Course Overview

ITM424:
Computer Systems
1st Lecture, Sep. 7, 2023

Instructors:
Moon Hea-Eun (Ocean)

Overview

- Introductions
- Big Picture
  - Course theme
  - Five realities
  - How the course fits into the CS/ECE/INI curriculum
- Logistics

Instructor

Bio

- 2014 ~ Now : NSHC Inc. Red Alert Research Institute Director
- 2002 ~ 2014 : NetMan Co., Ltd. Research Institute Director

Latest Activities

- 2020.02 : IAEA Conference Speaker in Austria
- 2019.08 : DEFCON ICS/SCADA CTF Organizer
- 2019.08 : CISS SCADA CTF – Winner
- 2018.08 : DEFCON ICS/SCADA CTF Organizer
- 2018.07 : IOT Singapore CTF – 2nd Winner
- 2017.11 : CODEBLUE Hack2Win 2017 – Winner
- 2017.08 : CISS SCADA CTF – Winner
- 2015.11 : CODEBLUE Hack2Win 2015 – Winner
- 2015.10 : Whitehat Contest Final
- 2014.10 : Whitehat Contest Final 3rd
- 2013.08 : DEF CON, Capture The Flag 8th
- 2010.10 : ISEC 2010 Capture The Flag winner
- 2010.08 : DEF CON, Capture The Flag 6th

- ocean.moon@seoultech.ac.kr
- mayfly74@gmail.com

Instructor - Research

- Offensive Research
- Bug hunting research
- More Focus on IoT, ICS/SCADA
- Found over 255 real world 0-day vulnerabilities
- Specialize in Red Teaming Exercise
- Provide Consulting, Test Bed, CTF, Penetration Test, Vulnerability Assessment and Training Service

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 / hacker can best use these resources
- **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 & ECE
    - Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems, Storage Systems, etc.

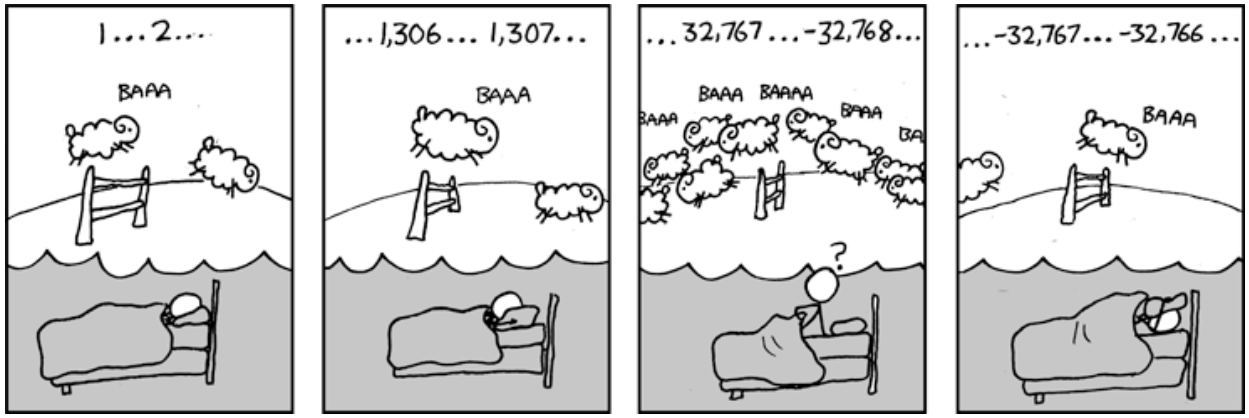
It's Important to Understand How Things Work

- **Why do I need to know this stuff?**
  - Abstraction is good, but don't forget reality
- **Most CS and CE courses emphasize abstraction; so do we!**
  - Abstract data types
  - Asymptotic analysis
- **But it's helpful to understand what abstractions build upon and their limits!**
  - Especially in the presence of bugs
  - 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

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
```

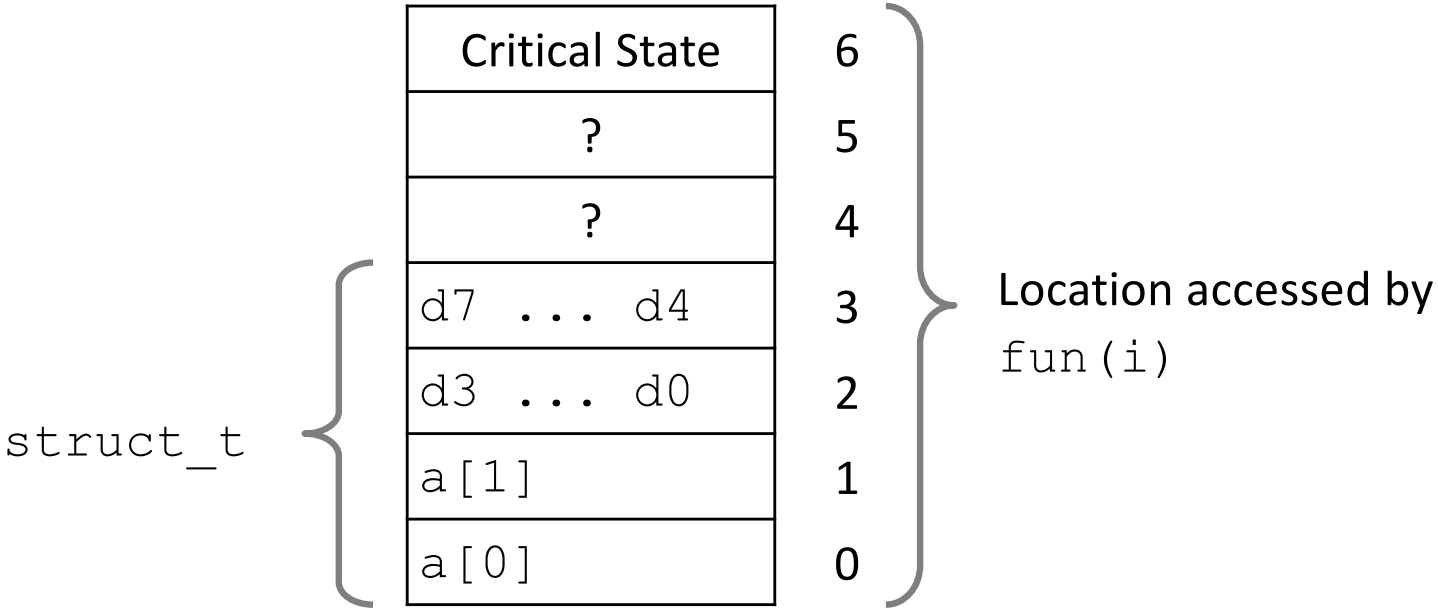
- Result is system specific

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) -->      2.0000006103515 6
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)

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 81.8ms

2.0 GHz Intel Core i7 Haswell

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

Logistics

Primary Textbook

- Randal E. Bryant and David R. O'Hallaron,
  - *Computer Systems: A Programmer's Perspective*, **Third Edition** (CS:APP3e), Pearson, 2016
  - https://csapp.cs.cmu.edu
  - This book really matters for the course!
    - How to solve labs
    - Practice problems typical of exam problems
  - Electronic editions available (*Don't get paperback/international version*!)

- Note: All textbooks have errors
  - Don't panic if you see something that seems wrong
  - Come talk to us about it if you can't make it make sense

Recommended reading

- Brian Kernighan and Dennis Ritchie,
  - *The C Programming Language*, Second Edition, Prentice Hall, 1988
  - Everyone calls this book “K&R”
  - Guide to C by the designers of the language
  - Well-written, concise
  - A little dated
    - Doesn’t cover additions to C since 1988 (that’s thirty years ago…)
    - Casual about issues we consider serious problems now

If you want more books about C

- C for Programmers with an introduction to C11
  - Paul and Harvey Deitel
  - Opposite of K&R: modern, verbose
  - Lots of worked-out examples
  - Ugly code style (compare readability to K&R)
- 21st Century C
  - Ben Klemens
  - Supplement to full C textbooks: goes into the corners of the language
  - Opinionated
  - First half is about how to *build* C programs in the Unix environment
    - So, if you want to understand the Makefiles we give you…
- Learn C the Hard Way
  - Zed A. Shaw
  - Extremely opinionated
  - Also has lots of worked-out examples
  - Only book I can find that takes “undefined behavior” seriously enough

Course Components

- Lectures
  - Higher level concepts
  - Provide in-depth understanding of an aspect of systems
  - Programming and measurement
- Assignments
  - Reinforce concepts
  - 2023-2_ITM424_Assignment.doc
- Exam
  - Test your understanding of concepts & mathematical principles
  - Covers content from the whole semester
- Grading system
  - Homework (30, Mid-term (30) and Final (40)

Programs and Data

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

The Memory Hierarchy

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

Exceptional Control Flow

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

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

Schedule

- Topics

W	Date	Contents
1	2023.09.08	Linux Overview
2	2023.09.15	C Programming Language
3	2023.09.22	Bits, Bytes, & Integers
4	2023.09.29	Machine Level Programming (Basic, Control)
5	2023.10.06	Machine Level Programming (Procedures, Data)
6	2023.10.13	Machine Level Programming (Advanced)
7	2023.10.20	Mid Term Exam
8	2023.10.27	Design and Debugging
9	2023.11.03	The Memory Hierarchy
10	2023.11.10	Linking
11	2023.11.17	Processes and Multitasking, Exceptional Control Flow
12	2023.11.24	System Level I/O
13	2023.12.01	Network Programming
14	2023.12.08	Network Programming
15	2023.12.15	Final Exam

Getting Help

- Class Web page:
  - e-Class
  - Complete schedule of lectures, exams, and assignments
  - Copies of lectures, assignments, exams, solutions
  - FAQ
- TA
  - Info.
- Email
  - Feel free to ask any questions

Welcome and Enjoy!