CIS-4020 Operating Systems

Peter C. Chapin
Vermont Technical College
Fall 2013
Goals

• **Learn operating system internals**
• **Gain experience with large programs**
  – Navigate a large code base written by others
  – Learn how to decipher someone else’s code
• **We will not be covering...**
  – Installing, configuring, or managing a system
  – Using operating systems
  – Application programming
Theory vs Practice

• 25% Theory
  – Discuss general topics

• 75% Practice
  – In lab we will write system software
    • Linux kernel programming
    • Modify and extend Phoenix
    • Programming the QNX microkernel system
  – Lots of programming
    • Mostly plain C
    • Some Assembly Language
Five Major Topics

• Five major resources that systems manage
  – Processor time
  – Memory
  – Disk space
  – Device management
  – Network bandwidth

• In this course we will discuss the first four
  – Traditional OS courses don’t usually talk about networking
Labs

• Virtual machine technology
  – Allows us to hack at the OS without disrupting the host system
  – Makes it easy (easier) to transport our experimental system to various machines

• Labs are somewhat long and involved
  – Most are multi-week
  – Some are open ended
  – All will require written reports (except the first one)
  – All will be fun and interesting!
LaTeX

• The LaTeX typesetting system is required for lab reports
  – Will spend some time covering how to use LaTeX
  – It’s a system worth knowing
    • Very high quality output
    • Plays well with software development methodologies
    • Used extensively in academic settings
    • A target output format of other systems (Doxygen)
Linux

• Complete source code available for study and modification

• The Linux distribution is about 6 MLOC
  – Finding one’s way around is the first challenge
  – We will use the cscope code browsing tool

• We will study kernel version 3.6.11
Phoenix

• Created by VTC Students
  – Nick Guertin, Curtis Aube, Wei Yao Lin
  – Senior project AY 2008/2009
  – Used in this course with permission
  – [https://github.com/pchapin/phoenix](https://github.com/pchapin/phoenix)

• Advantages
  – Microkernel design
  – Small and simple
  – Targets 16 bit “real mode” systems (8086)
QNX

- Commercial OS for Embedded Systems
  - Targets a variety of platforms
    - Including the desktop
  - Microkernel design
  - Flexible and well documented
  - Available for free for academic use
  - [http://www.qnx.com](http://www.qnx.com)
Monolithic Kernels

• Linux is monolithic
  – Entire kernel is one large program
  – Different parts of the kernel can call each other directly
  – Fast

• Traditional OS design is for monolithic kernels
Basic OS (Monolithic) Architecture

- Hardware Abstraction Layer
- Hardware Independent Kernel
- API Support Layer (POSIX Interface)
- User Mode:
  - Application Code
  - RTL
- Kernel Mode:
  - Device Driver
  - Device Driver
  - Device Driver
- Hardware
Microkernels

• Components of OS in separate processes
  – Components communicate via message passing
  – **More overhead.** Slower
  – **More robust.** If one component crashes the rest of the system continues to run
  – **More flexible**
    • Components can be mixed and matched
    • Components can be (trivially) distributed over a network
    • Components can be written in any programming language

• Most OS research uses microkernels
Basic OS (Microkernel) Architecture

User Mode:
- App
- TCP/IP
- File System
- NIC Driver
- Disk Driver

Kernel Mode:
- Microkernel (message passing, hardware abstraction)

Hardware
Windows?

• We won’t discuss Windows in this course
  – No source code available
  – No new architectural issues of significance
    • Windows is a microkernel internally but wraps all OS components into a monolithic lump for performance reasons

• Nothing wrong with Windows
  – It’s a modern system with a reasonable design
  – Not different enough from Linux to warrant special study
Android?

• We won’t discuss Android in this course
  – Modified Linux (although being merged)
  – Essentially all differences in the UI and libraries
  – From a kernel perspective...
    • ... not different enough from stock Linux to warrant separate handling.
Real Time Operating Systems

• If a computation does not complete in the specified time, then the system has failed.
• Many embedded systems have real time requirements
• Special OS support is needed
  – Arbitrary multi-tasking makes it impossible for a program to always meet deadlines
• Special programming methods needed
  – Programmer must take into account time issues
Distributed Operating Systems

• Spread the OS over multiple machines
  – Many open questions; systems not widely used
  – Interesting and may be important in the future
  – Clustering is a simple form of distributed OS

• Distributed OS...
  – Applications automatically distributed by the OS. *Applications unaware of their distributed nature.*

• Distributed applications...
  – Applications manage their distribution. *OS unaware of the application’s nature.*
Operating Systems Landscape

• Desktop/Server systems
  – Fairly stark. Unix and Windows dominate
  – Servers a little more interesting: clustering, scalability, advanced file systems

• Embedded systems
  – This is where the action is
  – Many, *many* different systems
  – Some very small (*Salvo* can run with zero RAM!)
  – Some are full scale systems adapted for embedded work
Class Organization

• Traditional Delivery
  – Face to face lectures and labs

• Course materials on my web site
  – [http://web.vtc.edu/users/pcc09070/cis-4020](http://web.vtc.edu/users/pcc09070/cis-4020)
  – Grading policy, late policy, etc
  – Handouts, assignments, useful links
  – Your first assignment is already posted

• Homework/Lab submissions on Moodle
Good Luck!

And don’t forget to have fun!