Slide 1

Administrivia

- Homework 3 on Web soon. Will consist of applying updates and (I think) compiling a new kernel. (Comments?)

Slide 2

Kernels

- In general (as taught in o/s courses), the job of the operating system is to
  - Provide "virtual machine" abstractions — processes, filesystems, etc.
  - Manage hardware resources on behalf of user programs.
- Kernel is heart of o/s — the part that's always loaded (in virtual memory anyway). Responsible for many system activities:
  - Managing processes (timesharing, address-space protection).
  - Providing interprocess communication (signals and semaphores, pipes).
  - Implementing virtual memory (paging, swapping, etc.).
  - Managing filesystems.

Slide 3

Device Drivers

- At lowest level, communicating with I/O devices is full of complicated and device-specific details.

  Big, big gap between these details and the application-program view of I/O (e.g., "read 10 bytes from file Z").
- Most o/s's bridge this gap with several "layers of abstraction".
- Lowest level of abstraction — "device driver" — encapsulates details of communicating with specific device, allows kernel to do I/O in terms such as "read block 200 from device 3".
- Can be incorporated into kernel, or loaded dynamically.

Slide 4

Configuring/Recompiling the Kernel

- Why would you want to?
  - For some systems, only way to get exactly the right set of device drivers / options.

    (In the old days of mainframes, had to perform "system generation" (sysgen), specifying all kinds of details about I/O configuration.)
  - As an efficiency measure — so unneeded drivers and options aren't included.
- How do you do it? Details vary among versions of Unix. For Linux, steps are all packaged as makefile in `/usr/src/linux*`. First `make xconfig` (creates a `.config` file specifying options), then `make` other targets to create kernel, loadable modules, etc., and copy to appropriate directories. (Of course, it's sensible not to overwrite old files until you know the new ones work!)

Slide 5

Tuning the Kernel

- Some systems also let you set some kernel parameters (e.g., maximum number of open files a process can have) "on the fly". Details vary among versions of Unix. For Linux, many are set by overwriting files in `/proc`. To do this "permanently", must add the overwrite commands to startup scripts (more about that later).
- (Aside: `/proc` filesystem is a sort of phony filesystem allowing you to examine/change o/s data structures. Take a look sometime!)

Slide 6

Device Drivers, Continued

- A key Unix principle — "everything's a file". So, (non-network) devices are represented as "files" in `/dev`. Two basic types, block and character. Associated with each — major and minor device numbers.
- Device drivers translate generic I/O operations (read, write, open, probe, etc.) into device-specific details. How? Each driver contains some or all corresponding functions; o/s figures out whose version of, e.g., `read` to call based on device number, etc.

  (Usually drivers "drive" actual I/O hardware, but not always — e.g., pseudo-TTY emulates old-time terminals, so programs written for them still work.)
- For Linux, source for drivers can be found in `/usr/src/linux*/drivers`.
- How to add a device driver? Details depend on version of Unix. Seems to be

Slide 7

easy for Solaris, complicated for Linux.

Slide 8

Loadable Kernel Modules

- Idea is to allow adding/removing kernel services (such as device drivers) without recompiling kernel. Support for this varies among versions of Unix; very good in Solaris, okay in Linux, etc.
- For Linux, `lsmod` lists currently loaded modules. `insmod` to add, `rmmod` to remove, `modprobe` to add/remove in a way that deals with dependencies, pre- and post- stuff.

Slide 9

Booting the System

- "Boot" is short for . . . ?

  (Some of the old mainframes called it something else — e.g., IPL (initial program load). More descriptive, but less colorful?)

- First step is for ROM/BIOS to read boot record from device being booted from — contains a small program that then loads the rest of the kernel. Not part of Unix, so details vary from system to system.

- Kernel then initializes itself, including checking how much memory is available, what hardware is present.

- (Continued on next slide . . .)

Slide 10

Booting the System, Continued

- Kernel creates "spontaneous" processes. Details vary from one version to another. For Linux, includes `init` and various handlers (e.g., `kswapd`).

- `init` process then mounts and initializes filesystems (performing `fsck` if needed).

- `init` executes startup shell scripts, which bring up (most of) the rest of the system. More later.

- All versions allow booting in normal mode or "single-user" mode. In single-user mode, more of what happens must/can be done manually. The single user is root, and some systems don't require you to supply a password to log in. Helpful if you forget the root password, but clearly some risk!

Slide 11

Boot Loaders Revisited

- On PC hardware, MBR contains program that loads "second-stage boot loader" from selected disk partition, which then loads the rest . . .
- For Linux systems, boot loaders include `lilo` and `grub`. Both make possible multiboot systems. To change options/configuration, must edit appropriate file and then reinstall boot loader.

Slide 12

Startup Scripts

- Kernel initialization starts up only a very few necessary services. Everything else is started by running "startup scripts" — normal shell scripts, kept in some location that (of course!) varies among versions of Unix.

  These days, most configuration changes are made not by actually changing these scripts but by changing the configuration files they read.
- In version of Unix based on (AT&T System V), there's a notion of "run levels" (1 is single-user, 2 through 5 are multi-user — e.g., on Linux, 3 is multi-user but no X, 5 is multi-user with X). `/etc/inittab` defines (some of) what happens at various levels.
- Startup scripts then live in `init.d` directory, usually in `/etc`. Each is responsible for one daemon (persistent background process) or other part of the system. All must accept `start` and `stop` parameters; some also allow `restart`. (Root can use these scripts to stop/start/restart things.) Which ones are run? Defined in `/etc/rc.d` and subdirectories.

Slide 13

Startup Scripts, Continued

- Many scripts use configuration files, often also in `/etc`. Details vary from version to version, and even from Linux distribution to distribution. For RedHat-based distributions, many things in `/etc/sysconfig` directory. (E.g., this is where information about networking is kept.)

Slide 14

Shutting Down

- No surprise — just turning off the power is not a good idea, since much filesystem info is cached and needs to be written out to disk.
- Several methods for graceful shutdown — `shutdown`, `halt`, `reboot`. Also may be options from graphical login screen.
- Can also change runlevel with `init` command. E.g., `init 3` (on Linux) shuts down X. `init 5` to start it up again.

Slide 15

Minute Essay

- None — sign in.