Course Overview

CSE4009: System Programming

Woong Sul

Overview

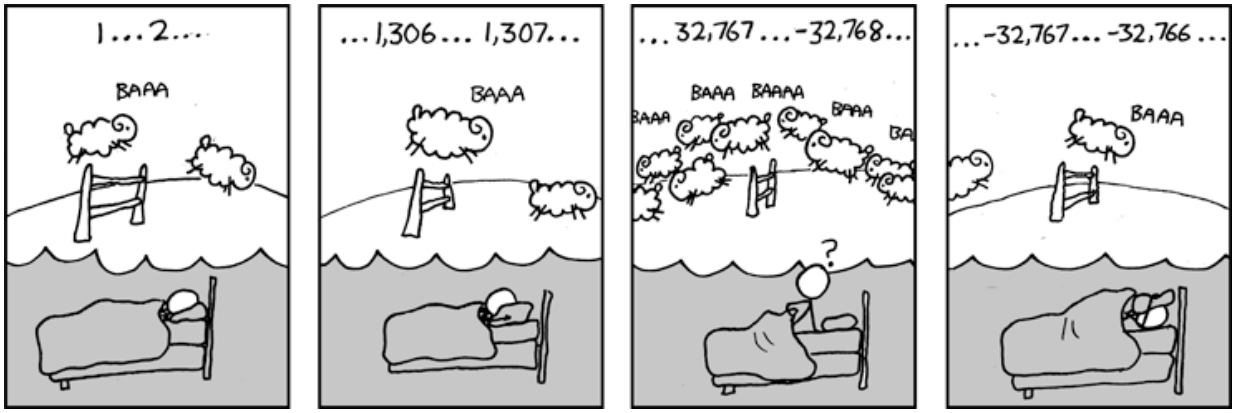
- Course Theme
- Four Realities
- Policies

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 this class**
  - 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
    - Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems, etc.

Great Reality #1: Ints and Floats in Computer

- **Example 1: Is $x^2 \geq 0$?**
  - Float's: Yes!



  - Int's:
    - 40000 * 40000 → 1600000000
    - 50000 * 50000 → ??
- **Example 2: Is (x + y) + z = x + (y + z)?**
  - Unsigned & Signed Int's: Yes!
  - Float's:
    - (1e20 + -1e20) + 3.14 --> 3.14
    - 1e20 + (-1e20 + 3.14) --> ??

Source: xkcd.com/571

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: 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

- **Memory referencing bugs especially pernicious**
  - Effects are distant in both time and space
- **Memory is not unbounded**
  - It must be allocated and managed
  - Many applications are memory dominated
- **Memory performance is not uniform**
  - *Cache* & *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

```
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;
}
```

```
fun(0)   →    3.14
fun(1)   →    3.14
fun(2)   →    3.1399998664856
fun(3)   →    2.00000061035156
fun(4)   →    3.14
fun(6)   →    Segmentation fault
```

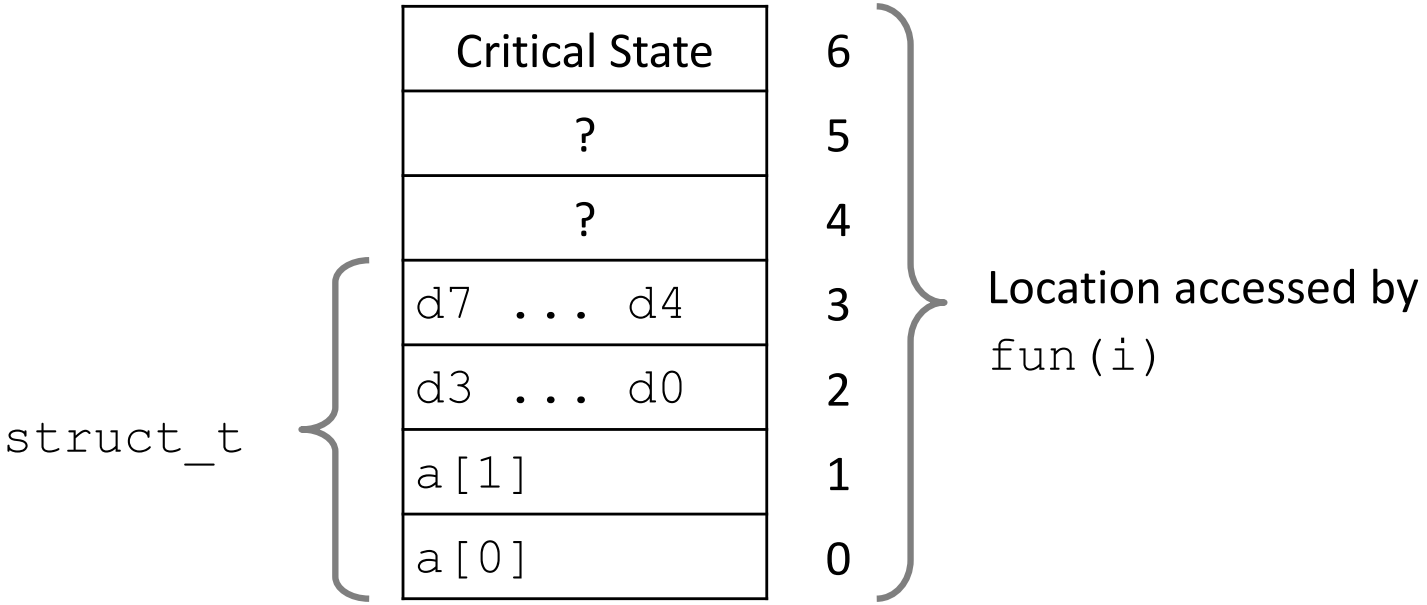
- Result is system specific

Memory Referencing Bug Example

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

```
fun(0)  →  3.14
fun(1)  →  3.14
fun(2)  →  3.1399998664856
fun(3)  →  2.0000000061035156
fun(4)  →  3.14
fun(6)  →  Segmentation fault
```

Explanation:



Memory Referencing Errors

- **C and C++ do not provide any memory protection**
  - Out of bounds array references
  - Invalid pointer values
  - Abuses of malloc/free
- **Then it could 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: Performance

- **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

```
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];
}
```

4.3ms 2.0 GHz Intel Core i7 Haswell 81.8ms

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

Course Perspective

- Most Systems Courses are Builder-Centric
  - Computer Architecture
    - Design pipelined processor in Verilog
  - Operating Systems
    - Implement sample portions of operating system
  - Compilers
    - Write compiler for simple language
  - 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!**

Textbooks

- Randal E. Bryant and David R. O'Hallaron,
  - ***Computer Systems: A Programmer's Perspective***, **Third Edition** (CS:APP3e), Pearson, 2016
  - This book really matters for the course!
    - How to solve labs
    - Practice problems typical of exam problems

- Brian Kernighan and Dennis Ritchie,
  - ***The C Programming Language***, Second Edition, Prentice Hall, 1988
  - Still the best book about C, from the originators

Course Schedule

Week	Topics
1	Course Introduction
2	Data: bits/integers 1
3	Data: floats
4	Programming: basics
5	Programming: control & procedure
6	Programming: data
7	Optimization
8	**Midterm exam (10/25)**
9	Linking
10	ECF: exceptions & processes
11	ECF: signals
12	Virtual Memory
13	Memory Allocation
14	System-level I/O
15	**Final exam (12/13)**

Programs and Data

- Topics
  - Bits operations, arithmetic, assembly language programs
  - Representation of C control and data structures
  - Includes aspects of architecture and compilers

Exceptional Control Flow

- Topics
  - Hardware exceptions, processes, process control, Unix signals, nonlocal jumps
  - Includes aspects of compilers, OS, and architecture

Virtual Memory

- Topics
  - Virtual memory, address translation, dynamic storage allocation
  - Includes aspects of architecture and OS

Class Structure

- 1 lecture class per a week
  - 1-2 video will be released each week

- 1 lab class per a week
  - Review, discussion, and recitation of the lecture of the week
  - Required your own Linux computer

Grading: On a Curve

- Exams (70%)
  - Each exam accounts for 35%
- Assignments (20%)
  - Each assignment accounts for 5%
- Etc (10%)

- Attendance Policy
  - *Absence* means no show or leaving w/o due notice
  - More than 10 absences → **F**
  - Three lates will be regarded as one absence

Questions

- Q&A Boards on piazza
  - https://piazza.com/hanyang.ac.kr/fall2025/cse4009profsul
  - Access code: cse4009_woong
  - Either Korean and English would be fine
  - Enrollment required
    - Combination of your name and the last two digits of your student ID

      2012123456 Woong Sul ➔ wsul_56, woong_56 would be OK

Welcome and Enjoy!