Operating Systems – Week 1 Lecture Notes

Instructor: SDB
Theme: Execution Management
Topic: *Role of Operating System & Execution Management*

Lecture Script

Welcome to Week 1 of Operating Systems!

I'm SDB, and this semester, we will look behind the scenes of modern computing — into how operating systems **manage execution**, **protect resources**, and **coordinate processes** that seem to run magically in parallel.

Let's begin with a common scenario:

You open Netflix in a browser, Spotify plays in the background, and your files sync to the cloud.
But under the hood, the OS is:

- Allocating CPU time to each task
- Managing memory and file buffers
- Scheduling disk and network I/O
- Ensuring Spotify crashing doesn't take Chrome down

Let's unpack how this works, starting with what an OS actually does.

Core Concepts and Contextual Definitions

❖ *What is an Operating System?*

An **Operating System (OS)** is **system software** that provides a **controlled interface** between the hardware and user-level applications.

It is responsible for:

- **Abstracting** low-level device details
- **Allocating and tracking** system resources (CPU, memory, I/O)
- **Protecting** users and programs from each other
- **Managing concurrent execution** of processes

❖ *OS Goals*

- **Convenience** – Provide a usable, robust environment for users and apps
- **Efficiency** – Maximize utilization of hardware resources
- **Portability/Evolvability** – OS should be maintainable and extensible

---

❖ *User Mode vs Kernel Mode*

- **Kernel Mode**: Full access to hardware and system instructions.
  The **kernel** runs in this mode — the core, privileged part of the OS.

- **User Mode**: Limited access; programs cannot directly interact with hardware.
  All user applications run here. If they need privileged operations (like file I/O), they invoke **system calls** to switch into kernel mode.

Why?
This separation enforces **protection**, ensuring that buggy or malicious programs can't crash or corrupt the whole system.

❖ *Kernel*

The **kernel** is the central component of the OS. It manages:

- Process and thread scheduling
- Memory allocation
- Device drivers and I/O coordination
- System calls interface
- Interrupt handling

It operates in **kernel mode** and is the first code loaded during boot.

❖ *Scheduler*

The **scheduler** is a kernel module that:

- Chooses which process gets the CPU
- Decides how long it runs (time quantum)
- Ensures fairness or priority depending on policy

Scheduling affects:

- Responsiveness of interactive apps
- Throughput of background jobs
- CPU utilization

❖ *Protection vs Security*

- **Protection**: Internal — mechanisms to ensure processes don't interfere with each other
  E.g., memory isolation, file access rights
- **Security**: External — defending against unauthorized access or misuse
  E.g., login authentication, permission enforcement, encryption

Both are responsibilities of the OS.

❖ *Execution Flow (Power to Process)*

1. **Bootloader** loads kernel from disk into RAM
2. **Kernel** initializes core subsystems (memory, CPU, devices)
3. **System daemons** and services launch
4. **User-level shell** or GUI starts

5. **Processes** begin execution under OS control

Live Demo: Observing Execution

Demo Commands

```
top          # Observe real-time scheduling and memory use
ps -ef       # Snapshot of process list with parent/child hierarchy
sleep 60 &   # Background a dummy task
jobs         # Inspect shell-managed background tasks
dmesg | tail # View last kernel messages (if permitted)
```

Concepts to Emphasize

- Each process has a **PID**
- Multiple states: Running, Ready, Waiting
- The shell can multitask because of the **scheduler** and **context switching**

Caselet: Your Laptop as a Scheduler Playground

Scenario: You are running Zoom, compiling code, syncing notes via OneDrive, and downloading updates.

Here's how the OS helps:

- **Zoom** gets prioritized CPU for low-latency communication
- **Compiler** is CPU-intensive, runs in background
- **OneDrive** uses network I/O, often blocked waiting for data
- **OS scheduler** switches between them every few ms based on policy

This happens **hundreds of times per second**.

Glossary of New Terms

Term	Meaning
Kernel	Core part of the OS with full control over hardware
User Mode	Restricted mode for user programs
Protection	Prevents unintended interactions between processes
Security	Prevents unauthorized access to the system
Scheduler	OS component deciding which process runs next
System Call	Controlled mechanism to invoke kernel services
Context Switch	Saving/restoring CPU state between process switches
Daemon	Background process for system tasks (e.g., sync, logging)
Multitasking	Running multiple programs concurrently using time-sharing

---

Related Topics to Explore

- History of OS design: UNIX, DOS, Linux
- Types of Operating Systems: batch, time-sharing, real-time, distributed
- Monolithic vs Microkernel architectures
- Kernel bypass (e.g., RDMA, DPDK for high-performance I/O)
- System call tracing using `strace` or `ltrace`

✍ Review & Guided Exercises

1. Define the terms: kernel, system call, protection, scheduler
2. Explain why kernel and user mode separation is critical
3. Describe the startup process from BIOS to shell prompt
4. Try `top`, `ps`, `sleep &`, `jobs`. Describe what each shows
5. Identify 3 real-world apps and map them to OS services they rely on

---

Open Exploration Questions

- What would happen if all programs had kernel access?
- How does the OS protect itself from a buggy antivirus engine?
- Explore `htop` (if available) — how does it visualize process states?
- Can you measure CPU time of a process using `time` or `/proc`?
