Administrivia

Slide 1

- Reminder: Homework 6 due today.
- Reminder: Quiz 6 Wednesday.
- Homework 7 on the Web; due next Monday.

Minute Essay From Last Lecture

Slide 2

- “Something about SSDs (solid-state disks)?” Executive-level summary: Faster access time, and no moving parts (so some optimizations for spinning disks not applicable). More expensive, and limits on how many times a location can be written. So, tradeoffs for device drivers are somewhat different.
- “Something about USB?” Executive-level summary: Protocol designed to provide more-uniform access to devices, replacing serial/parallel ports. Also can provide power to device.

GUI Hardware and Software — Recap/Review

- Hardware: Keyboard and mouse send very low-level events. Display at one point was fairly low-level, but now often contains its own processors.
- Software: Framework for providing graphical interfaces may be integral to o/s (Windows) or an add-on (UNIX/Linux).

Slide 3

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. See example programs in textbook.

Slide 4

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

Other I/O-Related Topics

- “Stable storage” — use two disks to provide what appears to be a single more reliable one (i.e., write either succeeds or leaves old data in place).
- Power management significant — some devices have “sleeping” and “hibernating” states, o/s can try to determine when it would make sense to use them. Example — screen blanking.

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. Major device number used to locate specific function.
- 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

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 9

Deadlocks — Introduction

- Some resources should not be shared — among processes, computers, etc.
- To enforce this, o/s (or whatever) provides mechanism to give one process at a time exclusive use, make others wait.
- Possibility exists that others will wait forever — deadlock.

Slide 10

Resources

- "Resource" is anything that should be used by only one process at a time — hardware device, piece of information (e.g., database record), etc.

  Can be unique (e.g, particular database record) or non-unique (e.g., one block of a fixed-size disk area such as swap space).
- Preemptible versus non-preemptible — preemptible resources can be taken away from current owner without causing something to fail (e.g., memory); non-preemptible resources can't (e.g., hardware device).
- Normal sequence for using a resource — request it, use it, release it. If not available when requested, block or busy-wait.

  Can easily implement this using semaphores, but then deadlock is possible if processes aren't disciplined.

Slide 11

Deadlocks — Definitions and Conditions

- Definition — set of processes is "deadlocked" if each process in set is waiting for an event that only another process in set can cause.
- Necessary conditions:
  - Mutual exclusion — resources can be used by at most one process at a time.
  - Hold and wait — process holding one resource can request another.
  - No preemption — resources cannot be taken away but must be released.
  - Circular wait — circular chain of processes exists in which each process is waiting for resource held by next.
- Modeling deadlock — "resource graphs" (examples in textbook).
- What do about them? Various approaches.

Slide 12

What To Do About Deadlocks — Nothing

- One strategy for dealing with deadlocks — "ostrich algorithm" (ignore potential for deadlocks, hope they don't happen).
- Does this work?

Slide 13

Do Nothing, Continued

- Doesn't always work, of course.
- But simple to implement, and in practice works most of the time.

Slide 14

What To Do About Deadlocks — Detection and Recovery

- How to detect deadlocks — DFS on resource graph, (or if more than one resource of each type, algorithm from text).
- When to check for deadlocks:
  - Every time a resource is requested.
  - At regular intervals.
  - When CPU utilization falls below threshold.
- What to do if deadlock is found?
  - Preemption.
  - Rollback.
  - Process termination.
- Does this work?

Detection and Recovery, Continued

Slide 15

- Does work.
- But potentially time-consuming, and “what to do” choices aren’t very attractive!

What To Do About Deadlocks — Avoidance

Slide 16

- Can base on idea of “safe” states (in which it’s possible to schedule to avoid deadlock) versus “unsafe” states (in which it’s not). Idea is to avoid unsafe states. (Details in textbook.)
- “Banker’s algorithm” (Dijkstra, 1965) — idea is to never satisfy request for resource if it leads to unsafe state. (Details in textbook.)
- Does this work?

Slide 17

Avoidance, Continued

- Does work.
- But not much used because it assumes a fixed number of processes, resource requirements known in advance.

Slide 18

What To Do About Deadlocks — Prevention

- Idea here is to make it impossible to satisfy one of the four conditions for deadlock:
  - Mutual exclusion — don't allow more than one process to use a resource. E.g., define a printer-spool process to manage printer.
  - Hold and wait — require processes to request all resources at the same time and either get them all or wait.
  - No preemption — allow preemption.
  - Circular wait — impose strictly increasing ordering on resources, and insist that all processes request resources "in order".
- Do these work?

Prevention, Continued

Slide 19

- Don't allow more than one process to use a resource:

  Solves immediate problem but may produce others.
- Require processes to request all resources at the same time and either get them all or wait:

  Works but may not be possible or efficient.
- Allow preemption.

  Not usually possible/desirable.
- Impose strictly increasing ordering on resources, and insist that all processes request resources "in order".

  Works, but finding an ordering may be difficult.

Deadlocks — Related Issues

Slide 20

- Classical description is in terms of "resources", but other kinds of deadlock are possible (e.g., involving communication).
- Other situations that aren't classical deadlock but are also not good include "livelock" and "starvation" (see textbook).

Deadlocks — Summary

- Take-home message — there's some interesting theory related to this topic, but not a lot of practical advice, except for deadlock prevention.

Slide 21

Minute Essay

- What's the smallest number of resources needed to have a deadlock?

Slide 22

Minute Essay Answer

- Two — with only one, a process may wait a long time for another process to release it, but that's not true deadlock.

Slide 23