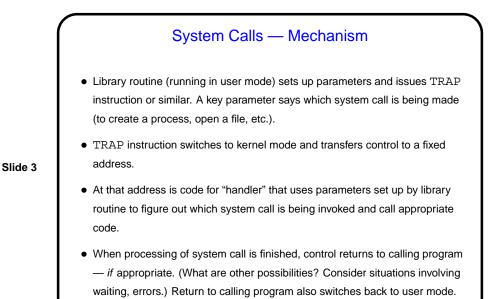


System Calls
Recall that some things can/should only be done by o/s (e.g., I/O), and hardware can help enforce that.
But application programs need to be able to request these services. How can we make this work? System calls ...

Slide 4



System Calls — Services Provided
Typical services provided include creating processes, creating files and directories, etc., etc. — details depend on (and in some ways define, from application programmer's perspective) operating system.
Examples discussed in textbook:
POSIX (Portable Operating System Interface (for UNIX)) — about 100 calls.
Win32 API (Windows 32-bit Application Program Interface) — thousands of calls.
Worth noting that the actual number of system calls is likely smaller — interface may contain function calls that are implemented completely in user space (no TRAP to kernel space).

Interrupts
Processing of TRAP instructions is similar to interrupts, so worth mentioning here:
Very useful to have a way to interrupt current processing when an unexpected or don't-know-when event happens — error occurs (e.g., invalid operation), I/O operation completes.
On interrupt, goal is to save enough of current state to allow us to restart current activity later:

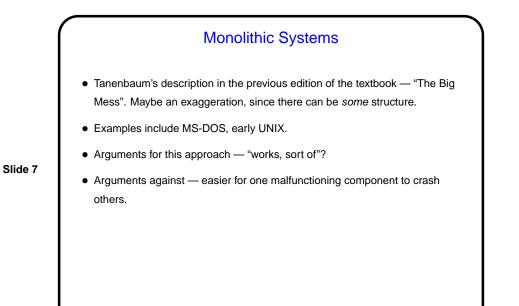
Save old value of program counter.
Disable interrupts.
Transfer control to fixed location ("interrupt handler" or "interrupt vector") — normally o/s code that saves other registers, re-enables interrupts, decides what to do next, etc.

Slide 5

Operating System Structures

• Clearly o/s could involve a whole lot of code (e.g., second edition of textbook says 29M lines of code for Windows 2000). How to structure?

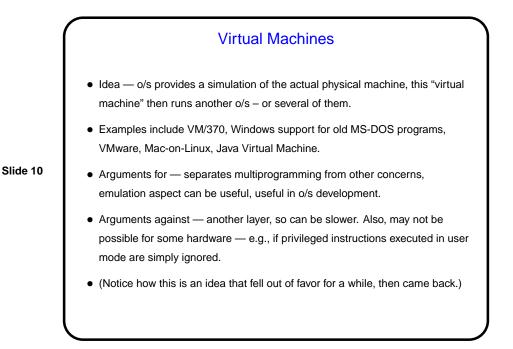
- Choices include:
 - Monolithic systems.
 - Layered systems.
 - Microkernels.
 - Client-server model.
 - Virtual machines.
 - Exokernels.

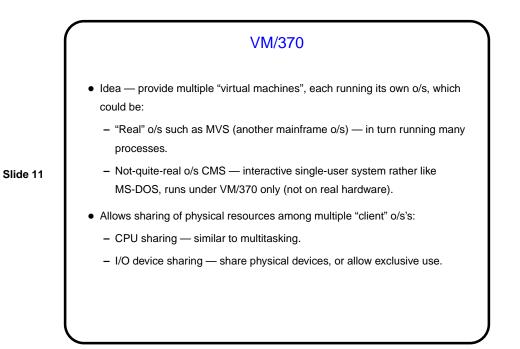


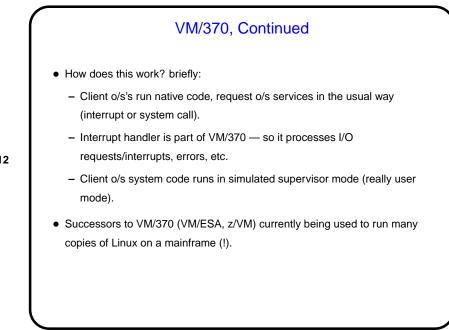
Layered Systems
Idea — use layers of abstraction, just as one structures application programs.
Examples include THE, MULTICS, OS/2, Windows NT (more so in early releases).
Arguments for — nice separation of concerns, modularity.
Arguments against — tricky to plan layers, performance can be slow.

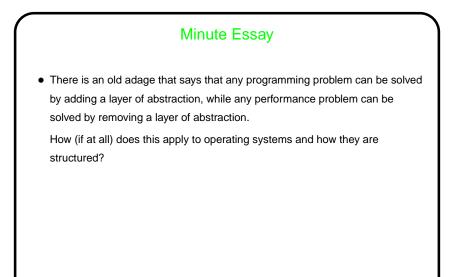


- Idea make kernel itself as small as possible, package other services separately, as independent processes.
- Examples include MINIX (written by Tanenbaum).
- Arguments for modularity, reliability.
- Slide 9
- Arguments against tricky to plan layers, performance might be reduced.









Slide 13

Minute Essay Answer

 Based on the descriptions of the various operating-system structures, it looks like the general principle applies here too — adding layers of abstraction can improve correctness and reliability, but there is likely to be a performance cost.