Course ~~Over~~Review

15-213: Introduction to Computer Systems
27th Lecture, August 2, 2019

Instructor:

Sol Boucher

The course that gives CMU its “ZIP”!

Overview

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

Course Theme:

Abstraction Is Good But Don't Forget Reality

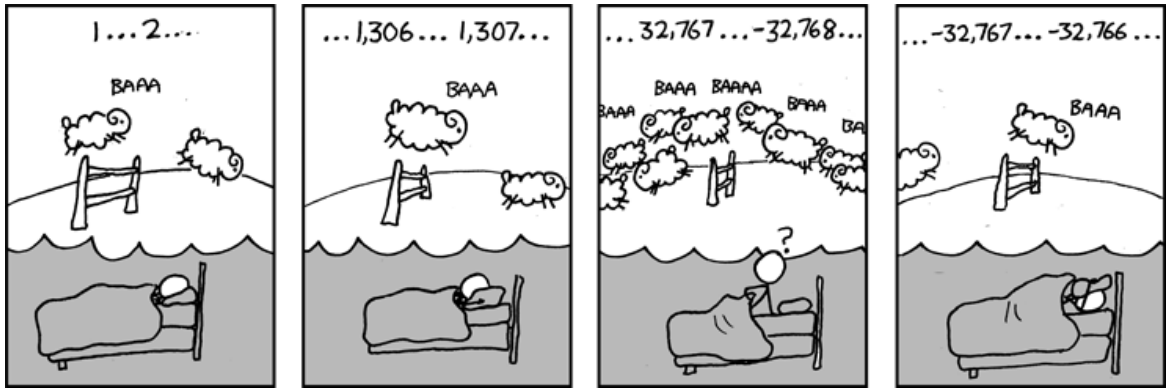
- **Most CS and CE courses emphasize abstraction**
  - Abstract data types
  - Asymptotic analysis
- **These abstractions have limits**
  - Especially in the presence of bugs
  - Need to understand details of underlying implementations
- **Useful outcomes from taking 213**
  - Become more effective programmers
    - 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.

Great Reality #1:

Ints are not Integers; Floats are not Reals

Example 1: Is $x^2 \geq 0$?

- Floats: Yes!



- Ints:
  - 40000 * 40000 --> 1600000000
  - 50000 * 50000 --> ?

Example 2: Is (x + y) + z = x + (y + z)?

- Unsigned & Signed Ints: Yes!
- Floats:
  - (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 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 Python, ML, Go, Rust, …
  - 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

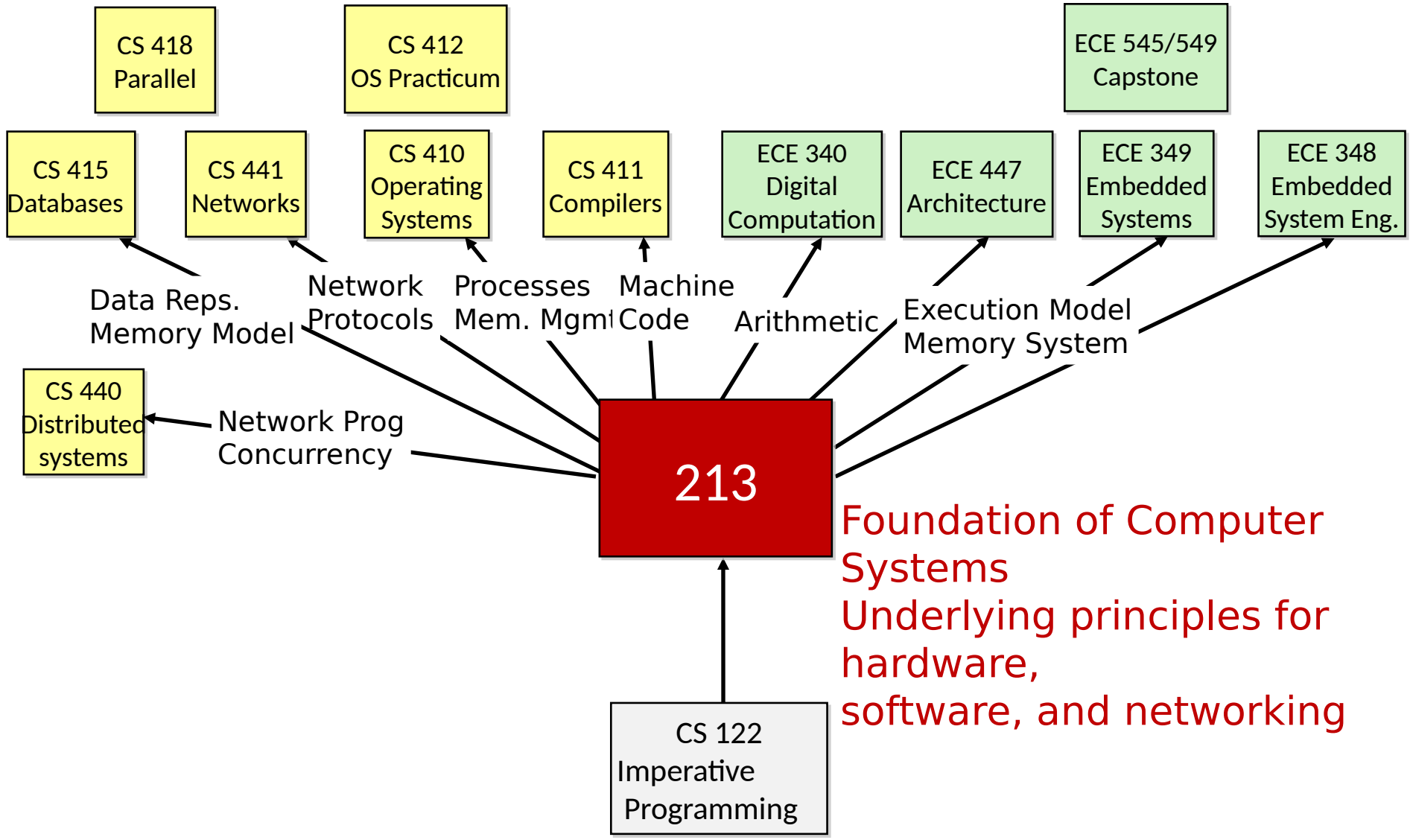
- **Hierarchical memory organization**
- **Performance depends on access patterns**
  - Including how step through multi-dimensional array

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

Role within CS/ECE Curriculum



Programs and Data

Topics

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

Assignments

- L1 (devicelab): Manipulating bits, characters, and strings
- L2 (bomblab): Defusing a binary bomb
- L3 (attacklab): The basics of code injection attacks

The Memory Hierarchy

Topics

- Memory technology, memory hierarchy, caches, 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.

Memory Allocation

Topics

- Dynamic storage allocation, virtual memory, address translation
- Includes aspects of architecture and OS

Assignments

- L5 (malloclab): Writing your own malloc package
  - Get a real feel for systems-level programming

Note: different topic/lab order vs. past terms!

Exceptional Control Flow

Topics

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

Assignments

- L6 (tshlab): Writing your own Unix shell.
  - A first introduction to concurrency

Networking and Concurrency

Topics

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

Assignments

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

Cheating: Consequences

Penalty for cheating:

- Best case: -100% for assignment
  - You would be better off to turn in nothing
- Worst case: Removal from course with failing grade
  - This is the default
- Permanent mark on your record
- Loss of respect by you, the instructors and your colleagues
- If you do cheat – come clean asap!

Detection of cheating:

- We have sophisticated tools for detecting code plagiarism
- In Fall 2015, 20 students were caught cheating and failed the course.
  - Some were **expelled** from the University
- In January 2016, 11 students were penalized for cheating violations that occurred as far back as Spring 2014.

Don't do it!

- Manage your time carefully
- Ask the staff for help when you get stuck

FCEs

Semester: Summer 2016

Course: 15213

Section: A

Course Title: INTR CMPUTER SYSTEMS

Instructor(s): BRIAN RAILING

In the case of multiple instructors, you will be asked to evaluate each instructor separately.

Instructor: BRIAN RAILING (PREVIEW MODE NOTE: The answers to these questions are viewable only by RAILING)

	1-3	4-6	7-9	10-12	13-15	16-18	19-21	22-24	25+
1. On average, how many hours per week have you spent on this class, including attending classes, doing readings, reviewing notes, writing papers and any other course related work?	○	○	○	○	○	○	○	○	○

	Excellent (5)	Above Average (4)	Average (3)	Below Average (2)	Poor (1)
2. Does the faculty member display an interest in students' learning?	○	○	○	○	○
3. Does the faculty member provide a clear explanation of the course requirements?	○	○	○	○	○
4. Does the faculty member provide a clear explanation of the learning objectives or goals of the course?	○	○	○	○	○

Final Exam

- Thursday, August 8th
  - You MUST sign up for this day, NOT the later ones!
  - Choice among 3 possible times: see Piazza post / registration system.

- The focus is on the second half of the course
  - IO
  - Signals
  - Processes
  - Virtual Memory
  - Malloc
  - Threads
  - Thread Synchronization
  - Other

IO

In the following code, a parent opens a file twice, then the child reads a character:

```
char c;
int fd1 = open("foo.txt", O_RDONLY);
int fd2 = open("foo.txt", O_RDONLY);
if (!fork()) read(fd1, &c, 1);
```

Clearly, in the child, `fd1` now points to the second character of `foo.txt`. Which of the following is now true in the parent?

(a) `fd1` and `fd2` both point to the first character.

(b) `fd1` and `fd2` both point to the second character.

(c) `fd1` points to the first character while `fd2` points to the second character.

(d) `fd2` points to the first character while `fd1` points to the second character

Signals

```
static void sigint_handler(int sig)
{
    pid_t pid = fgpid(job_list);
    /* Blocking signals */
    sigset_t mask, prev_mask;
    sigfillset(&mask);
    sigprocmask(SIG_BLOCK, &mask, &prev_mask);
    if (pid!=0)
    {
        /* Sending a SIGINT signal for the process group.
         * Deleting the job. */
        int jid = pid2jid(pid);
        kill(-pid, SIGINT);
        deletejob(job_list, pid);
    }
    /* Unblocking the masked signals */
    sigprocmask(SIG_SETMASK,&prev_mask,NULL);
    return;
}
```

Name three bugs in this code

Processes

What strings are possible? Is “15213”?

Swap exactly two of the string literals to “fix” this.

```
int main(int argc, char **argv)
{
    if (fork() == 0) { printf(“3”); return 0; }
    else { printf(“5”); }
    if (fork() == 0) { printf(“2”); }
    printf(“1”);
    return 0;
}
```

Malloc

- For an implicit allocator, with 16-byte alignment, 8-byte headers / footers, and prologue / epilogue.

  malloc(3)

  malloc(11)

  malloc(40)

  free (40)

  malloc(10)

- Draw the state of the heap in 8 byte units, label as header / footer (size, alloc or free), payload:

- What is the utilization for this allocator?
- How much space would be saved by removing footers?

Threads

- What is the range of value(s) that main will print?
- A programmer proposes removing `i` from `thread` and just directly accessing count. Does the answer change?

```
static volatile int count = 0;
static void *thread(void *v)
{
    int i = count;
    i = i + 1;
    count = i;
}
```

```
int main(int argc, char **argv)
{
    pthread_t tid[2];
    for(int i = 0; i < 2; i++)
        pthread_create(&tid[i],
NULL, thread, NULL);
    for (int i = 0; i < 2; i++)
        pthread_join(tid[i],
NULL);
    printf("%d\n", count);
    return 0;
}
```

Virtual Memory

TLB			
Index	Tag	PPN	Valid
0	03	C3	1
	01	71	0
1	00	28	1
	01	35	1
2	02	68	1
	3A	F1	0
3	03	12	1
	02	30	1

- Virtual addresses are 20 bits wide
- Physical addresses are 18 bits wide
- Page size is 1024 bytes
- TLB is 2-way set associative with 16 total entries

- Label each bit of a virtual address (Virtual Page offset, Virtual page number, TLB index, TLB tag):

- Given virtual address 0x04AA4, what happens?

The (Final!) Week Ahead

- **Tuesday, 8/6: Final exam review (led by the Tas)**
- **Wednesday, 8/7: "Future of Computing" guest lectures!**
- **Thursday, 8/8: Final exam (in the 5th-floor clusters)**
  - Remember to sign up for a time slot before then!
- **Friday, 8/9: Proxylab due**
  - There are NO LATE DAYS.
  - There are NO LATE SUBMISSIONS, not even for a point penalty!
- **Tuesday, 8/13: Final grades due to registrar**

Wednesday: Future of Computing!

Afsoon Afzal

Elliot Lockerman

Angela Jiang

Christopher Canel

Jenna Wise