Fundamental of computer system design

(Slides adapted from CSAPP book)
Announcements

- Programming assignment # 1 is uploaded in the course web page
  - Due date is Monday Sept. 12
Summary of Previous class

- Computer architect design a computer
  - to meet functional requirements AND
  - to meet price, performance and availability goals
- Techniques to quantify performance and availability goals of computer systems
  - Reliability/availability
  - Performance speedup
  - Amdahl’s law
  - CPU performance
- Computer system classes and technology trend
  - Performance of a computer system = hardware performance + software performance

This class:
- What are the fundamental organization and architecture of a typical computer system?
- How having knowledge about the computer system helps to improve the performance of applications
Agenda

- Primers on computer system organization
  - Life cycle of a program execution
  - System architecture
  - OS
  - Computer Network
- Why do programmers need to know about the computer system?
- Recent challenges in computer system design (examples)
- Programming assignment #1
Primers on computer system organization

- Life cycle of a program execution
- System architecture
- OS
- Computer Network
**Information is Bits + Context**

Example: A simple hello program:

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

- **Source program representation:** Sequence of bits, (e.g., ASCII to represent texts: representing each character with a byte sized integer)

<table>
<thead>
<tr>
<th>#</th>
<th>i</th>
<th>n</th>
<th>c</th>
<th>l</th>
<th>u</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>105</td>
<td>110</td>
<td>99</td>
<td>108</td>
<td>117</td>
<td>100</td>
<td>101</td>
</tr>
</tbody>
</table>

- **Data representation in computers:** *Finite approximation of numbers*
  - Example:
    - integers in C: 2bytes → 50000*50000<0
Lifetime of a simple C program

Ex: insert studio.h into the program text

Ex: merging printf.o and hello.o
Why we should know how compilation system works?

- Optimizing program performance
  - Basic understanding of machine level code, and the compilation help to write efficient programs
- Understanding link-time error
- Avoiding security holes
  - Buffer overflow vulnerabilities
Hardware organization of a system

CPU
- Register file
- ALU
- Bus interface

Main memory

I/O bridge
- System bus
- Memory bus

Main memory

I/O bus
- Expansion slots for other devices such as network adapters

USB controller
- Mouse
- Keyboard

Graphics adapter
- Display

Disk controller
- Disk

Disk
- hello executable stored on disk

PC
Reading the hello command from keyboard

User types "hello"
Loading the executable file to memory

CPU
- Register file
- ALU
- Bus interface

System bus
Memory bus

I/O bridge

Main memory
"hello, world\n" hello code

I/O bus

Expansion slots for other devices such as network adapters

USB controller
- Mouse
- Keyboard

Graphics adapter
- Display

Disk controller
- Disk

hello executable stored on disk
Writing the output string from memory to the display

```
PC
Register file
ALU
Bus interface
System bus
Memory bus
I/O bridge
Main memory
"hello,world\n"
```

```
CPU
"hello code"
```

```
Expansion slots for other devices such as network adapters
```

```
USB controller
Graphics adapter
Display
"hello,world\n"
```

```
Disk controller
Disk
"hello executable stored on disk"
```

```
"hello,world\n"
```

```
Mouse
Keyboard
```

```
"hello,world\n"
```

CEN591 Fall 2011
Storage hierarchy

- **L0**: CPU registers hold words retrieved from cache memory.
  - L1 cache holds cache lines retrieved from L2 cache.
  - L2 cache holds cache lines retrieved from L3 cache.
  - L3 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.

- **L1**: Larger, slower, and cheaper (per byte) storage devices.
- **L2**: Smaller, faster, and costlier (per byte) storage devices.
Operating System (OS)

- A layer of software interpreted between the application program and the hardware

Layered view of a system

<table>
<thead>
<tr>
<th>Application programs</th>
<th>Operating system</th>
<th>Processor</th>
<th>Main memory</th>
<th>I/O devices</th>
</tr>
</thead>
</table>

- OS major tasks
  - To protect the hardware from misuse
  - To provide applications with mechanisms
  - To manipulate the low level hardware

- OS abstractions
  - Process, virtual memory, files

Abstractions provided by OS

Virtual memory

Files
Process

- OS abstraction for a running program

- OS provides the illusion that the running program is the only one running in the system by the context switching mechanism among concurrent processes.

- Hello program example
  - concurrent processes include hello and shell
  - The shell carries out our command by invoking a system call that passes the control to the OS

![Diagram of process context switching and system calls](image)
OS operations for multiprogramming

- Operation of OS is based on what is called *multi-programming* and *time-sharing*

- Time-sharing
  - Method of running multiple programs on a single CPU apparently concurrently
  - Use of *time slice*

- A process that needs I/O will have to be put “on hold”, i.e. *wait* and be *blocked*, until the I/O is complete
  - In the mean time, another process is executed

- What process to run is a decision of scheduling

- OS relies on hardware timers to time its operation
Threads

- A process can consist of multiple execution units, called **threads**, each running in the context of process, sharing **codes** and **global data**

- Providing concurrency for a process
  - Ex: network services

- Multithreads are more efficient than multiprocesses
Virtual memory
- OS’s abstraction to provide the illusion that each process has exclusive use of the main memory

- Need for hardware translation of every address generated by the processor

Kernel virtual memory
- User stack (created at runtime)
- Memory mapped region for shared libraries
- Run-time heap (created by malloc)
- Read/write data
- Read-only code and data

Process virtual address space

Memory invisible to user code

printf function

Loaded from the hello executable file

Variable size

Variable size

>`0x08048000 (32)

>`0x00400000 (64)
Files

- Sequence of bytes
- Linux models every I/O device as a file to provide a uniform view of all varied I/O devices
Network adapter can be seen as an I/O
Ex.: Running a program remotely (Telnet)

1. User types "hello" at the keyboard
5. Client prints "hello, world\n" string on display

2. Client sends "hello" string to telnet server

3. Server sends "hello" string to the shell, which runs the hello program, and passes the output to the telnet server

4. Telnet server sends "hello, world\n" string to client
And Much More

- Concurrency and parallelism
  - Multicores
  - Hyperthreading
  - Instruction level parallelism

- Abstraction at different layers
What’s Next?

- Next Class: data representation in computers (CSAPP Ch 2)

- Plan for next few lectures
  - Program representation in computers (CSAPP Ch 3),
Programming assignment #1: datalab
Introduction

- How to access the handouts?
  - Download the instruction and handouts from the course Web page
    - Datalab.pdf (instruction)
    - Datalab-handout.tar or datalab-handout.zip (c files)
      - bits.c (the main c file that you need to modify)
      - Others: utilities

- Doing the homework as a team
  - Form groups (at most three people)
What is the homework about?

Table 1: Bit-Level Manipulation Functions.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Rating</th>
<th>Max Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>bitAnd(x,y)</td>
<td>x &amp; y using only</td>
<td>and ”</td>
<td>1</td>
</tr>
<tr>
<td>getByte(x, n)</td>
<td>Get byte n from x.</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>logicalShift(x, n)</td>
<td>Shift right logical.</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>bitCount(x)</td>
<td>Count the number of 1’s in x.</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>bang(x)</td>
<td>Compute !n without using ! operator.</td>
<td>4</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 2: Arithmetic Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Rating</th>
<th>Max Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmin()</td>
<td>Most negative two’s complement integer</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>fitsBits(x, n)</td>
<td>Does x fit in n bits?</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>divpwr2(x, n)</td>
<td>Compute x/2^n</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>negate(x)</td>
<td>-x without negation</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>isPositive(x)</td>
<td>x &gt; 0?</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>isLessOrEqual(x, y)</td>
<td>x &lt;= y?</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>ilog2(x)</td>
<td>Compute (\log_2(x))</td>
<td>4</td>
<td>90</td>
</tr>
</tbody>
</table>

Table 3: floating-point operations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Rating</th>
<th>Max Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>float_neg(uf)</td>
<td>Compute -f</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>float_i2f(x)</td>
<td>Compute (float) x</td>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>float_twice(uf)</td>
<td>Compute 2*f</td>
<td>4</td>
<td>30</td>
</tr>
</tbody>
</table>

Grading=
Functional correctness (41) +
Performance (30) +
Style (5) = 76

Full credit=68
Bonus=18
How to develop the functions?

- For full description of the functions and grading scheme:
  - Read datalab.pdf
  - Read the comments of the file bits.c

- You need to modify the file bits.c and develop the requested functions
  - You are free to develop the requested functions under every environment, and then copy the code in the bits.c file
  - Other alternatives to use provided utilities
    - Working under Linux
    - Working under cygwin and

```c
/*
 * bitAnd - x&y using only ~ and |
 * Example: bitAnd(6, 5) = 4
 * Legal ops: ~ |
 * Max ops: 8
 * Rating: 1
 */

int bitAnd(int x, int y) {
    return 2;
}
```
What do you need to know to develop the functions?

- C bitwise operations
- Signed and unsigned representation of integer data (CSAPP Ch 2: sections 2.2, 2.3, next class topic)
- Float representation of data (CSAPP Ch 2: section 2.4, next class topic)
Installing Cygwin (1)

Cygwin is:
- a collection of tools which provide a Linux look and feel environment for Windows.
- a DLL (cygwin1.dll) which acts as a Linux API layer providing substantial Linux API functionality.

You can find everything about cygwin from here

Use auto-install file (setup.exe) to perform a [fresh install](http://www.cygwin.com/) or to [update](http://www.cygwin.com/) an existing installation.
Installing Cygwin (2)

- When using Cygwin Setup for the first time, the default is to install a minimal subset of all available packages (gcc is not included).
- For building programs, you'll need gcc, make and probably other packages from the "Devel" category. Text editors can also be found under "Editors".
  - At the "Select Packages", scroll the screen to find the category "Devel", click on the word "default" so that it changes to "install". This tells Setup to install *everything*, from the category ‘Devel’
  - Do the same thing for the ‘Editors’ category
Cygwin command line

- You can run Linux commands in the cygwin command line
- Run ‘mount’ to figure out how you can access windows drive from the cygwin environment
  - Ex: C: on /cygdrive/c