Slide 1

Administrivia

- Reminder: Homework 6 due Friday (course Web site says today, for simplicity, but okay to turn in through Friday).

  (I just fixed a little glitch in one of the problems, namely the powers of 2 corresponding to K, M, and G.)
- Homework 7 on the Web. Due the Monday after the holiday.

Slide 2

GUIs — Review/Recap

- Keyboard: Hardware delivers very low-level info (individual key press/release actions). Device driver translates these to character codes, typically using configurable keymap.
- Mouse: Hardware delivers very low-level info (change in coordinates, status of buttons).
- Display: Hardware can be fairly simple ("raster graphics") or pretty sophisticated (independent processor capable of independent operation). How the O/S communicates with it varies by platform — Windows approach explicitly object-oriented, traditional UNIX/Linux based on protocol (X11) that works over network too.

Slide 3

GUIs — Programs

- Of course, many examples of software using this kind of device.
- Libraries for writing such software vary by language:
- Java and Scala include lots of library classes, mostly fairly high-level/abstract.
- Nothing standard in C, but most platforms offer various libraries. Lowest-level one in UNIXworld is “X11”.

Slide 4

GUI-Based Programming

- Input from keyboard and mouse captured by O/S and turned into messages to process owning appropriate window.
- Typical structure of GUI-based program is a loop to receive and dispatch these messages — “event-driven” style of programming.
- Details vary between Windows and X, but overall idea is similar. (Examples in Figures 5-34 and 5-36.) I’ve also written programs using the fairly low-level X11 interface, but — maybe not. But it’s doable, even from C, though of course not completely portable.

Slide 5

Network Terminals — Hardware

- Keyboard, mouse, and display as described previously, plus local processor; connected to remote system.
- Local processor can be very capable (X terminal, or even PC configured to run as one) or more primitive.

Slide 6

I/O in Windows

- Hardware Abstraction Layer (HAL) attempts to insulate rest of O/S from some low-level details — e.g., I/O using ports versus memory-mapped I/O.
- Standard interface to device drivers — Windows Driver Model. Drivers are passed I/O Request Packet objects.

Slide 7

I/O in UNIX/Linux

- Access to devices provided by special files (normally in `/dev/*`), to provide uniform interface for callers. Two categories, block and character. Each defines interface (set of functions) to device driver. Associated with each special file are major and minor device numbers, with major device number used to locate specific function. (Look at some output of `ls -l /dev`.)
- For block devices, buffer cache contains blocks recently/frequently used.
- For character devices, optional line-discipline layer provides some of what we described for text-terminal keyboard driver.
- Streams provide additional layer of abstraction for callers — can interface to files, terminals, etc. (This is what you access with `*scanf`, `*printf`.)

Slide 8

"Everything's a File" Revisited

- I mentioned the pseudofilesystem `/proc`? which supposedly you can read/write just as if it were a file?
- I wrote some throwaway code to access "files" within it and learned(?) that while C stream I/O (`fopen`, `fgetc`, etc.) didn't work well, the lower-level routines (`open`, `read`, etc.) did.

Slide 9

Linux Memory Management, Revisited

- I mentioned in a previous class that Linux systems (often?) "overcommit" memory — allow you to allocate more than you can actually use?
- I wrote a couple of programs illustrating this in action . . .

Slide 10

Minute Essay

- Do you plan to be here Monday?