Slide 1

## Administrivia

- Reminder: Homework 7 and 8 due next week (7 Monday, 8 accepted without penalty through Wednesday).
- Sample solutions to everything will be available before the exam — most things Monday, Homework 8 Wednesday.

Slide 2

## Minute Essay From Last Lecture

- How to embed macros/code in word-processing documents, PDFs, etc. Thoughts?
- Preferences in operating systems and whether some operating systems really are better. Thoughts?
- Mobile O/S.
- Network boots.

Slide 3

## Minute Essay From Last Lecture

- How to embed macros/code in word-processing documents, PDFs, etc. (Seems like it's a matter of file format, plus having a sensible way to represent executable code. VBA could be in source form, no?)
- Preferences in operating systems and whether some operating systems really are better. (What does "better" mean? different criteria/goals, no? various kinds of compability matter, no?)
- Mobile O/S. (Seems like it's the same basic problem as for all O/S, just possibly with different tradeoffs.)
- Network boots. (Seems like the key difference is getting data from a network connection rather than a local disk.)

Slide 4

## 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 going by that, from the book *Linux Kernel Internals* (see syllabus), and various online sources.

Slide 5

## 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.
- 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 6

## 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) 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.)

Slide 7

## Boot Loader

- LILO (or GRUB) looks at configuration files, 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 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).

Slide 8

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

Slide 9

## 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 10

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

Slide 11

## Initialization — `init` Program

- Aside: 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`. (This part of startup has changed in recent versions of RedHat-and-related Linux distributions.)
- `init` then waits for any requests to change the runlevel (e.g., using command `init`). Changing the runlevel — look again at `/etc/inittab`.

Slide 12

## Minute Essay

- None required — quiz.
- However, if you have time, now or later:

  Over the course of the semester I've told several "war stories" — tales of woe that taught me (or someone) something. Do you have a favorite war story to tell? (I'll read these in class Monday unless you tell me not to.)