OPERATING SYSTEM STRUCTURES

COMPONENTS OF OPERATING SYSTEM

- An operating system (OS) comprises several key components that manage hardware resources and provide a user interface.

➢These include

A) the kernel, a software, which is the core of the OS, handling essential tasks like memory and process management.

B) The shell, which acts as the interface between the user and the kernel, allowing users to interact with the system

C) Various management components like process management, file management, memory management, and I/O device management.

❖What is the Kernel of an Operating System?

- The **kernel** is the **core component** of an operating system. It acts as a **bridge between software and hardware**. All operating system functions depend on the kernel to manage hardware, memory, and processes.
- The **kernel** is the **heart of the operating system** that directly controls the computer's hardware and allows software (like apps) to use the system’s resources.

❑The kernel is like a manager in a factory:

Workers = Apps

Machines = Hardware

Manager (kernel) makes sure workers use machines properly, don’t fight, and everything runs smoothly.

- Without kernel, system will never know how to schedule apps, protect memory, talk to a keyboard, ssd or graphics card.
- If you open browser, spotify and vscode, kernel schedules tiny timeslicing in processor for it. This constant juggling is what you call as multitasking.
- Each app will be given its own memory space. When one app tries to access other apps memory space we get error. (segmentation fault error in C)
- When an app want to save a file or talk to wifi chip it cannot touch hardware directly but it makes a system call. Kernel then interrupts the current process and does the job.

❑Main Functions of a Kernel:

- Process Management - Starts, stops, and manages multiple programs (called processes).
- Memory Management - Gives memory to programs and takes it back when done.
- File System Access - Lets programs read and write files on disk.
- Device Management - Talks to hardware (keyboard, screen, printer) using drivers.
- Security & Protection - Keeps programs from interfering with each other.

❑Types of Kernels:

- Monolithic Kernel - All core OS functions are in one large block (e.g., Linux)
- Microkernel - Only essential parts are in the kernel; others are in user space (e.g., QNX)
- Hybrid Kernel - Mix of both; used by Windows and macOS

1. Monolithic Kernel:

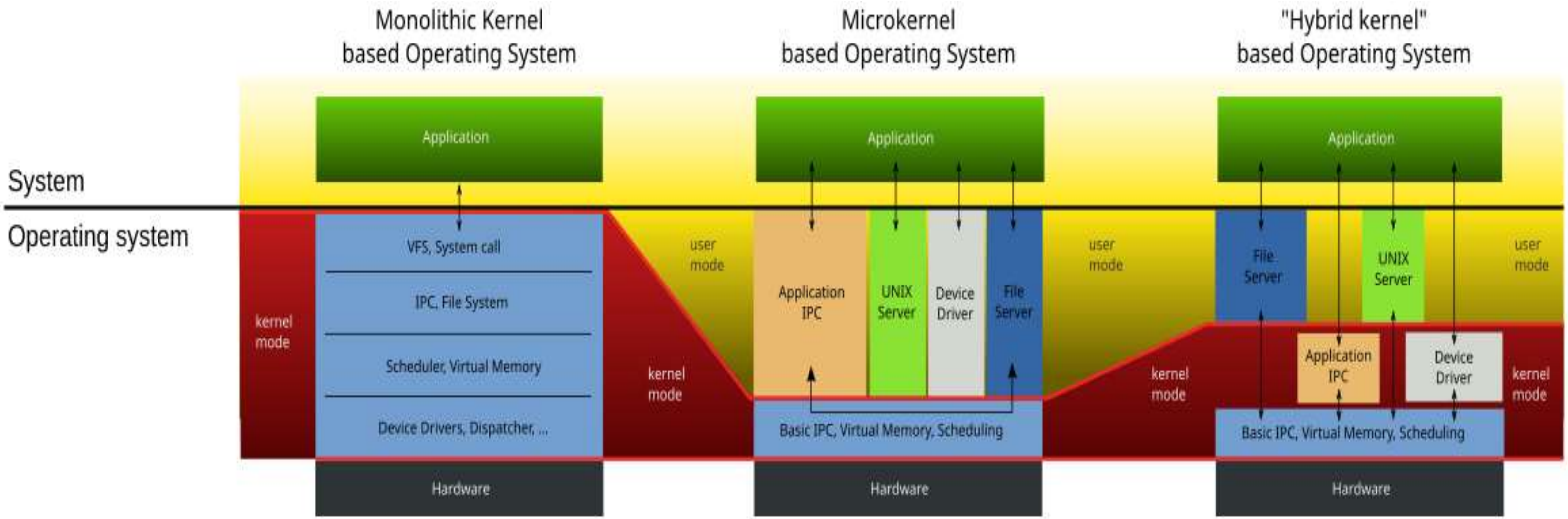
- In a monolithic kernel, all core operating system services (like memory management, process scheduling, file systems, etc.) run in the same memory space, within the kernel.
- This design can lead to faster execution speeds due to direct communication between components.
- However, a bug in one component can potentially crash the entire system.

2. Micro Kernel

- Microkernels keep the kernel code small and minimal, handling only essential functions like inter-process communication (IPC) and basic scheduling.
- Other OS services (like file systems, device drivers) run as separate processes in user space, communicating with the kernel via message passing.
- This design enhances system reliability, as a failure in a user-space process is less likely to affect the entire system.

3. Hybrid Kernel:

- Hybrid kernels combine aspects of both monolithic and microkernels.
- They aim to provide the performance benefits of monolithic kernels while maintaining some of the modularity and reliability of microkernels.
- Often, they include a small microkernel-like core with some services running in kernel space, while others are in user space.


Monolithic Kernel
based Operating System
Microkernel
based Operating System
"Hybrid kernel"
based Operating System
System
Operating system
Application
VFS, System call
IPC, File System
Scheduler, Virtual Memory
Device Drivers, Dispatcher, ...
kernel
mode
Hardware
user
mode
Application
IPC
UNIX
Server
Device
Driver
File
Server
Basic IPC, Virtual Memory, Scheduling
Basic IPC, Virtual Memory, Scheduling

- 1. Process Management - Manages processes (running programs): creation, scheduling, and termination.
- 2. Memory Management - Controls how RAM is allocated, used, protected, and freed.
- 3. File System Management - Organizes and manages files and directories on storage devices.
- 4. Device Management - Manages input/output devices using drivers and buffers.
- 5. I/O Management - Coordinates input and output operations with efficient data transfer.
- 6. Secondary Storage Management - Manages non-volatile storage (like HDDs/SSDs) and their access.
- 7. Security and Protection - Ensures data privacy, authentication, and controlled access to resources.
- 8. Networking - Supports communication between devices via networks (TCP/IP protocols).
- 9. User Interface (UI) - Allows users to interact with the system via CLI or GUI.
- 10. System Calls / API - Interface that allows programs to request services from the OS.

OPERATING SYSTEM SERVICES

- 1. Program Execution - Loads and runs programs and manages their execution.
- 2. I/O Operations - Handles input/output (keyboard, mouse, printer, etc.) so programs don’t deal directly with hardware.
- 3. File System Manipulation - Lets users and programs create, read, write, and delete files and directories.
- 4. Communication Services - Manages communication between processes (within one computer or across networks).
- 5. Error Detection - Monitors the system and notifies when errors occur (like file not found or hardware failure).

- 6. Resource Allocation - Decides who gets to use CPU, memory, files, etc. and how much.
- 7. Security and Protection - Protects data and resources from unauthorized access and ensures programs don’t harm each other.
- 8. User Interface (UI) - Provides a way for users to interact with the OS — like Command Line Interface(CLI) or Graphical User Interface(GUI).

SYSTEM CALLS

- A system call is like a special request that a program sends to the Operating System to ask it to do something important. Programs can't directly talk to hardware (like memory or printers), so they ask the **OS** to do it using **system calls**.
- A system call is a function provided by the Operating System that allows user-level programs to request services from the kernel.
- It’s the interface between:

✓User space (where your apps run)

✓Kernel space (where the OS core works)

- How System Calls Work:

1. Request:

- An application initiates a system call, specifying the desired operation (e.g., reading a file, creating a process).

2. Transfer to Kernel:

- The system call mechanism (often a software interrupt) transfers control to the kernel.

3. Kernel Execution:

- The kernel, with its higher privileges, performs the requested operation, accessing hardware or other resources as needed.

4. Return:

- Once the operation is complete, the kernel returns control back to the application, potentially providing the result of the operation.

SYSTEM PROGRAMS

- What Are System Programs?
- System programs are utilities and tools provided by the operating system to:

✓Manage system resources

✓Assist the OS in performing basic tasks

✓Provide an environment for user programs

✓They act as the interface between the OS kernel and the user.

- File Management Programs
    - These programs allow users and applications to create, read, write, delete, and organize files and directories.
    - Examples: cp (copy), mv (move), rm (remove), mkdir (make directory) in Linux, File Explorer in Windows
- Process Management Programs
    - These help manage processes (programs in execution), allowing monitoring, creation, termination, and prioritization.
    - Examples: ps, top, kill (Linux), Task Manager (Windows)
- Device Management Programs
    - Used for managing and communicating with hardware devices like printers, USB drives, and more.
    - Examples: Device Manager (Windows), Printer configuration tools (Linux: lpadmin)

- Information and Status Programs
- Provide information about the system’s status and performance.
- Examples: df (disk space), free (memory usage), uptime(how long the system has been running since the last boot)
- System Monitor (Linux), Performance Monitor (Windows)
- Communication Programs
- Enable communication between users, processes, or remote systems.
- Examples: mail, write, ssh, telnet, Remote Desktop, Chat tools (for system communication)
- Compilers and Assemblers
- These translate high-level programming code into machine code or low-level code that the computer can understand.
- Examples: GCC (GNU Compiler Collection), Java Compiler (javac), Assemblers etc.

- Editors
- Text editors used to create or edit source code or configuration files.
- Examples: nano, vim, gedit (Linux), Notepad (Windows)
- System Utilities
- Help with system maintenance, configuration, and optimization.
- Examples: Disk Cleanup, Disk Defragmenter (Windows), cron, logrotate, systemctl (Linux)
- Security and Access Control Programs
- Manage users, passwords, permissions, and overall system security.
- Examples: passwd, chmod, User Account Control (Windows)