

1



- 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?
- Slide 3
- 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.

## Limited Direct Execution — Review/Clarification, Continued

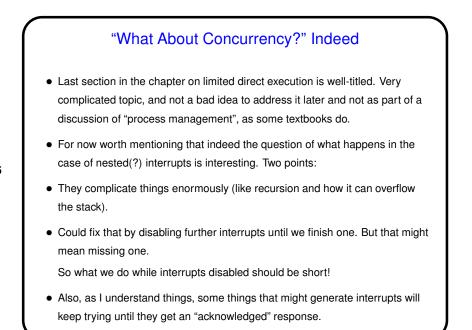
 Textbook figure 6.3 also helpful, though initially I was somewhat skeptical about details being applicable to all O/S's and architectures.
 However, on reflection it makes sense to talk about two separate save/restore operations:

Slide 4

a need to restore any machine state the interrupt handler messed up. If it says "switch processes" then more may be needed.

If the scheduler says "keep running the interrupted process" then there's only

• 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.

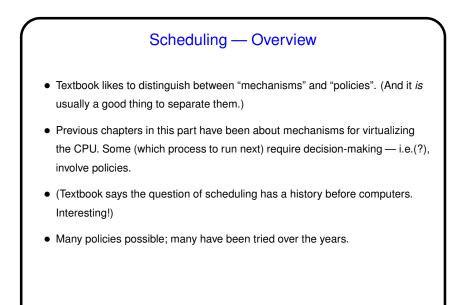


## Concurrency — One More Thing

- 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 5

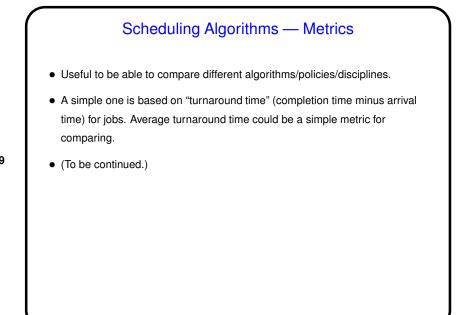
Slide 6



Scheduling — Simple View
Scheduling algorithms (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).
Slide 8
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 amount 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 7



Slide 9

