Bootstrapping

Steve Muckle
Dave Eckhardt

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
- 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**
  - Start address is “end of memory”
    - **0xFFFF0**
    - **Contains a jump to the real BIOS entry point**
- What’s the 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 0xC000
  - 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, 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

- 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
    - **Moves itself elsewhere in memory**
    - **Loads partition boot sector at 0x7C00**
- Now we’re executing the bootloader – the first “software” to execute on the PC

Bootloader

- Some bootloaders exist 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/

Bootloader's Job

- Mission: load operating system
- From where?
  - “/boot/sokoban.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”
- Recall: 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 stage 1”**
- 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
    - **Linux – often; Windows – absolutely!**

Brain-switching

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

Entry Point, Binary Format, ...

- Can't we just jump to the beginning?
- Probably not
  - If kernel is a “regular executable” it begins with an “executable file header”...
  - 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

- 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…

0x1badb002
flags
checksum
Header_addr
load_addr
load_end_addr
bss_end_addr
entry_addr

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
- - kernel_main() (at last, your code!)
  - exit(), if kernel_main() ever returns
    - press a key to reboot…

PXE

- Preboot Execution Environment
- “How a PC should net boot”
  - DHCP extensions to say
    - **“I am a PXE client of DHCP”**
    - **“My machine ID is ... my hardware type is ...”**
  - Libraries for downloaded boot 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 (standar, so 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

“Big Iron” (mainframes)

- “Boot loader” may be a <u>separate machine</u>
  - Main CPU powers on, does not run code
  - “Front-end” 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
- Goal: share devices across processor families
  - Ethernet, SCSI disk controller, ...
- Solution
  - Processor-independent BIOS modules on cards
  - Collection of FORTH methods
    - **test, boot, etc.**
- “Boot ROM” may contain a small debugger
  - Sun, Mac do this... PCs are just starting to catch up

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://howstuffworks.lycoszone.com/bios2.htm
  - http://bioscentral.com/
- A real memory tester - memtest86.com
- Open-source BIOS!
  - www.linuxbios.org
- PXE ftp://download.intel.com/labs/manage/wfm/download/pxespec.pdf