CS 134:
Operating Systems
Course Introduction
Overview

Class Introduction
  Administrivia
  Course Purpose

Real and Not-So-Real Systems
  OS/161

Class Rules

Resources

What is an Operating System?
  Some Ideas

Taxonomies
Prerequisites: CS105
  - Highly recommend CS 105, 140, etc.

Web page: http://www.cs.hmc.edu/~geoff/cs134

Email: geoff@cs.hmc.edu

Office: Olin 1245
  - Office hours on Web page
  - Will be changed in first few weeks of term
You’ve signed up for this course, but are you clear what it is about?

- What are you hoping to learn, and why it matters?
- What’s the overlap with other courses you have taken and will take?
- What OS-related topics do you know from taking 105?

Develop your answers

- Individually (3 minutes)
- In a group (3 minutes)
Discuss with the people around you:

► A topic you want to learn more about...
► A skill you’d like to better develop...

Compare notes on prior knowledge with the people around you...
What are you going to do to get the most out of this class?

- In class?
- Outside of class?

Develop answers

- Individually (3 minutes)
- In a group (3 minutes)
Design and implementation of operating systems, including processes, memory management, synchronization, scheduling, protection, filesystems, and I/O. These concepts are used to illustrate wider concepts in the design of other large software systems, including simplicity; efficiency; event-driven programming; abstraction design; client-server architecture; mechanism vs. policy; orthogonality; naming and binding; static vs. dynamic, space vs. time, and other tradeoffs; optimization; caching; and managing large codebases. Group projects provide experience in working with and extending a real operating system.
Design and implementation of operating systems, including processes, memory management, synchronization, scheduling, protection, filesystems, and I/O. These concepts are used to illustrate wider concepts in the design of other large software systems, including simplicity; efficiency; event-driven programming; abstraction design; client-server architecture; mechanism vs. policy; orthogonality; naming and binding; static vs. dynamic, space vs. time, and other tradeoffs; optimization; caching; and managing large codebases. Group projects provide experience in working with and extending a real operating system.
Design and implementation of operating systems, including processes, memory management, synchronization, scheduling, protection, filesystems, and I/O. These concepts are used to illustrate wider concepts in the design of other large software systems, including simplicity; efficiency; event-driven programming; abstraction design; client-server architecture; mechanism vs. policy; orthogonality; naming and binding; static vs. dynamic, space vs. time, and other tradeoffs; optimization; caching; and managing large codebases. Group projects provide experience in working with and extending a real operating system.

Are you looking forward to working with a real operating system?
<table>
<thead>
<tr>
<th>System</th>
<th>Source files</th>
<th>Lines of code</th>
<th>Number of functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC (MIPS)</td>
<td>1060</td>
<td>837,353</td>
<td>5647</td>
</tr>
<tr>
<td>GDB (MIPS)</td>
<td>2463</td>
<td>1,374,680</td>
<td>6859</td>
</tr>
<tr>
<td>Linux Kernel (2.4.x)</td>
<td>684 + 10,040</td>
<td>402,534 + 4,079,951</td>
<td>6366 + 39,552</td>
</tr>
<tr>
<td>Linux Kernel (2.6.x)</td>
<td>19,779</td>
<td>8,230,479</td>
<td>19,498 + 67,049</td>
</tr>
<tr>
<td>Mac OS X (Panther)</td>
<td>1895 + . . .</td>
<td>751,440 + . . .</td>
<td>8200 + 43,871</td>
</tr>
<tr>
<td>Mac OS X (Leopard)</td>
<td>2211 + . . .</td>
<td>1,077,164 + . . .</td>
<td>9740 + 43,810</td>
</tr>
</tbody>
</table>
## Real Systems

<table>
<thead>
<tr>
<th>System</th>
<th>Source files</th>
<th>Lines of code</th>
<th>Number of functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC (MIPS)</td>
<td>1060</td>
<td>837,353</td>
<td>5647</td>
</tr>
<tr>
<td>GDB (MIPS)</td>
<td>2463</td>
<td>1,374,680</td>
<td>6859</td>
</tr>
<tr>
<td>Linux Kernel (2.4.x)</td>
<td>684 + 10,040</td>
<td>402,534 + 4,079,951</td>
<td>6366 + 39,552</td>
</tr>
<tr>
<td>Linux Kernel (2.6.x)</td>
<td>19,779</td>
<td>8,230,479</td>
<td>19,498 + 67,049</td>
</tr>
<tr>
<td>Mac OS X (Panther)</td>
<td>1895 + . . .</td>
<td>751,440 + . . .</td>
<td>8200 + 43,871</td>
</tr>
<tr>
<td>Mac OS X (Leopard)</td>
<td>2211 + . . .</td>
<td>1,077,164 + . . .</td>
<td>9740 + 43,810</td>
</tr>
<tr>
<td>CS 70 Noronic</td>
<td>14</td>
<td>1702</td>
<td>55</td>
</tr>
</tbody>
</table>
What is OS/161?

System/161 simulates a real machine
- MIPS processor
- Bus with several I/O devices (Serial I/O, Disk Controller, etc.)
- “Remote” debugging support

OS/161 runs on System/161
- Unix-like
- Working
- Unfinished (lots there, lots for you to do...)

OS/161 runs on System/161
- Unix-like
- Working
- Unfinished (lots there, lots for you to do...)

2012-12-06
<table>
<thead>
<tr>
<th>System</th>
<th>Source files</th>
<th>Lines of code</th>
<th>Number of functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC (MIPS)</td>
<td>1060</td>
<td>837,353</td>
<td>5647</td>
</tr>
<tr>
<td>GDB (MIPS)</td>
<td>2463</td>
<td>1,374,680</td>
<td>6859</td>
</tr>
<tr>
<td>Linux Kernel (2.4.x)</td>
<td>684 + 10,040</td>
<td>402,534 + 4,079,951</td>
<td>6366 + 39,552</td>
</tr>
<tr>
<td>Linux Kernel (2.6.x)</td>
<td>19,779</td>
<td>8,230,479</td>
<td>19,498 + 67,049</td>
</tr>
<tr>
<td>Mac OS X (Panther)</td>
<td>1895 + ...</td>
<td>751,440 + ...</td>
<td>8200 + 43,871</td>
</tr>
<tr>
<td>Mac OS X (Leopard)</td>
<td>2211 + ...</td>
<td>1,077,164 + ...</td>
<td>9740 + 43,810</td>
</tr>
<tr>
<td>CS 70 Noronic</td>
<td>14</td>
<td>1702</td>
<td>55</td>
</tr>
<tr>
<td>OS/161</td>
<td>144</td>
<td>19,124</td>
<td>537</td>
</tr>
</tbody>
</table>
OS/161 is written in C. If you need a refresher, see:

http://people.cs.uchicago.edu/~iancooke/osstuff/ccc.html

You cannot understand all of OS/161 all at once

- Real programs are like this
Basic rules

- Done in pairs, but not necessarily “pair programming”
- Plan *ahead of time* when you will get together
- Plan first before coding, decide who will do what
- Document who wrote what
- Understand *all code* your pair develops
- Don’t be a jerk
You can

- Talk to other members of the class about assignments and project work

You may not

- Use an answer someone else has told you without understanding it
- Misrepresent other people's work as your own
- Use the Internet to find answers to assignments

This slide has animations.
Honor Code

You **should**

- Talk to other members of the class about assignments and project work

This slide has animations.
You **should**

- Talk to other members of the class about assignments and project work

**“In your head” rule:**

*If you discuss a problem with someone else, you must leave with everything in your head. You can’t take away anything on paper or electronically.*

You **may not**

- Use an answer someone else has told you without understanding it
- Misrepresent other people’s work as your own
- Use the Internet to find answers to assignments
After an assignment is submitted, I may ask your classmates to

- Review your patch
- Rank your patch
- Use your patch

The winning patch does need to *work*. You wrote it, you support it.
Prof. O’Neill weighted the course components as follows. I plan to follow that weighting in broad outline, but reserve the right to tinker as necessary. (In particular, there isn’t likely to be a Wiki.)

- 48% Assignments
- 9% Patch review
- 12% Midterm
- 18% Final
- 5% Class Participation
- 5% Wiki Participation
- 3% In-class Topic Presentations
Besides the textbook

- **Me:** Olin 1245 or cs134help@cs.hmc.edu
- **Website:** http://www.cs.hmc.edu/~geoff/cs134/
- Other members of the class

*Don’t be afraid to ask for help!*
You must make sure you’re on

▶ cs-134-l@hmc.edu

Mail listkeeper@hmc.edu with help in body for more details
Knuth “provided platform” for homework

- Submit code from Knuth (early and often!)
- Can use ssh to log in remotely
If you have your own machine, you can use that too. But you'll need to:

- Install (following provided instructions)
  - System/161
  - OS/161 toolchain
- Sync your code onto Knuth to submit it
What is an Operating System Anyway?

Class Exercise: Devise three separate definitions. Discuss.
Provide useful functionality to programs:

- Prevent duplicated work
- Promote reuse
What is an Operating System?

Some Ideas

It's a Control Program

Provide the rules for the how the machine will operate:

- Control the operation of the I/O devices
- Ensure smooth running of the machine
What is an Operating System?

Some Ideas

It's an Abstraction Layer

Make the machine “nicer”, easier to program, higher level...

- Hide some of the idiosyncrasies of the machine
- Provide functionality the underlying machine doesn’t have
It's an Abstraction Layer

Make the machine “nicer”, easier to program, higher level…

- Hide some of the idiosyncrasies of the machine
- Provide functionality the underlying machine doesn’t have
OS provides an *environment*
This environment can be seen as a “new machine”...

Hardware
- Core OS
- OS Libraries
- OS Utilities
- Application
  — Physical machine
  — Virtual machine
What is an Operating System?

Some Ideas

It's a Protection Layer

Make the machine more robust—less scope for a bug to have devastating consequences

- OS does everything programs can’t be trusted to do
- OS makes programs play nice with others
OS provides the mechanisms to enforce various policies
OS provides the mechanisms to enforce various policies

Class Exercise: Examples?
The operating system manages physical resources:

- Processor
- Memory
- Storage devices
- Network devices

etc...
The operating system manages virtual resources:

- Processes
- Files
- Users
- Network connections
- Windows
- etc...
Many operating systems are sold by commercial companies

- Market vs. technical considerations
- The operating system is what comes in the box marked "operating system"
Different computer systems ask different things from their OS
Different computer systems ask different things from their OS

Class Exercise: Give some dimensions across which computer systems vary
Partial Taxonomy of Computer Systems

Different computer systems ask different things from their OS:

- Special-purpose ↔ General-purpose
- Single-user ↔ Multi-user
- Non–Resource-sharing ↔ Resource sharing
- Single processor ↔ Multiprocessor
- Stand alone ↔ Networked
- Centralized ↔ Distributed
- Batch ↔ Interactive
- Deadline-free ↔ Real-time
- Insecure ↔ Secure
- Symmetric ↔ Asymmetric
- Simple ↔ Complex
- Small ↔ Large
- Inexpensive ↔ Expensive
- etc.