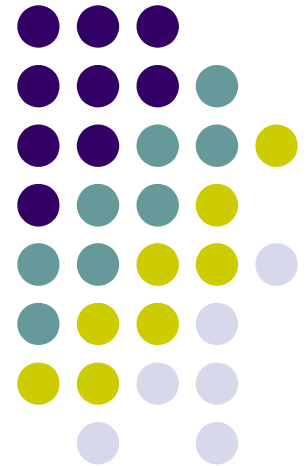
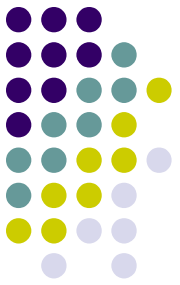


Bootstrapping

Dave Eckhardt

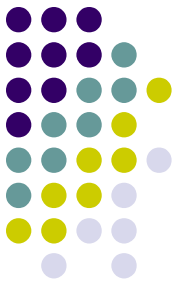
Steve Muckle





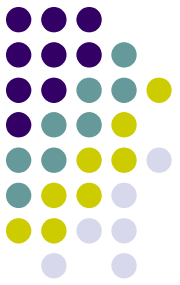
Synchronization

- ♦ Checkpoint 1
 - Monday! During this time period!
 - Meet in Wean 5207
 - **If your group number *ends* with**
 - 0-2 try to arrive 5 minutes early
 - 3-5 arrive at 10:42:30
 - 6-9 arrive at 10:59:27
 - **We will ask you to load and run a test program**
 - You must attend (unless you make arrangements by Saturday evening)
 - **Even if not everything is working!**



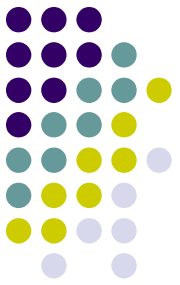
Synchronization

- ♦ Who uses...?
 - Wean 52xx Linux boxes? Full?
 - **West Wing Linux boxes? Full?**
 - Crash box?
 - Simics on cycle servers?
 - **Please limit yourself to 1 Simics on 1 machine**
- ♦ Partner reminder
 - If P2 was troubling, and P3 isn't improving, see us



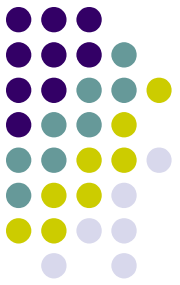
Motivation

- ♦ What happens when you turn on your PC?
- ♦ How do we get to `kernel_main()`?



Overview

- ◆ Requirements of Booting
- ◆ Ground zero
- ◆ BIOS
- ◆ Boot loader
- ◆ Our projects: Multiboot, OSKit
- ◆ BIOS extensions: PXE, APM
- ◆ Other universes: “big iron”, Open Firmware
- ◆ Further reading



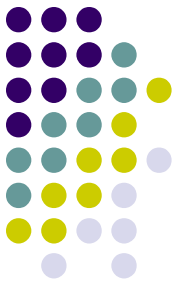
Requirements of Booting

- ♦ Initialize machine to a known state
- ♦ Make sure basic hardware works
- ♦ Inventory hardware
- ♦ Load a real operating system
- ♦ Run the real operating system



Ground Zero

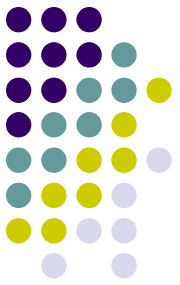
- ♦ You turn on the machine
- ♦ Execution begins in real mode at a specific memory address
 - Real mode - primeval x86 addressing mode
 - **Only 1 MB of memory is addressable**
 - First instruction fetch address is 0xFFFF0 (???)



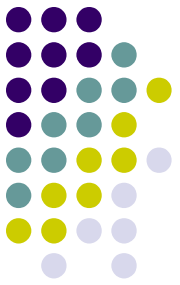
Ground Zero

- ♦ You turn on the machine
- ♦ Execution begins in real mode at a specific memory address
 - Real mode - primeval x86 addressing mode
 - **Only 1 MB of memory is addressable**
 - First instruction fetch address is 0xFFFF0 (???)
 - **“End of memory” (20-bit infinity), minus 15...**
 - **Contains a jump to the actual BIOS entry point**
 - Great, what’s a BIOS?

Basic Input/Output System (BIOS)

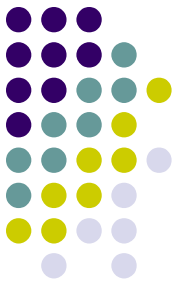


- ◆ Code stored in mostly-read-only memory
 - Flash (previously EEPROM, previously EPROM)
- ◆ Configures hardware details
 - RAM refresh rate or bus speed
 - Password protection
 - Boot-device order
- ◆ Loads OS, acts as mini-OS
- ◆ Provides some device drivers to real OS



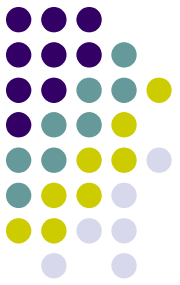
BIOS POST

- ♦ Power On Self Test (POST)
- ♦ Scan for critical resources
 - RAM
 - **Test it (only a little!)**
 - Graphics card – look for driver code at 0xC0000
 - Disk – look for driver code at 0xC8000
 - Keyboard
- ♦ Missing something?
 - Beep



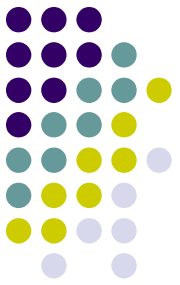
BIOS Boot-Device Search

- ◆ Consult saved settings for selected order
 - “A: C: G:” (maybe PXE)
- ◆ Load the first sector from a boot device
 - Could be a floppy, hard disk, CDRROM
 - Without a BIOS, we’d be in a bit of a jam
- ◆ If the last two bytes are AA55, we’re set
- ◆ Otherwise look somewhere else
 - If no luck, strike terror into user's heart:
 - **“No Operating System Present”**



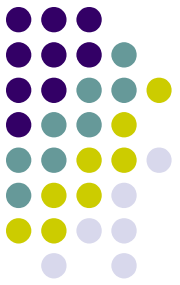
BIOS Boot-Sector Launch

- ♦ Boot sector is copied to 0x7C00
- ♦ Execution is transferred to 0x7C00
- ♦ Extra step for hard disk or CD-ROM
 - Boot sector (“MBR”) knows about partitions
 - **BIOS starts it running at 0x7C00, of course**
 - **Copies itself elsewhere in memory, jumps there**
 - **Loads “active” partition's boot sector at 0x7C00**
- ♦ Now we're executing the boot loader – the first “software” to execute on the PC



Boot Loader

- ♦ Some boot loaders designed to load one OS
- ♦ Others give you a choice of which to load
- ♦ Some are small and have a simple interface
 - “F1 FreeBSD F2 Windows”
- ♦ Some are large, contain GUI, shell prompt
- ♦ We use GRUB
 - <http://www.gnu.org/software/grub/>



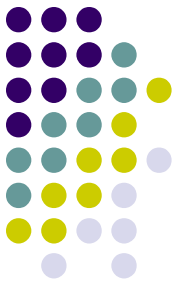
Boot Loader's Job

- ♦ Mission: load operating system
- ♦ From where?
 - “/boot/kernel.gz” is easier said than done!
 - May need to understand a file system
 - **Directories, inodes, symbolic links!**
 - May need to understand *multiple* file systems
 - **Single disk may contain more than one**
 - **Layout defined by “partition label”**
 - ...and “extended partition label”



Boot Loader's Job

- ♦ Mission: load operating system
- ♦ From where?
 - “/boot/kernel.gz” is easier said than done
 - May need to understand a file system
 - **Directories, inodes, symbolic links!**
 - May need to understand *multiple* file systems
 - **Single disk may contain more than one**
 - **Layout defined by “partition label”**
 - ...and “extended partition label”
- ♦ But...but...boot loader is **510 bytes** of code!



Multi-Stage Boot Loader

- ♦ GRUB is larger than one sector
- ♦ First sector, loaded in by the BIOS...
 - ...just loads the rest of the boot loader
 - **“GRUB Loading stage2”**
- ♦ GRUB then presents boot menu
- ♦ The OS-load challenge
 - BIOS runs in real mode – only 1 meg of RAM!
 - OS “may be” larger than 1 meg
 - **Linux – often; Windows – absolutely!**



Brain-Switching

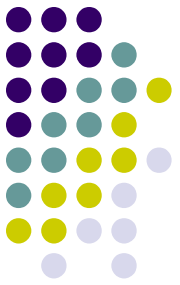
- ♦ Switch back and forth between real and protected mode
 - Real mode: BIOS works, provides disk driver
 - Protected mode: can access lots of memory
- ♦ Switching code is tricky
 - Somewhat like OS process context switch
 - Roughly 16 carefully-crafted instructions each way
- ♦ Load done \Rightarrow jump to the kernel's entry point
 - How do we know the kernel's entry point?

Entry Point, Binary Format, ...

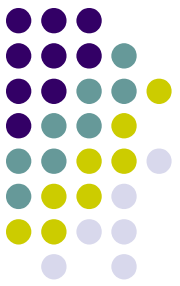


- ♦ Can't we just jump to the first byte?

Entry Point, Binary Format, ...



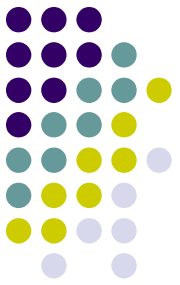
- ♦ Can't we just jump to the first byte?
- ♦ Probably not
 - If kernel is a “regular executable” it begins with an “executable file header” (e.g., ELF)
 - If the OS has the concept of “BSS”, the zeroes aren't in the file...
- ♦ Loading the bytes into RAM isn't enough
 - We must understand & mutate them



Multiboot Specification

- ♦ Attempt to define “portable kernel format”
- ♦ Multiboot “standard”
 - Binary specifies entry point &c
- ♦ The multiboot header must be located in the first 8192 bytes
- ♦ This is part of the mysterious 410kern/boot/head.S...

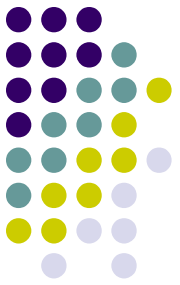
0x1badb002
flags
checksum
header_addr
load_addr
load_end_addr
bss_end_addr
entry_addr



410 “Pebbles” (from OSkit)

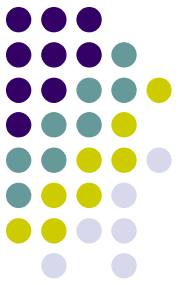
- ♦ Entry point is asm function `_start` in `head.o`
- ♦ This calls other assembly code to set up GDT, TSS, IDT
- ♦ This calls the first C function, `mb_entry()`

OSkit



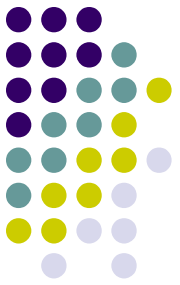
- ♦ `mb_entry()` calls:
 - `mb_util_Imm()`: tell LMM which RAM the BIOS and boot loader say to use
 - `mb_util_cmdline()`: parse “command line” provided by boot loader (yielding `argv[]`)
 - `sim_booted()`: tell Simics which kernel to debug
 - `kernel_main()`...that's you!

PXE



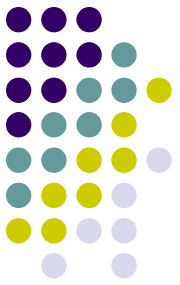
- ♦ Preboot Execution Environment
- ♦ “How a PC should net boot”
 - DHCP protocol extensions to say
 - “I am a **PXE client of DHCP**”
 - “**My machine ID is ... my hardware type is ...**”
 - DHCP server assigns IP address
 - **Instructs client: network settings, TFTP server, file**
 - Client downloads 2nd-stage boot via TFTP
- ♦ PXE libraries for downloaded loader to use
 - Ethernet, UDP, TFTP

APM



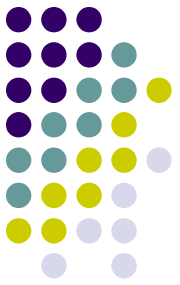
- ♦ Advanced Power Management
- ♦ Problem – Laptop hardware is “special”
 - Lots of power-critical hardware
 - Totally different from one machine to another
 - **Disk spin-down (“standard”, so may be fairly easy)**
 - **Display backlight, processor speed (not so easy)**
 - **South bridge, DRAM controller, keyboard...**
 - Sequencing these in the right order is very machine-specific
- ♦ Problem – user does things (close lid...)

APM



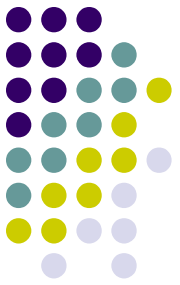
- ♦ Solution - “power kernel”
 - OS asks it to control power hardware
 - Power hardware tells OS about events
 - **Lid closed**
 - **Battery low**
- ♦ Complex rules for messaging back and forth
 - OS required to poll APM periodically
 - **May involve switch to 16-bit mode**
 - Suspend protocol: prepare/commit/abort...

ACPI



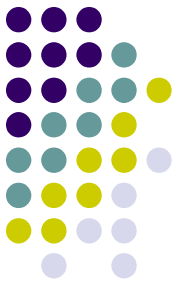
- ♦ Advanced Configuration & Power Interface
 - APM's “big brother”
- ♦ Good news
 - OS gets more understanding, control
 - BIOS provides ACPI code to OS in virtual-machine format

ACPI



- ♦ Bad news – implementation
 - What the BIOS tells you is often wrong
 - **Many “on this machine, patch this to that” fixes necessary**
 - **FreeBSD kernel contains “BIOS blacklist”**
 - Strings identifying BIOS versions known to have fatal ACPI bugs
 - **ACPI virtual-machine code often depends on being run by one particular virtual machine**
 - **ACPI “OS-independent” virtual machine code checks which OS is executing it and behaves differently(!!)**

ACPI



- ◆ Bad news – structural
 - Interaction between ACPI and other code is delicate and fraught with peril
 - **Should VGA BIOS “reset method” be called *before* or *after* restoring ACPI video device state?**
- ◆ Bad news – throw weight
 - Specification pages
 - **1.0 = 400**
 - **2.0 = 500**
 - **3.0 = 600**
 - **4.0 = 700**
 - **5.0 = 900**

“Big Iron” (mainframes)



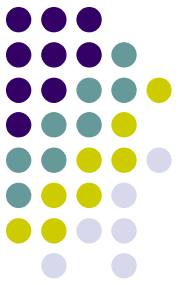
- ♦ “Boot loader” may be a separate machine
 - When main CPU powers on, it does not run code!
 - “Front-end processor” tasks
 - **Run thorough diagnostics on main machine**
 - **Store OS into its memory**
 - **Set its program counter to entry point**
 - **Turn on instruction fetching**
- ♦ “Front-end” also contains a debugger
 - Useful when your OS crashes

Open Firmware



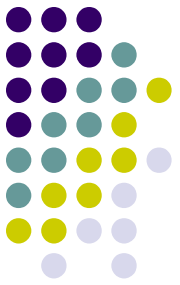
- ♦ Sun & Mac hardware (until 2006, sigh)
- ♦ Goal: share devices across processor families
 - Ethernet, SCSI disk controller, ...
- ♦ Solution
 - Processor-independent BIOS modules on cards
 - Collection of FORTH methods
 - **test, boot, open, close, read, write, etc.**
- ♦ “Boot ROM” may contain a small debugger
 - Sun, Mac do this... PCs are just starting to catch up

EFI



- ♦ “Next big thing” in the PC world
 - Including PC's made by Apple(!?)
- ♦ “Super sized”: #partitions, partition labels, ...
- ♦ More device drivers (not just disk, video)
 - May be signed, certified, protected
- ♦ Arrived mostly with x86-64 machines
- ♦ Many more interfaces, larger interfaces
 - Spec pages: EFI 1.10 = 1100, UEFI 2.1 = 1682, ...
 - EFI+ACPI: 2300 pages of fun for the whole family

Summary



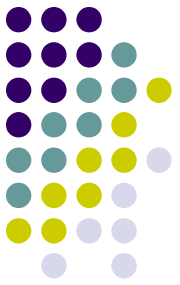
- ♦ It's a long, strange trip
 - Power on: maybe no RAM, maybe no CPU!!
 - **Maybe beep, maybe draw a sad face**
 - Locate OS
 - Load (N stages)
 - Tell kernel about the machine and the boot params
 - Provide support to kernel once it's running

Further Reading



- ♦ More BIOS details
 - <http://www.pcguide.com/ref/mbsys/bios/bootSequence-c.html>
 - <http://bioscentral.com/>
- ♦ A real memory tester - memtest86.com
- ♦ Open-source BIOS!
 - www.linuxbios.org
 - openbios.info
- ♦ PXE
 - <ftp://download.intel.com/labs/manage/wfm/download/pxespec.pdf>
 - <http://ipxe.org>

Further Reading



- ♦ ACPI
 - <http://www.acpi.info>
- ♦ EFI
 - <http://www.uefi.org>
 - (old) <http://www.intel.com/technology/efi/>