Operating System Structure

Dr Anas Al-dabbagh

In this lesson

- Operating System Services
- Command-Line Interface (CLI)
- User-Friendly Desktop Interface
- Touchscreen Interfaces
- System Calls
- Operating System Structure

Operating System Services

Operating systems provide an essential environment for the execution of programs, offering numerous services to both users and applications. These services enhance user experience and facilitate program operations. Some key services include:

- **User Interface (UI)**: Most operating systems feature a UI, which can vary between:
  - **Command-Line Interface (CLI)**: Text-based interaction.
  - **Graphical User Interface (GUI)**: Visual-based interaction with windows, icons, and menus.
  - **Batch**: Executes commands in sequence without direct user interaction.
- **Program Execution**: The OS can load a program into memory, execute it, and terminate it—either normally or abnormally, if errors occur.
- **I/O Operations**: Programs often need to perform input/output operations, such as reading/writing files or interacting with I/O devices, which the OS manages.

Operating System Services

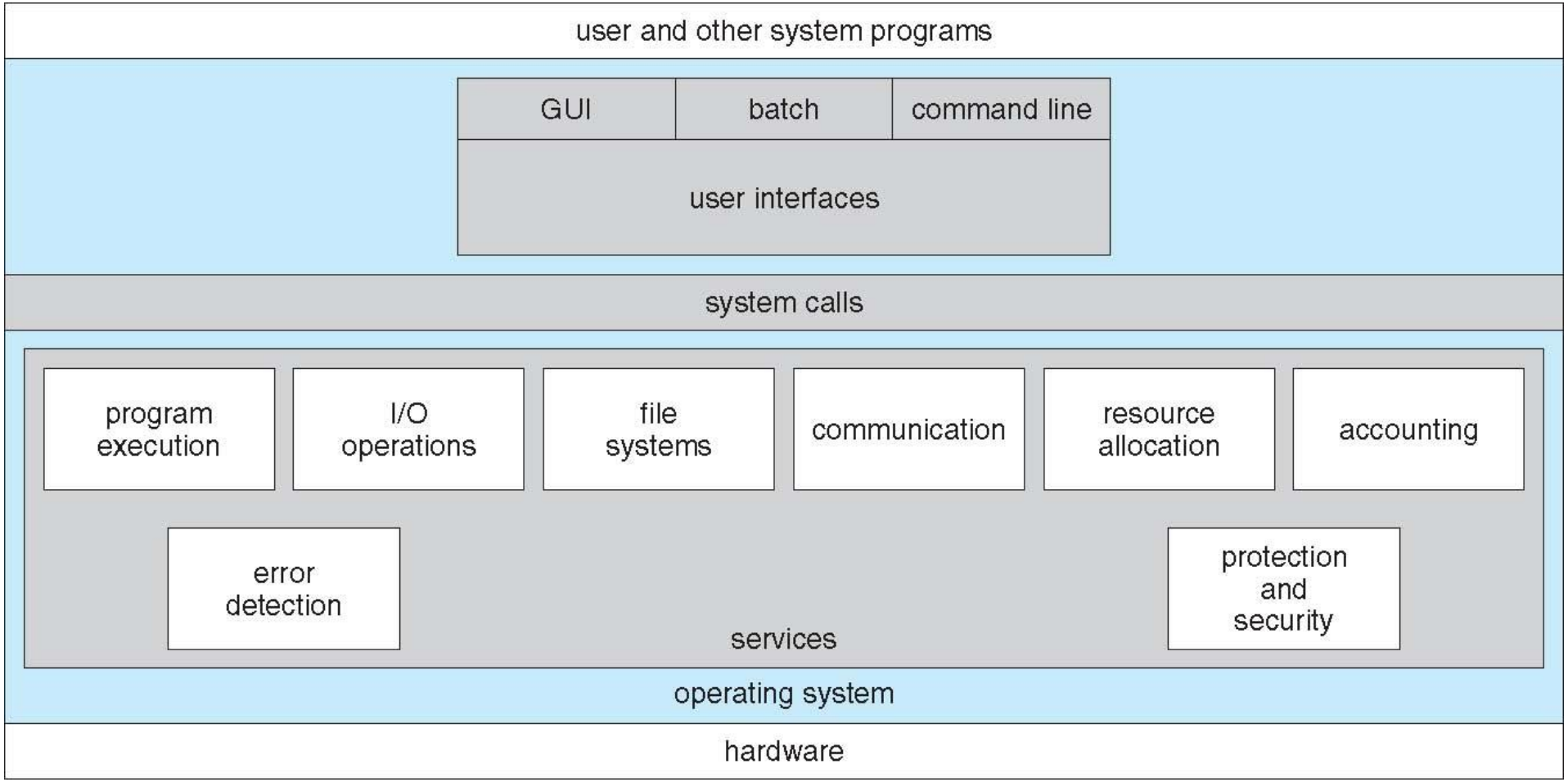
Operating systems provide various services to enhance user interaction and program management. Here are additional critical functions:

- **File-System Manipulation**: The OS enables programs to interact with the file system, allowing operations like reading, writing, creating, and deleting files or directories. It also manages file permissions and provides information retrieval capabilities.
- **Communications**: Facilitates process communication either on the same machine or across a network. This can be done using shared memory or message passing (packets managed by the OS).
- **Error Detection**: The OS monitors for errors across the CPU, memory, I/O devices, and user programs. It handles errors to maintain system stability and consistency, providing debugging tools for troubleshooting.

Operating System Services

- **Resource Allocation:** For concurrent users and jobs, the OS must allocate resources (CPU time, memory, file storage, and I/O devices) efficiently.
- **Accounting:** The OS tracks resource usage, identifying how much and what types of resources users or jobs consume. This is important for system monitoring and user billing.
- **Protection and Security:** The OS ensures that unauthorized access to system resources is prevented, maintaining protection between processes. It also provides:
- **User authentication** for access control.
- **Security mechanisms** to safeguard against external threats and unauthorized access attempts.

A View of Operating System Services



Command-Line Interface (CLI):

- The CLI, also known as the command interpreter, allows users to enter commands directly to the system.
- Multiple variations of the CLI may exist, known as *shells*.
- The primary function of the CLI is to receive commands from the user and execute them.
- Some commands are built into the shell, while others refer to external programs.

User-Friendly Desktop Interface:

- A desktop metaphor interface utilizing mouse, keyboard, and monitor.
- Icons represent files, programs, and actions.
- Mouse interactions trigger actions such as displaying information, executing functions, or opening directories (folders).
- Modern systems often include both CLI and GUI interfaces:
  - **Windows:** GUI with CLI “command” shell.
  - **Unix/Linux:** Primarily CLI, with optional GUIs like KDE and GNOME.

Touchscreen Interfaces

- Touchscreen devices require new interfaces
  - Mouse not possible or not desired
  - Actions and selection based on gestures
  - Virtual keyboard for text entry
- Voice commands.

System Calls

- **Programming Interface: Access to OS Services**
  - Provides a way for applications to interact with the operating system.
  - Usually implemented in high-level languages like C or C++.
  - Applications commonly use a high-level **Application Programming Interface (API)** instead of directly invoking system calls.

System Call Implementation

System calls are the interface between a running program and the operating system. They allow programs to request services such as file operations, process management, and communication with hardware.

Implementation Steps:

1. System Call Number: Each system call is associated with a unique identifier or number. To help the operating system to identify which service is requested.

2. Trap to Kernel: The user application triggers a software interrupt (trap) to switch from user mode to kernel mode, where the operating system can safely execute the requested service.

3. System Call Handler: The operating system has a handler that interprets the system call number and invokes the corresponding service routine.

System Call Implementation Steps

4. Execution in Kernel Mode: The kernel executes the requested operation with privileged access to system resources. After completion, the result (if any) is passed back to the calling program.

5. Return to User Mode: The control is transferred back to the user mode along with the results of the system call (e.g., file descriptor, process ID, or error code).

Example:

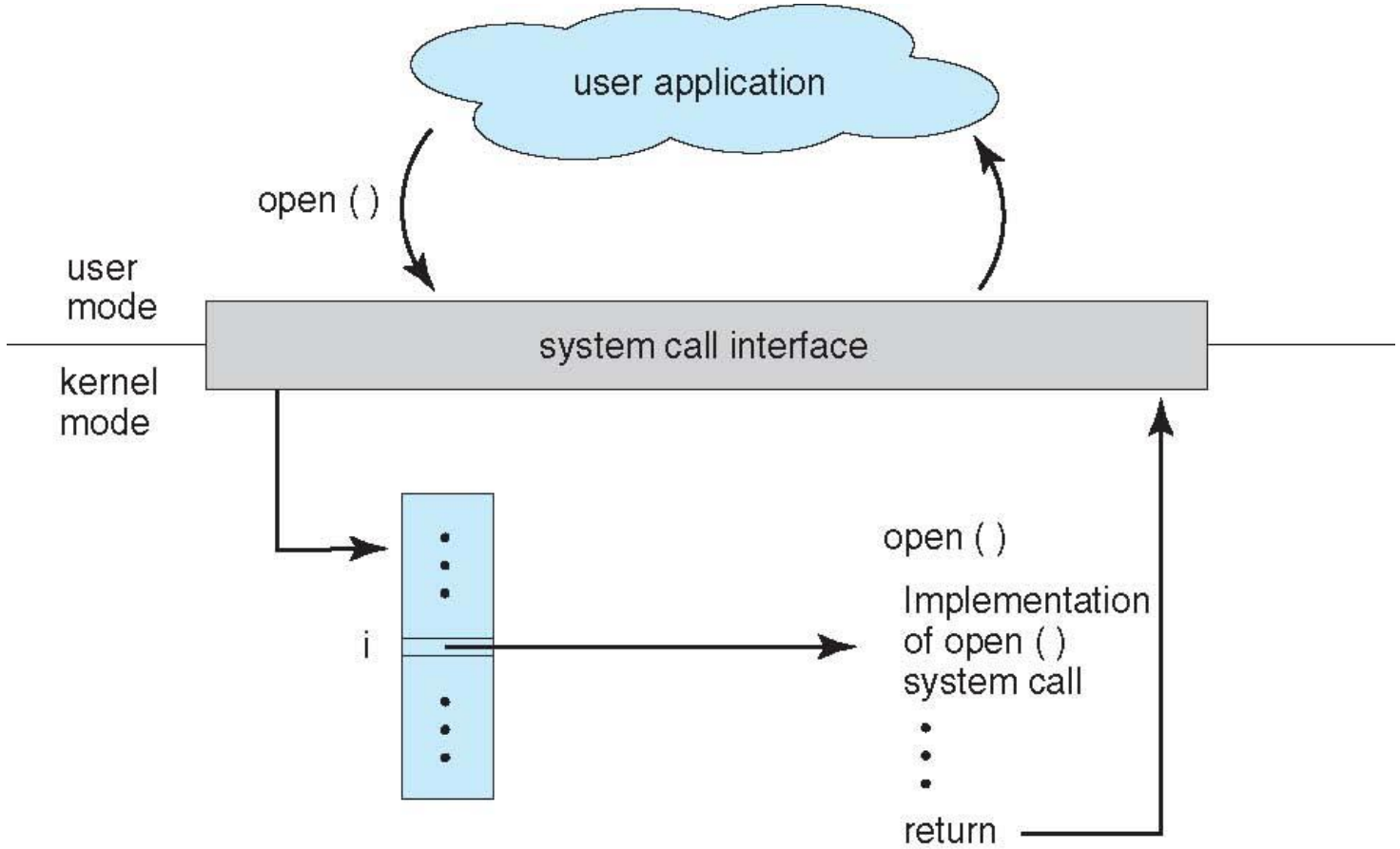
- int fd = open("file.txt", O_RDONLY);

- System call: open()

Action: Opens a file for reading

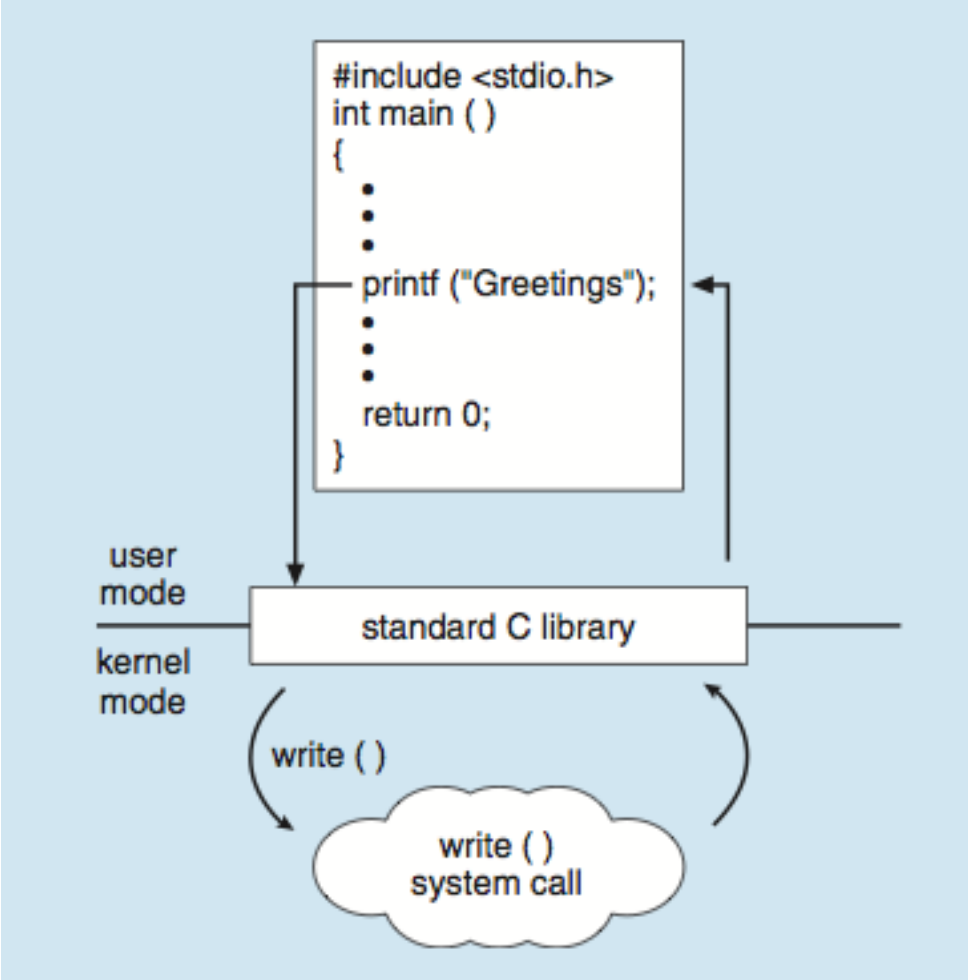
Kernel Service: Allocates resources and returns a file descriptor to the user program.

API – System Call – OS Relationship



Standard C Library Example

C program invoking printf() library call, which calls write() system call



Operating System Structure

The structure of an OS determines how it manages processes, memory, hardware devices, and file systems. Different architectures have been developed to optimize performance, security, and ease of maintenance.

1. Monolithic Kernel: A single large process that incorporates all system services (e.g., memory management, I/O operations) into one kernel space.

- Advantages:

  - High performance due to direct communication between services.

  - Simplified execution model without frequent context switches.

- Disadvantages:

  Harder to maintain or update due to lack of modularity.

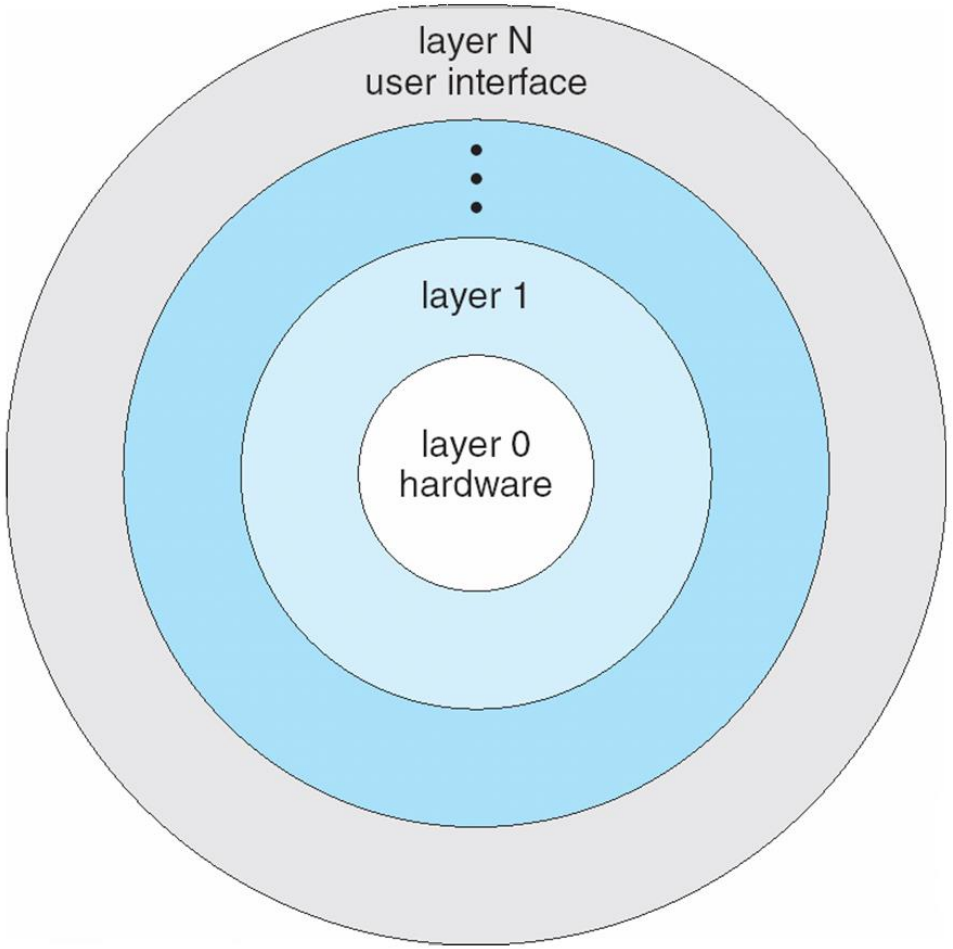
  A failure in one part of the kernel can potentially crash the whole system.

Operating System Structure: Layered Approach

2. Layered Approach: OS is divided into layers where each layer is built on the top of the previous one, offering a separation of concerns.

- Advantages:
    - Modularity allows each layer to be independently tested and maintained.
    - Easier to debug and enhance functionality by isolating changes.
- Disadvantages:
    - Performance degradation from overhead associated with crossing multiple layers.
    - Strict separation may lead to inefficiencies, especially in inter-layer communication.

Operating System Structure: Layered Approach



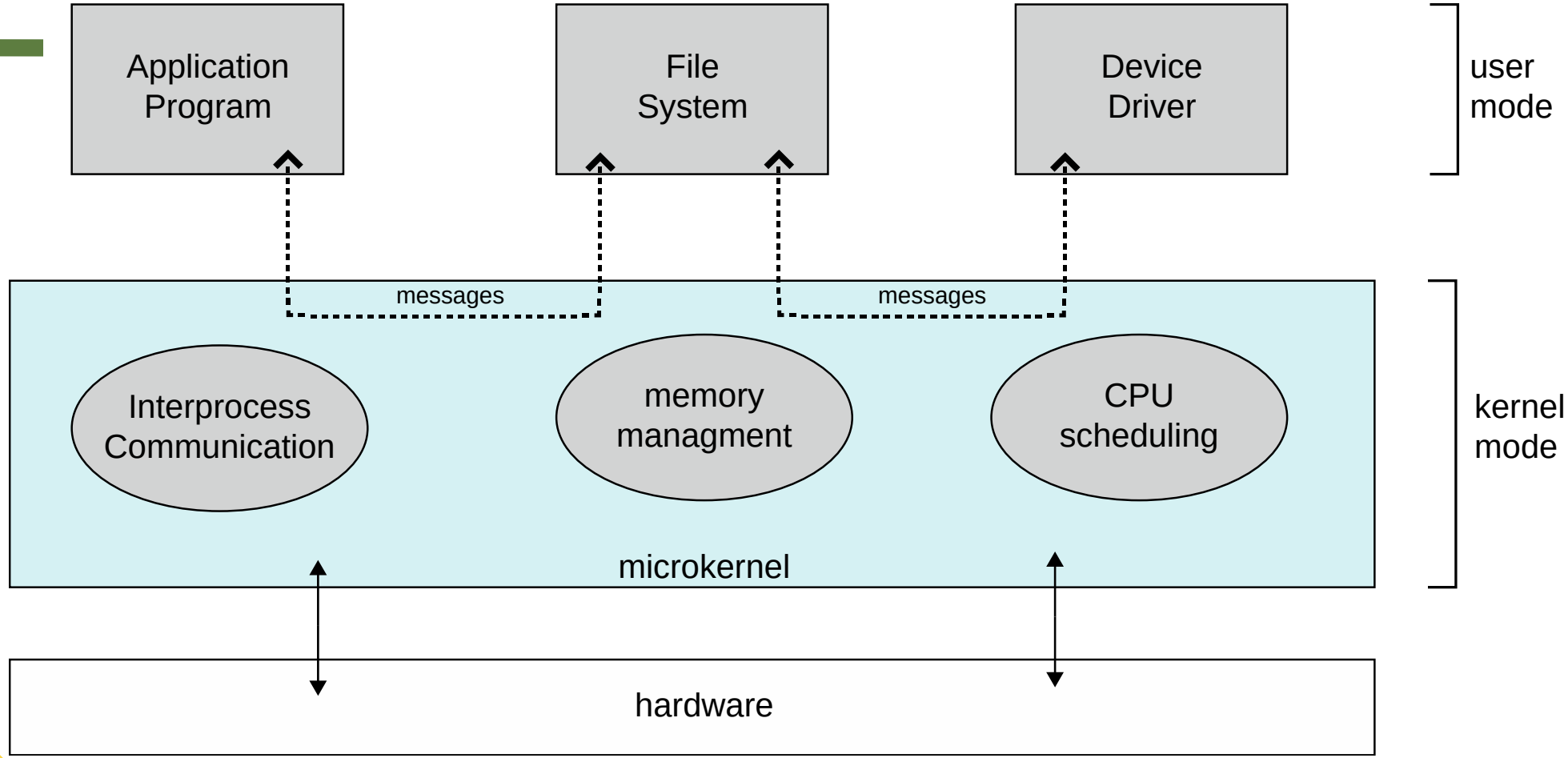
Operating System Structure: Microkernel

3. Microkernel: Minimalist kernel that only manages core functions like process scheduling and inter-process communication, while other services run in user space.

- Advantages:
  - Higher system security and stability—faults in user space services do not affect the kernel.
  - More flexible and easier to extend or modify.
- Disadvantages:
  - Increased overhead due to message passing between kernel and user services.
  - Potential for lower performance in comparison to monolithic kernels.

The choice of OS structure depends on the system requirements, with trade-offs between performance, modularity, and security. Monolithic kernels are faster but harder to maintain, layered designs offer simplicity, while microkernels provide flexibility and robustness at the cost of performance

Operating System Structure: Microkernel



Thank you