Running on the Bare Metal with GeekOS

David Hovemeyer, Jeffrey K. Hollingsworth,
and Bobby Bhattacharjee

University of Maryland, College Park

Outline

- Motivation
- Overview
- Projects
- Classroom Experience
- Conclusions

OS Kernel == Magic

- An operating system kernel is a program
- But, it makes the execution of other programs possible!
- This is a mind-expanding idea
  - Sort of like recursion: a snake eating its own tail
- Our belief: the operating system course should be faithful to this idea

How Are OS Projects Designed?

- Many approaches out there:
  - High level simulation [e.g., Dickinson SIGCSE 2000]
  - User mode process [e.g., Minix/Solaris]
    - Java?
  - CPU simulator + user level threads [e.g., Nachos]
  - Emulator [e.g., System/161]
  - Real Hardware [e.g., Minix, Topsy]

Our Approach

- Target commodity hardware platform (x86 PC)
- We provide minimal foundation, students build real, *complete* OS on top of it
  - Students add processes, VM, filesystem, IPC
- Two motivations:
  - Philosophical
  - Practical

Philosophical Motivations

- Targeting real hardware → 100% realism
  - Peel away all layers of abstraction
- Intellectual satisfaction
  - "So that's how it works!"
- Try it out on a real machine
  - Students can do this at the end of the course!

Practical Motivations

- x86 fairly easy to program
- Tool support is outstanding
  - GCC, binutils, gdb, nasm
  - Bochs emulator
- Lots of good documentation in books, on web

Outline

- Motivation
- Overview
- Projects
- Classroom Experience
- Conclusions

GeekOS Facts

- Project hosted at Sourceforge:
  - http://geekos.sourceforge.net
- Current version: 0.2.0
- Free software (MIT license)
- Includes manual with projects
  - We do *not* publically distribute project solutions

Overview of GeekOS

- Written in C and x86 assembly
- Size:
  - Minimal configuration: 7100 lines
  - Maximal configuration (all drivers, VFS, buffer cache, stubs for console, pipes, VM): 13300 lines
  - All projects completed: 17000 lines
- These figures include whitespace, comments, assertions, debug statements

GeekOS Features

- Threads, memory allocation, drivers for essential devices
- VFS layer, read-only filesystem (for loading programs), system call layer
- Minimal C library, small collection of user programs
- Students add everything else!

Outline

- Motivation
- Overview
- Projects
- Classroom Experience
- Conclusions

GeekOS and Bochs

- We develop and run GeekOS using the Bochs emulator:
  - http://bochs.sourceforge.net
- Advantages over running on actual hardware:
  - Runs as ordinary user mode process
  - Boots in seconds
  - Extensive diagnostics, debugging with gdb

Project 1—User Mode

- Students add user processes using segmentation for memory protection
  - Programs loaded from ELF executables
  - Each process allocated a fixed, contiguous block of memory
  - Segmentation is easier than paging

Project 2—Scheduling

- Original scheduler is static priority, round-robin
- Students implement
  - Alternative scheduler with dynamic priorities
  - Semaphores
- Students measure and evaluate both schedulers under workloads we provide
  - So they understand how the new scheduler addresses shortcomings of the original scheduler

Project 3—Virtual Memory

- Students replace the segmentation-based user mode with one based on paging
- When no free pages are available, a victim is selected using LRU and paged out
- Page fault handler:
  - Stack faults (add new page for accesses in red zone)
  - Page in previously evicted pages

Project 4—Filesystem

- Students implement a hierarchical read/write filesystem
- Students must handle locking for files and directories accessed concurrently

Project 5—Interprocess Communication

- Students extend the VFS to include
  - The console (keyboard and screen)
  - Pipes (anonymous, half-duplex, like Unix)
- ACLs are added to the filesystem
  - Each process gets a uid

Demo

Outline

- Motivation
- Overview
- Projects
- Classroom Experience
- Conclusions

Classroom Experience

- Classroom experience has been positive
  - Most students find GeekOS to be relatively easy to work with
- Using Bochs is a win
  - Good diagnostics when things go wrong
  - *Much* easier than dealing with actual hardware
- Lots of possibilities for alternative projects
  - Combat plagarism

Outline

- Motivation
- Overview
- Projects
- Classroom Experience
- Conclusions

Conclusions

- Targeting real hardware platform is a viable choice
  - Intellectually satisfying
  - A good emulator makes it practical
  - Instructors can easily choose how “low-level” they want students to get
  - Students gain experience with system-level programming

Future Work

- Fix bugs, improve documentation
- Make GeekOS smaller and simpler
- Port to a safe C dialect such as Cyclone
  - Statically demonstrate absence of memory errors!

Related Work

- Minix (real hardware, complete OS)
- Nachos (user process + CPU simulator)
- OS/161 (emulator, close to real HW)
- Topsy (microkernel, embedded MIPS target)
- Many others...

Questions?

Feature Comparison

OS	Target	Microkernel?	Lines
Minix	x86, others	Yes	32000
Nachos	MIPS (CPU sim)	No	9000
OS/161	MIPS (emulator)	No	15000
Topsy	MIPS (embedded)	Yes	13000
GeekOS	x86 PC	No	13000

- Of these, only Minix and GeekOS target commodity hardware

Motivation

- Why another educational OS?
- We wanted two properties:
  - Realism: target a real hardware platform
  - Simplicity: make it as simple as possible
- Give students a feel for “real” kernel hacking
  - Without overwhelming them with detail

How is GeekOS Different?

- GeekOS is different mainly in that
  - It targets commodity hardware
  - It tries to be “minimal”
- What we provide is merely a foundation
  - Students build all of the interesting parts of the kernel

Core Services

- GeekOS provides the minimum functionality needed to build higher level services:
  - Memory allocation: page and heap
  - Interrupt handling
  - Threads (with preemptive task switching)
  - Device drivers: screen, keyboard, floppy, IDE disk
  - VFS layer, minimal read-only filesystem
  - System call layer

GeekOS Userland

- Minimal libc:
  - Console I/O, file I/O, subprocess support, string routines
- Does not adhere to any standard API
- Small set of user programs
  - Shell, file and directory utils

Projects

- Students implement features of a modern kernel:
  - Multilevel feedback scheduler
  - User processes with paged virtual memory
  - Read/write hierarchical filesystem
  - Pipes
- When finished, resembles a simple version of Unix

x86 PC is a Good Platform

- Targeting the x86 PC has many practical benefits:
  - Excellent tool support: Linux and FreeBSD come with complete GeekOS-friendly toolchain installed by default
  - x86 is easy to program
  - Lots of good documentation for system-level programming