W4118 Operating Systems I

Junfeng Yang

Bad News

- This is a **TOUGH** course
  - “Most difficult” rated by CS alumni

- **Unfamiliar** low-level systems programming
  - C and Assembly
  - No abstraction, close to hardware

- **Intense**
  - “Should be 6 units instead of 3” ...
  - Most of those struggling in CS lounge or CLIC lab late or possibly overnight were OS students

- And you have to climb up N floors for lecture!
  - Or wait 10 minutes for elevator ...
Good News

- Not interested in learning OS or low-level systems programming? Don’t have to take this course!
  - New MS Breadth requirement
  - Waive if you have taken a similar course

More Good News

- Heavy, but totally worth it
  - “Most useful after graduating” rated by alumni
- Works hard ➔ good grade
- We’ll do our best to help you
- Climbing up N floors is good exercise!
Why Study OS?

- **OS = arguably the most fundamental software**
  - We do almost everything with computers through OS

- **By studying OS, you will**
  - Gain a good understanding of OS
  - Gain a good understanding of the big picture
    - How do hardware, programming language, compiler, algorithms, OS work together?
  - Learn some portable tricks

Possibly
- Land a job at Facebook/Google/Microsoft/VMware/...
- Get started in systems research
- Apply OS ideas to your research area
- ...
What Will We Learn?

- **OS concepts**
  - **What does an OS do?**
    - Abstract hardware: processes, threads, files
    - Manage resources: CPU scheduling, memory management, file systems

- **OS implementation techniques**
  - How does an OS implement X in general?
  - How do two kernels, xv6 and Linux, implement X?
What Will We Learn? (cont.)

- Hands on OS programming experience with several kernel programming assignments
  - Best way: learning by doing
  - Low-level systems programming skills
  - Practical programming skills
    - How to understand large codebase
    - How to modify large codebase
    - How to debug large codebase
    - ...
  - Tools and systems
    - QEMU, gdb, Android, ...
My Background

- **Research area: systems**
  - Publish in systems conferences (e.g., OSDI, SOSP, NSDI)

- **Research-wise, practical kind of guy; believe only in stuff that works and is useful**

- **System reliability research for N years**
  - Systems research shifted from pure performance to reliability starting around 2000
  - I was fortunate to be at the cutting edge of this shift
  - Hacked Linux & Windows, found some of the worst bugs
  - Current focus: concurrency

- **Cool projects available for interested students**
Some of My Previous Results

- Built several effective bug-finding tools
  - One transferred to Microsoft SQL Azure

- Found 100+ **serious bugs**
  - Security holes: write arbitrary memory
  - Data loss errors: lose entire file system data
  - Errors in commercial data center systems: stuck w/o progress

- Serious enough that developers immediately worked on fixes
  - google “lkml junfeng”

- Appeared at various website (e.g., cacm.org, lwn.net)

- Won a few awards (OSDI best paper, NSF Career, AFOSR YIP, Sloan)
Why Two Kernels?

- The xv6 teaching operating system from MIT
  - Small, easy to understand
    - Comes with code and a commentary
    - Make discussions concrete
  - Very good for illustrating OS concepts and implementation techniques

- The Linux kernel
  - Code readily available, many books
  - Real, widespread, relevant
  - Very good for helping you learn practical skills
xv6 Overview

- Created by MIT
- Implementation of Unix 6th Edition on x86
- A subset of Unix system calls
  - fork, exec, read, write, pipe, ...
- Runs with multiple processors/multicore
- User-mode programs (can do some real stuff)
  - mkdir, rm, ...
- Bootable on real PC hardware
Understanding xv6

- Lectures + study code and commentary + programming exercises
- Resources:
  http://www.cs.columbia.edu/~junfeng/os/resources.html
  - gcc inline assembly
  - Intel programming manual
  - QEMU monitor commands
  - gdb commands
  - PC hardware programming
xv6 Files

- **Generic:** `asm.h` (segmentation), `mmu.h`, `x86.h` (inline assembly), `elf.h` (ELF format), `types.h`, `param.h` (kernel constants), `string.c`
- **Boot:** `bootasm.S`, `bootother.S`, `bootmain.c`, `main.c`
- **Process and virtual memory:** `proc.h`, `proc.c`, `vm.c`, `pipe.c`, `exec.c`, `kalloc.c`, `sysproc.c`, `swtch.S`, `initcode.S`
- **System call and interrupt:** `syscall.h`, `traps.h`, `trap.c`, `syscall.c`, `trapasm.S`, `vector.S`
- **Synchronization and multicore:** `spinlock.h`, `mp.h`, `spinlock.c`, `mp.c`
- **Disk and file system:** `defs.h`, `fs.h`, `stat.h`, `file.h`, `buf.h`, `fcntl.h`, `bio.c`, `fs.c`, `file.c`, `sysfile.c`
- **Device:** `kbd.h`, `kbd.c`, `timer.c`, `lapic.c`, `picirq.c`, `uart.c`, `console.c`, `ide.c`, `ioapic.c`
- **User-mode programs:** `user.h`, `sh.c`, `wc.c`, `kill.c`, `cat.c`, `grep.c`, `ln.c`, `ulib.c`, `echo.c`, `init.c`, `ls.c`, `printf.c`, `umalloc.c`, `mkdir.c`, `rm.c`, `usys.S`
- **Initialize a file system:** `mkfs.c`
- **Build:** `Makefile`, `kernel.ld`
- **Test:** `stressfs.c`, `forktest.c`, `zombie.c`, `usertests.c`
Linux Overview

- A modern, open-source OS, based on UNIX standards
  - 1991, 0.1 MLOC, single developer
    - Linus Torvalds wrote from scratch
    - Main design goal: UNIX compatibility
  - Now, 10 MLOC, developers worldwide
    - Unique source code management model

- Linux distributions: ubuntu, redhat, fedora, Gentoo, CentOS, Android ...
  - Kernel is Linux
  - Different set of user applications and package management systems
  - Run on cloud, server, desktop, mobile, ...

- Run on 1 billion (Android) devices by end of this year, and 2 billion by end of 2014 or 2015 [Eric Schmidt]
Understanding Linux

- Lectures + study code and book + programming assignments
- Resources: http://www.cs.columbia.edu/~junfeng/os/resources.html
Linux Licensing

- The GNU General Public License (GPL)

- Anyone creating their own derivative of Linux may not make the derived product proprietary; software released under GPL may not be redistributed as a binary-only product
Linux kernel structure

Applications

System Libraries (libc)

System Call Interface

I/O Related
- File Systems
- Networking
- Device Drivers

Process Related
- Scheduler
- Memory Management
- IPC

Architecture-Dependent Code

Modules

Hardware
Linux kernel structure (cont.)

- **Core + dynamically loaded modules**
  - E.g., device drivers, file systems, network protocols

- **Modules were originally developed to support the conditional inclusion of device drivers**
  - Early OS has to include code for all possible device or be recompiled to add support for a new device

- **Modules are now used extensively**
  - Standard way to add new functionalities to kernel
  - Reasonably well designed kernel-module interface
Linux kernel source

- Download: kernel.org
- Browse: lxr.linux.no (with cross reference)
  - Android: http://androidxref.com/
- Directory structure
  - include: public headers
  - kernel: core kernel components (e.g., scheduler)
  - arch: hardware-dependent code
  - fs: file systems
  - mm: memory management
  - ipc: inter-process communication
  - drivers: device drivers
  - usr: user-space code
  - lib: common libraries
Additional Course Info


- Next: tour of course website
Homework 1

- **Written part:** basic OS concepts

- **Programming part:** warmup, sanity test
  - Get you familiar with some basic tools
  - Set up xv6 and qemu
  - Learn xv6 boot loader, kernel, calling conventions
  - A little bit of low-level C coding
TA Sessions (Optional)

- First TA session
  - Wei Wang
  - Introduction to git, qemu, gdb, ssh
  - Friday 9/6, 3-4pm, CLIC lab
Other Action Items

- Highly recommended: apply for a CS account
- Find groupmates for Linux kernel programming assignments
- Buy Nexus 7