Here early? Try going to http://chimein.cla.umn.edu/ and see if you can answer an ice cream question.

Course Overview and Introduction

CSci 2021: Machine Architecture and Organization
Lecture #1, September 5th, 2018

Your instructor: Stephen McCamant

Based on slides originally by:
Randy Bryant, Dave O’Hallaron

Overview

- Course themes
- Four realities
- Intermission: ChimeIn
- How the course fits into the CS curriculum
- Logistics

Course Theme:
Abstraction Is Good But Don’t Forget Reality

- Most CS courses emphasize abstraction
  - Abstract data types
  - Asymptotic analysis
- These abstractions have limits
  - Especially in the presence of bugs
  - Need to understand details of underlying implementations
- Useful outcomes
  - Become more effective programmers
  - Able to find and eliminate bugs efficiently
  - Able to understand and tune for program performance
  - Prepare for later “systems” classes in CS & EE
  - Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems

Great Reality #1:
Ints are not Integers, Floats are not Reals

- Example 1: Is $x^2 \geq 0$?
  - Floats: Yes!
  - Ints:
    - $40000 \times 40000 \rightarrow 1600000000$
    - $50000 \times 50000 \rightarrow ??$
- Example 2: Is $(x + y) + z = x + (y + z)$?
  - Unsigned & Signed Ints: Yes!
  - Floats:
    - $(1e20 + -1e20) + 3.14 \rightarrow 3.14$
    - $1e20 + (1e20 + 3.14) \rightarrow ??$

Code Security Example

```c
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy at most maxlen bytes from kernel region to user buffer */
int copy_from_kernel(void *user_dest, int maxlen) {
  /* Byte count len is minimum of buffer size and maxlen */
  int len = KSIZE < maxlen ? KSIZE : maxlen;
  memcpy(user_dest, kbuf, len);
  return len;
}
```

- Similar to code found in FreeBSD’s implementation of getpeername
- There are legions of smart people trying to find vulnerabilities in programs

Typical Usage

```c
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy at most maxlen bytes from kernel region to user buffer */
int copy_from_kernel(void *user_dest, int maxlen) {
  /* Byte count len is minimum of buffer size and maxlen */
  int len = KSIZE < maxlen ? KSIZE : maxlen;
  memcpy(user_dest, kbuf, len);
  return len;
}

#define MSIZE 528

void getstuff() {
  char mybuf[MSIZE];
  copy_from_kernel(mybuf, MSIZE);
  printf("%s\n", mybuf);
}
```

Cartoon source: xkcd.com/571
Malicious Usage

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy at most maxlen bytes from kernel region to user buffer */
int copy_from_kernel(void *user_dest, int maxlen) {
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

Computer Arithmetic

- Does not generate random values
  - Arithmetic operations have important mathematical properties
- Cannot assume all “usual” mathematical properties
  - Due to finiteness of representations
  - Integer operations satisfy “ring” properties
  - Commutativity, associativity, distributivity
  - Floating point operations satisfy "ordering" properties
    - Monotonicity, values of signs
- Observation
  - Need to understand which abstractions apply in which contexts
  - Important issues for compiler writers and serious application programmers

Great Reality #2: You’ve Got to Know Assembly

- Chances are, you’ll never write full programs in assembly
  - Compilers are much better & more patient than you are
- But, assembly is key to the machine-level execution model
  - Behavior of programs in the presence of bugs
    - High-level language models break down
  - Tuning program performance
    - Understand optimizations done or not done by the compiler
    - Understanding sources of program inefficiency
  - Implementing system software
    - Compiler has machine code as target
  - Operating systems must manage process state
  - Creating / fighting malware
  - x86 assembly is the lingua franca

Assembly Code Example

- Time Stamp Counter
  - Special 64-bit register in Intel-compatible machines
  - Incremented every clock cycle
  - Read with rdtscl instruction
- Application
  - Measure time (in clock cycles) required by procedure

```
double t;
start_counter();
P();
t = get_counter();
printf("P required \%f clock cycles\n", t);
```

Great Reality #3: Memory Matters

Random Access Memory Is an Unphysical Abstraction

- Memory is not unbounded
  - It must be allocated and managed
  - Many applications are memory dominated
- Memory referencing bugs are especially pernicious
  - Effects are distant in both time and space
- Memory performance is not uniform
  - Cache and virtual memory effects can greatly affect program performance
  - Adapting program to characteristics of memory system can lead to major speed improvements

Code to Read Counter

- Write small amount of assembly code using GCC's asm facility
- Inserts assembly code into machine code generated by compiler

```
/* Return the cycle count as a 64-bit integer */
unsigned long access_counter(void) {
    unsigned long high, low;
    asm("rdtsc");
    return (high << 32) | low;
}
```

Great Reality #3: Memory Matters

Random Access Memory Is an Unphysical Abstraction

- Memory is not unbounded
  - It must be allocated and managed
  - Many applications are memory dominated
- Memory referencing bugs are especially pernicious
  - Effects are distant in both time and space
- Memory performance is not uniform
  - Cache and virtual memory effects can greatly affect program performance
  - Adapting program to characteristics of memory system can lead to major speed improvements

Great Reality #3: Memory Matters

Random Access Memory Is an Unphysical Abstraction

- Memory is not unbounded
  - It must be allocated and managed
  - Many applications are memory dominated
- Memory referencing bugs are especially pernicious
  - Effects are distant in both time and space
- Memory performance is not uniform
  - Cache and virtual memory effects can greatly affect program performance
  - Adapting program to characteristics of memory system can lead to major speed improvements
Memory Referencing Bug Example

```c
typedef struct {
    int a[2];
    double d;
} struct_t;

double fun(int i) {
    volatile struct_t s;
    s.d = 3.14;
    s.a[i] = 1073741824; /* Possibly out of bounds */
    return s.d;
}
```

**Explanation:**
- **Critical State:**
  ```
  6
  5
  4
  3
  2
  1
  0
  ```
- **Location accessed by:**
  ```
  fun(1)
  ```
- **Result is system specific**

Memory Referencing Errors

- **C and C++ do not provide any memory protection**
  - Out of bounds array references
  - Invalid pointer values
  - Abuses of malloc/free
- **Can lead to nasty bugs**
  - Whether or not bug has any effect depends on system and compiler
  - Action at a distance
    ```
    Corrupted object logically unrelated to one being accessed
    Effect of bug may be first observed long after it is generated
    ```
- **How can I deal with this?**
  - Program in Java, Python, Ruby, ML, etc.
  - Understand what possible interactions may occur
  - Use or develop tools to detect referencing errors (e.g. Valgrind)

Memory System Performance Example

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

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

**Great Reality #4: There’s more to performance than asymptotic complexity**

- **Constant factors matter too!**
- And even exact op count does not predict performance
  - Easily see 10:1 performance range depending on how code written
  - Must optimize at multiple levels: algorithm, data representations, procedures, and loops
- **Must understand system to optimize performance**
  - How programs compiled and executed
  - How to measure program performance and identify bottlenecks
  - How to improve performance without destroying code modularity and generality
Example Matrix Multiplication

Matrix-Matrix Multiplication (MMM) on 2 x Core 2 Duo 3 GHz (double precision)

- Standard desktop computer, vendor compiler, using optimization flags
- Both implementations have exactly the same operations count ($2n^3$)
- What is going on?

Best code (K. Goto)

Triple loop

- Reason for 20x: Blocking or tiling, loop unrolling, array scalarization, instruction scheduling, search to find best choice
- Effect: fewer register spills, L1/L2 cache misses, and TLB misses

Intermission: ChimeIn

- I'll periodically break up lectures with opportunities for you to think about the material and maybe talk with the people sitting next to you
- To anonymously submit answers, we'll use ChimeIn
- If you have a laptop with you, please go to: http://chimein.cla.umn.edu/
- And answer today's (non-CS) question
- (Can also supposedly set up to answer with a cell phone)

Course Perspective

- Most Systems Courses are Builder-Centric
  - Computer Architecture (CSci 4203)
  - Design pipelined processor in Verilog
  - Compilers (CSci 5161)
    - Write compiler for simple language
- 2021 is Programmer-Centric
  - Purpose is to show how by knowing more about the underlying system, one can be more effective as a programmer
  - Including, enable you to write programs that are more reliable and efficient
  - Not just a course for dedicated hackers
    - We bring out the hidden hacker in everyone
  - Cover material in this course that you won't see elsewhere

Role within Computer Science

- CSci 4211 Hardware
- CSci 4241 Design
- CSci 5204 Micro-Computer Architecture
- CSci 5203 Compiler

Mismatch: CSci 2021

Underlying principles for hardware and software

- Machine Architecture and Organization
- CSci 1039/1139
  - Programming, data structures
  - Virtual Memory

Things That Are Different This Semester

- Lab sections instead of discussion sections
  - More interactive discussion and hands-on assistance
- More in-lecture coverage of C and GDB
  - Tools you’ll use throughout the course
- Less coverage of some more specialized topics
  - E.g., Floating-point rounding, pipelining implementation, instruction-level parallelism
  - See the textbook if you’re still curious
- Allowing external references for homework assignments
  - Prohibition was unrealistic; but still not necessary or recommended
- Smaller to non-existent end-of-semester curve
  - Adjust for difficulty as we go, to be more predictable
Textbooks

- Required: Randal E. Bryant and David R. O’Hallaron,
  - [http://csapp.cs.cmu.edu](http://csapp.cs.cmu.edu)
  - Paper version recommended
  - Tests are open book, open notes, any paper, no electronics
  - Used quite heavily

- How to solve assignments
  - Practice problems with similar style as exam problems

Supplemental: a book about C

- Labs, homework, and tests require reading and writing code in C
- One free tutorial is recommended on the course site
- Other tutorial/reference books can also substitute

Course Components

- Lectures: Higher level concepts
- Lab Sections
  - Wednesdays in 1-250 Keller. Try new ideas out in a supportive environment, graded only on attendance.

- Hands-on Assignments (5)
  - The heart of the course, fun but often time-consuming
  - About 2 weeks each
  - Provide in-depth understanding of an aspect of systems
  - Programming and measurement

- Written Exercise Sets (5)
  - Practice thinking and writing, similar to tests, partially graded

- Two Midterms and a Final Exam
  - Test your understanding of concepts & mathematical principles

Electronic Resources

- Class Web Page:
  - Complete schedule of lectures, exams, and assignments
  - Lecture slides, assignments, practice exams, solutions
  - Watch for announcements

- Moodle Page
  - Discussion forums
  - Online turn-in of hands-on assignments

- Where to send electronic questions?
  1. Moodle forum
  2. cs2021f18-010-help@umn.edu (general mailing list)
  3. Individual staff members have higher latency

Policies: Assignments and Exams

- Groups? No.
  - You must work alone on all homework assignments

- Hand-in process
  - Hands-on assignments due online, by 11:55pm on a weekday evening
  - Exercise sets due on paper, by start of class on Mondays

- Conflicts
  - There will be no makeup midterms
  - One excused missed midterm will be replaced by more weight on final

- Appealing grades
  - Within 7 days of completion of grading
    - Following procedure described in syllabus and forum
  - Note, we will regrade the whole assignment/exam

Facilities

- Do labs using CSELabs Linux machines
  - Accessible from on-campus labs or remotely (VOLE/FastX, SSH)
  - Get an account if you don’t have one already, login with UMN account name and password
  - Working on your own machines may sometimes be possible, but is not a priority for support by course staff
  - Grade based on how it runs on our machines, so at least be sure to test there

Timeliness

- Late exercises and hands-on assignments
  - Late period is 24 hours from due date, 85% credit
  - For assignments after class, bring to instructor’s office (4-225E Keller)
  - No credit after 24 hours

- Catastrophic events
  - Major illness, death in family, … (full list in syllabus)
  - Are an exception, and can be excused

- Advice
  - The course is fast-paced
  - Once you start running late, it’s really hard to catch up
Cheating

- What is cheating?
  - Sharing code: by copying, retyping, looking at, or supplying a file
  - Coaching: helping your friend to write a lab, line by line
  - Copying code/text from previous course or from elsewhere on WWW
- What is NOT cheating?
  - Explaining how to use systems or tools
  - Helping others with high-level design issues
  - Getting ideas from public books or web sites, if you give credit
- Penalty for cheating:
  - Minimum: 0 grade on assignment or exam, report to campus OSCAI
- Detection of cheating:
  - We check with both human and automated efforts
  - Avoid surprises that would be unpleasant for all of us

C Language Basics

- Topics
  - Variables and operations, control flow and functions, data structures
  - Differences from Java and high-level C++
  - Just enough to get you started: various topics return in more depth later
- Assignments
  - HA1: Write a modest 19x3-style program, but in pure C

Data Representation

- Topics
  - Bit-level operations
  - Machine-level integers and floating point
  - C operators and things that can go wrong
- Assignments
  - HA2 (formerly "Data lab"): Manipulating bits

Machine-level Program Representation

- Topics
  - Assembly language programs
  - Representation of C control and data structures
  - E.g., what does a compiler do?
  - How dynamic memory allocation works
- Assignments
  - HA3 (formerly "Bomb lab"): Defusing a binary bomb with a debugger
  - HA4 (formerly "Malloc lab"): Implement your own memory allocator

CPU Architecture

- Topics
  - The parts of a CPU and how they work together
  - How CPUs save time by doing multiple things at once (pipelining)
- Lab activities
  - Work with a CPU simulator
  - Implement your own instruction

Policies: Grading

- Exams (60%): weighted 15%, 15%, 30% (final)
- Hands-on Assignments (20%)
- Written Exercise Sets (15%)
- Attending at least 12 out of 15 lab sections (5%)
- Guaranteed:
  - ≥ 85%: at least A-
  - ≥ 72%: at least B-
  - ≥ 60%: at least C-
- Curve:
  - May apply, in your favor only, so that grade distribution is similar to historical averages.
The Memory Hierarchy

- **Topics**
  - Memory technology, memory hierarchy, caches, disks, locality
  - How virtual memory works

- **Assignments**
  - HA5: Simulate and optimize cache behavior

Shorter Topics

- **Optimization**
  - Some code features that are good or bad for performance
  - Profiling code to know what parts are slow

- **Linking**
  - How compilers put code and data together into a final program
  - How code from libraries can be loaded as a program runs

Welcome and Enjoy!