Course Overview

15-213: Introduction to Computer Systems
26th Lecture, August 2, 2016

Instructor:
Brian Railing

The course that gives CMU its “Zip”!
Overview

- Course theme
- Five realities
- How the course fits into the CS/ECE curriculum
- Academic integrity
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 $x^2 \geq 0$?
  - Float’s: Yes!
  - Int’s:
    - $40000 \times 40000$ --> 1600000000
    - $50000 \times 50000$ --> ?

■ 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 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
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
Textbooks

- Randal E. Bryant and David R. O’Hallaron,
  - http://csapp.cs.cmu.edu
  - This book really matters for the course!
    - How to solve labs
    - Practice problems typical of exam problems

- Brian Kernighan and Dennis Ritchie,
  - Still the best book about C, from the originators
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 (mallocclab): 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
  - Post intermediate results (anonymized) on Autolab scoreboard for glory!
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!
Role within CS/ECE Curriculum

Foundation of Computer Systems
Underlying principles for hardware, software, and networking

- CS 415 Databases
- CS 441 Networks
- CS 410 Operating Systems
- CS 411 Compilers
- ECE 340 Digital Computation
- ECE 447 Architecture
- ECE 349 Embedded Systems
- ECE 348 Embedded System Eng.
- ECE 545/549 Capstone

CS 418 Parallel
CS 412 OS Practicum
CS 412 OS Practicum

Data Reps. Memory Model
Network Protocols
Processes Mem. Mgmt
Machine Code
Arithmetic
Execution Model Memory System

Network Prog Concurrency
Concurrency

CS 440 Distributed systems

213

CS 122 Imperative Programming
Cheating: Consequences

**Penalty for cheating:**
- Removal from course with failing grade (no exceptions!)
- Permanent mark on your record
- Your instructors’ personal contempt
- If you do cheat – come clean asap!

**Detection of cheating:**
- We have sophisticated tools for detecting code plagiarism
- Last Fall, 20 students were caught cheating and failed the course.
- Some were *expelled* from the University

**Don’t do it!**
- Start early
- Ask the staff for help when you get stuck
# FCEs

**Semester:** Summer 2016  
**Course:** 15213  
**Section:** A  
**Course Title:** INTR COMPUTER 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)

<table>
<thead>
<tr>
<th>Question</th>
<th>1-3</th>
<th>4-6</th>
<th>7-9</th>
<th>10-12</th>
<th>13-15</th>
<th>16-18</th>
<th>19-21</th>
<th>22-24</th>
<th>25+</th>
</tr>
</thead>
<tbody>
<tr>
<td>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?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Does the faculty member display an interest in students’ learning?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Does the faculty member provide a clear explanation of the course requirements?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Does the faculty member provide a clear explanation of the learning objectives or goals of the course?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Excellent (5)**  
**Above Average (4)**  
**Average (3)**  
**Below Average (2)**  
**Poor (1)**
Final Exam

- **August 5th**
  - Pittsburgh 11am – Close
  - Silicon Valley
  - Qatar

- **The focus is on the second half of the course**
  - IO
  - Signals
  - Processes
  - Virtual Memory
  - Malloc
  - Threads
  - Thread Synchronization
  - Other
In the following code, a parent opens a file twice, then the child reads a character:

```c
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

```c
void sigint_handler(int sig)
{
    pid_t pid = fgpid(job_list); /* Masking 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”? 

```c
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;
}
```
Virtual Memory

- 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?

<table>
<thead>
<tr>
<th>TLB</th>
<th>Index</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>03</td>
<td>C3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>71</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>00</td>
<td>28</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>35</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>68</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3A</td>
<td>F1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>12</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>02</td>
<td>30</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
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, versus 54 bytes?
- 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?

```c
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]);
    printf("%d\n", count);
    return 0;
}

volatile int count = 0;

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

```c
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]);
    printf("%d\n", count);
    return 0;
}

volatile int count = 0;

void* thread(void* v)
{
    int i = count;
    i = i + 1;
    count = i;
}
```
Thread Synchronization

Make FIFO -> LIFO

```c
void sbuf_init(sbuf_t *sp, int n)
{
    sp->buf = Calloc(n, sizeof(int));
    sp->n = n;
    sp->front = sp->rear = 0;
    Sem_init(&sp->mutex, 0, 1);
    Sem_init(&sp->slots, 0, n);
    Sem_init(&sp->items, 0, 0);
}

void sbuf_insert(sbuf_t *sp, int item)
{
    P(&sp->slots);
    P(&sp->mutex);
    sp->buf[(++sp->rear)%sp->n] = item;
    V(&sp->mutex);
    V(&sp->items);
}

int sbuf_remove(sbuf_t *sp)
{
    int item;
    P(&sp->items);
    P(&sp->mutex);
    item = sp->buf[(++sp->front)%sp->n];
    V(&sp->mutex);
    V(&sp->slots);
    return item;
}
```