Introduction to Computer Systems

Randal E. Bryant
August 26, 2008

Topics:
- Theme
- Five great realities of computer systems
- How this fits within CS curriculum
- Logistical issues

Course Theme

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

Most CS courses 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:
    - $(1e20 + -1e20) + 3.14 \rightarrow 3.14$
    - $1e20 + (-1e20 + 3.14) \rightarrow ??$

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;
}
```

- Similar to code found in FreeBSD’s implementation of getpeername.
- There are legions of smart people trying to find vulnerabilities in programs
  - Think of it as a very stringent testing environment
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;
}

#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);
    . . .
}

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
- 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 required by procedure
  - In units of clock cycles

```c
double t;
start_counter();
P();
t = get_counter();
printf("P required %.6f 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");
}
```

Great Reality #3

**Memory Matters:** Random Access Memory is an un-physical 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

```c
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) -> 3.14
fun(1) -> 3.14
fun(2) -> 3.1399998664856
fun(3) -> 2.00000061035156
fun(4) -> 3.14, then segmentation fault
```
**Referencing Bug Explanation**

- C does not implement bounds checking
- Out of range write can affect other parts of program state

<table>
<thead>
<tr>
<th>Saved State</th>
<th>Location accessed by <code>fun(i)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>d7 ... d4</td>
<td>4</td>
</tr>
<tr>
<td>d3 ... d0</td>
<td>3</td>
</tr>
<tr>
<td><code>a[1]</code></td>
<td>2</td>
</tr>
<tr>
<td><code>a[0]</code></td>
<td>1</td>
</tr>
</tbody>
</table>

**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 or ML
- Understand what possible interactions may occur
- Use or develop tools to detect referencing errors

**Memory System Performance Example**

```c
void copyji(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 copyij(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];
}
```

- `copyji`: 59,393,288 clock cycles
- `copyij`: 1,277,877,876 clock cycles

- 21.5 times slower!

**The Memory Mountain**

- Pentium III Xeon
- 550 MHz
- 16 KB on-chip L1 d-cache
- 16 KB on-chip L1 i-cache
- 512 KB off-chip unified
- L2 cache

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

(Measured on 2GHz Intel Pentium 4)
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

Code Performance Example

There are 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

/* Compute product of array elements */
double product(double d[], int n)
{
    double result = 1;
    int i;
    for (i = 0; i < n; i++)
        result = result * d[i];
    return result;
}

Loop Unrollings

Do two loop elements per iteration
- Reduces overhead
- Cycles per element:
  - u2: 7.0
  - u2r: 3.6

/* Unroll by 2. Assume n is even */
double product_u2(double d[], int n)
{
    double result = 1;
    int i;
    for (i = 0; i < n; i+=2)
        result = (result * d[i]) * d[i+1];
    return result;
}

/* Unroll by 2. Assume n is even */
double product_u2r(double d[], int n)
{
    double result = 1;
    int i;
    for (i = 0; i < n; i+=2)
        result = result * (d[i] * d[i+1]);
    return result;
}

Computation (length=12)

(((1 * d[0]) * d[1]) * d[2]) * d[3]) *
d[4]) * d[5]) * d[6]) *
d[7]) * d[8]) * d[9]) *
d[10]) * d[11])

Performance
- N elements, D cycles/operation
- N*D cycles

result = (result * d[i]) * d[i+1];
u2r: Reassociated Computation

Performance
- N elements, D cycles/operation
- \((N/2+1)^*D\) cycles

result = result * (d[i] * d[i+1]);

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 Curriculum

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

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
Course Perspective (Cont.)

Our 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
  - We bring out the hidden hacker in everyone
- Cover material in this course that you won’t see elsewhere

Teaching staff

Instructors

- Prof. Randal E. Bryant
- Prof. Greg Ganger

TA’s

- Taiyang Chen
- Tessa Eng
- Elie Krevat
- Bryant Lee
- Christopher Lu
- Swapnil Patil
- Vijay Prakash
- Jiri Simsa

Course Admin

- Cindy Chemsak (NSH 4303)

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,


Course Components

Lectures

- Higher level concepts

Recitations

- Applied concepts, important tools and skills for labs, clarification of lectures, exam coverage

Labs

- The heart of the course
- 2 or 3 weeks
- Provide in-depth understanding of an aspect of systems
- Programming and measurement

Exams

- Test your understanding of concepts & mathematical principles
  - Critical component of grade
Getting Help

Class Web Page
- http://www.cs.cmu.edu/~213
- Copies of lectures, assignments, exams, solutions
- Clarifications to assignments

Message Board
- http://autolab.cs.cmu.edu
- Clarifications to assignments, general discussion
- The only board your instructors will be monitoring (No blackboard or Andrew)

Getting Help

Staff mailing list
- 15-213-staff@cs.cmu.edu
- “The autolab server is down!”
- “Who should I talk to about ...”
- “This code {...}, which I don’t want to post to the bboard, causes my computer to melt into slag.”

Teaching assistants
- I don't get “associativity”...
- Office hours, e-mail, by appointment
  - Please send mail to 15-213-staff, not a randomly-selected TA

Professors
- Office hour or appt.
- “Should I drop the class?” “A TA said ... but ...”

Policies: Assignments

Work groups
- You must work alone on all but final lab

Handins
- Assignments due at 11:59pm on Tues or Thurs evening
- Electronic handins using Autolab (no exceptions!).

Conflict exams, other irreducible conflicts
- OK, but must make PRIOR arrangements with Prof. Ganger.

Appealing grades
- Within 7 days of completion of grading.
  - Following procedure described in syllabus
- Labs: Talk to the lead person on the assignment
- Exams: Talk to Prof. Ganger.

Timeliness

Grace Days
- 4 for the course
- Covers scheduling crunch, out-of-town trips, illnesses, minor setbacks
- Save them until late in the term!

Lateness Penalties
- Once grace days used up, get penalized 15%/day
- Typically shut off all handins 2—3 days after due date

Catastrophic Events
- Major illness, death in family, ...
- Work with your academic advisor to formulate plan for getting back on track

Advice
- Once you start running late, it's really hard to catch up
Cheating

What is cheating?
- Sharing code: either by copying, retyping, looking at, or supplying a copy of a file.
- Coaching: helping your friend to write a lab, line by line.
- Copying code from previous course or elsewhere on WWW
  - Only allowed to use code we supply, or from CS:APP website

What is NOT cheating?
- Explaining how to use systems or tools.
- Helping others with high-level design issues.

Penalty for cheating:
- Removal from course with failing grade.

Detection of cheating:
- We do check and our tools for doing this are much better than you think!

Policies: Grading

Exam Score E (out of 100):
- Two in class exams (25% each)
- Final (50%)
- All exams are open book / open notes.

Labs Score L (out of 100):
- 6 labs (10-25% each)

Composite Score:
\[ S = \frac{L + E + \min(L,E)}{3} \]
\[ \text{if } L < E: \quad \frac{2L + E}{3} \]
\[ \text{if } E < L: \quad \frac{L + 2E}{3} \]

Facilities

Labs will use the Intel Computer Systems Cluster (aka “the fish machines”)
- 15 Pentium Xeon servers donated by Intel for CS 213
- Dual 3.2 Ghz 64-bit (EM64T) Nocona Xeon processors
- 2 GB, 400 MHz DDR2 SDRAM memory
- Rack mounted in the 3rd floor Wean Hall machine room.
- Your accounts are ready nearing readiness.

Getting help with the cluster machines:
- See course Web page for login directions
- Please direct questions to your TA’s first
Programs and Data (7)

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 (bomlab): Defusing a binary bomb
- L3 (buflab): Hacking a buffer bomb

The Memory Hierarchy (3)

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

Assignments

Exceptional Control Flow (3)

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

Assignments
- L4 (tshlab): Writing your own shell with job control

Virtual Memory (4)

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

Assignments
- L5 (malloclab): Writing your own malloc package
  - Get a real feel for systems programming
Networking, and Concurrency (6)

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
- L6 (proxylab): Writing your own Web proxy

Performance (2)

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

Assignments

Lab Rationale
Each lab should have a well-defined goal such as solving a puzzle or winning a contest.
Doing a 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 Web page for glory!

Autolab Web Service
Labs are provided by the Autolab system
- Autograding handin system developed in 2003 by Dave O’Hallaron
- Apache Web server + Perl CGI programs
- Beta tested Fall 2003, very stable by now

With Autolab you can use your Web browser to:
- Review lab notes, clarifications
- Download the lab materials
- Stream autoresults to a class status Web page as you work.
- Handin your code for autograding by the Autolab server.
- View the complete history of your code handins, autoresult submissions, autograding reports, and instructor evaluations.
- View the class status page
Good Luck!