Exceptional control flow

Exceptions:

Interrupts -> asynchronous

Traps -> synchronous

↳ depend on program

fault

~~Trap~~ examples:

- page faults
- divide by 0
- protection faults
- the int instruction

trap

What does the system need to do on an exception?

- Save return address
- Save registers (and other state)

architectural

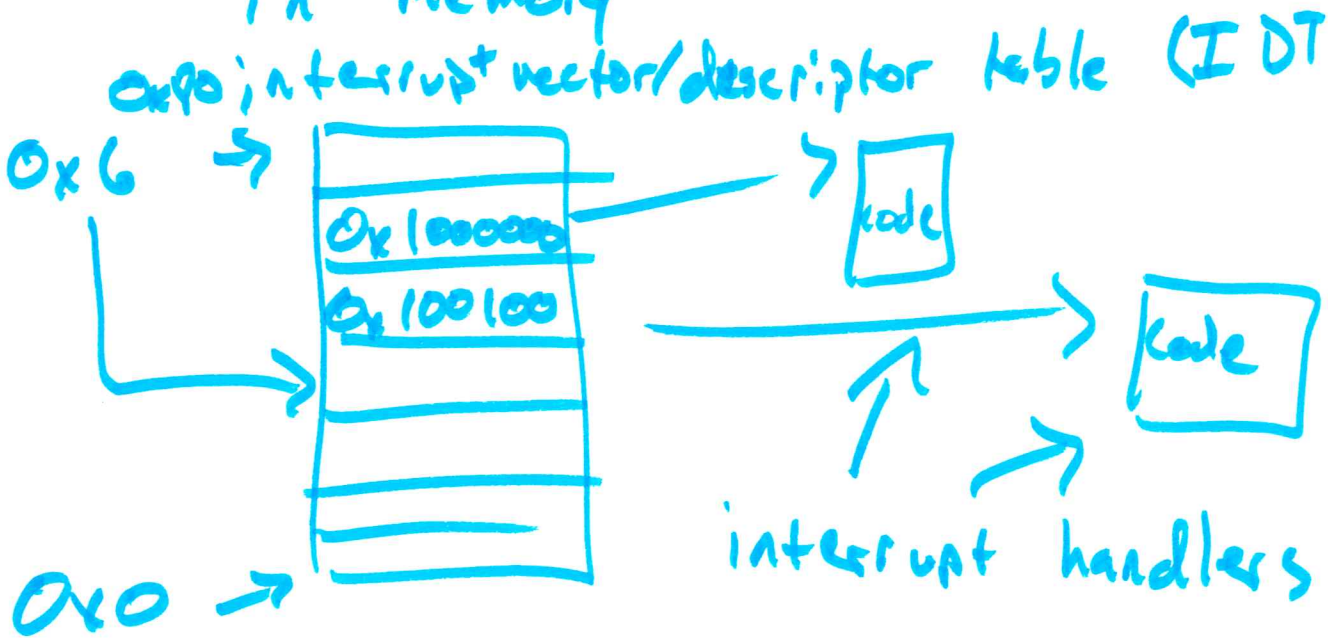
★ - jump to the handler

- Set the reason for exception
  - ↳ privilege level
  - ↳ set registers (%esp

- Done executing: system needs to fix up everything

How do we know where to jump?

- interrupt vectors defined somewhere in memory



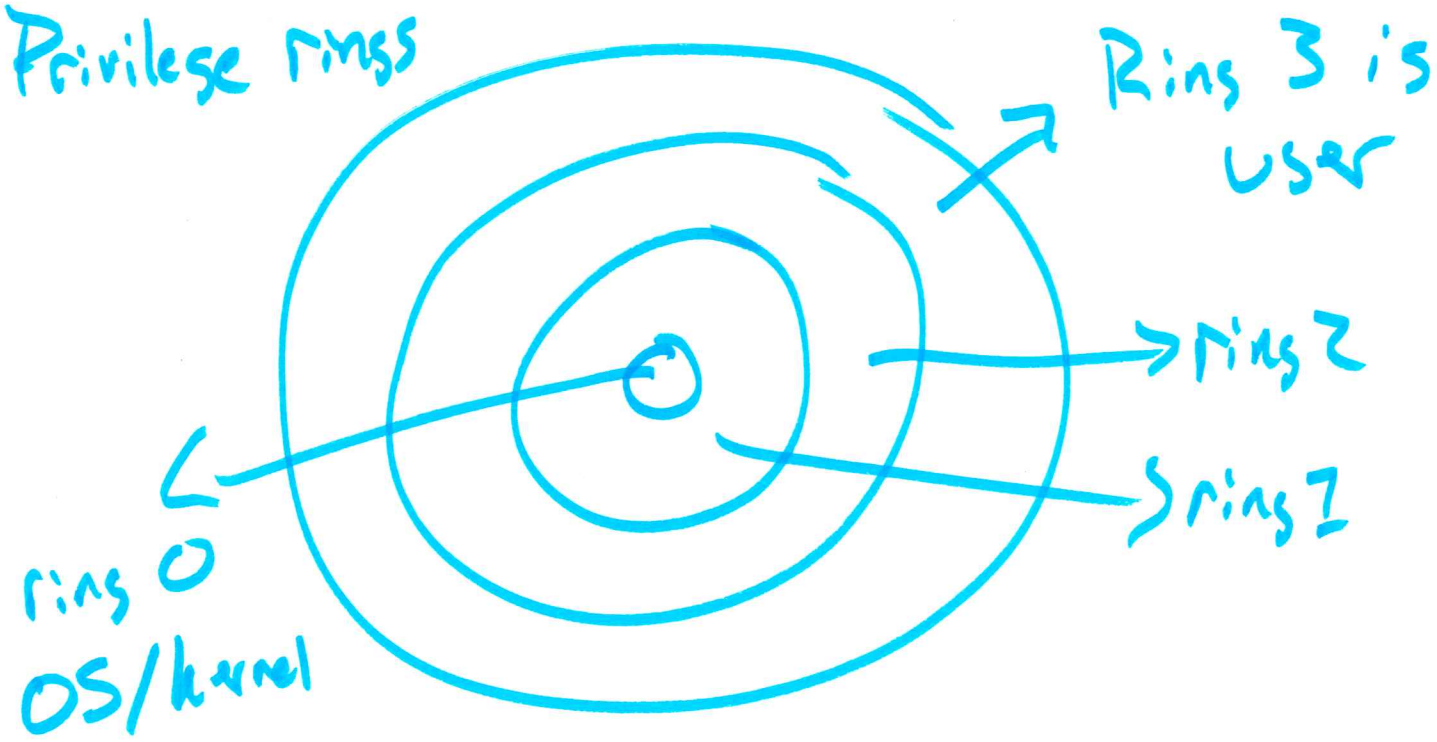
How is table initialized?

→ The OS: on boot OS points to handlers by writing the IDT

Set privileges:

user-mode

kernel-mode (ring 0)



prevents users from:

- executing certain instructions
- modifying certain registers
- messing w/ interrupt vectors
- and lots more

need to be in Ring 0 to run int. handler

Asking the OS to do something.

read(fd, buf, 100);

System call (syscall)

in x86:

```
movl $72, %eax
int $0x80        (128)
```

int handler
jmp *%eax

trap

→ exception

↳ changes privilege level

→ jumps ~~to~~ to OS

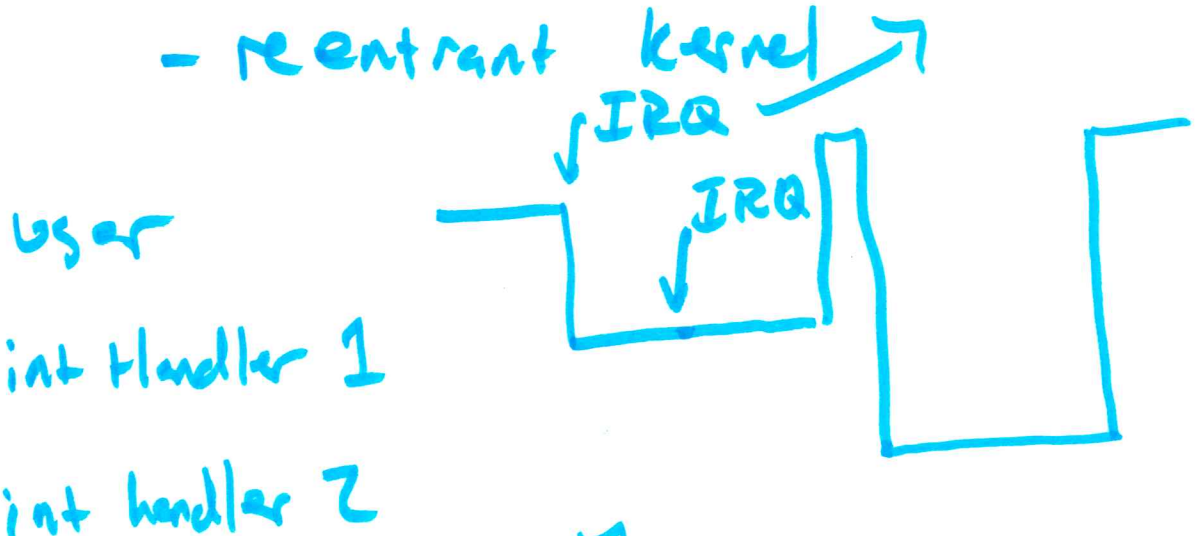
```
movl $0, %eax   → exit
int $0x80
```

What happens if you get an interrupt while servicing an interrupt?

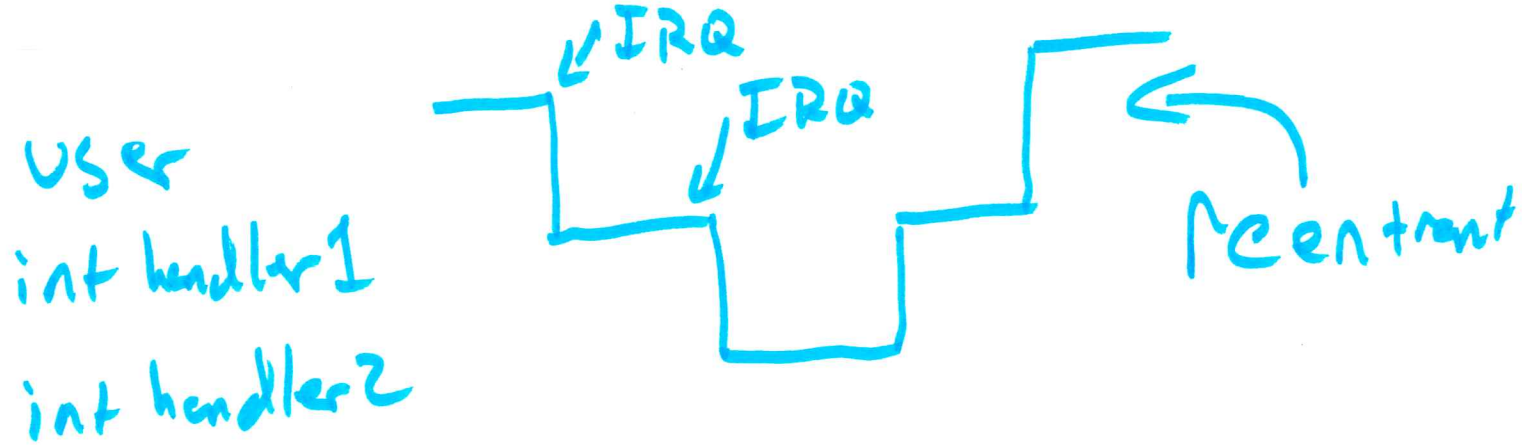
- Stall the new interrupt
  - → easiest thing to do
  - * non-reentrant code

↳ common in real-time OS

- Service the second interrupt + then go back to the first Interrup ReQuest
  - reentrant kernel



non-reentrant



reentrant ~~kernel~~ kernels very hard

To prevent interrupts: interrup enable flag (IF)