COURSE SYLLABUS

CS330 • Operating Systems & Concurrency

School of Computational Arts & Sciences • Undergraduate • 6 ECTS

Overview

Explore operating-system fundamentals: processes/threads, scheduling, memory management, files, and concurrency primitives. Students build disciplined debugging habits and reason about race conditions, deadlocks, and performance at the systems boundary.

LOGISTICS

Credits: 6 ECTS
Level: Undergraduate
School: School of Computational Arts & Sciences
Prerequisites: CS210 (or equivalent data structures)
Tags: systems, concurrency
Meeting time: Weekly lecture + concurrency lab
Instruction mode: Hands-on systems: build small components and debug real failures

LEARNING OUTCOMES

You will be able to:
- Explain key OS abstractions (processes, threads, memory, files)
- Diagnose performance issues and resource contention
- Implement small components that demonstrate core OS concepts
- Explain core OS abstractions (processes, threads, memory, files) and their trade-offs
- Diagnose concurrency bugs using traces and minimal reproducers
- Reason about performance bottlenecks and resource contention

ASSESSMENT

Components
- Labs: 45%
- Midterm: 20%
- Final project (systems report + demo): 35%

Assignments emphasize disciplined reasoning: you must describe what you observed (trace),
what you believe is happening (hypothesis), and what you changed (fix). Correct fixes
without an explanation receive partial credit.

WEEKLY PLAN

Schedule

Week 1: Processes and scheduling

- Context switching
- Scheduling tradeoffs
- Measurement

Week 2: Concurrency

- Locks
- Deadlocks
- Testing concurrency

Week 3: Memory

- Paging
- Allocation
- Performance

Week 4: Files and I/O

- Buffers
- Failure modes
- Tooling

Extended outline

- Processes and scheduling: latency, throughput, and context switching
- Memory: address spaces, paging intuition, and leaks
- Files and IO: buffering and failure modes
- Concurrency primitives: locks, semaphores, and deadlocks
- Debugging systems: traces, profiling, and invariants
- Final: implement a small concurrent component + report

POLICIES & RESOURCES

- Safety: do not run untrusted binaries; keep experiments isolated.
- Postmortems are required when work fails.
- Collaboration: discuss debugging strategies; implement independently.

Suggested resources

- Concurrency bug checklist: race, deadlock, starvation
- Profiling notes: where time goes, how to measure
- Template: debugging log with trace + hypothesis + fix