Slide 1

Administrivia

- Reminder: If you haven't watched the recorded lecture for last week, do so as soon as you can.
- Reminder: Reading Quiz 2 due Monday.

  Everyone turned something in for Reading Quiz 1. Yay!!
- First homework coming soon, I hope before Monday. I'll send mail when it's ready, but allow you at least a week to actually work on it.

Slide 2

Minute Essay From Last Lecture

- One person asked about my notes — I intend to post them before class, but apparently I had forgotten. Okay to say something in class!

Slide 3

Limited Direct Execution — Recap

- First an unrelated bit of usage/terminology: I'll use O/S to mean operating system henceforth!
- "Direct execution" means as opposed to, say, emulation.
- "Limited" in that we want to do *something* to ensure that the O/S can "defend itself" and also protect processes from each other.

  Note that this is only even possible if hardware provides some support. May explain why early O/S's for PCs didn't always provide this kind of safety.

Slide 4

Limited Direct Execution — Review/Clarification

- Textbook figure 6.2 very helpful, but I'm skeptical of some details — seems to indicate that O/S always returns to caller after system call, but clearly that can't be true if sometimes it terminates the process!
- "Trap table" is a name I had not encountered before, and I wonder about the name. Could it be specific to x86?
- More broadly: System calls (whatever the name is — trap, MIPS `syscall`, etc.) are a type of interrupt. There are other kinds of interrupts. Part of the interaction between the O/S and the hardware is the address(es) of handlers for various kinds of interrupts — possibly only one, or possibly different ones for different interrupts. Details might vary among architectures, but in general, textbook is (as far as I know) right that the O/S has to set this up at boot time — if nothing else, put its code at the hardware-specified fixed address.

Slide 5

Limited Direct Execution — Review/Clarification, Continued

- Textbook figure 6.3 also helpful, though initially I was somewhat skeptical about details being applicable to all O/S's and architectures.

  However, on reflection it makes sense to talk about two separate save/restore operations:

  If the scheduler says "keep running the interrupted process" then there's only a need to restore any machine state the interrupt handler messed up. If it says "switch processes" then more may be needed.

- In any case, in my usage (and I think this is standard), "context switch" usually refers to what happens when the O/S switches from one process to another.

Slide 6

Aside: This and That

- Sometimes I find myself wondering whether an advantage to being self-published is that the authors feel more free to add more humor and opinions?
- I was amused by "Very Bad Idea" and the comments on rebooting. (Elsewhere I've heard Linux called "the operating system that makes you feel guilty about rebooting".)
- Even the bibliographies aren't all-business!

Slide 7

“What About Concurrency?” Indeed

- Last section in the chapter on limited direct execution is well-titled. Very complicated topic, and not a bad idea to address it later and not as part of a discussion of “process management”, as some textbooks do.
- For now worth mentioning that indeed the question of what happens in the case of nested(?) interrupts is interesting. Two points:
- They complicate things enormously (like recursion and how it can overflow the stack).
- Could fix that by disabling further interrupts until we finish one. But that might mean missing one.

  So what we do while interrupts disabled should be short!
- Also, as I understand things, some things that might generate interrupts will keep trying until they get an “acknowledged” response.

Slide 8

Concurrency — One More Thing

- A key problem — how to ensure that a sequence of actions happens, or appears to happen, as one “atomic” thing — i.e., without interference from anything else.
- This is what the textbook was getting at in talking about “atomically” — but I found their explanation unclear and possibly misleading.
- From an application programmer’s point of view, not guaranteeing atomicity of a sequence of operations can lead to race conditions, which can be solved via “locks”.
- Actually implementing locks is not as easy as it might sound! (Later.)

Slide 9

Scheduling — Overview

- Textbook likes to distinguish between "mechanisms" and "policies". (And it *is* usually a good thing to separate them.)
- Previous chapters in this part have been about mechanisms for virtualizing the CPU. Some (which process to run next) require decision-making — i.e.(?), involve policies.
- (Textbook says the question of scheduling has a history before computers. Interesting!)
- Many policies possible; many have been tried over the years.

Slide 10

Scheduling — Simple View

- Scheduling algorithm (textbook calls them disciplines) usually based on "jobs" (units of work — name goes back to batch systems, where users submitted "jobs" to system operator, and there was no notion of interactive users.
- Textbook lays out some simplifying assumptions. I say we can start with slightly less restrictive ones:
  - Each job arrives at some predefined time.
  - Each job runs for some fixed predefined amoun of time.
  - Once started, a job runs to completion (so, no switching back and forth among processes).
  - Jobs use only the CPU (i.e., no I/O).

Slide 11

Scheduling Algorithms — Metrics

- Useful to be able to compare different algorithms/policies/disciplines.
- A simple one is based on “turnaround time” (completion time minus arrival time) for jobs. Average turnaround time could be a simple metric for comparing.
- (To be continued.)

Slide 12

Minute Essay

- Are you finding the reading quizzes useful or interesting?
- I said I wanted you to do them all individually, but I’m having second thoughts. Do you think you’d benefit from being allowed to discuss them with classmates? I worry that what might happen then is not everyone reads everything.