CSE 333
Lecture 1 - Systems programming

Steve Gribble
Department of Computer Science & Engineering
University of Washington
Welcome!

Today’s goals:

- big picture introduction
- discuss course syllabus
- set expectations
Welcome!

Today’s goals:

- big picture introduction
- discuss course syllabus
- set expectations
Course map: 100,000 foot view

- OS / app interface (system calls)
- HW/SW interface (x86 + devices)

**Hardware**
- CPU
- memory
- storage
- network
- GPU
- clock
- audio
- radio
- peripherals

**Operating system**
- C application
- C standard library (glibc)

**C++ application**
- C++ STL / boost / standard library

**Java application**
- JRE

**C++ application**
- C++ application

**Java application**
- Java application

**C standard library**
- C standard library

**C++ application**
- C++ application
Software “System”

A platform, application, or other structure that:

- is composed of multiple modules
  - the system’s **architecture** defines the *interfaces of* and *relationships between* the modules

- usually is complex
  - in terms of its implementation, performance, management

- hopefully has requirements
  - performance, security, fault tolerance, data consistency
A layered view

- your system

- layer below

- layer below

- • • •

- understands and relies on layers below

- provides service to layers above

- client

- client

- client
A layered view

- more useful, portable, reliable abstractions
- constrained by performance, footprint, behavior of the layers below

Your system

- layer below
- layer below
- ...
Example system

Operating system

- a software layer that abstracts away the messy details of hardware into a useful, portable, powerful interface

- modules:
  - file system, virtual memory system, network stack, protection system, scheduling subsystem, ...
  - each of these is a major system of its own!

- design and implementation has tons of engineering tradeoffs
  - e.g., speed vs. (portability, maintainability, simplicity)
Another example system

Web server framework

- a software layer that abstracts away the messy details of OSs, HTTP protocols, and storage systems to simplify building powerful, scalable Web services

- modules:
  - HTTP server, HTML template system, database storage, user authentication system, ...

- also has many, many tradeoffs
  - programmer convenience vs. performance
  - simplicity vs. extensibility
Systems programming

The programming skills, engineering discipline, and knowledge you need to build a system

- **programming**: C / C++

- **discipline**: testing, debugging, performance analysis

- **knowledge**: long list of interesting topics
  - concurrency, OS interfaces and semantics, techniques for consistent data management, algorithms, distributed systems, ...
  - most important: deep understanding of the “layer below”

- quiz: what data is guaranteed to be durable and consistent after a power loss?
Programming languages

Assembly language / machine code

- (approximately) directly executed by hardware
- tied to a specific machine architecture, not portable
- no notion of structure, few programmer conveniences
- possible to write really, really fast code
Structured but low-level languages (C, C++)

- hides some architectural details, is kind of portable, has a few useful abstractions, like types, arrays, procedures, objects
- permits (forces?) programmer to handle low-level details like memory management, locks, threads
- low-level enough to be **fast** and to give the programmer **control** over resources
  - double-edged sword: low-level enough to be complex, error-prone
  - shield: engineering discipline
Programming languages

High-level languages (Python, Ruby, JavaScript, ...)

- focus on productivity and usability over performance
- powerful abstractions shield you from low-level gritty details
  (bounded arrays, garbage collection, rich libraries, ...)
- usually interpreted, translated, or compiled via an intermediate representation
- slower (by 1.2x-10x), less control
Discipline

Cultivate good habits, encourage clean code

- coding style conventions
- unit testing, code coverage testing, regression testing
- documentation (code comments, design docs)
- code reviews

Will take you a lifetime to learn

- but oh-so-important, especially for systems code
  - avoid write-once, read-never code
Knowledge

Tools

- gcc, gdb, g++, objdump, nm, gcov/lcov, valgrind, IDEs, race detectors, model checkers, ...

Lower-level systems

- UNIX system call API, relational databases, map/reduce, Django, ...

Systems foundations

- transactions, two-phase commit, consensus, RPC, virtualization, cache coherence, applied crypto, ...
Welcome!

Today’s goals:

- big picture introduction
- discuss course syllabus
- set expectations
C / C++ programming

Major focus of this course

- ~2 weeks of diving deeper into C
  ‣ review some material from 351 and go deeper
- ~4 weeks of a (sane subset) of C++
- exposure to programming tools
  ‣ unit testing frameworks, performance profiling and analysis, revision control systems
Interacting with UNIX and standard libraries

The “layers below” we will be relying on

- learn C’s standard library and some of C++’s STL
  ▶ including memory management (malloc/new, free/delete)
- learn major aspects of the UNIX system call API
  ▶ I/O: storage, networking
  ▶ process management, signals
Some additional topics

Concurrency
- asynchronous I/O and event-driven programming
- probably won’t cover parallelism, threads

Security
- will be mindful of security topics as they come up
- e.g., how to avoid buffer overflow issues in C/C++
Welcome!

Today’s goals:
- big picture introduction
- discuss course syllabus
- set expectations
What you will be doing

Attending lectures and sections
- lecture: ~30 of them, MWF in this room
- sections: ~10 of them, Thu (8:30 or 9:30) in MGH

Doing programming projects
- ~4 of them, successively building on each other
- includes C, C++; files, networking; writing a server

Exams
- midterm is tentatively on May 2nd  [may change]
- final is non-negotiably on Wed. June 8th, 2:30-4:20pm
Requirements

CSE351 is a prerequisite
- I assume you have just a little exposure to C

CSE332 is a corequisite
- I assume you know what a linked list, tree, hash table is

You need access to a CSE linux environment
- undergraduate labs, ssh into attu.cs, use CSE home VM
Textbooks

Required:

- Computer Systems, A Programmer’s Perspective ("CSAAP")
  ▸ [2nd Ed]. CSE351 textbook; do you already have it?

Recommended (strongly):

- C: A Reference Manual ("CARM") [5th Ed]
- C++ Primer ("C++P") [4th Ed]

Optional (but cool):

- Effective C++ [3rd Ed]
Caveat emptor

This is the first time this course is being offered

- most of it doesn’t exist yet. :)  
- be flexible, provide tons of feedback about topics and pace
  - we need to know if we’re moving too slowly or too quickly
  - we need to know if you’re working too little or too much
  - we need to know if the projects work or are completely busted
Collaboration

Some of the projects will be individual, some in teams
- assume individual unless explicitly stated otherwise

Cross-team collaboration is useful and expected
- help other teams with programming fundamentals, concepts

Plagiarism and cheating is verboten
- helping other teams with assignments, debugging their code
- relying on help without attributing in your writeups
Administrivia

As usual, everything is on the course web

See you on Wednesday!