CSC139 Operating System Principles

Fall 2019, Part 4-2

Instructor: Dr. Yuan Cheng

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?

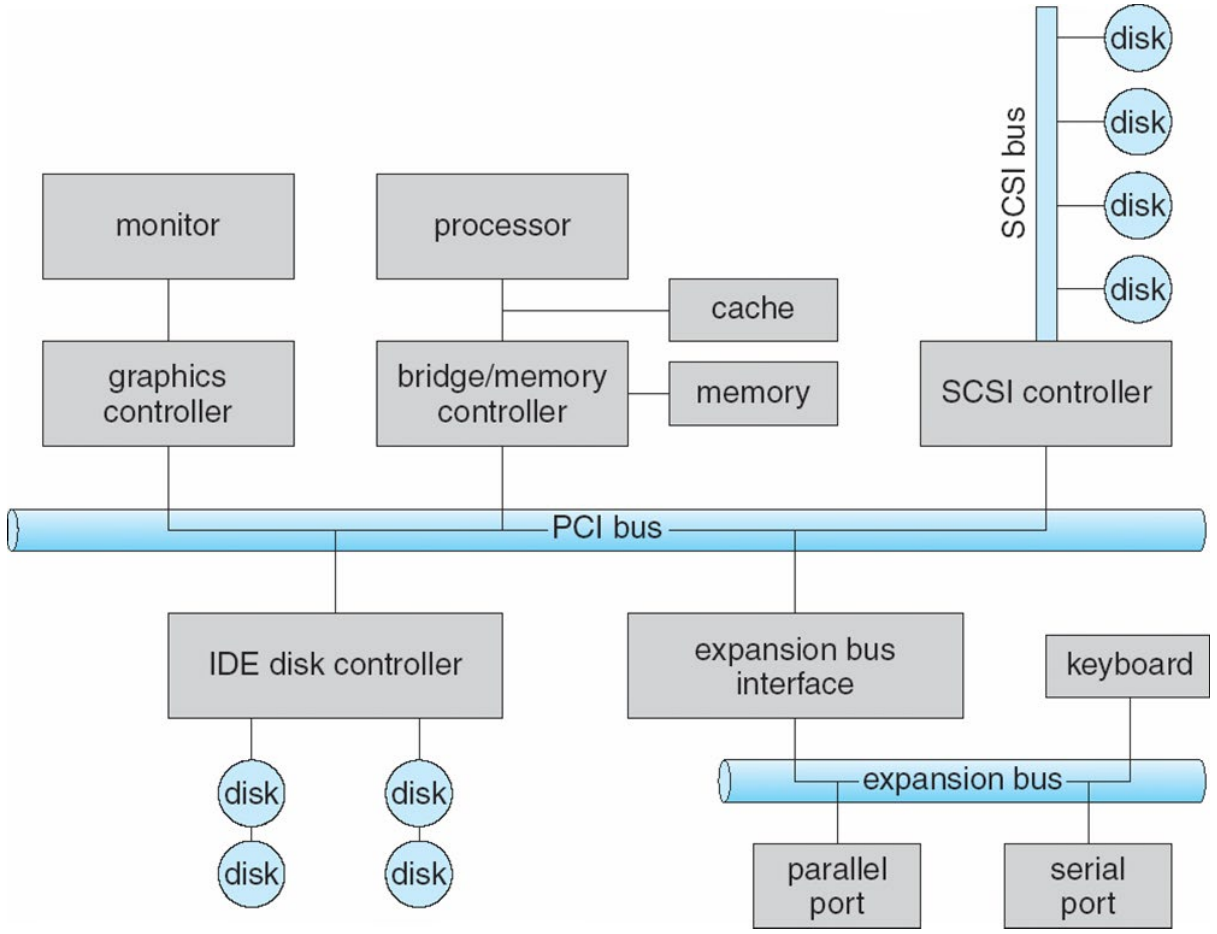
Overview

- I/O management is a major component of operating system design and operation
  - Important aspect of computer operation
  - I/O devices vary greatly
  - Various methods to control them
  - Performance management
  - New types of devices frequent
- Ports, busses, device controllers connect to various devices
- Device drivers encapsulate device details
  - Present uniform device-access interface to I/O subsystem

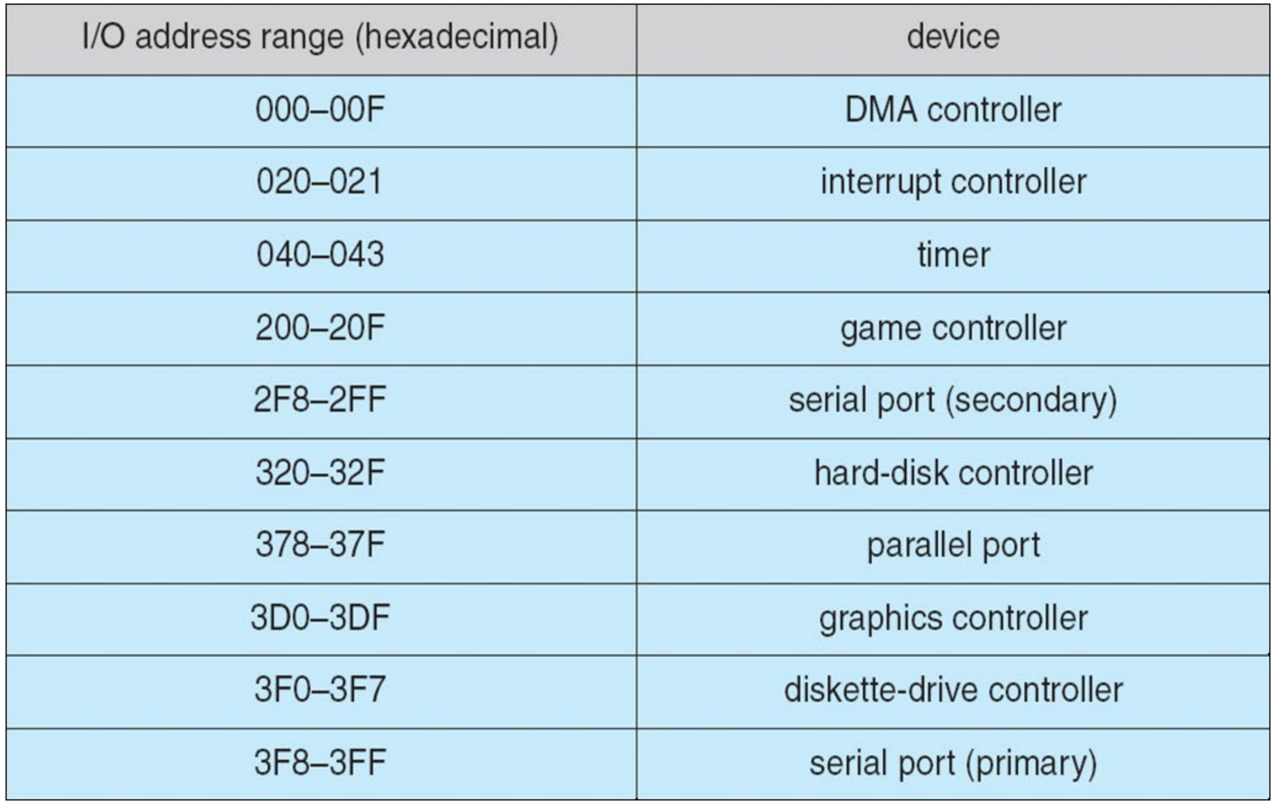
Architecture of I/O Systems

- 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, translate 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: joysticks, robot actuators, flying surfaces of an airplane, fuel injection system of a car, …

A Typical PC Bus Structure



Device I/O Port Locations 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

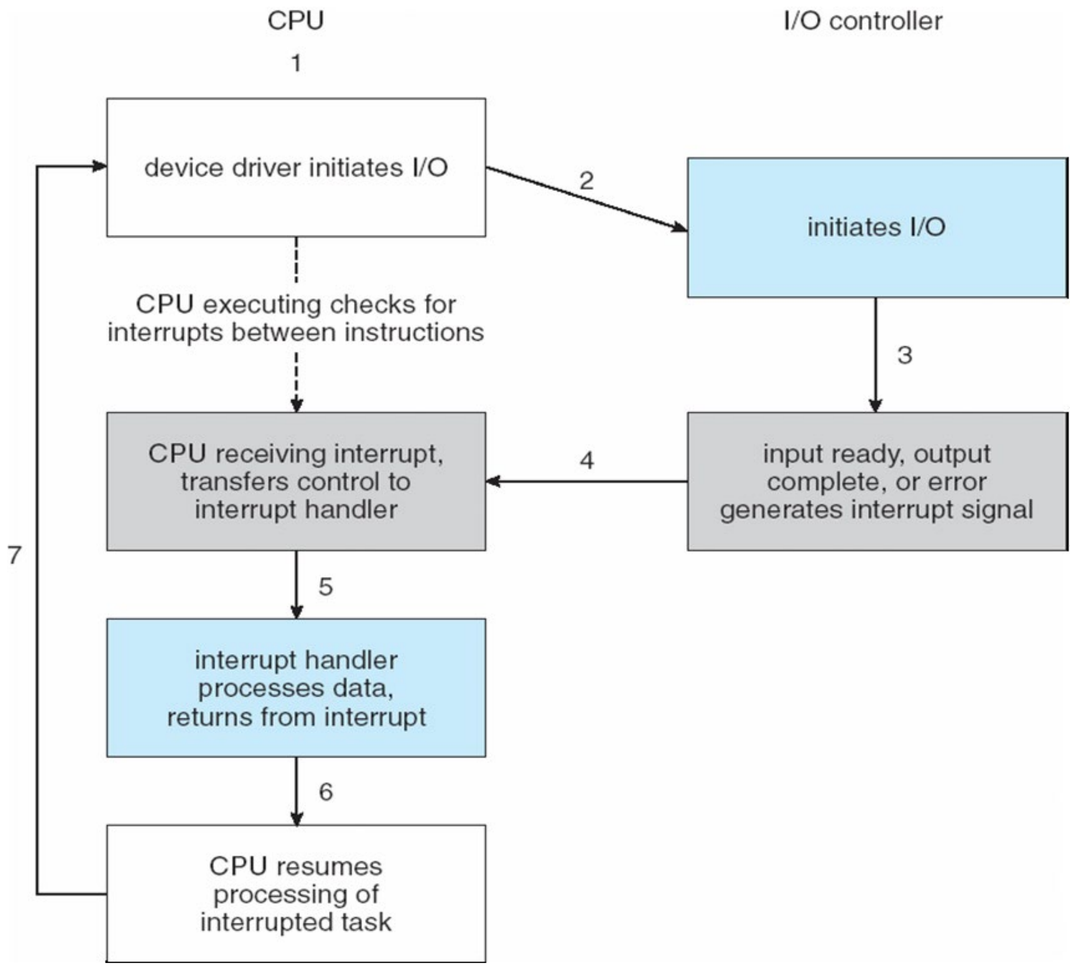
- Naming of files and devices. (On Unix, devices appear as files in the /dev directory)
- Access control
- Operations appropriate to the files and devices
- Device allocation
- Buffering, caching, and spooling to allow efficient communication with devices
- I/O scheduling
- Error handling and failure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

Communication using Polling

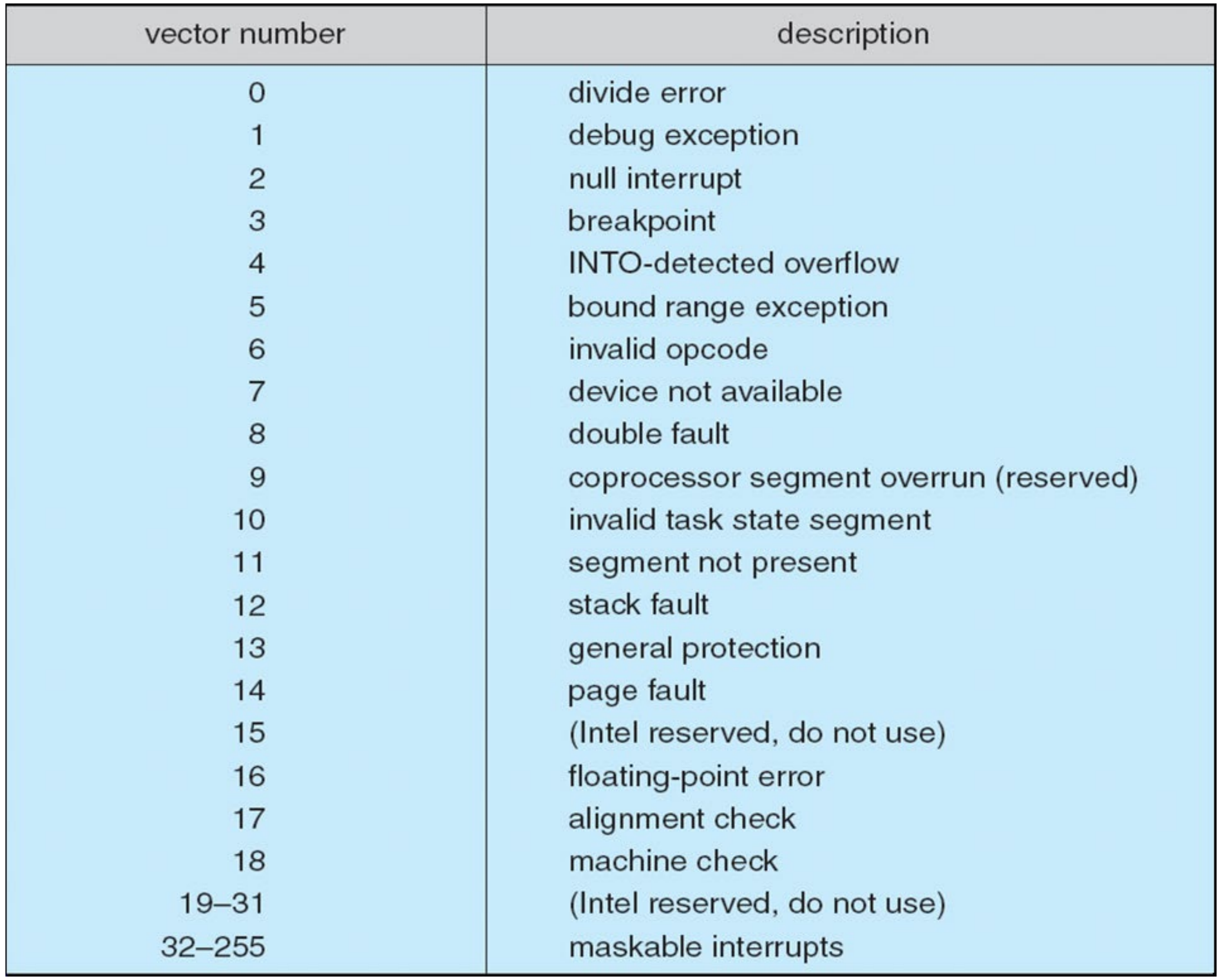
- CPU busy-waits until the status is idle
- CPU sets the command register and data-out if it is an output operation
- CPU sets status to command-ready => controller sets status to busy
- Controller reads the command register and performs the command, placing a value in data-in if it is an input command
- If the operation succeeds, the controller change 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 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 Pentium Processor 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
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

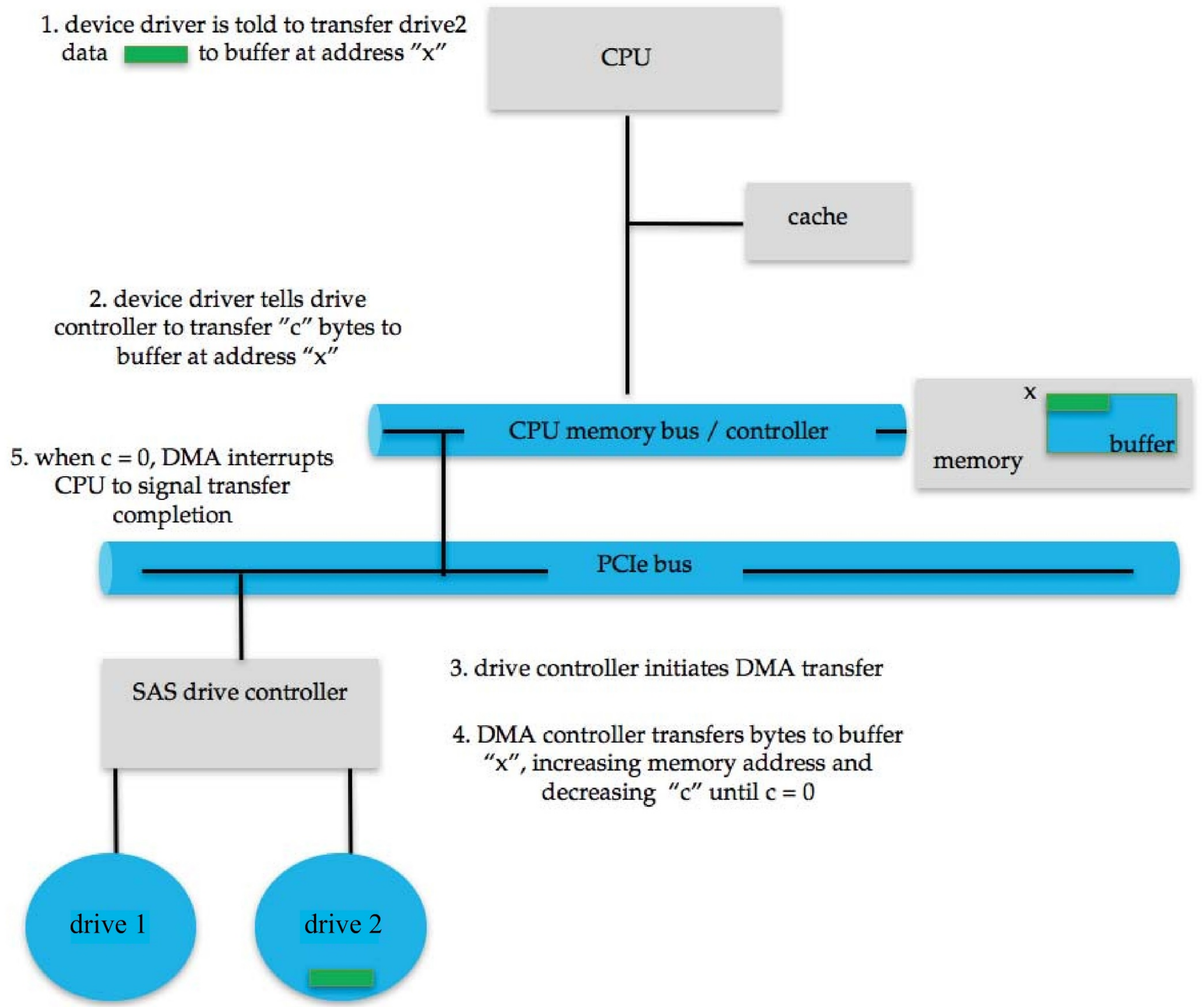
Interrupts (cont.)

- Interrupt mechanism also used for exceptions
  - Terminate process, crash system due to hardware error
- Page fault executes when memory access error
- System call executes via trap to trigger kernel to execute request
- Multi-CPU systems can process interrupts concurrently
  - If operating system designed to handle it
- Used for time-sensitive processing, frequent, must be fast

Direct Memory Access

- Used to avoid programmed I/O (one byte at a time) for large data movement
- 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.
    - Cycle stealing from CPU but still much more efficient
  - 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.

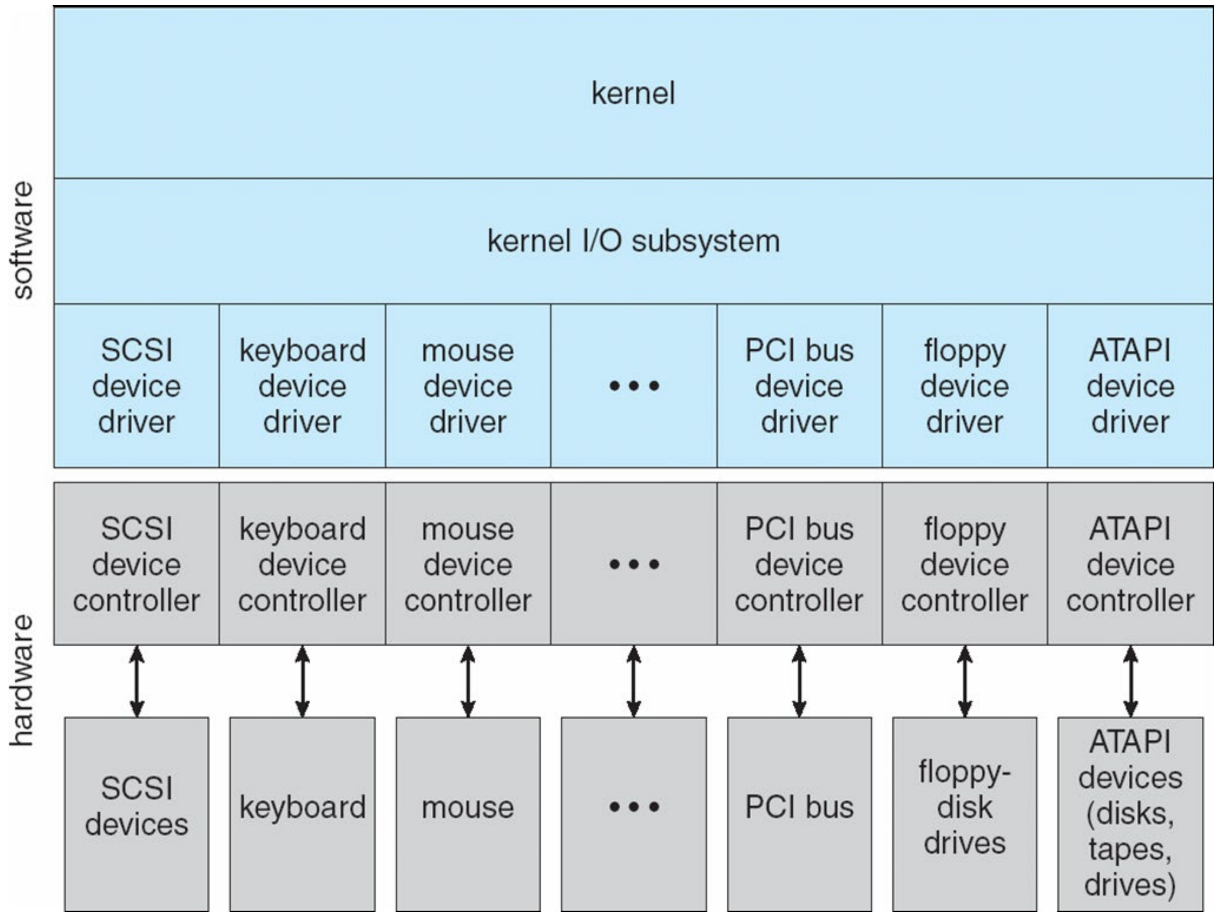
Six Steps Process to Perform DMA Transfer


1. device driver is told to transfer drive2 data to buffer at address "x"
CPU
cache
2. device driver tells drive controller to transfer "c" bytes to buffer at address "x"
CPU memory bus / controller
x
buffer
memory
5. when c = 0, DMA interrupts CPU to signal transfer completion
PCIe bus
SAS drive controller
3. drive controller initiates DMA transfer
4. DMA controller transfers bytes to buffer "x", increasing memory address and decreasing "c" until c = 0
drive 1
drive 2

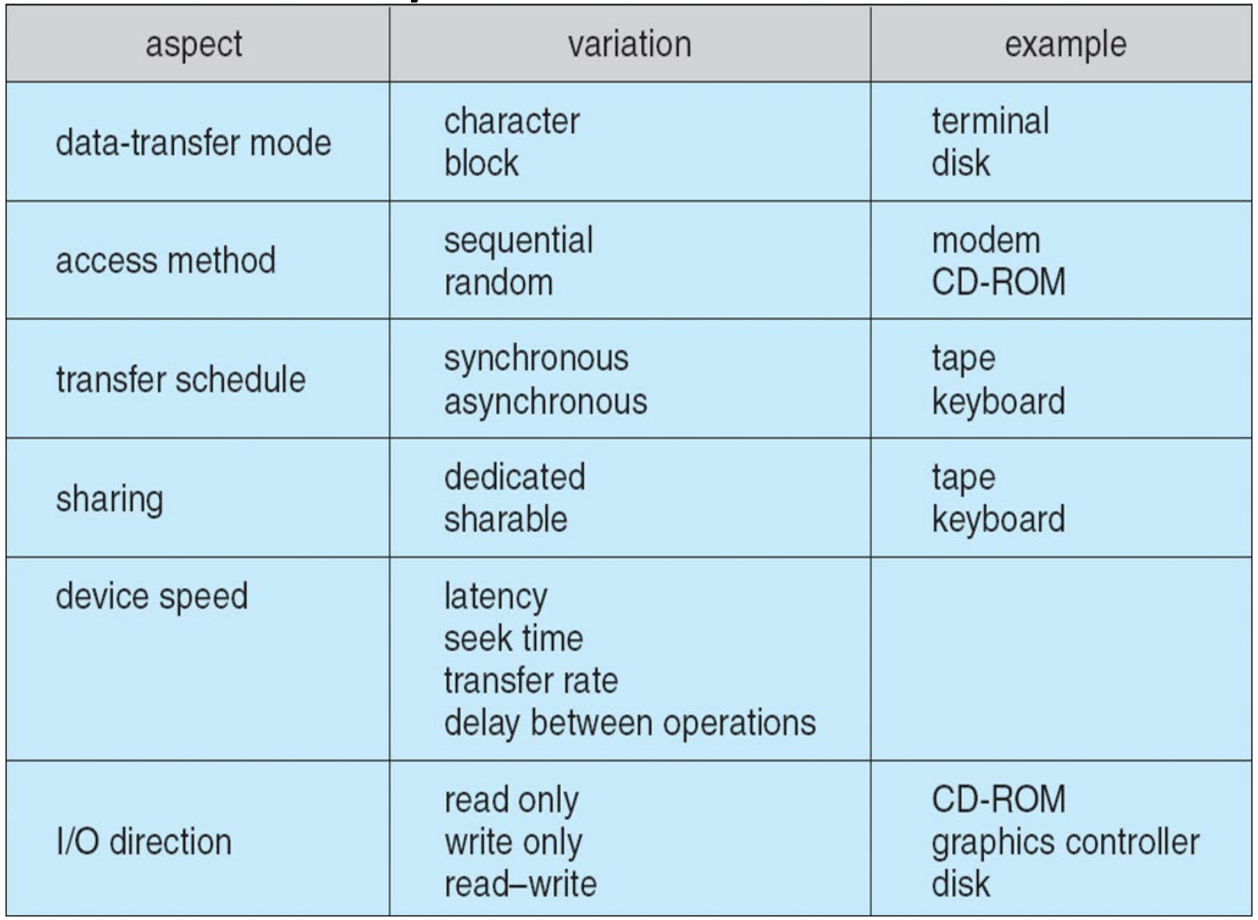
Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes
- Device-driver layer hides differences among I/O controllers from kernel
- New devices talking already-implemented protocols need no extra work
- Each OS has its own I/O subsystem structures and device driver frameworks
- Devices vary in many dimensions
  - Character-stream or block
  - Sequential or random-access
  - Synchronous or asynchronous (or both)
    - Most devices are asynchronous, while I/O system calls are synchronous => The OS implements blocking I/O
  - Sharable or dedicated
  - Speed of operation
  - read-write, read only, or write only

A Kernel I/O Structure



Characteristics of I/O Devices



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 read–write	CD-ROM graphics controller disk

Characteristics of I/O Devices (cont.)

- Subtleties of devices handled by device drivers
- Broadly I/O devices can be grouped by the OS into
  - Block I/O
  - Character I/O (Stream)
  - Memory-mapped file access
  - Network sockets
- For direct manipulation of I/O device specific characteristics, usually an escape / back door
  - Unix **`ioctl()`** call to send arbitrary bits to a device control register and data to device data register

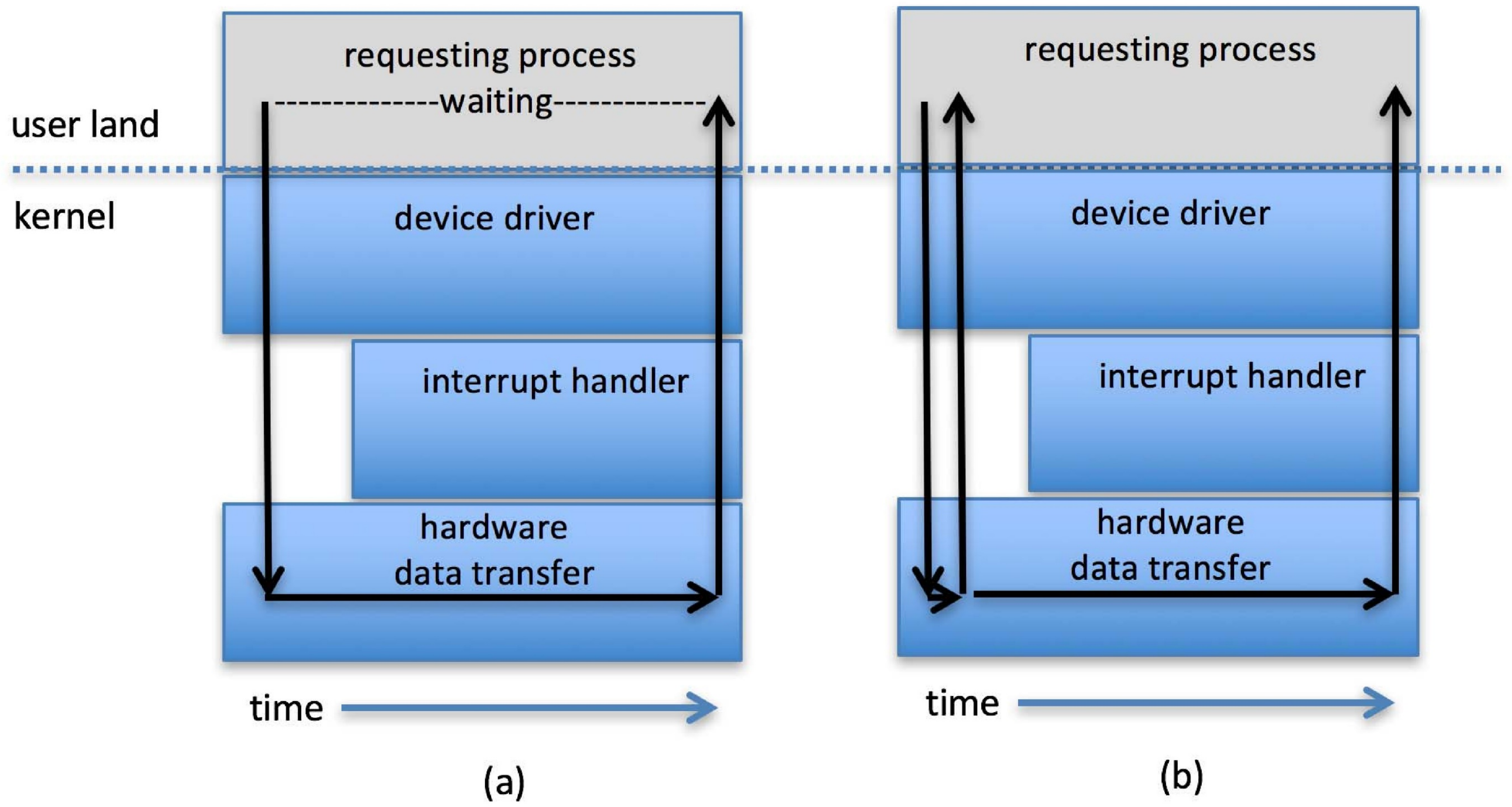
Block and Character Devices

- Block devices include disk drives
  - Commands include read, write, seek
  - Raw I/O, direct I/O, or file-system access
  - Memory-mapped file access possible
    - File mapped to virtual memory and clusters brought via demand paging
  - DMA
- Character devices include keyboards, mice, serial ports
  - Commands include **`get(), put()`**
  - Libraries layered on top allow line editing

Nonblocking and Asynchronous I/O

- Blocking - process suspended until I/O completed
  - Easy to use and understand
  - Insufficient for some needs
- Nonblocking - I/O call returns as much as available
  - User interface, data copy (buffered I/O)
  - Implemented via multi-threading
  - Returns quickly with count of bytes read or written
  - **`select()`** to find if data ready then **`read()`** or **`write()`** to transfer
- Asynchronous - process runs while I/O executes
  - Difficult to use
  - I/O subsystem signals process when I/O completed

Two I/O Methods



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 a time. Stores to disk happen one block 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 and Spooling

- Caching - faster device holding copy of data
  - Always just a copy
  - Key to performance
  - Sometimes combined with buffering
- Spooling - hold output for a device
  - If device can serve only one request at a time
  - e.g., Printing

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)

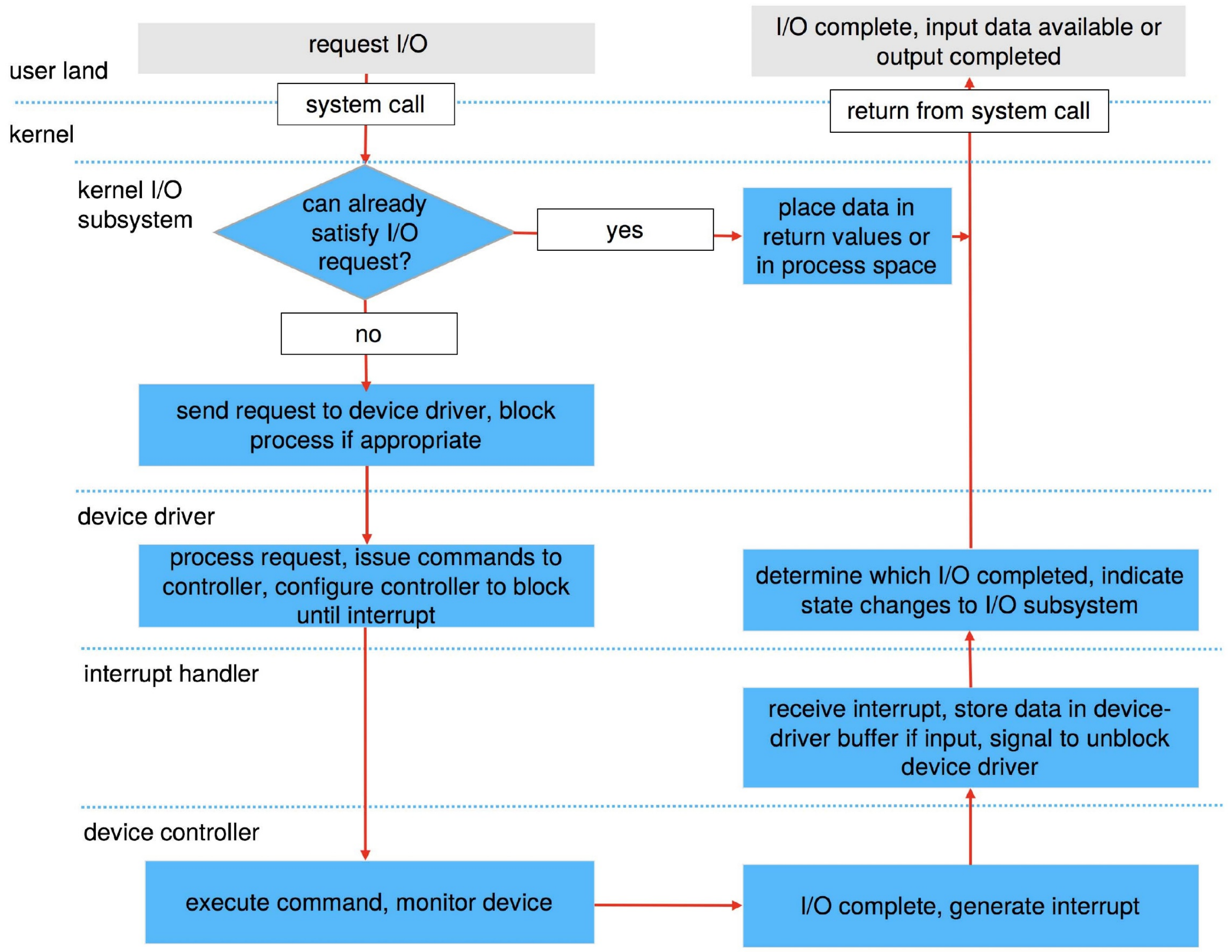
    If (block in memory) update value in memory

    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,
    1. OS tells the device driver to perform input
    2. Device driver tells the DMA controller what to do and blocks itself
    3. DMA controller transfers the data to the kernel buffer when it has all been retrieved from the device
    4. 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

Lifecycle of an I/O Request


request I/O
user land
system call
kernel
return from system call
I/O complete, input data available or output completed
kernel I/O subsystem
can already satisfy I/O request?
yes
place data in return values or in process space
no
send request to device driver, block process if appropriate
device driver
process request, issue commands to controller, configure controller to block until interrupt
determine which I/O completed, indicate state changes to I/O subsystem
interrupt handler
receive interrupt, store data in device-driver buffer if input, signal to unblock device driver
device controller
execute command, monitor device
I/O complete, generate interrupt

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 interrupts handling, which are slow
- Approaches to improve 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

Exit Slips

- Take 1-2 minutes to reflect on this lecture
- On a sheet of paper write:
  - One thing you learned in this lecture
  - One thing you didn’t understand

Next class

- We will discuss:
  - File Systems
- Reading assignment:
  - SGG: Ch. 13

Acknowledgment

- The slides are partially based on the ones from
  - The book site of *Operating System Concepts (Tenth Edition)*: http://os-book.com/