Computer Organization
CMSC 311

William Arbaugh
August 31, 2009

Topics:
- Staff, text, and policies
- Lecture topics and assignments
- Lab rationale
Textbooks

Randal E. Bryant and David R. O’Hallaron,
- csapp.cs.cmu.edu

Brian Kernighan and Dennis Ritchie,
Course Components

Lectures
- Higher level concepts

Labs
- The heart of the course
- 1 or 2 weeks
- Provide in-depth understanding of an aspect of systems
- Programming and measurement
Getting Help

Web
- http://elms.umd.edu
- Copies of assignments, exams, solutions
- Clarifications to assignments
- Student and News forums for general discussions

Personal help
- My door open means come on in (no appt necessary) but please try and make an appointment first (unless it is office hours)
- TAs: please mail first.
- Office hours posted in syllabus
Policies: Assignments

Work groups
- You must work alone on all labs

Handins
- Assignments due by midnight on specified due date.
- Electronic handins only for labs (details TBD)

Makeup exams and assignments
- OK in extenuating circumstances, but must make PRIOR arrangements with Prof. Arbaugh

Appealing grades
- Within 7 days of due date or exam date.
- Assignments: Talk to the lead person on the assignment
- Exams: Talk to Prof. Arbaugh.
Cheating

What is cheating?
- Sharing code: either by copying, retyping, looking at, or supplying a copy of a file. This includes pulling something from Google!

What is NOT cheating?
- Helping others use systems or tools.
- Helping others with high-level design issues.
- Helping others debug their code.

Penalty for cheating:
- Referral to academic counsel for academic dishonesty.
Policies: Grading

Exams (35%)
- Two in class exams (10% each)
- Final (15%)
- All exams are open book/open notes.

Labs (60%)
- 6 labs (10% each)

Class Participation (5%)
- "Pop" quizzes in class
- Participation in forum

Grading Characteristics
- Lab scores tend to be high
  - Serious handicap if you don’t hand a lab in
- Tests typically have a wider range of scores
Policies: Classroom Behavior

• Strive to arrive on time

• **Cell phone use** (to include texting and ringing) will not be tolerated.
  • You will be asked to leave the classroom
  • I will give a pop quiz to the rest of the class (you’ll get a zero)
Facilities

Assignments may use virtual machine technology or LinuxLab (Discussion)
Course web page is utilizing ELMS for online communication

You’ll find:
- Syllabus
- Interactive forums
  - News (I post only)
  - Discussion (anyone can post)
- Assignments
- Distribution of grades
Class Topics

- Programs and Data
- Reverse Engineering
- Performance
- The Memory Hierarchy
- Linking and Exceptional Flow Control
- Virtual Memory
- Virtualization
- I/O, Networking, and Concurrency
Programs and Data

Topics

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

Assignments

- Lab 1: Manipulating bits
- Lab 2: Defusing a binary bomb
Data Representation

What does 0x41 mean?
Safe Programming and Reverse Engineering

Topics
- Safe Coding techniques
- Disassembly
- Program tracing
- Buffer and Heap Overflows

Assignments
- Lab 2: Defusing a binary bomb
- Lab 3: Buffer Overflows
Performance

Topics

- High level processor models, code optimization (control and data), measuring time on a computer
- Includes aspects of architecture, compilers, and OS

Assignments

- Lab 4: Optimizing Code Performance
The Memory Hierarchy

Topics

- Memory technology, memory hierarchy, caches, disks, locality
- Includes aspects of architecture and OS.

Assignments

- Lab 4: Optimizing Code Performance
Linking and Exceptional Control Flow

Topics

- Object files, static and dynamic linking, libraries, loading
- Hardware exceptions, processes, process control, Unix signals, non-local jumps
- Includes aspects of compilers, OS, and architecture

Assignments

- Lab 5: Writing your own shell with job control
Virtual memory

Topics
- Virtual memory, address translation, dynamic storage allocation
- Includes aspects of architecture and OS

Assignments
- Lab 5: Writing your own malloc package
Virtualization

Topics

- Processor specific virtualization instructions
- IO Chipset additions to support virtualization
I/O, 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

- Lab 6: Writing your own web proxy program
Lab Rationale

Each lab should have a well-defined goal such as solving a puzzle or winning a contest.

- Defusing a binary bomb.
- Winning a performance contest.

Doing a lab should result in new skills and concepts

- Data Lab: computer arithmetic, digital logic.
- Bomb/buf Labs: assembly language, using a debugger, understanding the stack
- Perf Lab: profiling, measurement, performance debugging.
- Shell Lab: understanding Unix process control and signals
- Malloc Lab: understanding pointers and nasty memory bugs.
- Proxy Lab: Understanding how to write network programs
Course Theme

- Abstraction is good, but don’t forget reality!

Courses to date emphasize abstraction
- Abstract data types
- Asymptotic analysis

These abstractions have limits
- Especially in the presence of bugs
- Need to understand underlying implementations

Useful outcomes
- Become more effective programmers
  - Able to find and eliminate bugs efficiently
  - Able to tune program performance
- Prepare for later “systems” classes in CS & ECE
  - Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems
Great Reality #1

*Int’s are not Integers, Float’s are not Reals*

**Examples**

- Is $x^2 \geq 0$?
  - **Float’s:** Yes!
  - **Int’s:**
    - $40000 \times 40000 \rightarrow 1600000000$
    - $50000 \times 50000 \rightarrow ??$

- Is $(x + y) + z = x + (y + z)$?
  - **Unsigned & Signed Int’s:** Yes!
  - **Float’s:**
    - $(1e20 + -1e20) + 3.14 \rightarrow 3.14$
    - $1e20 + (-1e20 + 3.14) \rightarrow ??$
Computer Arithmetic

Does not generate random values

- Arithmetic operations have important mathematical properties

Cannot assume “usual” 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 program in assembly

- Compilers are much better & more patient than you are

Understanding assembly key to machine-level execution model

- Behavior of programs in presence of bugs
  - High-level language model breaks down

- Tuning program performance
  - Understanding sources of program inefficiency

- Implementing system software
  - Compiler has machine code as target
  - Operating systems must manage process state
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 required by procedure
  - In units of clock cycles

```c
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

```c
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");
}
```
void start_counter()
{
    access_counter(&cyc_hi, &cyc_lo);
}

double get_counter()
{
    unsigned ncyc_hi, ncyc_lo;
    unsigned hi, lo, borrow;
    access_counter(&ncyc_hi, &ncyc_lo);
    lo = ncyc_lo - cyc_lo;
    borrow = lo > ncyc_lo;
    hi = ncyc_hi - cyc_hi - borrow;
    return (double) hi * (1 << 30) * 4 + lo;
}
Great Reality #3

Memory Matters

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 a memory system can lead to major speed improvements
main ()
{
    long int a[2];
    double d = 3.14;
    a[2] = 1073741824;
    printf("d = %.15g\n", d);
    exit(0);
}

<table>
<thead>
<tr>
<th>Alpha</th>
<th>MIPS</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>-g</td>
<td>5.30498947741318e-315</td>
<td>3.1399998664856</td>
</tr>
<tr>
<td>-O</td>
<td>3.14</td>
<td>3.14</td>
</tr>
</tbody>
</table>

(Linux version gives correct result, but implementing as separate function gives segmentation fault.)
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, Lisp, Python, or ML
- Understand what possible interactions may occur
- Use or develop tools to detect referencing errors
Memory Performance Example

Implementations of Matrix Multiplication

- Multiple ways to nest loops

```c
/* ijk */
for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
        sum = 0.0;
        for (k=0; k<n; k++)
            sum += a[i][k] * b[k][j];
        c[i][j] = sum;
    }
}

/* jik */
for (j=0; j<n; j++) {
    for (i=0; i<n; i++) {
        sum = 0.0;
        for (k=0; k<n; k++)
            sum += a[i][k] * b[k][j];
        c[i][j] = sum;
    }
}
```
Matmult Performance (Alpha 21164)

Too big for L1 Cache

Too big for L2 Cache

Matmult Performance (Alpha 21164)

Too big for L1 Cache

Too big for L2 Cache
Great Reality #4

There’s more to performance than asymptotic complexity

Constant factors matter too!
- 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
Course Perspective

Most Systems Courses are Builder-Centric

- **Computer Architecture**
  - Design pipelined processor in Verilog

- **Operating Systems**
  - Implement large portions of operating system

- **Compilers**
  - Write compiler for simple language

- **Networking**
  - Implement and simulate network protocols

- **Security**
This Course is Programmer-Centric

- Purpose is to show how 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
- Not just a course for dedicated hackers
- Cover material in this course that you won’t see elsewhere, but is needed to be successful and save time in future system classes