Introduction

CS 351: Systems Programming
Michael Lee <lee@iit.edu>
Michael Lee

- lee@iit.edu

- http://moss.cs.iit.edu
Agenda

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

1. Course website moss.cs.iit.edu/cs351
   - static information
   - lecture calendar, assignment writeups, slides, links, etc.
Resources

2. Blackboard
   - grade reporting
   - midterm & final exams
Resources

   - Live lecture recordings
   - Supplementary concept-focused videos
Resources

4. Discord: discussion forum
   - all class-related questions
   - all office hours here!
   - supports audio chat and screen-sharing
   - feel free to @ me outside office hours
Textbooks

SECOND EDITION
THE C PROGRAMMING LANGUAGE
BRIAN W. KERNIGHAN, DENNIS M. RITCHIE
PRENTICE HALL SOFTWARE SERIES

COMPUTER SYSTEMS
A Programmer’s Perspective
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
Labs

- ~5 machine problems
- 100-1000 LOC
- real-world application of concepts covered in lecture & textbook
Late Policy

- 7-day late pool, distributed however you like across labs (a day at a time)
- If you’re out of late days, late submissions will *not* be accepted!
Exams

- Midterm and Final exams both administered online, both open-book, open-notes

- Scores may be linearly scaled so that median/mean (whichever lower) is 75%

- Midterm tentatively scheduled for March 12
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';
}
```
Online workflow

- Asynchronous / Synchronous components
- Pre-recorded videos / Zoom lectures
- Labs / In-class demos and QAs
Online workflow

- Watch any pre-recorded videos \textit{before} lecture

- Lectures may start with quiz (not graded) and end with practicum

- Lots of live coding / debugging / problem solving during Zoom sessions
Course Overview
Prerequisites

- “substantial” programming experience
- data structures: concepts & implementation
- basic run-time analysis (big O)
- 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
“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 \textit{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 READS 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
Fourier

- All labs must be tested and submitted on the class Linux server: fourier.cs.iit.edu
- You will receive an e-mail with account info
- Log in via SSH client, submit work by pushing to Github