CS 4400

Computer Systems

LECTURE 1

Administrative details

What to expect from CS 4400

Overview of computer systems

Course Information

http://www.eng.utah.edu/~cs4400/

Background

- The purpose of this course is to help you bridge the gap between high-level programming and the actual computer system.
  - “from a programmer's point of view”
- A computer system consists of hardware and software working together to run application programs.
  - processors, memory, OS, compilers, networks, ...
- Must have taken CS 3810. CS 3505 is recommended.
  - Familiarity with C++ is assumed. We will use C.

What to Expect from CS 4400

- Some of the topics to be covered
  - How information is represented—data and code
  - Optimizing code
  - The memory hierarchy
  - Communication among programs
- Heavy use of C, “Unix”, and the x86 architecture.
- C vs. Java
  - similar syntax and control statements
  - C: pointers, explicit dynamic memory allocation, formatted I/O, little support for abstractions (no classes)

Lab Work

- *Data representation*—implement logical and arithmetic functions by manipulating the bits that make up values.
- *Disassembling an object code file*—reverse engineer a program to determine what it does.
- *Implementing a buffer overrun attack*—modify the run-time behavior of a program to exploit a buffer overflow bug.
- *Performance optimization*—transform source code to make it run faster.
- *Processes and signals*—implement your own Unix shell.
- *Dynamic storage allocation*—implement your own version of C's `malloc` and `free` functions.

Your Responsibilities

- *Attend class.* When you must miss, check the web and ask classmates for material covered.
- *Complete the problem sets.* They will prepare you for exams, and some (randomly selected) are for credit.
- *Submit lab assignments on time.* See syllabus for the late policy. All programs must run on CADE `labl-`... machines.
- *Keep up with your grades.* See the course staff promptly if you have a dispute (within one week of getting grade).
- *Do not fall behind.* If you do, see the course staff ASAP.

Why Study Computer Systems?

- Do programmers need to know what is “under the hood”?
- How could doing so affect the programs you write?
- *Example*:

```
#include <stdio.h>

int main(void) {
  int x = 1073741824;

  printf("%i * 2 = %i\n", x, x*2);

  return 0;
}
```

```
unix> 1073741824 * 2 = -2147483648
```

- Computers don't follow the rules of math??!!

Life of a Program—Source

- *Example*:

```
#include <stdio.h>

int main(void) {
  printf("hello, world\n”);

  return 0;
}
```

hello.c

23 is ASCII code for '#' in hex (35 in decimal)

- How is this program “born”?

```
2369 6e63 6c75 6465 203c 7374 6469 6f2e
683e 0a0a 696e 7420 6d61 696e 2876 6f69
6429 207b 0a20 2070 7269 6e74 6628 2268
656c 6c6f 2c20 776f 726c 645c 6e22 293b
0a0a 2020 7265 7475 726e 2030 3b0a 7d0a
```

hello.c in hexl-mode (emacs)

Life of a Program—Translation

- Same bit patterns read by the computer during execution?
- The compiler translates a source file into an executable object file.

```
unix> gcc -o hello hello.c
```

  - preprocessing—more later
  - compilation—translates `hello.c` into `hello.s` (assembly lang)
  - assembly—translates `hello.s` into `hello.o` (machine lang)
  - linking—links `hello.o` to standard C library to get `hello`

```
...     call    printf
        addl    $16, %esp
        movl    $0, %eax
...
```

hello.s (x86 assembly language)

Should Compiler ≡ Black Box?

- Program performance
  - Is a `switch` always faster than an `if`-`else` `if` chain?
  - Should you choose recursion or iteration?
  - Are pointers more efficient than array indexes?
- Understanding linker errors
  - What does it mean that a symbol is undefined?
  - What is the difference in static and dynamic libraries?
- Avoiding security holes
  - How can buffer overflow corrupt the run-time stack?

Life of a Program—Execution

- The shell is a command-line interpreter.
  - first word is built-in shell command or name of executable file

```
unix> ./hello
hello, world
unix>
```

Unix shell

- The shell loads the executable file `hello` (copying the code and data from disk to main memory).
- The processor executes the instructions in `main`.
  - copies the data (i.e., the string) from main memory to cache
  - then copies the string to the display device (and to the screen)

Organization of Storage: A Hierarchy

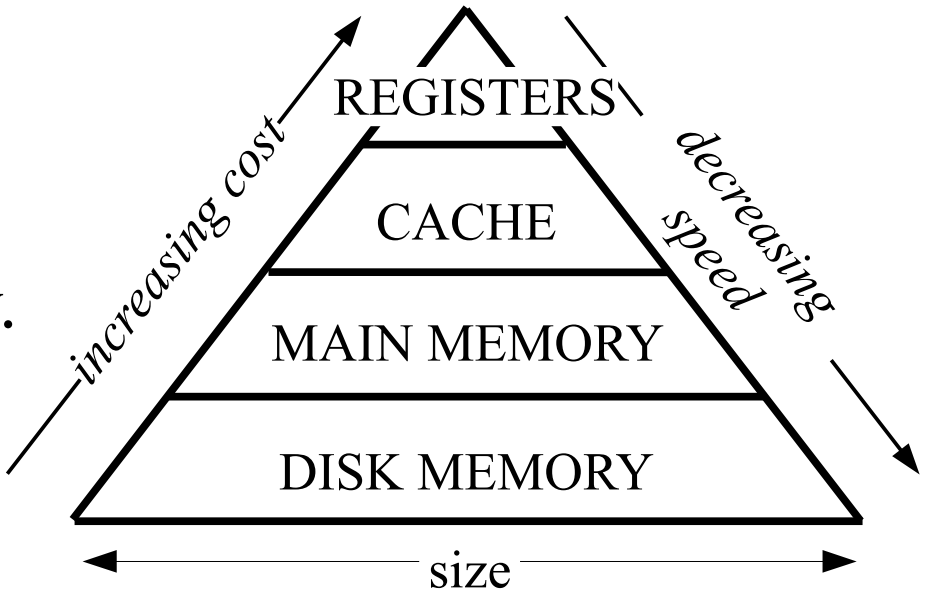
- A lot of time is spent moving information around.
  - Does all of this copying slow the "real work" of the program?
- Larger storage devices are slower than smaller ones.

  Faster storage devices are more expensive than slower.
- The processor can read data

  from a register nearly 100x

  faster than from main memory.
- But, the register file stores

  only a few hundred bytes.



Cache Memories

- To deal with this processor-memory gap, caches are used.
  - Small, fast storage devices serving as temporary staging areas for information the processor is likely to need in the near future.
  - How is it known what info will be needed in the future?
- Typically, two levels of cache exist (on and off chip).
- The programmer has no direct control on how information is stored. It's great that caches exist, but why do they matter to the programmer?
  - Caches can be exploited to improve run time by a factor of 10.

The Operating System

- Although `hello` requires the keyboard, display, disk, and main memory—none of them are accessed directly.
- The operating system is a layer of software between the application program (`hello`) and the hardware.
- Three fundamental abstractions:
  - *Processes*—the illusion that `hello` is the only program running
  - *Virtual memory*—each process appears to have exclusive use of main memory
  - *Files*—every I/O device viewed uniformly as a file

Useful Outcomes of CS 4400

- You will be a more effective programmer.
  - detecting and fixing bugs more efficiently
  - understanding and tuning program performance
- You will be more resourceful and self-sufficient.
  - better at figuring things out yourself
  - making effective use of available documentation
- You will be comfortable using the terminal and command-line (having already mastered IDEs in previous classes).
- You will have a firm foundation for specialized systems classes in CS and real-world SW development.