Slide 1

Administrivia

- Reminder: Homework 2 due Tuesday.
- Reminder: Midterm Thursday (3/05). (Unless we reschedule — but there's no consensus on that. If you feel strongly about it one way or another and have not told me so, do so by 5pm. I will make a decision and announce it by e-mail.)

  Review sheet to be on the Web soon. (Short version: Open book/notes like quizzes; topics from chapters 1 and 2.)

  Homework sample solutions to be available no later than next Tuesday.

Slide 2

Minute Essay From Last Lecture

- Many answers, some of which I'm not sure I completely understand!
- Clarification: A replacement DLL / shared library should have all the (documented?) functions from the old version, with the same interface (names and parameters and return types).
- The key point is of course that newer is not always better in all respects . . .

Slide 3

Linking — Example

- (Work through example starting on p. 127. Notice that we also need information about locations of “text segment” (code) and “data segment” (variables).)

Slide 4

Sidebar: Parallel Execution and Synchronization

- A lot of commodity hardware these days features multiple processing units (“cores”) sharing access to memory. One reason for this is that in theory we can make individual applications faster by splitting computation up among processing elements.
- Having processing elements share memory makes parallel programming easier in some ways but has risks (“race conditions”). Avoiding the risks requires some way to control access to shared variables (e.g., to implement notion of “lock”).

Slide 5

Parallel Execution and Synchronization, Continued

- Most texts on operating systems discuss synchronization issues and present several solutions ("synchronization mechanisms"), some rather high-level and others not.

  (Why is this in o/s textbooks?)

- The most primitive can (with some simplifying assumptions) be implemented with no hardware support. But hardware support is very useful.

Slide 6

Instructions for Synchronization

- Key goal in designing hardware support for synchronization is to provide "atomic" (indivisible) load-and-store. This allows writing a low-level implementation of "lock" idea.
- Many architectures do this with a single instruction (e.g., "test and set" or "compare and swap"). Requires two accesses to memory so may be difficult to implement efficiently.
- MIPS approach — same idea, but using a pair of instructions, `ll` ("load linked") and `sc` ("store conditional"). Example of use in textbook (p. 122). `sc` "succeeds" only if value at target location has not changed since previous `ll` — i.e., if one can regard the pair of instructions as forming a single atomic load/store.

Slide 7

Minute Essay

- None — quiz.