Introduction

CS/COE 0449
Introduction to Systems Software

wilkie
(with content borrowed from Vinicius Petrucci)

Spring 2019/2020

Syllabus / Administrivia

I'm obligated to inform you that this is, in fact, a university course.

Welcome!

- Hello!
- I’m wilkie.

- I do research and implementation on Distributed Systems.
  - My area of research interest is Software Preservation / Repeatability

- I’ve worked on Social Networks, Embedded Systems, Operating Systems, and Virtualization-based software archival.
  - I have a full-time job! Bear with me!

- Traditional Computer Scientists think I’m “weird” I think…

All the fun on Day One

- This is Introduction to Systems Software CS/COE 0449 !!!

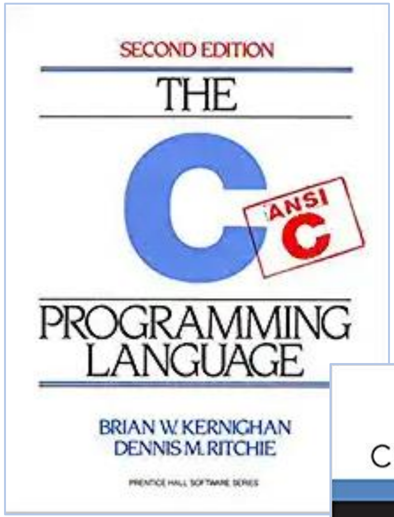
- Course website: https://wilkie.github.io/cs449
  - Schedule
  - Syllabus
  - Announcements
  - **LOOK AT THE WEBSITE**

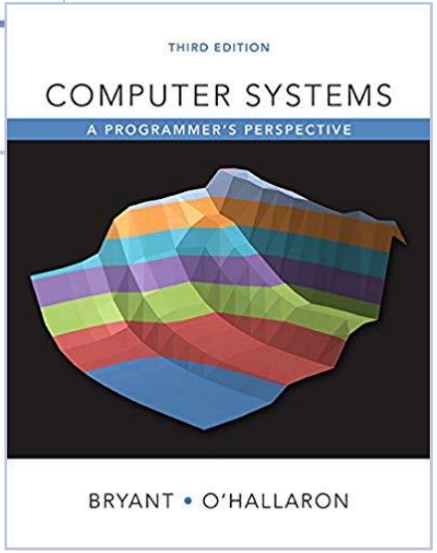
- Office: 5413 Sennott Square (across from the mail room)
- Office Hours: TBA

The Textbooks

- *The ANSI C Programming Language* (2nd Edition)
  - By Brian Kernighan and Dennis Ritchie
  - Published by Prentice Hall, 1988
  - Often called the K&R book.
  - Conventions referred to as K&R style.
  - Old but trusty!

- *Computer Systems: A Programmer's Perspective* (3rd Edition)
  - By Randal E. Bryant and David R. O'Hallaron
  - Published by Pearson, 2016





Course Layout

- Lectures
  - Present high-level concepts.
- Recitations
  - Applied concepts and introduce tools and skills for lab-work.
  - Clarify lectures and review topics.
- Programming Assignments (Labs)
  - **THE BULK OF YOUR COURSEWORK!**
  - Roughly two weeks per assignment.
  - Provide deeper dive into some new skill or systems concept.
  - Programming, measurement, design.
- Exams (Midterm + Final)
  - Tests comprehension of concepts

Policies: Lab Assignments

- **Collaboration**
  - You **MUST WORK ALONE** on all lab assignments.
- **Submission**
  - Electronic submission using Gradescope (no exception)
  - Check due dates on the course website
- **Thoth Machine**
  - Many labs will assume the use of a specialized machine.
  - You must use this machine:
    - `ssh thoth.cs.pitt.edu`
  - Use your Pitt username and password.
  - Talk to your TA if you have any issues. (Do NOT start assignments late!)

Policies: Late Work

- You get **5 Late Days**
  - Covers most normal setbacks and life and schedule mishaps.
  - A **maximum of 2 Late Days** per lab assignment.
  - That is, assignments 3 days late will always take at least 1 penalty day.

- When you run out…
  - Late penalty incurs a **15% penalty for each day**. (out of original 100%)
  - An assignment **cannot be submitted after the 3rd penalty day**.
    - Four days late: that's a 0.

- Emergencies
  - Major emergencies require haste communication with me and your advisor.

- **Start everything early!!**

Policies: Grading

- I don't keep track of attendance
  - But you should come to class!
  - A lot of the concepts are best demonstrated interactively.

- Labs: 50% (Weighted according to effort)
- Midterm: 20%
- Final: 25% (Necessarily Cumulative)
- Homework: 5% (Online problem sets)

- You **CANNOT** pass without doing the lab assignments.

Policies: Conduct / Academic Integrity

- **Disability Resources / Services:**
  - Contact DRS **412-648-7890**; TTY: **412-383-7355**
  - They will email me, and I will listen to what they tell me to do.

- **Cheating:**
  - First time: 0 on the lab/assignment/project/exam
  - Second: **Fail the course**. Reported. (Applies to **all** involved.)
  - Pro-tip: **DON'T CHEAT**. Start early. Ask appropriate staff for help.
  - The syllabus online has a more thorough policy.

- **Conduct:**
  - Jokes/comments about sex, gender, race, ethnicity, religion, etc are not tolerated. Includes any online spaces involved.

More Notes about Cheating

- **Again, do not cheat.**
- **I'm not grading lab assignments, but I still look at your work.**
- **Ask for help (There are PLENTY of resources)**
  - TAs and my own office hours
  - Undergraduate Helpdesk (CRC)
  - We want you to succeed!

- **I can definitely tell when someone cheats.**
  - It is very obvious.
  - **Do not do it**.
  - The University is justifiably strict about it.

- **Do not publish your code until after the semester (if at all)**

Teaching Pedagogy / Philosophy

- **I don't like teaching from slides or from a book.**
  - You can do that yourself. I prefer *interactivity*.
- **I want to demonstrate practical applications.**
  - Including humanist and artistic applications.
- **I want you to walk away with a direction/goal to do something else.**
  - Hopefully, you find something to be inspired by.
- **I trust my students that they could learn on their own.**
  - But don't want them to *have* to do so.
  - Ask questions! Challenge concepts! Ask for help!

Course Overview

If food were knowledge, this would be, like, our restaurant menu.

Topics (Subject to deviation)

- We're going to (tentatively) learn **SO MUCH** fun stuff!

- The C Systems Programming Language
  - Some x86 Assembly
- Memory Models
  - Addresses and Pointers
  - Memory management
- Memory Caches
- Operating Systems
  - Processes / Signals
  - Interprocess Communication
  - The Basics of Virtual Memory
  - Basic Network Programming

Skills

- C Programming
  - Abstractions and coping without them
  - x86 assembly (ISA) / calling conventions (ABI)
  - Interactive debugging
  - Data representation
  - **We gain an appreciation of abstraction (and respecting limitations)**

- Systems Design
  - Learning the “why” for many systems abstractions
  - Manipulating systems and existing programs
  - Thinking about how systems might change in the future
  - **We demystify software so as to no longer be a hostage to its design**

What is Systems?

- Systems is broad
  - A subfield of CS dealing with the interactions between software/hardware.
  - A layer that provides abstractions and must constantly reevaluate them.
    - Operating Systems
    - File Systems
    - Program Analysis / Debugging Tools
    - Intra/Inter System Protocols
  - A house built from trade-offs in approach…
    - Do you build better hardware? Add more memory?
    - Or, do you design better software?
  - And trade-offs in design…
    - Do you choose the specialized path?
    - Or, do you create a general system?
    - Both??
  - Very opinionated!!!!!!!!!!

What is Systems??

Looking for guidance by looking at recent research:

- Research Conferences
  - SOSP/OSDI/EuroSys – OS design, kernel design, virtualization
    - *Parit models: erasure-coded resilience for prediction serving systems*
    - *Teechain: a secure payment network with asynchronous blockchain access*
    - *Finding semantic bugs in file systems with an extensible fuzzing framework*
    - *File systems unfit as distributed storage backends: lessons from 10 years of Ceph evolution*
    - *Snap: a microkernel approach to host networking*
  - HotOS – Positions on Systems' future
    - *Machine Learning Systems are Stuck in a Rut*
    - *Granular Computing*
    - *I/O Is Faster Than the CPU -- Let's Partition Resources and Eliminate (Most) OS Abstractions*
    - *I'm Not Dead Yet!: The Role of the Operating System in a Kernel-Bypass Era*
    - *Unikernels: The Next Stage of Linux's Dominance*
    - *The Case for I/O-Device-as-a-Service*

Why the C Programming Language?

- Because B sucks and D wasn't invented yet. J/K.

- C was invented in 1972 alongside UNIX to an effort to aid application development of that system.
- Eventually UNIX itself was rewritten in C cementing C as a systems language.
- As such, C provides a high-level abstraction of assembly / machine-code and a low-level abstraction of memory, from the perspective of the C programmer.
  - This is important for programming systems code!
  - Allows full manipulation of memory (to one's peril, often.)
  - This, in turn, allows for full manipulation of cpu/hardware.

Why the C Programming Language??

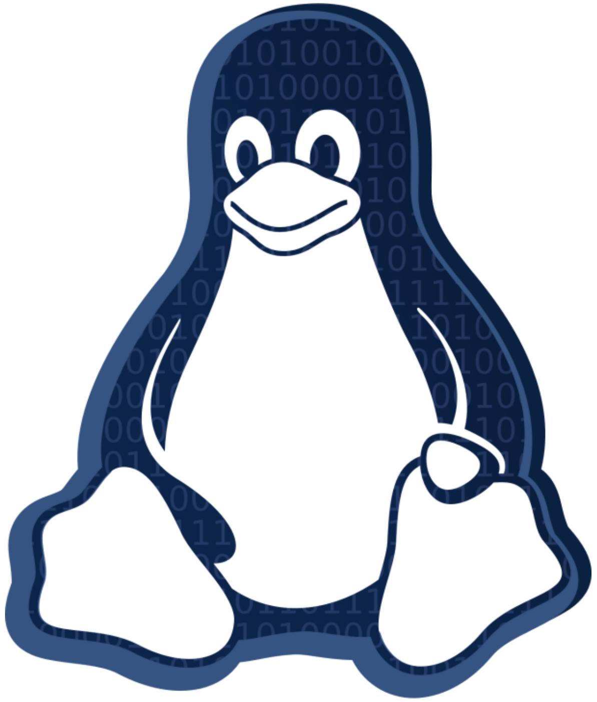
- **Learning C helps you understand Systems.**
- **Understanding Systems lets you make them better.**
  - Or break them. ☺
- **C reveals the underlying memory model and execution environment.**
  - Lets you understand ***any*** program.
  - Even if you do not have the original code.
- **Failing at C helps you learn…**
  - Because then you debug your program.
  - And debuggers are very useful tools.

How people use these skills

- Writing Operating Systems
  - not the entire thing hopefully
  - … but parts are generally gonna be C/C-like

  - Understanding systems means knowing how to mitigate/improve performance.
    - Important that your abstractions don't hurt performance because EVERY user application suffers.
    - Yet, performance is not the only consideration; understanding abstractions should help alleviate design fatigue. https://wilkie.how/posts/kaashoeks-law

  - Linux and Device Drivers: 10+ million lines of C
    - Yikes.
    - But, learning C means you can potentially read this and learn more about / improve / extend Linux.



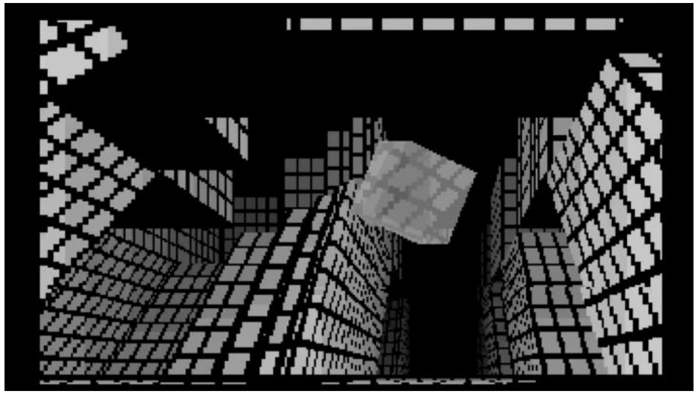
How people use these skills

- **Debugging Higher-level Programs**
  - Yes, even Python itself crashes!
  - … and the Python interpreter is written in C …
  - … and computers don’t understand C …
  - … so it’s gonna give you an assembly dump.

How people use these skills

- Creating Art
  - Real-time art includes not just video games.
  - There is a lot of fun and skill involved.
  - Being creative within a constraint has been very alluring.

  - The Demoscene is such a community.



How people use these skills

- Breaking Things for Great Good
  - Or great bad… I'm not your parents.
  - Why? Old programs with copy-protection are still useful.
    - Original source code backed up?? What time do you think this is?? Never????
  - And it is technically legal to reverse-engineer and/or change them.
    - The best kind of legal.
    - But I'm not a lawyer and this is not legal advice. lol
  - You will typically use a "debugger" to break down a program's behavior.
  - And then patch it to do / not-do things.
  - Generally done professionally by librarians/archivists.
  - We will also do this!!

How YOU will use these skills

- All of the above!!
- And, of course, **TO HAVE FUN!!**