Slide 1

Administrivia

- Reminder: Homework 6 due today. Homework 7 due next Monday.

Slide 2

The Boot Process

- What happens between the time you turn the computer on (or initiate reboot) and the point at which you get a login prompt is . . . complicated, mysterious, and involves both hardware and software.
- Today's topic is to demystify it as much as possible. Textbook has some useful short information, indexed under "boot" and "BIOS". I'm basing this lecture on that, a book *Linux Kernel Internals* and various online sources.

Slide 3

Booting — Hardware

- When a PC is powered on, hardware starts the BIOS (Basic Input Output System), a program that lives in/on some form of nonvolatile memory. It contains functions to read from the keyboard, write to the screen, and do disk I/O. (Caveat: Recent hardware replaces this with UEFI (Unified Extensible Firmware Interface). Conceptually similar as best I can tell.)
- This BIOS first does a "Power-On Self Test" (POST) — check how much memory is installed, whether basic devices are installed and responding.
- It determines which device to try to boot from based on information also stored in non-volatile memory. It then reads the first sector from this device — "boot sector" or "master boot record".

Slide 4

Boot Sector / Master Boot Record

- First sector on device from which we're booting must contain (in a format known to the hardware / BIOS) a little bit of code, enough to get the boot process going.
- For partitioned devices, this first sector (MBR) also contains a partition table, indicating which partition contains the logical device from which booting is supposed to be done, and where to find that logical device's boot sector.
- Either way, we get a little bit of code, which when executed (presumably with the help of the BIOS) reads in — something else — from disk to memory, and transfers control to it. The "something else" could be the actual operating system, or a "boot loader" (such as GRUB or LILO, for Linux systems).
- (From here on, the discussion will be somewhat Linux-specific.)

Slide 5

Boot Loader

- GRUB looks at configuration files (findable from `/boot`), possibly gets input from the keyboard, and decides what to boot.
- If it's Linux, part of the configuration is the name of the file containing the (compressed) kernel. This gets uncompressed and read into memory, and control is transferred to it.
- (What happens if it's Windows being booted? good question, but my guess is that GRUB reads in whatever boot sector would have been used to boot Windows in a single-boot system, and transfers control to its little bit of code).

Slide 6

Starting the Kernel

- First thing executed is assembly code that does hardware initialization, including:
- Put the processor in protected mode.
- Set up kernel stack.
- Do initialization for the MMU (set up page table for kernel).
- Do initialization for interrupt processing (interrupt table/vector).

Slide 7

Starting the Kernel, Continued

- Next, control is transferred to a C function that begins initializing data structures for the kernel.
- What's executing at this point is "process 0", which will become the "idle process", after doing a little more initialization.

Slide 8

Initialization (Old Way)

- Daemons to manage the buffer cache (`bdflush`) and swapping (`kswap`) are started.
- Filesystems are initialized and the root filesystem mounted.
- An attempt is made to connect with the console and open file descriptors for stdin, stdout, stderr.
- `init` program is found and started.

Slide 9

Initialization — `init` Program

- Background: UNIX/Linux has a notion of "run levels" — typically 1 is single-user, 3 is text-only, 5 is graphical, etc.
- `init` does more initialization (including closing/reopening stdin, etc.), reads `/etc/inittab`, and "does what it says", depending on run level. Default level (for boot) is specified in `/etc/inittab`. Rest of the file says what to do, depending on run level. Some of "what to do" involves running scripts in `/etc/rc.d`.
- Typically some of what's started is one or more processes that accept logins — "virtual consoles" and/or graphical login manager.
- `init` then waits for any requests to change the runlevel (e.g., using command `init`).

Slide 10

Initialization (Old Versus New)

- The preceding slides describe how things were in UNIX/Linux for a long time.
- Over the past few years many Linux distributions have dropped `init` in favor of `systemd`. Controversial move ("what was wrong with the old way?"), but proponents of `systemd` say it's more capable and flexible.
- With `systemd`, apparently kernel starts it up immediately, rather than first starting some daemons and mounting filesystems

Slide 11

Initialization — `systemd` Program

- Filesystems defined in `/etc/fstab` are mounted.
- `/etc/systemd/system/default.target` is read to determine what services to start. (It's actually a symbolic link to another file, making it possible to start the system either with a graphical login prompt or in "headless" mode.)
- Like `init`, `systemd` then waits for requests to start/stop/restart services (using `systemctl`).

Slide 12

Minute Essay

- None really — sign in.
- (And best wishes for a good holiday!)