• Class web page: https://cs.uwaterloo.ca/cs350/
  ▶ All assignments and handouts

• My web page: https://rcs.uwaterloo.ca/~ali/
  ▶ Lecture notes

• Textbooks
  ▶ Operating System Concepts
  ▶ Operating Systems: Three Easy Pieces
• Q&A through Piazza (see class website)

• Quizzes and Final through Waterloo LEARN

• Four projects due throughout term
Course Goals: Introduce you to Systems

- Operating Systems
- Distributed Systems
- Networking
- Internet of Things
- Computer Architecture
- Embedded Systems
- Database Systems
- Systems and Machine Learning
- ...

...
Course Goals: Practical Understanding of OSes

• Introduce you to operating systems
  ▶ Every computer, phone and watch runs an OS
  ▶ Makes you a more effective programmer
  ▶ How the OS affects your software

• General systems concepts
  ▶ Concurrency, memory management, and I/O
  ▶ Security and protection
  ▶ Tools for software performance

• Practical skills
  ▶ Learn to work with large code bases
  ▶ My lectures: industry and research experience
What is an operating system?

- Layer between applications and hardware
- Makes hardware useful to the programmer
- Usually: Provides abstractions for applications
  - Manages and hides details of hardware
  - Accesses hardware through low/level interfaces unavailable to applications
- Often: Provides protection
  - Prevents one process/user from clobbering another
Why study operating systems?

- Operating systems are a maturing field
  - Most people use a handful of mature OSes
  - Hard to get people to switch operating systems
  - Hard to have impact with a new OS
- High-performance servers are an OS issue
  - Face many of the same issues as OSes
- Resource consumption is an OS issue
  - Battery life, radio spectrum, etc.
- Security is an OS issue
  - Security requires a solid foundation
- New “smart” devices need new OSes
- Web browsers, databases, and game engines look like OSes
Course topics

- Threads & Processes
- Concurrency & Synchronization
- Scheduling
- Virtual Memory
- I/O
- Disks, File systems, Network file systems
- Protection & Security
- Virtual machines
- Will often use Unix as the example
  - Most OSes heavily influenced by Unix (e.g. OS161)
  - Windows is a notable exception
Primitive Operating Systems

- Just a library of standard services (no protection)
  - Standard interface above hardware-specific drivers, etc.
- Simplifying assumptions
  - System runs one program at a time
  - No bad users or programs (often bad assumption)
- Problem: Poor utilization
  - ...of hardware (e.g., CPU idle while waiting for disk)
  - ...of human user (must wait for each program to finish)
Multitasking

- Idea: Run more than one process at once
  - When one process blocks (waiting for user input, IO, etc.) run another process
- Problem: What can ill-behaved process do?

**Operating System**
- gcc
- emacs

**Hardware:** CPU, Memory and Devices
Multitasking

- Idea: Run more than one process at once
  - When one process blocks (waiting for user input, IO, etc.) run another process
- Problem: What can ill-behaved process do?
  - Go into infinite loop and never relinquish CPU
  - Scribble over other processes’ memory to make them fail
- OS provides mechanisms to address these problems
  - Preemption – take CPU away from looping process
  - Memory protection – protect process’s memory from one another
• Many OSes use *protection* to serve distrustful users/apps
• Idea: With \( N \) users, system not \( N \) times slower
  ▶ User demand for CPU is bursty
• What can go wrong?
Multi-user OSes

- Many OSes use *protection* to serve distrustful users/apps
- Idea: With $N$ users, system not $N$ times slower
  - User demand for CPU is bursty
- What can go wrong?
  - Users are gluttons, use too much CPU, etc. (need policies)
  - Total memory usage greater than in machine (must virtualize)
  - Super-linear slowdown with increasing demand (thrashing)
Protection

• Mechanisms that isolate bad programs and people

• Pre-emption:
  ▶ Give application a resource, take it away if needed elsewhere

• Interposition/mediation:
  ▶ Place OS between application and “stuff”
  ▶ Track all pieces that application allowed to use (e.g., in table)
  ▶ On every access, look in table to check that access legal

• Privileged & unprivileged modes in CPUs:
  ▶ Applications unprivileged (unprivileged user mode)
  ▶ OS privileged (privileged supervisor/kernel mode)
  ▶ Protection operations can only be done in privileged mode
• Most software runs as user-level processes (P[1-4])
• OS kernel runs in *privileged* mode *(shaded)*
  ▶ Creates/deletes processes
  ▶ Provides access to hardware
Applications can invoke kernel through **system calls**

- Special instruction transfers control to kernel
- ...which dispatches to one of few hundred syscall handlers
System calls (continued)

- Goal: Do things app. can’t do in unprivileged mode
  - Like a library call, but into more privileged kernel code

- Kernel supplies well-defined system call interface
  - Applications set up syscall arguments and trap to kernel
  - Kernel performs operation and returns result

- Higher-level functions built on syscall interface
  - printf, scanf, gets, etc. all user-level code

- Example: POSIX/UNIX interface
  - open, close, read, write, …
System call example

- Standard library implemented in terms of syscalls
  - `printf` – in libc, has same privileges as application
  - calls `write` – in kernel, which can send bits out serial port