Computer Systems
CSE 410 Winter 2017

Instructor: Justin Hsia
Teaching Assistants: Kathryn Chan, Kevin Bi, Ryan Wong, Waylon Huang, Xinyu Sui
A Comic

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.

http://xkcd.com/676/
Welcome to CSE410!

See the key abstractions “under the hood” to describe “what really happens” when a program runs:

- How is it that “everything is 1s and 0s”?
- Where does all the data get stored and how do you find it?
- How can more than one program run at once?
- What happens to a Java or C program before the hardware can execute it?
- And much, much, much more...

An *introduction* that will:

- Profoundly change/augment your view of computers and programs
- Connect your source code down to the hardware
- Leave you impressed that computers ever work
Who: Course Staff

- Your Instructor: just call me Justin
  - From California (UC Berkeley and the Bay Area)
  - I like: teaching, the outdoors, board games, and ultimate
  - Excited to be teaching a non-majors course for the 1st time!

- 5 TAs:
  - Available in office hours, via email, on Piazza
  - An invaluable source of information and help

- Get to know us
  - We are here to help you succeed
  - And to make the course better – with your help
Who are You?

- 85 students registered
  - See me if you are interested in taking the class but are not yet registered
- Undergrads, grads, many different majors
  - Most of you will find almost everything in the course new
- Submit Start-of-Quarter Survey so we can find out more
- Get to know each other and help each other out!
  - Learning is much more fun with friends
  - Working well with others is a valuable life skill
  - Diversity of perspectives expands your horizons
What is this Class About?

- What do we mean by hardware? software?
- Why do we need a hardware/software interface?
- Why do we need to understand both sides of this interface?
C/Java, assembly, and machine code

if (x != 0) y = (y+z)/x;

Compiler

cmpl $0, -4(%ebp)
je .L2
movl -12(%ebp), %eax
movl -8(%ebp), %edx
leal (%edx,%eax), %eax
movl %eax, %edx
sarl $31, %edx
idivl -4(%ebp)
movl %eax, -8(%ebp)

.L2:

Assembly Language

Assembler

100000110111100001001000001110000000000011000000000011000100010110100010000100100000101001000101101000110001001010001010010001101000001000000001010001001110000101100000111111010000111111111011101111100001001000001110010001001010001000010010000011000

Machine Code
C/Java, assembly, and machine code

- All program fragments are equivalent
- You’d rather write C! (more human-friendly)
- Hardware executes strings of bits
  - In reality everything is voltages
  - The machine instructions are actually much shorter than the number of bits we would need to represent the characters in the assembly language
HW/SW Interface: Historical Perspective

- Hardware started out quite primitive
HW/SW Interface: Historical Perspective

- Hardware started out quite primitive
  - Programmed with very basic instructions (*primitives*)
  - e.g. a single instruction for adding two integers
- Software was also very basic
  - Closely reflected the actual hardware it was running on
  - Specify each step manually
HW/SW Interface: Assemblers

- Life was made a lot better by assemblers
  - 1 assembly instruction = 1 machine instruction
  - More human-readable syntax
    - Assembly instructions are character strings, not bit strings
  - Can use symbolic names

Diagram:
- User:
  - Program in assembly language
- Assembler:
  - Assembler specification
- Hardware:
HW/SW Interface: Higher-Level Languages

- Higher level of abstraction
  - 1 line of a high-level language is *compiled* into many (sometimes very many) lines of assembly language

![Diagram showing the process from user program in C to Hardware]
HW/SW Interface: Compiled Programs

Code Time

Compile Time

Run Time

User program in C

C Compiler

Assembler

Hardware

.c file

.exe file

Note: The compiler and assembler are just programs, developed using this same process.
Operating Systems: Traditional Picture

- "The OS is everything you don’t need to write in order to run your application"
  - This depiction invites you to think of the OS as a library
  - In some ways, it is:
    - All operations on I/O devices require OS calls (syscalls)
  - In other ways, it isn't:
    - You use the CPU/memory without OS calls
    - It intervenes without having been explicitly called
Big Theme: Abstractions and Interfaces

- Computing is about abstractions
  - (but we can’t forget reality)
- What are the abstractions that we use?
- What do you need to know about them?
  - When do they break down and you have to peek under the hood?
  - What bugs can they cause and how do you find them?
- How does the hardware relate to the software?
  - Become a better programmer and begin to understand the important concepts that have evolved in building ever more complex computer systems
Course Roadmap

Higher-level language (C):

```c
car *c = malloc(sizeof(car));
c->miles = 100;
c->gals = 17;
float mpg = get_mpg(c);
free(c);
```

Assembly language:

```assembly
get_mpg:
    pushq %rbp
    movq %rsp, %rbp
    ...
    popq %rbp
    ret
```

Machine code:

```
0111010000011000
1000110100000100000000101000100111000010110000011111101000011111
```

Computer system:

Memory & data
Integers & floats
Machine code & C
x86 assembly
Procedures & stacks
Arrays & structs
Memory & caches
Processes
Virtual memory
Operating Systems

Memory & data
Integers & floats
Machine code & C
x86 assembly
Procedures & stacks
Arrays & structs
Memory & caches
Processes
Virtual memory
Operating Systems
Little Theme 1: Representation

- All digital systems represent everything as 0s and 1s
  - The 0 and 1 are really two different voltage ranges in the wires
  - Or magnetic positions on a disc, or hole depths on a DVD, or even DNA...

- “Everything” includes:
  - Numbers – integers and floating point
  - Characters – the building blocks of strings
  - Instructions – the directives to the CPU that make up a program
  - Pointers – addresses of data objects stored away in memory

- Encodings are stored throughout a computer system
  - In registers, caches, memories, disks, etc.

- They all need addresses (a way to locate)
  - Find a new place to put a new item
  - Reclaim the place in memory when data no longer needed
Little Theme 2: Translation

- There is a big gap between how we think about programs and data and the 0s and 1s of computers
  - Need languages to describe what we mean
  - These languages need to be translated one level at a time

- We know Java as a programming language
  - Have to work our way down to the 0s and 1s of computers
  - Try not to lose anything in translation!
  - We’ll encounter C language, assembly language, and machine code (for the x86 family of CPU architectures)
Little Theme 3: Control Flow

- How do computers orchestrate everything they are doing?

- Within one program:
  - How do we implement if/else, loops, switches?
  - What do we have to keep track of when we call a procedure, and then another, and then another, and so on?
  - How do we know what to do upon “return”?

- Across programs and operating systems:
  - Multiple user programs
  - Operating system has to orchestrate them all
    - Each gets a share of computing cycles
    - They may need to share system resources (memory, I/O, disks)
  - Yielding and taking control of the processor
Writing Assembly Code? In 2017???

- Chances are, you’ll never write a program in assembly
  - Compilers are much better and more patient than you are
- But understanding assembly is the key to the machine-level execution model
  - Behavior of programs in presence of bugs
    - High-level language model breaks down
  - Tuning program performance
    - Understand optimizations done/not done by the compiler
    - Understanding sources of program inefficiency
  - Implementing system software
    - Operating systems must manage process state
  - Fighting malicious software
  - Using special units (timers, I/O co-processors, etc.) inside processor!
Course Outcomes

- Understanding of some of the abstractions that exist between programs and the hardware they run on, why they exist, and how they build upon each other

- Knowledge of some of the details of underlying implementations
  - Less important later, but cannot “get it” without “doing it” and “doing it” requires details

- Become more effective programmers
  - Understand some of the many factors that influence program performance
  - More efficient at finding and eliminating bugs
  - Facility with more languages that we use to describe programs and data
  - Better understand *new hardware*
Course Perspective

- CSE410 will make you a better programmer
  - Purpose is to show how software really works
  - Understanding the underlying system makes you more effective
    - Better debugging
    - Better basis for evaluating performance
    - How multiple activities work in concert (e.g. OS and user programs)
  - Not just a course for hardware enthusiasts!
    - What every programmer needs to know (plus many more details)
    - “Stuff everybody learns and uses and forgets not knowing”

- CSE410 presents a world-view that will empower you
  - The intellectual and software tools to understand the trillions+ of 1s and 0s that are “flying around” when your program runs
Some fun topics that we will touch on

- Which of the following seems the most interesting to you? (vote at http://PollEv.com/justinh)
  
a) What is a GFLOP and why is it used in computer benchmarks?
b) How and why does running many programs for a long time eat into your memory (RAM)?
c) What is stack overflow and how does it happen?
d) Why does your computer slow down when you run out of disk space?
e) What was the flaw behind the original Internet worm and the Heartbleed bug?
f) What is the meaning behind the different CPU specifications? (e.g. # of cores, # and size of cache, supported memory types)
Lecture Outline

- Course Introduction
- **Course Policies**
- Command-Line Demo
Communication

- **Website:** [http://cs.uw.edu/410](http://cs.uw.edu/410)
  - Schedule, policies, sections, links, assignments, etc.

- **Discussion:** [http://piazza.com/washington/winter2017/cse410](http://piazza.com/washington/winter2017/cse410)
  - Announcements made here
  - Ask and answer questions – staff will monitor and contribute

- **Office Hours:** spread throughout the week
  - Can also e-mail to make individual appointments

- **Anonymous feedback:**
  - Comments about anything related to the course where you would feel better not attaching your name
  - Can send to individual staff member of whole staff
Textbooks

- **Computer Systems: A Programmer’s Perspective**
  - Randal E. Bryant and David R. O’Hallaron
  - Website: [http://csapp.cs.cmu.edu](http://csapp.cs.cmu.edu)
  - Must be 3rd edition
    - [http://csapp.cs.cmu.edu/3e/changes3e.html](http://csapp.cs.cmu.edu/3e/changes3e.html)
    - [http://csapp.cs.cmu.edu/3e/errata.html](http://csapp.cs.cmu.edu/3e/errata.html)
  - This book really matters for the course!
    - How to solve labs
    - Practice problems and homework

- **A good C book – any will do**
  - *The C Programming Language* (Kernighan and Ritchie)
  - *C: A Reference Manual* (Harbison and Steele)
Videos / Online course

- Gaetano Borriello and Luis Ceze made videos in 2013 covering the course content for an online version
  - And self-check quiz questions
- A great resource – I encourage you to watch them
  - Generally optional unless class is cancelled or something
  - Occasionally may “require before class” so you don’t get lost in an example
- **Warning:** some content has since changed
  - Now “all 64-bit” so some videos may have extra information no longer relevant
  - When in doubt, go with current lectures (but do ask first)
Course Components

- Lectures (26)
  - Introduce the concepts; supplemented by textbook
- Written homework assignments (5)
  - Mostly problems from textbook to solidify understanding
- Programming lab assignments (4.5)
  - Provide in-depth understanding (via practice) of an aspect of system
- Exams (2)
  - **Midterm**: Friday, February 10, in lecture
  - **Final**: Tuesday, March 14, 2:30-4:20pm
- *Optional* Sections (8-9)
  - Applied concepts, important tools and skills for labs, clarification of lectures, exam review and preparation
Grading

- **Exams:** Midterm (15%) and Final (30%)
  - Some old exams on course website
- **Homework:** equally weighted (20% total)
- **Labs:** weighted according to effort (30% total)
- **EPA:** Effort, Participation, and Altruism (5%)
Late Days

- Lab submissions due at 11:59pm
  - Count lateness in days (even if just by a second)
- Late penalty is 20% deduction of score per day
  - No submissions accepted more than two days late
  - Special: weekends count as one day
- You are given 4 late days for the whole quarter
  - No benefit to having leftover tokens
- Use at own risk – don’t want to fall too far behind
  - Intended to allow for unexpected circumstances
    - Illness, family emergency, etc.
Midterm Clobberer Policy

- Final is cumulative
- Can replace Midterm score with midterm portion score of Final
  - Make up for a bad testing day, illness, other unforeseen circumstance
  - In practice, final is harder than the midterm
- See course policies for exact formula
  - https://courses.cs.washington.edu/courses/cse410/17wi/policies.html#exams
EPA

- Encourage class-wide learning!

- Effort
  - Attending office hours, completing all assignments
  - Keeping up with Piazza activity

- Participation
  - Making the class more interactive by asking questions in lecture, office hours, and on Piazza
  - Peer instruction voting

- Altruism
  - Helping others in office hours and on Piazza
Peer Instruction

- Increase real-time learning in lecture, test your understanding, increase student interactions
  - Lots of research supports its effectiveness

- Multiple choice question at end of lecture “segment”
  - 1 minute to decide on your own
  - 2 minutes in pairs to reach consensus
  - Learn through discussion

- Vote using Poll Everywhere
  - Use website (https://www.polleverywhere.com) or app
  - Linked to your UWNetID
Working with Others

- We are encouraging study groups and homework parties
- Beneficial for the class
  - Someone to bounce ideas off of and work with and keep you on task
  - Don’t have to go it alone – meet new people!
- Valuable life skill: working well with others
  - Distribution of work, communication skills, time management, teaching skills
Collaboration and Academic Integrity

- All submissions are expected to be yours and yours alone
- You are encouraged to discuss your assignments with other students (ideas), but we expect that what you turn in is yours
- It is NOT acceptable to copy solutions from other students or to copy (or start your) solutions from the Web (including Github)
- Our goal is that *YOU* learn the material so you will be prepared for exams, interviews, and the future
To-Do List

- Explore website thoroughly:  [http://cs.uw.edu/410](http://cs.uw.edu/410)
- Check that you are enrolled in Piazza
- Start-of-Course survey [Canvas] due Friday (1/6)
- **Get your machine set up for this class (VM or klaatu) as soon as possible**
- HW1 released today, due Wednesday (1/11)
  - Can do “Course Policies” now, work on the rest after Friday’s lecture
  - Feel free to work together, but it is your responsibility to understand the solutions
Lecture Outline

- Course Introduction
- Course Policies
- Command-Line Demo
Connecting and Transferring Files

- This information is summarized at https://courses.cs.washington.edu/courses/cse410/17wi/linux-instructions.html

- Secure Shell (SSH)
  - Connect to klaatu.cs.washington.edu
  - Login is your UWNetID; password will be emailed to you

- CSE Linux Virtual Machine
  - Instructions: https://www.cs.washington.edu/lab/vms

- Secure Copy (scp) to transfer files
File Manipulation

- **touch** to create a file
- Choose a text editor to use/learn
  - Usually **vim** or **emacs** recommended on Linux
  - Note that some require X graphics when connected remotely
- List (**ls**) to see directory contents
  - **ls -l** provides more details
- Move (**mv**) to change file names
- Copy (**cp**) to duplicate files
Compiling and Running a C File

- “Compile” using `gcc <source file name>`
  - Default executable name is `a.out`
  - Can specify output file name using `-o` flag
    - Example: `gcc -o out <source file name>`
  - Must deal with compiler `errors` before executable is produced
  - Compiler `warnings` are good to pay attention to

- Run executable using `./<executable name>`
  - Example: `./a.out`