Systems Programming

Introduction

Textbook coverage:
Ch 2: Representing and Manipulating Information

Byoungyoung Lee
Seoul National University
byoungyoung@snu.ac.kr
https://lifeasageek.github.io

Lecture slides are prepared based on materials provided by CSAPP authors.

Overview

- **Introductions**
- **Big Picture**
  - Course theme
  - Five realities
  - How the course fits into the CS/ECE curriculum
- **Academic integrity**
- **Logistics and Policies**

About Instructor: Byoungyoung Lee

- **Research Area: Hacking, Systems Security, Software Security**
  - Microsoft Research, Research Intern (2012)
  - Google, Software Engineering Intern (2014)
  - Purdue University, Assistant Professor (2016-2018)
  - Seoul National University, Assistant/Associate/Full Professor (2018-Current)

- **Three times DEFCON CTF Finalist (2007,2009, and 2011)**
- **Internet Defense Prize by Facebook and USENIX (2015)**
- **DARPA Cyber Grand Challenge (CGC) Finalist (2016)**
- **Google ASPIRE Awards (2019)**
- **Found 100++ vulnerabilities from Windows kernel, Linux kernel, Chrome, Firefox, etc.**

TAs

- **Jaeyoung Chung**
  - jjy600901@snu.ac.kr
  - https://github.com/J-jaeyoung
- **Eulgyu Kim**
  - eulgyukim@snu.ac.kr
  - https://eulgyukim.github.io
- **Jonguk Cheong**
  - jdal3031@snu.ac.kr
  - https://jongukc.github.io

The Big Picture

Course Theme:
(Systems) Knowledge is Power!

- **Systems Knowledge**
  - How hardware (processors, memories, disk drives, network infrastructure) plus software (operating systems, compilers, libraries, network protocols) combine to support the execution of application programs
  - How you as a programmer can make the best use of these resources

- **Useful outcomes from taking this course**
  - Become more effective programmers
    - Able to understand and tune for program performance
    - Able to find and eliminate bugs efficiently
  - Prepare for later “systems” classes in CS & ECE
    - Compilers, Operating Systems, Networks, Computer Architecture
    - Embedded Systems, Storage Systems, Software/Systems Security, etc.

It's Important to Understand How Things Work

- **Why do I need to know this stuff?**
  - Abstraction is good, but don't forget reality
- **Most Computer Science/Engineering courses emphasize abstraction**
  - Abstract data types
  - Asymptotic analysis
- **These abstractions have limits**
  - Abstraction doesn't necessarily mean that you can ignore the details
  - Need to understand details of underlying implementations
  - Sometimes the abstract interfaces don't provide the level of control or performance you need

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 --> ?
- **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
- **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
    - Undefined behavior w.r.t. the high-level language model
  - Tuning program performance
    - Understand optimizations done / not done by the compiler
  - Implementing system software
    - You will need to write machine-specific code
    - Virtual memory, interrupt handling, etc.
      - which needs a very precise control
  - Reverse-engineering
    - Understanding proprietary software
    - Analyzing malware

Great Reality #3: Memory Matters

Random Access Memory Is Abstracted Resource

- **Memory is a bounded resource**
  - It must be allocated and managed
  - Many applications are memory dominated
- **Memory referencing bugs are critical**
  - Due to space and time issues (spatial/temporal memory bugs)
- **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 memory protection**
  - Out of bounds array references
  - Invalid pointer values

- **Can lead to nasty bugs**
  - The same bug may show different errors
    - Depending on compilers/systems/machines/etc.
  - Why difficult?
    - Distance in space: Corrupted object logically unrelated to one being accessed
    - Distance in time: Effect of bug may be first observed long after it is generated

- **How can I deal with this?**
  - Program in Java, Ruby, Python, ML, Go, Rust, …
  - Understand what possible interactions may occur
  - Use or develop tools to detect referencing errors (e.g. Valgrind, AddressSanitizer)

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

- **Constant factors matter too!**
  - Algorithmic asymptotic complexity may not matter much
- **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
  - How to identify bottlenecks

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 **81.8ms**

2.0 GHz Intel Core i7 Haswell

- **Hierarchical memory organization**
- **Performance depends on access patterns**
  - Including how multi-dimensional array is stepped through

Great Reality #5:
Computers do more than execute programs

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

- **They communicate with each other over networks**
  - Many system-level issues arise in presence of network
    - Coping with unreliable media
    - Cross platform compatibility

Course Perspective

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

Course Perspective (Cont.)

- **This Course is Programmer-Centric**
  - By knowing more about the underlying system, you can be more effective as a programmer
  - Enable you to
    - Prepare yourself as a “true” system programmer
    - Write programs that are more reliable and efficient
    - Troubleshoot your program with systems knowledge
    - Incorporate systems features in your program
      - E.g., concurrency, signal handlers

Primary Textbook

- ***Computer Systems: A Programmer's Perspective*, Third Edition (CS:APP3e), Pearson, 2016**
  - Randal E. Bryant and David R. O'Hallaron,
  - https://csapp.cs.cmu.edu
  - Electronic editions available

Recommended reading

- The C Programming Language, Second Edition, Prentice Hall, 1988

- The Linux Programming Interface: A Linux and UNIX System Programming Handbook, 1st Edition, No Starch Press (2010)

Course Outline: Programs and Data

- **Topics**
  - Assembly language programs
  - Representation of C control and data structures
  - Includes aspects of architecture and compilers

Course Outline: The Memory Hierarchy

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

Course Outline: Virtual Memory

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

Course Outline: Exceptional Control Flow

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

Course Outline: 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

Lab

- **CTF**
  - Hacking the system!
  - Problem solving
  - 5-7 CTF problems
    - Stack buffer overflow, heap overflow, use-after-free
    - PLT/GOT, Fault handling, race condition,
- **Programming Assignments**
  - Implementation assignments
  - Assembly, Malloc, Shell, Debugger, Proxy
- **All information will be available at the class homepage**
  - **https://compsec.snu.ac.kr/class/systems-programming**

Lab: What you should know before and what you will learn

- **What you should know before**
  - C, C++, Python
  - Data structures

- **What you will learn**
  - x86 asm
  - You will taste how the real-world systems are implemented!
  - Tooling
    - Linux
    - gdb
    - pwntools

Academic Integrity

- **If you happen to do**
  - Cheating
  - Plagiarism

- **You will very likely get F**
  - And your mis-conduct will be reported to the student council

Plagiarism

- **All projects should be done individually**
  - You should not directly copy the code from any other source
  - Do not share your code!
    - The one who shared the code will get the same penalty!
  - We will run plagiarism detection software!
  - You may discuss and help others, but you should write your own code