Course Overview

CS230: System Programming
1st Lecture, Aug. 31, 2021

Instructors:

Jongse Park

Jongse Park

- Background
  - Assistant professor, CASys lab, KAIST (2019 – present)
  - Deep learning acceleration system architect, Bigstream (2018 – 2019)
  - Ph.D. from ACT Lab, Georgia Institute of Technology (2018)
  - M.S. from CASys lab, KAIST (2012)
  - B.S. from Sogang University (2010)
- Contact
  - E3-1, Room 4403
  - jspark@casys.kaist.ac.kr
  - https://jongse-park.github.io/
- Research Interests
  - Computer Architecture
  - Systems for Artificial Intelligence
  - Edge-to-cloud computing

Overview

- Course theme
- Academic integrity
- Course logistics
- Course schedule
- Five realities

COURSE THEME

Computing in every aspect of our lives

- Home Automation
- Bioinformatics
- Sharing Economy
- Ecommerce
- Online Education
- Precision Agriculture
- Computational Finance

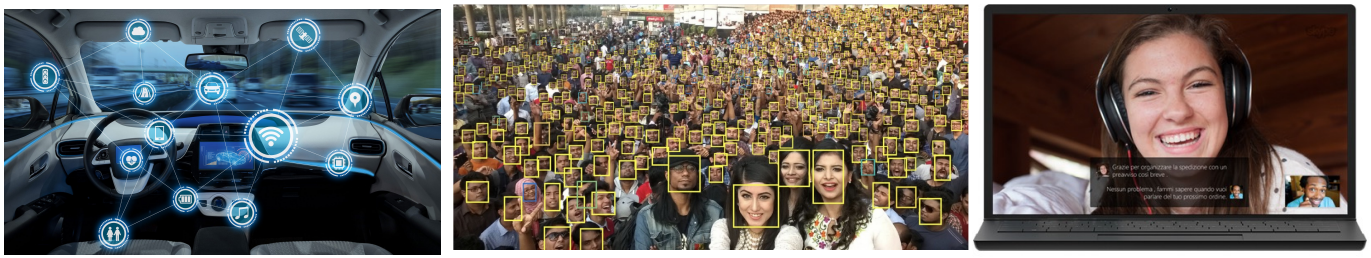
System: Foundation of All Innovations

All innovations in Computer Science

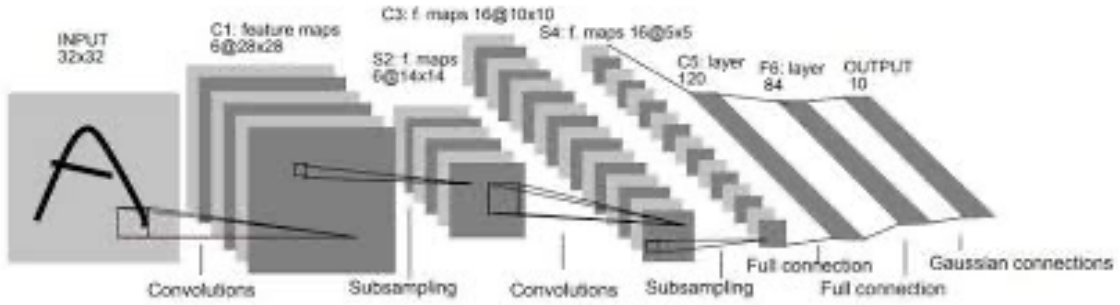
Computer Systems

Example: Huge Success of Deep Learning

Applications

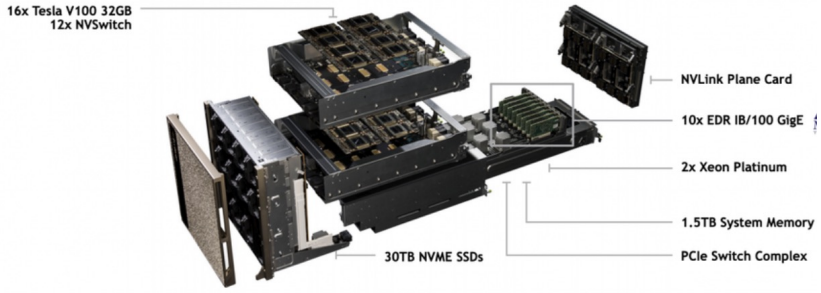


Algorithms



System

V100



DGX-2

This course is

- CS230 System Programming

- The course is an “introduction to computer systems” from the perspective of programmers

- If you expected System Programming, this is not a right course.

Course Perspective

- Most Systems Courses are Builder-Centric
  - Computer Architecture
    - Design pipelined processor
  - 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!**

ACADEMIC INTEGRITY

Cheating: Description

- **Pay Close Attention**
- What is cheating?
  - Sharing code: by copying, retyping, **looking at**, or supplying a file
  - Describing: verbal description of code from one person to another.
  - Coaching: helping your friend to write a lab, line by line
  - Searching the Web for solutions
  - Copying code from a previous course or online solution
    - You are only allowed to use code we supply, or from the CS:APP website
- What is NOT cheating?
  - Explaining how to use systems or tools
  - Helping others with high-level design issues
- Ignorance is not an excuse

Cheating: Consequences

- Penalty for cheating:
  - Minimum penalty: one letter grade downgrade (A0 → B0)
  - Possible removal from course with failing grade (F)
  - Your instructors' personal contempt

- Detection of cheating:
  - We have sophisticated tools for detecting code plagiarism
  - In the prior semester, >30 students were caught cheating

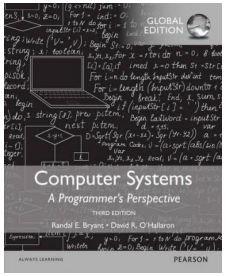
- Don't do it!
  - Start early
  - Ask the staff for help when you get stuck

COURSE LOGISTICS

Live video lecture via Zoom

- Join Zoom Meeting
  - https://kaist.zoom.us/j/88319898349?pwd=NVVya2RSMmNERWxxbnBISVQvYlV2UT09
- Meeting ID: 883 1989 8349
- Passcode: 083121

Textbooks



- Randal E. Bryant and David R. O'Hallaron,
  - *Computer Systems: A Programmer's Perspective*, **Third Edition** (CS:APP3e), Pearson, 2016
  - http://csapp.cs.cmu.edu
  - Highly recommend the original English version (you'll have to get used to textbooks in English as soon as possible to survive in CS)
  - This book really matters for the course! (for labs and exams)
  - **Reading the textbook is NOT optional, but required**
    - I will post required sections in KLMS.
    - There will be occasional quizzes about the reading assignments.
- Optional textbook: Brian Kernighan and Dennis Ritchie,
  - *The C Programming Language*, Second Edition, Prentice Hall, 1988
  - Still the best book about C, from the originators

Course Components

- Lectures
  - Higher level concepts
- Labs (7 assignments)
  - **The heart of the course**
  - About 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 web page: KLMS
  - Complete schedule of lectures, exams, and assignments
  - Copies of lectures, assignments, exams, solutions
  - **Update your email address (Important announcements can be made via emails and KLMS postings)**
- Q&A board: Piazza
  - Clarifications to assignments
  - Signup link: piazza.com/kaist.ac.kr/fall2021/cs230
  - Access code: cs230casys
  - **Students are also encouraged to answer questions**

Getting Help

- Staff mailing list: **cs230b_ta@casys.kaist.ac.kr**
  - Use this for all communication with the teaching staff
  - Always CC staff mailing list during email exchanges

- Office hours or TA sessions by TAs
  - The schedule will be announced on KLMS

Policies: Labs And Exams

- Work groups
  - You must work alone on all lab assignments
- Handins
  - Labs due at 11:59pm on Tues or Thurs
  - Electronic handins (no exceptions!)
- Exams
  - Midterm + Final (traditional exams)
  - **At least 2/3 of exam questions are related to the lab assignments**
- Appealing grades
  - In **writing** to Prof. Jongse Park within 7 days of completion of grading

Timeliness

- Lateness penalties
  - Get penalized **30%**
  - No handins later than **1 day after due date**
- Catastrophic events
  - Major illness, death in family, …
  - Formulate a plan (with your academic advisor) to get back on track
- Advice
  - Once you start running late, it's really hard to catch up

Policies: Grading

- Exams (54%):
  - midterm (27%), final (27%)
- Labs (40%): weighted according to effort
  - Labs + quiz
  - There will be occasional quizzes
- Attendance (6%)
  - We will use KLMS for checking
  - Attendance of the first week (8/31 and 9/1) will not be checked as the enrollment changes during the period
  - If you miss 1/3 of class meetings (9 class sessions), the final grade will be automatically F.

COURSE SCHEDULE

Labs

- *Tentative* lab schedule

- Lab 1: Linked list (Tuesday, 9/14)
- Lab 2: Data lab (Thursday, 9/30)
- Lab 3: Bomb lab (Tuesday, 10/12)
- Lab 4: Attack lab (Thursday, 10/28)
- Lab 5: Tsh lab (Thursday, 11/11)
- Lab 6: Malloc lab (Thursday, 11/25)
- Lab 7: Proxy lab (Thursday, 12/7)

Lab 1: Linked List

- The first lab for getting used to C and programming environments
- Due date: 9/14
- Implement simple functions for manipulating doubly-linked list data
  - Insert, delete, search
- Need to understand the basics of C, including pointers

Programs and Data

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

- Assignments
  - L2 (datalab): Manipulating bits
  - L3 (bomblab): Defusing a binary bomb
  - L4 (attacklab): The basics of code injection attacks

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.

Concurrency

- Topics
  - Concurrency, concurrent server design, threads
  - I/O multiplexing with select
  - Includes aspects of networking, OS, and architecture

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

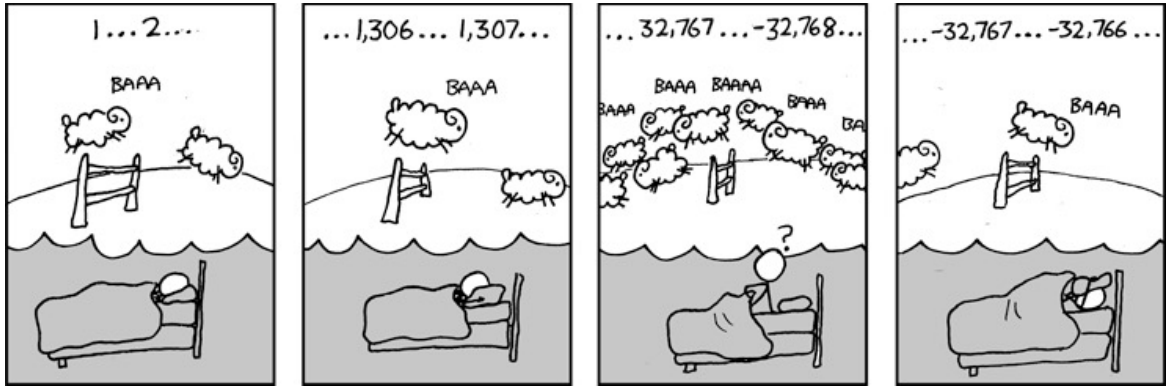
FIVE REALITIES

Course Theme: 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 CS230**
  - Become more effective programmers
    - Able to 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, Storage Systems, 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 → ??



- **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: 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
```

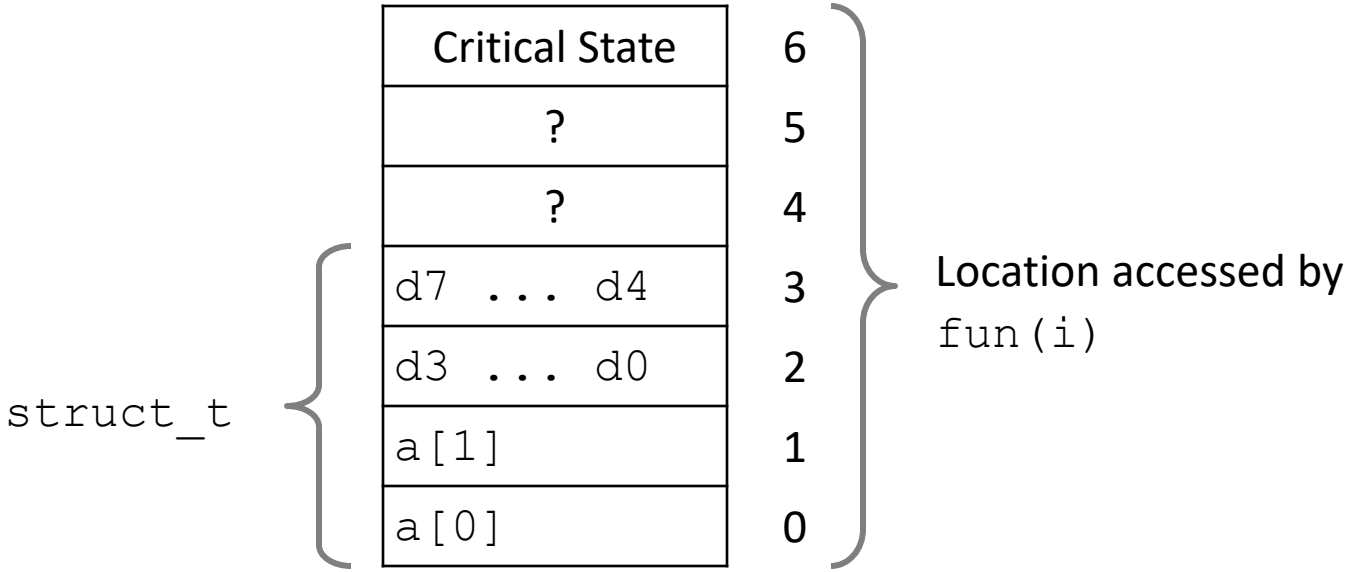
- Result is system specific

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)

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

Welcome and Enjoy!