### Welcome to Café 201 (also known as CS 201 Computer Systems Programming)

#### Spring 2019 Course Introduction

Instructor: Dr. Karen L. Karavanic



### Hello, World

```
#include <stdio.h>
int main()
{
    printf("hello, world\n");
}
```







### **Course Theme:**

### **Abstraction Is Good But Don't Forget Reality**

#### Most CS and CE courses emphasize abstraction

- Abstract data types
- Processes, Files

#### These abstractions have limits

- Especially in the presence of bugs
- Need to understand details of underlying implementations

#### Useful outcomes

- 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, Architecture

### **Great Reality #1:**

### Ints are not Integers, Floats are not Reals

### **Example 1:** Is $x^2 \ge 0$ ?

Float's: Yes!



Int's:

- 40000 \* 40000 → 160000000
- 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) --> ??

### **Code Security Example**

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];
/* Copy at most maxlen bytes from kernel region to user buffer */
int copy_from_kernel(void *user_dest, int maxlen) {
    /* Byte count len is minimum of buffer size and maxlen */
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}</pre>
```

- Similar to code found in FreeBSD's implementation of getpeername
- There are legions of smart people trying to find vulnerabilities in programs

### **Typical Usage**

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];
/* Copy at most maxlen bytes from kernel region to user buffer */
int copy_from_kernel(void *user_dest, int maxlen) {
    /* Byte count len is minimum of buffer size and maxlen */
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}</pre>
```

```
#define MSIZE 528
```

```
void getstuff() {
    char mybuf[MSIZE];
    copy_from_kernel(mybuf, MSIZE);
    printf("%s\n", mybuf);
}
```

### **Malicious Usage**

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];
/* Copy at most maxlen bytes from kernel region to user buffer */
int copy_from_kernel(void *user_dest, int maxlen) {
    /* Byte count len is minimum of buffer size and maxlen */
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}</pre>
```

```
#define MSIZE 528
```

```
void getstuff() {
    char mybuf[MSIZE];
    copy_from_kernel(mybuf, -MSIZE);
    . . .
}
```

### **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!

### **Assembly Code Example**

#### Time Stamp Counter

- Special 64-bit register in Intel-compatible machines
- Incremented every clock cycle
- Read with rdtsc instruction

#### Application

Measure time (in clock cycles) required by procedure

```
double t;
start_counter();
P();
t = get_counter();
printf("P required %f clock cycles\n", t);
```

### **Code to Read Counter**

- Write small amount of assembly code using GCC's asm facility
- Inserts assembly code into machine code generated by compiler

```
static unsigned cyc_hi = 0;
static unsigned cyc_lo = 0;
/* Set *hi and *lo to the high and low order bits
of the cycle counter.
*/
void access_counter(unsigned *hi, unsigned *lo)
{
    asm("rdtsc; movl %%edx,%0; movl %%eax,%1"
        : "=r" (*hi), "=r" (*lo)
        :
        : "%edx", "%eax");
}
```

### **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**

```
double fun(int i)
{
    volatile double d[1] = {3.14};
    volatile long int a[2];
    a[i] = 1073741824; /* Possibly out of bounds */
    return d[0];
}
```

fun(0)	$\rightarrow$	3.14
fun(1)	$\rightarrow$	3.14
fun(2)	$\rightarrow$	3.1399998664856
fun(3)	$\rightarrow$	2.0000061035156
fun(4)	$\rightarrow$	3.14, then segmentation fault

#### Result is architecture specific

### **Memory Referencing Bug Example**

```
double fun(int i)
{
    volatile double d[1] = {3.14};
    volatile long int a[2];
    a[i] = 1073741824; /* Possibly out of bounds */
    return d[0];
}
```

fun(0)	$\rightarrow$	3.14		
fun(1)	$\rightarrow$	3.14		
fun(2)	$\rightarrow$	3.13999986648	356	
fun(3)	$\rightarrow$	2.00000061035	5156	
fun(4)	→	3.14, then se	egmen	tation fault
Explana	tion:	Saved State	4 `	
-		d7 d4	3	
		d3 d0	2	Location accessed by
		α[1]	1	

al 0 |

0

### **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 or ML
- Understand what possible interactions may occur
- Use or develop tools to detect referencing errors (e.g. Valgrind)

### **Memory System Performance Example**



### 21 times slower

Hierarchical memory organization

### Pentium 4)

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

### **The Memory Mountain**

Read throughput (MB/s)



Intel Core i7 2.67 GHz 32 KB L1 d-cache 256 KB L2 cache 8 MB L3 cache

# 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

### **Example Matrix Multiplication**

Matrix-Matrix Multiplication (MMM) on 2 x Core 2 Duo 3 GHz (double precision) Gflop/s



- Standard desktop computer, vendor compiler, using optimization flags
- Both implementations have exactly the same operations count (2n<sup>3</sup>)
- What is going on?

### **MMM Plot: Analysis**

#### Matrix-Matrix Multiplication (MMM) on 2 x Core 2 Duo 3 GHz



- Reason for 20x: Blocking or tiling, loop unrolling, array scalarization, instruction scheduling, search to find best choice
- **Effect:** fewer register spills, L1/L2 cache misses, and TLB misses

### **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
- Data movement is currently seen as our biggest obstacle to pushing the high end of computing even higher

#### 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

### Great Reality #6: Security Matters!!!

If you do not understand this level of programming, you are at risk to introduce security holes in code you write

### **Required Course Textbook**

#### Randal E. Bryant and David R. O'Hallaron,

- "Computer Systems: A Programmer's Perspective, Third Edition" (CS:APP3e), Prentice Hall, 2015
- Additional student materials at: <u>http://csapp.cs.cmu.edu</u>
- NOTE: BRAND NEW BOOK
- Key change from 2<sup>nd</sup> edition: focus on 64-bit architecture
  - "what's 64-bit architecture?" stay tuned!
- C Programming notes available online for free: https://www.eskimo.com/~scs/cclass/cclass.html

### **Getting Help**

- Class Web Page: <u>http://web.cecs.pdx.edu/~karavan/cs201</u>
- We will transition to D2L
- TA Office Hours: TBD
- Instructor office hour: TBD
- Appointments are always possible modulo my schedule:
  - Email me and list some different days/times you can meet
  - If you cannot make it please email and let me know if at all possible

### "Open Door" Policy

- I am in my office and the door is open == Welcome !
- I am in my office and the door is [latched] shut == Oops! Not Now.
  - I am in a meeting or on a deadline and request no interruptions other than life and death emergencies

### **Facilities**

#### Labs and homeworks will use the Computer Science Linux Lab

- Remote login: <u>ssh myloginname@linuxlab.cs.pdx.edu</u>
- CS tutors sit outside of the lab during posted hours
- Small library of relevant books maintained by tutors

#### Homeworks

- Homeworks will be tested and graded on the Linux Lab machines
- We do not have the resources to accommodate your individual personal machine setups – please test your work on the lab machines before submitting

### Timeliness

#### Grace days

- 2 "free passes" (48 hours each) for the homeworks
- Covers scheduling crunch, out-of-town trips, illnesses, minor setbacks
- Murphy Says: Save them until late in the term!

#### Lateness penalties

- Once free passes(s) used up, receive a score of 0
- TURN IN WHATEVER YOU HAVE !

### Free Lunch

Your homework score is computed using the BEST 7 of 8 homeworks

### Advice

- Once you start running late, it's really hard to catch up
- 8 Weeks goes by VERYVERY quickly

### Cheating



#### What is cheating?

- Sharing code: by copying, retyping, looking at, or supplying a file
- Coaching: helping your friend to write a lab, line by line
- Copying code from previous course or from elsewhere on WWW
  - Only allowed to use code we supply
- Looking at anyone else's exam or showing anyone yours, in the exam room
- Posting in any form or forum the homework or exam answers

### What is NOT cheating?

- Explaining how to use systems or tools or getting that explained
- Helping others with high-level design issues or getting that help
- Getting help from the tutors or Course Expert

### Murphy Says:

- Tends to happen when you're tired, behind, and worried so stay on track
- 2 min rule: after an explanation, 2 mins before hands are back on keyboard
- "Why are we spending all this time talking about cheating?"
  - Because it happens EVERY YEAR and if it happens to you, you will FAIL

### **Other Rules of the CS 201 Classroom**

#### Laptops: permitted

#### Electronic communications: *forbidden*

- No email, instant messaging, cell phone calls, etc
- You will be asked to leave
- Why? This is a Learning Environment
- OK- looking things up as we go, electronic textbook, etc.
- OK asking questions in the class

Note: Some students have been granted specific permission to record the lectures. Without permission it's a no-no.



### HW #1 Part 1

- The Full Homework will be available on Thursday (April 4)
- But it's a good idea to get started
- **1.** Readings:

B&O [textbook] chapter 1

**Steve Summit's C Programming Notes Chapter 1:** 

https://www.eskimo.com/~scs/cclass/notes/top.html

- 1) Hands on:
  - Online Tutorial (SKIP 105.7): <u>http://pages.cs.wisc.edu/~remzi/OSTEP/lab-tutorial.pdf</u>
    - Focus on your initial goals: to be able to write, compile, run C Programs on the PSU Linux Lab machines
  - The Key: Learn a command line editor: vim or emacs

# Welcome and Enjoy!