15-213
"The Class That Gives CMU Its Zip!"™

Introduction to Computer Systems

Greg Ganger
August 29, 2007
(adapted from Randal E. Bryant's slides)

Topics:
- Theme
- How this fits within CS curriculum
- Five great realities of computer systems

15-213 Theme

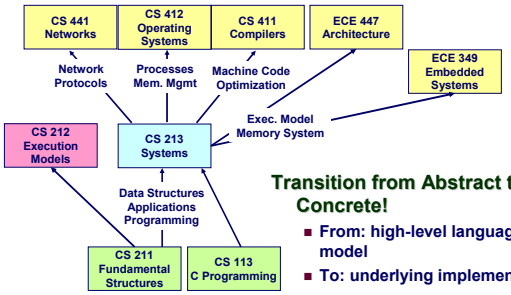
- Abstraction is good, but don't forget reality!

Most programming classes emphasize abstraction
- Abstract data types
- Asymptotic analysis

These abstractions have limits
- And those limits can result in nasty bugs
- Need to understand underlying implementations

Useful outcomes
- Become more effective programmers
  - Able to find and eliminate bugs efficiently
  - Able to tune program performance
- Prepare for later "systems" classes in CS & ECE
  - Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems

Role within Curriculum



Transition from Abstract to Concrete!
- From: high-level language model
- To: underlying implementation

Course Perspective

Most "Systems" Courses are Builder-Centric
- Computer Architecture
  - Design pipelined processor in Verilog
- Operating Systems
  - Implement large portions of operating system
- Compilers
  - Write compiler for simple language
- Networking
  - Implement and simulate network protocols

Course Perspective (Cont.)

15-213 is Programmer-Centric
- Purpose is to show how 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
- Not just a course for dedicated hackers
  - We bring out the hidden hacker in everyone
- Lets talk about some example realities

Great Reality #1

Int's are not Integers, Float's are not Reals

Examples
- Is $x^2 \geq 0$?
  - Float's: Yes!
  - Int's:
    - » 40000 * 40000 --> 1600000000
    - » 50000 * 50000 --> ??
- Is (x + y) + z = x + (y + z)?
  - Unsigned & Signed Int's: Yes!
  - Float's:
    - » (1e20 + -1e20) + 3.14 --> 3.14
    - » 1e20 + (-1e20 + 3.14) --> ??

Computer Arithmetic

Should not generate random values
- Arithmetic operations have important mathematical properties

But, cannot assume "usual" 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

Memory Matters: **Random Access Memory is an un-physical 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

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 (and painful debugging
- Whether or not bug has any effect depends on system+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

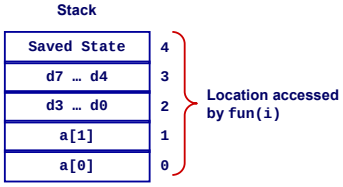
How can I deal with this?
- Never make mistakes
- Program in Java, Lisp, or ML
- Understand what possible interactions may occur
- Use or develop tools to detect referencing errors

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
```


Referencing Bug Explanation



- C does not implement bounds checking
- Out of range write can affect other parts of program state

Great Reality #3

There's more to performance than asymptotic complexity

Constant factors matter too!
- 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

```
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];
}
```

59,393,288 clock cycles

1,277,877,876 clock cycles

21.5 times slower!

(Measured on 2GHz Intel Pentium 4)

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

Great Reality #4

You've got to know assembly

Chances are, you'll never write program in assembly

- Compilers are much better & more patient than you are

But, understanding assembly enables one to understand machine-level execution behavior

- Behavior of programs in presence of bugs
  - When high-level language model breaks down
- Tuning program performance
  - Understanding sources of program inefficiency
- Implementing system software
  - Compiler has machine code as target
  - Operating systems must manage device and process state
- Creating / fighting malware
  - x86 assembly is the language of choice

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

What's next

Data representation (Fri): bits, bytes, and integers

- Reading
  - 2.1-2.3
- Suggested problems
  - 2.44, 2.45, 2.49, 2.54
- First lab will be handed out
  - and it will be due two weeks from today
- Welcome to 15-213! ☺