Course Overview

15-213 (18-213): Introduction to Computer Systems
1st Lecture, Aug. 26, 2014

Instructors:
Greg Ganger, Greg Kesden, and Dave O’Hallaron

The course that gives CMU its “Zip”!
Overview

- Course theme
- Five realities
- How the course fits into the CS/ECE curriculum
Course Theme:
Abstraction Is Good But Don’t Forget Reality

- Most CS and CE 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 from taking 213
  - 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 & ECE
    - Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems, Storage Systems, etc.
Great Reality #1: Ints are not Integers, Floats are not Reals

Example 1: Is $x^2 \geq 0$?
- Float’s: Yes!
- Int’s:
  - $40000 \times 40000 \rightarrow 1,600,000,000$
  - $50000 \times 50000 \rightarrow ??$

Example 2: Is $(x + y) + z = x + (y + z)$?
- Unsigned & Signed Int’s: Yes!
- Float’s:
  - $(1e20 + -1e20) + 3.14 \rightarrow 3.14$
  - $1e20 + (-1e20 + 3.14) \rightarrow ??$

Source: xkcd.com/571
Computer Arithmetic

- Math 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 programs in assembly
  - Compilers are much better & more patient than you are

- But: Understanding assembly is key to machine-level execution model
  - Behavior of programs in presence of bugs
    - High-level language models break down
  - Tuning program performance
    - Understand optimizations done / 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 language of choice!
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 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
```c
double fun(int i)
{
    volatile double d[1] = {3.14};
    volatile long int a[2];
    a[i] = 1073741824; /* Possibly out of bounds */
    return d[0];
}

fun(0) ➞ 3.14
fun(1) ➞ 3.14
fun(2) ➞ 3.1399998664856
fun(3) ➞ 2.00000061035156
fun(4) ➞ 3.14, then segmentation fault
```

- Result is architecture specific
Memory Referencing Bug Example

double fun(int i)
{
    volatile double d[1] = {3.14};
    volatile long int a[2];
    a[i] = 1073741824; /* Possibly out of bounds */
    return d[0];
}

fun(0) ➔ 3.14
fun(1) ➔ 3.14
fun(2) ➔ 3.1399998664856
fun(3) ➔ 2.00000061035156
fun(4) ➔ 3.14, then segmentation fault

Explanation:

<table>
<thead>
<tr>
<th>Saved State</th>
</tr>
</thead>
<tbody>
<tr>
<td>d7 ... d4</td>
</tr>
<tr>
<td>d3 ... d0</td>
</tr>
<tr>
<td>a[1]</td>
</tr>
<tr>
<td>a[0]</td>
</tr>
</tbody>
</table>

Location accessed by fun(i)
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, Ruby, Python, ML, ...
  - Understand what possible interactions may occur
  - Use or develop tools to detect referencing errors (e.g. Valgrind)
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
Memory System Performance Example

- Hierarchical memory organization
- Performance depends on access patterns
  - Including how step through multi-dimensional array

```c
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];
}

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];
}
```

5.2ms  2.8 GHz Intel Core i7  162ms
Great Reality #5: Computers do more than execute programs

- They need to get data in and out
  - I/O system critical to program reliability and performance

- They communicate with each other over networks
  - Many system-level issues arise in presence of network
    - Concurrent operations by autonomous processes
    - Coping with unreliable media
    - Cross platform compatibility
    - Complex performance issues
Role within CS/ECE Curriculum

- CS 415 Databases
- CS 441 Networks
- CS 410 Operating Systems
- CS 411 Compilers
- CS 440 Distributed systems
- CS 412 OS Practicum
- ECE 340 Digital Computation
- ECE 447 Architecture
- ECE 349 Embedded Systems
- ECE 545/549 Capstone
- ECE 348 Embedded System Eng.

Data Reps. Memory Model
Network Protocols
Processes Mem. Mgmt
Machine Code
Arithmetic
Execution Model Memory System

Network Prog Concurrency
Concurrency

Foundation of Computer Systems
Underlying principles for hardware, software, and networking

CS 122 Imperative Programming

213
Course Perspective

- **Most Systems Courses are Builder-Centric**
  - Operating Systems
    - Implement large portions of operating system
  - Distributed Systems
    - Build services and applications that use multiple computers
  - Embedded Systems
    - Develop control software for embedded hardware
  - Compilers
    - Write compiler for simple language
  - Computer Architecture
    - Design pipelined processor in Verilog
  - Networking
    - Implement and simulate network protocols
Course Perspective (Cont.)

- Our Course is Programmer-Centric
  - Purpose is to show that by knowing more about the underlying system, one can be more effective as a programmer
  - Enable you to
    - Write programs that are more reliable and efficient
    - Incorporate features that require hooks into OS
      - E.g., concurrency, signal handlers
  - Cover material in this course that you won’t see elsewhere
  - Not just a course for dedicated hackers
    - We bring out the hidden hacker in everyone!
Teaching staff

Greg Ganger

Dave O’Hallaron

Greg Kesden