Course Overview

CS 485G-006: Systems Programming

Spring 2016

Instructor:

Neil Moore

Office Hours:

Hardymon 207, Wednesdays 2:00-4:00 PM

Web site:

http://www.cs.uky.edu/~neil/485/

Syllabus:

http://www.cs.uky.edu/~neil/485/syllabus.html

Adapted from slides by R. Bryant and D. O'Hallaron (http://csapp.cs.cmu.edu/3e/instructors.html), including revisions by J. Griffioen.

Course Goals

- **Develop an understanding of computing systems as a whole.**
  - Hardware, OS, libraries, windowing systems, network… working together.
  - How these components fit together to provide the environment in which an application executes.
  - You should come away with a complete, demystified view of the system.
- **Develop better programmers**
  - Identify causes of problems with your programs.
  - Class takes the perspective of a programmer, not an OS (etc.) designer.
  - Experience comes from developing/running programs on real machines.
  - Abstraction is wonderful, but must be grounded in reality.
- **Lay the foundation for upper-level classes**
  - Many upper-division classes assume the ability to write and debug large programs that interact with a variety of components of a system (e.g., compilers, operating systems, databases, networking, graphics, etc.)

Programs are just a small part of the picture

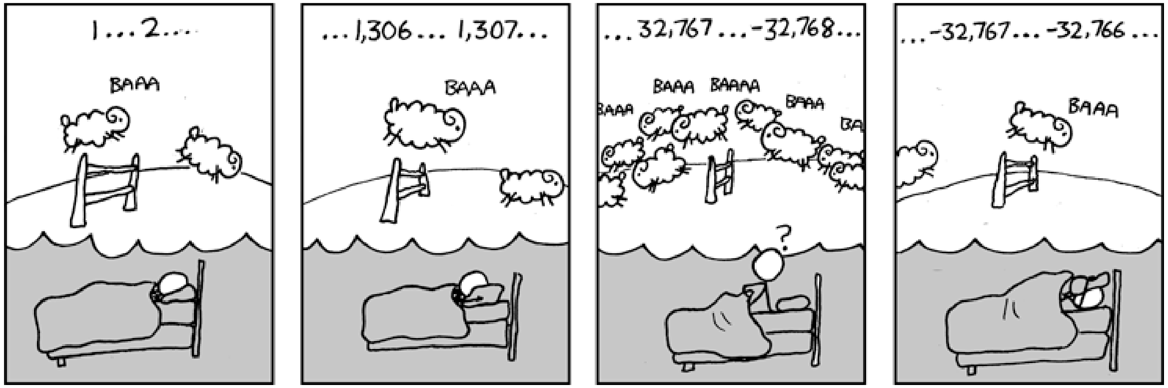
- **Even simple programs are part of a larger system**
  - Rely on several other system components to “run”.
  - One source of bugs: incorrect assumptions about those components.
    - Or not even having thought about them at all!
- **Need an understanding of the system in which you code will run:**
  - To debug your program.
  - To write efficient code.
  - To write secure code.
- How a computer system does something is too often “magic” to programmers. It should not be: computers are not magic!
- **What happens when you run a program?**

Abstraction Is Good, But Don’t Forget Reality

- **Most CS and CE courses emphasize abstraction**
  - Matches how humans think
  - Hides complexity so you can think about more at once
  - E.g. abstract data types, asymptotic analysis
- **But abstractions are only a model of reality**
  - Especially in the presence of bugs
  - Hardware has limitations not reflected in the model.
  - Hiding implementation details makes it harder to know how things interact with other components of the system.
  - Hiding complexity can lead to inefficiency.
- **What can go wrong?**

Pure math vs computer math: Ints are not Integers, Floats are not Reals

- **Example 1: Is $x^2 \geq 0$?**
  - Floats: Yes!



  - Ints:
    - 40000 * 40000 → 1600000000
    - 50000 * 50000 → ??
- **Example 2: Is (x + y) + z = x + (y + z)?**
  - Unsigned & Signed Ints: Yes!
  - Floats:
    - (1e20 + -1e20) + 3.14 --> 3.14
    - 1e20 + (-1e20 + 3.14) --> ??

Source: xkcd.com/571

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**

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);
    . . .
}
```

High-Level Languages Map to Assembly

- **Chances are, you'll never write programs in assembly**
  - Compilers are much better and 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!

Memory Referencing Errors

- **C and C++ do not provide any memory protection**
  - Out of bounds array references
  - Invalid pointer values
  - Abuses of malloc/free (or new/delete)
  - C++11 can help avoid some problems, but not all.
- **Can lead to nasty bugs**
  - Whether or not the 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 Referencing Bug Example

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

```
fun(0)   →   3.14
fun(1)   →   3.14
fun(4)   →   3.1399998664856
fun(5)   →   2.00000061035156
fun(8)   →   3.14, then segmentation fault
```

- **Result depends heavily on architecture and even compiler flags**

Memory Referencing Bug Example

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

Explanation:

Saved State	8
…	6-7
d[0] (part 2)	5
d[0] (part 1)	4
…	2-3
a[1]	1
a[0]	0

Location accessed by `fun(i)`

Memory System Performance Example

```
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];
}
```

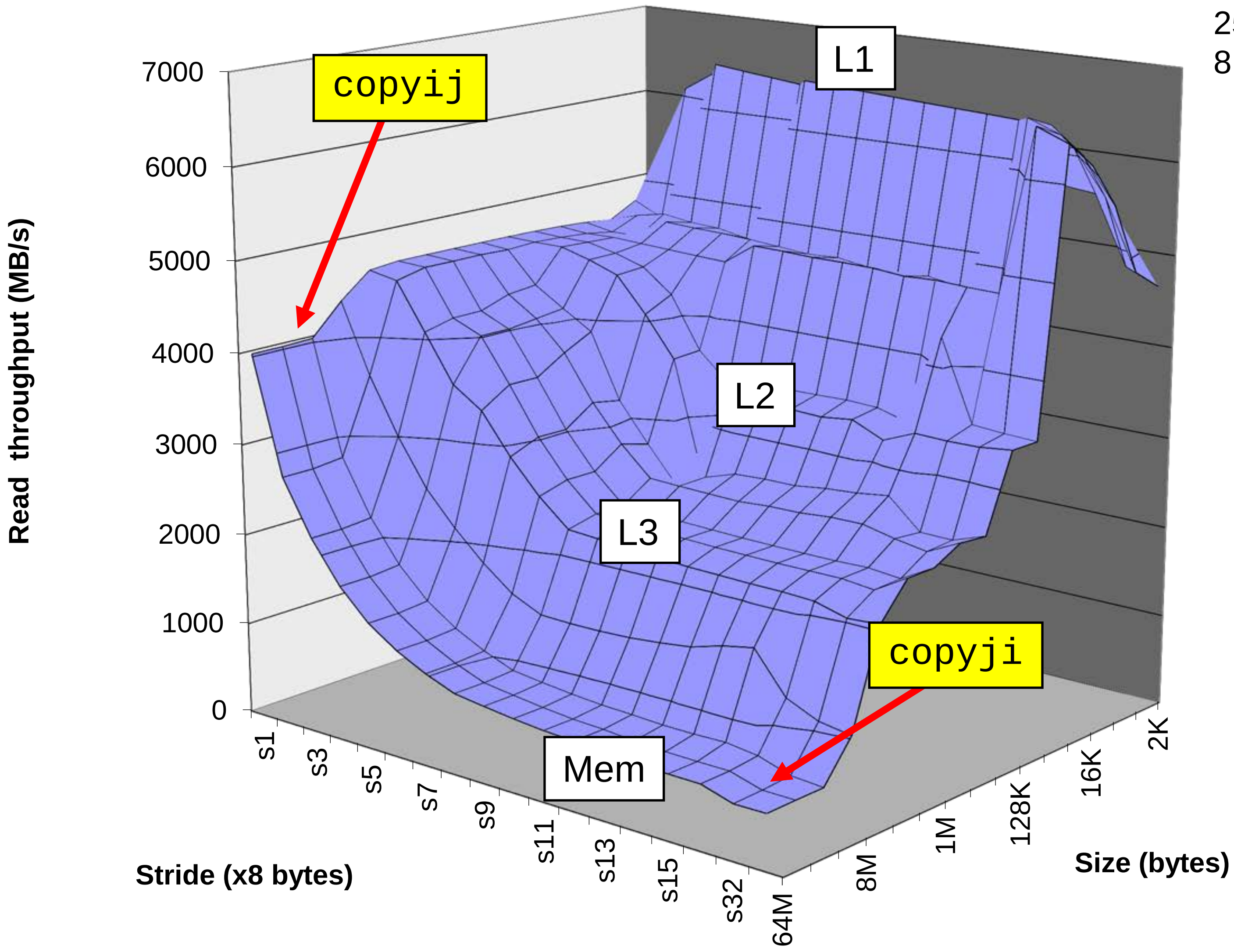
```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

~26 times slower on the class VMs!

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

The Memory Mountain

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



Policies etc.

- **Assignments/Grading**
  - Programming assignments: 50% (45% for grad students)
    - Approximately every 3 weeks.
  - In-class labs: 5%
    - Generally on Fridays
    - Bring your laptop on Friday! (required)
  - Midterm exam: 20%
    - Friday, 4 March, in class
  - Final exam: 25%
    - Monday, 2 May, 10:30am
  - Grad students: research paper: 5%
    - Decide on a topic with me.
    - More details around midterm.
  - Standard grading scale: 90%+ A, 80%+ B, 70%+ C, 60%+ D, else E

Policies etc.

- **Academic Integrity**
  - Submitted work must be your own.
  - Can discuss assignments with others, *not* share or show code.
  - If you get ideas, code snippets, etc. from somewhere, **cite it!**
    - Somewhere prominent in your documentation.
    - That includes fellow students, tutors, etc.

      "Jane Doe, personal communication, March 1 2016."
  - When in doubt, **ask me!**
- **Attendance**
  - Not taken most days, but still expected.
  - Required on lab and exam days (most Fridays)
  - See syllabus for make-up and absence policy.
- **Late policy (if no excused absence)**
  - -10% of the total per business day, no credit after 5 days.