

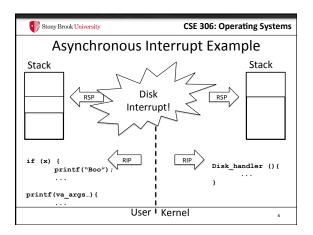
CSE 306: Operating Systems

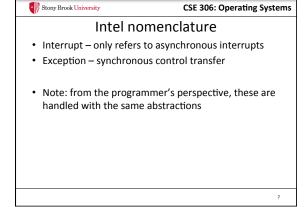
Two types of interrupts

Synchronous: will happen every time an instruction executes (with a given program state)

Divide by zero
System call
Bad pointer dereference

Asynchronous: caused by an external event
Usually device I/O
Timer ticks (well, clocks can be considered a device)





CSE 306: Operating Systems

Lecture outline

Overview

How interrupts work in hardware

How interrupt handlers work in software

How system calls work

New system call hardware on x86

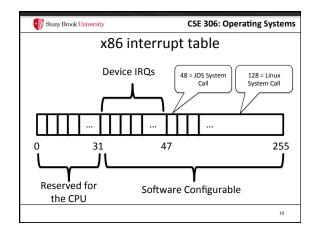
CSE 306: Operating Systems

Interrupt overview

Each interrupt or exception includes a number indicating its type

E.g., 14 is a page fault, 3 is a debug breakpoint

This number is the index into an interrupt table



\*\*CSE 306: Operating Systems

\*\*x86 interrupt overview\*\*

\*\*Each type of interrupt is assigned an index from 0—255.

\*\*O—31 are for processor interrupts; generally fixed by Intel

- E.g., 14 is always for page faults

\*\*32—255 are software configured

- 32—47 are for device interrupts (IRQs) in JOS

\*\*Most device's IRQ line can be configured

\*\*Look up APICs for more info (Ch 4 of Bovet and Cesati)

- 0x80 issues system call in Linux (more on this later)

Software interrupts

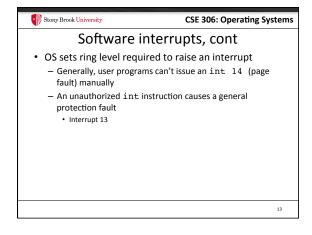
The int <num> instruction allows software to raise an interrupt

Ox80 is just a Linux convention. JOS uses 0x30.

There are a lot of spare indices

You could have multiple system call tables for different purposes or types of processes!

Windows does: one for the kernel and one for win32k



CSE 306: Operating Systems

What happens (high level):

Control jumps to the kernel

At a prescribed address (the interrupt handler)

The register state of the program is dumped on the kernel's stack

Sometimes, extra info is loaded into CPU registers

E.g., page faults store the address that caused the fault in the cr2 register

Kernel code runs and handles the interrupt

When handler completes, resume program (see iret instr.)

CSE 306: Operating Systems

How is this configured?

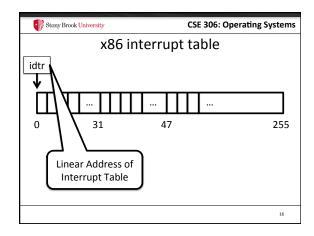
Kernel creates an array of Interrupt descriptors in memory, called Interrupt Descriptor Table, or IDT

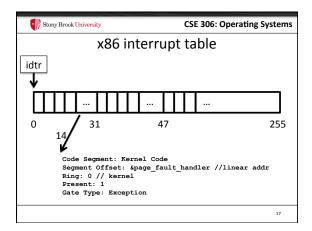
Can be anywhere in memory

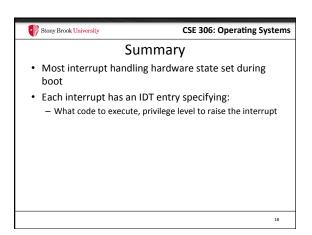
Pointed to by special register (idtr)

c.f., segment registers and gdtr and ldtr

Entry 0 configures interrupt 0, and so on









- · How system calls work
- New system call hardware on x86

Stony Brook University **CSE 306: Operating Systems** High-level goal · Respond to some event, return control to the appropriate process What to do on: - Network packet arrives - Disk read completion - Divide by zero - System call

Stony Brook University **CSE 306: Operating Systems Interrupt Handlers** 

- Just plain old kernel code
- Sort of like exception handlers in Java
- But separated from the control flow of the program
- The IDT stores a pointer to the right handler routine

21

Stony Brook University CSE 306: Operating Systems Lecture outline Overview · How interrupts work in hardware · How interrupt handlers work in software · How system calls work New system call hardware on x86 22

CSE 306: Operating Systems Stony Brook University What is a system call?

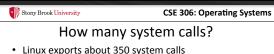
- A function provided to applications by the OS kernel
  - Generally to use a hardware abstraction (file, socket)
  - Or OS-provided software abstraction (IPC, scheduling)
- Why not put these directly in the application?
  - Protection of the OS/hardware from buggy/malicious programs
  - Applications are not allowed to directly interact with hardware, or access kernel data structures

Stony Brook University

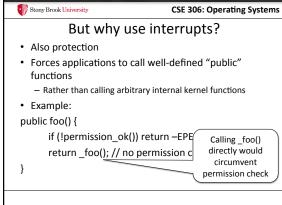
CSE 306: Operating Systems

# System call "interrupt"

- Originally, system calls issued using int instruction
- Dispatch routine was just an interrupt handler
- Like interrupts, system calls are arranged in a table
- See arch/x86/kernel/syscall\_table\*.S in Linux source
- · Program selects the one it wants by placing index in eax register
  - Arguments go in the other registers by calling convention
  - Return value goes in eax



- · Windows exports about 400 system calls for core
- APIs, and another 800 for GUI methods



Stony Brook University CSE 306: Operating Systems Summary

- System calls are the "public" OS APIs
- Kernel leverages interrupts to restrict applications to specific functions
- · Lab 1 hint: How to issue a Linux system call?
  - int \$0x80, with system call number in eax register

CSE 306: Operating Systems

Lecture outline Overview · How interrupts work in hardware · How interrupt handlers work in software · How system calls work New system call hardware on x86

Stony Brook University

CSE 306: Operating Systems Stony Brook University Around P4 era...

- · Processors got very deeply pipelined
  - Pipeline stalls/flushes became very expensive
  - Cache misses can cause pipeline stalls
- System calls took twice as long from P3 to P4

  - IDT entry may not be in the cache
  - Different permissions constrain instruction reordering

Stony Brook University **CSE 306: Operating Systems** Idea · What if we cache the IDT entry for a system call in a special CPU register? - No more cache misses for the IDT! - Maybe we can also do more optimizations Assumption: system calls are frequent enough to be worth the transistor budget to implement this - What else could you do with extra transistors that helps performance?



CSE 306: Operating Systems

### AMD: syscall/sysret

- These instructions use MSRs (machine specific registers) to store:
  - Syscall entry point and code segment
  - Kernel stack
- A drop-in replacement for int 0x80
- Everyone loved it and adopted it wholesale
  - Even Intel!

31

Stony Brook University

CSE 306: Operating Systems

#### Aftermath

- Getpid() on my desktop machine (recent AMD 6core):
  - Int 80: 371 cycles
  - Syscall: 231 cycles
- So system calls are definitely faster as a result!

32

Stony Brook University

CSE 306: Operating Systems

#### In Lab 1

- You will use the int instruction to implement system calls
- You are welcome to use syscall if you prefer

Stony Brook University

CSE 306: Operating Systems

## Summary

- Interrupt handlers are specified in the IDT
- Understand how system calls are executed
  - Why interrupts?
  - Why special system call instructions?