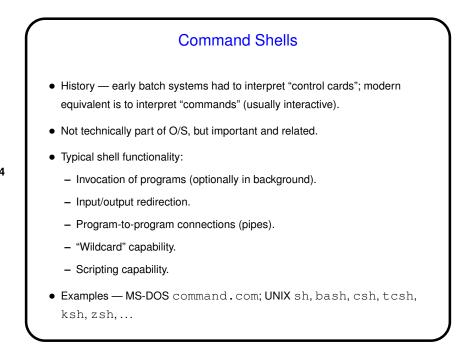


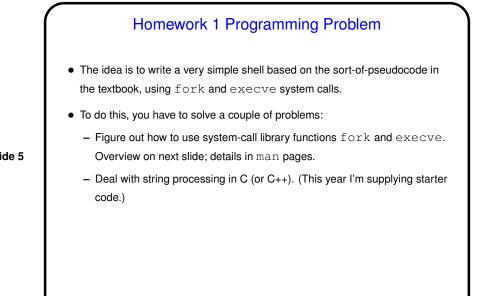


• Recall(?) typical mechanism for regular program calls: Put parameters in agreed-on locations (registers, stack, etc.), issue instruction that saves current program counter (in another register maybe) and transfers control to called program. Called program returns using saved program counter.

Slide 3

• System calls are similar *except* that the "called program" is at a fixed address *and* the transfer of control also puts the processor in supervisor/kernel mode.



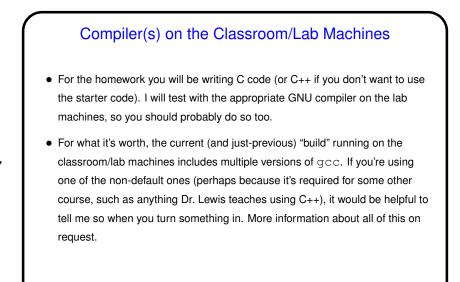


## Homework 1 Programming Problem, Continued

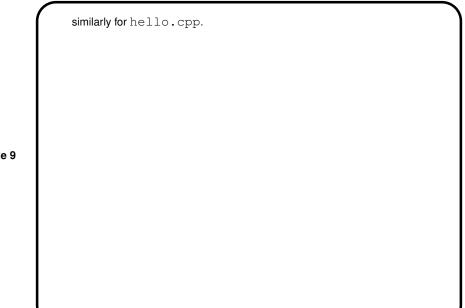
• fork () function creates and starts a new process. Both original ("parent") and new ("child") processes execute the same program, continuing at whatever follows call to  ${\tt fork}$  ( ) . Return value from function says which process is which.

Slide 6

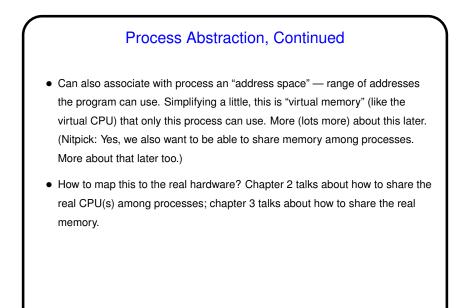
 $\bullet\,\, \texttt{execve}$  ( ) function discards current program and loads and starts a new one. If it fails, execution continues with whatever follows; otherwise whatever follows is ignored!

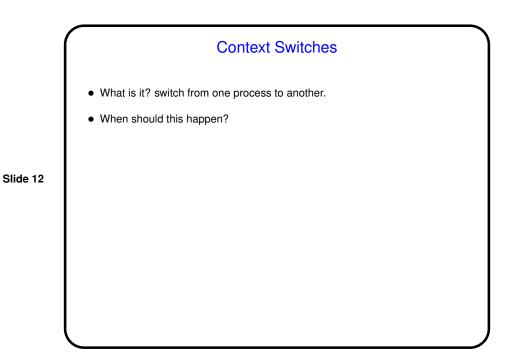


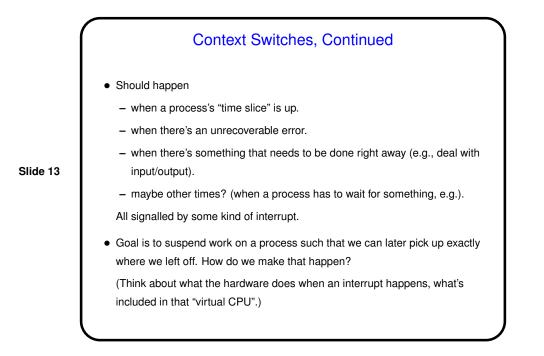
Slide 8	Sidebar: C/C++ Programming Advice
	<ul> <li>I strongly recommend always compiling with flags to get extra warnings. There are lots of them, but you can get a lot of mileage just from -Wall. Add -pedantic to flag nonstandard usage. Warnings are often a sign that something is wrong. Sometimes the problem is</li> </ul>
	a missing #include. man pages tell you if you need one.
	<ul> <li>If you want to write "new" C (including C++-style comments), you may need to add -std=c99.</li> </ul>
	<ul> <li>If typing all of these gets tedious, consider using a simple makefile. Create a file called Makefile containing the following (the first line for C, the second for C++):</li> </ul>
	CFLAGS = -Wall
	CXXFLAGS = -Wall
	and then compile hello.c to hello by typing make hello, or

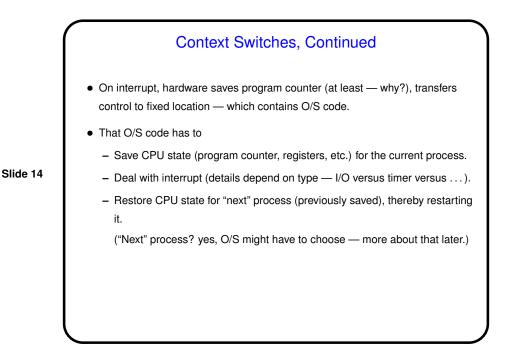


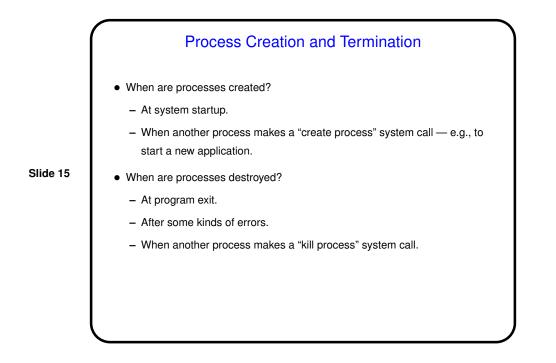
# Process Abstraction We want O/S to manage "things happening at the same time" — applications, hidden tasks such as managing a device, etc. Key abstraction for this — "process" — program plus associated data, including program counter. Slide 10 True concurrency ("at the same time") requires more than one CPU/processor/core. Can get apparent concurrency via interleaving — model one virtual CPU per process and have the real processor switch back and forth among them ("context switch"). (Aside: In almost all respects, this turns out to be indistinguishable from true concurrency. "Hm!"?)

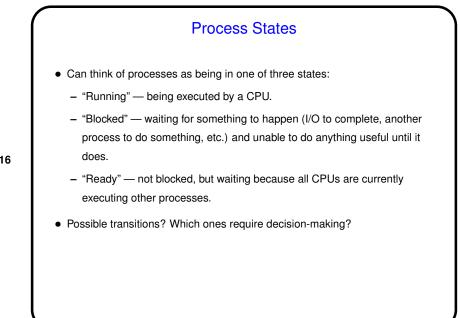


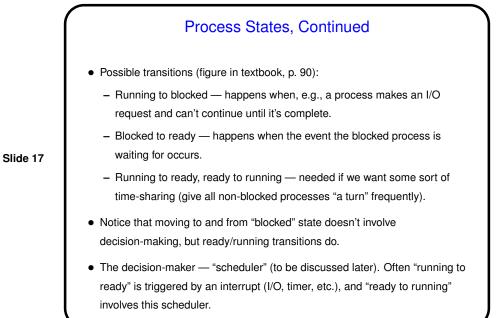


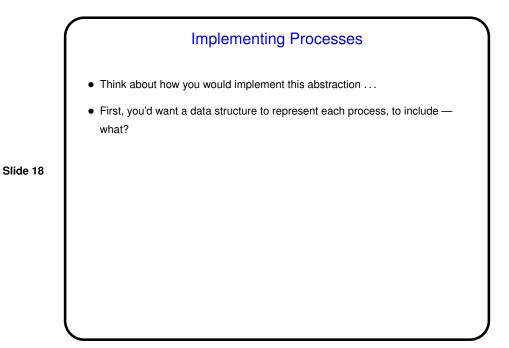


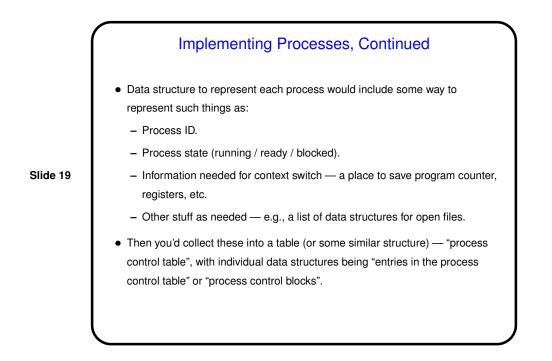


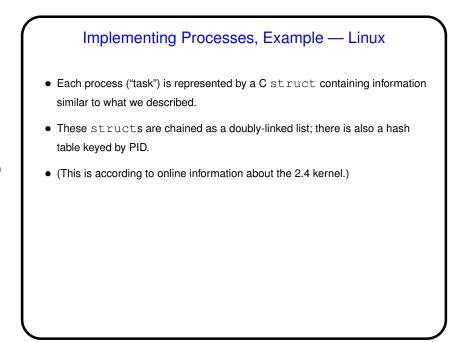


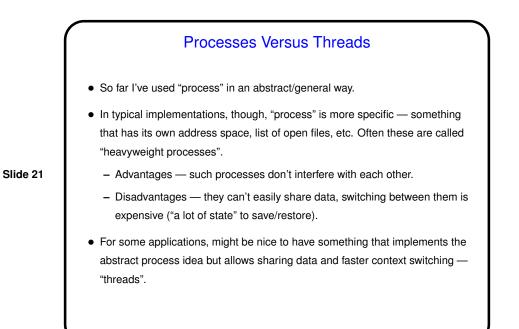


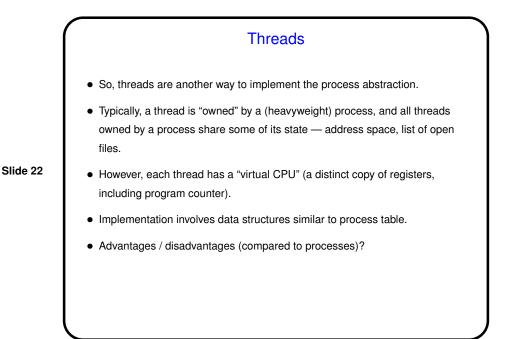


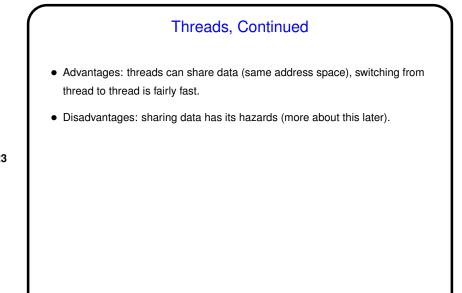




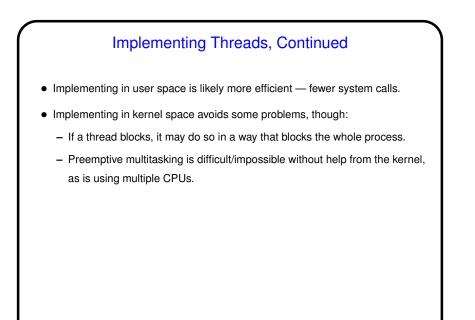








# Implementing Threads Two basic approaches — "in user space" and "in kernel space" Various hybrid schemes also possible. Basic idea of "in user space" — operating system thinks it's managing single-threaded processes, all the work of managing multiple threads happens via library calls within each process. Basic idea of "in kernel space" — operating system is involved in managing threads, the work of managing multiple threads happens via system calls (rather than user-level library calls). How do they compare?...



### Adding Multithreading

- If you've written multithreaded applications moving from single-threaded to multithreaded not trivial:
  - Figure out how to split up computation among threads.
  - Coordinate threads' actions (including dealing properly with shared variables).
- Similar problems in adding multithreading to systems-level programs:
  - Deal properly with shared variables (including ones that may be hidden).
  - Deal properly with signals/interrupts.

