CS350: Operating Systems

Lecture 1: Introduction

University of Waterloo

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 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

Administrivia

- Class web page: https://student.cs.uwaterloo.ca/~cs350/W24/
  - All assignments, lecture notes, handouts, policies
- Instructor web page:
  - Bernard Wong: https://cs.uwaterloo.ca/~bernard/
  - Hong Zhang: https://hongzhangblaze.github.io
- Textbooks
  - *Operating System Concepts*
  - *Operating Systems: Three Easy Pieces*

Administrivia Continued

- Q&A through Piazza
- Midterm is scheduled for March 7, 2024
  - Suggestion: DO NOT skip midterm using your short-term absence
- Final will be announced later
- Three projects due throughout the term

Grading Scheme

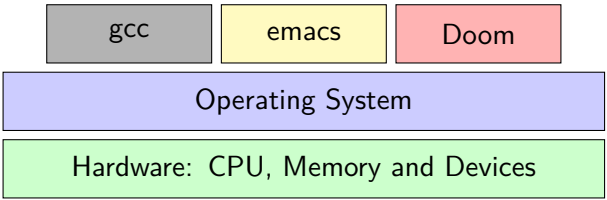
Component	Weight	Description
A	40%	Your weighted average grade on assignments as a percentage.
M	20%	Your grade on the midterm as a percentage.
F	40%	Your final exam grade as a percentage.

```
Normal  = 0.40*A + 0.20*M + 0.40*F
E = (0.20*M + 0.40*F)/0.60

if (A < 50 or E < 50) then
    Grade = min(Normal, 46)
else
    Grade = Normal
```

What is an operating system?

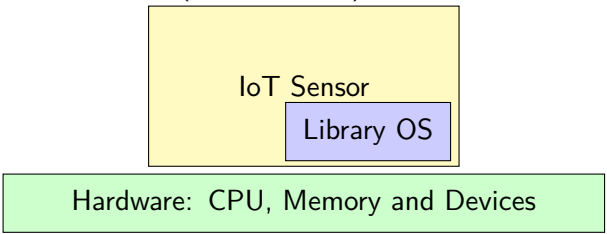
- Layer between applications and hardware



- 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

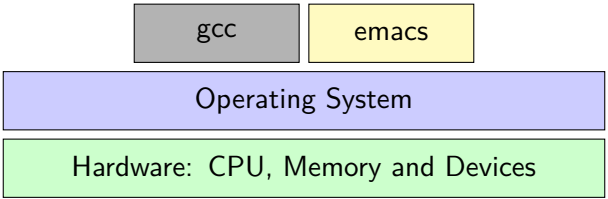
Primitive Operating Systems

- Just a library of standard services (no protection)



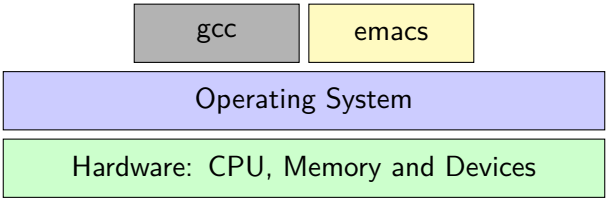
  - Standard interface above hardware-specific drivers, etc.
- Simplifying assumptions
  - System runs one program at a time
  - No bad users or programs (often bad assumption)
- Problem: Poor utilization
  - . . . of hardware (e.g., CPU idle while waiting for disk)
  - . . . of human user (must wait for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

Multitasking



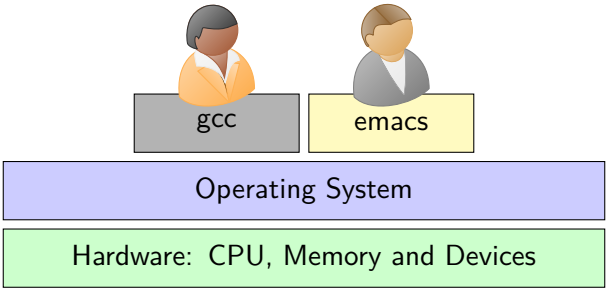
- Idea: Run more than one process at once
  - When one process blocks (waiting for user input, IO, etc.) run another process
- Problem: What can ill-behaved process do?

Multitasking



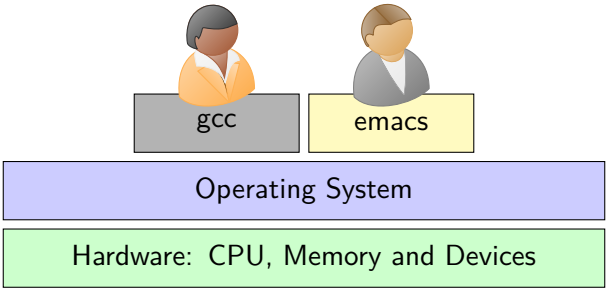
- Idea: Run more than one process at once
  - When one process blocks (waiting for user input, IO, etc.) run another process
- Problem: What can ill-behaved process do?
  - Go into infinite loop and never relinquish CPU
  - Scribble over other processes' memory to make them fail
- OS provides mechanisms to address these problems
  - *Preemption* – take CPU away from looping process
  - *Memory protection* – protect process's memory from one another

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?

Multi-user OSes

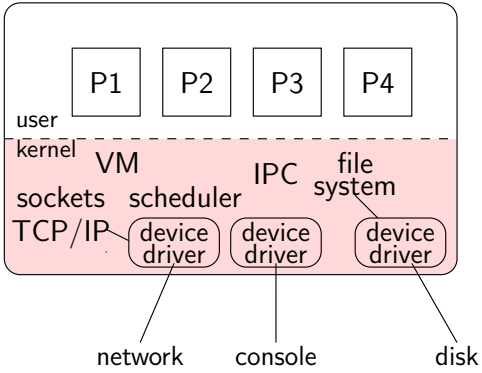


- Many OSes use *protection* to serve distrustful users/apps
- Idea: With N users, system not N times slower
  - User demand for CPU is bursty
- What can go wrong?
  - Users are gluttons, use too much CPU, etc. (need policies)
  - Total memory usage greater than in machine (must virtualize)
  - Super-linear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

Protection

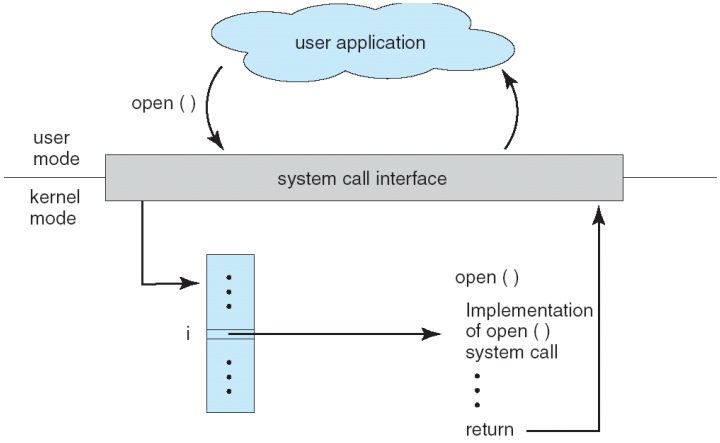
- Mechanisms that isolate bad programs and people
- Pre-emption:
  - Give application a resource, take it away if needed elsewhere
- Interposition/mediation:
  - Place OS between application and "stuff"
  - Track all pieces that application allowed to use (e.g., in table)
  - On every access, look in 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

Typical OS structure



- Most software runs as user-level processes (P[1-4])
- OS *kernel* runs in *privileged* mode (shaded)
  - Creates/deletes processes
  - Provides access to hardware

System calls

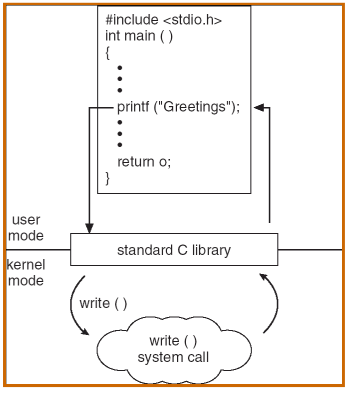


- Applications can invoke kernel through *system calls*
  - Special instruction transfers control to kernel
  - . . . which dispatches to one of few hundred syscall handlers

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