Advanced Operating Systems

For the Cloud-Native & AI Era

Week 1: What Problems Does This Course Solve?

This Week's Plan

Session 1 (80 min): Course Introduction

- Who are we?
- What is this course about?
- How will you be evaluated?
- What do you need to set up?

Session 2 (80 min): Environment Setup

- Get your tools working
- Run your first measurement

Who Am I?

Dong Dai

- Associate Professor, Computer & Information Sciences, University of Delaware
- PhD from University of Science and Technology of China (USTC)

Research Areas:

- High-Performance Computing (HPC) systems
- Parallel file systems and I/O optimization
- AI for Systems (AI4HPC)

Data Intelligence Research Lab (DIRLab)

What we do:

- Optimize data-intensive infrastructure: parallel file systems, metadata management, job scheduling
- Apply machine learning to understand and predict system behavior
- Build intelligent tools for HPC storage optimization

Recent work (2025):

- [SC'25] STELLAR: Using LLM autonomous reasoning to tune parallel file systems
- [SC'25] Improving SpGEMM performance through matrix-reordering
- [HPDC'25] TSUE: Two-stage data update for erasure-coded cluster file systems

🔗 sites.google.com/udel.edu/dirlab

Teaching Assistant

Minqiu Sun

- Ph.D. Student, DIRLab Member
- Research: LLM Post-training and Agent System Design
- [PDSW'25] LLMTailor: Layer-wise tailoring for efficient LLM checkpointing

Office Hours: TBA

Responsibilities: Lab support, grading, office hours

Who Are You?

Round table introduction:

- Your name
- Your program (PhD / MS / Undergrad)
- Your background (what OS/systems courses have you taken?)
- One system you want to understand better

(Take notes — you might find project partners here!)

Course Information

Class Time	12:45 – 2:05pm
Location	Memorial Hall, Room 113
Lecturer	Dong Dai (daidong.github.io)
Office	Fintech 416B
TA	TBA
Office Hours	TBA

Course Policies: Attendance

This is an in-person course.

- ✅ All students are required to attend every class
- 🏥 Sick leave requires a doctor's note
- 📊 **20% of your grade** is participation
- 📝 In-class quizzes and activities

If you're not in class, you can't participate!

Course Policies: Assignments

Assignments:

- Most are **hands-on labs** (coding + measurement + explanation)
- May include some written assignments

Deadlines & Late Submission:

- ⚠️ All deadlines are **final**
- 🛡️ In case of unforeseen events: **3-day safety margin** (use wisely!)
- ❌ **NO late submissions** will be accepted after the margin

Plan ahead. Start early.

Course Policies: Projects & Exams

Programming Projects:

- Final project is a system case study (40% of grade)
- Can work individually or in pairs
- Milestones throughout the semester

Exams:

🎉 NO EXAMS! 🎉

Your work (labs + project) speaks for itself.

Academic Integrity

Individual Work:

- All assignments are to be completed **individually** (unless specified)
- ❌ No code from other students
- ✅ Code from generative AI (ChatGPT, Copilot, etc.) is **welcome**

Collaboration & AI Use:

- ✅ You are encouraged to **discuss** with classmates
- ✅ You are encouraged to use **ChatGPT** to facilitate learning
- ⚠️ But you must **understand** what you submit

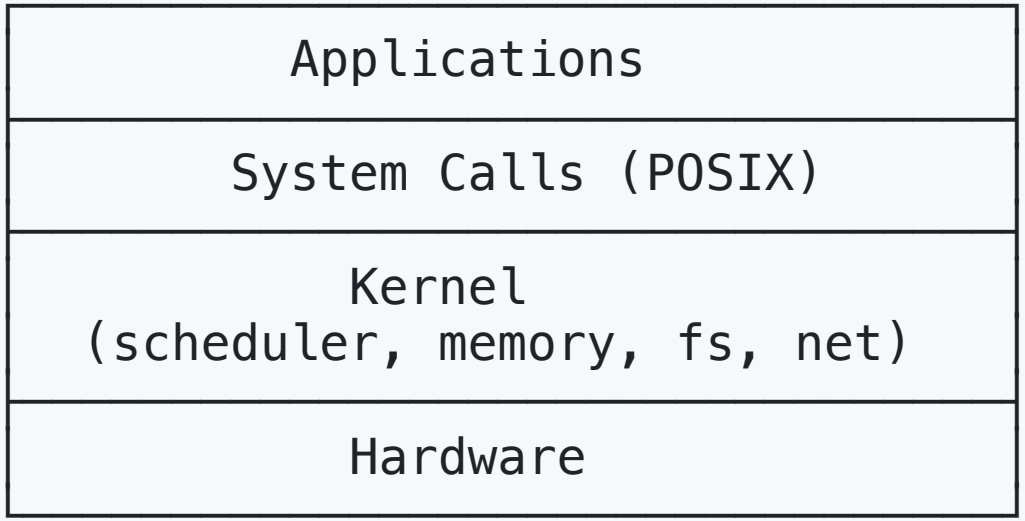
If you can't explain your code, you didn't write it.

The Million Dollar Question

> What is an "Operating System" in 2025?

- Is it just the Linux kernel?
- Where does the OS end and the "platform" begin?
- What should a graduate student learn about OS today?

Traditional OS Course



Focus: internals of a single kernel on a single machine.

This remains essential. We will cover these core concepts.

OS Core Concepts We Will Cover

Concept	What You'll Learn
Process & Thread	Creation, lifecycle, context switch
Scheduling	Policies, fairness, latency
Virtual Memory	Paging, TLB, page cache
Concurrency	Locks, synchronization, races
I/O & Storage	Filesystems, buffering, persistence
Security	Isolation, permissions, sandboxing

These are **not optional**. You need them to understand what's happening.

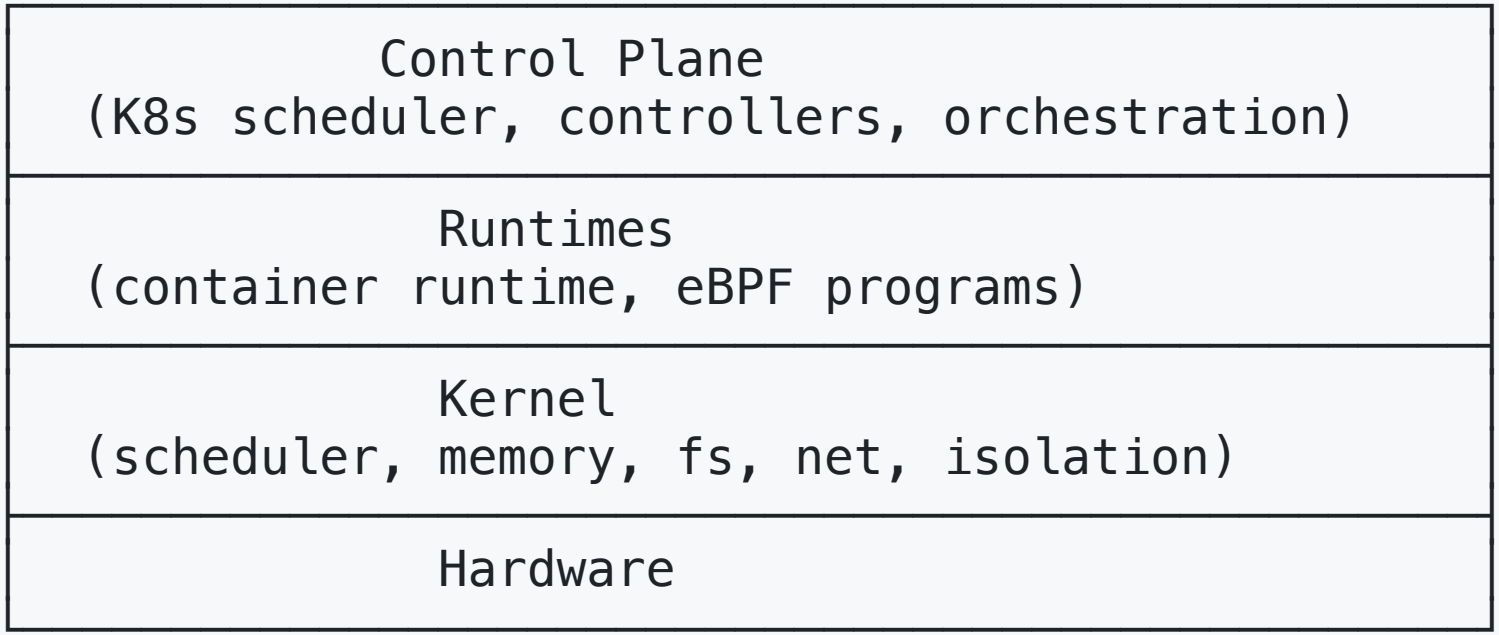
What's Different in Practice Today?

The core concepts remain, but the **context** has changed:

- **Containers**: process isolation using namespace + cgroup
- **Kubernetes**: cluster-level resource management
- **eBPF**: kernel observability without modifying kernel code
- **New workloads**: LLM inference, AI agents

We'll use these as **case studies** to apply OS concepts.

This Course: A Perspective



This is **one way** to view modern systems. Not the only way.

Course Thesis

> **OS core concepts** (processes, memory, scheduling, I/O, security) are the foundation.
> We learn them by **applying them to real systems** and **measuring real behavior**.

This course trains you to:

1. **Understand** — Master OS fundamentals: processes, memory, scheduling, I/O, isolation
2. **Measure** — Design and run reproducible experiments with real metrics
3. **Explain** — Trace observed behavior back to specific OS mechanisms

Why This Approach?

The skills you need to debug a "p99 spike" or "container OOM" are **built on** OS internals:

Symptom	OS Concept	Tool
p99 latency spike	Scheduling, page fault	perf, eBPF
Container OOM	Memory management, cgroups	cgroup stats
CPU throttling	CFS bandwidth control	/proc, /sys
Network delay	TCP/IP stack, context switch	tcpdump, bpftrace

Understanding **why** requires knowing the OS.

Learning Objectives

By the end of this course, you will be able to:

1. **Explain** core OS concepts: process lifecycle, scheduling policies, virtual memory, file systems, and concurrency primitives
2. **Use measurement tools** (perf, eBPF, strace) to investigate performance issues with reproducible experiments
3. **Construct isolation boundaries** (namespace, cgroup, seccomp) and explain the underlying OS mechanisms
4. **Analyze resource management** in container and cluster environments (reproduce OOM, throttling, eviction)
5. **Profile modern workloads** (LLM inference, agent tools) and trace bottlenecks to OS-level causes

Two Sessions Every Week

Session	What Happens	Graded?
Session 1: Lecture	I teach concepts + case studies	In-class quiz → Participation (20%)
Session 2: Lab Workshop	You start the lab, I help	Lab submission → Labs (40%)

Both sessions are **required**. Missing lecture = missing participation points.

How It All Fits Together

```
SESSION 1: LECTURE (in class, 80 min)
  • I teach OS concepts + modern context + case study
  • Preview what you'll do in the lab
  • 📝 In-class quiz/activity → Participation (20%)
```

↓

```
SESSION 2: LAB WORKSHOP (in class, 80 min)
  • You START the lab with me and TA available
  • Get help, ask questions, understand requirements
  • ⚠️ You will NOT finish the lab in class!
```

↓

```
AFTER CLASS: Complete Lab Assignment
  • Continue working on the lab outside of class
  • Deadline: before next week's lecture
  • 📊 Submit script + data + explanation → Labs (40%)
```

Lecture: What We'll Cover

Each lecture will include:

1. **OS concepts** (30-40 min): The fundamentals you need to know
2. **Modern context** (20-30 min): How these concepts apply today
3. **Case study** (15-20 min): A real scenario demonstrating the concepts
4. **Lab preview** (10 min): What you'll be measuring and why

I'll provide reading materials, but the lecture is where we build shared understanding.

Lab Workshop: What Happens in Class

The lab workshop is for **getting started**, not finishing:

1. **Warm-up** (10 min): Review objectives and metrics
2. **Guided Build** (30 min): I show the skeleton and common pitfalls
3. **Work Time** (30 min): You start working, ask questions
4. **Debrief** (10 min): Share progress, identify common issues

After class: You continue working and submit before next lecture.

Lab Assignment: What You Submit

After each lab workshop, you complete the lab and submit:

1. **Reproducible scripts** (Makefile or bash)
2. **Data files** (CSV, logs, measurements)
3. **Explanation document** (1-2 pages)

Deadline: Before next week's lecture (exact time on each lab handout)

~8 labs total, each worth ~5% of your grade.

What Makes a Good Lab Report?

A good lab report demonstrates:

1. **Reproducibility**: I can run your script on a clean VM and get similar results
2. **Clear metrics**: Numbers have units, sampling windows, and defined baselines
3. **Comparison**: At least one baseline (before vs. after, config A vs. B)
4. **Causal explanation**: Link observed behavior to a specific OS mechanism

Lab Report Anti-Patterns

❌ "It was slow" — What does "slow" mean? Compared to what?

❌ "We ran the experiment" — Can I reproduce it? Where's the script?

❌ "The latency increased" — By how much? Why? What OS mechanism?

✅ "p99 latency increased from 2ms to 15ms when memory limit was reduced to 512MB, because page reclaim (kswapd) was triggered every 100ms"

A Note on Student Diversity

This class has PhD students, Master's students, and undergraduates.

My approach:

- Lectures will cover concepts clearly — no one gets left behind
- Labs will have tiered difficulty (basic requirements + advanced challenges)
- Final project scope can be adjusted based on your level
- Ask questions! There are no dumb questions.

Final Project: System Case Study

Each student (or pair) picks a **"system case"** — a real system exhibiting a behavior worth investigating.

Examples:

- PostgreSQL p99 spike under memory pressure
- Redis latency when competing with noisy neighbor
- nginx container OOM under bursty load
- llama.cpp inference timeout under concurrent requests

Final Project Milestones

Week	Milestone	Deliverable
4	Proposal	1-page problem statement + minimal repro
6	Checkpoint	Tracing evidence (eBPF or perf)
10	Freeze	Clear metric + intervention plan
12	Defense	Presentation + reproduction by peers

Your project is **40% of grade**. Start thinking now.

Grading Breakdown

Component	Weight	What It Includes
Final Project	40%	System case study (4 milestones)
Labs	40%	~8 lab assignments (~5% each)
Participation	20%	In-class quizzes, activities, discussion

No closed-book exams. Your work speaks for itself.

What Counts as Participation?

Earned through **in-class activities during lectures**:

- In-class quizzes (unannounced, short)
- In-class activities and discussions
- Asking and answering questions
- Helping classmates

You must attend lecture to earn participation credit.

12-Week Roadmap (Preview)

Week	Theme	OS Concepts
1	Introduction	Setup + methodology
2	Performance	Memory hierarchy, cache, measurement
3	Concurrency	Threads, scheduling policies
4	eBPF	Tracepoints, kernel observability
5	Containers	Namespaces, cgroups
6	Kubernetes	Resource limits, QoS, eviction

12-Week Roadmap (continued)

Week	Theme	OS Concepts
7	Network(*)	TCP/IP stack, socket buffers
8	Storage	Filesystems, page cache, fsync
9	Security(*)	Capabilities, seccomp, sandboxing
10	LLM Workloads	Memory pressure, inference profiling
11	Agent Runtime	Tool isolation, least privilege
12	Final Defense	Presentations + peer reproduction

Part 2: Environment Setup

We'll spend the rest of Session 1 walking through environment setup.

Why this matters:

- Labs require **kernel-level access** (perf, eBPF, cgroups)
- Different environments = different results = debugging nightmares
- We all need to be on the **same page**

The Two-Environment Model

You'll work with **two environments**:

Environment	Purpose	What Runs Here
Your laptop	Writing code, reports, browsing	Editor, browser, PDF viewer
Ubuntu VM	Running labs, measurements	perf, eBPF, containers, experiments

Why? Your laptop (macOS/Windows) can't run kernel-level tools reliably.

⚠️ Important: What NOT to Use

Environment	Problem	Result
WSL2	Kernel features limited	eBPF/perf may fail silently
Docker	BPF capabilities restricted	Can't access PMU, limited /proc
macOS	Different kernel	No /proc, no perf, no cgroups

The only supported environment: Ubuntu VM (VirtualBox) or native Ubuntu.

Step 1: Install VirtualBox

Download: virtualbox.org

For Intel/AMD (most Windows/Linux):

- Download the standard installer
- Follow the wizard

For Apple Silicon (M1/M2/M3 Mac):

- VirtualBox support is limited
- Consider **UTM** as an alternative
- Ask me or TA if you need help

Step 2: Download Ubuntu ISO

Recommended: Ubuntu 22.04 LTS or Ubuntu 24.04 LTS

Course requirement (hard): Linux kernel **5.15+**

- Check your kernel with: `uname -r`
- Ubuntu 22.04 LTS ships with a 5.15 kernel series by default

Download links:

- Intel/AMD: ubuntu.com/download/desktop
- ARM (Apple Silicon): ubuntu.com/download/server/arm

File size: ~4-5 GB — download before class if possible!

Step 3: Create the VM

In VirtualBox, click "New":

Setting	Recommended Value
Name	`Ubuntu-OS-Course`
Type	Linux
Version	Ubuntu (64-bit)
RAM	4 GB (8 GB if possible)
CPU	2 cores
Disk	25 GB or more

Then attach the Ubuntu ISO and install.

Step 4: Install Ubuntu

1. Boot the VM with ISO attached
2. Follow the installer (defaults are fine)
3. Create a user account (remember your password!)
4. Wait for installation to complete
5. Remove ISO and reboot

Time estimate: 15-30 minutes

Step 5: Install Development Tools

Open a terminal in Ubuntu and run:

```
# Update package list
sudo apt update

# Install basics
sudo apt install -y build-essential git curl wget

# Install perf
sudo apt install -y linux-tools-common linux-tools-$(uname -r)

# Install strace (system call tracing)
sudo apt install -y strace
```

Step 6: Verify Basic Tools

```
# Check GCC
gcc --version

# Check Make
make --version

# Check Git
git --version

# Check perf (needs sudo)
sudo perf stat ls
```

✓ **All should print version numbers or run without errors.**

Step 7: Install eBPF Tools (For Week 4+)

```
# eBPF tracing tools
sudo apt install -y bpftrace bpfcc-tools

# eBPF development libraries
sudo apt install -y clang llvm libbpf-dev libelf-dev

# Kernel headers (needed for eBPF compilation)
sudo apt install -y linux-headers-$(uname -r)
```

Note: If these fail, it's OK — we don't need them until Week 4.

Step 8: Install Optional Tools

```
# Stress testing tools (for generating workloads)
sudo apt install -y stress-ng fio

# Python for data analysis
sudo apt install -y python3 python3-pip
pip3 install matplotlib pandas numpy

# htop (better process viewer)
sudo apt install -y htop
```

Tool Summary: What You'll Use

Tool	Purpose	When
`gcc`	Compile C code	Every lab
`perf`	Performance measurement	Week 2+
`strace`	Trace system calls	Week 2+
`bpftrace`	eBPF tracing	Week 4+
`stress-ng`	Generate CPU/memory load	Week 5+

Step 9: Run the Course Environment Check Script

From the course repo (see Lab 0), run:

```
bash env_check.sh | tee lab0_env_check.txt
```

Goal: no **critical errors** (including kernel < 5.15).

Self-Test: Is Your Environment Ready?

Run these commands. All should work:

```
# 1. Compile a C program
echo 'int main(){return 0;}' > test.c && gcc -o test test.c && ./test && echo "OK"

# 2. Run perf
sudo perf stat ls >/dev/null

# 3. Explore /proc
cat /proc/cpuinfo | head -5

# 4. Run strace
strace -c ls 2>&1 | head -5
```

Self-Test Checklist

Check	Command	Status
GCC works	`gcc --version`	☐
Make works	`make --version`	☐
Perf works	`sudo perf stat ls`	☐
/proc accessible	`cat /proc/cpuinfo`	☐
strace works	`strace -c ls`	☐

All checked = Ready for Lab 0!

Your First C Program

Create `hello.c`:

```c
#include <stdio.h>
#include <unistd.h>

int main() {
    printf("Hello from process %d\n", getpid());
    return 0;
}
```

Compile and run:

```
gcc -o hello hello.c
./hello
```

Your First System Call Trace

```
strace ./hello
```

You'll see something like:

```
execve("./hello", ["./hello"], ...) = 0
...
write(1, "Hello from process 12345\n", 25) = 25
exit_group(0)                             = ?
```

This is what OS does behind the scenes!

Your First Measurement: perf stat

```
sudo perf stat ./hello
```

Output:

```
 Performance counter stats for './hello':
          0.42 msec task-clock
             1      context-switches
             0      cpu-migrations
            54      page-faults
       912,345      cycles
       456,789      instructions
```

You just measured your program's OS-level behavior!

Understanding perf Output

Metric	What It Means
task-clock	CPU time used by your program
context-switches	Times OS paused your program
page-faults	Memory pages allocated or loaded (minor + major faults)
cycles	CPU clock cycles consumed
instructions	Machine instructions executed

This is the foundation of performance analysis!

Lab 0: Submission (Canvas)

Submit ONE ZIP file on Canvas.

- **ZIP name (recommended):** `lab0_<lastname>_<firstname>.zip`
- **Due:** Before Week 2 lecture (see Canvas for the exact timestamp)

Minimum requirement: the ZIP must contain **`lab0_report.md`**.

Lab 0: What Goes Into the ZIP?

File	Required?	What It Is For
`lab0_report.md`	✅ Yes	Your answers + results (the graded report)
`lab0_env_check.txt`	Recommended	Raw output of `bash env_check.sh | tee lab0_env_check.txt`
`hello.c`	Recommended	Your “first program” source code (trace/measure target)
`perf_stat_hello.txt`	Optional	Raw `perf stat ./hello` output (supports reproducibility)
`strace_hello.txt`	Optional	Raw `strace ./hello` output (supports reproducibility)

Notes:

Lab 0: Grading Rubric (90 pts + 10 bonus)

Criterion	Points
Part A (30 pts)	
Environment check passes	20
Issues documented clearly	10
Part B (60 pts)	
strace output included	10
perf stat results (3 runs)	20
Explanations are correct and thoughtful	30
Part C (10 bonus pts)	
Experiment completed	5

Troubleshooting: Common Issues

"command not found"

→ Install the missing package: `sudo apt install <package>`

"Permission denied" with perf

→ Use `sudo`: `sudo perf stat ...`

perf shows "not supported"

→ You might be in a VM with limited PMU access — that's OK, some counters work

"No kernel headers"

→ `sudo apt install linux-headers-$(uname -r)`

Troubleshooting: VirtualBox Issues

VM is very slow

→ Enable VT-x/AMD-V in BIOS, allocate more RAM

No network in VM

→ Check VirtualBox network settings (NAT or Bridged)

Screen resolution stuck

→ Install Guest Additions: `sudo apt install virtualbox-guest-utils`

Can't copy/paste between host and VM

→ Install Guest Additions and enable clipboard sharing

Getting Help

During lab session:

- Ask me or TA — we're here to help!

Outside of class:

- Check course materials first
- Post on course forum (if available)
- Office hours

Don't struggle alone. Environment issues are common and fixable.

Week 2 Preview: What's Coming

Topic: Performance Methodology — Measurement to Mechanism

We will cover:

- **OS concepts**: Memory hierarchy, cache, TLB
- **Measurement**: What makes a valid experiment?
- **Tools**: perf, time, valgrind
- **Lab**: Analyze a sorting algorithm's performance

Summary: What We Covered Today

1. **Course structure:** Lecture + Lab Workshop + Lab Assignment
2. **Grading:** 40% Project + 40% Labs + 20% Participation
3. **Environment:** Ubuntu VM with perf, strace, eBPF tools
4. **First measurement:** `perf stat` and `strace`

Session 2: Let's actually set up your VMs!

Questions?

- About the course?
- About the project?
- About the environment?

Let's get your VMs set up!