Introduction to Operating Systems and Execution Management

Week 1

SDB

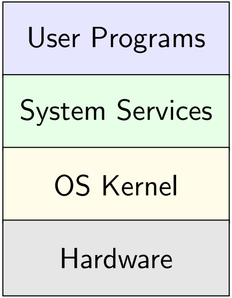
Autumn 2025

Agenda

What is an Operating System? I

Definition

An **Operating System (OS)** is a foundational software layer that manages computer hardware and software resources, and provides a consistent interface for users and applications to interact with the system.



Core Responsibilities of an OS:

- **Managing Programs:** Starts, stops, and coordinates applications (processes and threads).
- **Handling Memory:** Allocates and tracks memory usage for efficient and safe execution.
- **Controlling Devices:** Manages input/output devices like keyboards, disks, and printers.

What is an Operating System? II

- **Organizing Files:** Provides file systems to store, retrieve, and secure data.
- **Ensuring Security:** Protects system resources and user data from unauthorized access.
- **Scheduling Tasks:** Decides which tasks run and when, optimizing performance.

Did You Know?

The first widely used OS was GM-NAA I/O, developed in the 1950s for IBM mainframes. Today, OSes power everything from supercomputers to smartwatches.

Types of Operating Systems

- **Batch OS** – Executes batches of jobs with minimal user interaction. *(e.g., IBM 7094)*
- **Time-Sharing OS** – Allows multiple users to share system resources simultaneously. *(e.g., UNIX)*
- **Real-Time OS (RTOS)** – Guarantees response within strict time constraints. *(e.g., VxWorks, FreeRTOS)*
- **Distributed OS** – Coordinates multiple machines to appear as a single system. *(e.g., Amoeba, Plan 9)*
- **Network OS** – Provides services to computers connected over a network. *(e.g., Novell NetWare)*
- **Mobile/Embedded OS** – Optimized for low-power, resource-constrained devices. *(e.g., Android, iOS, Zephyr)*

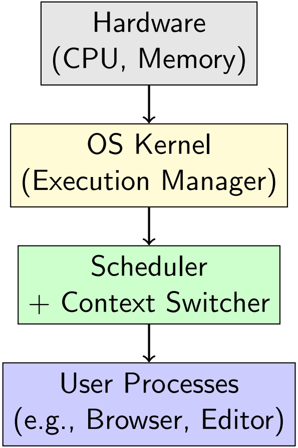
Did You Know?

UNIX, developed in the 1970s at Bell Labs, laid the foundation for many modern OSes including Linux and macOS.

Execution Management Responsibilities

Key functions of an OS during program execution:

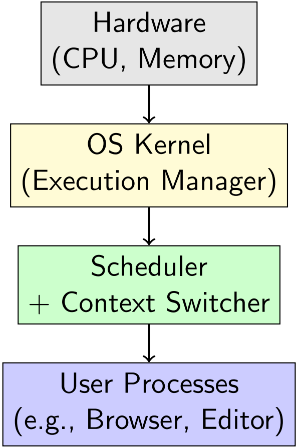
- **Process Lifecycle:** Creating, scheduling, and terminating processes.
- **CPU Scheduling:** Allocating CPU time fairly and efficiently.
- **Context Switching:** Saving and restoring process states.
- **Thread Management:** Coordinating concurrent execution within processes.



Execution Management Responsibilities

Key functions of an OS during program execution:

- **Process Lifecycle:** Creating, scheduling, and terminating processes.
- **CPU Scheduling:** Allocating CPU time fairly and efficiently.
- **Context Switching:** Saving and restoring process states.
- **Thread Management:** Coordinating concurrent execution within processes.



Example

When you open a browser, the OS creates a process, assigns it CPU time, and switches between it and other tasks like music playback.

Modes of CPU Operation

- **User Mode:** Limited access — used for running application code. Cannot directly access hardware or critical memory.
- **Kernel Mode:** Full access — used by the OS to execute privileged instructions and manage system resources.

Modes of CPU Operation

- **User Mode:** Limited access — used for running application code. Cannot directly access hardware or critical memory.
- **Kernel Mode:** Full access — used by the OS to execute privileged instructions and manage system resources.

Why do we need separate modes?

To protect the system from accidental or malicious interference by user programs.

▶ Without this separation, a faulty or malicious app could:

- Overwrite critical OS memory
- Access or corrupt other users' data
- Disable hardware or crash the system

Modes of CPU Operation

- **User Mode:** Limited access — used for running application code. Cannot directly access hardware or critical memory.
- **Kernel Mode:** Full access — used by the OS to execute privileged instructions and manage system resources.

Why do we need separate modes?

To protect the system from accidental or malicious interference by user programs.
▶ Without this separation, a faulty or malicious app could:

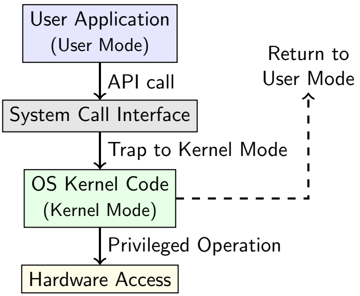
- Overwrite critical OS memory
- Access or corrupt other users' data
- Disable hardware or crash the system

Questions to Ponder

▶ What could happen if a video game could directly access your disk or memory?
▶ Why is it dangerous to allow user programs to execute privileged instructions?
▶ How does the OS enforce this separation in modern CPUs?

System Call and Mode Switch

A **system call** is a controlled request from a user program to the OS for services like file access or memory allocation. It triggers a **switch** from *user mode* to *kernel mode* to safely execute privileged operations.



Analogy

A system call is like ringing a service bell at a hotel desk. You (the guest) can't go behind the desk (kernel), but you can request help through a formal channel.

Questions to Ponder

- Why is direct hardware access by user programs unsafe?
- What ensures only valid system calls are executed?
- How does the OS return control safely to user mode?

Case Study: Running a Program I

What happens when you run `gcc file.c` in a Linux shell?

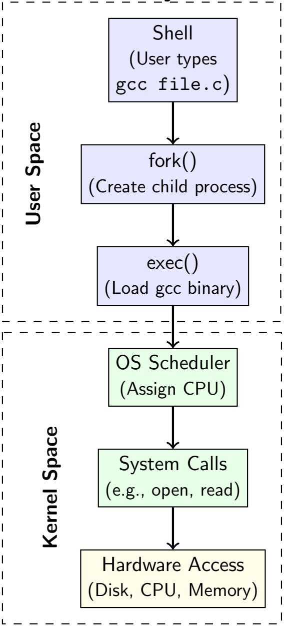
Step-by-step Breakdown:

1. **User Input:** The user types `gcc file.c` in the shell. The shell parses the command and prepares to execute it.
2. **Process Creation – `fork()`:** The shell uses the `fork()` system call to create a new child process. This child is an exact copy of the shell process, including its memory space.
3. **Program Replacement – `exec()`:** In the child process, the shell calls `exec()` to replace its memory image with that of the `gcc` compiler. This loads the binary code of `gcc` into memory and begins execution from its entry point.

Case Study: Running a Program II

4. **Scheduling and Execution:** The OS scheduler places the new process in the ready queue. When the CPU is available, the process is scheduled for execution. The OS performs a context switch to load the process state into the CPU.
5. **System Calls and Mode Switching:** During execution, `gcc` makes system calls (e.g., to read the source file, write output, allocate memory). Each system call triggers a switch from user mode to kernel mode, allowing the OS to safely perform privileged operations.
6. **Completion and Exit:** Once compilation is complete, the process calls `exit()`, and the OS reclaims its resources. The parent shell process is notified via `wait()`.

Case Study: Running a Program III



Analogy

Think of this like a relay race: the shell hands off control to a new runner (the compiler), who then takes over the track (CPU) and uses tools (system calls) to complete the job.

Try This!

Run `strace gcc file.c` in a terminal to see the actual system calls made during execution.

Think-Pair-Share: Direct Hardware Access

Prompt: What could go wrong if a user-level process could directly access disk hardware?

Think-Pair-Share: Direct Hardware Access

Prompt: What could go wrong if a user-level process could directly access disk hardware?

Hint: Consider implications for:

- **System Stability** – Could crash or corrupt the OS.
- **Security** – Could read or overwrite sensitive data.
- **Performance** – Could bypass scheduling and cause resource starvation.

Think-Pair-Share: Direct Hardware Access

Prompt: What could go wrong if a user-level process could directly access disk hardware?

Hint: Consider implications for:

- **System Stability** – Could crash or corrupt the OS.
- **Security** – Could read or overwrite sensitive data.
- **Performance** – Could bypass scheduling and cause resource starvation.

Discussion Extension

How do modern OSes prevent this? What role do device drivers and permission models play?

Why Execution Control Matters

Execution control is not just academic — it powers real-world systems:

- **Multicore CPUs:** OSes manage thousands of threads across cores for responsiveness.
- **Real-Time Systems:** In robotics and automotive, timing is critical (e.g., ABS braking).
- **Cloud Computing:** OS-level scheduling underpins Kubernetes and container orchestration.
- **Embedded Devices:** OSes like FreeRTOS must be ultra-efficient and deterministic.

Why Execution Control Matters

Execution control is not just academic — it powers real-world systems:

- **Multicore CPUs:** OSes manage thousands of threads across cores for responsiveness.
- **Real-Time Systems:** In robotics and automotive, timing is critical (e.g., ABS braking).
- **Cloud Computing:** OS-level scheduling underpins Kubernetes and container orchestration.
- **Embedded Devices:** OSes like FreeRTOS must be ultra-efficient and deterministic.

Did You Know?

NASA's Mars rovers run on real-time operating systems to ensure precise control and fault tolerance.

Wrap-up and Summary

Key Takeaways:

- The OS manages execution through processes, threads, and CPU scheduling.
- Kernel mode enables privileged operations; user mode ensures safety and isolation.
- System calls are the bridge between user programs and kernel services.
- Execution control is foundational to performance, security, and reliability.

Wrap-up and Summary

Key Takeaways:

- The OS manages execution through processes, threads, and CPU scheduling.
- Kernel mode enables privileged operations; user mode ensures safety and isolation.
- System calls are the bridge between user programs and kernel services.
- Execution control is foundational to performance, security, and reliability.

Reflect

Can you think of a real-world failure caused by poor execution control or lack of isolation?

Next Week Preview: System Calls and OS Structures

Coming Up:

- Deep dive into `fork()`, `exec()`, `wait()`, and `exit()`.
- Explore OS architectures: Monolithic, Layered, Microkernel.
- Hands-on: Build and visualize a process tree in Linux.

Next Week Preview: System Calls and OS Structures

Coming Up:

- Deep dive into `fork()`, `exec()`, `wait()`, and `exit()`.
- Explore OS architectures: Monolithic, Layered, Microkernel.
- Hands-on: Build and visualize a process tree in Linux.

Prep Activity

Try running `ps -ef --forest` on a Linux system to preview how processes are structured.

Outline

Quiz: Test Your Understanding

Choose the correct answer or discuss briefly:

1. **Which of the following operations requires a switch to kernel mode?**
   - A. Adding two numbers in a program
   - B. Reading a file from disk
   - C. Printing to the screen
   - D. Declaring a variable

Quiz: Test Your Understanding

Choose the correct answer or discuss briefly:

1. **Which of the following operations requires a switch to kernel mode?**
   - A. Adding two numbers in a program
   - B. Reading a file from disk
   - C. Printing to the screen
   - D. Declaring a variable
2. **What is the main purpose of a system call?**
   - A. To compile code
   - B. To switch between applications
   - C. To request OS services from user space
   - D. To allocate memory in hardware

Quiz: Test Your Understanding

Choose the correct answer or discuss briefly:

1. **Which of the following operations requires a switch to kernel mode?**
   A. Adding two numbers in a program
   B. Reading a file from disk
   C. Printing to the screen
   D. Declaring a variable
2. **What is the main purpose of a system call?**
   A. To compile code
   B. To switch between applications
   C. To request OS services from user space
   D. To allocate memory in hardware
3. **Short Answer:** Why is it dangerous to allow user programs direct access to hardware?

Quiz: Test Your Understanding

Choose the correct answer or discuss briefly:

1. **Which of the following operations requires a switch to kernel mode?**
   - A. Adding two numbers in a program
   - B. Reading a file from disk
   - C. Printing to the screen
   - D. Declaring a variable
2. **What is the main purpose of a system call?**
   - A. To compile code
   - B. To switch between applications
   - C. To request OS services from user space
   - D. To allocate memory in hardware
3. **Short Answer:** Why is it dangerous to allow user programs direct access to hardware?
4. **True or False:** A context switch only involves changing the program counter.

Quiz: Test Your Understanding

Choose the correct answer or discuss briefly:

1. **Which of the following operations requires a switch to kernel mode?**
   A. Adding two numbers in a program
   B. Reading a file from disk
   C. Printing to the screen
   D. Declaring a variable
2. **What is the main purpose of a system call?**
   A. To compile code
   B. To switch between applications
   C. To request OS services from user space
   D. To allocate memory in hardware
3. **Short Answer:** Why is it dangerous to allow user programs direct access to hardware?
4. **True or False:** A context switch only involves changing the program counter.
5. **Challenge:** Describe a real-world scenario where poor execution control could lead to system failure.

Quiz: Apply What You've Learned

Challenge your understanding with these deeper questions:

1. **Scenario:** A real-time system controlling a robotic arm misses a deadline due to a delayed context switch. *What OS-level mechanisms could prevent this?*

Quiz: Apply What You've Learned

Challenge your understanding with these deeper questions:

1. **Scenario:** A real-time system controlling a robotic arm misses a deadline due to a delayed context switch. *What OS-level mechanisms could prevent this?*
2. **Multiple Choice:** Which of the following best describes the role of the scheduler?
   - A. It compiles user programs.
   - B. It decides which process gets CPU time next.
   - C. It manages file permissions.
   - D. It handles hardware interrupts directly.

Quiz: Apply What You've Learned

Challenge your understanding with these deeper questions:

1. **Scenario:** A real-time system controlling a robotic arm misses a deadline due to a delayed context switch. *What OS-level mechanisms could prevent this?*
2. **Multiple Choice:** Which of the following best describes the role of the scheduler?
   - A. It compiles user programs.
   - B. It decides which process gets CPU time next.
   - C. It manages file permissions.
   - D. It handles hardware interrupts directly.
3. **Short Answer:** How does the OS ensure that a system call made by a user process is safe and valid?

Quiz: Apply What You've Learned

Challenge your understanding with these deeper questions:

1. **Scenario:** A real-time system controlling a robotic arm misses a deadline due to a delayed context switch. *What OS-level mechanisms could prevent this?*
2. **Multiple Choice:** Which of the following best describes the role of the scheduler?
   - A. It compiles user programs.
   - B. It decides which process gets CPU time next.
   - C. It manages file permissions.
   - D. It handles hardware interrupts directly.
3. **Short Answer:** How does the OS ensure that a system call made by a user process is safe and valid?
4. **True or False:** All system calls must result in a context switch.

Quiz: Apply What You've Learned

Challenge your understanding with these deeper questions:

1. **Scenario:** A real-time system controlling a robotic arm misses a deadline due to a delayed context switch. *What OS-level mechanisms could prevent this?*
2. **Multiple Choice:** Which of the following best describes the role of the scheduler?
   - A. It compiles user programs.
   - B. It decides which process gets CPU time next.
   - C. It manages file permissions.
   - D. It handles hardware interrupts directly.
3. **Short Answer:** How does the OS ensure that a system call made by a user process is safe and valid?
4. **True or False:** All system calls must result in a context switch.
5. **Discussion:** Compare execution control in a general-purpose OS (like Linux) vs. a real-time OS (like FreeRTOS).

Exercise: Simulating a Context Switch

Objective: Understand what happens during a context switch and what data the OS must manage.

Task

Write pseudocode or a flowchart to simulate a context switch between two processes. Identify what information must be saved and restored by the OS.

Exercise: Simulating a Context Switch

Objective: Understand what happens during a context switch and what data the OS must manage.

Task

Write pseudocode or a flowchart to simulate a context switch between two processes. Identify what information must be saved and restored by the OS.

Hints:

- Think about registers, program counter, stack pointer, and memory state.
- Consider how the scheduler decides which process to run next.
- What role does the process control block (PCB) play?

Exercise: Simulating a Context Switch

Objective: Understand what happens during a context switch and what data the OS must manage.

Task

Write pseudocode or a flowchart to simulate a context switch between two processes. Identify what information must be saved and restored by the OS.

Hints:

- Think about registers, program counter, stack pointer, and memory state.
- Consider how the scheduler decides which process to run next.
- What role does the process control block (PCB) play?

Challenge

Can you identify a real-world scenario where context switching is critical (e.g., gaming, real-time control)?