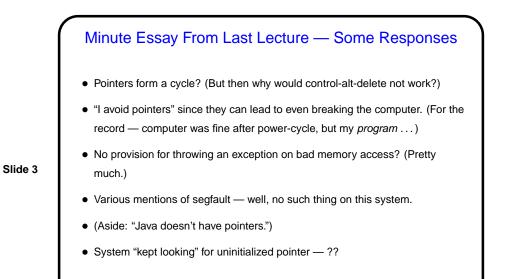


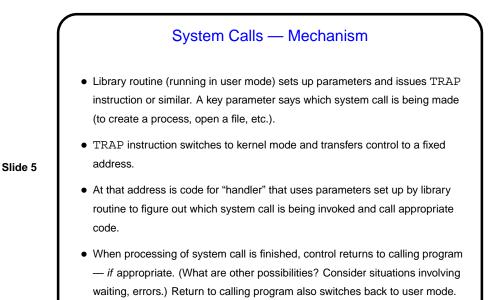
Minute Essay From Last Lecture

- Many people did figure out that the problem was related to using an unitialized pointer, but beyond that a lot of variation, and some confusion about what pointers are. (In context basically memory addresses.)
- Slide 2
- Key point is that MS-DOS didn't protect its own memory, so my little application program could (and presumably did) overwrite something important in the o/s's memory. Symptoms suggest that "something important" here was something related to processing keyboard input.

(The story may be badly titled, since it's not clear what's at fault — the hardware for not providing memory protection or MS-DOS for not using it. Either way it illustrates the risk of not having and using memory protection?)



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 ...



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:

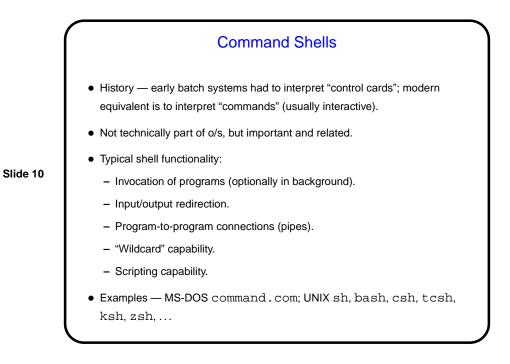
Save old value of program counter.
Disable interrupts.
Transfer control to fixed location ("interrupt handler" or "interrupt vector") — normally o/s code that saves other registers, re-enables interrupts, decides

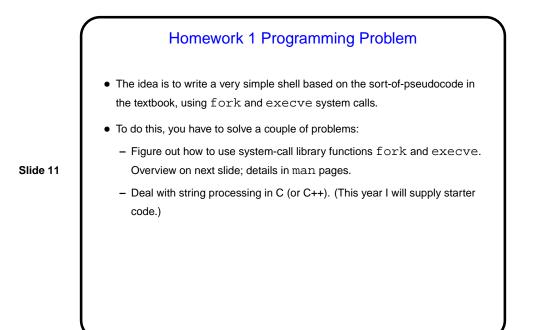
what to do next, etc.

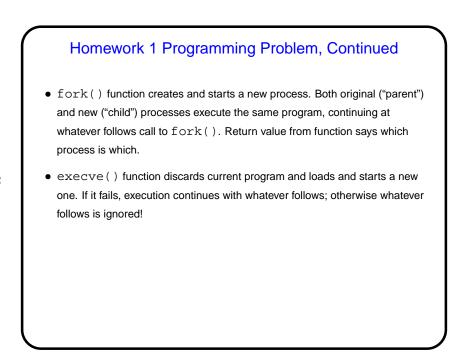
Slide 7

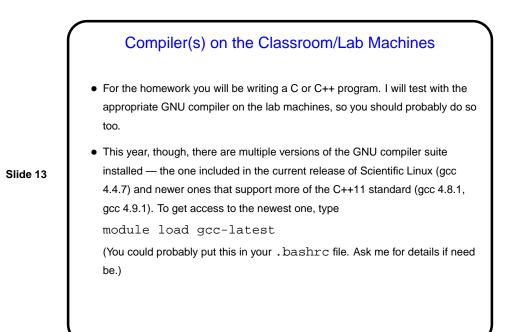
Slide 8
 MIPS instruction set includes syscall instruction that generate a system-call exception. MIPS interrupts/exceptions use special-purpose registers to hold type of exception and address of instruction causing exception. Before issuing syscall program puts value indicating which service it wants in register \$v0. Parameters for system call are in other registers (can be different ones for different calls).
 Interrupt handler for system calls looks at \$v0 to figure out what service is requested, other registers for other parameters.
 When done, it uses rfe instruction to restore calling program's environment, then returns to caller using value from EPC register.

```
Example: System Calls in MIPS/SPIM
• SPIM simulator — a primitive o/s! — defines a short list of system calls.
Example code fragment:
    la $a0, hello
    li $v0, 4 # "print string" syscall
    syscall
    ....
    .data
hello: .asciiz "hello, world!\n";
```









| | Sidebar: C/C++ Programming Advice |
|------|--|
| e 14 | 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 |

