Unit-1 Introduction to Operating System

Lecture – 4

Kernel

- The kernel is the core part of an operating system. It acts as a bridge between software applications and computer hardware.
- The kernel is the first part of the operating system to load into memory during booting (i.e., system startup), and it remains there for the entire duration of the computer session because its services are required continuously.
- It manages system resources, such as the CPU, memory, and devices, ensuring everything works together smoothly and efficiently.

Operating System Structures

Monolithic Structure

- In this approach, all the core functionalities of the operating system—such as process management, memory management, file systems, I/O management, and so on—are combined into a single large binary file. This entire binary runs in kernel mode.
- It is one of the most common techniques used for designing early operating systems.
- Whenever a user application needs to access any of these OS functionalities, it does so by using a system call.
- To better understand this approach, let's consider the structure of a traditional UNIX operating system, As shown in the following figure-1.

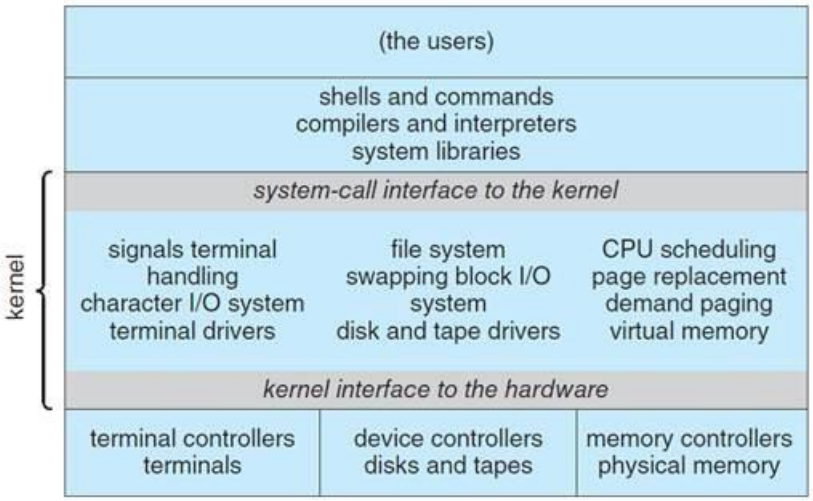

Figure-1 Traditional UNIX system structure.

- In this structure, the OS is broadly divided into two parts: System programs, and the kernel
- Everything below the system call interface and above the hardware interface is considered part of the kernel.
- The kernel is responsible for providing important OS services like:
    - File system management
    - CPU scheduling
    - Memory management
    - Device drivers
    - Other low-level operations
- All these services are accessible to user programs through system calls.
- In a monolithic structure, most OS services reside in the same address space (i.e., the kernel space) and can call each other directly.
- While the monolithic kernel looks simple conceptually, in reality, it is difficult to design, implement, and maintain, especially as the system grows in size.
- However, the biggest advantage is performance—since all components are tightly integrated; the system is fast and efficient, with minimal overhead from system calls or internal communication.
- In a monolithic structure, all operating system services run in kernel mode and share the same address space. So, if any one service (like a device driver or file system module) fails or has a bug, it can crash the entire operating system.

Layered Approach

- In the layered approach, the operating system is broken down into multiple layers (or levels).
    - The lowest layer (Layer 0) is the hardware.
    - The highest layer (Layer N) is the user interface.
- This layering structure is depicted in Figure-2 below

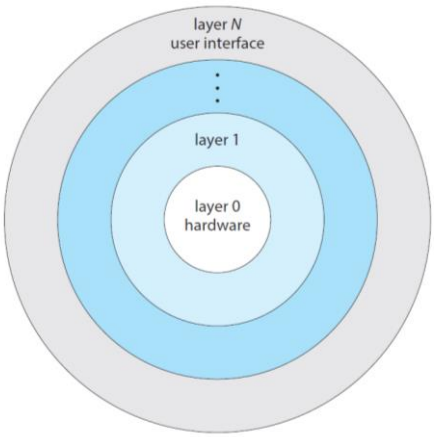

Figure-2 A layered operating system

- Each layer is an abstract object, consisting of data and operations that can manipulate that data.
- A higher layer can call (invoke) the operations provided by the layer directly below it, but not those of layers further down directly.
- Key Advantages
    - **Simplicity:** The system is easier to build and maintain because each layer performs a well-defined function.
    - **Ease of debugging:** Errors can be located more easily because functionality is separated by layers.
- Design Rules
    - Each higher layer uses only the services of the layer directly below it.
    - A layer does not need to know how the lower layer works—it just uses the services provided by it.
    - Lower layers cannot access higher layers.
- Challenges and Limitations
    - **Performance overhead:** If a service needed by a layer is several layers below, the request must pass through each intermediate layer, which slows down execution.
    - **Layer arrangement is tricky:** It's not always easy to decide what functionality should go into which layer. Once set, lower layers cannot be used anything from higher layers, which limits flexibility.
    - **Indirect access to distant layers:** If a layer wants to use a feature from a non-adjacent (farther) layer, it must go through all the in-between layers, which can make the system less efficient.

Microkernels

- The microkernel approach structures the operating system by removing all nonessential components from the kernel and implementing them as user-level programs that run in separate address spaces. As a result, the kernel becomes smaller and simpler.
- Typically, a microkernel provides only the core functionalities, such as:
    - Basic process management
    - Memory management
    - Inter-process communication (IPC) through message passing
- The architecture of a typical microkernel-based operating system is shown in the figure-3 below.

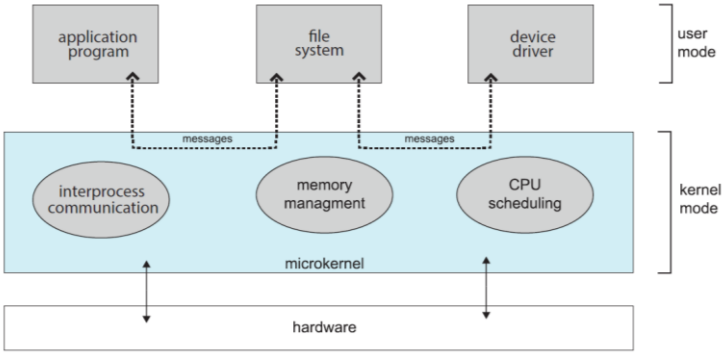

Figure-3 Architecture of a typical microkernel.

- The primary role of the microkernel is to manage communication between:
- Client programs (user applications) and System services (such as file systems, device drivers), which are also running in user space. This communication is handled using message passing.
- Example: When a client program wants to access a file, It does not communicate directly with the file server. Instead, the client and file server exchange messages indirectly through the microkernel, which acts as a mediator.
- **Advantages of the Microkernel Approach**
    - **Easy to extend:** New services can be added in user space without modifying the kernel.
    - **Simpler kernel updates:** Since the kernel is small, changes to it are minimal and less risky.
    - **Portability:** A smaller kernel is easier to adapt to different hardware platforms.
    - **Improved security and reliability:** If a service crashes, it does not affect the rest of the system, because services are isolated.
- **Disadvantages**
    - **Higher overhead:** Frequent message passing between user-level services and the kernel causes more context switching, which can degrade performance.
- **Real-world Example:**
    - One of the most well-known implementations of a microkernel is Darwin, the core of macOS and iOS.

Comparision of Monolithic V/S Layered V/S Microkernel OS architecture.

Feature	Monolithic Kernel	Layered Approach	Microkernel
Structure	Single large kernel binary	Multiple layers	Minimal core kernel + user-level services
Execution Mode	All services run in kernel mode	Each layer runs with layered control	Kernel in kernel mode, services in user mode
Performance	High (tight integration)	Moderate (layer overhead)	Lower (due to message passing)

Reliability	Less reliable (one bug crashes OS)	Better (layer isolation)	High (service isolation)
Ease of Maintenance	Difficult	Easier due to modular layers	Easier, modular and extensible
Communication	Direct function calls	Layer-by-layer calls	Message passing between components
Example	UNIX	The OS	Darwin (macOS/iOS)

References:

- Operating System Concepts: Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Ninth Edition, Wiley India.