Welcome to CS 449: Introduction to System Software

Instructor: Wonsun Ahn
What is a “System”? 

• Merriam-Webster dictionary: 
  – A group of related parts that work together

• Your computer hardware is a system
  – Comprised of distinct parts
    • Motherboard, CPU, Memory
    • Devices (network, hard drive, keyboard, mouse)
  – With well-defined interfaces
    • DDR, PCI Express, SATA, USB
    • ISA (Instruction Set Architecture)
  – Interfaces allow parts to be interchanged
Is Computer Software a System?

• Is software composed of distinct parts?
  • Not in the old days...
    – Applications were monolithic pieces of software
    – That ran directly on top of computer hardware
    – That included code to manage / control HW
    – Often written in machine (assembly) language

• Problems with monolithic software
  – Repetitive implementations of code to control HW
  – Software is tied to underlying HW (not portable)
  – Cannot have two pieces of SW running on same HW
Modern Software is a System

• Software is divided into application and system software
• Application Software
  – Software that performs some useful task
  – E.g. Web browser, word processor, video streamer ...
• System Software
  – Software that is provide services that enable apps to run
  – Operating System
    • Gate keeper to HW resources (CPUs, Memory, Devices ...)
    • Distributes HW resources among apps in a fair manner
    • Abstracts HW into a well-defined interface for apps
  – Compiler
    • Translates apps from programming language to HW machine language
    • Allows apps to be written in high level languages (e.g. Java, Python...)
What we will Learn

• Theory
  – How different parts of the software system interact with each other and the hardware system
  – From the perspective of the application
    • Its well-defined interface with the HW
    • Its well-defined interface with the OS
    • What role the compiler plays in translating the application

• Practice
  – (Unix) C programming language
  – System software programming
Where it Fits in

• What you should have taken before
  – CS 445: Data Structures
  – CS 447: Computer Organization and Assembly Language Programming

• What you can take after
  – CS 1550: Introduction to Operating Systems
  – CS 1622: Introduction to Compiler Design
Why Learn System Software?

• Learn how to build system software
  – System software provide services for many apps
    • Needs to be efficient
    • Needs to be reliable
  – Need skills beyond application programming
  – As computer scientists and engineers, you will be called upon to build system software

• Learn how to build efficient applications
  – By learning how the underlying system works
Why Learn C?

• **Java** (Also applies to Python, JavaScript, PHP...)
  – Designed for **application software**
  – Goal: Ease of programming
  – Program runs on top of Java Virtual Machine (JVM)
    • JVM: layer of system SW that emulates an abstract machine
  – In this abstract machine...
    • Objects float around in limitless space
    • Objects access each other by calling (well-defined) methods
    • Exceptions protect objects from illegal accesses
  – A real machine looks nothing like that
Why Learn C?

• C
  – Designed for **system software**
  – Goal: Direct control over HW
  – Programs run on top of bare machine
    • Just like an assembly language program
  – In a real machine...
    • Objects reside in memory HW (typically DRAM)
    • Memory is a sequence of bytes with physical limits
    • Objects accessed by read/write of memory locations
    • Illegal accesses result in data corruption (or crashes)
Learning C will be Painful

• But will make you a better programmer
The syllabus
The Hello World Program

#include <stdio.h>

int main()
{
    printf("Hello world");
    return 0;
}
## Data Types

<table>
<thead>
<tr>
<th>Integers</th>
<th>Reals</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
<td>float</td>
<td>char</td>
</tr>
<tr>
<td>int</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>long</td>
<td>long double</td>
<td></td>
</tr>
<tr>
<td>long long</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Integer types have signed or unsigned versions
  - Examples: “signed int”, “unsigned short”, ...
- Data types derived from these basic data types
  - Pointer, Array, Struct
  - Will be discussed later in detail
- Note: no classes like in Java
Pointer Data Type

• Data type that holds a memory address
• Memory address: A number representing a location in memory
  – Locations in memory are numbered from 0 onwards
  – Cf. Java has no concept of memory addresses in its abstract machine, hence no pointers
• Examples: “float *”, “int *”, “char *”
  – “float *”: memory address of a location holding a float
  – “float” is called the base type of the pointer
**Pointer Data Type**

<table>
<thead>
<tr>
<th>Address</th>
<th>Memory</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>5</td>
<td>y</td>
</tr>
<tr>
<td>0002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0003</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0005</td>
<td>3.14</td>
<td>x</td>
</tr>
<tr>
<td>0006</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Example for below code:**
  ```c
  float x = 3.14;
  float *y = (float*) 5;
  ```

- **In example...**
  - `x` resides in address 0005
  - Value of `x` is 3.14
  - Type of `x` is float
  - `y` resides in address 0001
  - Value of `y` is 5
  - Type of `y` is float *
  - `y` is said to point to `x`
Sizeof() Operator

• Returns the size of data type (in bytes)
  – Examples: “sizeof(int)”, “sizeof(x)”, ...
• Why would you want to know this information?
  – C manipulates memory directly through addresses
  – Layout of variables in memory becomes important in C
• Returned size may be different for each system
  – Why? Underlying machine is designed for a specific size
  – E.g. if CPU has 2-byte wide integer registers, it’s more efficient to translate int types to 2-byte values
• Not a function call
  – Compiler substitutes sizeof() to a number during translation
    (Taking into consideration design of target machine)
  – No cost associated with it – do not try to hardcode a number!
## Sizeof() for Various Systems

<table>
<thead>
<tr>
<th>Data Type</th>
<th>C Standard</th>
<th>32-bit</th>
<th>Windows 64-bit</th>
<th>Unix/Linux 64-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>8 bits</td>
<td>8 bits</td>
<td>8 bits</td>
<td>8 bits</td>
</tr>
<tr>
<td>short</td>
<td>at least 16 bits</td>
<td>16 bits</td>
<td>16 bits</td>
<td>16 bits</td>
</tr>
<tr>
<td>int</td>
<td>at least 16 bits</td>
<td>32 bits</td>
<td>32 bits</td>
<td>32 bits</td>
</tr>
<tr>
<td>long</td>
<td>at least 32 bits</td>
<td>32 bits</td>
<td>32 bits</td>
<td>64 bits</td>
</tr>
<tr>
<td>long long</td>
<td>at least 64 bits</td>
<td>64 bits</td>
<td>64 bits</td>
<td>64 bits</td>
</tr>
</tbody>
</table>

- `sizeof(char)`?
  - Always 1
- `sizeof(int)`?
  - Depends on the compiler, but at least 2 according to Standard C
- `sizeof(char*)`? (char pointer)
  - Again, depends on the compiler, but usually 4 (32 bits) on 32-bit machines, and 8 (64 bits) on 64-bit machines