CS 261
Fall 2017

Mike Lam, Professor

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.

Computer Systems I: Introduction

Group warm-up activity

(2-3 people per group; introduce yourself to one or two others first!)

1) Choose one of the following C programs and predict its output.

```
#include <stdio.h>
int main() {
    int x = 40000;
    int y = 50000;
    if ((x * x) < (y * y)) {
        printf("Less than\n");
    } else {
        printf("Not less than\n");
    }
    return 0;
}
```

```
#include <stdio.h>
int main() {
    double a = 1e20;
    double b = -a;
    double c = 3.14;
    if (((a+b) + c) == (a + (b+c))) {
        printf("Equal!\n");
    } else {
        printf("Not equal!\n");
    }
    return 0;
}
```

2) Log into a lab laptop with your e-ID/password (or as "`student`" with no password).

3) Create a file called `main.c` and enter the code for the program you chose.

4) Compile and *run* your program from a terminal with the following commands:

```
gcc -o program main.c
./program
```

(alternatively, just copy/paste into `ideone.com`)

5) What is the output?

More intrigue

- Which of the following versions of a “matrix copy” routine will run the fastest?
  - Or will they always take the same amount of time?

```
for (int i = 0; i < 2048; i++) {
    for (int j = 0; j < 2048; j++) {
        dst[i][j] = src[i][j];
    }
}
```

```
for (int j = 0; j < 2048; j++) {
    for (int i = 0; i < 2048; i++) {
        dst[i][j] = src[i][j];
    }
}
```

What's happening?

- Something about our **mental model** of these programs does not match the **system** on which we're running them.

Systems

- What is a “system?”
  - Can you give some examples?

Systems

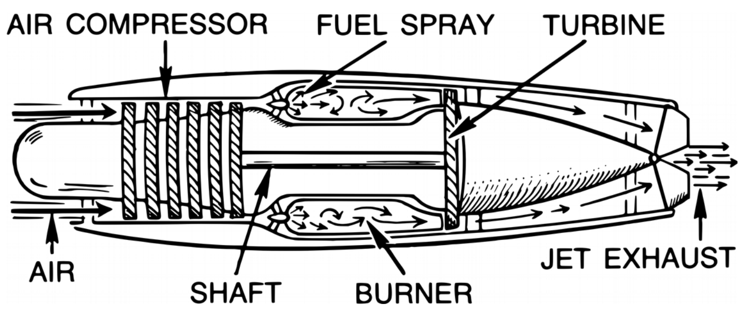
- What is a “system?”
  - Set of interacting components
  - More than the sum of its parts

Jet engine

Computer

Systems

- What is a "system?"
  - Set of interacting components
  - More than the sum of its parts

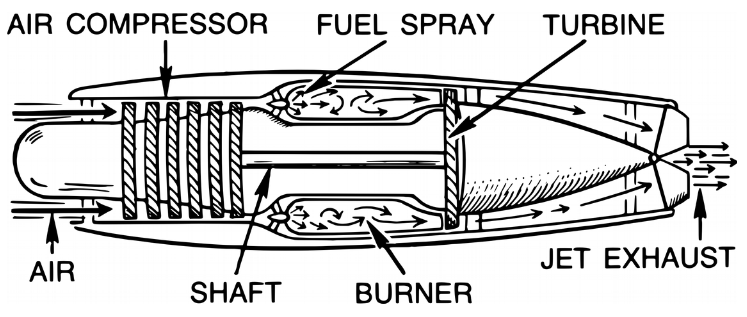


Jet engine

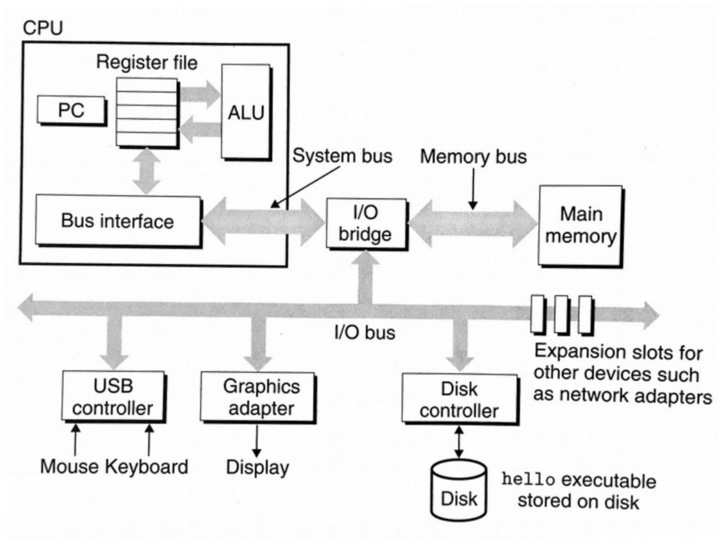
Computer

Systems

- What is a “system?”
  - Set of interacting components
  - More than the sum of its parts



Jet engine



Computer

Systems

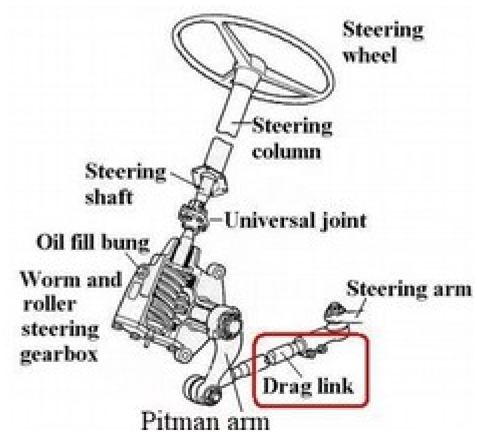
- A **computer system** consists of multiple hardware and software components that work together to run user applications.
- Why do we care about computer systems?
  - We use complex computer systems every day
  - Also: it's your major!
- Our goal: peel back some of the complexity
  - See what’s “under the hood”
  - It’s also worth thinking about <u>why</u> the complexity was hidden; i.e., what is the purpose of *abstraction*?

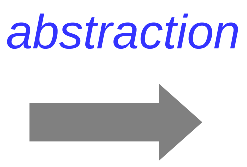
Systems

- What is a *process*? What is a *file*?

Systems

- What is a *process*? What is a *file*?
  - These are examples of abstraction; "fake" views of reality that reduce complexity for users
  - Key ideas: **ignore details** and **focus on interfaces**
  - Especially important in large, complicated systems
  - Understanding abstractions can improve your ability to use them effectively





Caveat

- Software system vs systems software
  - Former: interconnected software components
  - Latter: software providing services to other software
  - We are concerned with both!
    - Examples: multiprocessing, networking, operating systems, compilers, distributed systems

Course Objectives

- Explain machine-level representation of data and code
- Summarize the architecture of a computer
- Explain how complex systems are built from simple components
- Translate high-level code into assembly and machine language
- Write code to emulate the functionality of a computer

- Cultivate a sense of control over computer systems
- Gain an appreciation for software development tools
- Develop a sense of play when writing code
- Appreciate the complexity of systems-level software

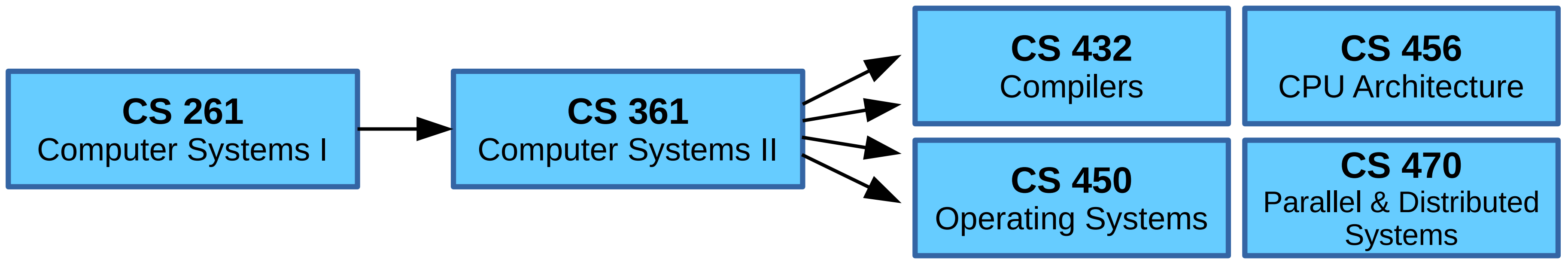
Course Objectives

- Explain machine-level representation of data and code
- Summarize the architecture of a computer
- Explain how complex systems are built from simple components
- Translate high-level code into assembly and machine language
- Write code to emulate the functionality of a computer

- Cultivate a sense of control over computer systems
- Gain an appreciation for software development tools
- Develop a sense of play when writing code
- Appreciate the complexity of systems-level software

Systems courses

- CS 261 units:
  - C and Linux (3 weeks)
  - Binary Representations (2-3 weeks)
  - Assembly and Machine Code (2 weeks)
  - Computer Architecture (3 weeks)
  - Operating Systems Concepts (3 weeks)



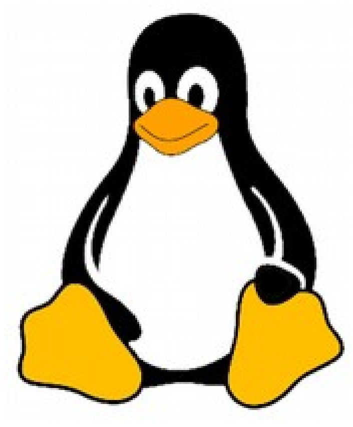
Fundamentals of digital, single-process systems

Multi-process systems and networking

In-depth study of a particular kind of complex system

CS 261

- What this course is NOT:
  - Programming 101 – I will assume you can program
    - However, we will spend a few weeks learning C
  - Electronics 101 – we won’t be going THAT deep
    - If you’re interested, check out PHYS 240/250
  - Linux 101 – but you have the Unix Users Group
    - InstallFest on Wed, Sep 6 at 6:30 in ISAT/CS 246
    - Weekly meetings thereafter (same time and place)



CS 261

- This is not an “**easy**” course
  - *But you **can** handle it!*
  - Be prepared to **read** and **work** a lot
  - Don't be afraid to experiment
  - Learn the **why** and not just the **what**
  - Some stuff is worth memorizing
    - (e.g., powers of two and hex characters)
  - For other stuff, **Google** is your friend
  - **Piazza** is also your friend (literally)
  - Start assignments **early** and <u>ask questions</u>

“More software projects have gone awry for lack of time than for all other causes combined. Why is this so common?

First, our estimating techniques reflect an untrue assumption; i.e., that all will go well.

Second, our estimating techniques confuse effort with progress, hiding the assumption that people and time are interchangeable.”

– Fred Brooks, “*The Mythical Man-Month*” (edited)

Course Components

- **Public website** (`w3.cs.jmu.edu/lam2mo/cs261`)
  - Syllabus, **calendar**, assignments, and resources (links)
- **Canvas course**
  - Concept quizzes and unit tests
  - Grades and private files (i.e., solutions)
  - Piazza Q&A and discussion forum
- **Student server** (`stu.cs.jmu.edu`)
  - Project development and submission

- Make sure you can access all of these!

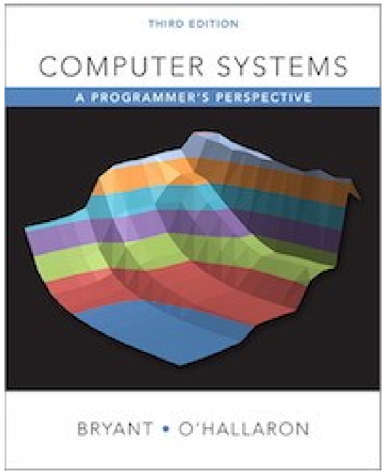
Course Grades

Quizzes and Labs	20%
Programming Projects	40%
Online Unit Tests	20%
Written Exams	20%

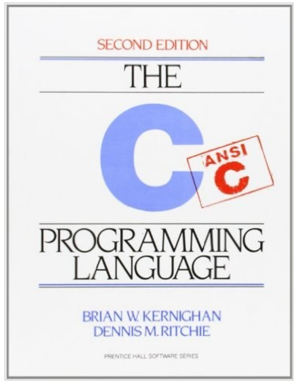
- Quizzes and labs are **formative**
  - Designed to help you learn
- Projects and tests/exams are **summative**
  - Designed to assess what you have learned

Textbook(s)

- Required textbook: “Computer Systems”
  - “CS:APP” textbook from Carnegie-Mellon
  - A practical, example-filled introduction to systems
  - Reserve copy at the library



- Recommended book: "The C Programming Language"
  - **Brian Kernighan** and **Dennis Ritchie** (creator of C)
  - This is “the book” about C
  - Available on Safari Books through the library



Class Policies

- Check Canvas daily for quizzes
- Class attendance is necessary
  - We will be “learning by doing” much of the time
  - Find a group (2-3 people) to work with consistently, or switch it up
- Slides will be posted on the website
  - No need to copy them to your notes
- Please silence your cell phones during class
  - Be respectful with laptop and tablet usage

Course Policies

- Submit programming projects using submit script on `stu`
  - No thumb drives, CDs, or emails
- Project grading will be based on automated test results
  - You will **not** have access to all test code
  - Deductions will be applied for unacceptable style and/or documentation based on a manual inspection of the code
  - See the code guidelines on the website for policies
- Project late policy: submissions up to 72 hours late will receive a letter grade penalty per 24 hr period
  - Projects are due on Friday so that you can take advantage of the weekend for this if necessary

Course Policies

- The JMU Honor Code applies on ALL assignments
  - Violations may be sent to the honor council
  - See relevant section in the syllabus
- All submitted project code must be YOUR work entirely
  - You may work in groups to discuss general approaches (in fact, I encourage this; use *pseudocode* if necessary)
  - However, the primary goal of the projects in this course is to develop individual competency, so **you may NOT share code**
  - This includes letting someone examine or take a photo of your code, or "talking it through" with them line-by-line
  - If you have questions about this, please ask!

Course Policies

- There are a total of three sections of CS 261
  - Two Lam sections and one Weikle section (all T-Th)
  - Projects, unit tests, and exams are common
  - **Quizzes and labs may differ**
  - You are welcome to study with students from other sections, but you must attend and submit assignments to the section you are registered for

Intro lab

- Material from Chapter 1
- Front page: **Computer Organization**
- Back page: **C Compilation**
- Work in groups of 2-3 (no computer required)
- Submit at end of class

Have a great semester!

- Before Thursday:
  - **Take the intro survey on Canvas**
  - **Read sections 1.1-1.4 and 1.8 in CS:APP and take quiz**
  - **Make sure you can log into `stu`**
  - Make sure you can access Piazza
  - Review these slides
  - Read project guide on website