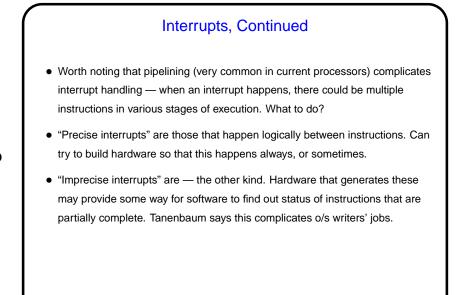
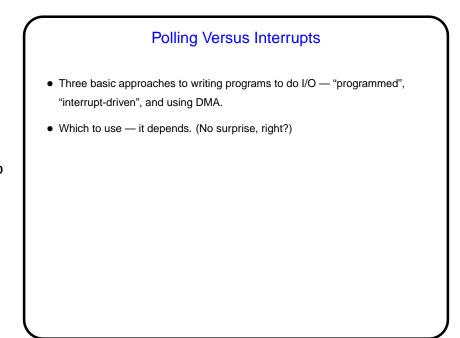
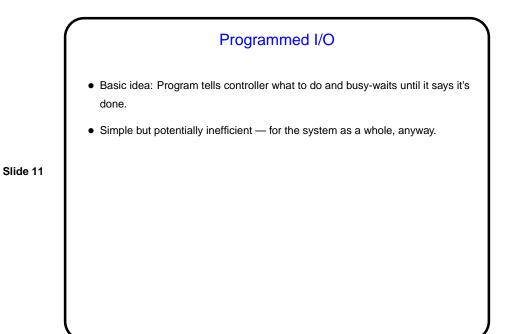


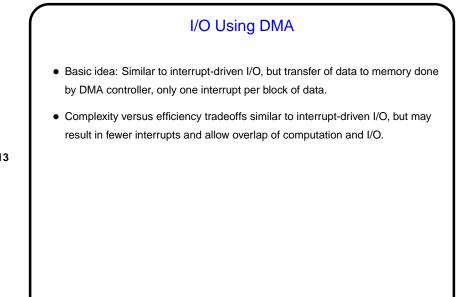
## Interrupts, Continued On interrupt, hardware locates proper interrupt handler (probably using interrupt vector), saves critical info such as program counter, and transfers control (switching into supervisor/kernel mode). Interrupt handler saves other info needed to restart interrupted process, tells interrupt controller when another interrupt can be handled, and performs minimal processing of interrupt.

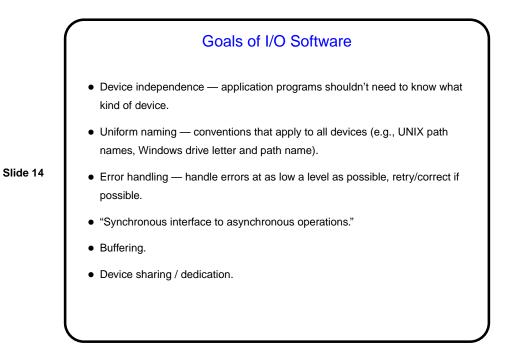






Interrupt-Driven I/O
Basic idea: Program tells controller what to do and then blocks. While it's blocked, other processes run. When requested operation is done, controller generates interrupt. Interrupt handler unblocks original program (which, on a read operation, would then obtain data from device controller).
Slide 12
More complex, but allows other processing to happen while waiting, so potentially more efficient for system as a whole. Could, however, result in lots of interrupts. (Tanenbaum says one per character/byte. Can that be true for disks?? Open question ...)





## Minute Essay

 We talked about two approaches to communicating with I/O devices special I/O instructions, and "memory-mapped I/O" (reading/writing particular memory locations). What implications do you think the two choices have for programmers' ability to write device drivers in a (moderately) high-level language such as C?

Slide 15

Slide 16

## Minute Essay Answer • With memory-mapped I/O it should be possible to write device drivers entirely in C; with special I/O instructions this would not be possible without compiler modifications or some amount of assembly-language code.