Operating Systems

Structure, Execution, Coordination

Core Modules

OS Fundamentals

- Definitions, goals, evolution
- System structure, kernel vs user mode, system calls

Process & Thread Management

- Lifecycle, context switching, threading models

CPU Scheduling

- Algorithms (FCFS, SJF, RR, Priority, MLFQ), evaluation metrics

Synchronization & Deadlocks

- Semaphores, monitors, classical problems, deadlock strategies

Memory Management

- Paging, segmentation, virtual memory, page replacement

File Systems & Storage

- File structure, allocation, directories, disk scheduling

Virtualization & Containers

- VMs, hypervisors, Docker/LXC, namespaces, cgroups

Faculty-wise Thematic Division

Execution Management (SDB)	Resource Management (SSB)	Coordination & Modern OS (RD)
• OS intro, services, system calls • Process lifecycle, PCB, context switching • Thread types and libraries • CPU scheduling algorithms • Case studies: Linux CFS, Windows Scheduler	• Memory: Allocation, Paging, Segmentation • Virtual memory, page replacement strategies • File systems: ext2/ext4, FAT, NTFS • Disk scheduling, RAID, directory mgmt	• IPC: Shared memory, message passing • Synchronization: Semaphores, monitors • Classical concurrency problems • Deadlocks: prevention, avoidance, detection • Virtualization & containers: Docker, LXC

Sample Weekly Progression (Indicative)

Week Range	Focus Areas	Labs/Practicals
1–4	OS intro, system calls, processes, memory basics	System call tracing, simple allocators
5–8	Threads, scheduling, paging, synchronization	Thread creation, scheduler sim, semaphores
9–12	Deadlocks, file systems, virtualization	File system layout demo, deadlock simulation, Docker run/labs

Mid-course Project Ideas - Thematic

Execution Management

- **Mini Shell:** Implement fork, exec, wait
- **CPU Scheduler Simulator**: FCFS, SJF, RR, Priority with visualization
- **Thread Pool Executor**: Simulate task queuing with multiple threads

Resource Management

- **Memory Allocator Simulator**: First-fit, best-fit with memory compaction
- **Virtual Memory Visualizer**: Page table + TLB + page fault animation
- **FUSE Filesystem Skeleton**: Implement basic file ops in user space

Coordination & Modern Systems

- **Producer-Consumer Simulation**: Semaphore-guarded multithreaded buffer
- **Deadlock Analyzer**: RAG simulation + cycle detection
- **Container Runtime Monitor**: Docker resource control with cgroup inspection

Final Capstone Project Ideas I

- **Multitasking OS Kernel Emulator**: Simulate processes, scheduling, memory, and I/O in a user-mode kernel model
- **Container-Aware Resource Allocator**: Build a mock cgroup-like resource controller for CPU, memory
- **Lightweight Hypervisor Simulator**: Simulate basic VM scheduling over host CPU/memory using hypercalls
- **Paging & Swapping Engine with Visualization**: Simulate working set, page faults, swapping to disk
- **System Behavior Profiler**: Analyze /proc, /sys, and I/O logs to infer live system state in Linux

Final Capstone Project Ideas II

- **User-Level Multitasking Environment**: Combine process/thread simulation with scheduling and memory modules.
- **Educational OS Kernel Emulator**: Simulate basic OS behaviors: process state mgmt, memory access, I/O scheduling.
- **Lightweight Container Runtime**: Build a small container launcher using Linux namespaces and cgroups with process isolation.
- **Virtual Memory and Swap Emulator**: Model how virtual memory and paging interacts with disk, with visualization.
- **Kernel-Level Log Analyzer**: Analyze /proc and /var/log to reverse-engineer process and memory behavior in real Linux systems.

What You'll Learn

- OS design, internals, and practical constraints
- Writing, debugging, and scheduling concurrent programs
- Managing system resources securely and efficiently
- Operating systems' role in modern computing platforms (cloud, mobile, embedded)
- Preparing for roles in systems programming, devops, embedded OS, and low-level security