Interrupts & Exceptions

Required reading: xv6 `trapasm.S`, `trap.c`, `syscall.c`, `usys.S`.
You will need to consult IA32 System Programming Guide chapter 5 (skip 5.7.1, 5.8.2, 5.12.2).

Overview

Big picture: kernel is trusted third-party that runs the machine. Only the kernel can execute privileged instructions (e.g., changing MMU state). The processor enforces this protection through the ring bits in the code segment. If a user application needs to carry out a privileged operation or other kernel-only service, it must ask the kernel nicely. How can a user program change to the kernel address space? How can the kernel transfer to a user address space? What happens when a device attached to the computer needs attention? These are the topics for today's lecture.

There are three kinds of events that must be handled by the kernel, not user programs: (1) a system call invoked by a user program, (2) an illegal instruction or other kind of bad processor state (memory fault, etc.). and (3) an interrupt from a hardware device.

Although these three events are different, they all use the same mechanism to transfer control to the kernel. This mechanism consists of three steps that execute as one atomic unit. (a) change the processor to kernel mode; (b) save the old processor somewhere (usually the kernel stack); and (c) change the processor state to the values set up as the “official kernel entry values.” The exact implementation of this mechanism differs from processor to processor, but the idea is the same.

We'll work through examples of these today in lecture. You'll see all three in great detail in the labs as well.

A note on terminology: sometimes we'll use interrupt (or trap) to mean both interrupts and exceptions.

Setting up traps on the x86

See handout Table 5-1, Figure 5-1, Figure 5-2.

xv6 Sheet 07: `struct gatedesc` and `SETGATE`.

xv6 Sheet 28: `tvinit` and `idtinit`. Note setting of gate for `T_SYSCALL`

xv6 Sheet 29: `vectors.pl` (also see generated `vectors.S`).

System calls

xv6 Sheet 16: `init.c` calls `open("console")`. How is that implemented?

xv6 `usys.S` (not in book). (No saving of registers. Why?)

Breakpoint `0x1b:"open"`, step past `int` instruction into kernel.

See handout Figure 9-4 [sic].

xv6 Sheet 28: in `vectors.S` briefly, then in `alltraps`. Step through to `call trap`, examine registers and stack. How will the kernel find the argument to `open`?

xv6 Sheet 29: `trap`, on to `syscall`.

xv6 Sheet 31: `syscall` looks at `eax`, calls `sys_open`.

(Briefly) xv6 Sheet 52: `sys_open` uses `argstr` and `argint` to get its arguments. How do they work?

xv6 Sheet 30: `fetchint`, `fetcharg`, `argint`, `argptr`, `argstr`.

What happens if a user program divides by zero or accesses unmapped memory? Exception. Same path as system call until `trap`.

What happens if kernel divides by zero or accesses unmapped memory?

Interrupts

Like system calls, except: devices generate them at any time, there are no arguments in CPU registers, nothing to return to, usually can't ignore them.

How do they get generated? Device essentially phones up the interrupt controller and asks to talk to the CPU. Interrupt controller then buzzes the CPU and tells it, “keyboard on line 1.” Interrupt controller is essentially the CPU's ~~secretary~~ administrative assistant, managing the phone lines on the CPU's behalf.

Have to set up interrupt controller.

(Briefly) xv6 Sheet 63: `pic_init` sets up the interrupt controller, `irq_enable` tells the interrupt controller to let the given interrupt through.

(Briefly) xv6 Sheet 68: `pit8253_init` sets up the clock chip, telling it to interrupt on `IRQ_TIMER` 100 times/second. `console_init` sets up the keyboard, enabling `IRQ_KBD`.

In Bochs, set breakpoint at 0x8:"vector0" and continue, loading kernel. Step through clock interrupt, look at stack, registers.

Was the processor executing in kernel or user mode at the time of the clock interrupt? Why? (Have any user-space instructions executed at all?)

Can the kernel get an interrupt at any time? Why or why not? `cli` and `sti`, `irq_enable`.