Centre Universitaire de Mila

2nd year of Computer Science degree (LMD)

Module : Operating Systems 1

Bessouf Hakim

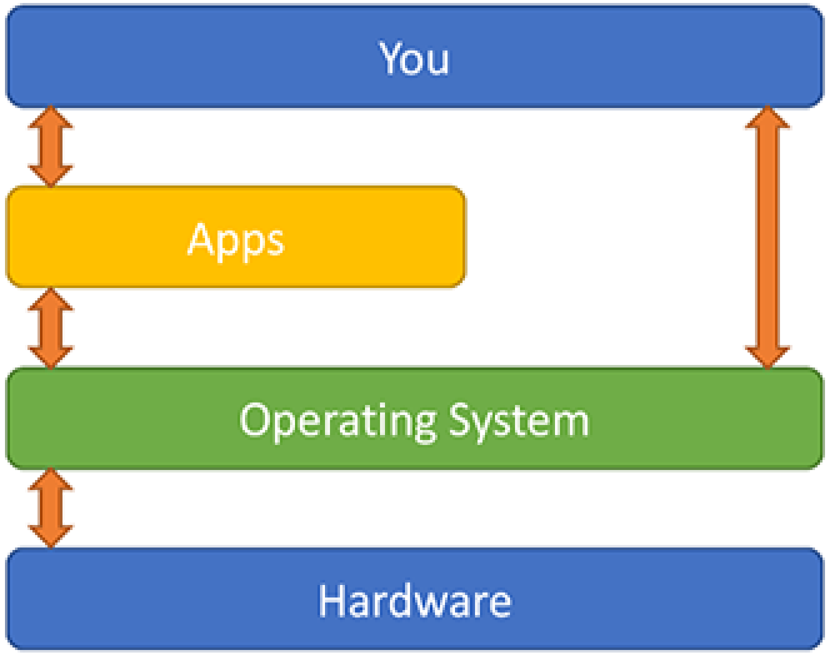
Program

- **Chapter 1: Introduction to Operating Systems**
- **Chapter 2: Basic Mechanisms of Program Execution**
- **Chapter 3: Physical Input/Output Management**
- **Chapter 4: Central Processor Management**
- **Chapter 5: Central Memory Management**
- **Chapter 6: Peripheral Management**
- **Chapter 7: File Management**

Chapitre 1
Introduction to Operating Systems

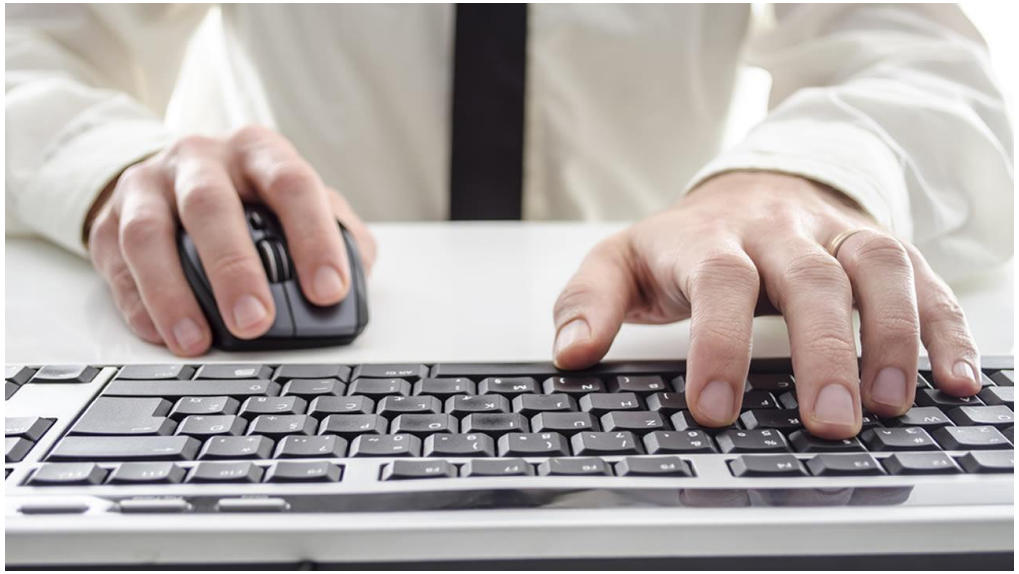
What is an Operating System?

- **Definition:** "An OS is system software that manages hardware and software resources and provides common services for computer programs."
- **Key functions:**
  - ✓Process management
  - ✓Memory management
  - ✓File management
  - ✓I/O management
  - ✓Security



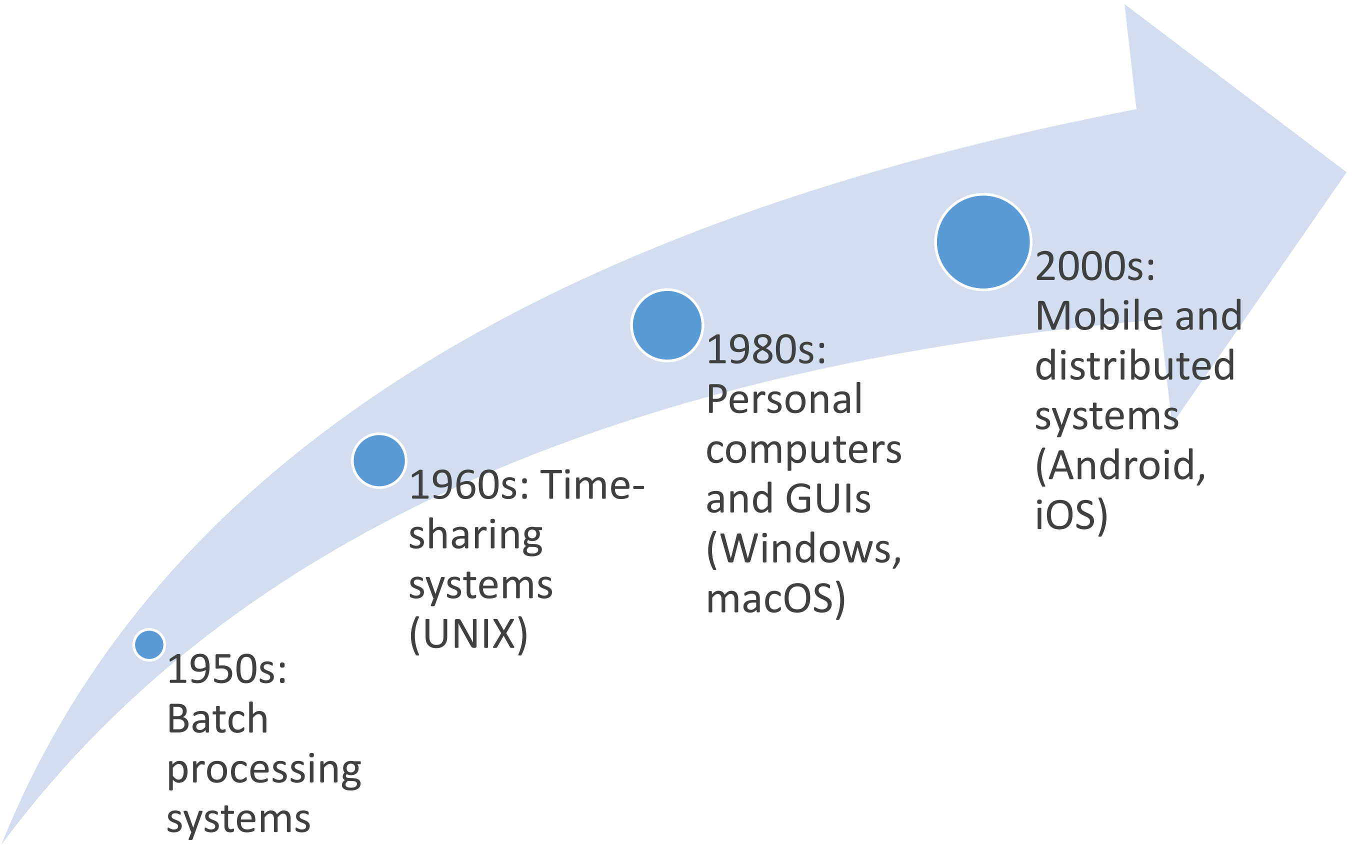
Why are Operating Systems Important?

- Manages hardware resources efficiently.
- Provides a user-friendly interface.
- Enables multitasking and resource sharing.
- Ensures security and protection.



History of Operating Systems

- Open door systems
- Systems with chain monitor
- Batch processing systems
- Multiprogramming systems
- Time sharing systems
- Parallel systems
- Distributed systems
- Personal computer systems
- Real time systems
- Embedded systems



Types of Operating Systems

Type of OS	Description	Features	Examples
Batch OS	Processes a group of jobs without interaction with the user.	Jobs are collected, processed, and outputted in batches; no real-time interaction.	IBM 1401, early mainframe OS
Time-sharing OS	Allows multiple users to share system resources simultaneously.	Time is divided into small intervals to allocate CPU time to each user or process.	UNIX, Multics
Real-time OS	Designed for applications that require immediate response to external events.	Predictable and deterministic responses; can be hard or soft real-time.	VxWorks, RTEMS, QNX
Distributed OS	Coordinates multiple computers to act as one unified system.	Multiple machines work together, sharing resources, and processing tasks.	Google Fuchsia, OpenMosix
Network OS	Manages and provides resources for networked computers and devices.	Focuses on communication and resource sharing over networks.	Novell NetWare, Microsoft Windows Server
Mobile OS	Designed specifically for mobile devices like smartphones and tablets.	Optimized for touch interfaces, mobility, and low power consumption.	Android, iOS, Windows Phone

Eevolution of Computer Systems

- Single user mode systems
- Systems with Job Monitor
- Batch processing systems
- Multiprogramming systems
- Time sharing systems
- Parallel systems
- Distributed systems
- Personal computer systems
- Real time systems
- Embedded systems

Single user mode systems

These computer systems consist of a card reader for reading programs and data, a computer for executing the programs, and a printer for outputting the results. These systems do not use an operating system. To run a program, the user follows these steps:

➢ Code the source program on punched cards (written in Fortran or assembly language).

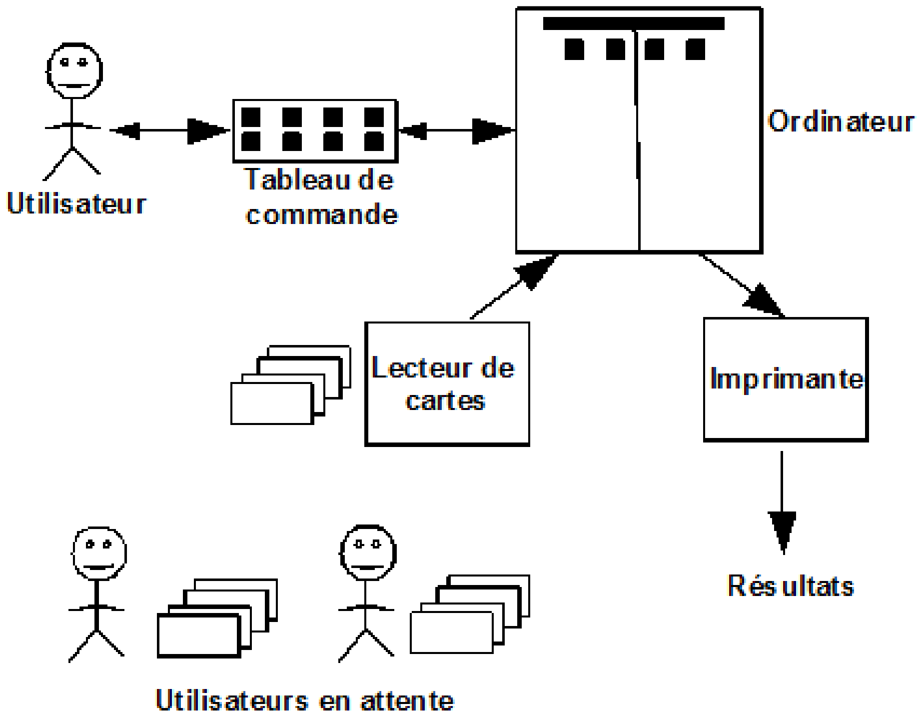
➢ Load the card reading program.

➢ Compile the source program.

➢ Insert the data cards into the card reader.

➢ Execute the compiled program.

➢ Retrieve the results from the printer.

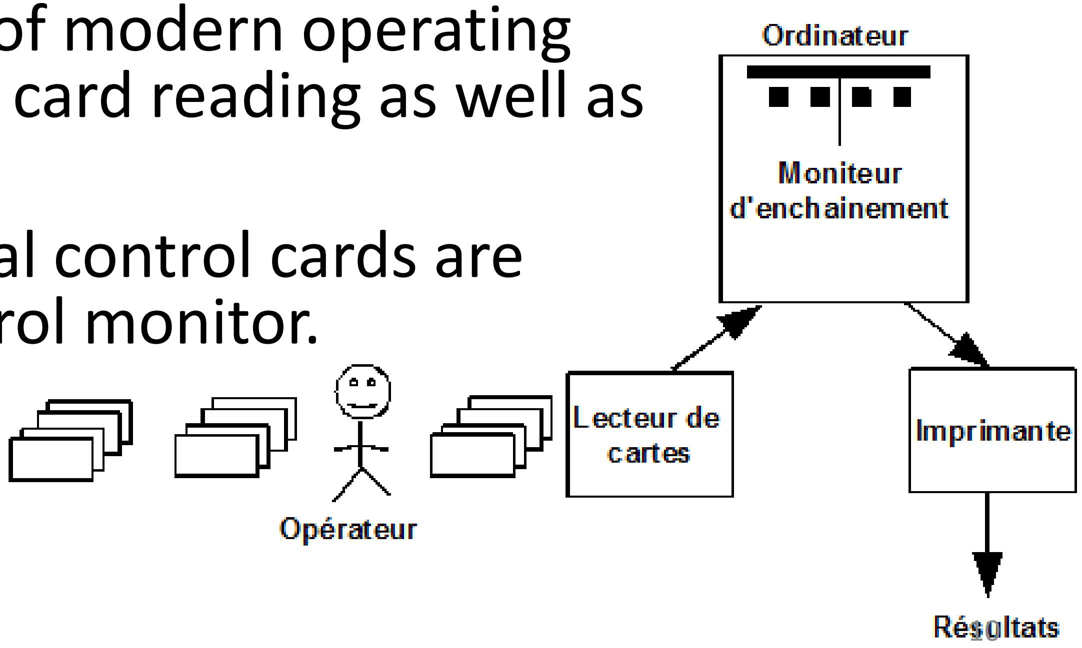


Systems with Job Monitor

In these systems, an operator simply loads the job cards into the card reader and retrieves the results from the printer.
The **job control monitor** (a special program) is responsible for reading, loading, compiling, and executing the programs, thereby saving time.

The job control monitor is the predecessor of modern operating systems. It resides in memory and manages card reading as well as program execution.

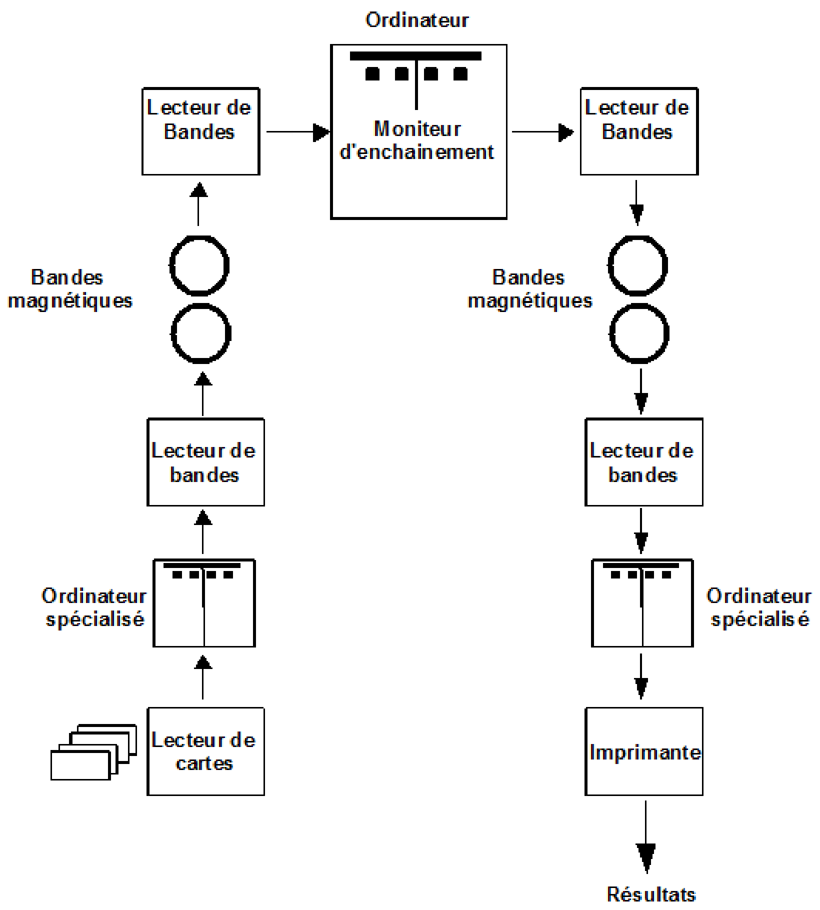
To control the execution of programs, special control cards are used, which are interpreted by the job control monitor.



Batch processing systems

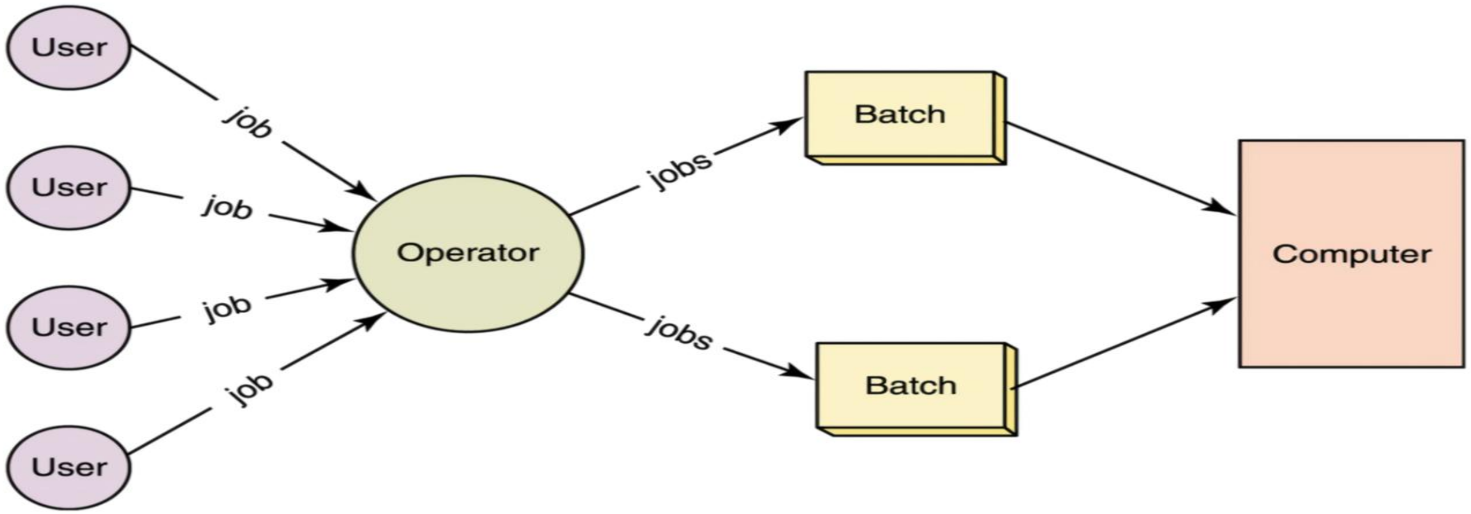
In these systems, specialized intermediate machines handle input/output operations. These machines read the job cards and store them on a magnetic tape. Then, the job control monitor executes these jobs one by one and saves the results on another magnetic tape. A third machine then prints the results on paper.

Since the magnetic tape reader is faster than the card reader, data reading and result writing are accelerated. Additionally, card reading, result printing, and job execution can occur simultaneously, improving overall performance.



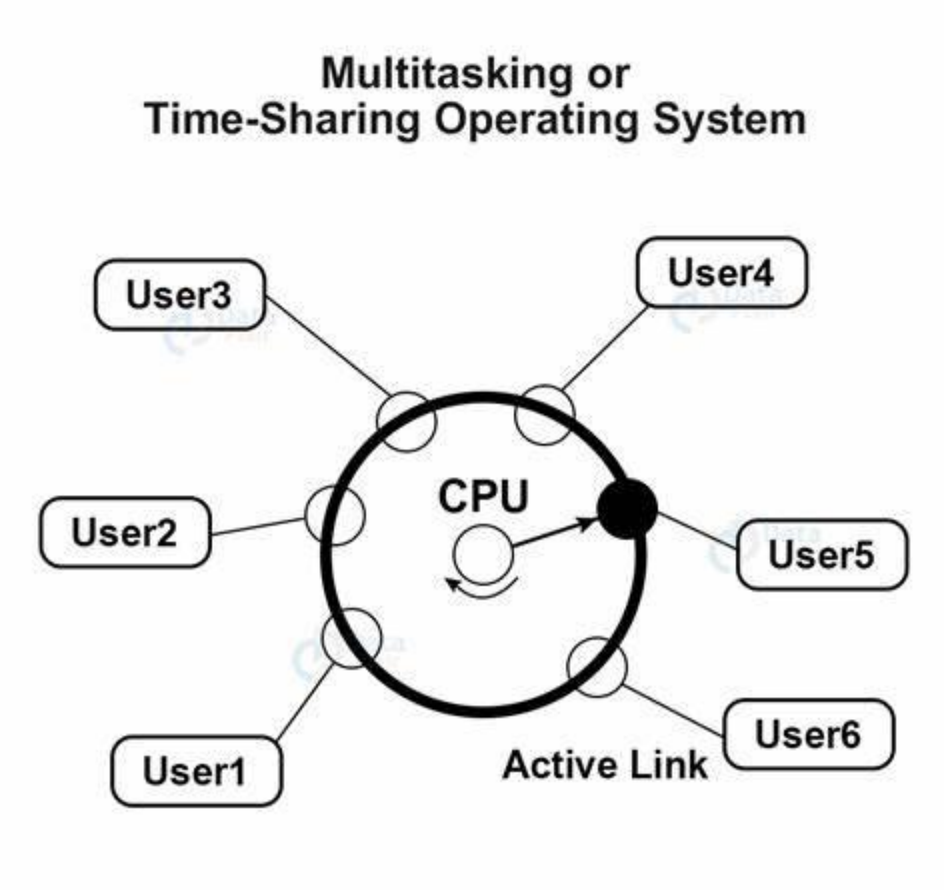
Batch Operating Systems

- Definition: Jobs are executed in batches without user interaction.
- Example: Early mainframe systems.
- Pros: Efficient for large-scale tasks.
- Cons: No user interaction, long wait times.



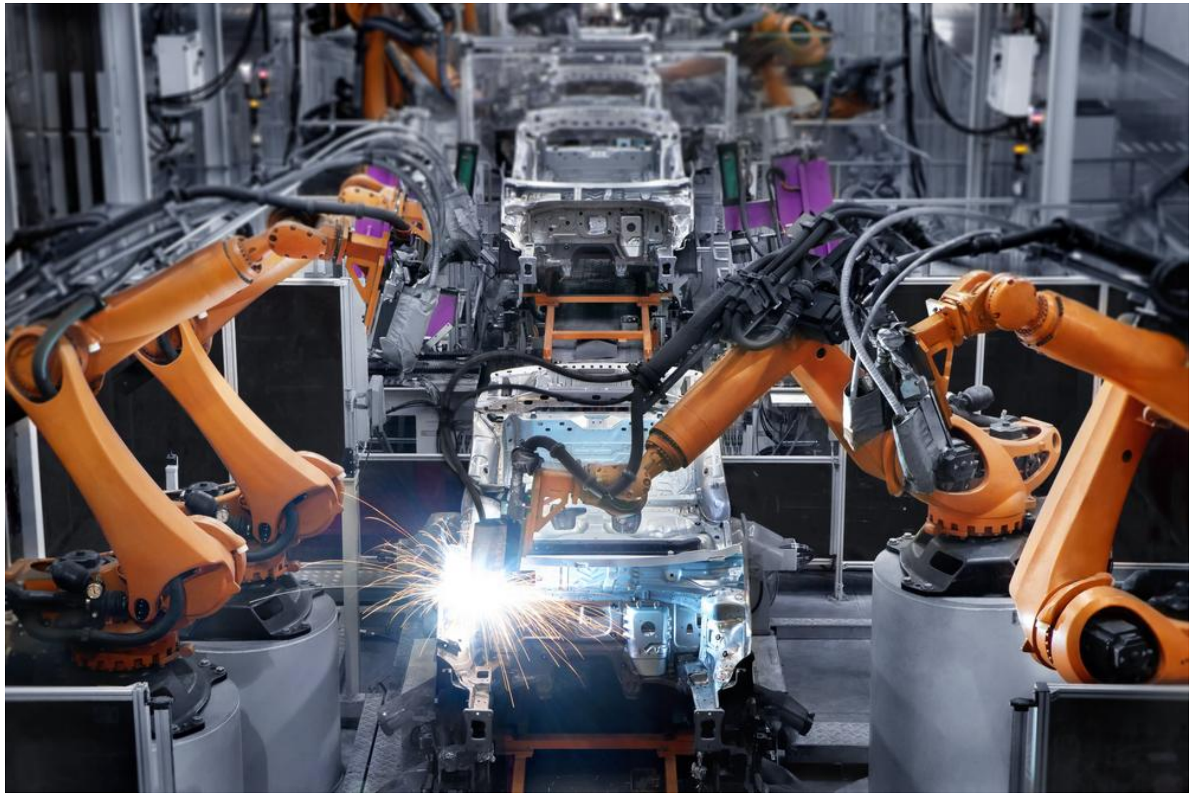
Time-Sharing Operating Systems

- Definition: Multiple users share system resources simultaneously.
- Example: UNIX.
- Pros: Efficient resource utilization, interactive.
- Cons: Complex scheduling, potential for performance issues.



Real-Time Operating Systems

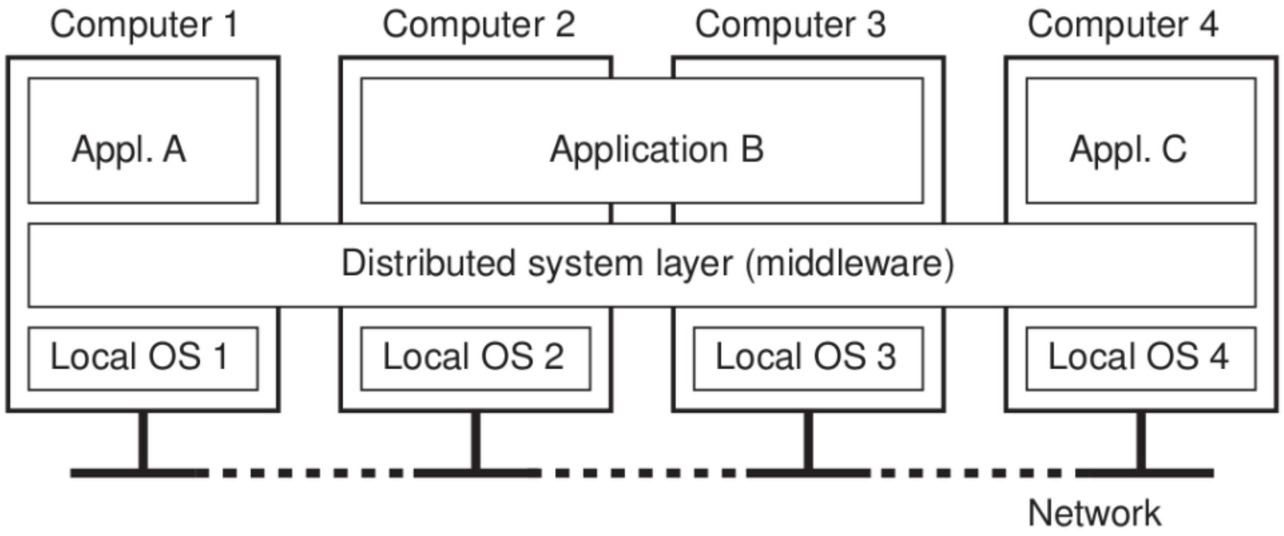
- Definition: OS designed for real-time applications (e.g., robotics, embedded systems).
- Example: VxWorks, FreeRTOS.
- Pros: Predictable and fast response times.
- Cons: Limited functionality, specialized use cases.



Distributed Operating Systems

- Definition: Manages a group of independent computers as a single system.
- Example: Google's distributed systems.
- Pros: Scalability, fault tolerance.
- Cons: Complexity, network

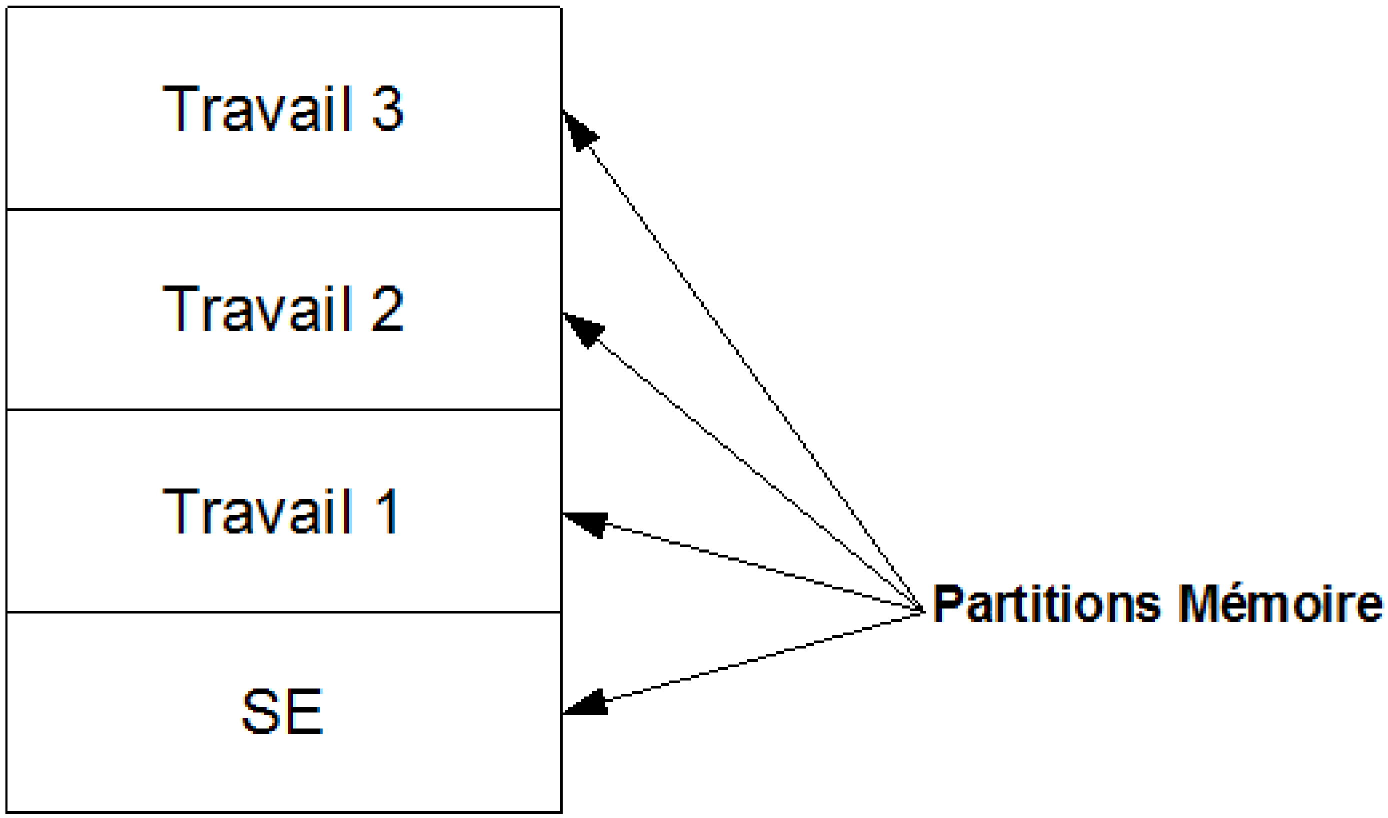
dependency.



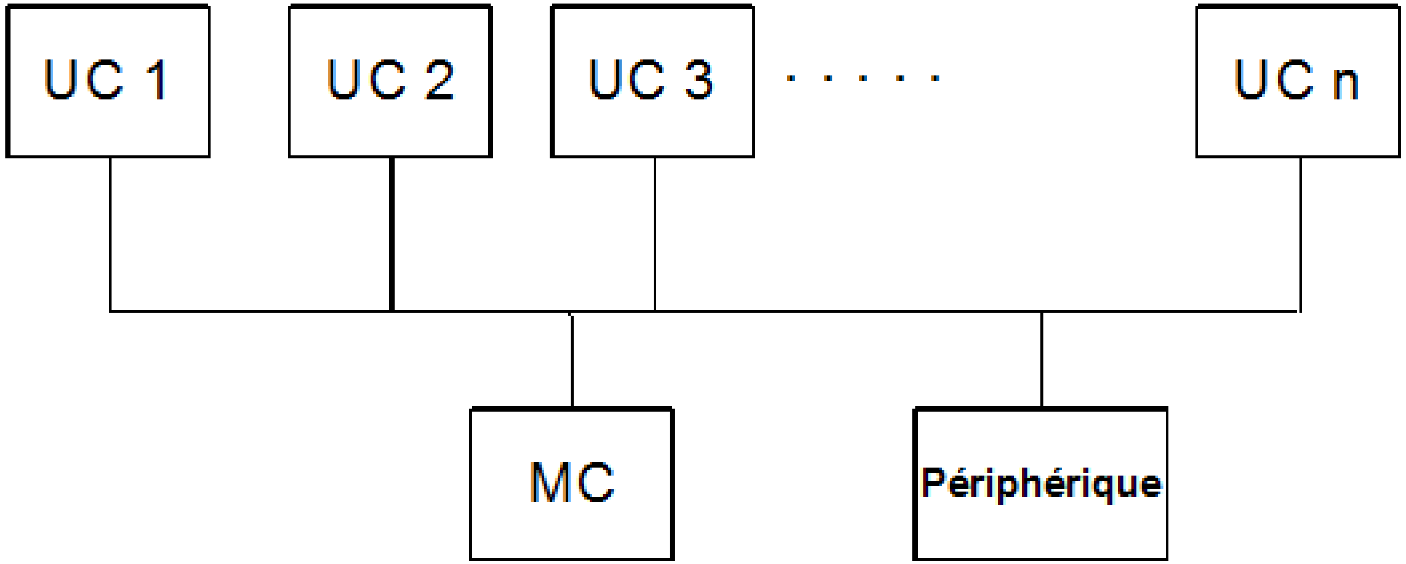
Mobile Operating Systems

- Definition: OS designed for mobile devices (e.g., smartphones, tablets).
- Example: Android, iOS.
- Pros: Portability, touch-friendly interfaces.
- Cons: Limited hardware resources, security challenges.

Multiprogramming systems



Parallel systems



Personal computer systems

Embedded systems

OS Structures

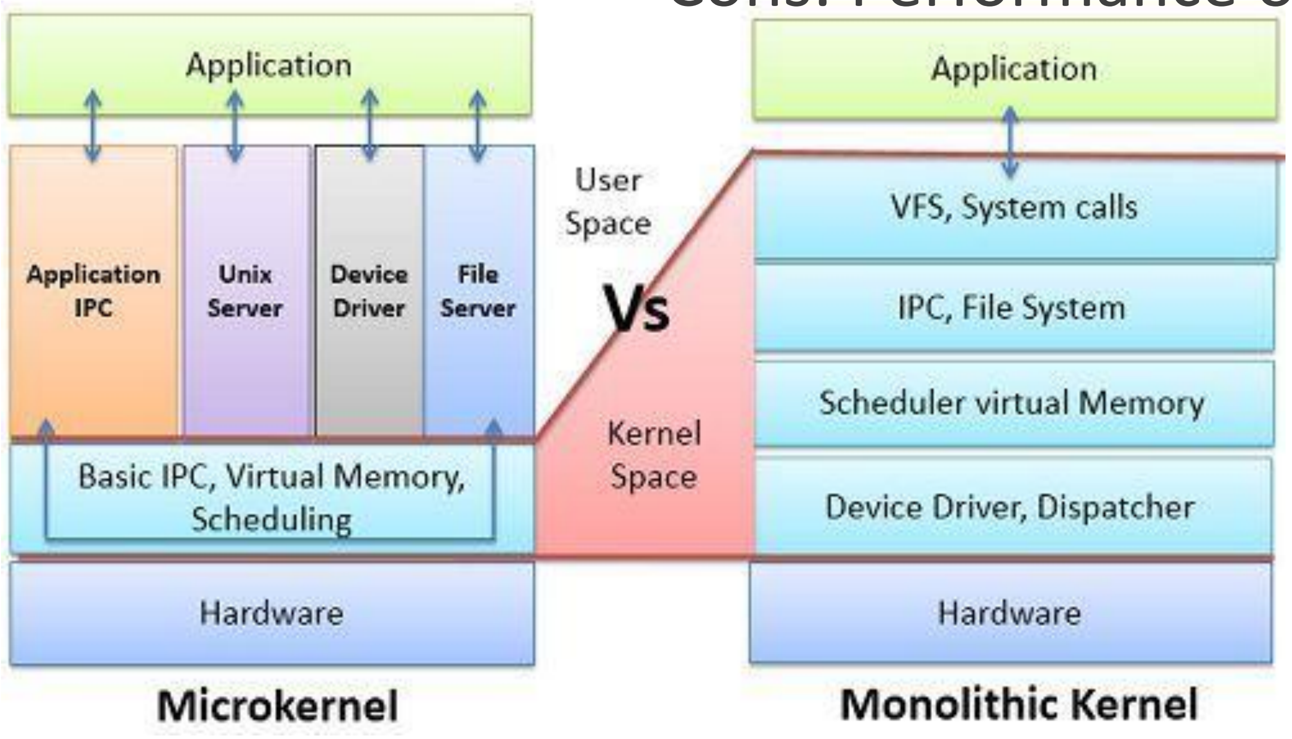
- Monolithic kernel
- Microkernel
- Layered architecture
- Modular architecture

Monolithic Kernel

- Definition: All OS services run in kernel space.
- Example: Linux.
- Pros: High performance.
- Cons: Less modular, harder to maintain.

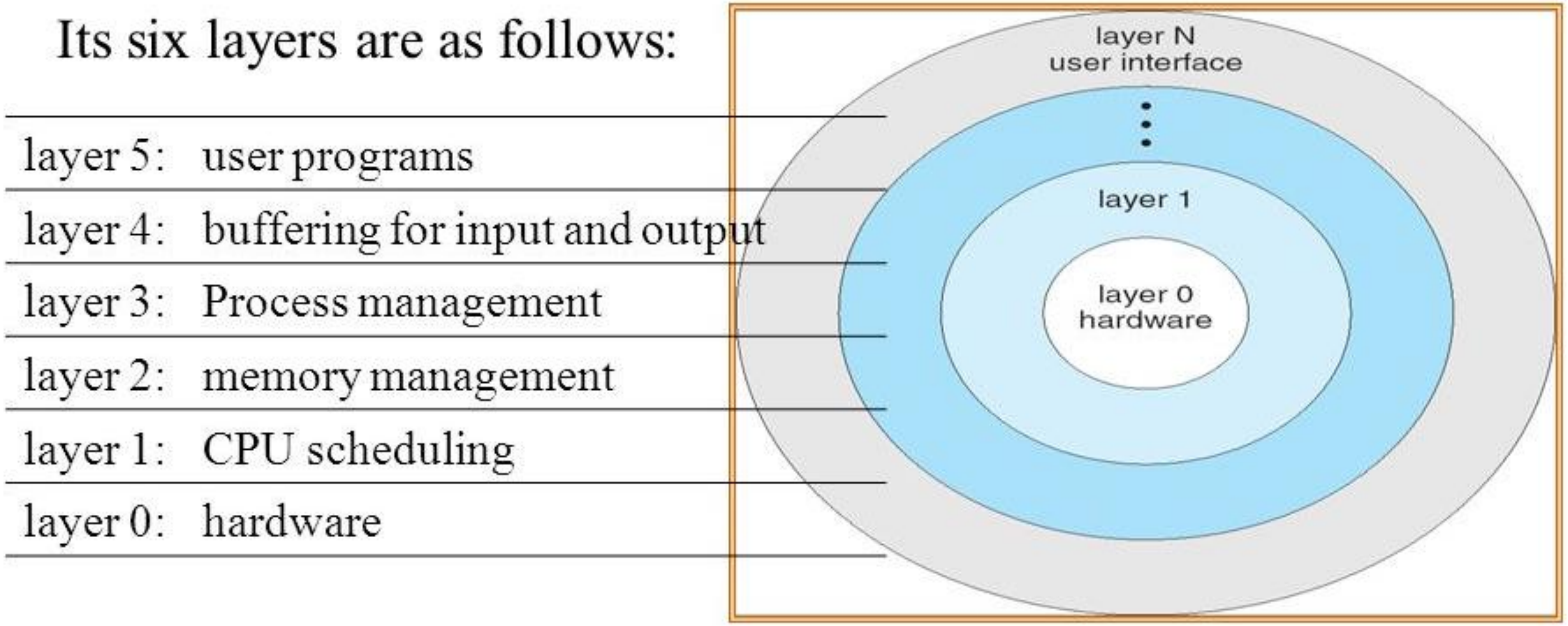
Microkernel

- Definition: Minimal kernel with most services running in user space.
- Example: macOS (based on Mach kernel).
- Pros: Modular, easier to maintain.
- Cons: Performance overhead.



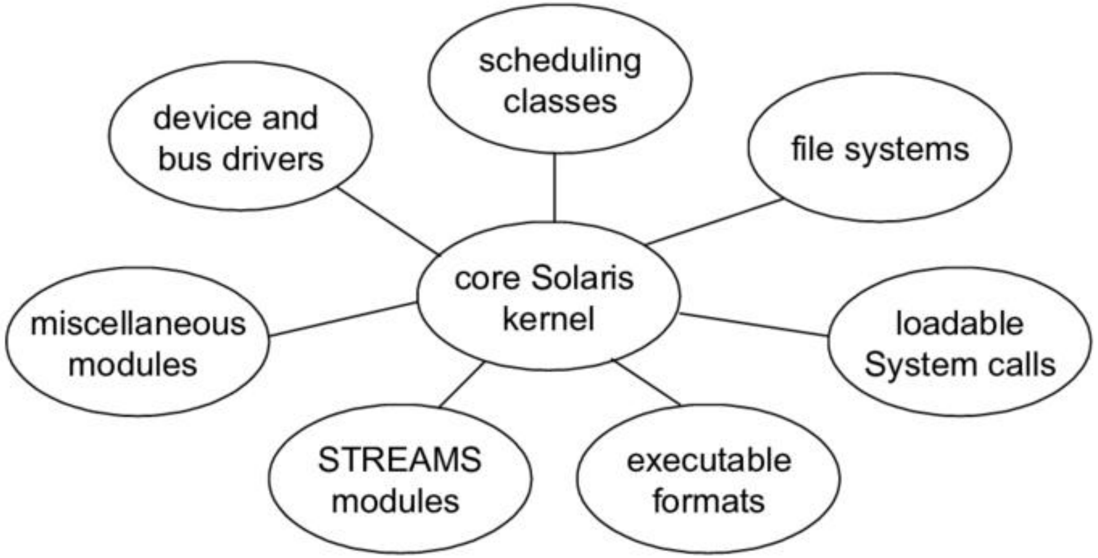
Layered Architecture

- Definition: OS is divided into layers, each with specific functionality.
- Pros: Easy to debug and maintain.
- Cons: Performance overhead due to layer interactions.



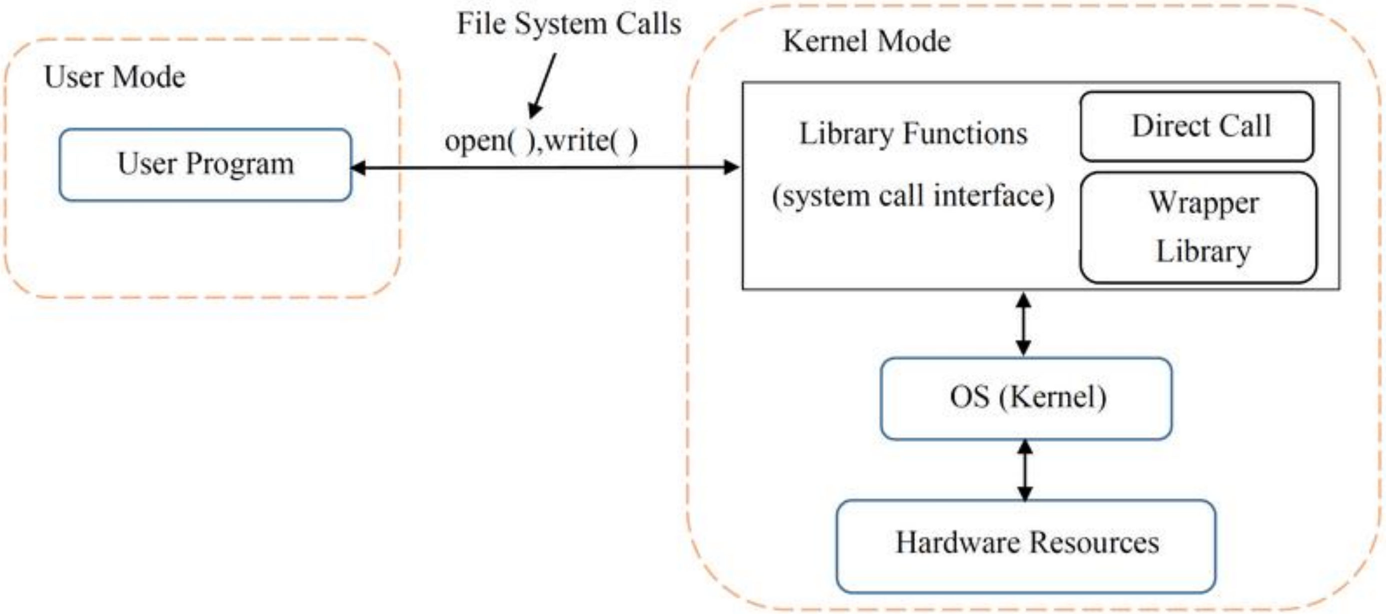
Modular Architecture

- Definition: OS is built as a set of modules that can be loaded dynamically.
- Example: Modern Linux kernels.
- Pros: Flexible, easy to extend.
- Cons: Complexity in module management.



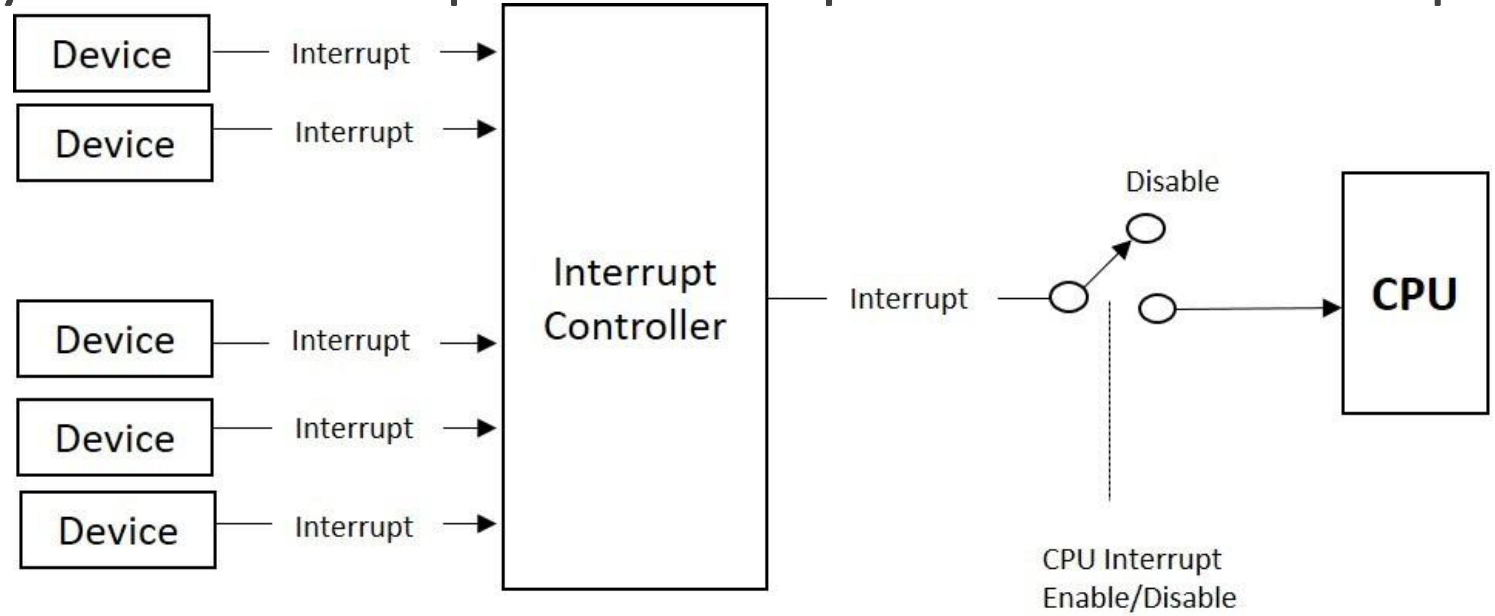
System Calls

- Definition: Interface between user programs and the OS.
- Examples: File operations (open, read, write), process control (fork, exec).
- How they work: User program → System call → Kernel → Hardware.



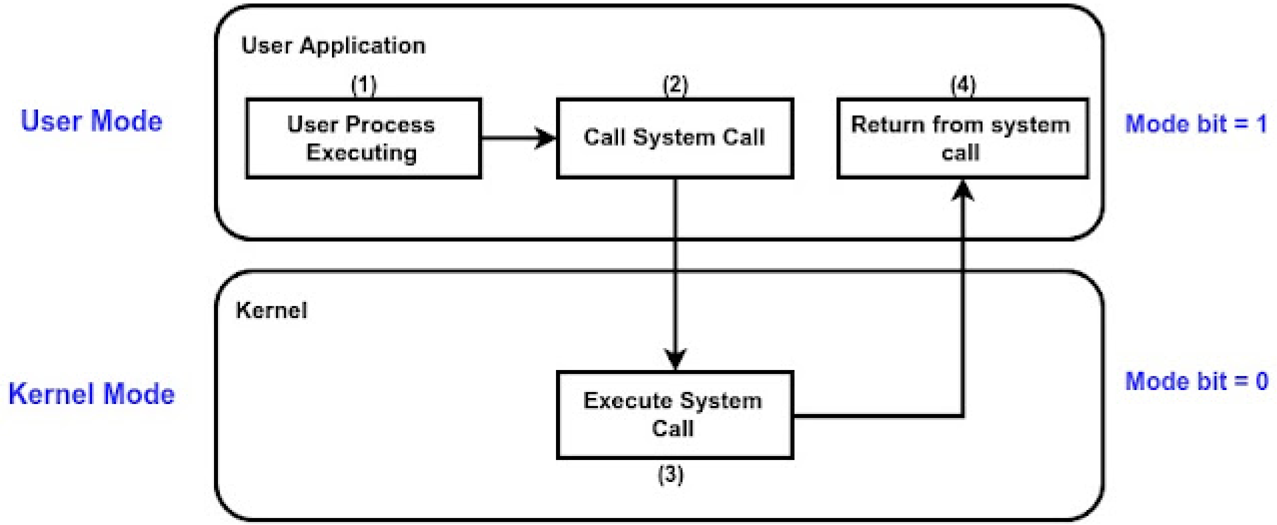
Interrupts

- Definition: Signals from hardware or software to gain the OS's attention.
- Types: Hardware interrupts (e.g., keyboard input), software interrupts (e.g., system calls).
- How they work: Interrupt → Interrupt handler → OS response.

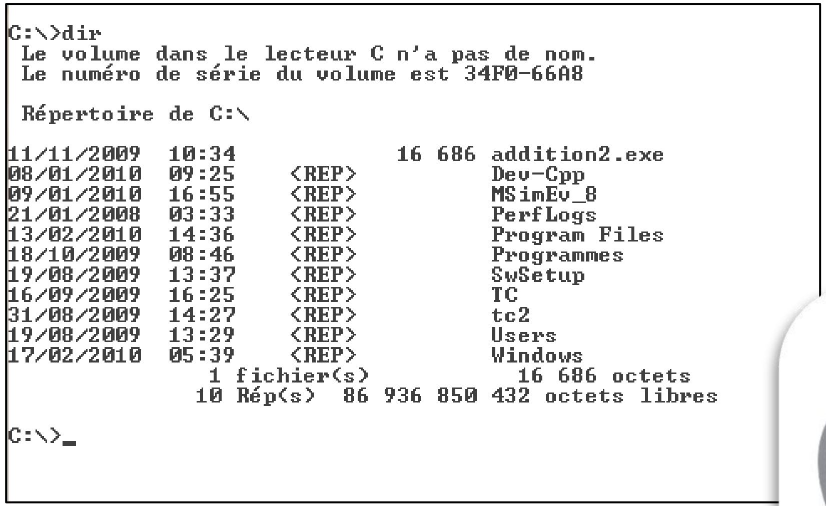


System Calls and Interrupts Together

- How system calls and interrupts work together to manage resources.
- Example: A user program requests a file read (system call), and the disk sends an interrupt when the data is ready.



Ms-DOS



WINDOWS

iOS

ANDROID

Linux

MacOS

Summary

- OS manages hardware and software resources.
- Types: Batch, time-sharing, real-time, distributed, mobile.
- Structures: Monolithic, microkernel, layered, modular.
- System calls and interrupts enable OS functionality.