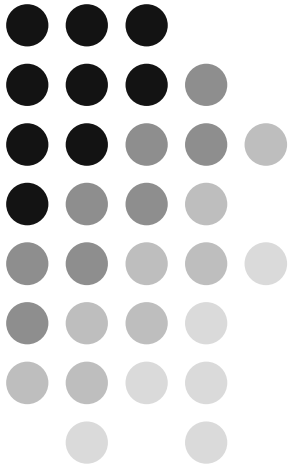
Bootstrapping on x86

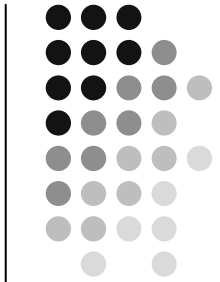
Steve Muckle

Wednesday, March 12th 2003

15-412 Spring 2003

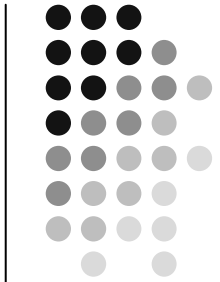


Motivation



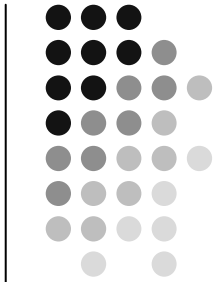
- What happens when you turn on your PC?
- How do we get to main() in kernel.c?

Overview



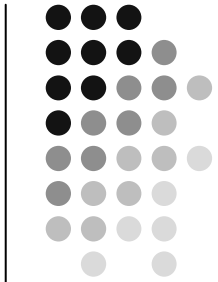
- Requirements of Booting
- Ground Zero
- The BIOS
- The Bootloader
- Our projects: Multiboot, OSKit

Requirements of Booting



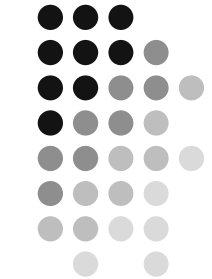
- Initialize machine to a known state
- Make sure basic hardware works
- Load a real operating system
- Run the real operating system

Ground Zero



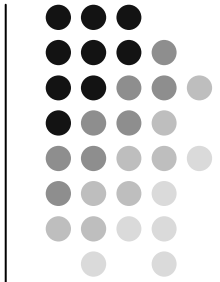
- You turn on the machine
- Execution begins in real mode at a specific memory address
- Real mode: only 1mb of memory is addressable
- Start address is in an area mapped to BIOS read-only memory
- What's the BIOS?

Basic Input/Output System (BIOS)



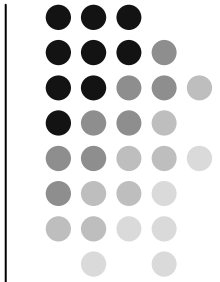
- Code stored in Electrically Erasable Programmable Read Only Memory (EEPROM) on most modern systems
- Useful for testing hardware and loading data from storage into memory
- Can also be used to configure hardware details like RAM refresh rate or bus speed

Basic Input/Output System (BIOS)



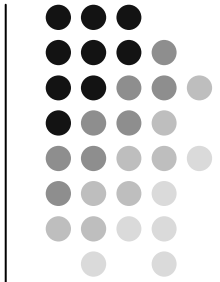
- BIOS performs a Power On Self Test (POST)
- BIOS loads the first sector from a boot device
  - could be a floppy, hard disk, CDROM
  - without a BIOS, we'd be in a bit of a jam
- If the last two bytes are AA55, we're in business
- Otherwise we look somewhere else

Basic Input/Output System (BIOS)



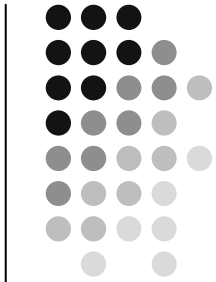
- Sector is copied to 0x7C00
- Execution is transferred to 0x7C00
- If it's a hard disk or CDROM, there's an extra step or two (end result is the same)
- Now we're executing the bootloader – the first "software" to execute on the PC

Bootloader



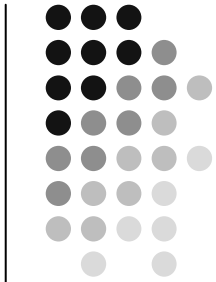
- We’re now executing a bootloader
- Some bootloaders exist to load one OS
- Others give you a choice of what to load
- We use grub
  http://www.gnu.org/software/grub/

Bootloader



- GRUB is larger than one sector
- The sector loaded in by the BIOS just loads… the rest of the bootloader
- GRUB then presents you with a boot menu
- To load a kernel, it must switch back and forth between real and protected mode
- It then jumps to the kernel's entrypoint
  - How do we know the kernel's entrypoint?

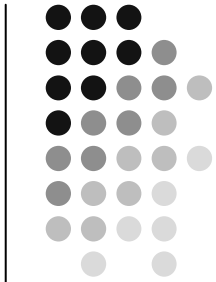
Multiboot Specification



- Many OSes require their own bootloader
- Multiboot offers a standard way for kernels to communicate entrypoint and other info
- The multiboot header must be located in the 8192 bytes
- This is the mysterious multiboot.o…

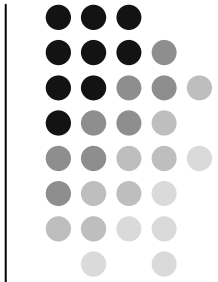
0x1badb002
flags
checksum
header_addr
load_addr
load_end_addr
bss_end_addr
entry_addr

OSkit



- The kernel entrypoint is an assembly 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…)