CIS 330: UNIX and C/C++

Introduction

Prof. Kevin Butler
Winter 2012
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)

operating system

hardware

C application
C standard library (glibc)

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

Java application
JRE

CPU  memory  storage  network
GPU  clock  audio  radio  peripherals
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

provides service to layers above

understands and relies on layers below

your system

layer below

layer below
A layered view

more useful, portable, reliable abstractions

constrained by performance, footprint, behavior of the layers below

client

client

client

your system

layer below

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
Programming languages

• 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
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: ~20 of them, TTh in this room
  ‣ sections: ~10 of them: Wed or Thurs, 2-3 PM, 100 Deschutes

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

• Exams
  ‣ midterm is tentatively on Feb 23
  ‣ final is March 22, 8:00-10:00 AM
Requirements

• CIS 313
  ‣ I assume you know what a linked list, tree, hash table is

• CIS 323
  ‣ You should have had some exposure to C++

• You need access to a UNIX environment
  ‣ ix.cs.uoregon.edu gives you a Solaris environment
  ‣ We may use Linux: LiveCD on room 100 machine, VirtualBox, VM on home machine, external hard disk, etc.
Textbooks

• Recommended:
  ‣ The C Programming Language (“K&R”) [2nd Ed]
  ‣ Programming in C: http://www.cs.cf.ac.uk/Dave/C/CE.html
  ‣ C++ Primer (“C++P”) [4th Ed]
  ‣ C++ for Java Programmers

• Optional (but cool):
  ‣ Effective C++ [3rd Ed]
  ‣ Advanced Programming in the UNIX Environment (“APUE”)
  ‣ The Linux Programming Interface (“TLPI”)

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
  ‣ Note: we will be checking code for similarity with MOSS
• Everything is on the course website

  ‣ [http://www.cs.uoregon.edu/Classes/12W/cis330/](http://www.cs.uoregon.edu/Classes/12W/cis330/)
Attribution

• The slides I’ll be using are a mixture of:
  ‣ my own material, heavily influenced by Steve Gribble (UW)
  ‣ slides from Magda Balazinska, Marty Stepp, John Zahorjan, Hal Perkins, and others (UW)
  ‣ material from other universities’ courses (particularly CMU’s 15-213 and some Harvard courses; thanks to Randy Bryant, Dave O’Hallaron, Matt Welsh, and others)
  ‣ examples and material from colleagues at Penn State, Georgia Tech, NC State, UPenn

• All mistakes are mine
• Created in 1972 by Dennis Ritchie
  ‣ designed for creating system software
  ‣ portable across machine architectures
  ‣ most recently updated in 1999 (C99)

• Characteristics
  ‣ low-level, smaller standard library than Java
  ‣ procedural (not object-oriented)
  ‣ typed but unsafe; incorrect programs can fail spectacularly

This book was typeset (pic|tbl|eqn|troff -ms) using an Autologic
APS-5 phototypesetter and a DEC VAX
8550 running the 9th Edition of the
UNIX operating system.
Dennis Ritchie (dmr)

• Co-creator of UNIX
• Co-creator of C
• Turing Award winner
  ‣ Reflections on Software Research

• "C is quirky, flawed, and an enormous success."
Mindset of C

• “The PDP-11/45 on which our UNIX installation is implemented is a:
  ‣ 16-bit word (8-bit byte) computer with
    • 144K bytes of core memory; UNIX occupies 42K bytes
    • a 1M byte fixed-head disk
    • a moving-head disk with 40M byte disk packs
  ‣ The greater part of UNIX software is written in C.”

  Dennis M. Ritchie and Ken Thompson
  Bell Laboratories
  1974
C workflow

Editor (emacs, vi) or IDE (eclipse)

source files (.c, .h)

foo.h

bar.c

object files (.o)

foo.o

bar.o

compile

link

link

link

link

load

process

execute, debug, profile, ...

bar

bar

statically linked libraries

libZ.a

shared libraries

libc.so

execute,

link

link

link

link
From C to machine code

C source file (dosum.c)

```
int dosum(int i, int j) {
    return i+j;
}
```

C compiler (gcc -S)

assembly source file (dosum.s)

```
dosum:
pushl  %ebp
movl   %esp, %ebp
movl   12(%ebp), %eax
addl   8(%ebp), %eax
addl   12(%ebp), %eax
popl   %ebp
ret
```

machine code (dosum.o)

```
80483b0: 55
89 e5 8b 45
0c 03 45 08
5d c3
```
To Do

• Fill out survey on website