CIS 415: Operating Systems
Introduction

Prof. Kevin Butler
Spring 2011
Administrative

- CIS 415: Operating Systems (that’s what this is)
  - Tuesday/Thursday, 12:00-13:20
  - 105 Deady Hall (here)
- Ensure that you’re registered for this course and a recitation section
- Instructor (me): Kevin Butler
  - 245 Deschutes Hall
  - Office Hours: Wednesday 11AM-12PM unless otherwise specified and by appointment
  - email: butler@cs.uoregon.edu
Admin (part 2)

- **TA:** Kaveh Kazemi
  - kaveh@cs.uoregon.edu
  - Office hours: Wed. 1300-1400, Fri. 1300-1400
- **Mailing list**
  - cis415-sp11@cs.uoregon.edu (you should be on)
- **Textbook**
  - Silberschatz et al., *Operating System Concepts*, 8/e
  - Recommended: UNIX/C programming books
- **Web page:**
  - http://www.cs.uoregon.edu/classes/11S/cis415/
Course Structure

- Lectures (what you’re in now)
  - focus on core OS content, also worksheets and quizzes (in-class exercises)

- Recitations (supplemental help by the TA)
  - programming assignment help, tutorials, practice, material on C and UNIX, threads, signals, etc

- Grading
  - 15% Quizzes & Homeworks
  - 20% Midterm (in-class, May 3)
  - **30% Programming Projects (do not procrastinate)**
  - 35% Final (8:00 June 7)
This course ... 

- This course is a systems course covering fundamental and applied topics in computer operating systems, including:
  - system calls and interfaces, processes, concurrent programming, resource scheduling and management (CPU, memory, I/O), multicore systems, virtual memory, deadlocks, distributed systems, networks, security, other topics as time permits
You need to understand ...

- Computer organization and architecture (CIS 314)
- Basic data structures and algorithms
- C and UNIX programming environments (you can learn this as you go)
- How to look things up from source material and books
Goals

- My goal: *to provide you with the tools to understand fundamentals of modern operating systems.*
  - Basic technologies
  - Engineering/research trade-offs
  - In-depth practical OS knowledge and programming experience

• *This is going to be a hard course.* The key to success is sustained effort. Failure to keep up with readings and assignments will likely result in poor grades, and ultimately little understanding of the course material.

• Pay-off: fundamental knowledge, marketable skills
Course Calendar

- The course calendar as all the relevant readings, assignments and test dates
- The calendar page contains electronic links to online papers assigned for course readings.

Please check the website frequently for announcements and changes to the schedule. Students are responsible for any change on the schedule.
What’s an Operating System?

- Consider you want to do the following:
  - print “this is not a printout” on the printer
  - terminate

- Simple!

```c
main()
{
    printf("this is not a printout\n");
}
```
Without an OS

- Get printer manual
  - figure out how to send messages to it
- Write the program
  - put the character string “this is not a printout” in a memory buffer
  - do the stuff printer requires to send buffer to it
  - go into endless loop
    - wait for someone to turn off computer eventually
- Get hold of a computer
  - it has to be all yours
- Translate your program into machine code
- Figure out how to get program into memory
  - font panel switches
  - burn a ROM
  - punch cards
- Somehow start program
- Turn off computer when done
With an OS

- Put program in file
  - put character string “this is not a printout” in a memory buffer
  - Issue *system call* to send buffer to printer
- Compile the program and tell OS to run the program file
  - That’s everything!
  - What’s a system call?
An Operating System

- Program that acts as intermediary between a computer user and the computer hardware
  - i.e., an abstract interface to programs

- Goals as a resource manager:
  - Execute user programs and making solving user problems easier
  - Make the computer system convenient and safe to user
  - Use the computer hardware in an efficient manner
Things your OS does

- **Processes**
  - hides programs from one another

- **Traffic cop**
  - resource management
  - who gets to run, when?

- **Memory management**
  - protection from other programs’ mistakes

- **Security**
  - protection from other programs’ malice

- **System call interface**
  - abstract, simplified interface to services
  - like a function library but communicates with the OS

- **Portability**
  - programs don’t have to take into account details of their environment

- **Device management**

- **Communication**
  - between processes
  - to devices & networks
Related to an OS

- GUls and user interfaces
- Applications (e.g., web browser)
- Compilers
- Libraries
- These are implemented as user-space programs but not really the core of the OS
  - Linux
  - GNU/Linux
  - Ubuntu vs Debian vs Fedora vs...
Operating System History

- 1950s: Simplify operators’ job
- 1960s: Structure, concepts, everything
- 1970s: Small and flexible (UNIX)
- 1980s: Individual user systems (PCs)
- 1990s: Internet, Windows
- 2000s: Security
Operating Systems (1950s)

- Primitive systems
  - Little memory, programs stored on tape
- Single user
  - Batch processing
  - Computer executes one function at a time
- No overlap of I/O and computation
Operating Systems (1960s)

- Multiprogramming
  - Timesharing
  - Multiple programs run concurrently
- Many operating systems concepts invented
  - Virtual memory, Hierarchical File Systems, Synchronization, Security and many more
- End up with slow, complex systems on limited hardware (Multics)
Operating Systems (1970s)

- Becoming more available
  - UNIX
    - First OS written in a high-level language
- Becoming more flexible
  - Extensible system
  - Community forms beyond developers
- Performance focus
  - Optimization of algorithms from 1960s
Operating Systems (1980s)

- Critical Mass Reached
  - A variety of well-known systems, concepts
  - UNIX fragments
- PC Emerges
  - Simple, single user, no network
  - Simple OSes: DOS
- Graphical User Interfaces
  - X Windows and Apple Macintosh
Operating Systems (1990s)

- Connect to Internet
  - “Real OSes” for PCs
    - NT/2000+, Linux, eventually Mac OS X

- Server Systems Galore
  - Mainframes even reemerge

- Complex Systems and Requirements
  - Parallel, Real-time, Distributed, etc.
Operating Systems (2000s)

- Challenges facing us now include
  - Security
  - Multicore
  - Ubiquitous
  - Virtual Machines
  - Embedded
  - Mobile
OS Functions

- What does it do?
  - Mostly behind the scene

- Example
  - Page Fault Handling
Page Fault Handling

- **Cause:** Access a virtual memory location not backed by a physical page
- **Trap generated by hardware**
- **Handler in OS determines how to obtain memory**
  - If page is still on disk, then handler
    - allocates physical page
    - makes I/O request to disk via file system and driver
- **Driver copies page from disk into new physical page**
- **OS restarts the process at the trapped instruction**
Page Fault Handling

- There are multiple processes, so the OS has to make trade-offs
  - What is there are no physical pages available?
  - The disk is slower than memory access, so how to process?
  - There may be multiple outstanding disk requests, so what order should they be processed?
  - How does the OS interact with hardware effectively?
  - Many others…
Learning About OSes

- OS has a zillion protocols like page fault handling
  - You will need to know them
- OS designers add layers of indirection concepts to simplify programming (e.g., virtual memory)
  - You will need to understand these concepts
- The design of protocols using these concepts involves trade-offs (e.g., optimize disk read performance)
  - You will need to understand why OS protocols are written the way that they are
Some Basics
Storage Hierarchy

<table>
<thead>
<tr>
<th>Registers</th>
<th>Fast</th>
<th>Expensive</th>
<th>Small</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU cache</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>on-board cache</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>main memory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>secondary memory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slow secondary memory</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Slow
- Cheap
- Large

Volatile

Non-Volatile
Device Input/Output

- CPU
- Memory: Data and Instructions
- Device

Connections:
- I/O Request
- Data
- Interrupt
- Data: DMA

Diagram shows the flow of data and control signals between CPU, memory, and device.
Scheduling

- Determine which task to perform given that there are:
  - Multiple user processes
  - Multiple hardware components
- Provide effective performance
  - Responsive to users, CPU utilization
- Provide fairness
  - Do not starve low priority processes
Security

- Control access to shared resources
  - E.g., Files

- Ensure that only authorized processes can access a file
  - User’s process can access user’s files
  - Most file systems enable sharing among users
  - Some operating systems represent devices as files
Next Time

▪ Next class
  ‣ Background on Computer Systems

▪ Do the following this week:
  ‣ Set up an Ubuntu virtual machine on the room 100 computers in your NFS space (or with a portable hard drive) using Virtualbox
  ‣ Email me if you haven’t received an email welcoming you to the course
  ‣ More information about you (background, interests, etc)
  ‣ Register for Security Day