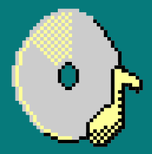
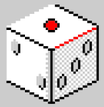


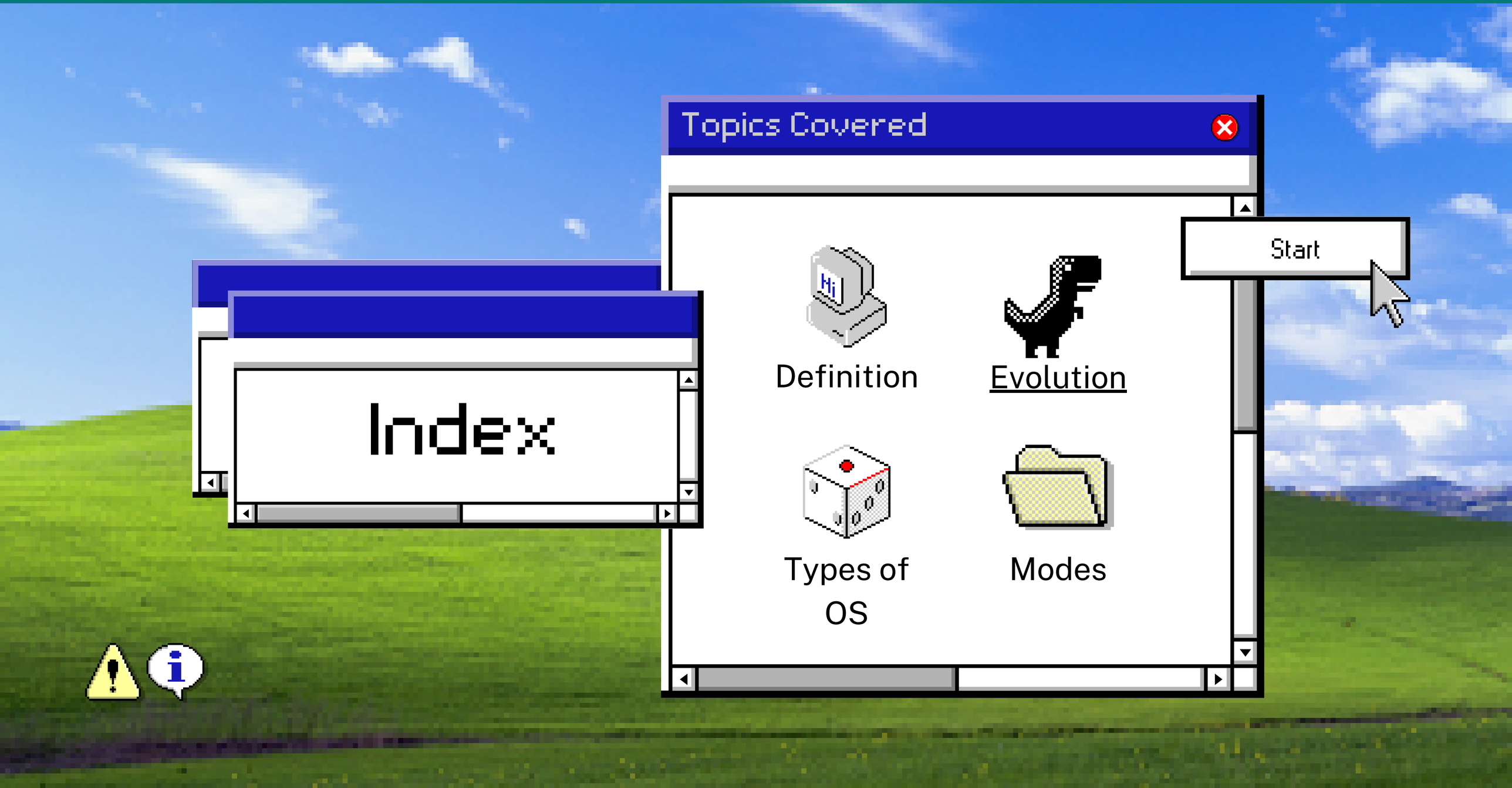

Introduction to Operating Systems

presentation by group 8



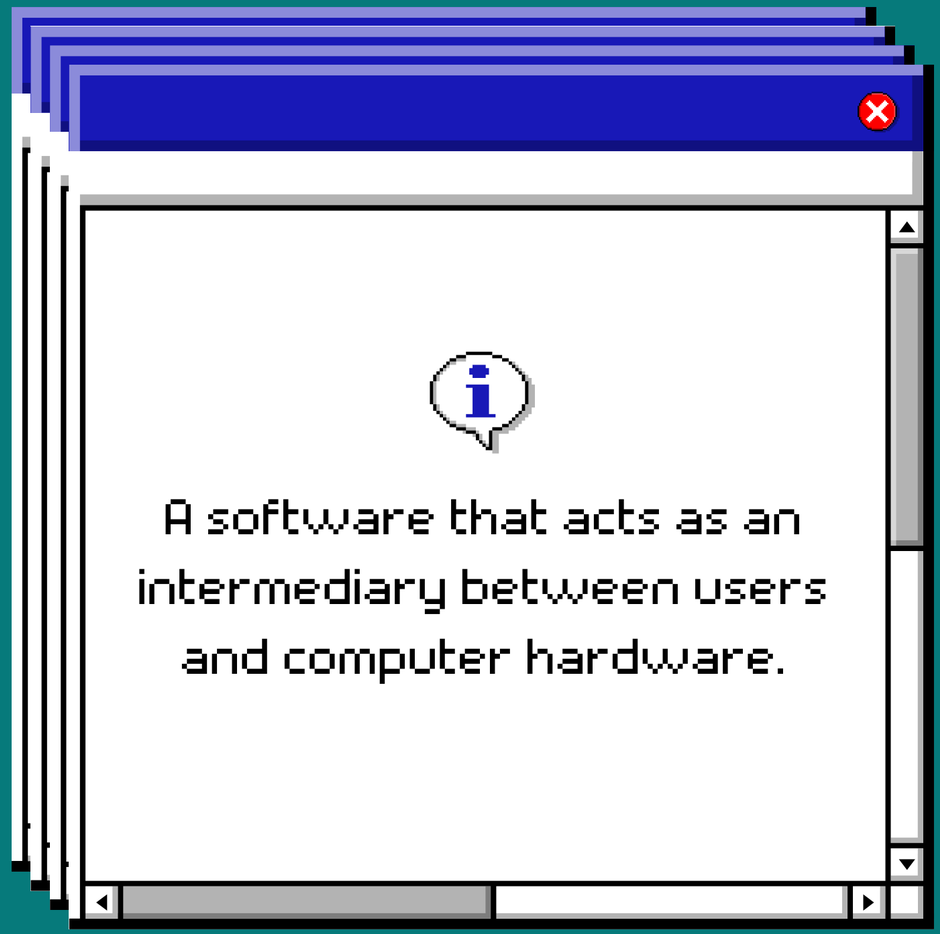



11:30 AM
14th Sept, 2024


Topics Covered
Index
Definition
Evolution
Types of OS
Modes
Start

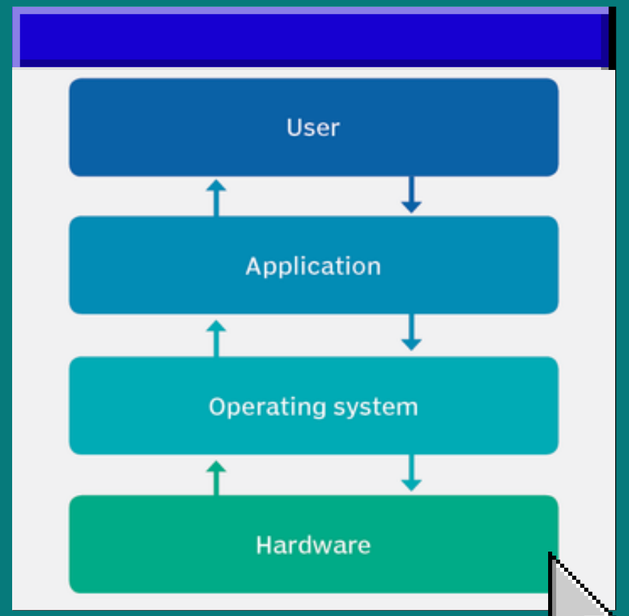
What is an Operating System?

An operating system (OS) is the program that, after being initially loaded into the computer by a boot program, manages all of the other application programs in a computer.



Back to index Page

An operating system brings powerful benefits to computer software and software development. Without an operating system, every application would need to include its own UI, as well as the comprehensive code needed to handle all low-level functionality of the underlying computer, such as disk storage, network interfaces and so on.



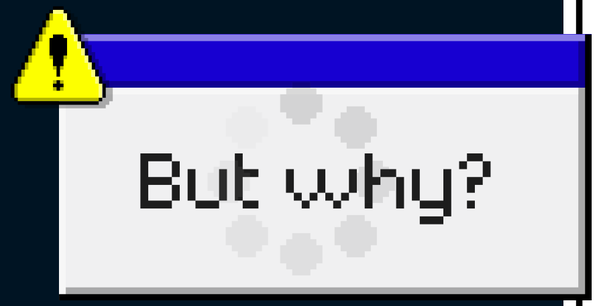
Operating System placement

Back to index Page

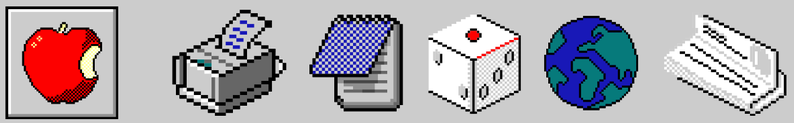
1: zsh

```
~
❯ print "Operating System"
Operating System

~
❯ clear
```



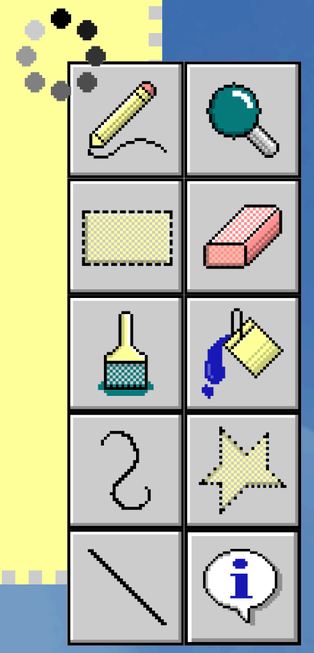
Without an operating system, every application would need to include its own UI, as well as the comprehensive code needed to handle all low-level functionality of the underlying computer, such as disk storage, network interfaces and so on.



Back to index Page

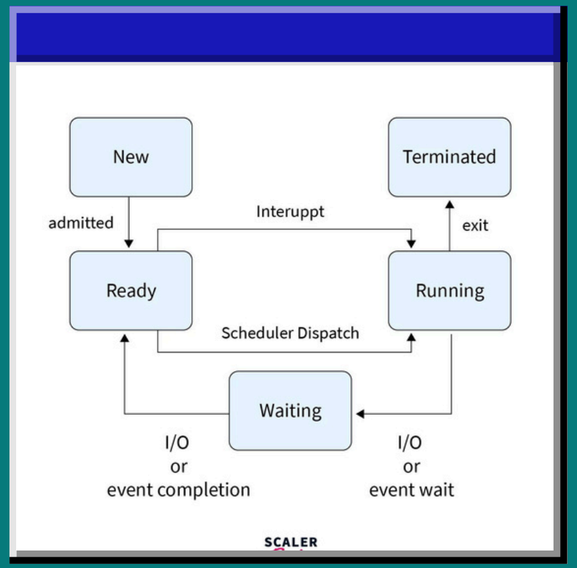
Functions of an Operating System

Back to Index Page

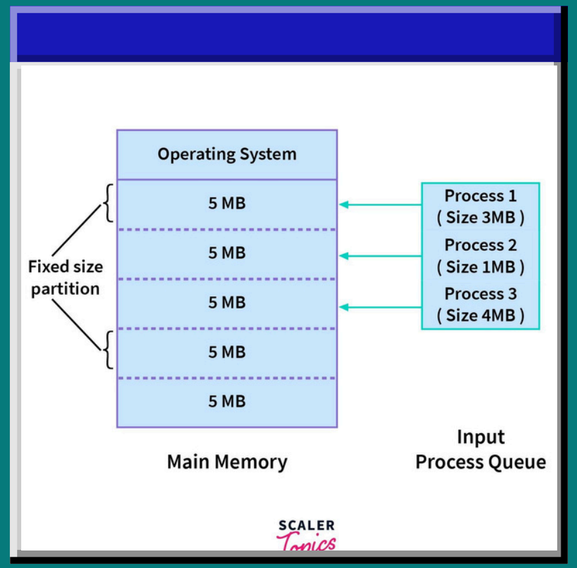


Functions of an OS

The OS manages processes by scheduling, creating, and terminating them. It allocates CPU time, handles synchronization and communication between processes, and manages resources. The OS also oversees process states (running, waiting, etc.) and ensures efficient multitasking.



Process Management



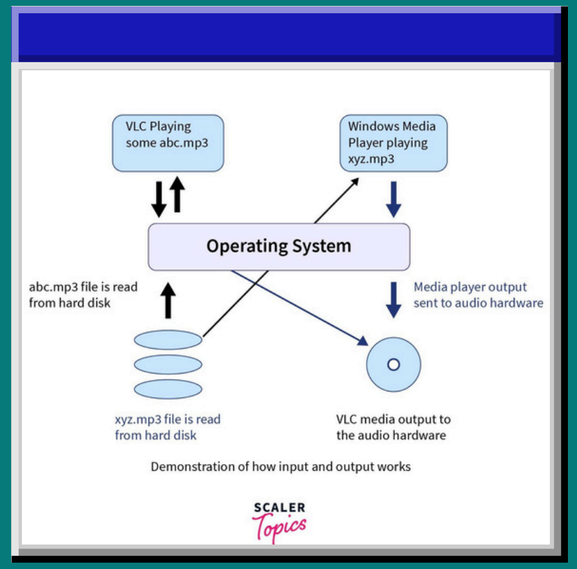
Memory Management

The OS manages memory through allocation, tracking, and optimization. It assigns memory to processes, ensuring efficient use through paging, segmentation, and virtual memory. The OS also handles memory protection, preventing processes from accessing each other's memory, and manages swapping to avoid memory overflow.

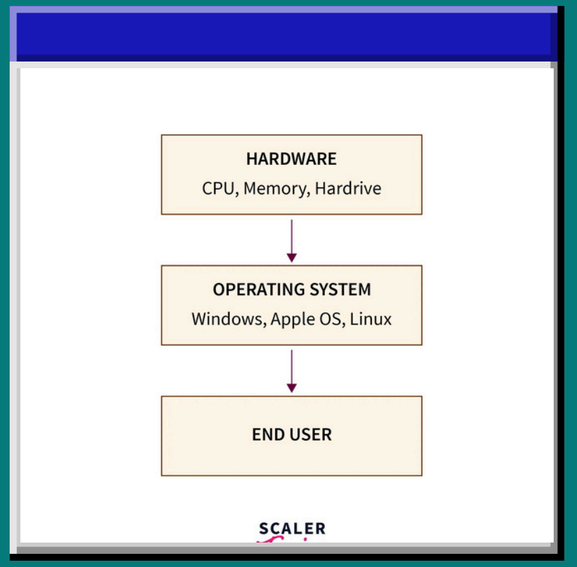
Back to Agenda Page

Functions of an OS

The OS manages devices by controlling hardware through drivers, facilitating communication between devices and applications. It handles device allocation, monitors status, manages input/output operations, and ensures efficient data transfer, enabling smooth interaction between devices and the system.



Device Management



File System Management

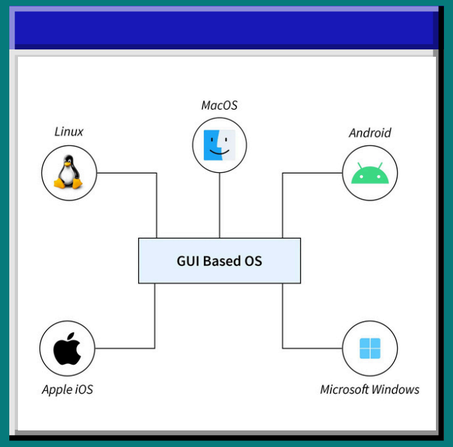
The OS manages the file system by organizing, storing, and retrieving data. It handles file creation, deletion, and access control, maintains directories, ensures data integrity, and manages permissions. It also allocates storage space and optimizes disk usage.

Back to Agenda Page

Functions of an OS

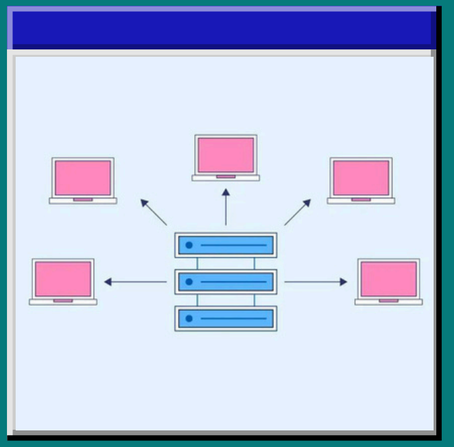
Security and Access Control

Protects the system from unauthorized access by implementing user authentication and permissions.



User Interface (UI)

Provides a way for users to interact with the system (e.g., Graphical User Interface or Command-Line Interface).

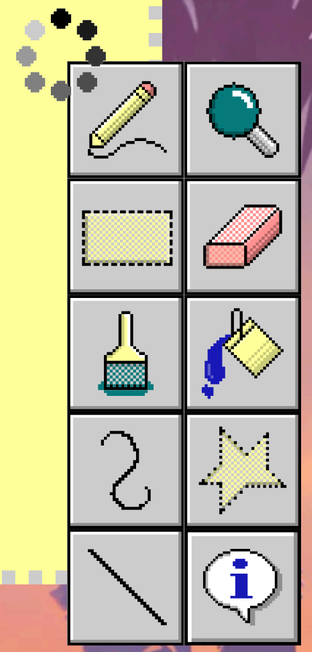


Networking

Facilitates communication between devices over a network.

Back to Agenda Page

Operating System as a Resource Manager

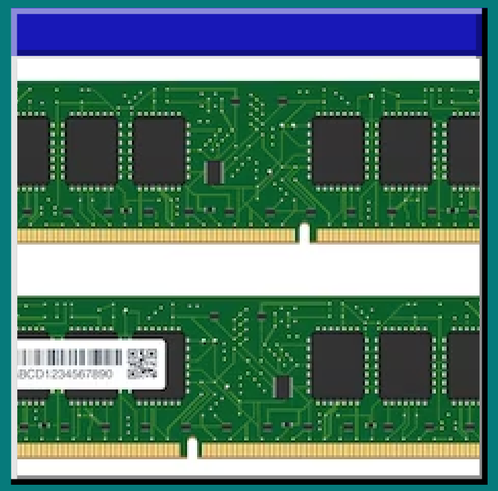


Back to Agenda Page

Resource Allocation and Management

CPU Scheduling

The OS schedules processes using algorithms like Round Robin or Priority Scheduling.



Memory Management

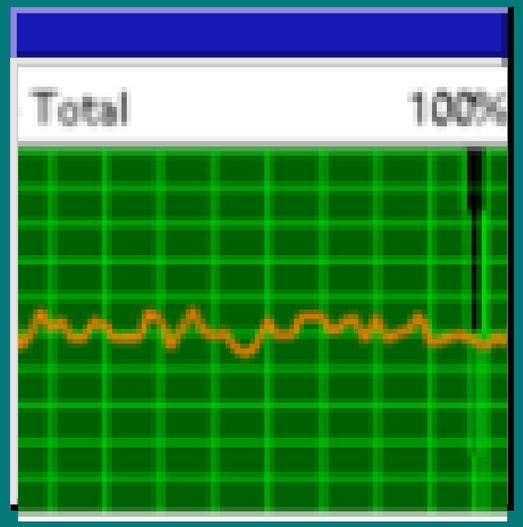
OS handles memory allocation using paging or segmentation.

Storage Management

Manages file systems, ensures secure access to files, and handles data storage.

Back to Agenda Page

OS as a Resource Manager -

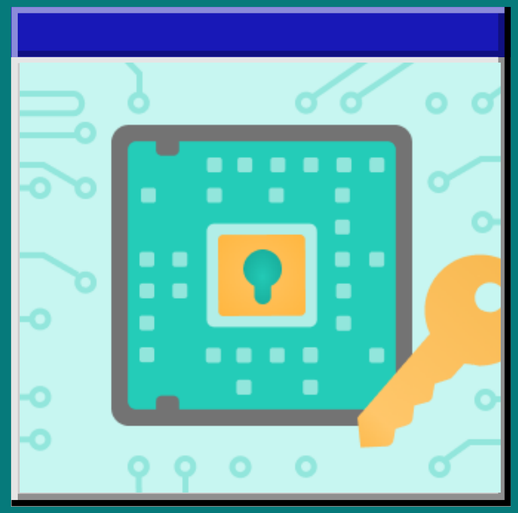


Maximizes Resource Utilization:

Ensures all hardware resources are used effectively, minimizing idle time.

Fair Resource Distribution:

Provides fair access to resources for multiple processes, avoiding starvation and deadlocks.



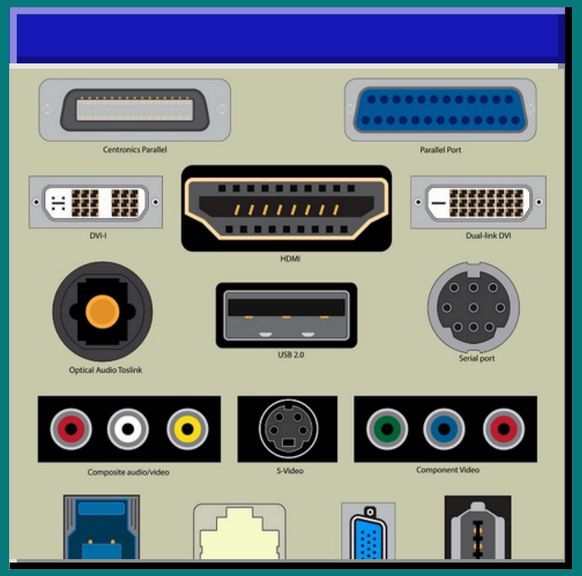
Security and Protection:

Protects processes from unauthorized access and ensures secure data management.

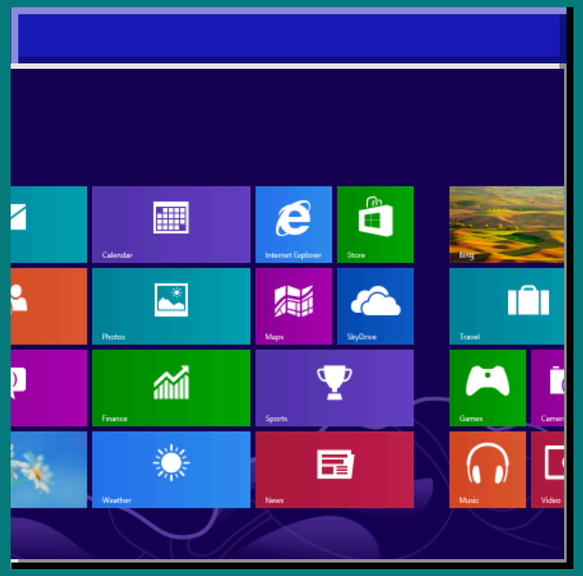
Back to Agenda Page

OS as a Resource Manager -

The OS manages I/O devices by allocating resources, controlling data flow, and ensuring efficient, synchronized communication between hardware and software for seamless input/output operations.



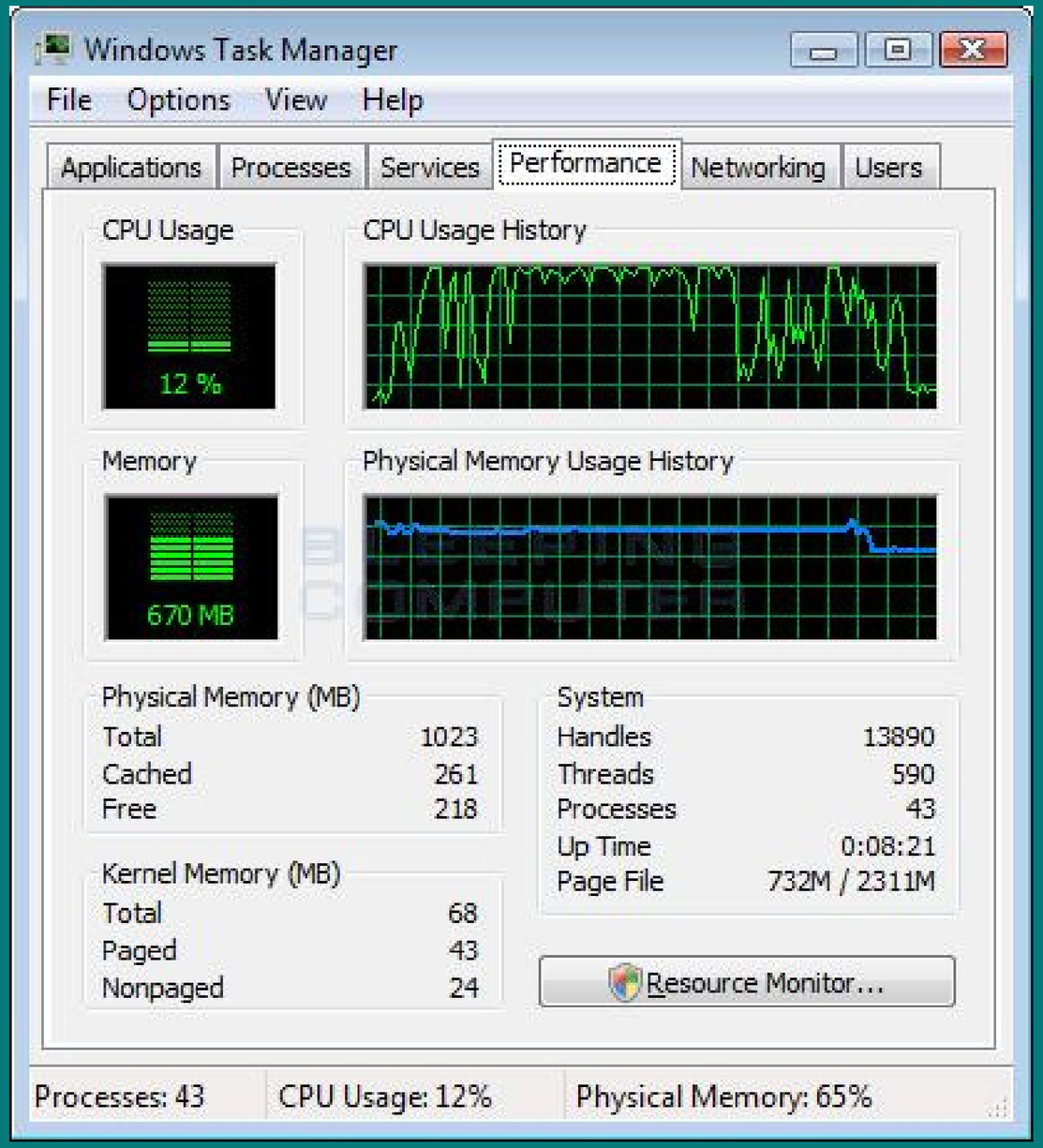
I/O Management



Abstracts Hardware Complexity:

The OS abstracts hardware complexity by providing a user-friendly interface, managing hardware resources, and handling device interactions, enabling applications to access hardware without low-level management.

Back to Agenda Page

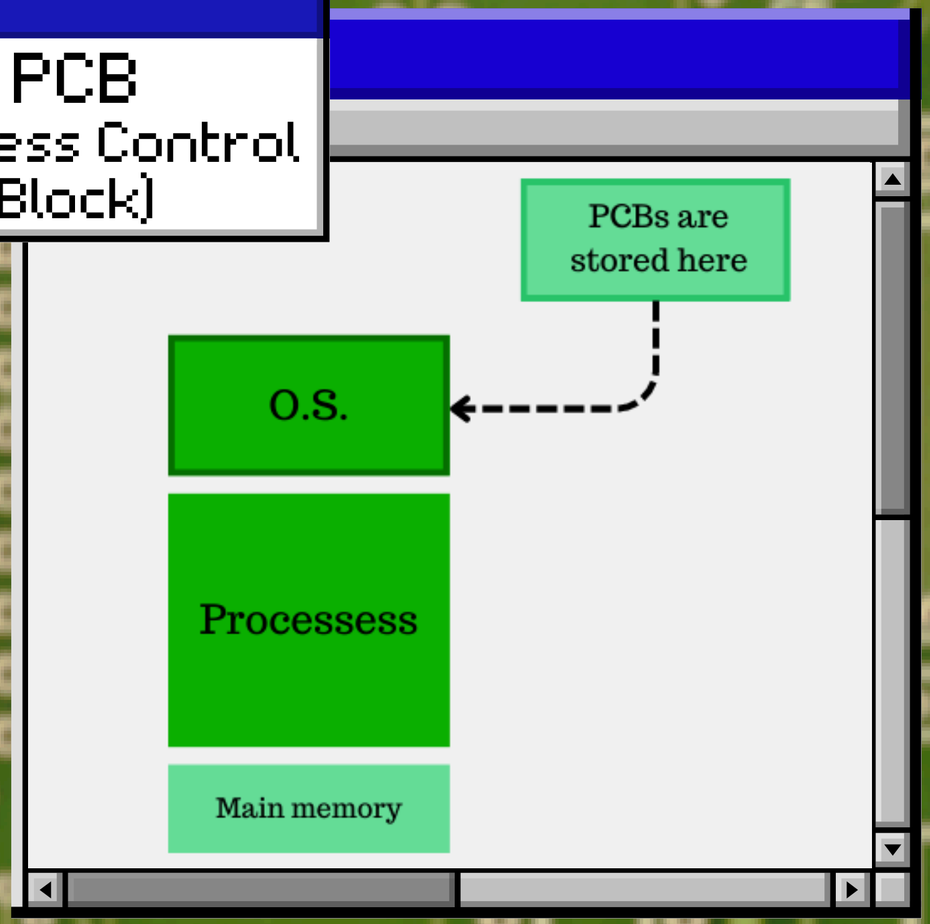


OS as a Resource Manager - Task Manager Example

As you can see here, the windows 7 Task Managaer provides a nice GUI to check out all kinds of system properties like Applications running, Processes, Background Services, Performance and Utilization of system resources (RAM, CPU usage etc...), Networking and Users

Back to Agenda Page

PCB (Process Control Block)



The OS maintains a PCB for each process, storing data like process state, program counter, CPU registers, memory, and I/O status.

Evolution of Operating Systems

Back to Agenda Page

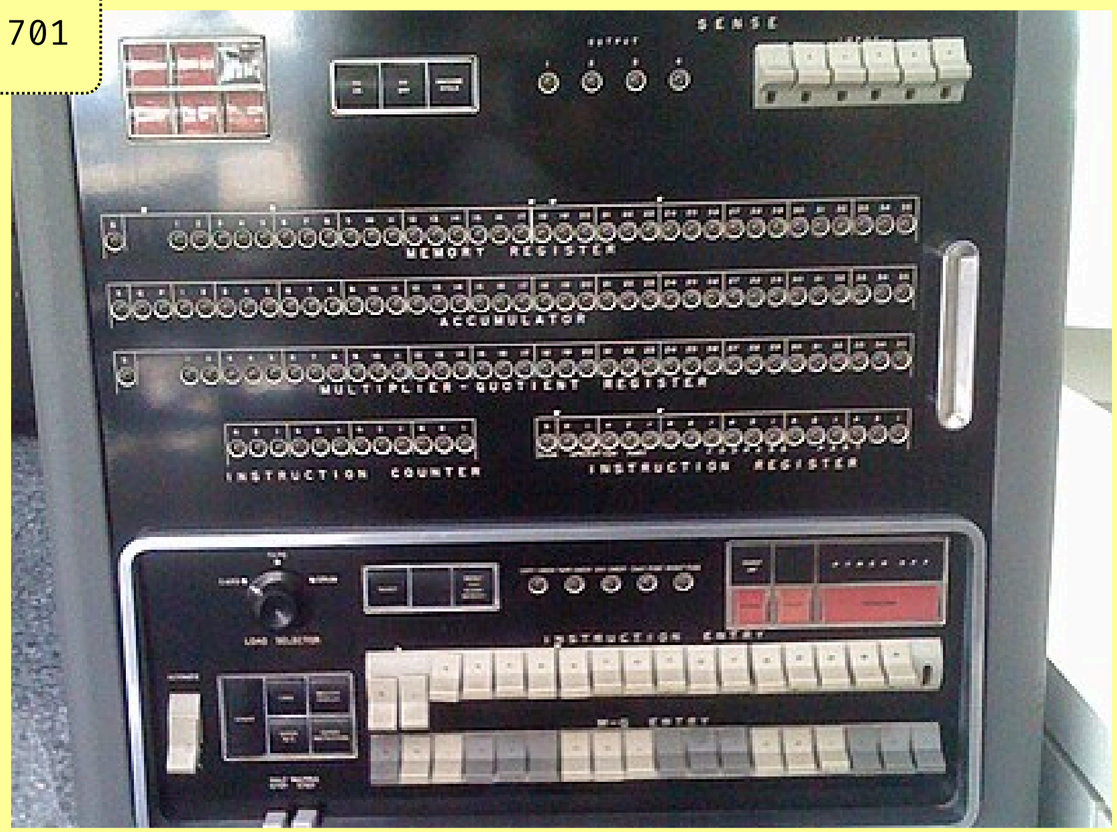
1940s - 50s

First Generation

The first electronic computer came in 1940 to replace the mechanical computer. It **did not have any OS** and was used to do simple calculations using plug-boards.

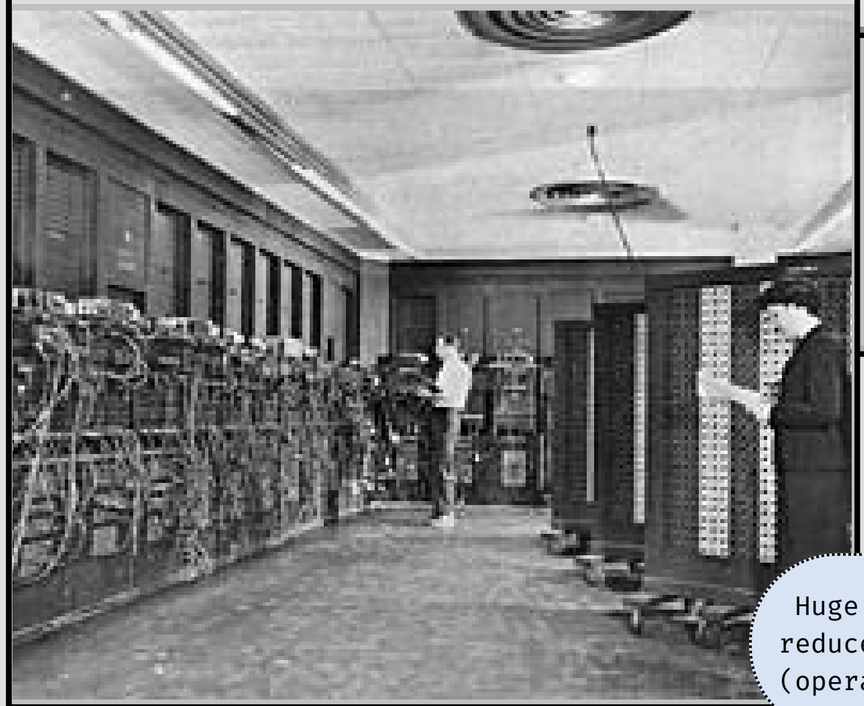
For more than a decade, the computing systems developed did not support any OS and the users had to manually feed the program and the data. This period is also called the **Open Shop phase**. Each user was allotted a fixed amount of time to use the computer. **Time for setting up was much higher than actual time of computation**. The exercise was extremely laborious and error prone. IBM 701 machine is an example of an Open Shop machine.

IBM 701



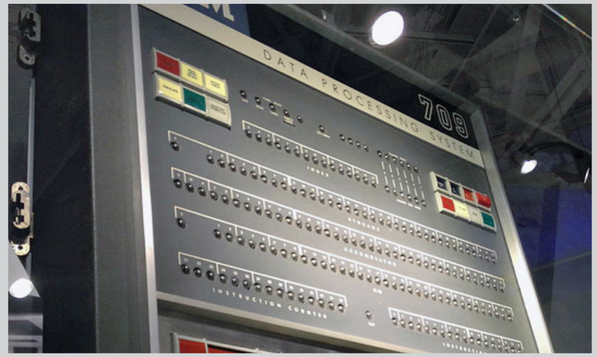
Second Generation

1955 - 1965



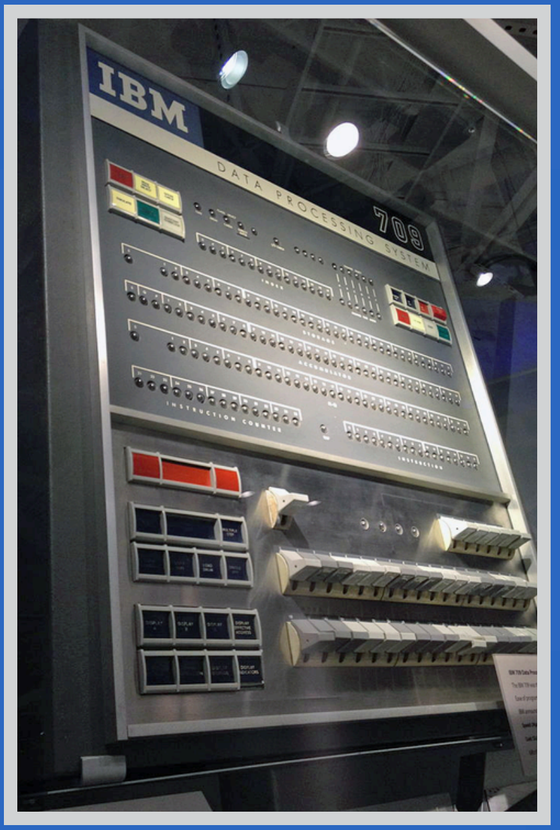
Huge wastage of users' time was drastically reduced by appointing skilled professionals (operators) in the computer room.

Even though much improved over the Open shop era, it was still plagued by slow speed of I/O devices. The SHARE operating system for the **IBM 709 (shown below)** was an early batch processing system.



The users were asked to prepare their 'job' (punched cards of programs and data) and submit to the operators who run the jobs on users' behalf. Similar jobs (based on requirement of similar set of resources) were put in a batch and jobs scheduled for execution sequentially one after another batch-wise. This phase is also known as **Closed Shop** (as users were not allowed in the computer room) or **batch processing**

What could be improved?



The Batch processing automated the task of making manually plugging in wires on the plug-board.
However, as computers became more popular, it became cumbersome to PROGRAM specific instructions in Assembly for each new I/O device introduced in the market.

The idea of “Device Drivers” arose out of this!

Let's see what improved in the Third Generation!

When one program went for an I/O operation (taking input or displaying / printing output), the CPU was idle

CPU idling due to I/O

Sequential execution in batch processing forced the computer processor to stall or remain idle during I/O operation leading to poor utilization of CPU time.

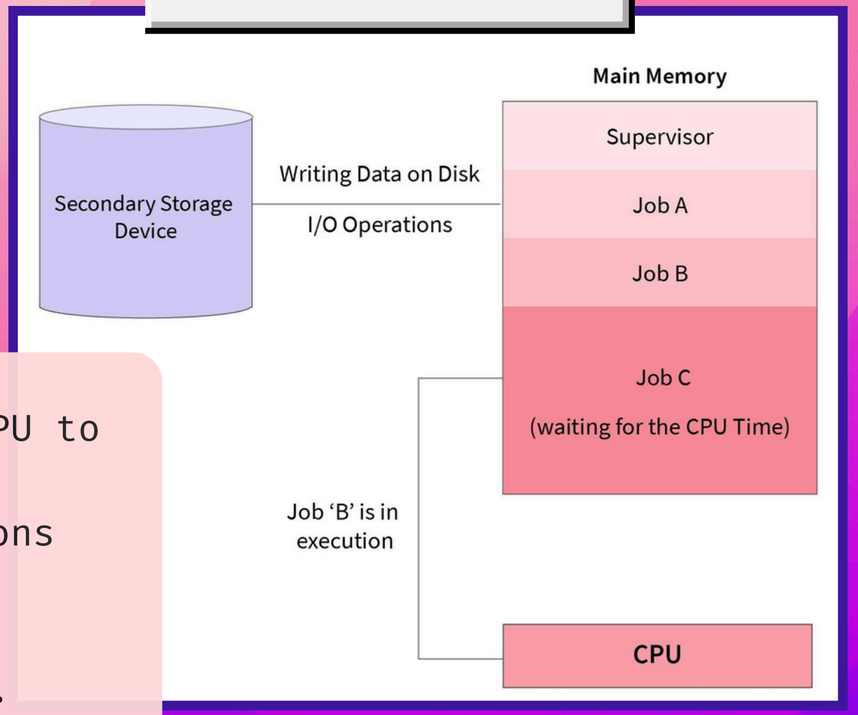
The Third Generation

1965 - 1980s

Key Technological Advancements include:

- Large core memories
- secondary storage with random access
- data channels and hardware interrupts.

Multi-programming

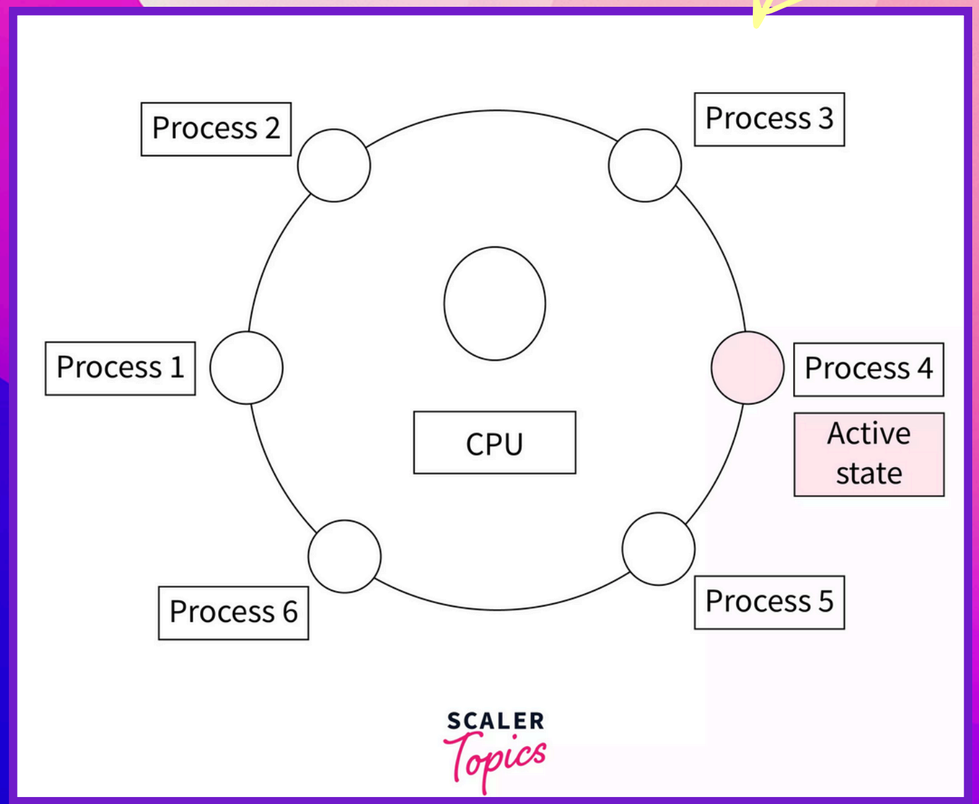


- Multiple programs loaded in memory, with interrupts allowing CPU to switch between tasks.
- CPU worked on one program while another waited for I/O operations (Simultaneous Peripheral Operation Online or Spooling).
- Examples: Atlas Supervisor, Burroughs B5000, Exec II systems.
- CPU utilization improved, but lacked interactive user response.

Time-sharing

Time-Sharing Systems:

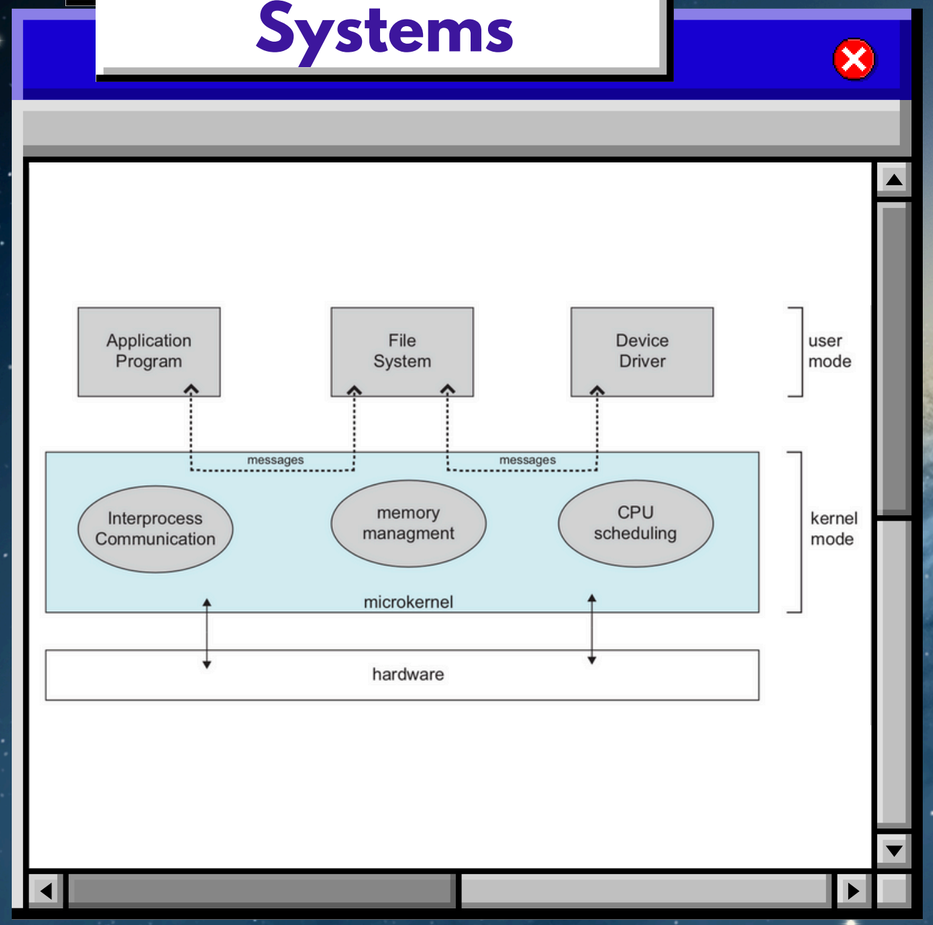
- Enabled interaction between users and the system via multiple terminals.
- Users experienced the illusion of a dedicated processor due to rapid task switching.
- Examples: CTSS (Compatible Time Sharing System), MULTICS (Multiplexed Information and Computing Service).
- Introduced file systems, protection, and passwords.



Concurrent Programming:

- Solved issues like deadlocks that arose from multiprogramming and time-sharing.
- Introduced synchronization tools like semaphores and monitors.
- Examples: THE OS (Edger Djikstra, 1968), RC 4000 (P. Brinch Hansen, 1970).

Modern Operating Systems



The Fourth Generation

1 9 8 0 s - P r e s e n t

From Time-Sharing to Personal Computing:

- Advances in microprocessors and semiconductor memories during the 1970s made computing affordable for individuals.
- Shifted focus from resource efficiency (multi-user) to user convenience (single-user environment).
- Personal Computing Era:
- The rise of single-user operating systems.
- Introduction of Graphical User Interface (GUI) for easier interaction through mouse clicks.
- Examples:
- MS-DOS, Windows 95, Macintosh.

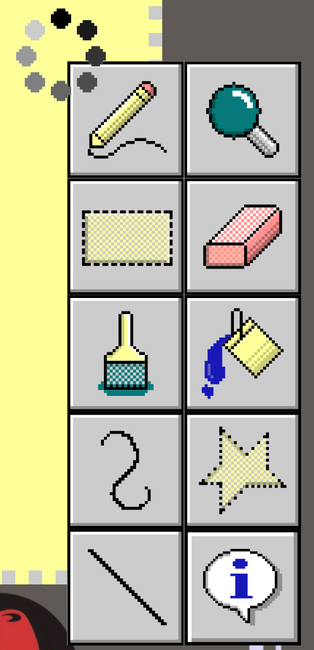
Key Features:

- Focus on user experience, accessibility, and ease of use.
- Enhanced multitasking, seamless user interaction with sophisticated interfaces.
- Technological Advancements like **Powerful processors**, memory management, and development of GUIs.
- Personal computing became mainstream, shaping the technology landscape.
- Evolution of operating systems with continuous updates and improvements.

A few prominent modern day Operating Systems:

Types of Operating Systems

Back to Agenda Page

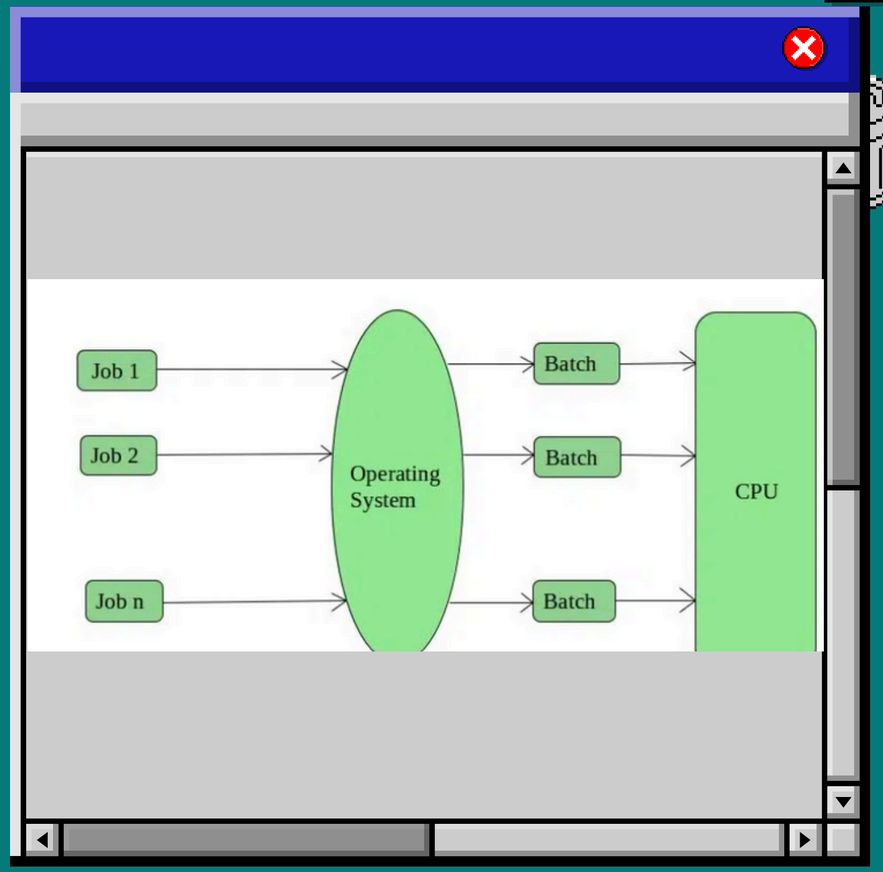


redhat.

Batch Operating Systems

Batch processing systems, dating back to the 1950s, handle periodic tasks such as payroll and billing without requiring user intervention. These tasks are queued for automatic execution, with results reviewed later if needed. Modern operating systems, like IBM's MVS, initially focused on batch processing with interfaces like job control language (JCL). Today, even interactive sessions can submit these tasks for processing.

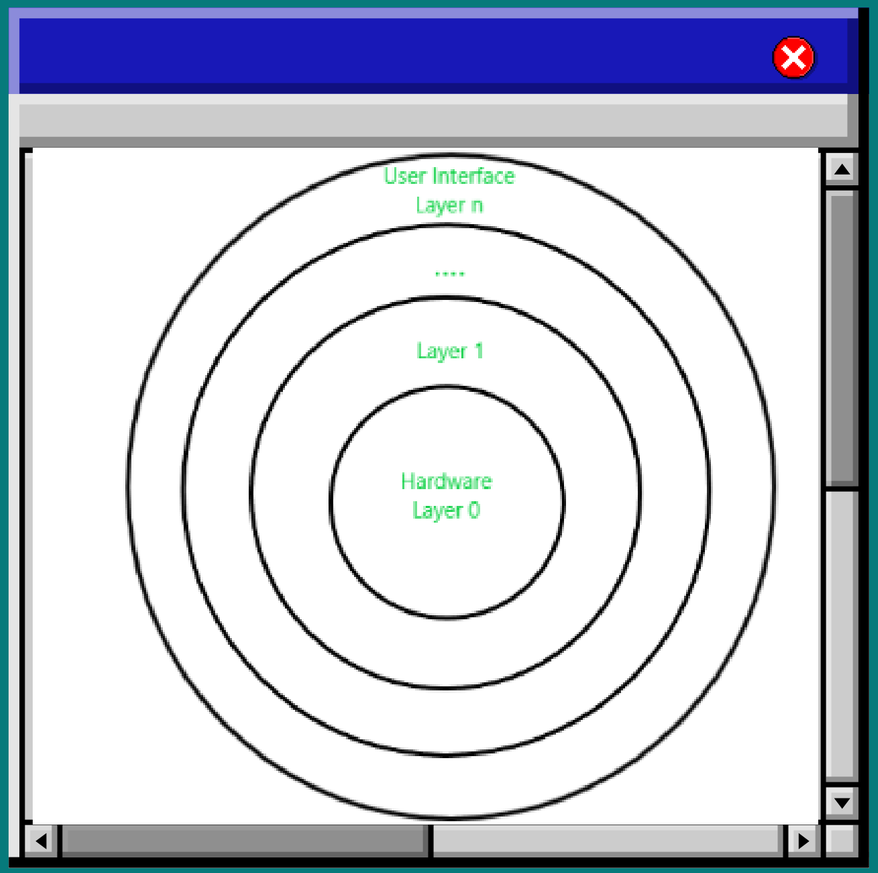
Batch OS processes batches of tasks with minimal user interaction. It groups similar jobs together, executing them in sequence without needing constant supervision.



Back to Agenda Page

Layered Operating Systems

A layered operating system is structured into multiple layers, each providing services to the one above it, ensuring modularity and simplicity. Each layer handles a specific function, such as process management or device handling. Communication between layers happens in a controlled manner. This design improves maintainability, security, and system development, allowing easier debugging and testing by isolating functionalities across distinct levels.



This OS structure is organized in layers, each with a specific function.

Back to Agenda Page

Interactive Operating Systems

These systems were built for faster response times, enabling users to debug programs efficiently. Time-sharing software supported quick interactions. Early systems like IBM MVS with CICS and TSO interfaces pioneered this, and now, interactive operating systems like Windows, UNIX, and Linux dominate the market.

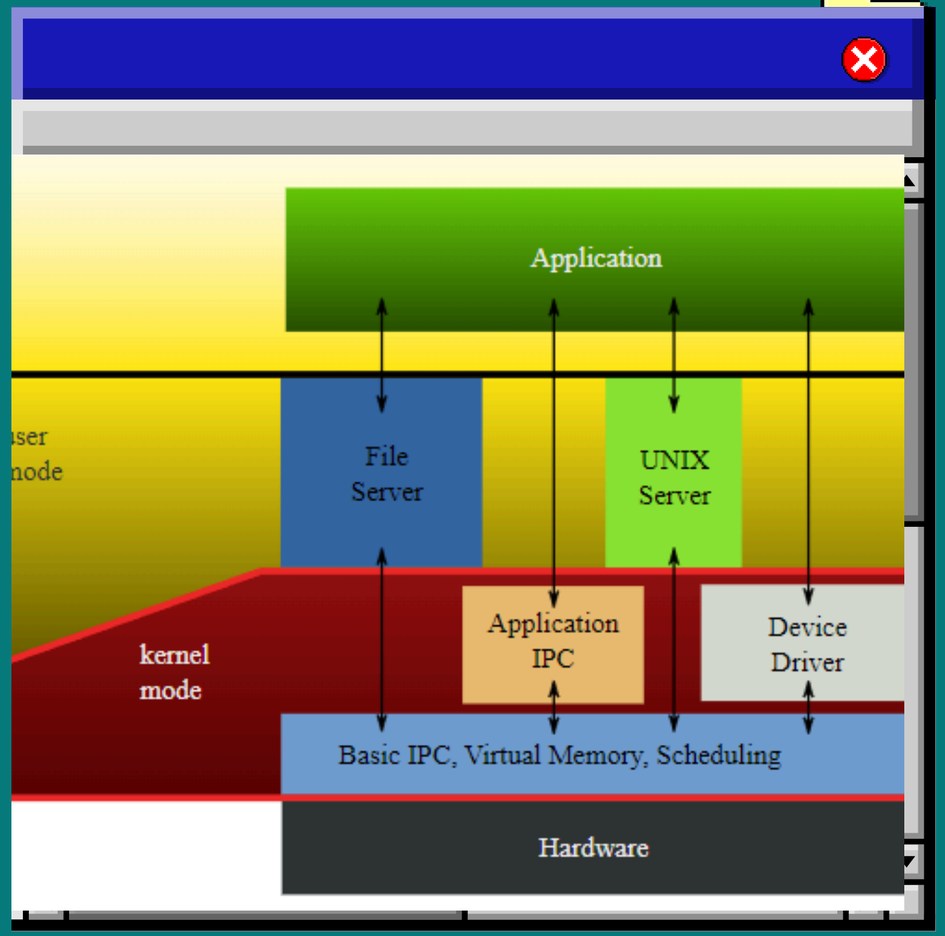
A distinctive feature of interactive systems is their real-time feedback, allowing users to instantly interact and adjust based on the system's responses.

Back to Agenda Page

Hybrid Operating Systems

Hybrid operating systems integrate features of both batch and time-sharing systems, allowing efficient resource management while enabling user interaction. They handle concurrent processes effectively, optimizing performance across various tasks. Examples include modern Windows and UNIX systems, which support multitasking and real-time user interfaces.

Hybrid systems efficiently integrate batch and interactive processing to enhance performance and resource utilization.

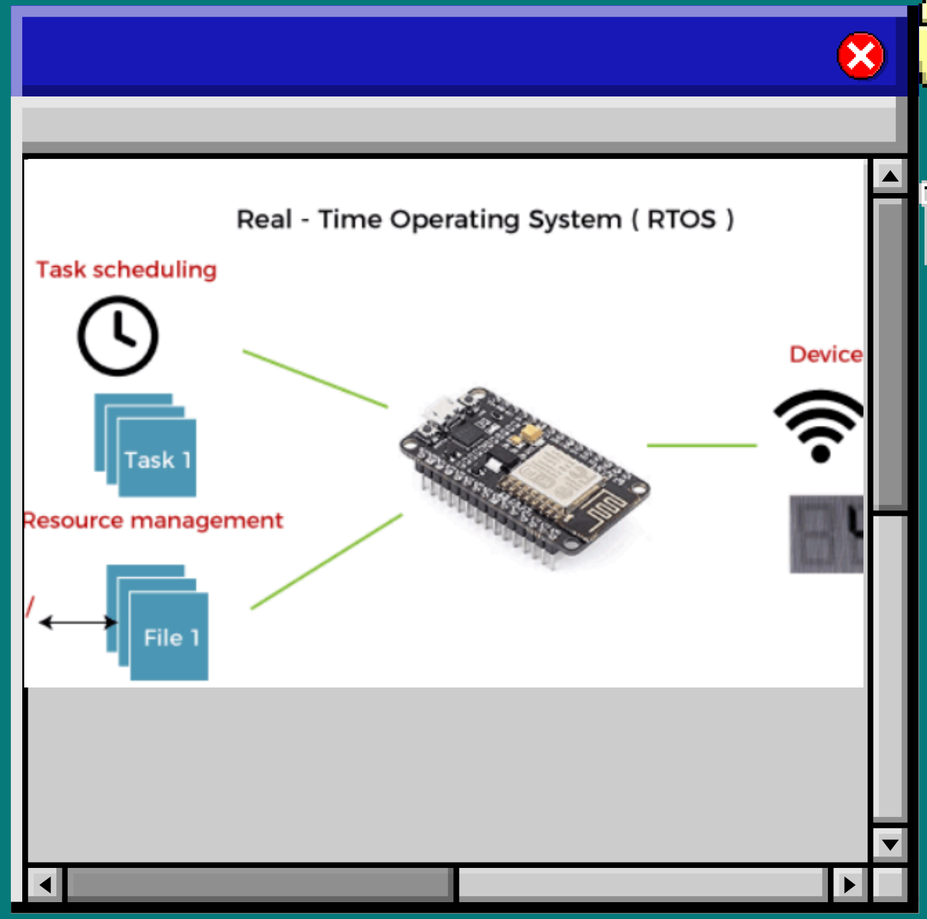


Back to Agenda Page

Real-Time Operating Systems

Real-time systems are categorized into two types: hard and soft real-time systems. Hard real-time systems ensure consistent task completion within strict time constraints, typically within milliseconds. In contrast, soft real-time systems allow more flexibility, accommodating delays of several hundred milliseconds with increased variability or "jitter" in response times.

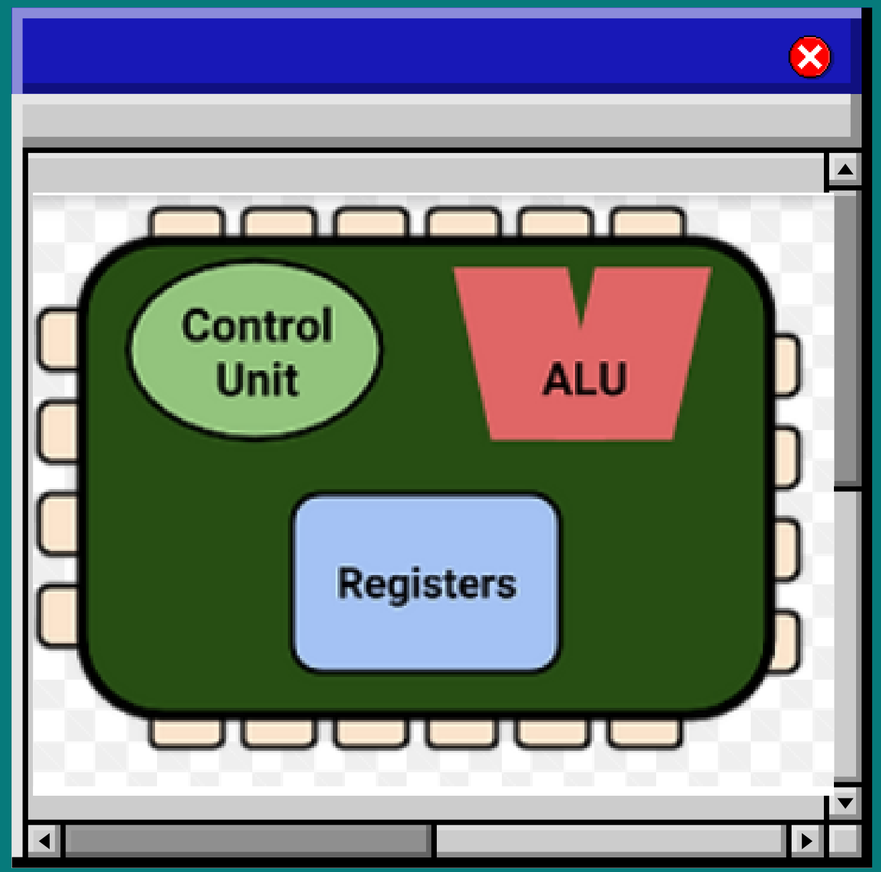
A distinctive feature of real-time operating systems is their ability to guarantee timely task completion, ensuring critical processes meet strict timing constraints.



Back to Agenda Page

Single-Processor Operating Systems

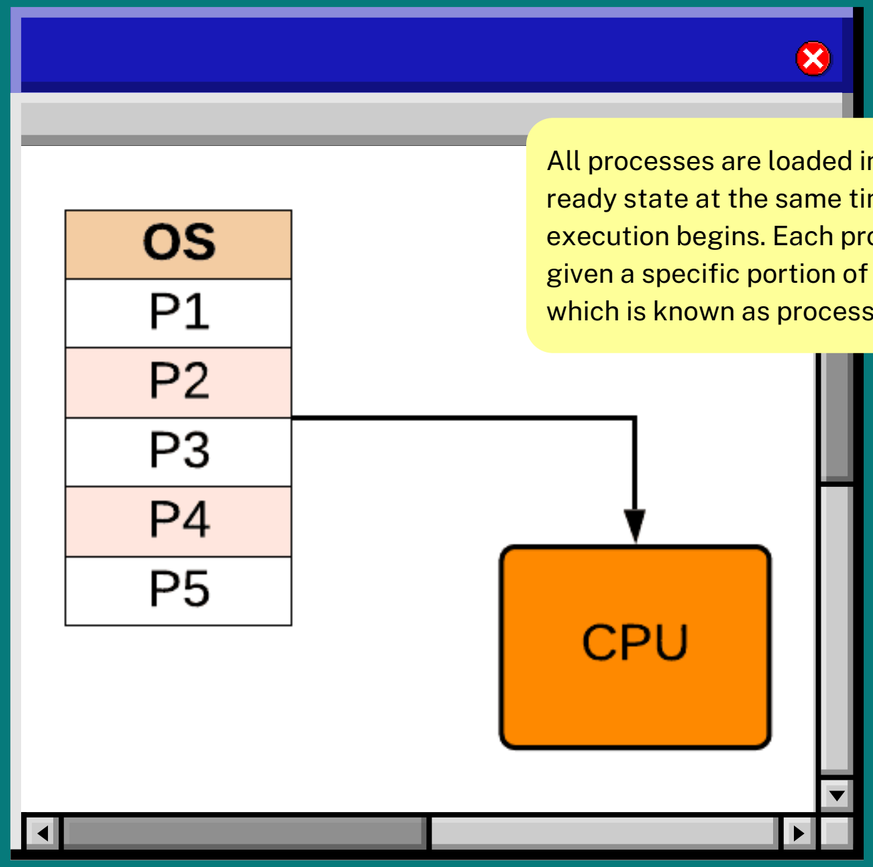
Single processor systems contain one general-purpose processor with a single core for executing instructions and handling data, supported by special-purpose processors for managing I/O devices like keyboards or disk controllers. These systems are simple in design, but most modern computers use multi-core processors, making uniprocessor systems less common. While the special-purpose processors manage I/O operations, freeing up the CPU, it lacks the efficiency of parallel processing available in multi-core systems.



Single processor systems feature a single general-purpose CPU, simplifying design but limiting multitasking capabilities and processing

Back to Agenda Page

Multiprogramming

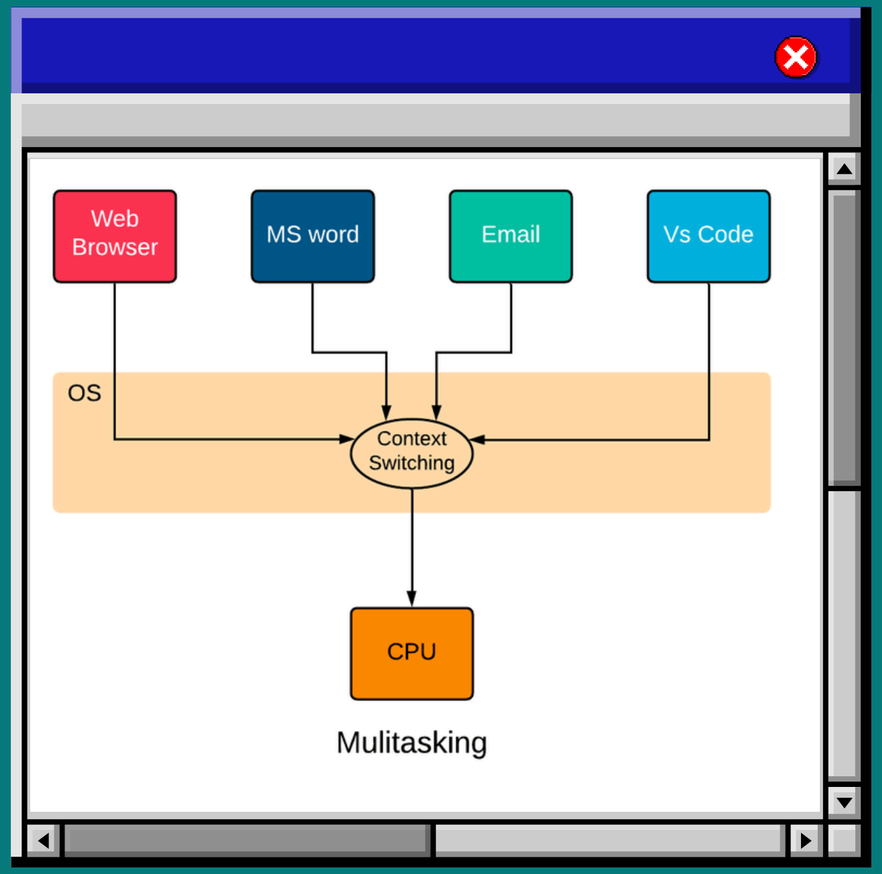


All processes are loaded into the ready state at the same time before execution begins. Each program is given a specific portion of memory which is known as process.

A multiprogram system allows multiple application programs to run concurrently, necessitating efficient memory management to accommodate them in main memory. Each program has private code and data while sharing some modules, like library filesgramming means more than one program can be active at the same time.

Back to Agenda Page

Multitasking



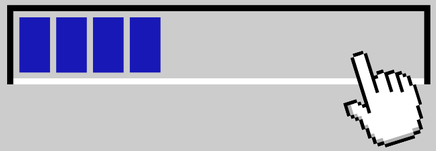
A distinctive feature of multitasking operating systems is their ability to run multiple applications simultaneously, enhancing user productivity and resource utilization through effective CPU scheduling and memory management.

A multitasking operating system allows multiple processes to run simultaneously, improving resource utilization and user experience. It efficiently manages CPU time allocation among processes, enabling them to share system resources while maintaining isolation. Popular examples include Windows, Linux, and macOS, which support both foreground and background tasks effectively.

Back to Agenda Page

Multitasking vs Multiprogramming

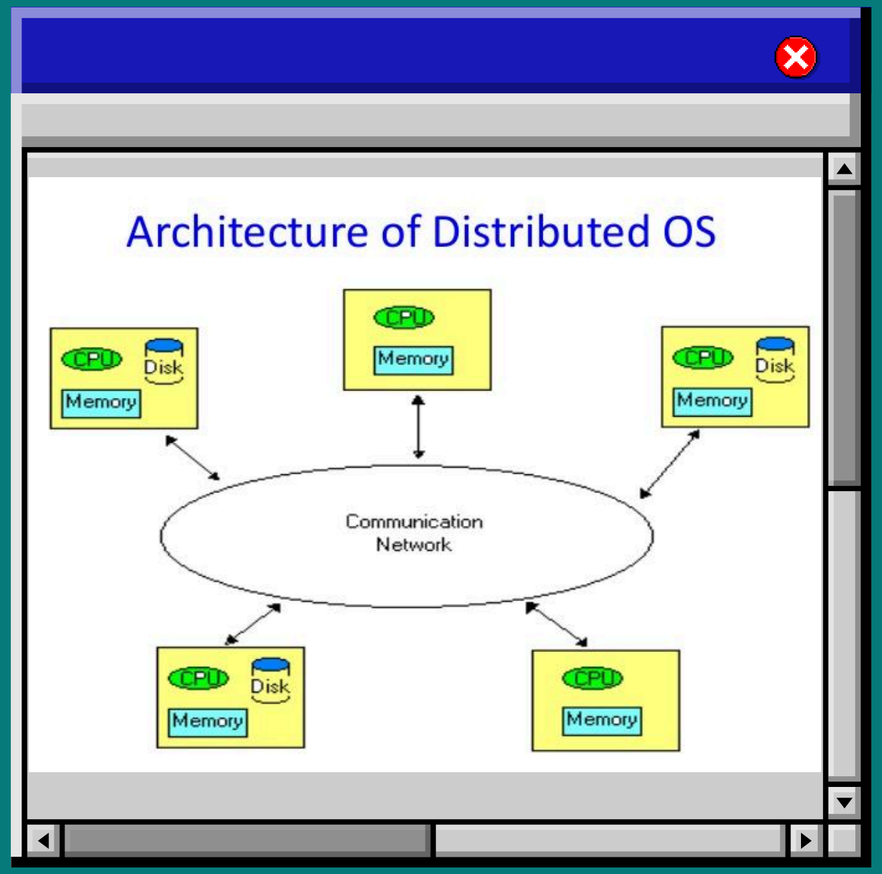
Differences summarized:



Multitasking	Multiprogramming
Multitasking is pre-emptive, i.e., multiple processes execute simultaneously.	Multiprogramming is non pre-emptive, i.e., only 1 process is executed at a time.
In multi-tasking also increases CPU utilization, it also increases responsiveness.	Multi-programming increases CPU utilization by organizing jobs
Time sharing mechanism is used so that multiple tasks can run at the same time.	It uses job scheduling algorithms so that more than one program can run at the same time.
Execution of process takes less time.	Execution of process takes more time.

Back to Agenda Page

Distributed Operating Systems



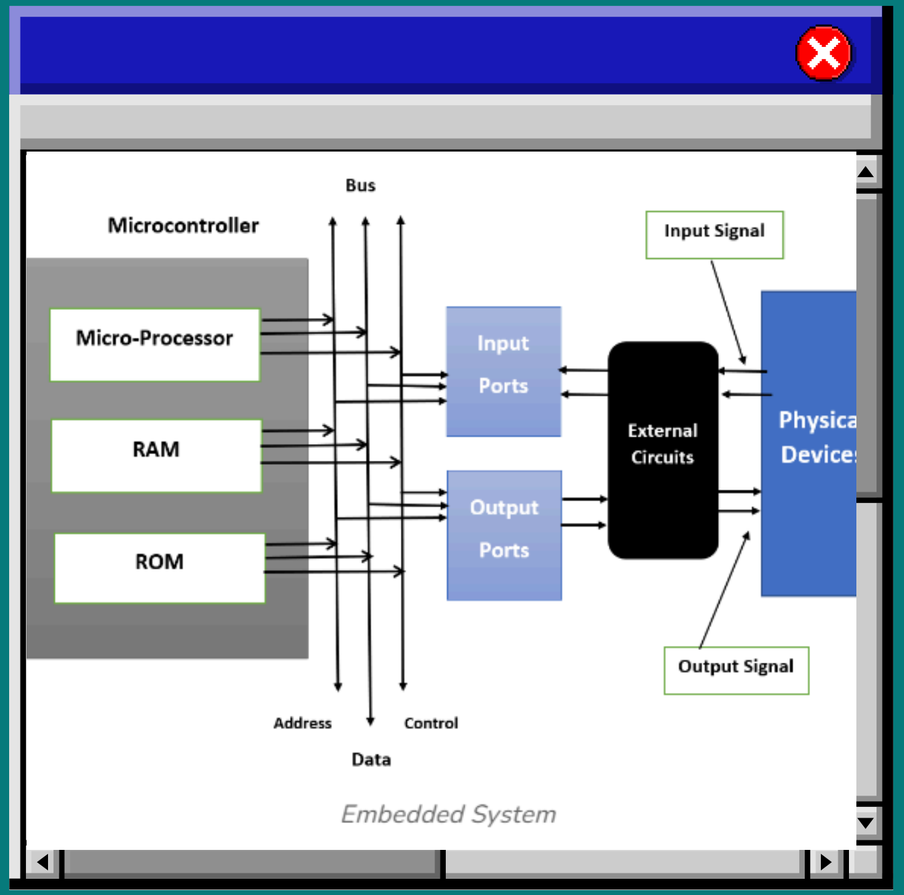
A distributed OS manages a group of separate computers as if they were a single system.

Distributed systems extend multiprocessor systems, connecting independent computers via networks, enabling resource sharing. Operating systems facilitate communication through methods like message passing and RPC, allowing users to interact seamlessly. They also implement fault-tolerant mechanisms to maintain reliability.

Back to Agenda Page

Embedded Operating Systems

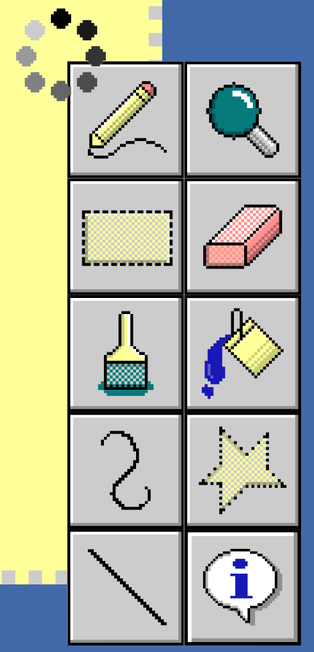
An embedded operating system is a specialized OS designed for specific tasks within larger machines, such as cars, washing machines, and robots. These systems are typically compact, requiring only a few megabytes of storage, and often have limited or no user interfaces, focusing on efficiency and functionality.



Embedded operating systems are designed for specific tasks, emphasizing efficiency, minimal resource usage, and real-time performance. They often have limited user interfaces and are integrated into hardware devices, ensuring reliable operation with low power consumption.

Back to Agenda Page

User and Kernel Modes

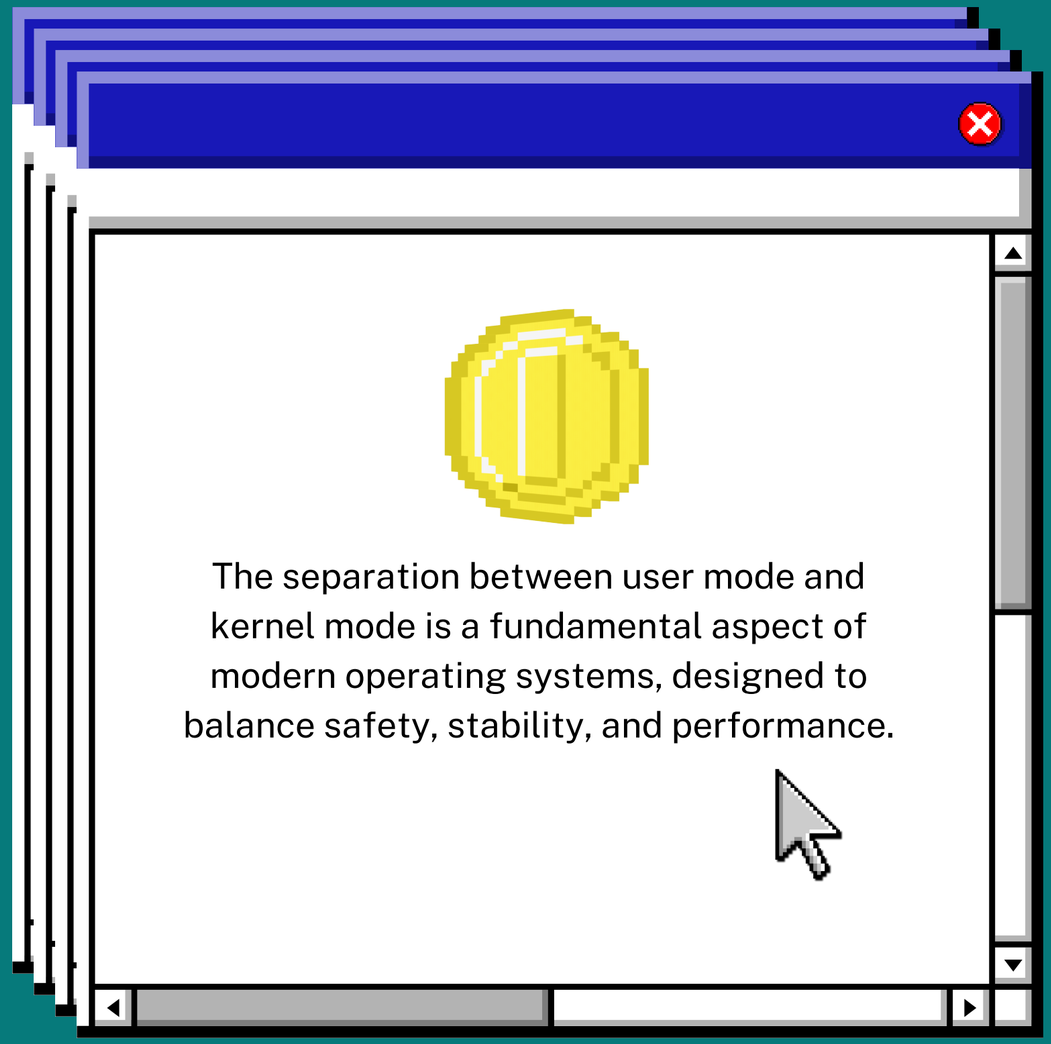


Back to Agenda Page

User Mode

User mode is the mode of the process which has less control and privileges to the OS

It has to send a System Call to access the kernel mode which executes the syscall.

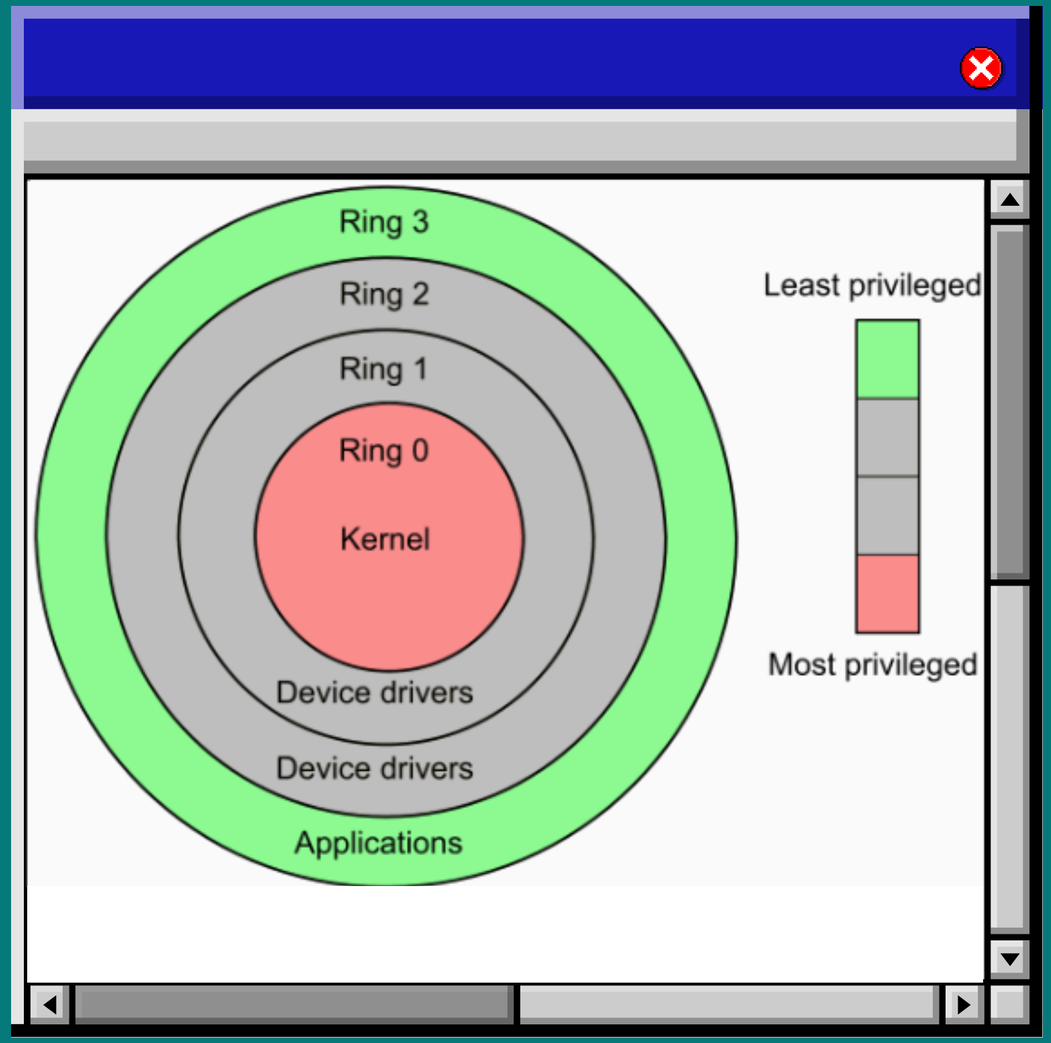


Back to Agenda Page

Kernel Mode

For contrast, kernel mode has full access to all system resources and can execute any CPU instruction.

User mode applications rely on kernel mode to perform tasks that require higher privileges.



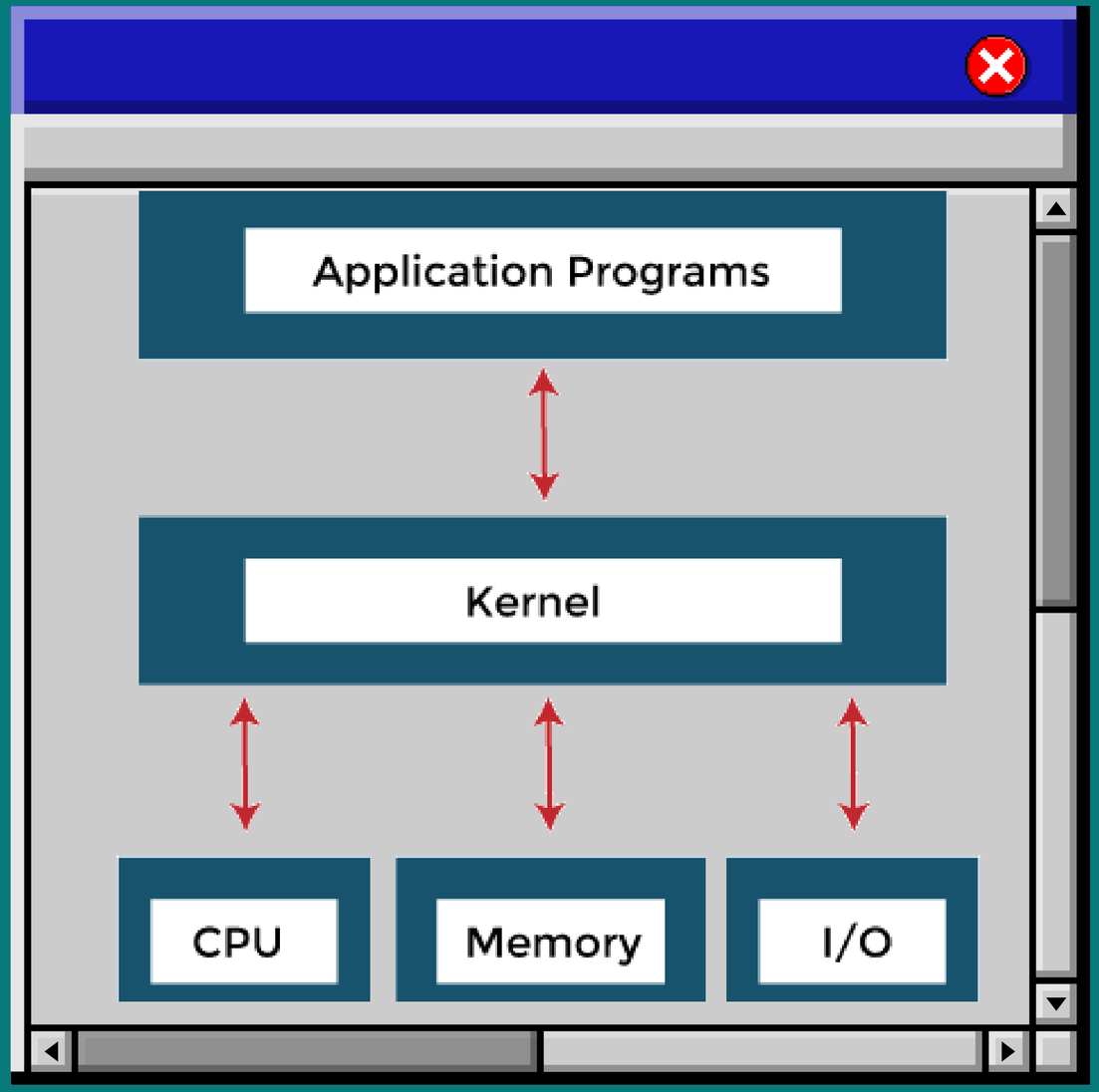
Back to Agenda Page

Link between User Mode and Kernal Mode: System Call

Privileged instructions, like direct hardware control, are restricted to kernel mode, allowing only the OS to execute them.

If a user application needs access to these resources, it must request an OS service via a system call, which temporarily switches to kernel mode to fulfill the request.

Attempts to execute privileged instructions directly in user mode trigger a hardware trap or interrupt, shifting control to the OS and protecting the system from potential errors or security risks.



Back to Agenda Page

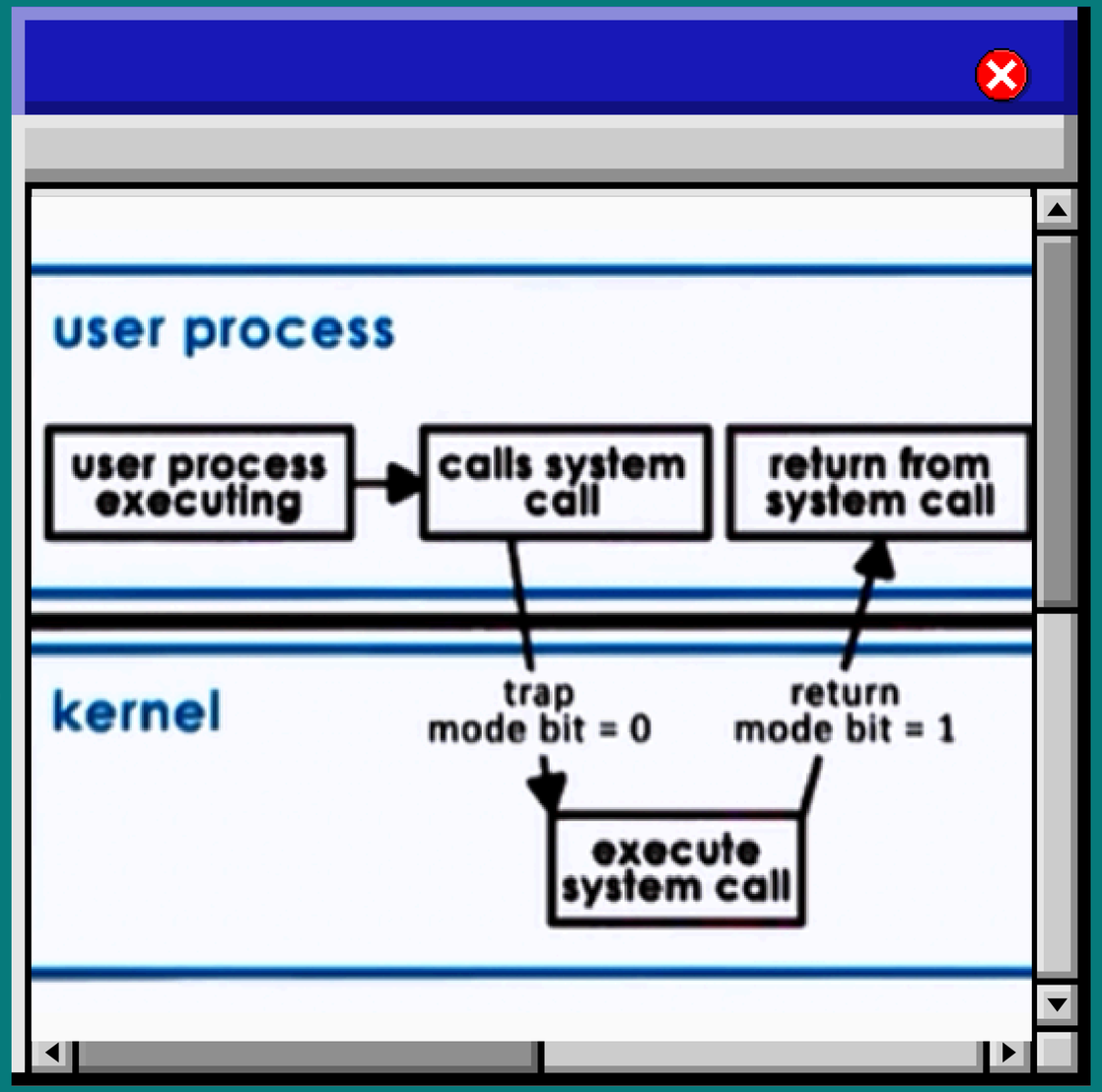
Link between User Mode and Kernal Mode: System Call

In computer systems, dual-mode operation—user mode and kernel (or privileged) mode—safeguards critical system resources and ensures secure task execution.

A hardware mode bit (0 for kernel mode, 1 for user mode) helps distinguish whether the system is executing OS instructions or user-level instructions.

At boot, the system begins in kernel mode to load the operating system and set up the environment.

Once user applications are launched, the system switches to user mode, restricting access to sensitive resources.



Back to Agenda Page

Thank you!

Hope you liked our presentation!

A presentation by:
GROUP - 8:

- Ayan (iit2023002)
- Chirag (iit2023005)
- Samay (iit2023069)
- Samrat (iit2023066)
- Ranjeet (iit2023064)
- Anirudh (iit2023007)