Introduction

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

- lee@iit.edu
- http://moss.cs.iit.edu
- Office: SB 226C
- Hours: Wed & Fri 3:15PM-5:15PM
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, assignment writeups, slides, screencasts, links, etc.
Online resources

2. Blackboard
- only for grade reporting!
Online resources

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

4. Piazza: discussion forum
   - all class-related questions
   - monitored by TAs
   - scales *way* better than e-mail
Textbooks

1. The C Programming Language
   - Brian W. Kernighan
   - Dennis M. Ritchie

2. Computer Systems: A Programmer’s Perspective
   - Bryant
   - O’Hallaron

3. The Linux Programming Interface
   - Michael Kerrisk
Grading

- 40% Labs
- 30% Midterm exam
- 30% Final exam
  - exam scores may be *normalized* to 70%
Labs

- 5-6 fairly substantial machine problems
- 100-1000 LOC
- real-world application of concepts covered in lecture & textbook
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 ≈ 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., get/put/del for a hashtable
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*
“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
Fourier

- All labs must be tested and submitted on the class Linux server: fourier.cs.iit.edu

- You should’ve received an e-mail with account info already — let me know if not!

- Log in via SSH client, submit work via Git

- BitBucket invitations are coming