Today

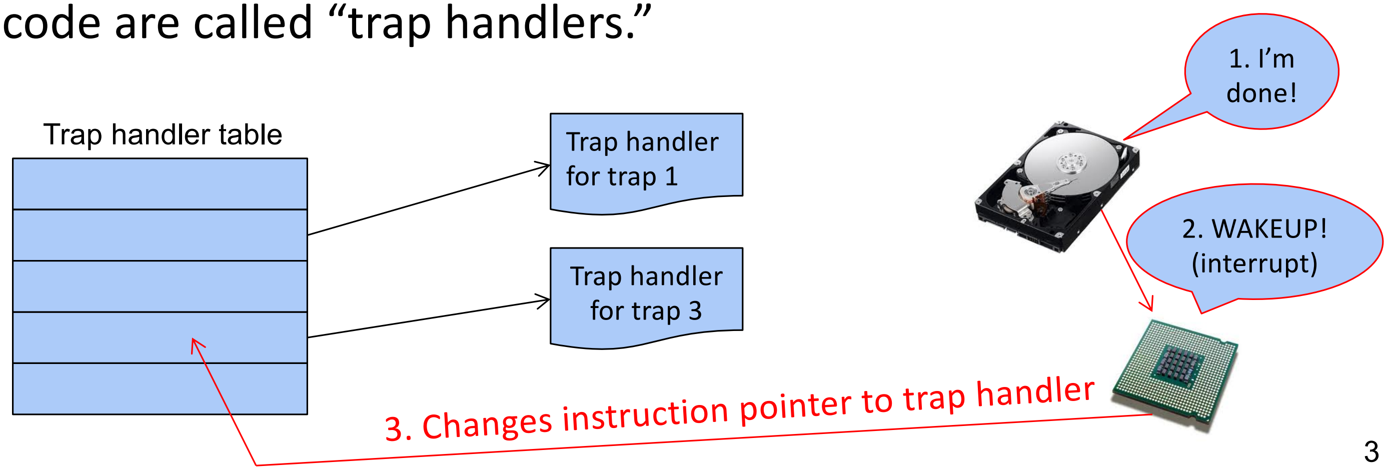
- Learning Outcomes
  - Describe how control transfers from an application to the operating system.
  - Distinguish synchronous and asynchronous control transfers
  - Categorize transfers as one of:
    - system call
    - exception
    - interrupt
- Reading
  - Chapter 8: 8.1-8.3

Control Transfer

- Regardless of why and when control must transfer to the operating system, the mechanism is the same.
- First, we'll talk about what must happen in the abstract (i.e., not in the context of any particular processor).
- Then, we'll step talk about the x86 transfer control mechanism specifically.
- Key points:
  - We can invoke the operating system explicitly via a system call.
  - The operating system can be invoked implicitly via an exception (sometimes called a software interrupt), such as a divide by zero, or a bad memory reference.
  - The operating system can be invoked asynchronously via (hardware) interrupts, such as a timer, an I/O device, etc.

Trap Handling: In the abstract

- Each type of trap is assigned a number. For example:
  - 1 = system call
  - 2 = timer interrupt
  - 3 = disk interrupt
  - 4 = interprocessor interrupt
- At startup, the operating system sets up a table, indexed by trap number, that contains the address of the code to be executed whenever that kind of trap happens.
- These pieces of code are called “trap handlers.”



X86 Trap Handling

- Interrupt Descriptor Registers (IDT): Plays the role of the trap handler table
  - Contains special objects called gates.
  - Gates provide access from lower privileged segments to higher privileged segments.
    - When a low-privilege segment invokes a gate, it automatically raises the CPL (current privilege level) to the higher level.
    - When returning from a gate, the CPL drops to its original level.
  - First 32 gates reserved for hardware defined traps.
  - Remaining entries are available to software using the INT (interrupt) instruction.
- Hardware register traditionally called PIC (Programmable Interrupt Controller), then APIC (advanced PIC) and most recently LAPIC (local advanced PIC, one per CPU in the system)
  - Has wires to up to 16 devices
  - Maps wires to particular locations in IDT.
  - PIC sends the appropriate value for the interrupt handler dispatch to the processor.

x86 System Calls

- There are multiple ways to handle system calls and different operating systems use different ways (these are all assembly instructions):
  - Old Linux systems use a single designated INT instruction (triggers a software interrupt) and then dispatches again within a single handler.
    - Return from privileged to unprivileged via IRET.
  - Modern Linux systems use the SYSENTER/SYSEXIT calls.
  - Depending on the processor on which you are running (Intel or AMD) and other details, sometimes Linux uses SYSCALL/SYSRET.
  - (You don’t need to know the details here, just that there are special instructions that let you invoke privileged code.)

Recap

- The operating system is just a bunch of code that sits around waiting for something to do (e.g., help out a user process, respond to a hardware device, process a timer interrupt, etc).
- The operating system runs in privileged (or supervisor) mode.
- Hardware provides some sort of mechanism to transfer control from one privilege level to another.
- We use the term trap to refer to any mechanism that transfers control into the operating system.
- There are different kinds of traps:
  - System calls: intentional requests of the operating system on behalf of a program; synchronous with respect to the program)
  - Exceptions (software interrupts; synchronous with respect to programs)
  - Interrupts (caused by hardware; asynchronous)