Mal. Abubakar sadik Mustapha

Operating System Principles: Lecture Notes General Overview

1. **Structuring Methods:**

Operating systems can be structured in different ways, each with its advantages and disadvantages. Some common methods include:

Monolithic: All components are tightly coupled in a single unit, leading to simplicity but also rigidity and difficulty in extending.

Microkernel: Only essential functionalities reside in the kernel, while others run as user-space processes, offering flexibility but potentially impacting performance.

Layered: The system is organized in layers, where each layer provides services to the one above it, promoting modularity and maintainability.

Hybrid: Combines elements of different methods for a balanced approach.

2. **Abstraction:**

Operating systems provide abstractions that hide the complexity of hardware from applications and users. These abstractions include:

Virtual memory: Presents a contiguous memory space to applications despite physical memory fragmentation.

Files and directories: Provide a logical organization for storing and accessing data.

Processes: Represent units of execution with their own resources and environment.

Devices: Offer uniform access to diverse hardware components.

3. **Resource Management:**

Operating systems manage various resources, including:

CPU: Scheduling processes to utilize the processor efficiently.

Memory: Allocating and reclaiming memory for processes and data.

Storage: Managing disk access and file systems.

I/O devices: Handling device requests and interrupts.

4. **Concept of Resources:**

A resource is any shared entity in the system that can be requested and used by multiple processes. Resources have properties like:

Sharability: Can be used by multiple processes concurrently (e.g., memory) or exclusively (e.g., printer).

Granularity: Can be divided into smaller units (e.g., memory blocks) or cannot (e.g., CPU).

Lifetime: Can be used for a specific duration (e.g., file access) or indefinitely (e.g., CPU).

5. **Concept of APIs:**

Application programming interfaces (APIs) provide standardized ways for applications to interact with the operating system and access resources. APIs typically define:

Function calls: Methods for requesting specific services.

Parameters: Data passed to and returned from function calls.

Return values: Indicate success or failure of operations.

6. **Device Organization and Interrupts:**

Devices are connected to the system through device controllers that manage communication with the CPU. Interrupts are signals sent by devices to notify the CPU about events, such as:

Completion of an I/O operation.

Arrival of new data.

Error conditions.

Monolithic Operating System Structure

In a monolithic operating system structure, all major components, including the kernel, device drivers, memory management, file systems, and process management, exist as a single, large program running directly on the hardware. This approach offers both advantages and disadvantages.

Advantages:

Simplicity: Easier to develop and debug due to tight integration of components.

Efficiency: Direct communication between components leads to potentially faster performance.

Small memory footprint: No need for separate processes for different functionalities.

Disadvantages:

Rigidity: Difficult to extend or modify due to tightly coupled components.

Security vulnerabilities: An issue in one component can compromise the entire system.

Scalability challenges: Scaling can be difficult as the entire system grows larger.

Example:

MS-DOS: A classic example of a monolithic operating system. All functionalities, including file management, memory management, and command processing, resided within a single executable.

Early versions of Windows: Up to Windows 95, Microsoft used a monolithic kernel design. Later versions adopted a hybrid approach with some elements running in user space.

Operation of a Monolithic System:

System calls: Applications and user programs access operating system services through system calls.

Kernel execution: The system call triggers the execution of specific code within the kernel.

Direct hardware access: The kernel directly interacts with hardware components like CPU, memory, and devices.

Resource management: Kernel manages resources like memory, processes, and devices, enforcing policies and scheduling access.

Device drivers: Device drivers, integrated within the kernel, handle communication with specific hardware devices.

Note:

Monolithic kernels, despite their limitations, still power many embedded systems and real-time operating systems due to their simplicity and efficiency.

Modern operating systems often use hybrid approaches, leveraging the benefits of both monolithic and microkernel architectures.

microkernels

microkernel aim to keep the kernel, the core of the operating system, as small and focused as possible. This means only the most essential functionalities, such as memory management, process scheduling, and inter-process communication (IPC), reside within the kernel itself. Other non-essential services like device drivers, file systems, and network protocols run as separate processes in user space, outside the privileged kernel environment.

Advantages:

Enhanced security: Isolating services in user space minimizes the impact of potential vulnerabilities, as a compromised service won't directly affect the core kernel.

Modularity and flexibility: Adding or removing features becomes easier as services are independent, facilitating customization and extension.

Stability: Since the kernel is smaller and simpler, bugs or crashes in user-space services are less likely to bring down the entire system.

Disadvantages:

Performance overhead: Communication between kernel and user-space services through IPC can introduce some performance overhead compared to direct interaction in a monolithic system.

Development complexity: Designing and managing separate services can be more complex than a single integrated system.

Examples:

Mach: Developed by Carnegie Mellon University, Mach served as the basis for several microkernel-based systems.

MINIX: Designed by Andrew S. Tanenbaum, MINIX aimed to provide a transparent and educational microkernel-based system.

L4 microkernel family: A family of research microkernels focusing on security and modularity, used in systems like seL4 and Fiasco.

Operation of a Microkernel System:

Applications and user programs request services through IPC messages.

Microkernel:

Handles IPC messages, routing them to appropriate user-space services.

Manages core functionalities like memory, processes, and scheduling.

User-space services:

Run as independent processes with their own address space.

Provide specific functionalities like device drivers, file systems, and network protocols.

Communicate with each other and the kernel through IPC.

Note:

Microkernel may not be ideal for performance-critical systems due to potential IPC overhead.

Hybrid approaches combining microkernel principles with some monolithic elements are becoming increasingly common in modern operating systems.

Layered:

A layered operating system structure organizes functionalities into distinct, well-defined layers, each building upon the services provided by the layer below. Each layer exposes an API (Application Programming Interface) that allows higher-level layers to interact with it without needing to know the implementation details. This approach offers several advantages:

Advantages:

- Modular design: Easier to develop, maintain, and modify individual layers independently.
- Flexibility: New layers can be added or removed to provide new functionalities or adapt to specific needs.
- Abstraction: Hides hardware complexity and provides consistent interfaces for applications.

Disadvantages:

- Performance overhead: Communication between layers can introduce some overhead compared to direct interaction in a monolithic system.

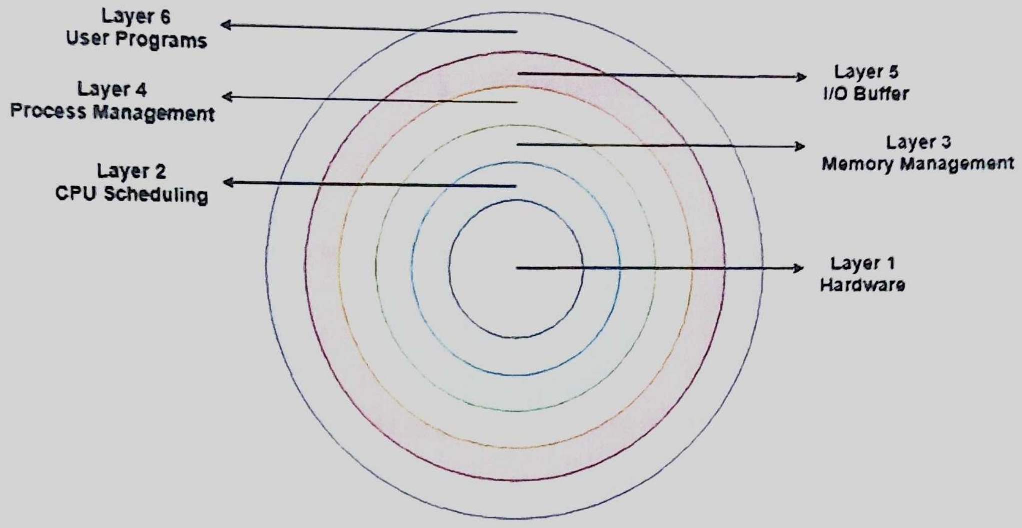
- Potential complexity: Understanding interactions between layers can be challenging for developers.

Examples:

- THE SYSTEM/360: An early IBM operating system considered one of the first to implement a layered design.
- Multics: Another pioneering layered system developed by MIT, aiming for security and resource sharing.
- Modern operating systems: While not purely layered, many contemporary systems like Linux and Windows incorporate elements of layer-based organization for specific functionalities.

Operation of a Layered System:

1. Applications and user programs interact with the highest layer, often providing user interface or application logic.
2. Each layer:
   - Provides specific services and functionalities (e.g., memory management, file access, I/O operations).
   - Uses services from the layer below through its API.
   - Exposes its own API for the layer above.
3. Hardware layer: The bottom layer interacts directly with the physical hardware components.



Layers of operating system

Note:

- The number and specific functionalities of layers can vary depending on the operating system design.
- Layered structures can be combined with other approaches like microkernels for enhanced flexibility and security.

Hybrid:

A hybrid operating system structure combines elements of other approaches, such as monolithic and microkernel, to overcome their limitations and achieve a balance between performance, flexibility, and security. Here's how it works:

Structure:

- Core kernel: Handles essential functionalities like memory management, process scheduling, and inter-process communication (IPC). This core may be closer to a microkernel design, focused on minimal functionality and security.
- User-space modules: Additional functionalities like device drivers, file systems, and network protocols run as separate processes outside the kernel, similar to a microkernel approach.
- Hybrid components: Some functionalities may have elements of both approaches. For example, a driver's core logic might run in the kernel for performance, while device-specific communication runs in user space for flexibility.

Advantages:

- Flexibility: Easier to extend and modify compared to monolithic systems. New modules can be added or removed in user space without affecting the core kernel.
- Security: Isolating non-essential services in user space reduces the attack surface and potential impact of vulnerabilities.
- Performance: Core functionalities in the kernel benefit from direct hardware access, while user-space modules offer more flexibility and potential for optimization.

Disadvantages:

- Complexity: Managing interactions between kernel and user-space modules can be more complex than a monolithic system.
- Performance overhead: IPC between kernel and user space can introduce some overhead compared to direct interaction in a monolithic system.

Examples:

- Apple macOS: Combines a microkernel (XNU) with user-space modules for device drivers and frameworks.
- Microsoft Windows NT family: Uses a hybrid kernel where core functionalities like process management reside in the kernel, while user-space subsystems handle graphics, networking, and other services.
- Android: Leverages a Linux kernel (closer to a microkernel) with custom user-space modules for hardware interactions and Android frameworks.

Operation of a Hybrid System:

1. Applications and user programs interact with user-space modules or directly with the kernel depending on the required functionality.
2. User-space modules:
   - Provide specific functionalities.
   - May interact with the kernel through IPC for core services.
   - May interact with other user-space modules directly.
3. Kernel:
   - Handles essential functionalities and provides services to user-space modules through IPC.
   - Manages hardware resources like memory and CPU.

Note:

- The specific design and balance between kernel and user-space components can vary depending on the operating system.
- Hybrid approaches offer a compromise between different design philosophies, striving for a balance between efficiency, flexibility, and security.