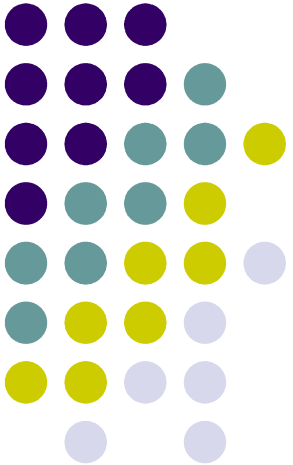
Bootstrapping

Steve Muckle

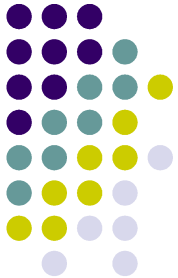
Dave Eckhardt



Synchronization

- Project 3 checkpoint 1
  - Bboard post, web page
  - Paging, COW optional
- “No linked lists” - not what I said (I think)
- Homework 1: Monday 17:00
- Exam: Tuesday evening (19:00)
  - Known conflicts will receive e-mail from me
- Monday: Review

Motivation

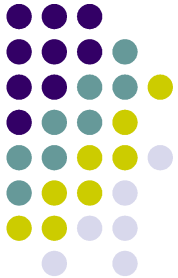


- What happens when you turn on your PC?
- How do we get to main() in kernel.c?

Overview

- Requirements of Booting
- Ground Zero
- The BIOS
- The Boot Loader
- Our projects: Multiboot, OSKit

Requirements of Booting

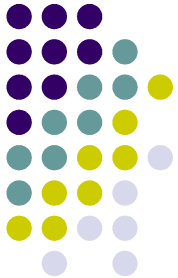


- Initialize machine to a known state
- Make sure basic hardware works
- Load a real operating system
- Run the real operating system

Ground Zero

- You turn on the machine
- Execution begins in real mode at a specific memory address
  - Real mode: only 1mb of memory is addressable
  - Start address is in an area mapped to BIOS (r/o)
- What's the BIOS?

Basic Input/Output System (BIOS)

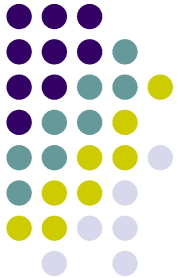


- Code stored in mostly-read-only memory
  - Flash, previously EEPROM
- Configures hardware details
  - RAM refresh rate or bus speed
  - Password protection
  - Boot-device order
- Loads OS, acts as mini-OS
- Scary things (power management)

BIOS POST

- Power On Self Test (POST)
- Scan for critical resources
  - RAM
    - Test it (only a little!)
  - Graphics card
  - Keyboard
- Missing something?
  - Beep

BIOS Boot-Device Search

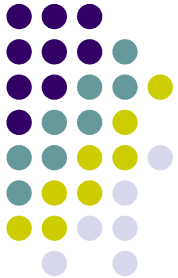


- Consult settings for selected order
  - “A: C: G:” (maybe PXE)
- Load the first sector from a boot device
  - could be a floppy, hard disk, CDROM
  - without a BIOS, we’d be in a bit of a jam
- If the last two bytes are AA55, we’re set
- Otherwise look somewhere else
  - “No Operating System Present”

BIOS Boot-Sector Launch

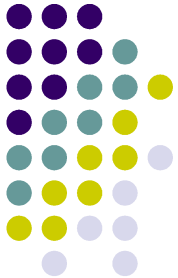
- Sector is copied to 0x7C00
- Execution is transferred to 0x7C00
- If it’s a hard disk or CDROM, there’s an extra step or two (end result is the same)
- Now we’re executing the bootloader – the first “software” to execute on the PC

Bootloader



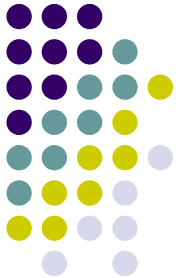
- We’re now executing a bootloader
- Some bootloaders exist to load one OS
- Others give you a choice of which to load
- We use grub
  http://www.gnu.org/software/grub/

Bootloader's Job



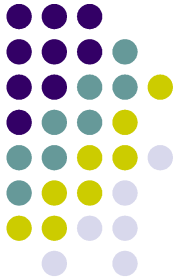
- Mission: load operating system
- But where?
  - May need to understand a file system
    - Directories, inodes, symbolic links!
  - May need to understand multiple file systems
    - Single disk may contain more than one
    - Layout defined by “partition label”
      - ...and “extended partition label”
- Recall: Boot loader is 510 bytes of code!

Multi-Stage Boot Loader



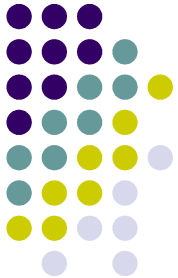
- GRUB is larger than one sector
- Sector loaded in by the BIOS just…
  - ...loads the rest of the boot loader
- GRUB then presents boot menu
- OS load challenge
  - BIOS runs in real mode – only 1 meg of RAM!
  - OS may be larger than 1 meg

Brain-switching

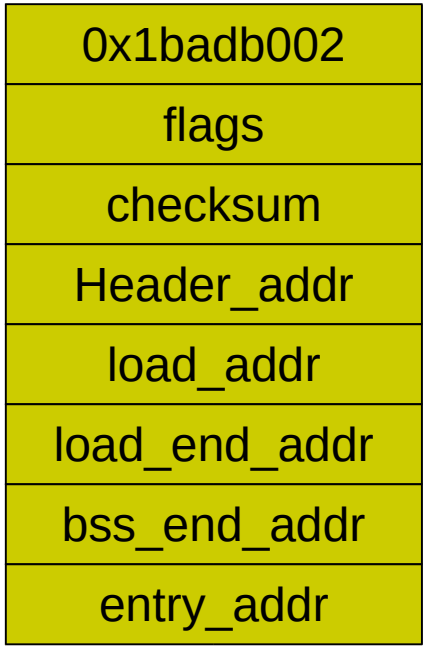


- Switch back and forth between real and protected mode
  - Real mode: BIOS works, can operate disk
  - Protected mode: can access lots of memory
- Switching code is tricky
  - Somewhat like OS process context switch
- Done: jump to the kernel's entry point
  - How do we know the kernel's entrypoint?

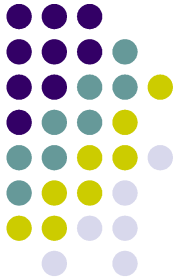
Multiboot Specification



- Many OSes require their own bootloader
- Multiboot “standard”
  - Kernel specifies entry point &c
- The multiboot header must be located in the first 8192 bytes
- This is the mysterious multiboot.o…

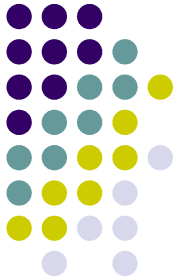


410 “Pebbles” (from Oskit)



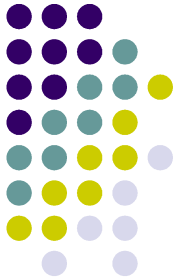
- Entry point is asm function in multiboot.o
- This calls the first C function, multiboot_main

OSkit



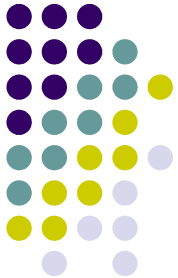
- multiboot_main calls:
  - base_cpu_setup: init GDT, IDT, and TSS
  - base_multiboot_init_mem: init LMM
  - base_multiboot_init_cmdline
    - parse cmdline passed to kernel by bootloader
- - main (yes, your main in kernel.c!)
  - exit, if main ever returns
    - press a key to reboot…

Other Universes



- OpenFirmware
  - Sun & Mac hardware
  - Goal: share devices across processor families
  - Solution: FORTH boot-loader code in each device
- “Big Iron” (mainframes)
  - “Boot loader” may be a separate machine
  - Run thorough diagnostics on main machine
  - Debugger support for crashes

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