Slide 1

Administrivia

- Reminder: Midterm exam Wednesday. Open (text)book, open notes. Review sheet on Web. Review session Tuesday at 4pm in HAS 228 (assuming it's available).
- Reminder: Homework 2 due tomorrow at noon. Not accepted late (except for the extra-credit programming problem).
- Homework 1 solution available today (except for programming problem). Homework 2 solution available tomorrow at noon (outside my office).

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

What Do Real Systems Use?

- Traditional Unix: two-level approach (upper level to swap processes in/out of memory, lower level for CPU scheduling), using multiple-queue scheduling for CPU scheduling. See chapter 10 for details.
- Linux: facilities for soft real-time scheduling and "timesharing" scheduling, with the latter a mix of priority and round-robin scheduling. See chapter 10 for details.
- Windows NT/2000: multiple-queue scheduling of threads, with round-robin for each queue. See chapter 11 for details.
- MVS (IBM mainframe): three-level scheme with lots of options for administrator(s) to define complex policies.

Slide 4

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 5

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 6

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 To Do About Deadlocks — Nothing

Slide 7

- One strategy for dealing with deadlocks — “ostrich algorithm” (ignore potential for deadlocks, hope they don’t happen).
- Does this work? not always, but simple to implement, and in practice works most of the time.

What To Do About Deadlocks — Detection and Recovery

Slide 8

- 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? yes, but potentially time-consuming, and “what to do” choices aren’t very attractive!

Slide 9

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? yes, but not much used because it assumes a fixed number of processes, resource requirements known in advance.

Slide 10

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.

  Solves immediate problem but may produce others.
- Hold and wait — 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.
- No preemption — allow preemption. Not usually possible/desirable.
- Circular wait — impose strictly increasing ordering on resources, and insist that all processes request resources "in order".

  Works, but finding an ordering may be difficult.

Slide 11

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 12

Minute Essay

- Anything you'd particularly like for me to talk about at the review session? (Or send me mail between now and then.)