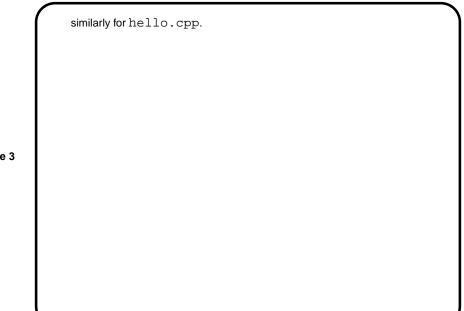
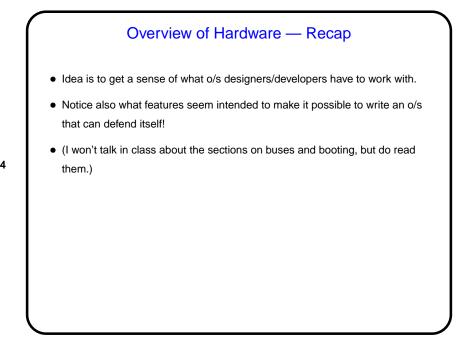
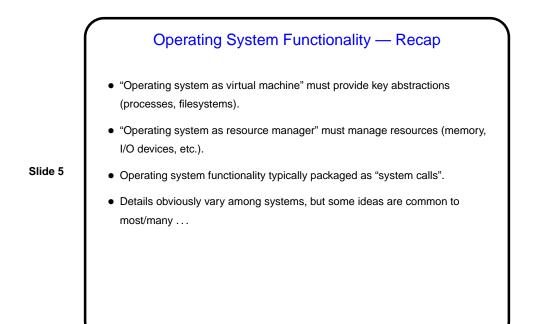


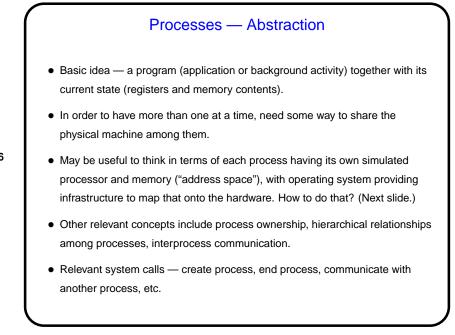
Sidebar: C/C++ Programming Advice
 I strongly recommend always compiling with flags to get extra warnings. There are lots of them, but you can get a lot of mileage just from -Wall. Add -pedantic to flag nonstandard usage.
Warnings are often a sign that something is wrong. Sometimes the problem is a missing #include. man pages tell you if you need one.
 If you want to write "new" C (including C++-style comments), add -std=c99.
 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++):
CFLAGS = -Wall
CXXFLAGS = -Wall
and then compile hello.c to hello by typing make hello, or





Slide 4





Processes — Implementation

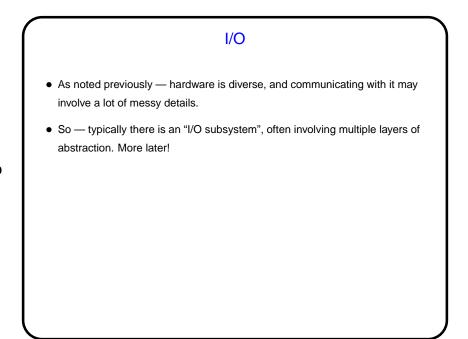
 Managing the "simulated processor" aspect requires some way to timeshare physical processor(s). Typically do that by defining a per-process data structure that can save information about process. Collection of these is a "process table", and each one is a "process table entry".

Slide 7

Slide

 Managing the "address space" aspect requires some way to partition physical memory among processes. To get a system that can defend itself (and keep applications from stepping on each other), memory protection is needed probably via hardware assist. Some notion of address translation may also be useful, as may a mechanism for using RAM as a cache for the most active parts of address space, with other parts kept on disk.

Filesystems
 Most common systems are hierarchical, with notions of "files" and "folders"/"directories" forming a tree. "Links"/"shortcuts" give the potential for a more general (non-tree) graph.
 Connecting application programs with files — notions of "opening" a file (yielding a data structure programs can use, usually by way of library functions).
 Many, many associated concepts — ownership, permissions, access methods (simple sequence of bytes, or something more complex?), whether/how to include direct access to I/O devices in the scheme.
• Relevant system calls — create file, create directory, remove file, open, close, etc., etc.
• See text for some UNIXisms — single hierarchy, regular versus special files, pipes, etc.



Slide 10

Hardware, Software, and History Textbook has a section called "Ontogeny Recapitulates Phylogeny". Many interesting general observations: What seems like a good idea in software is strongly influenced by what the hardware can do. (I think it goes the other way too, but that's speculation.) As in other areas of human endeavor, evolution of operating systems is in some ways cyclic: What seems brilliant now may be "ready for the scrap heap" in a few years — and then resurface as brilliant later. (This is why it's not useless to read about approaches not currently in use.)

