Introduction

CS 351: Systems Programming
Michael Saelee <lee@iit.edu>
Michael (Sae) Lee

- lee@iit.edu
- http://moss.cs.iit.edu
- Office: SB 226A
Agenda

- Syllabus & Administrivia
- Course overview ("Systems Programming")
§Syllabus
Prerequisites

- “substantial” programming experience
- data structures: concepts & implementation
- basic run-time analysis (big O)
- knowledge of (any) assembly language
- computer organization essentials
- computer organization essentials:
  - data representation (binary, two’s comp, f.p. inaccuracy, etc.)
  - von Neumann model
    - CPU, memory, I/O
  - stack usage / conventions
Online resources

1. Course website
   moss.cs.iit.edu/cs351
   - static information
   - lecture calendar, lab writeups, slides, screencasts, links, etc.
Online resources

2. Piazza: discussion forum
   - all class-related questions
   - monitored by TAs
   - scales *way* better than e-mail
   - announcements, links to additional readings & resources
Online resources

3. Blackboard

- only for grade reporting!
Online resources

4. Vimeo channel: screencasts
- vimeo.com/channels/cs351
- walkthroughs & tutorials (check before starting labs!)
Textbooks

  - Brian W. Kernighan
  - Dennis M. Ritchie

  - Bryant
  - O’Hallaron

- *The Linux Programming Interface* (A Linux and UNIX® System Programming Handbook)
  - Michael Kerrisk
Grading

- 50% Labs
- 25% Midterm exam
- 25% Final exam
  - exam scores normalized to 70%
  - need ≥ 50% on both exams to pass
Grade Scale

```c
char letter_grade(float score) {
    if (score >= 90.0) return 'A';
    else if (score >= 80.0) return 'B';
    else if (score >= 70.0) return 'C';
    else if (score >= 60.0) return 'D';
    else return 'E';
}
```
Labs

- fairly substantial machine problems
  - 100-1000 LOC
- real-world application of concepts covered in lecture & textbook
- 1-3 weeks allotted for each
Course Overview
“Systems Programming”

system  |ˈsɪstəm|

noun
1 a set of connected things or parts forming a complex whole

(New Oxford American Dictionary)
“Systems Programming”

- Programming the *operating system*
- What does *that* mean?
OS vs. OS kernel

- OS kernel \approx \text{smallest subset of OS code needed to bootstrap system and provide basic services to user programs}

- “smallest” is debatable
How to “program” it?

- Require some API
  - Application Programming Interface
    - A collection of (documented) functions
      - e.g., Java Math class
OS API

- a.k.a. “system call” interface
  - OS as a very low-level library

- common purpose: provide services to user level programs
  - def: program in execution = process
The Process

- A program in execution

- Code + Data \{ global, local, dynamic \}
  
  + OS kernel data

- OS hides complexity of machine from processes by creating abstractions
AN x64 processor is screaming along at billions of cycles per second to run the XNU kernel, which is frantically working through all the POSIX-specified abstraction to create the Darwin system underlying OS X, which in turn is straining itself to run Firefox and its Gecko renderer, which creates a Flash object which renders dozens of video frames every second

because I wanted to see a cat jump into a box and fall over.

I am a god.

“Abstraction”

http://xkcd.com/
Primary Abstractions

- Logical control flow
- Exceptional (extra-process) control flow
- Logical address space
- Uniform I/O
- Interprocess Communication
In the old days ...

- ... every program had to include its own implementation of all the above!

- Now, OS simplifies life for all of us.

  - Only need to know how to use them, not how they’re implemented.
But!

- In this class we dig a bit deeper
- What facilities are encapsulated by syscalls?
- What limitations/restrictions do they have?
- Why are they designed the way they are?
- How do they work behind the scenes?
But why should I care?
- **efficiency**: know how to use tools optimally; reuse existing features and design/layer new ones appropriately

- **robustness**: avoid bugs/failures & know how to diagnose and fix them
the real reason: it’s fun to take things apart!
goal: turn you into a hacker
(or: make you a better hacker)
hacker |ˈhækər|
noun
1 A person who enjoys exploring the details of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn only the minimum necessary.
Our tools (& approach)

- C & Linux
  - C: low-level language
  - GNU Linux: open source kernel & tools
    - GNU gdb & gcc; debugger & compiler
    - course server: braeburn.cs.iit.edu