CSE 333
Lecture 22 -- wrapup

Hal Perkins
Department of Computer Science & Engineering
University of Washington
Administrivia

HW4 due tonight 11pm
  - (plus usual late days *if* you have them)

Second exam Friday in class
  - Topic list and old exams on the web
    ‣ Anything all quarter is possible, but biased toward 2nd half
  - Review in section tomorrow — bring your questions!
So what have we been doing for the last 10 9 weeks?
Course goals

Explore the gap between

- Intro: the computer is a magic appliance that runs programs
- CSE 351: the computer is a stupid appliance that executes really, really simple instructions (really, really, really fast)
Course map: 100,000 foot view

- Hardware
  - CPU
  - Memory
  - Storage
  - Network
  - GPU
  - Clock
  - Audio
  - Radio
  - Peripherals

- Operating system
  - OS / app interface (system calls)
    - C application
      - C standard library (glibc)
    - C++ application
      - C++ STL / boost / standard library
  - Java application
    - JRE

- HW/SW interface (x86 + devices)
Goals

Skills

- Programming closer to the hardware: C/C++
- Disciplined design, testing, debugging

Knowledge

- OS interface and semantics, languages, some networking
- A deep(er) understanding of “the layer below”
  
  ‣ quiz: when is the data safely on disk after a write? Actually received over the network? How many copies are made along the way?
Main topics

C Programming, tools, and workflow
Memory management
System interfaces and services (files, etc.)
C++ : the 800-lb gorilla of programming languages
  - “better C” + classes + STL + smart pointers + …
Networking basics: TCP/IP, sockets, …

Drilling deeper…
The C/C++ Ecosystem

System layers: C/C++, libraries, operating system

Building programs

- `cpp`: `#include`, `#ifndef`, and all that
- `compiler (cc1)`: source → `.o`
- `loader (ld)`: `.o + libraries → executable`

Make and related tools to automate the process

- dependency graphs
Program execution

What's a process?
- Address space
- Thread(s) of execution
- Environment (arguments, open files, ...)

Diagram:
- OS kernel [protected]
- Stack
- Shared libraries
- Heap (malloc/free)
- Read/write segment: `.data`, `.bss`
- Read-only segment: `.text`, `.rodata`
- Address space
- Thread(s) of execution
- Environment (arguments, open files, ...)
C language

Structure of C programs

- Header files and implementations; declaration vs definition
- Internal vs external linkage
- Standard types and operators (scalars, including things like uint64_t, structs, arrays, typedef, etc.)
- Functions: defining, using, execution model
- Standard libraries and data structures (strings, streams, ...)
  - C standard library, system calls, and how they are connected
- Handling errors in a language without exception handling
  - return codes, errno, and friends
Memory

Object scope and lifetime (static, automatic, dynamic)

Pointers and associated operators (&, *, ->, [])
- Using pointers for call-by-reference as well as linked data

Dynamic memory allocation (malloc/free; new/delete)
- Who is responsible for dynamic memory & what happens if not done right (dangling pointers, memory leaks, ...)

Tools: debuggers (gdb), monitors (valgrind), ...
- Most important tool: thinking(!)
C++ (and C++11)

A “better C”
- Type-safe streams and memory mgmt (new, delete, delete[ ]), etc.

References and const

C with classes (and objects)
- Constructors, copy constructor, destructor, assignment

Subclasses and inheritance
- Dynamic vs static dispatch & why it matters, virtual functions, vtables
- Pure virtual functions and abstract classes

C++ casts - what are they and why so many (compared to C)?
Templates, STL, and smart ptrs

Templates: parameterized classes and functions
- How the idea is similar to Java generics and what’s different
- How C++ implements templates (expansion)

STL: basics = vector, list & map containers and iterators
- Copy semantics

Smart pointers: unique, shared, and weak
- Reference counting, resource management

Using class hierarchies with STL
- Pointer vs value semantics, assignment slicing
Networking

Layered protocol model, particularly TCP and IP
- What they do, how they are related, how they differ

Network addressing and protocols: IP addresses, DNS, IPv4, IPv6, ports

Application protocols: where HTTP fits in the scheme
# Network Programming

<table>
<thead>
<tr>
<th>Client side</th>
<th>Server side</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. get IP address / port</td>
<td>1. get IP address / port</td>
</tr>
<tr>
<td>2. create socket</td>
<td>2. create socket</td>
</tr>
<tr>
<td><strong>connect</strong> socket to server</td>
<td>3. <strong>bind</strong> socket to address / port</td>
</tr>
<tr>
<td>4. <strong>read</strong> / <strong>write</strong> data</td>
<td>4. indicate that socket is a <strong>listener</strong></td>
</tr>
<tr>
<td>5. <strong>close</strong> socket</td>
<td>5. <strong>accept</strong> connection from client</td>
</tr>
<tr>
<td></td>
<td>6. <strong>read</strong> / <strong>write</strong> data</td>
</tr>
<tr>
<td></td>
<td>7. <strong>close</strong> socket</td>
</tr>
</tbody>
</table>
Concurrency

Why?
- Better resource utilization
- Better throughput

Processes
- Heavyweight, isolated, created by cloning: fork()

Threads
- Lightweight, share address space, pthreads

Synchronization (particularly threads)
- What are the main issues?
Processes vs threads on one slide

OS kernel [protected]
- stack
- shared libraries
- heap (malloc/free)

read/write segment
- .data, .bss

read-only segment
- .text, .rodata

SP →

PC →

parent
fork()

child

OS kernel [protected]
- stack
- shared libraries
- heap (malloc/free)

read/write segment
- .data, .bss

read-only segment
- .text, .rodata

SP →

PC →

SPparent →

SPchild →

PCchild →

PCparent →

OS kernel [protected]
- stack
- shared libraries
- heap (malloc/free)

read/write segment
- .data, .bss

read-only segment
- .text, .rodata

CSE333 lec 22 wrapup // 07-16-17 // Perkins
Phew! That’s it!!

But that’s a lot!!!

Studying for the exam

- Review lecture slides, assignments, exercises
- Try some of the end-of-lecture problems for practice
- Look at old exams and topic list on the web
  ‣ Try the old exam questions first, before looking at answers
- Study groups! Ask questions / trade ideas on the discussion board! Ask course staff questions!
- The goal is learning and mastery
That's it (almost)

But first, ...
This doesn’t happen without great help!
Thanks!!

Course staff:
Phillip Quinn
Soumya Vasisht
Jack Xu
One more thing...

Course evals

- Constructive feedback (positive we hope, but negative when called for) is what helps us get better
  ‣ What did you think of laptop-free lectures?
- Please fill out online before it closes (i.e., today or tomorrow - take a couple of minutes after class - thx)
Congratulations and good luck on the exam!!

You’ve learned a lot – go out and build great things!!!

See you Friday!