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

- **HW/SW interface** (x86 + devices)
- **OS / app interface** (system calls)
  - C application
  - C standard library (glibc)
  - CPU
  - GPU
- **operating system**
  - C++ application
  - C++ STL / boost / standard library
  - memory
  - clock
- **hardware**
  - storage
  - network
  - audio
  - radio
  - peripherals
- **Java application**
  - JRE
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 $\rightarrow$ .o

loader (ld): .o + libraries $\rightarrow$ 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, ...)

<table>
<thead>
<tr>
<th>0xFFFFFFFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS kernel [protected]</td>
</tr>
<tr>
<td>stack</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>shared libraries</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>heap (malloc/free)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>read/write segment</td>
</tr>
<tr>
<td>.data, .bss</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>read-only segment</td>
</tr>
<tr>
<td>.text, .rodata</td>
</tr>
</tbody>
</table>
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 heirarchies 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

Client side
1. get IP address / port
2. create socket
3. **connect** socket to server
4. **read** / **write** data
5. **close** socket

Server side
1. get IP address / port
2. create socket
3. **bind** socket to address / port
4. indicate that socket is a **listener**
5. **accept** connection from client
6. **read** / **write** data
7. **close** socket
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

<table>
<thead>
<tr>
<th>OS kernel [protected]</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack</td>
</tr>
<tr>
<td>shared libraries</td>
</tr>
<tr>
<td>heap (malloc/free)</td>
</tr>
<tr>
<td>read/write segment</td>
</tr>
<tr>
<td>.data, .bss</td>
</tr>
<tr>
<td>read-only segment</td>
</tr>
<tr>
<td>.text, .rodata</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OS kernel [protected]</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack</td>
</tr>
<tr>
<td>shared libraries</td>
</tr>
<tr>
<td>heap (malloc/free)</td>
</tr>
<tr>
<td>read/write segment</td>
</tr>
<tr>
<td>.data, .bss</td>
</tr>
<tr>
<td>read-only segment</td>
</tr>
<tr>
<td>.text, .rodata</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OS kernel [protected]</th>
</tr>
</thead>
<tbody>
<tr>
<td>stackparent</td>
</tr>
<tr>
<td>stackchild</td>
</tr>
<tr>
<td>shared libraries</td>
</tr>
<tr>
<td>heap (malloc/free)</td>
</tr>
<tr>
<td>read/write segment</td>
</tr>
<tr>
<td>.data, .bss</td>
</tr>
<tr>
<td>read-only segment</td>
</tr>
<tr>
<td>.text, .rodata</td>
</tr>
</tbody>
</table>
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!