Administrivia

- Reminder: Homework 7 due Monday.

- Homework 3 programming problem graded at last! Sample solution on the Web.

Slide 1

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 2

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.

Slide 3

- 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".

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.

Slide 4

- For partitioned devices, this first sector (MBR) 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 LILO or GRUB, for Linux systems).

- (From here on, the discussion will be somewhat Linux-specific, and alas will be based on Linux as it existed a few years ago.)

Boot Loader

- LILO (or GRUB) looks at configuration files, possibly gets input from the keyboard, and decides what to boot. (This used to be somewhat transparent, with configuration in `/boot/grub/grub.conf`, but has become less so.)

Slide 5

- 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 LILO/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).

Starting the Kernel

- First thing executed is assembly code that does hardware initialization, including:

- Put the processor in protected mode.

- Do initialization for the MMU (set up page table for kernel).

Slide 6

- Do initialization for interrupt processing (interrupt table/vector).

Starting the Kernel, Continued

- Next, control is transferred to C function start_kernel, which 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 7

Initialization — Process 0

- 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.

Slide 8

- An attempt is made to execute one of /etc/init, /bin/init, /sbin/init.

Initialization — `init` Program

- (This is based on how things were some years ago. On most distributions, replaced by something called `systemd`, about which people have opinions!).

- 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

Slide 9

---

command `init`). Changing the runlevel — look again at `/etc/inittab`.

Slide 10

Minute Essay

- None really — sign in.

- (And best wishes for a good holiday!)

Slide 11