Slide 1

Administrivia

- Reminder: Homework 3 due Monday at class time.
- Reminder: Midterm next Wednesday. Short review sheet on the Web now. Monday's class will be review, and:
  - Solutions to written problems will be distributed in class Monday. *This means all written homework needs to be in by class time Monday.*
  - Solutions to programming problems will be on the Web around 5pm Monday. *This means all programming problems need to be submitted by 5pm Monday.*

  (Graded work should be returned soon too.)
- Local copy of (year-old) Linux kernel source in `/users/cs4320/LinuxSource`. Something to browse, for the curious?

Slide 2

One More Recap — Scheduling Algorithms

- Main idea — decide which process to run next (when running process exits, becomes blocked, or is interrupted).
- Many possibilities, ranging from simple to complex. Real systems seem to use hybrid strategies.
- How to choose one?
  - Be clear on goals.
  - Maybe evaluate some possibilities to see which one(s) meet goals — analytic or experimental evaluation.
  - Build in some tuning knobs — "separate policy from mechanism".

Slide 3

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 4

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 5

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 pp. 165-166.
- What do about them? Various approaches.

Slide 6

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 7

Do Nothing, Continued

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

Slide 8

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 of pp. 171–172).
- 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?

Slide 9

Detection and Recovery, Continued

- Does work.
- But potentially time-consuming, and "what to do" choices aren't very attractive!

Slide 10

What To Do About Deadlocks — Avoidance

- 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. See discussion p. 176.
- "Banker's algorithm" (Dijkstra, 1965) — idea is to never satisfy request for resource if it leads to unsafe state. Details on pp. 178–179.
- Does this work?

Slide 11

Avoidance, Continued

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

Slide 12

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?

Slide 13

Prevention, Continued

- 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.

Slide 14

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 15

Minute Essay

- We will use Monday's class for review. Anything you'd particularly like for me to talk about? (Or send me mail between now and then.)