Introduction to Computer Systems

• Topics:
  – Theme
  – Five realities of computer systems
  – Compilation System
  – Computer Organization

Course Theme:

Abstraction 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
    • Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems
Great Reality #1

- *Ints are not Integers, Floats are not Reals*

- Examples
  - Is \( x^2 \geq 0? \)
    - Float: Yes!
    - Int:
      - 40000 * 40000 → 1600000000
      - 50000 * 50000 → ??
  - Is \((x + y) + z = x + (y + z)\)?
    - Unsigned & Signed Int: Yes!
    - Float:
      - (1e20 + -1e20) + 3.14 → 3.14
      - 1e20 + (-1e20 + 3.14) → ??

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

• Have to know assembly
• Probably never write in assembly
  – Compilers much better & more patient
• Understanding assembly key to machine-level execution model
  – Behavior of programs in presence of bugs
    • High-level language model breaks down
  – Tuning program performance
    • Understand optimization (not) done by compiler
    • Understand 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 rdtscl instruction
• Application
  – Measure time required by procedure
    • In 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

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
- 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
main ()
{
    long int a[2];
    double d = 3.14;
    a[2] = 1073741824; /* Out of bounds reference */
    printf("d = %.15g\n", d);
    exit(0);
}
```

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, 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;
    }
}
```

```c
/* 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;
    }
}
```

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

Compilation System

- **Hello program**

```c
#include <stdio.h>
int main()
{
    printf("hello, world\n");
}
```

- Compile
  - gcc -o hello hello.c
Quad Core

New Quad-Core Intel® Xeon® 5300 for 2006
Running Hello Program—hello command

Load hello object file from disk to main

User types "hello"

"hello" executable stored on disk

Write output to display

Storage Hierarchy

- Program execution involves a lot of info movements
  - Make copy operations efficient
- Huge gap between CPU speed and main memory
  - Introduce caches
Memory Hierarchy

Local secondary storage (local disks)

Remote secondary storage (distributed file systems, Web servers)

CPU registers hold words retrieved from cache memory.

L1 cache holds cache lines retrieved from L2 cache.

L2 cache holds cache lines retrieved from memory.

Main memory holds disk blocks retrieved from local disks.

Local disks hold files retrieved from disks on remote network servers.

Hardware Manager – Operating System

• Layered view of a computer system

Application programs

Operating system

Processor Main memory I/O devices

Hardware

• Abstraction provided by OS

Virtual memory

Files

Processes
Process

- OS abstraction of running a program
  - Provide multiple processes running concurrently
  - Switches from one process to another when a process needs to wait

Virtual Memory

- Physical memory is limited and is partitioned
- Give the illusion of using the unlimited amount of memory space