Last Class: Mass storage, disk scheduling

- Minimizing seek time and rotational latency
- Disk head scheduling
  - FCFS, SSTF, SCAN, C-SCAN
- Types of mass storage
  - HDDs, SSDs, tertiary storage, RAID

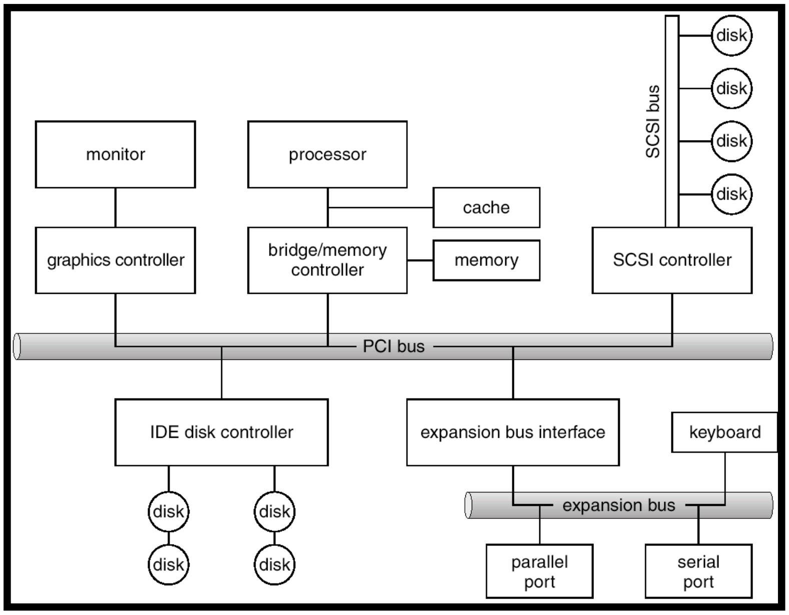
Today: I/O Systems

- How does I/O hardware influence the OS?
- What I/O services does the OS provide?
- How does the OS implement those services?
- How can the OS improve the performance of I/O?

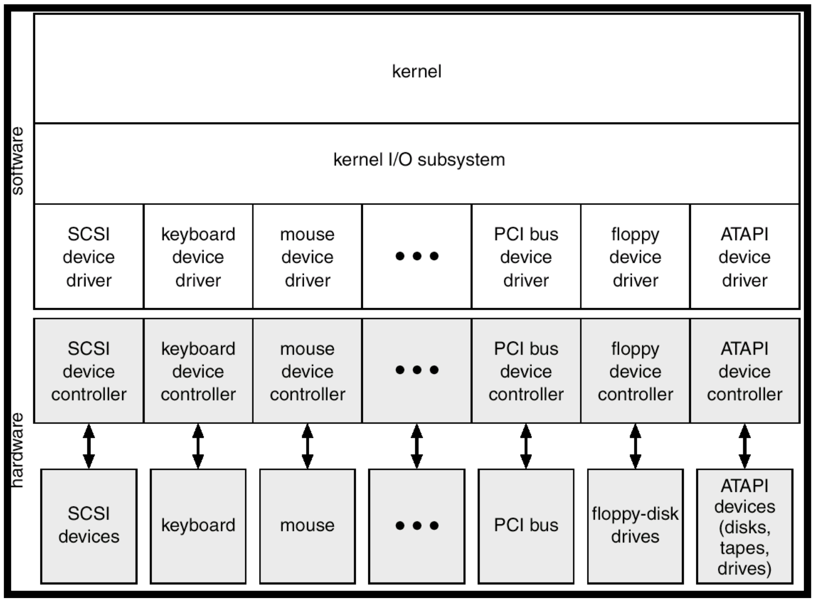
Architecture of I/O Systems

- Key components
  - **System bus:** allows the device to communicate with the CPU, typically shared by multiple devices.
  - A device **port** typically consisting of 4 registers:
    - **Status** indicates a device busy, data ready, or error condition
    - **Control:** command to perform
    - **Data-in:** data being sent from the device to the CPU
    - **Data-out:** data being sent from the CPU to the device
  - **Controller:** receives commands from the system bus, translates them into device actions, and reads/writes data onto the system bus.
  - The device itself
- Traditional devices: disk drive, printer, keyboard, modem, mouse, display
- Non-traditional devices: joystick, robot actuators, flying surfaces of an airplane, fuel injection system of a car, ...

PCI Bus Structure



Kernel I/O Subsystem



software

hardware

kernel

kernel I/O subsystem

SCSI device driver | keyboard device driver | mouse device driver | • • • | PCI bus device driver | floppy device driver | ATAPI device driver

SCSI device controller | keyboard device controller | mouse device controller | • • • | PCI bus device controller | floppy device controller | ATAPI device controller

SCSI devices | keyboard | mouse | • • • | PCI bus | floppy-disk drives | ATAPI devices (disks, tapes, drives)

Device I/O Port location on PCs

I/O address range (hexadecimal)	device
000-00F	DMA controller
020-021	interrupt controller
040-043	timer
200-20F	game controller
2F8-2FF	serial port (secondary)
320-32F	hard-disk controller
378-37F	parallel port
3D0-3DF	graphics controller
3F0-3F7	diskette-drive controller
3F8-3FF	serial port (primary)

I/O Services Provided by OS

- Naming of files and devices. (On Unix, devices appear as files in the /dev directory)
- Access control.
- Operations appropriate to the files and devices.
- Device allocation.
- Buffering, caching, and spooling to allow efficient communication with devices.
- I/O scheduling.
- Error handling and failure recovery associated with devices (command retries, for example).
- Device drivers to implement device-specific behaviors.

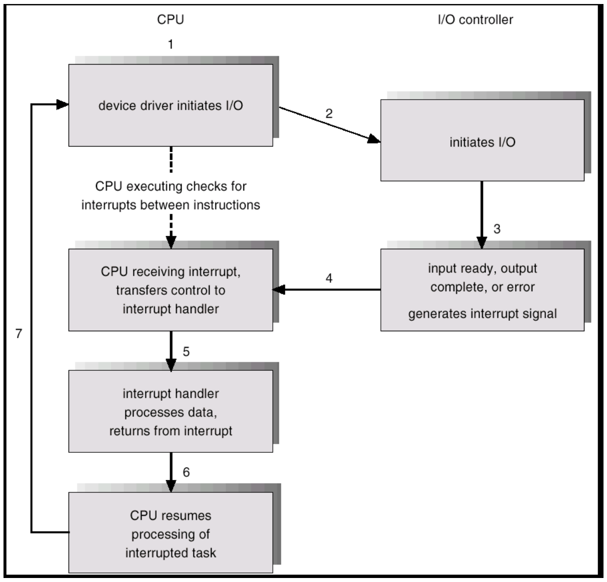
Communication using Polling

- CPU busy-waits until the status is idle.
- CPU sets the command register and data-out if it is an output operation.
- CPU sets status to command-ready => controller sets status to busy.
- Controller reads the command register and performs the command, placing a value in data-in if it is an input command.
- If the operation succeeds, the controller changes the status to idle.
- CPU observes the change to idle and reads the data if it was an input operation.
- Good choice if data must be handled promptly, like for a modem or keyboard.
- What happens if the device is slow compared to the CPU?

Communication using Interrupts

- Rather than using busy waiting, the device can interrupt the CPU when it completes an I/O operation.

- On an I/O interrupt:
  - Determine which device caused the interrupt.
  - If the last command was an input operation, retrieve the data from the device register.
  - Start the next operation for that device.


CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
CPU receiving interrupt, transfers control to interrupt handler
4
input ready, output complete, or error
generates interrupt signal
7
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task

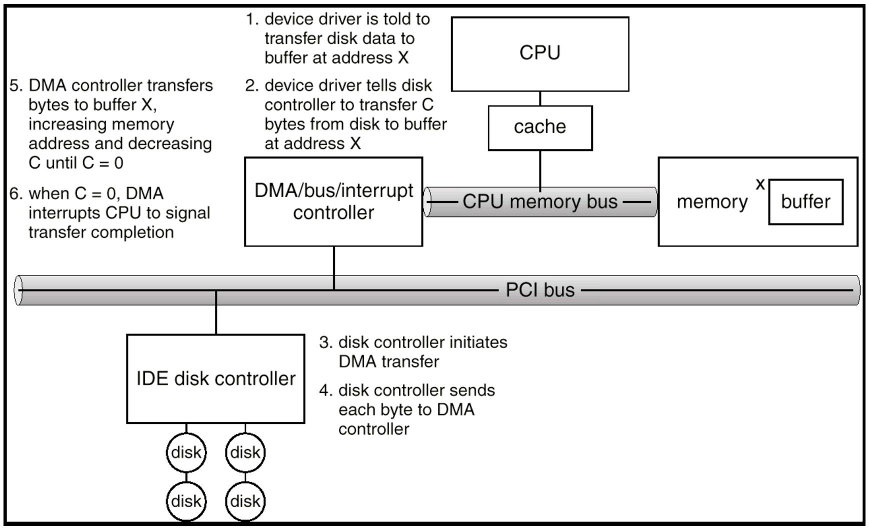
Intel x86 Event Vectors

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19Ð31	(Intel reserved, do not use)
32Ð255	maskable interrupts

Direct Memory Access

- For devices that transfer large volumes of data at a time (like a disk block), it is expensive to have the CPU retrieve these one byte at a time.
- **Solution:** Direct memory access (DMA)
  - Use a sophisticated DMA controller that can write directly to memory. Instead of data-in/data-out registers, it has an address register.
  - The CPU tells the DMA the locations of the source and destination of the transfer.
  - The DMA controller operates the bus and interrupts the CPU when the entire transfer is complete, instead of when each byte is ready.
  - The DMA controller and the CPU compete for the memory bus, slowing down the CPU somewhat, but still providing better performance than if the CPU had to do the transfer itself.

Steps in DMA transfer



1. device driver is told to transfer disk data to buffer at address X
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
6. when C = 0, DMA interrupts CPU to signal transfer completion

Application Programmer's View of I/O Devices

- The OS provides a high-level interface to devices, greatly simplifying the programmer's job.
  - Standard interfaces are provided for related devices.
  - Device dependencies are encapsulated in device drivers.
  - New devices can be supported by providing a new device driver.
- **Device characteristics:**
  - Transfer unit: **character** or **block**
  - Access method: **sequential** or **random** access
  - Timing: **synchronous** or **asynchronous**
    - Most devices are asynchronous, while I/O system calls are synchronous => The OS implements *blocking I/O*
  - Sharable or dedicated
  - Speed
  - Operations: Input, output, or both
  - Examples: keyboard (sequential, character), disk (block, random or sequential)

Examples of I/O Device types

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only readÐwrite	CD-ROM graphics controller disk

Block and Character devices

- Block devices include disk drives
  - Commands include read, write, seek
  - Raw I/O or file-system access
  - Memory-mapped file access possible
- Character devices include keyboards, mice, serial ports
  - Commands include `get, put`
  - Libraries layered on top for added functionality (e.g., line editing, backspace)

I/O Buffering

I/O devices typically contain a small on-board memory where they can store data temporarily before transferring to/from the CPU.

- A disk buffer stores a block when it is read from the disk.
- It is transferred over the bus by the DMA controller into a buffer in physical memory.
- The DMA controller interrupts the CPU when the transfer is done.

Why buffer on the OS side?

- To cope with speed mismatches between devices.
  - Example: Receive file over a network (slow) and store to disk (faster)

- To cope with devices that have different data transfer sizes.
  - Example: ftp brings the file over the network one packet at a time. Stores to disk happen one block at a time.

- To minimize the time a user process is blocked on a write.
  - Writes => copy data to a kernel buffer and return control to the user program. The write from the kernel buffer to the disk is done later.

Caching

- Improve disk performance by reducing the number of disk accesses.
  - **Idea:** keep recently used disk blocks in main memory after the I/O call that brought them into memory completes.
  - **Example:** Read (diskAddress)

    If (block in memory) return value from memory

    Else ReadSector(diskAddress)
  - **Example:** Write (diskAddress)

    If (block in memory) update value in memory

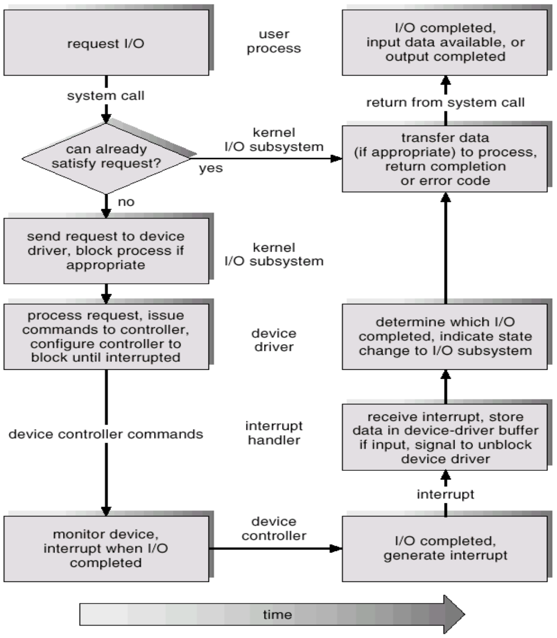
    Else Allocate space in memory, read block from disk, and update value in memory

- What should happen when we write to a cache?
  - **write-through** policy (write to all levels of memory containing the block, including to disk). High reliability.
  - **write-back** policy (write only to the fastest memory containing the block, write to slower memories and disk sometime later). Faster.

Putting the Pieces Together - a Typical Read Call

1. User process requests a read from a device.
2. OS checks if data is in a buffer. If not,
   a) OS tells the device driver to perform input.
   b) Device driver tells the DMA controller what to do and blocks itself.
   c) DMA controller transfers the data to the kernel buffer when it has all been retrieved from the device.
   d) DMA controller interrupts the CPU when the transfer is complete.
3. OS transfers the data to the user process and places the process in the ready queue.
4. When the process gets the CPU, it begins execution following the system call.

I/O request Lifecycle


request I/O
user process
I/O completed, input data available, or output completed
system call
return from system call
can already satisfy request?
kernel I/O subsystem
yes
transfer data (if appropriate) to process, return completion or error code
no
send request to device driver, block process if appropriate
kernel I/O subsystem
process request, issue commands to controller, configure controller to block until interrupted
device driver
determine which I/O completed, indicate state change to I/O subsystem
device controller commands
interrupt handler
receive interrupt, store data in device-driver buffer if input, signal to unblock device driver
interrupt
monitor device, interrupt when I/O completed
device controller
I/O completed, generate interrupt
time

Summary

- I/O is expensive for several reasons:
  - Slow devices and slow communication links
  - Contention from multiple processes.
  - I/O is typically supported via system calls and interrupt handling, which are slow.
- Approaches to improving performance:
  - Reduce data copying by caching in memory
  - Reduce interrupt frequency by using large data transfers
  - Offload computation from the main CPU by using DMA controllers.
  - Increase the number of devices to reduce contention for a single device and thereby improve CPU utilization.
  - Increase physical memory to reduce amount of time paging and thereby improve CPU utilization.