Course Overview

CSE251: System Programming
1st Lecture, Feb. 25, 2019

Instructor:

Hyungon Moon

All slides for this course are from:

https://www.cs.cmu.edu/afs/cs/academic/class/15213-f15/www/schedule.html

Overview

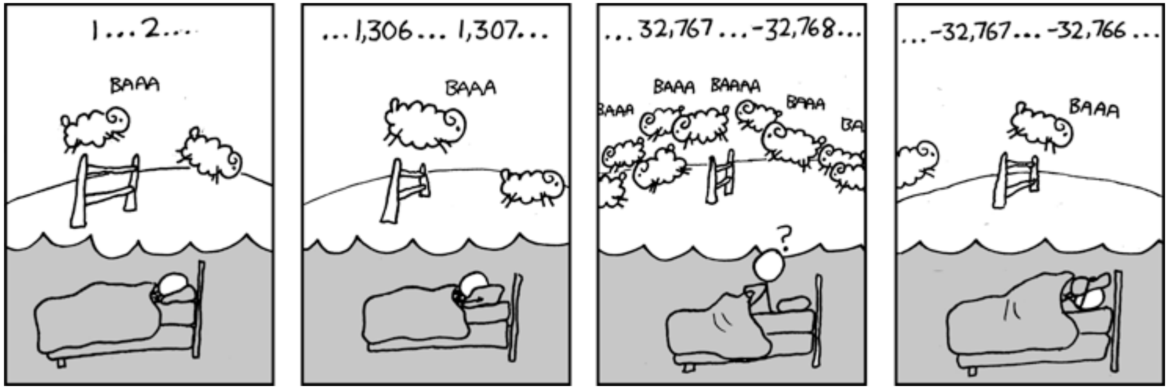
- Course theme
- Five realities
- Course logistics

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

- **Most CSE 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 251**
  - 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 CSE
    - Compilers, Operating Systems, Networks, Computer Architecture, 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 * 40000 ➞ 1600000000
    - 50000 * 50000 ➞ ??



Source: xkcd.com/571

- **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) --> ??

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 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

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
```

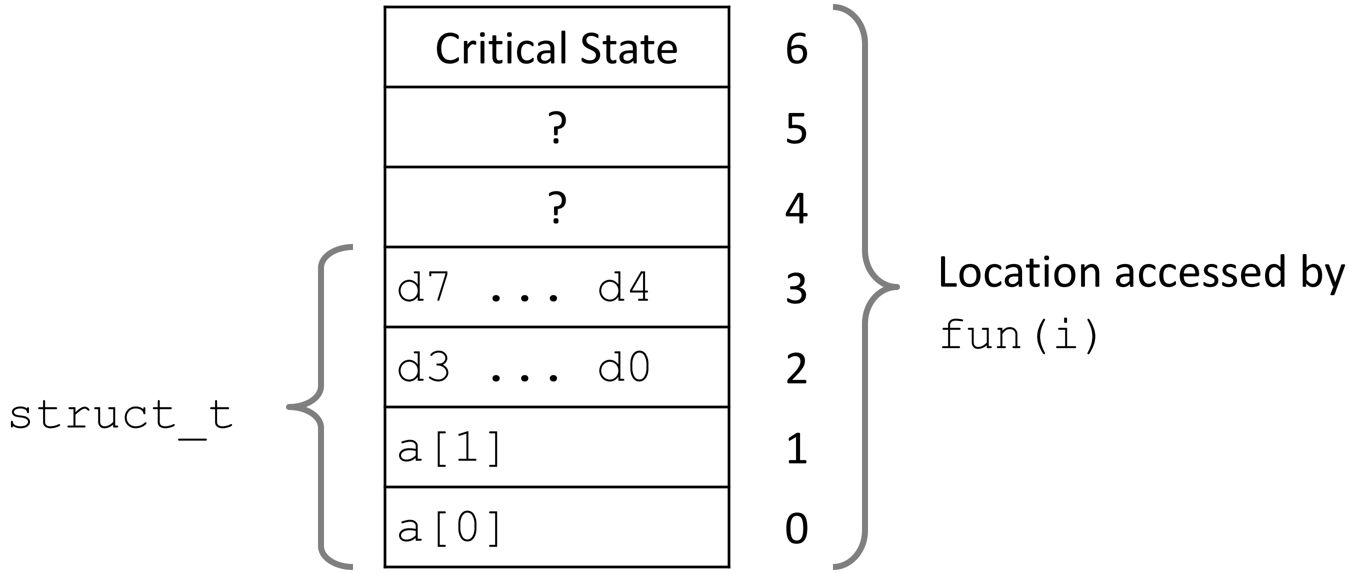
- Result is system specific

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)  →   2.00000061035156
fun(4)  →   3.14
fun(6)  →   Segmentation fault
```

Explanation:



Memory Referencing Errors

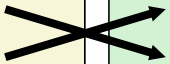
- **C and C++ do not provide any memory protection**
  - Out of bounds array references
  - Invalid pointer values
  - Abuses of malloc/free
- **Can lead to nasty bugs**
  - Whether or not bug has any effect depends on system and compiler
  - Action at a distance
    - Corrupted object logically unrelated 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, asan)

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

```
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
- Performance depends on access patterns
  - Including how step through multi-dimensional array

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

Course Perspective

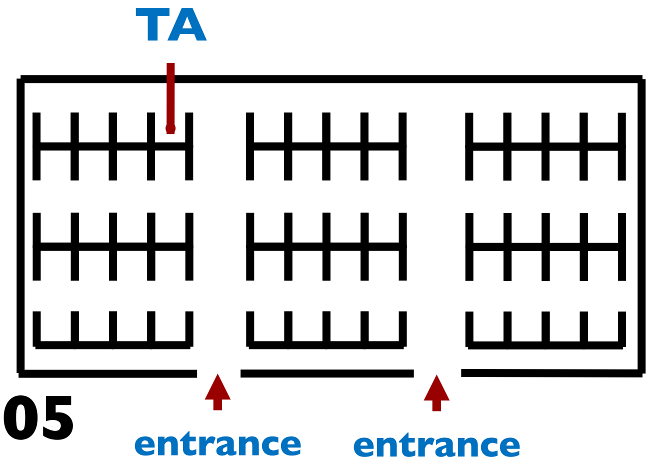
- Most Systems Courses are Builder-Centric
  - Computer Architecture
    - Design pipelined processor in Verilog
  - Operating Systems
    - Implement sample portions of operating system
  - Compilers
    - Write compiler for simple language
  - Networking
    - Implement and simulate network protocols

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!**

Logistics: Teaching staff

- Instructor
  - Hyungon Moon (hyungon@unist.ac.kr)
  - Office hour: by email
    - Best timing to email: ~ 8am in the morning.
  - Office: 106-701-8
- TAs
  - Sehoon Kim (sshhhee@unist.ac.kr) @ 106-605
  - TBD1
  - TBD2
  - TBD3



106 - 605

Grading

- General rule
  - 5%: Class participation
  - 20%: Midterm
  - 25%: Final
  - 50%: Lab
- Special rule
  - Not submitting two or more lab will make your grade C+ or lower.
  - We follow university and school policies about attendance and cheating.
    - e.g., Missing 8 or more classes will make your grade F.
  - Class participation: online discussion, in-class participation.
    - **Doesn't** include the attendance.

The *official* ECE rule on the Academic Integrity

- On the 1st violation
  - **Zero grade on the item** involved (e.g., homework, midterm, etc.).
  - **Lower the final grade by at least one letter grade** (e.g., B+ --> C+).
  - Attain **a "signed" personal letter from the student** stating this will not happen again, and he/she is well aware of consequence if it does.
  - **Provide a written report** of the student and violation **to School Head and ECE education committee**.
- On the 2nd violation
  - **Give an F on the course**.
  - **Share the identity of the student** with the entire faculty.
  - **Report to the University Student Scholarship Counseling Committee** for further disciplinary action.

Textbook

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

Course Components

- Lectures
  - Higher level concepts
- Recitations
  - Applied concepts, important tools and skills for labs, clarification of lectures, exam coverage
- Labs (7+1)
  - The heart of the course
  - 1-2 weeks each
  - Provide in-depth understanding of an aspect of systems
  - Programming and measurement
- Exams (midterm + final)
  - Test your understanding of concepts & mathematical principles

Getting Help

- Class Gitlab: https://class.unicss.org/cse251-2019-spring
- Accounts:
  - Registered students will get password reset email by 2/26. Get registered!
- General questions:
  - Create/check issues on the master repository.
    - https://class.unicss.org/cse251-2019-spring
- Private questions:
  - You are not likely to have these.
  - Send an email or create an issue on your submission repository.
- Blackboard or Piazza
  - We won’t be using Blackboard or Piazza for the course

Policies

- Work groups
  - You must work alone on all lab assignments
- Handins
  - Labs due at 11:59pm on each deadline
  - Electronic handins using the class Gitlab.
  - The Lab 0 will cover the basics of git.

Facilities

- Labs will be tested on the uni server
  - `linux> ssh uni06.unist.ac.kr`

- Getting help with the uni server
  - Please create an issue on the class Gitlab.

Timeliness

- No late submissions allowed.
- Depending on the labs, you may get earlier feedback:
  - You will have a chance to revise your submission, only if you submit it early.
- ***Start early***
  - It could take quite a lot of time: you should start early to finish and submit it.

Other Rules of the class room

- Laptops: permitted

- Electronic communications: forbidden
  - No email, instant messaging, cell phone calls, etc

- Presence in lectures, recitations: voluntary, recommended

- No recordings of ANY KIND

Programs and Data

- Topics
  - Becoming familiar with git, the de facto standard version control system.
- Assignments
  - L0 (gitlab): Trying out the lab infra and becoming familiar with Linux shell.

Programs and Data

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

- Assignments
  - L1 (datalab): Manipulating bits
  - L2 (bomblab): Defusing a binary bomb
  - L3 (attacklab): The basics of code injection attacks

The Memory Hierarchy

- Topics
  - Memory technology, memory hierarchy, caches, disks, locality
  - Includes aspects of architecture and OS

- Assignments
  - L4 (cachelab): Building a cache simulator and optimizing for locality.
    - Learn how to exploit locality in your programs.

Exceptional Control Flow

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

- Assignments
  - L5 (tshlab): Writing your own Unix shell.
    - A first introduction to concurrency

Virtual Memory

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

- Assignments
  - L6 (malloclab): Writing your own malloc package
    - Get a real feel for systems-level programming

Networking, and Concurrency

- Topics
  - High level and low-level I/O, network programming
  - Internet services, Web servers
  - concurrency, concurrent server design, threads
  - I/O multiplexing with select
  - Includes aspects of networking, OS, and architecture

- Assignments
  - L7 (proxylab): Writing your own Web proxy
    - Learn network programming and more about concurrency and synchronization.

Lab Rationale

- Each lab has a well-defined goal such as solving a puzzle or winning a contest

- Doing the lab should result in new skills and concepts

- We try to use competition in a fun and healthy way
  - Set a reasonable threshold for full credit

Gitlab showcase

- There are plenty of resources that you can follow.

- The steps you are going to follow:
  - Create a password and login
  - Read the syllabus (README.md file on the master repository)
  - Create and register your ssh key
    - https://docs.gitlab.com/ee/ssh/

Welcome
and Enjoy!