Networking wrap-up and I/O systems

CS 241

April 28, 2014

University of Illinois

Today

Network programming tips

Network Address Translation

I/O systems

Tip #1: Can't bind?

Problem: `bind()` says "address already in use"

- You have stopped your server, and then re-started it right away
- The sockets that were used by the first incarnation of the server are still active

setsockopt

```
int yes = 1;

setsockopt (fd, SOL_SOCKET, SO_REUSEADDR,
                (char *) &yes, sizeof (yes));
```

- Call just before bind()
- Allows bind to succeed despite the existence of existing connections in the requested TCP port
- Connections in limbo (e.g. lost final ACK) will cause bind to fail

Tip #2: Dealing with abruptly closed connection

Tip #2: Dealing with abruptly closed connection

Problem: Socket at other end is closed

- Write to your end generates `SIGPIPE`
- This signal kills the program by default!

```
signal (SIGPIPE, SIG_IGN);
```

- Call at start of main in server
- Allows you to ignore broken pipe signals
- Can ignore or install a proper signal handler
- Default handler exits (terminates process)

Tip #3: Beej's guide

Beej's Guide to Network Programming

`http://beej.us/guide/bgnet/`

Review

Threads in your web server

Why are multiple threads useful in your web server – as opposed to serving all clients with a single thread in a single process? (Check all that apply)

- Multiple threads can spread work across multiple cores / CPUs to decrease processing time.
- Multiple threads have greater memory space to read files and write them to the network.
- A single thread would have to switch back and forth between each connection, which is slow and annoying to program.
- One thread can be reading/writing from the network while another is waiting to read a file off disk.

DNS caching

Why does the DNS system use caching? (Check all that apply)

- Returns more up-to-date results
- Improves speed of response
- Decreases workload on root and authoritative DNS servers
- Improves security
- Improves robustness (things still work even if some DNS servers fail)

Network Address Translation (NAT)

The problem

Your ISP gives you one IP address. But...

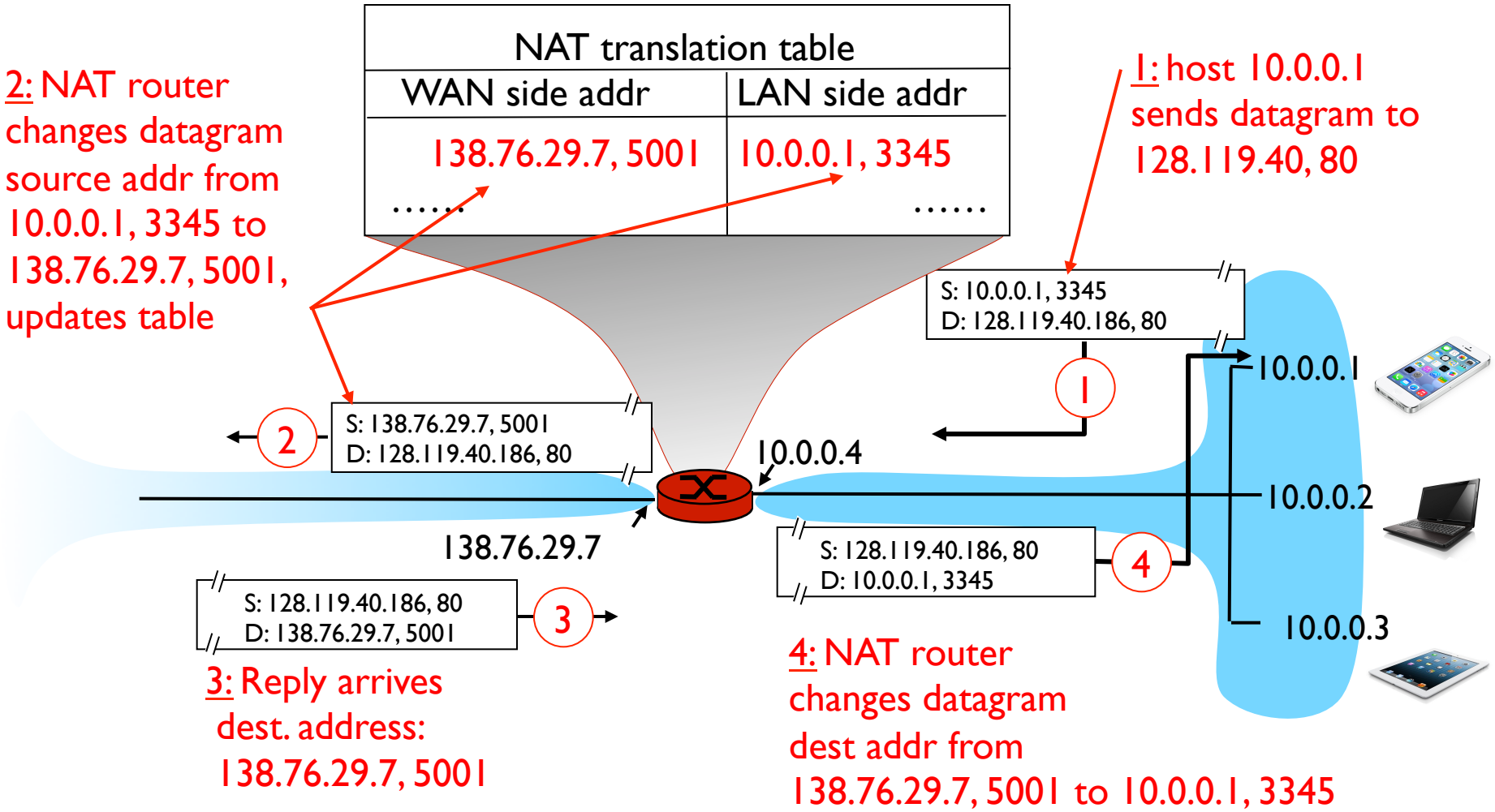
NAT: Network Address Translation

Approach

- Assign one router a global IP address
- Assign internal hosts local IP addresses
- A box in the middle converts between them
  - (e.g. wireless router in your home)

When a packet arrives from the Internet, how do you know which internal host it's destined for?

NAT: Network Address Translation



NAT: Benefits

Local network uses just one (or a few) IP address as far as outside world is concerned

- No need to be allocated range of addresses from ISP
  - 16-bit port-number field: 60,000 simultaneous connections with a single LAN-side address!
  - Might use a few IPs in a large private enterprise network
- Can change addresses of devices in local network without notifying outside world
- Can change ISP without changing addresses of devices in local network

Security

- Devices inside local net not explicitly addressable by outside world
- Connection needs to be initiated by inside device before a specific host can reach it from the public Internet

NAT example use: load balancing

Load balancing

- Balance the load on a set of identical servers, which are accessible from a single IP address

NAT solution

- Servers are assigned private addresses
- NAT acts as a proxy for requests to the server from the public network
- NAT changes the destination IP address of arriving packets to one of the private addresses for a server
- Balances load on the servers by assigning addresses in a round-robin fashion

NAT: Consequences

End-to-end connectivity broken

- NAT destroys universal end-to-end reachability of hosts on the Internet
- A host in the public Internet often cannot initiate communication to a host in a private network
- Even worse when two hosts that are in different private networks need to communicate with each other

NAT: Consequences

Broken if IP address in application data

- Applications often carry IP addresses in the payload of the application data
- No longer work across a private-public network boundary
- Hack: Some NAT devices inspect the payload of widely used application layer protocols and, if an IP address is detected in the application-layer header or the application payload, translate the address according to the address translation table

NAT: Consequences

Ossification of Internet protocols

- NAT must be aware of port numbers which are inside transport header
- Existing NATs don't support your fancy new transport protocol
  - and might even block standard protocols like UDP
- Result: Difficult to invent new transport protocols
  - ...unless they just pretend to be TCP

I/O systems

Input and Output

A computer's job is to process data

- Computation (CPU, cache, and memory)
- Move data into and out of a system (between I/O devices and memory)

Challenges with I/O devices

- Different categories: storage, networking, displays, etc.
- Large number of device drivers to support
- Device drivers run in kernel mode and can crash systems

Goals of the OS

- Provide a generic, consistent, convenient and reliable way to access I/O devices
- As device-independent as possible
- High performance I/O

How does the CPU talk to devices?

Device controller: Hardware that enables devices to talk to the peripheral bus

Host adapter: Hardware that enables the computer to talk to the peripheral bus

Bus: Wires that transfer data between components inside computer

Device controller allows OS to specify simpler instructions to access data

Example: a disk controller

- Translates "access sector 23" to "move head reader 1.672725272 cm from edge of platter"
- Disk controller "advertises" disk parameters to OS, hides internal disk geometry
- Most modern hard drives have disk controller embedded as a chip on the physical device

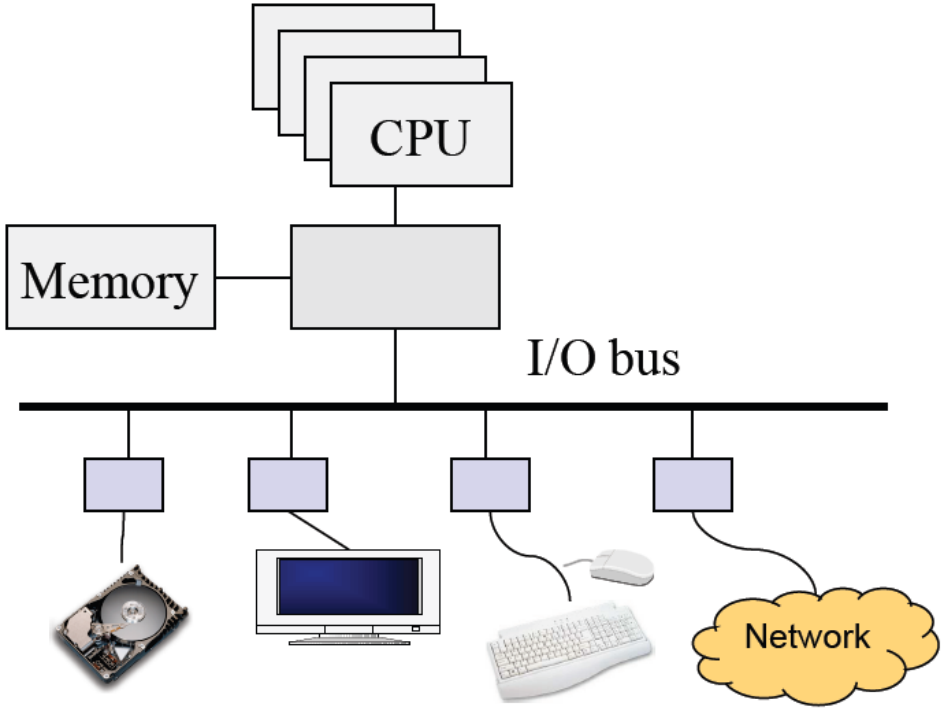
Review: Computer Architecture

Compute hardware

- CPU and caches
- Chipset
- Memory

I/O Hardware

- I/O bus or interconnect
- I/O controller or adaptor
- I/O device



Two types of I/O

- Programmed I/O (PIO)
  - CPU does the work of moving data
- Direct Memory Access (DMA)
  - CPU offloads the work of moving data to DMA controller

Programmed Input Device

Device controller

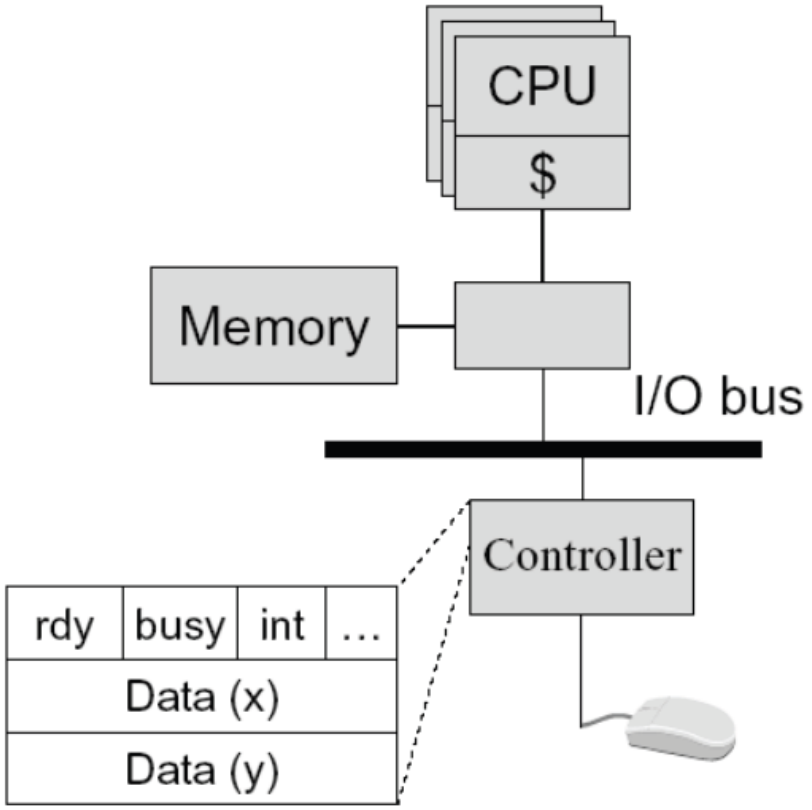
- Status registers
  - ready: tells if the host is done
  - busy: tells if the controller is done
  - int: interrupt
  - …
- Data registers

A simple mouse design

- When moved, put (X, Y) in mouse's device controller's data registers
- Interrupt CPU

Input on an interrupt

- CPU saves state of currently-executing program
- Reads values in X, Y registers
- Sets ready bit
- Wakes up a process/thread or execute a piece of code to handle interrupt



Programmed Output Device

Device

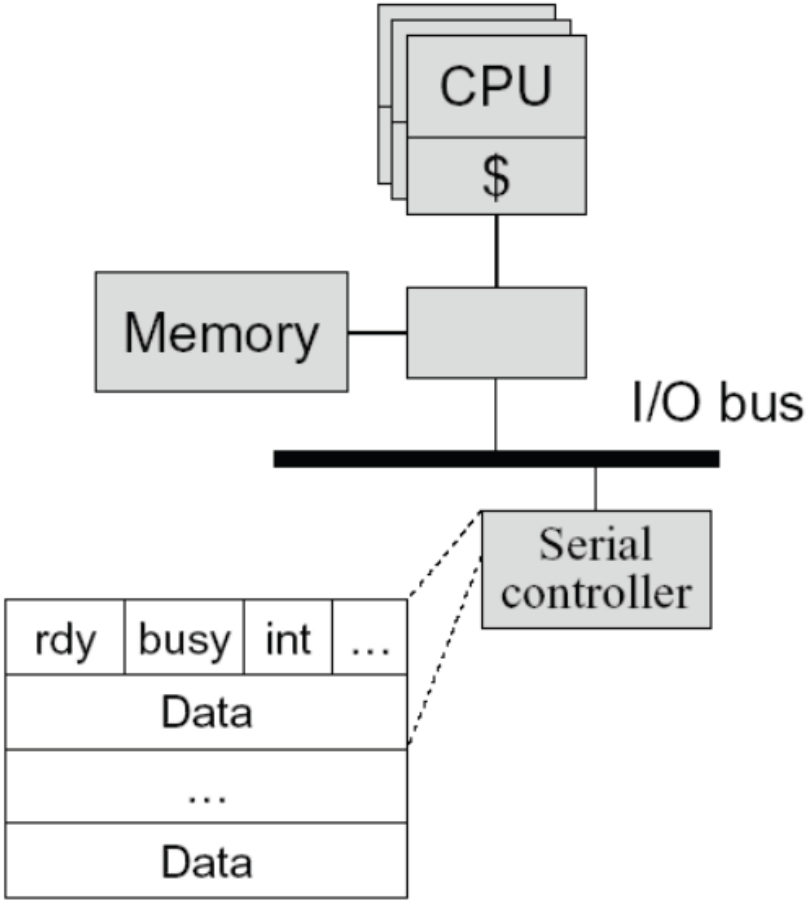
- Status registers (ready, busy, …)
- Data registers

Example

- A serial output device

Perform an output

- CPU: Poll the busy bit
- Writes the data to data register(s)
- Set ready bit
- Controller sets busy bit and transfers data
- Controller clears the busy bit



Direct Memory Access (DMA)

DMA controller or adaptor

- Status register (ready, busy, interrupt, …)
- DMA command register
- DMA register (address, size)
- DMA buffer

Host CPU initiates DMA

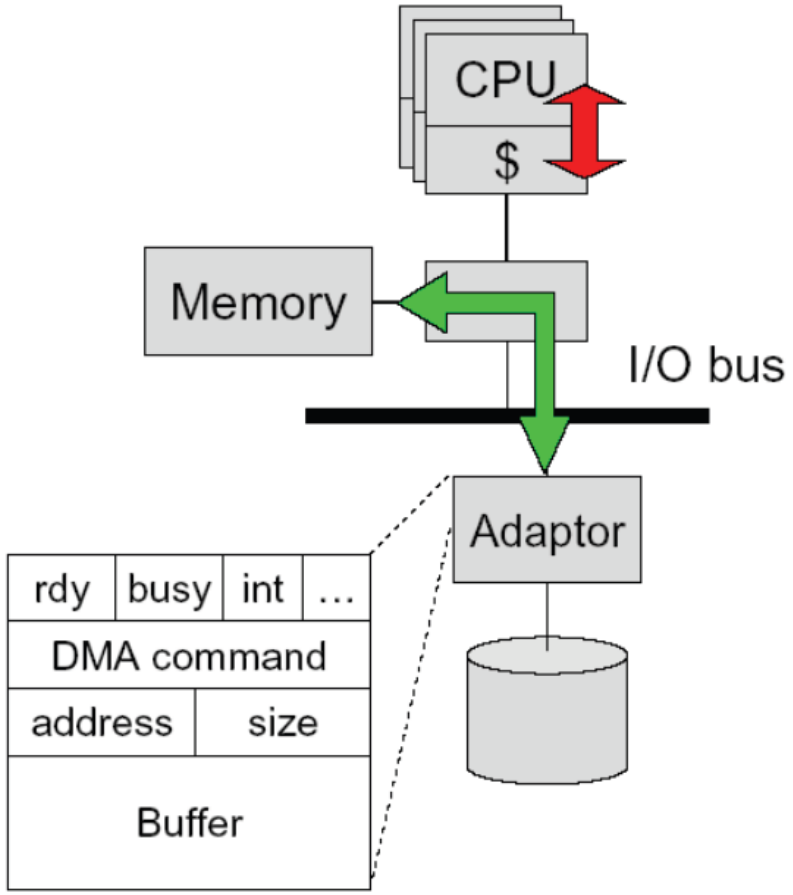
- Device driver call (kernel mode)
- Wait until DMA device is free
- Initiate a DMA transaction (command, memory address, size)
- Block

Controller performs DMA

- DMA data to device (size--; address++)
- Issue interrupt on completion (size == 0)

CPU's interrupt handler

- Wakeup the blocked process



Memory-mapped I/O

Use the same address bus to address both memory and I/O devices

- The memory and registers of I/O devices are mapped to address values
- Allows same CPU instructions to be used with regular memory and devices

I/O devices, memory controller, monitor address bus

- Each responds to addresses they own

Orthogonal to DMA

- May be used with, or without, DMA

Polling- vs. Interrupt-driven I/O

Polling

- CPU issues I/O command
- CPU directly writes instructions into device's registers
- CPU busy waits for completion

Interrupt-driven I/O

- CPU issues I/O command
- CPU directly writes instructions into device's registers
- CPU continues operation until interrupt

Direct Memory Access (DMA)

- Typically done with Interrupt-driven I/O
- CPU asks DMA controller to perform device-to-memory transfer
- DMA issues I/O command and transfers new item into memory
- CPU module is interrupted after completion

Which is better, polling or interrupt-driven I/O?

Polling- vs. Interrupt-driven I/O

Polling

- Expensive for large transfers
- Better for small, dedicated systems with infrequent I/O

Interrupt-driven

- Overcomes CPU busy waiting
- I/O module interrupts when ready: event driven

How Interrupts are implemented

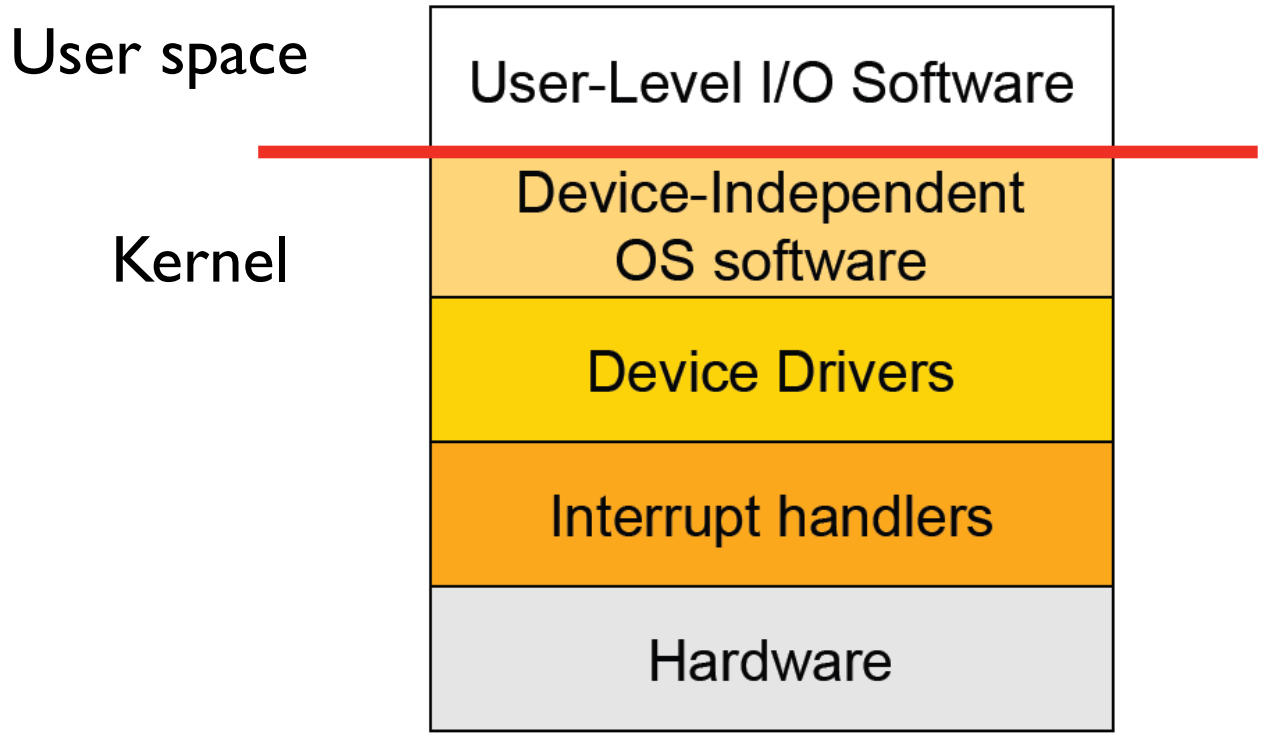
CPU hardware has an interrupt report line that the CPU tests after executing every instruction

- If a(ny) device raises an interrupt by setting interrupt report line
  - CPU catches the interrupt and saves the state of current running process into PCB
  - CPU dispatches/starts the interrupt handler
  - Interrupt handler determines cause, services the device and clears the interrupt report line

Other uses of interrupts: exceptions

- Division by zero, wrong address
- System calls (software interrupts/signals, trap)
- Virtual memory paging

I/O Software Stack



Announcements

MPs, Take Two

- May resubmit, after MP7 due date and before final
- Half credit on resubmissions
- Watch Piazza later this week for grading and timing details

Talk today

- Scott Shenker (UC Berkeley)
- "Software-Defined Networking: Introduction and Retrospective"