Problems to over come, details to deal with

- BIOS
  - Initially only ~512 *bytes* of an OS are loaded
  - disk geometry needs to be known at boot time
  - needed for all disk I/O until we can access the actual kernel’s drivers
  - Not accessible from protected mode
  - Provides a common API for dealing with hardware
- Real Mode
  - can only access first 640KB of RAM (address 0xA00000)
  - 20 bits addressable (1MB), but only with BIOS hacks
- Protected Mode
  - needed to access full 4GB (32 bit) address space
  - implements protected memory and page management

Booting from a floppy: simple boot example

- assume compressed system image is <512 (explained later)
- assume *not* using LILO (handled later)
- follow along in ./arch/i386/boot/bootsect.S
- BIOS has just loaded the contents of sector zero (512 bytes) into memory
- CS=0 IP=0x7c00 (line 57)
- currently in real mode
- Intel Syntax assembly

- Step1: move loader out of the way
  - lines 57-65
  - out of the way: to address 0x90000, near the end of addressable memory
  - loader is <256 bytes in size
  - ’rep’ and ’movsw’ do the work
  - reason: clear up mem for large contiguous chunk
- Step 2: load setup and system from disk
  - do some guess work to find out geometry of floppy drive (103-125)
  - from drive 0, head 0, track 0, sector 2, read the next two sectors into memory at 0x90200 (lines 131-197)
  - print "loading" message via BIOS
  - load "system" into memory at 0x010000 (64KB) (lines 213-218)
  - the first 64KB is used for BIOS memory, so leave it alone
  - run setup (line 248)

Booting from a floppy: setup.S

- follow along in ./arch/i386/boot/setup.S
- still in real mode
- CS/IP = 0x90200
- Intel Syntax assembly
- Step 4: load more of setup if need be
  - recall only 2 sectors (1KB) of setup was loaded by loader
  - look for a 2 word signature at end of setup address
  - if it does not exist, load the next 4 sectors off the disk (2KB)
  - if signature still not found, assume error, else move on
  - reason: this file deals with different types of boot processes, some with different setup lengths (?)
- Step 5: initialize hardware (via BIOS)
  - set keyboard repeat parameters
  - initialize video in ./arch/i386/boot/video.S
  - check for hd0 and hd1
  - initialize power management
  - check for Micro Channel Bus (??)
  - check for PS/2 mouse (??)
- Step 6: move system into place, and go into protected mode
  - all hardware needed for booting already initialized by BIOS
  - initialize interrupt controlers
  - move kernel to 0x01000 (4KB), first page reserved (475-495)
  - the protected mode magic jump happens at line 638
  - final act is to jump to 0x01000 for decompression

Booting from a floppy: decompressing kernel

- follow along in ./arch/i386/boot/compressed/head.S
- in protected mode (no more 640KB limit)
- no BIOS
- assuming single CPU (first time we needed this assumption)
- CS=0x0000 IP=0x1000 (4KB)
- changed to AT&T syntax (!?!)
- Note: standard linux kernel has a 1+MB memory requirement
- Step 7:
  - makes calls to ./arch/i386/boot/compressed/misc.c (gzip library: c code)
  - decompresses kernel to 0x0010:0000 (1MB) (88-94)
  - jumps to newly decompressed kernel (line 118)
  - lands in ./arch/i386/kernel/head.S

Booting from a floppy: real kernel head.S

- follow along in ./arch/i386/kernel/head.S
- protected mode (here to stay)
- CS:IP 0x0010:0000
- Step 8: Initialize Laundry list of kernel things
  - setup paging (71-83)
  - setup interrupt descriptor table: (106, 283-297)
  - save boot parameters from page 0x0000 (119-136)
  - get CPU id/vendor info and save it (178-198)
  - initialize any co-processors (212, 258-271)
  - jump to ./init/main.c: start_kernel (245)

Booting from a floppy: using a "big" kernel

- "big" kernel is one whose compressed image is >512KB
- using BIOS calls, you can move memory into high memory
- Changes in loader
  - in the loader, when loading the compressed image, jump indirectly to bootsect_helper in setup.S, and return
  - in 64KB chunks copy image to 0x0010:0000 (1MB)
- Changes in setup
  - instead of moving compressed kernel from 0x10000 to 0x1000, leave it in place (that is not where it exists)
  - jump into high memory address 0x0010:0000 (lines 643-654)
- Changes in head.S/misc.c
  - kernel decompressed to low memory first, then to high memory if needed
  - moves copying instructions to low memory
  - moves finals uncompressed kernel to 0x0010:0000 (where it is expected)
  - jumps to 0x0010:0000