Course Overview

Computer Systems Organization (Fall 2014)
Section 001 (Honors) and Section 002 (Regular)

Professor Andrew Case
Teaching Assistants: Paige Connelly & TBD

Slides adapted from Jinyang Li, Mohamed Zahran, Randy Bryant and Dave O’Hallaron
Not that kind of organization
This class adds to your CV...

- C programming
- UNIX
- X86 assembly
- Low level debugging
- Reverse engineering

Not what the class is about either
What this class is about

• Those details that set hackers apart from novice programmers
  – How your program runs on the hardware
  – Why it fails
  – Why it is slow

• Modern computer systems are shrouded in layers of abstraction
AN x64 PROCESSOR IS SCREAMING ALONG AT BILLIONS OF CYCLES PER SECOND TO RUN THE XNU KERNEL, WHICH IS FRANTICALLY WORKING THROUGH ALL THE POSIX-SPECIFIED ABSTRACTION TO CREATE THE DARWIN SYSTEM UNDERLYING OS X, WHICH IN TURN IS STRAINING ITSELF TO RUN FIREFOX AND ITS GECKO RENDERER, WHICH CREATES A FLASH OBJECT WHICH RENDERS DOZENS OF VIDEO FRAMES EVERY SECOND BECAUSE I WANTED TO SEE A CAT JUMP INTO A BOX AND FALL OVER.

I AM A GOD.

Many layers of abstraction
Course Theme: Abstraction Is Good But Don’t Forget Reality

• Most CS classes emphasize abstraction
• This class peeks “under-the-hood” in those layers
• Goal:
  – Make you more effective programmers
    • Debug problems
    • Tune performance
  – Prepare you for later “systems” classes in CS
    • Compilers
    • Operating Systems
    • Networks
    • Computer Architecture
    • Distributed Systems
Reality #1:
Ints are not Integers, Floats are not Reals

• $32767 + 1 = 32768$?
• $x^2 \geq 0$?
• $(x + y) + z = x + (y + z)$?

Source: xkcd.com/571
Reality #2:
You’ve Got to Know Assembly

• Little programming in assembly
• Knowledge of assembly helps one understand machine-level execution
  – Debugging
  – Performance tuning
  – Writing system software (e.g. compilers, OS)
  – Reverse engineering software
  • Creating / fighting malware
    – x86 assembly is the language of choice!
Reality #3: Memory Matters

• Memory is not unbounded
  – It must be allocated and managed
• Memory referencing bugs especially wicked
• Memory performance is not uniform
  – Cache and virtual memory effects can greatly affect performance
Memory Referencing Errors

• C/C++ let programmers make memory errors
  – Out of bounds array references
  – Invalid pointer values
  – Double free, use after free

• Errors can lead to nasty bugs
  – Corrupt program objects
  – Effect of bug observed long after the corruption
  – Security vulnerabilities
double fun(int i)
{
    double d[1] = {3.14};
    int a[2];
    a[i] = 1073741824; /* Possibly out of bounds */
    return d[0];
}

fun(0) ➞ 3.14
fun(1) ➞ 3.14
fun(2) ➞ ?
fun(3) ➞ ?
fun(4) ➞ ?
• There are legions of smart people trying to find vulnerabilities in programs
/* 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;
}

#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;
}

#define MSIZE 528

void getstuff() {
    char mybuf[MSIZE];
    copy_from_kernel(mybuf, -MSIZE);
    ...
}
Reality #4: Asymptotic performance analysis (e.g. Big O) is not always sufficient

- Constant factors matter
- Even operation count might not predict performance
- Must understand system to optimize performance
  - How programs compiled and executed
  - How to measure performance and identify bottlenecks
  - How to improve performance without destroying code modularity and generality
Memory System Performance Example

- Performance depends on access patterns

```c
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];
}

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];
}
```

21 times slower (Pentium 4)
Example Matrix Multiplication

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

Gflop/s

- Standard desktop computer and compiler
- Both implementations have exactly the same operations count ($2n^3$)

Best code (K. Goto)

Triple loop

160x
Matrix-Matrix Multiplication (MMM) on 2 x Core 2 Duo 3 GHz

Gflop/s

Reason for 20x: Blocking or tiling, loop unrolling, array scalarization

Effect: fewer register spills, L1/L2 cache misses, and TLB misses
Course Perspective

• Most Systems Courses are Builder-Centric (building things)
  – Computer Architecture
    • Designing a pipelined processor
  – Operating Systems
    • Implement large portions of operating system
  – Compilers
    • Write compiler for simple language
  – Networking
    • Implement and simulate network protocols
Course Perspective (Cont.)

• This course is programmer-centric
  – Understanding of underlying systems (to make us more effective programmers)
  – Bring out the hidden hacker in everyone
  – Dissecting the frog
Textbooks

• Randal E. Bryant and David R. O’Hallaron,
  – [http://csapp.cs.cmu.edu](http://csapp.cs.cmu.edu)
  – Available at NYU bookstore

• Brian Kernighan and Dennis Ritchie,
  – On reserve at Courant library
Course Components

• Lectures
  – Higher level concepts
  – Mini-demos

• Homework assignments (2)
  – Familiarize you with C
  – 1-2 weeks each

• Programming labs (3)
  – The heart of the course
  – 2-3 weeks each
  – Provide in-depth understanding of some aspect of systems

• One midterm exam
• One final exam
Course Grading

• Homeworks (2x 5%): 10%
• Programming labs: 35%
  – Buffer Lab: 11%
  – Cache Lab: 11%
  – Malloc Lab: 13%
• Midterm exam: 15%
• Final exam: 40%
Course Syllabus

- Basic C
  - Homework 1 and 2
- Assembly: Representation of programs, data, and reverse engineering
  - Bomb Lab
- System hardware, memory hierarchy for optimizations
  - Cache Lab
- Virtual Memory: address translation, allocation,
  - Malloc Lab
- Interacting with OS: processes, exceptions, parallelization
Getting Help

• Class webpage: https://cs.nyu.edu/~acase/fa14/CS201
  – Lectures notes
  – Assignments
  – Technical documentation and resources

• Discussion webpage: http://piazza.com
  – Announcements
  – Discussion
Getting Help

• Staff contacts:
  – Professor:
    Andrew Case acase@cs.nyu.edu
    When emailing me include in the subject line: CS201
  office hours:
    Monday / Thursday 3:30-5pm; or by appointment
  – Teaching Assistants:
    • Paige Connelly and TBA
Policies: Assignments

• You must work alone on all assignments
  – Post all questions on discussion group
  – You are encouraged to answer others’ questions, but refrain from explicitly giving away solutions.

• Hand-ins
  – Assignments due at 11:55pm on the due date
  – Everybody has 5 grace days for the entire semester
  – Zero score if a lab is handed in >=5 days late
TA Tutoring

– Lab tutorial sessions will be held for each lab
  • generally be demo based/recitation style
  • Schedule: TBA
  • Location: TBA

– One on one tutoring available as well
  • Schedule: TBA
  • Location: TBA
Facilities

• Assignment Lab environments:
  – Use official class VM (virtual machine) image
    • Software to run VMs:
      – VirtualBox (free) for Windows/Mac/Linux
    • VM used for lab:
      – Download VM image from course web page

• Physical CIMS/ITS Labs
  – CIMS Lab machines – open 24/7, UNIX based
    • Contact me for an account if needed
  – ITS Lab machines – Washington Place or 3rd Avenue both have VirtualBox installed
Cheating

• What is cheating?
  – Sharing code: by copying, looking at others’ files
  – Coaching: helping your friend to write a lab
  – Copying code from a fellow student, from a previous course, or from anywhere else including the Internet
    • You can only use code we supply

• Penalty for cheating:
  – Removal from course with failing grade
  – Permanent mark on your record

• CLU – Code Likeness Utility
  – Department tool sed for plagiarism detection
  – Uses heuristics to compare both comments and code
Time Management

– Labs present significant programming challenges
  • require a significant number of focused working hours
– Failure to complete assignments is usually due to:
  • starting too late
– Cheating is usually a result of:
  • starting too late
– Think ahead, ask questions, and plan your time accordingly
Feedback/Criticism

– I want as much feedback/criticisms as possible from you
  • as early as possible

– Let me know (anonymously if desired) if:
  • You feel you or others are missing key concepts
  • You are confused about any topic
  • You are unfamiliar with any terms
  • You have a suggestion on improving the course

– Keep in mind: If you have a question, undoubtedly others do too; and we will all benefit from your input. Do not be shy!