CPS 210: Operating Systems
The operating system (OS) is the interface between user applications and the hardware.

An OS implements a sort of virtual machine that is easier to program than the raw hardware.

[McKinley]
Operating Systems: The Classical View

processes

threads

The Kernel
**Key Concepts**

**kernel**
The software component that controls the hardware directly, and implements the core privileged OS functions.

Modern hardware has features that allow the OS kernel to protect itself from untrusted user code.

**thread**
An executing stream of instructions and its CPU register context.

**virtual address space**
An execution context for thread(s) that provides an independent name space for addressing some or all of physical memory.

**process**
An execution of a program, consisting of a virtual address space, one or more threads, and some OS kernel state.
Operating Systems: The Classical View

- Processes in private virtual address spaces
- System call traps
- \( \ldots \) and upcalls (e.g., signals)
- Shared kernel code and data in shared address space
- The kernel sets up process execution contexts to "virtualize" the machine.
- Threads or processes enter the kernel for services.

CPU and devices force entry to the kernel to handle exceptional events.
The basic issues/questions in this course are *how to*:  

- allocate memory and storage to multiple programs?  
- share the CPU among concurrently executing programs?  
- *suspend* and *resume* programs? 
- share data safely among concurrent activities? 
- protect one executing program’s storage from another? 
- protect the code that implements the protection, and mediates access to resources? 
- prevent rogue programs from taking over the machine? 
- allow programs to interact safely?
The OS and User Applications

The OS defines a framework for users and their programs to coexist, cooperate, and work together safely, supporting:

- concurrent execution/interaction of multiple user programs
- shared implementations of commonly needed facilities
  "The system is all the code you didn’t write."
- mechanisms to share and combine software components
  *Extensibility*: add new components on-the-fly as they are developed.
- policies for safe and fair sharing of resources
  *physical* resources (e.g., CPU time and storage space)
  *logical* resources (e.g., data files, programs, mailboxes)
Overview of OS Services

**Storage**: primitives for files, *virtual memory*, etc.
Control devices and provide for the “care and feeding” of the memory system hardware and peripherals.

**Protection** and security
Set boundaries that limit damage from faults and errors.
Establish user identities, priorities, and accountability.
Mediate/control access for logical and physical resources.

**Execution**: primitives to create/execute programs
support an environment for developing and running applications

**Communication**: “glue” for programs to interact
The OS and the Hardware

The OS is the “permanent” software with the power to:

- control/abstract/mediate access to the hardware
  - CPUs and memory
  - I/O devices
- so user code can be:
  - simpler
  - device-independent
  - portable
  - even “transportable”
Architectural Foundations of OS Kernels

- One or more privileged execution modes (e.g., kernel mode)
  protected device control registers
  privileged instructions to control basic machine functions

- System call trap instruction and protected fault handling
  User processes safely enter the kernel to access shared OS services.

- Virtual memory mapping
  OS controls virtual-physical translations for each address space.

- Device interrupts to notify the kernel of I/O completion etc.
  Includes timer hardware and clock interrupts to periodically return
  control to the kernel as user code executes.

- Atomic instructions for coordination on multiprocessors
Introduction to Virtual Addressing

User processes address memory through virtual addresses.

The kernel and the machine collude to translate virtual addresses to physical addresses.

The kernel controls the virtual-physical translations in effect for each space.

The machine does not allow a user process to access memory unless the kernel “says it’s OK”.

The specific mechanisms for implementing virtual address translation are machine-dependent.
CPS 210, Spring 2002

Part I
• The stuff you should already know.

Part II
• The stuff you should learn.

Part III
• The questions we’re trying to answer now through ongoing research in “systems”.

Tanenbaum: undergrad OS text.
Research papers: 10-12 to 20.
CPS 210: Part I

Concurrency and synchronization

- Threads and processes, race conditions, mutexes, semaphores, coordination, condition variables, starvation and deadlock
- Everyone has to know this stuff.
- A few lectures, problem set + exam 1/29

Classical operating systems

- Processes and the kernel, system calls, kernel services, file I/O, virtual memory.
- A few more lectures.
- New this semester: the infamous Nachos labs: 2/5 and 2/19.
Nachos is designed to look, feel, and crash like a “real” OS. Both the Nachos “OS” and test programs run together as an ordinary process on an ordinary Unix system (Solaris).
Nachos runs *real* user programs on a *simulated* machine.

MIPS simulator in Nachos executes *real* user programs.

The real OS is treated as part of the underlying hardware.
Nachos: A Peek Under the Hood

SPIM
MIPS emulator

ExceptionHandler()

Nachos kernel

Machine::Run()

fetch/execute
examine/deposit

process page tables

sp

Machine object

registers

page table

memory

User space
MIPS instructions
executed by SPIM

shell

MIPS instructions
executed by SPIM

cp

SaveState/RestoreState
examine/deposit

save state

restore state
Overview of the Nachos Labs

Lab 1

• Synchronization primitives “from scratch”.
  Uniprocessor kernel-mode mutexes and condition variables.

• Kernel process management system calls.
  Like Unix fork/exec/exit/wait with simple virtual memory.

Lab 2

• Interprocessor communication using pipes and I/O descriptors.

• Simple command shell and user programs.

• Paged virtual memory with page cache management.
Secrets of the Nachos Labs

*It’s the thought that counts.*

- Think before you design it.
- Think before you code it.
- Think before you run it.
- Think before you debug it.

The time needed to conceive and write the code is moderate, but debugging time is potentially unbounded.
CPS 210: Part II

Classical OS view: advanced topics

• Deconstructing the OS
• Servers, network storage, RAID, end-system networking
• Resource management, continuous media
• Quantitative system performance
• Reliability and robustness

“Systems” as an experimental research discipline

• Research vs. development
• Styles of research
• Goals and methodology
• What/how to measure?

Performance? Dependability? Performability?
Effect of Hardware on Software

Advances in hardware technology drive software/OS changes.

In the beginning, humans were cheap, computers were expensive.
- centralized computers, batch processing, no direct user interaction

Now computers are cheap.
- dedicated workstations, PCs, and servers in a networked world
- emphasize ease-of-use and effective interaction over raw performance

Faster and cheaper hardware is the defining force in systems.

- OS interfaces and policies depend on relative speed and cost of the different components.
- E.g., faster networks allow tighter coupling of clustered systems.

[McKinley]
History Lesson

From 1950 to now (50-year history of computing) we’ve seen a 3-9 order-of-magnitude change in almost every component.

- MIPS: from 0.5 in 1983 to 500 in 1998.
- Price/MIP: from $100K in 1983 to $300 today.
- Memory: 1 MB memories in 1983 to 1 GB memories today.
- Network: 10 Mb/s in 1983 to 1 Gb/s or more today.
- Secondary store: 1 GB in 1983 to 1 TB today.
- Virtual address space: 32 in 1983 to 64 today.

Compare to:

transportation: horseback to the Concorde in 200 years

[McKinley]
The World Today

- Internet
- LAN/SAN Network
- Servers
  - database
  - file
  - web
  -...
- Server farms (clusters)
- mobile devices
- Internet appliances
- desktop clients

Systems & Architecture
CPS 210: Part III

Contemporary research directions

• Incorporating processing into network and storage elements.
  Active storage, extensible switches and active networks, active proxies, firewall appliances and other intermediaries, etc.

• Server-based computing and computing utilities.

• Mobile computing and power management.

• Harnessing massive storage resources.

• Massively decentralized systems.
  Massive scale and robustness: sensor networks, peer-to-peer.
  New evaluation methodologies.
E-Track, G-Track, and Grading

Problem sets and labs
  • 3-5: 45%

Exams
  • 3: 45%

Exit interview, subjective factors
  • 10%

E-track
  • EC on labs and problem sets
  • Semester project
  • Does not affect quals pass!