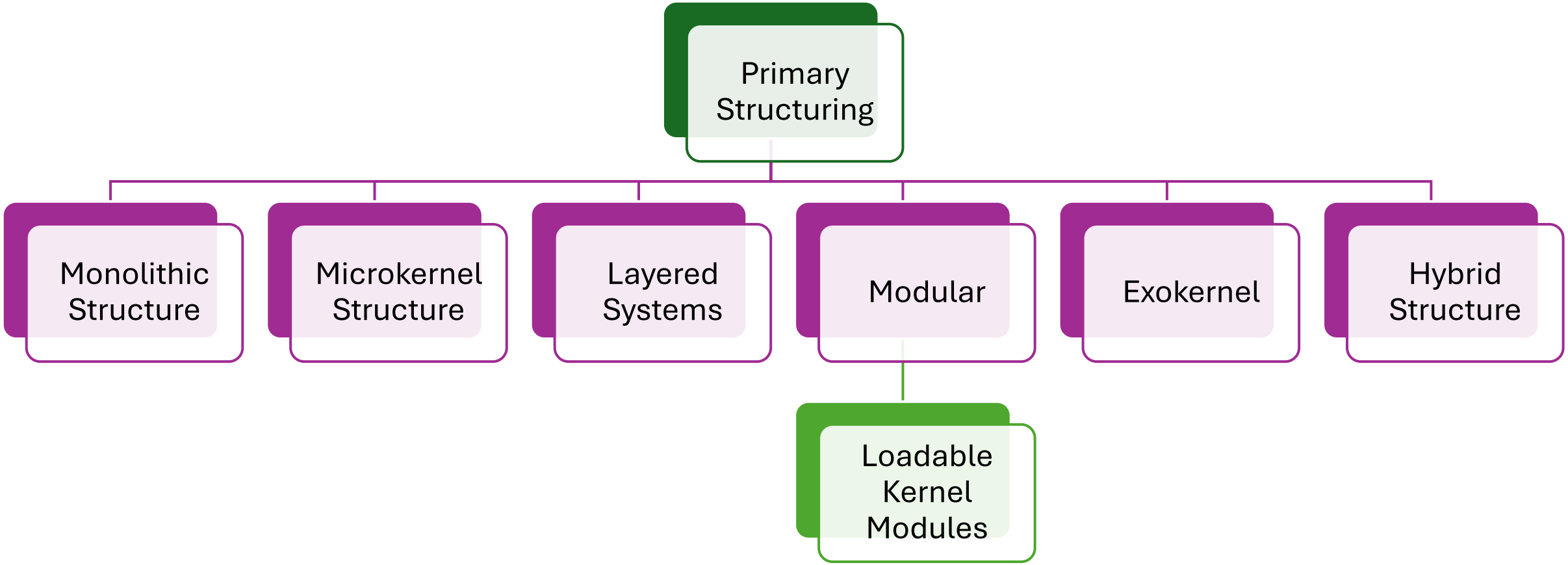
- Application Software (e.g., browsers, office suites)

User Interface Layer:

- Graphical User Interface (GUI)
- Command-Line Interface (CLI)

Operating System – Structuring

- Objective of Structuring a Operating system is to balance performance, maintainability, security, and flexibility



Operating System – Structuring

- The structuring of an Operating System (OS) is strongly related to **how it divides its components and services** between **Kernel Space and User Space**.

Kernel Space

- The privileged area of memory where the OS kernel runs.
- Has full access to hardware resources.
- Runs in privileged mode (Ring 0 in x86 architecture).
- Handles critical tasks like CPU scheduling, interrupt handling, device drivers, memory management.

User Space

- The area of memory where user applications and some system services run.
- Runs in unprivileged mode (Ring 3 in x86).
- Has restricted access — must use system calls to request services from the kernel.
- Examples: your web browser, word processor, or user-level services like print spoolers (in some architectures).

Operating System – Monolithic Structure

Monolithic Kernel

- All OS services (process management, memory management, device drivers, file systems) run in a single, unified kernel space with full hardware access.
- No separation between components; tightly integrated.
- Direct hardware access for speed.
- Examples: Traditional Unix systems (e.g., BSD, Linux in its core design).

Advantages

- High Performance : Minimal overhead due to direct communication between components.
- Simple design for developers.

Disadvantages :

- Complexity : Large codebase makes debugging and updates challenging.
- Instability : A bug in any component can crash the entire system.
- Security risks due to lack of isolation.

Operating System – Microkernel Structure

Microkernel

- Only essential services (IPC, scheduling, memory management) run in kernel space. Non-essential services (device drivers, file systems) operate in user space .

Key Characteristics

- Minimalist core with modular user-space services.
- Examples: QNX, L4, Mach (basis for macOS/iOS).

Advantages

- Reliability : Failures in user-space services don't crash the kernel.
- Security : Isolation reduces attack surfaces.
- Easier to update components.

Disadvantages

- Performance Overhead : Frequent context switches between user/kernel modes.
- Complex IPC mechanisms can slow down operations.

Operating System – Modular Structure

Modular Kernel

- A hybrid approach where the core is monolithic but supports loadable kernel modules (LKMs) for extending functionality at runtime.

Key Characteristics

- Core services (e.g., process scheduling) are static.
- Modules (e.g., drivers, filesystems) can be added/removed dynamically.
- Example: Linux kernel.

Advantages

- Flexibility : Add/remove features without rebooting.
- Balances performance (core services in kernel) and modularity.

Disadvantages

- Modules still run in kernel space; a faulty module can crash the system.
- Slightly higher complexity than pure monolithic kernels.

Operating System – Layered Structure

Layered Kernel

- OS components are organized into hierarchical layers , each providing abstracted services to the layer above.
- Example: Hardware → Kernel → Device Drivers → File Systems → User Apps.

Key Characteristics

- Strict abstraction (e.g., lower layers handle hardware, upper layers handle applications).
- Example: MINIX (educational OS by Andrew Tanenbaum).

Advantages

- Simplicity : Each layer has a well-defined role.
- Easier debugging and testing due to isolation.

Disadvantages

- Latency : Layered communication adds overhead.
- Reduced performance compared to monolithic designs.

Operating System – Structuring

Hybrid Kernel

- Combines a microkernel core with monolithic elements . Critical services run in kernel space for speed, while others run in user space.

Key Characteristics

- Core services (e.g., hardware abstraction, threading) in kernel space.
- Higher-level services (e.g., drivers) may run in user space or kernel space.
- Examples: Windows NT, macOS (XNU kernel), iOS.

Advantages

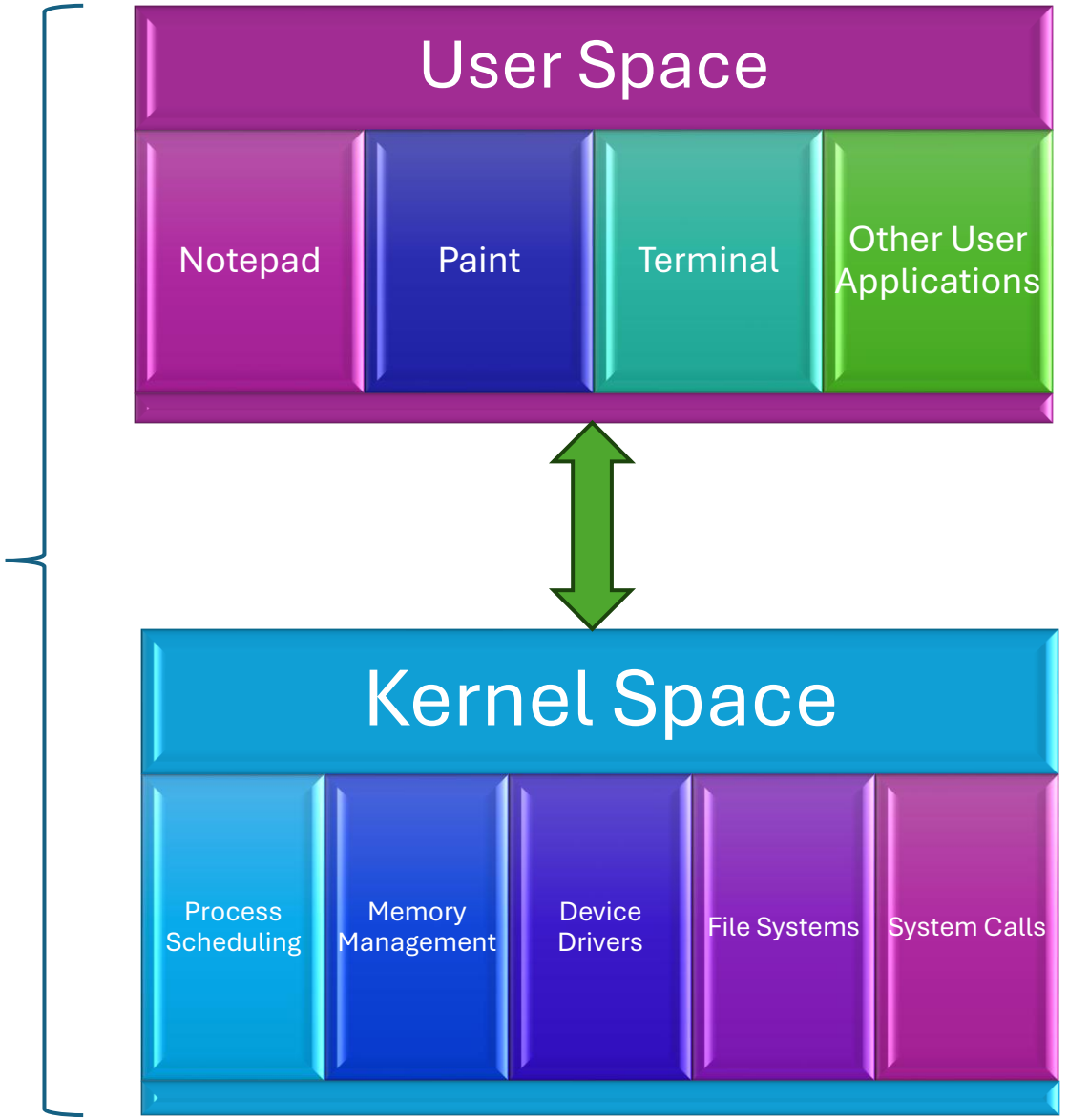
- Balanced Performance : Critical tasks are optimized; non-critical tasks are modular.
- Improved stability over pure monolithic kernels.

Disadvantages

- Complexity in design and maintenance.
- Less isolation than pure microkernels.

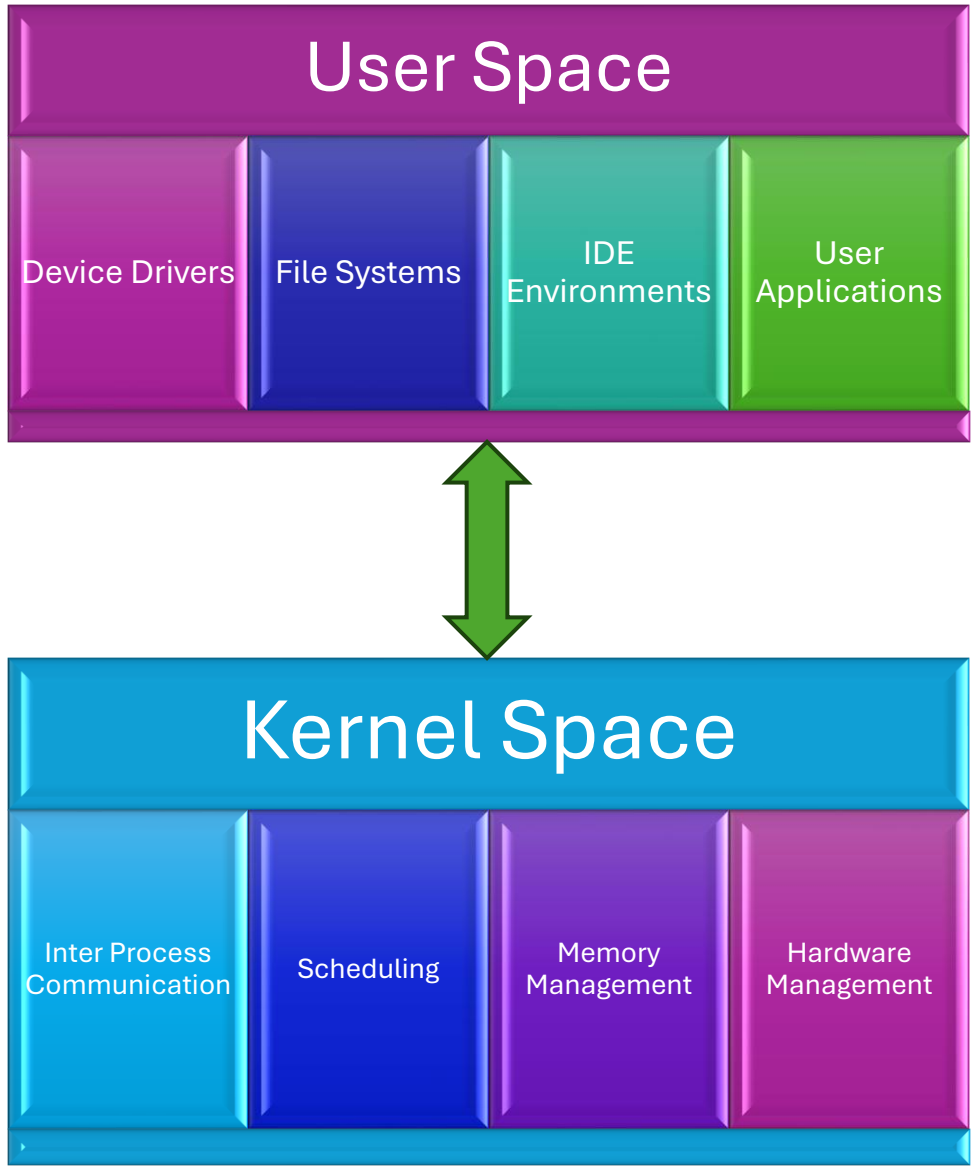
Structuring Kernel - ? Kernel

- All OS services run in the same address space (kernel space)
- Everything is tightly integrated.
- No separation between subsystems



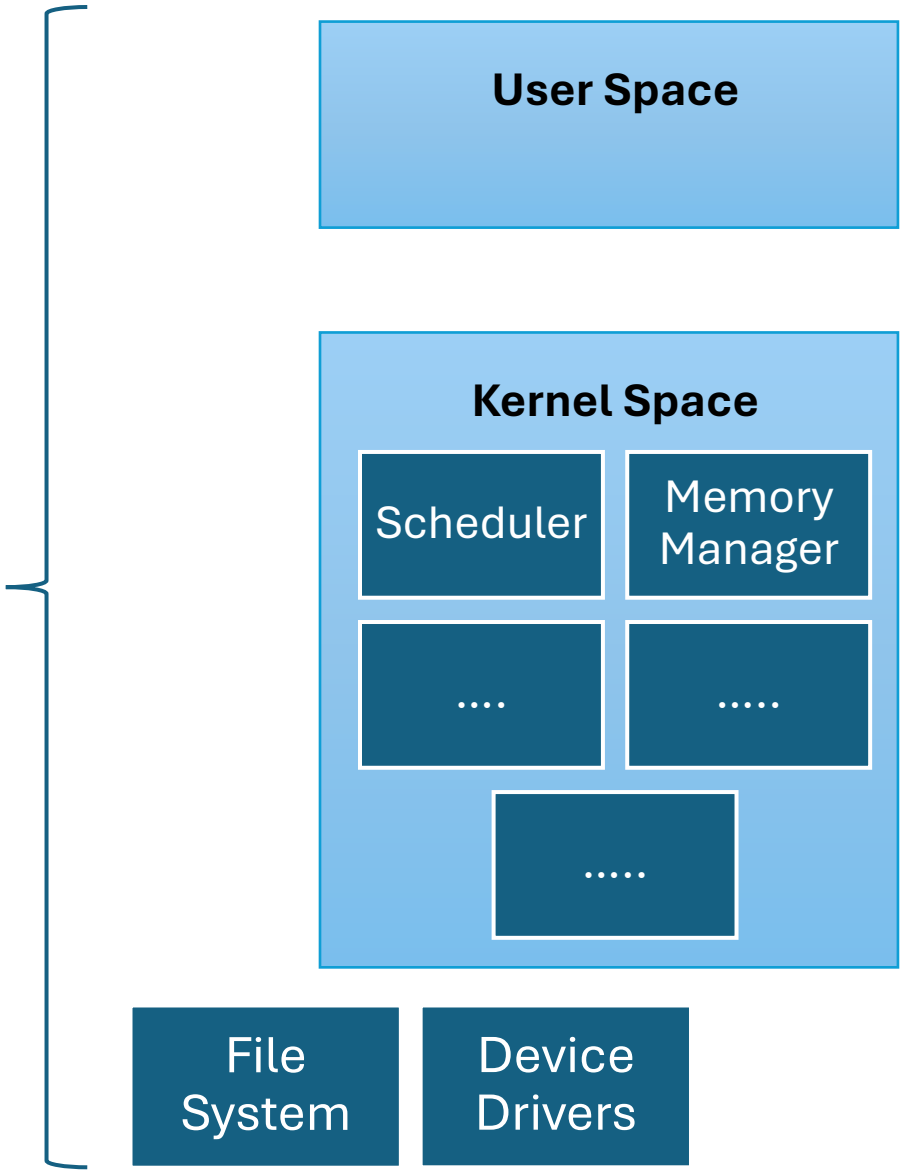
Structuring Kernel - ? Kernel

- Minimal kernel functionality in kernel space
- Most services (like file systems, device drivers) run in user space.
- Only essential services are in the kernel; other services are outside communicating via IPC (Inter-Process Communication).



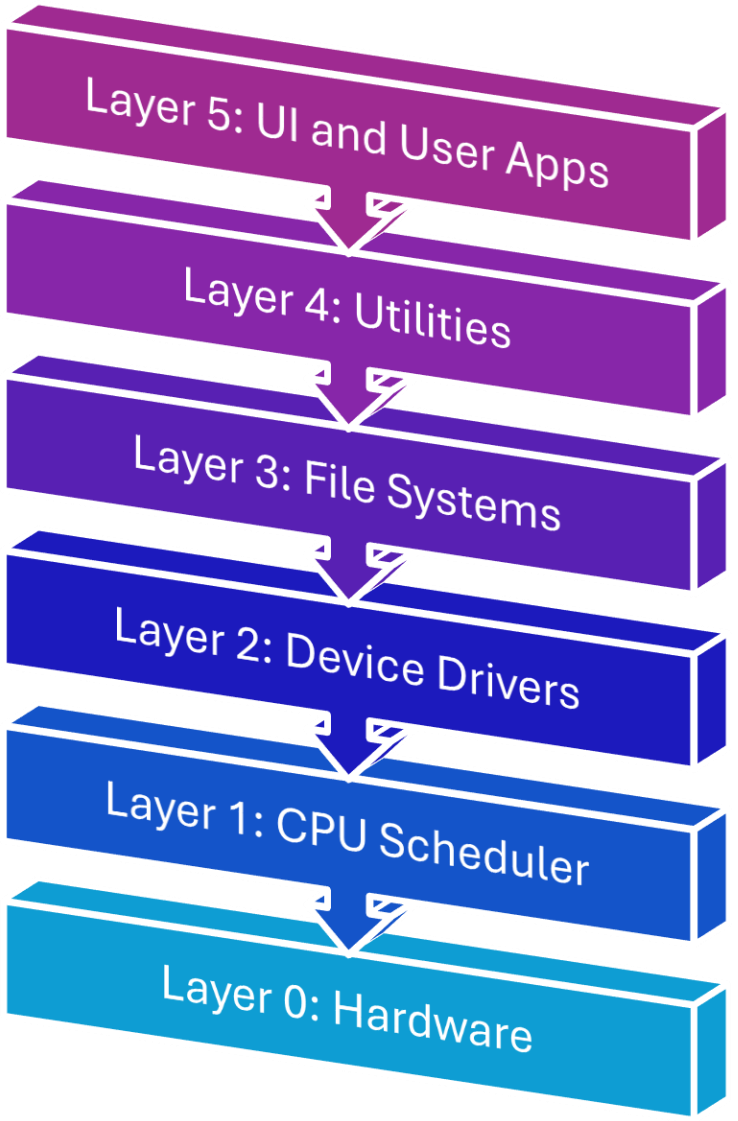
Structuring Kernel - ? Kernel

- Combines aspects of monolithic and microkernel.
- Core functions are in the kernel, but modules can be loaded/unloaded dynamically.



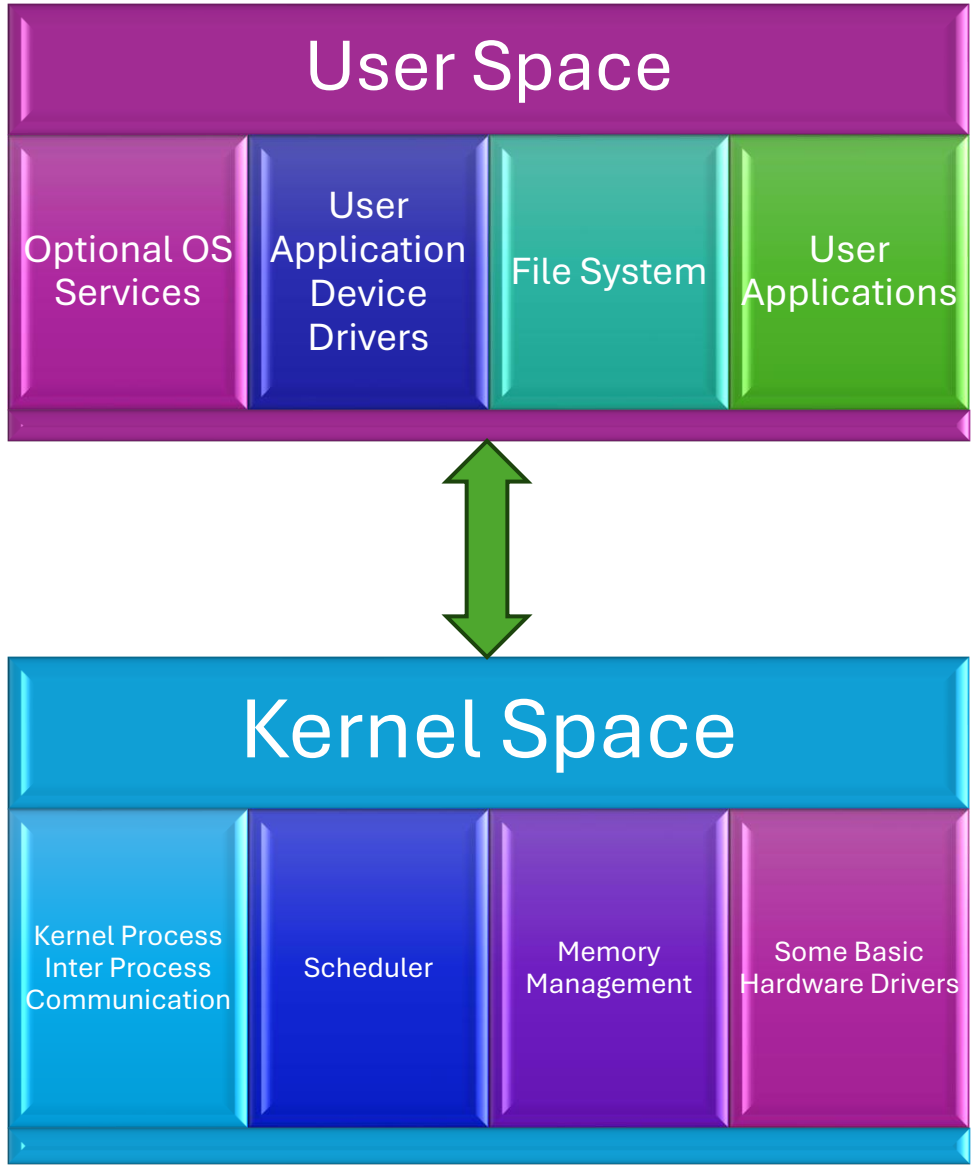
Structuring Kernel - ? Kernel

- The OS is divided into layers; each built on top of the lower layer.
- Top layers depend on lower ones.
- Each layer sits neatly above the previous, showing dependency from top to bottom.



Structuring Kernel - ? Kernel

- Mix of microkernel and monolithic concepts.
- Runs some services in user space, others in kernel space for performance.
- Looks similar to a microkernel, but with more services optionally included in the kernel for speed.



Operating System – Structuring

Structure Type	Key Trait	Performance	Stability	Flexibility	Example
Monolithic	All services in kernel space	High	Low	Low	Linux (core)
Microkernel	Minimal core; services in user space	Low	High	High	QNX
Modular	Loadable modules + monolithic core	Medium-High	Medium	High	Linux (modules)
Layered	Strict hierarchical layers	Low	Medium	Low	MINIX
Hybrid	Microkernel core + monolithic elements	Medium-High	Medium-High	Medium	Windows NT, macOS

Abstraction in Kernels

- **Abstraction in Monolithic Kernel - Minimal**
  - All components (e.g., device drivers, file systems) run in the same address space (kernel space).
  - Applications interact with hardware through system calls , but the kernel itself doesn't abstract its internal components from each other.
- **Abstraction in Microkernel - High**
  - Only core services (e.g., process scheduling, memory management) run in kernel space.
  - Non-essential services (e.g., device drivers, file systems) are abstracted into user-space servers that communicate via message passing .

Abstraction in Kernels

- **Abstraction in Modular Kernel – Moderate**
  - Core services (e.g., process management) are built into the kernel.
  - Optional components (e.g., drivers, file systems) are abstracted as loadable modules that can be added/removed dynamically.
- **Abstraction in Layered Kernel – High**
  - The OS is split into hierarchical layers (e.g., hardware → CPU scheduling → memory management → file systems → user interface).
  - Each layer abstracts the layer below it, providing a simplified interface for the layer above.

Abstraction in Kernels

- **Abstraction in Hybrid Kernel – Selective**
  - Combines microkernel and monolithic ideas. Critical services (e.g., thread scheduling) run in kernel space, while others (e.g., drivers) may run in user space or kernel space.
  - Abstraction is applied selectively to balance performance and isolation.
- **Benefits of Abstraction**
  - **Simplifies Complexity** : Applications and developers don't need to manage hardware directly (e.g., a program just asks to "read a file" without knowing how the disk works).
  - **Improves Security** : Isolating components (e.g., microkernel services) limits the damage from bugs or attacks.
  - **Enables Modularity** : Abstraction allows features to be added/removed without rewriting the entire OS (e.g., Linux modules).
  - **Balances Trade-offs** : Hybrid and layered kernels use abstraction to strike a balance between speed, stability, and flexibility.

Abstraction in Kernels

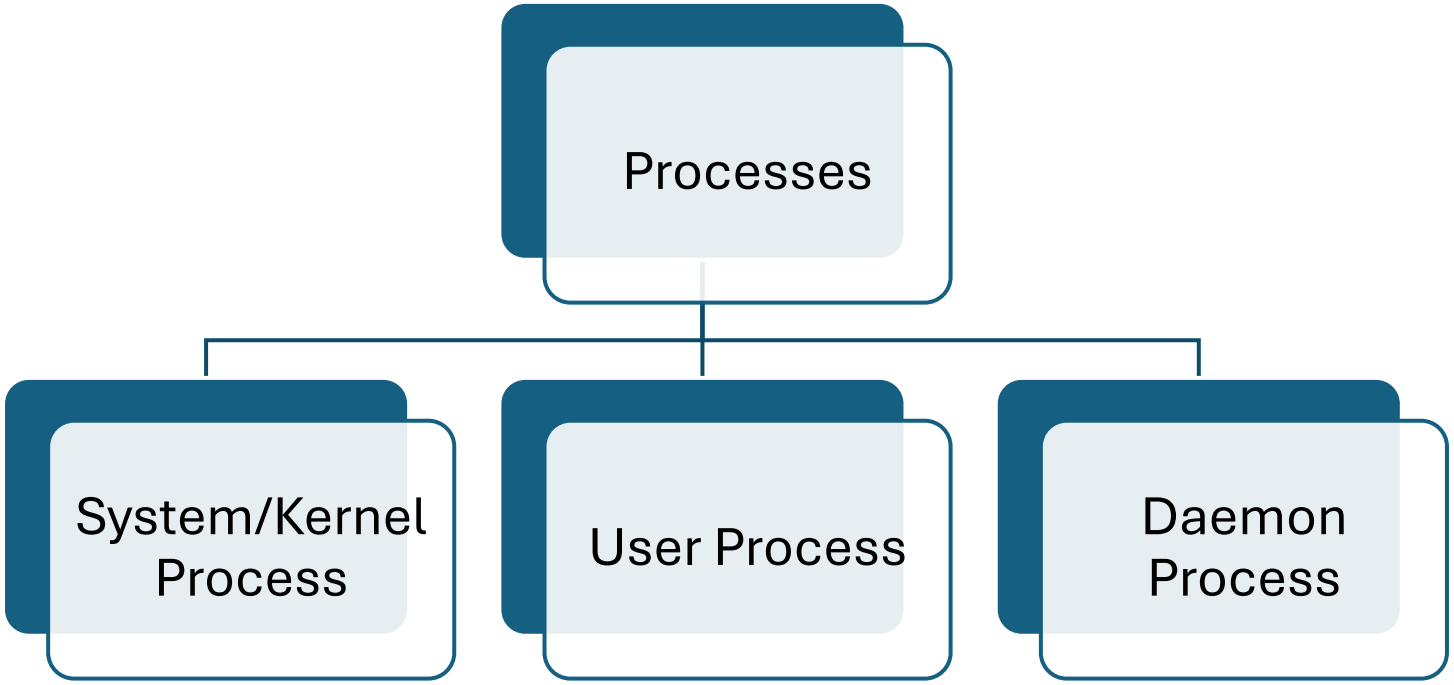
- Its primary role is to **hide the complexity of hardware** from the user and application programs.
- Instead of dealing with the intricate details of disk drives, memory chips, and CPU registers, we work with simpler, high-level representations.
- For example, you interact with files and folders, not with the physical sectors and tracks on a storage drive. The OS abstracts the hardware details into this easy-to-use file system model. This makes software development drastically simpler and allows programs to be portable across different hardware configuration

Process & Resources

❑Processes

- ❑A process is a program in execution.
- ❑ It's the basic unit of work in an OS.
- ❑ The OS's job is to manage numerous processes, switching between them rapidly to give the illusion of simultaneous execution (multitasking).
- ❑ This management ensures that each process gets a fair share of the CPU and runs in isolation, preventing it from interfering with others.

Process & Resources



Process & Resources

❑Resources

- ❑ These processes need resources to do their work.
- ❑Resources include:
  - ❑ **CPU time**: The processing power needed to execute instructions.
  - ❑ **Memory**: Space to store the program and its data.
  - ❑ **I/O devices**: Access to hardware like keyboards, displays, and network cards.
- ❑ The OS acts as a **resource manager**, allocating these resources efficiently and fairly among all the competing processes.

Process & Resources

Physical Resources

- CPU (Central Processing Unit)
- Memory (Main Memory or RAM)
- I/O Devices (Keyboard, Mouse, Printer, etc.)
- Storage Devices (Hard Disk, Solid State Drive, etc.)

Virtual Resources

- Virtual Memory (Swap Space or Page File)
- Virtual CPUs (in a virtualized environment)

Software Resources

- Files and Directories
- I/O Streams (Input/Output Channels)
- Sockets (Network Connections)
- Semaphores (Synchronization Mechanisms)

Abstract Resources

- Processes (Threads, Tasks, or Jobs)
- Threads (Lightweight Processes)
- Synchronization Objects (Mutexes, Locks, etc.)

Process & Resources

- The Operating System manages and performs following operations with resources :

Allocation	Deallocation	Scheduling	Protection	Sharing
• Assigning resources to processes or programs.	• Releasing resources when no longer needed.	• Allocating CPU time to processes.	• Ensuring processes access only authorized resources.	• Allowing multiple processes to access shared resources.

Influence of Security

- Initially, OS security was minimal. Today, it is a foundational requirement.
- With systems connected to networks and storing sensitive data, the OS must protect against
  - unauthorized access,
  - malware, and
  - other threats.
- It enforces security policies through mechanisms like
  - user accounts,
  - permissions, and
  - firewalls,
- ensuring data integrity and user privacy

Networking

- The rise of the internet and local networks transformed computers from isolated machines into communication devices.
- Consequently, networking became a core OS service.
- The OS manages network connections, protocols (like TCP/IP), and interfaces.
- It provides the foundation for everything from web Browse and email to distributed computing and cloud services.

Networking

- The demand for high-quality audio and video has significantly influenced OS design.
- Multimedia applications require the OS to handle high-throughput data streams and meet real-time deadlines.
- A slight delay in processing a video frame can ruin the user experience.
- This led to the development of better scheduling algorithms and improved I/O techniques to ensure smooth, continuous playback and processing of multimedia content.