Storage Management

- File System Interface – Chapter 11
- File System Implementation – Chapter 12
- Mass Storage Structure – Chapter 10
- Input-Output Systems – Chapter 13

- NOTE: We are covering this topic in slightly different order than in textbook

Chapter 13: I/O Systems

- I/O hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- STREAMS
- Performance

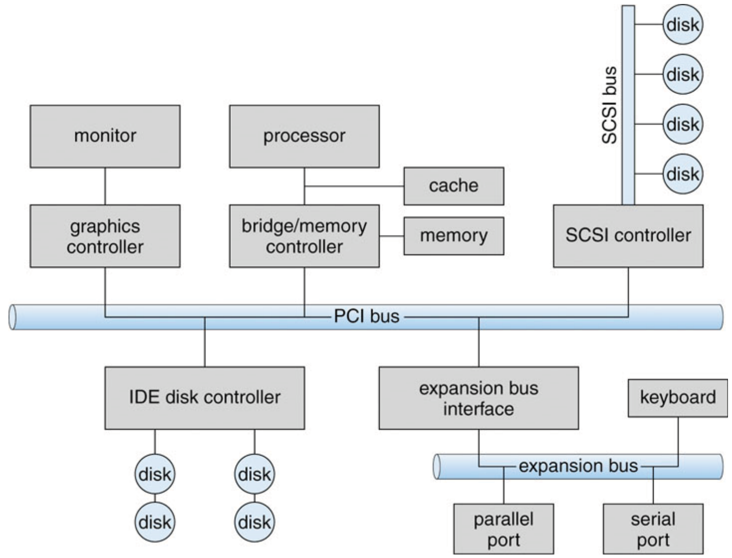
Overview

- 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

I/O Hardware

- Incredible variety of I/O devices
- Common concepts
  - I/O Port number uniquely identify devices
  - I/O Bus (daisy chain or shared direct access)
  - Device Controller (host adapter)
    - "smart" processor tailored to I/O peripheral
- I/O instructions to control devices
- Devices have addresses, used by
  - Direct I/O instructions (via port number)
  - Memory-mapped I/O
- Each device may have multiple registers
  - status, control, data-in, data -out

A Typical PC Bus Structure



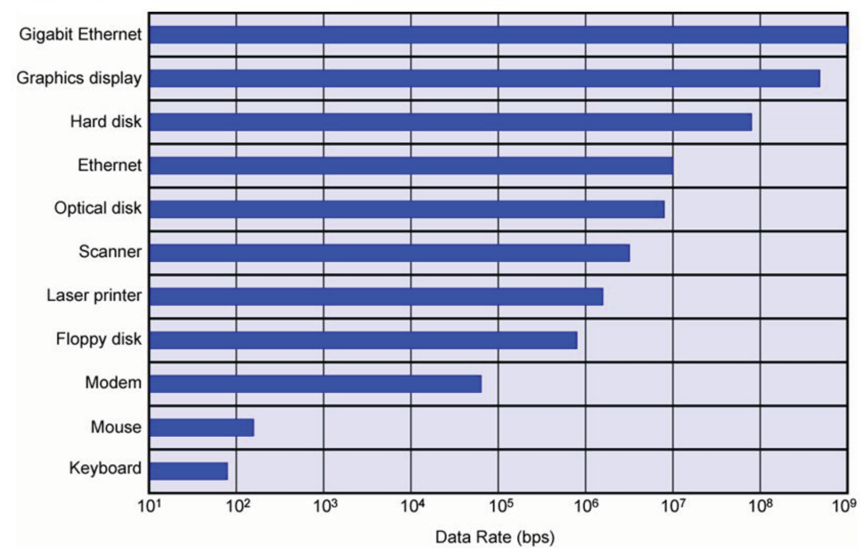
I/O Hardware (Cont'd)

- I/O instructions control devices
- Devices usually have registers where device driver places commands, addresses, and data to write, or read data from registers after command execution
  - Data-in register, data-out register, status register, control register
  - Typically 1-4 bytes, or FIFO buffer
- Devices have addresses, used by
  - Direct I/O instructions
  - **Memory-mapped I/O**
    - Device data and command registers mapped to processor address space
    - Especially for large address spaces (graphics)

Device I/O Port Locations on PCs (partial)

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)

Typical Device Data Rates



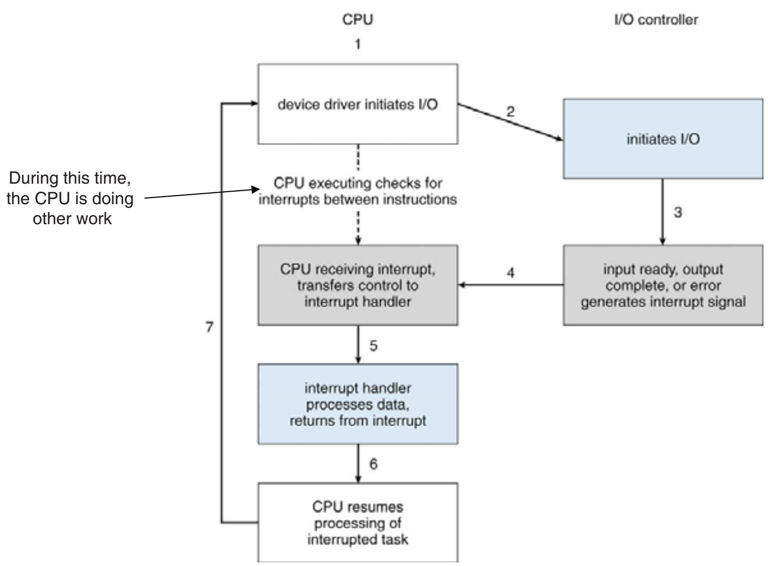
I/O Using Polling Technique

- For each byte of I/O
  1. Read busy bit from status register until 0
  2. Host sets read or write bit and if write copies data into data-out register
  3. Host sets command-ready bit
  4. Controller sets busy bit, executes transfer
  5. Controller clears busy bit, error bit, command-ready bit when transfer done
- Step 1 is **busy-wait** cycle to wait for I/O from device
  - Reasonable if device is fast
  - But inefficient if device slow
  - CPU switches to other tasks?
    - But if miss a cycle data will be lost

I/O Using Interrupt Techniques

- Polling can happen in 3 instruction cycles
  - Read status, logical-and to extract status bit, branch if not zero
  - How to be more efficient if excessive waiting involved?
- CPU **Interrupt-request line** triggered by I/O device
  - Checked by processor after each instruction
- **Interrupt handler** receives interrupts
  - **Maskable** to ignore or delay some interrupts
- **Interrupt vector** to dispatch interrupt to correct handler
  - Context switch at start and end of handling execution
  - Based on priority
  - Some **nonmaskable**

Interrupt-driven I/O Cycle



Intel Pentium Processor Interrupt-Vector Table

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

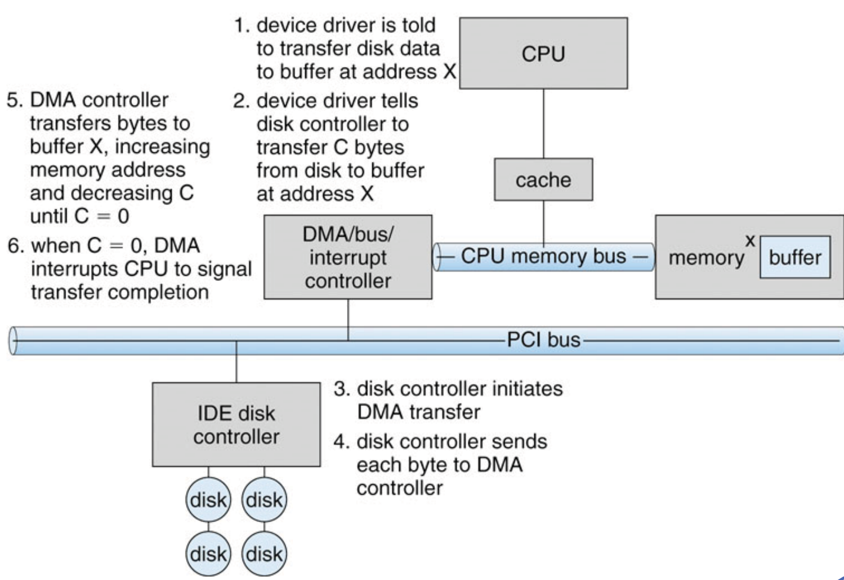
Interrupts (Cont'd)

- Interrupt mechanism also used for **exceptions**
  - Terminate process, system crash due to hardware error
- Page fault executes when memory access error
- System call executes via **trap** (software initiated interrupt) 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
  - may be frequent
  - must be fast

Direct Memory Access (DMA)

- Used to avoid programmed I/O for moving large amounts of data from device buffers to main memory
  - programmed I/O requires CPU to execute instructions to move each word of data
  - this occupies CPU time performing low-level activity
- Requires hardware capability (a DMA controller)
- Allows sharing system memory bus with the CPU so data movement can occur without intervention by the CPU
  - DMA "steals" bus cycles from CPU when CPU not using the memory bus
- O/S notified by interrupt when data movement is complete

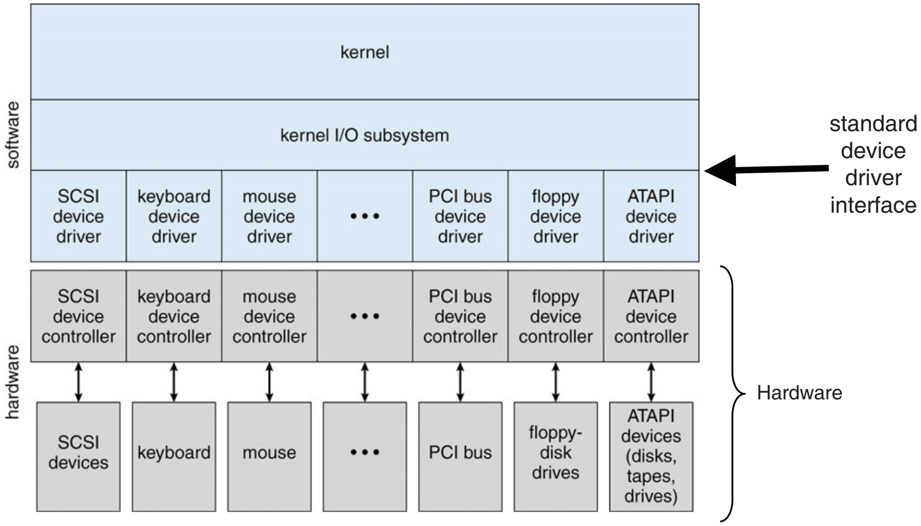
Six step process to perform DMA transfer



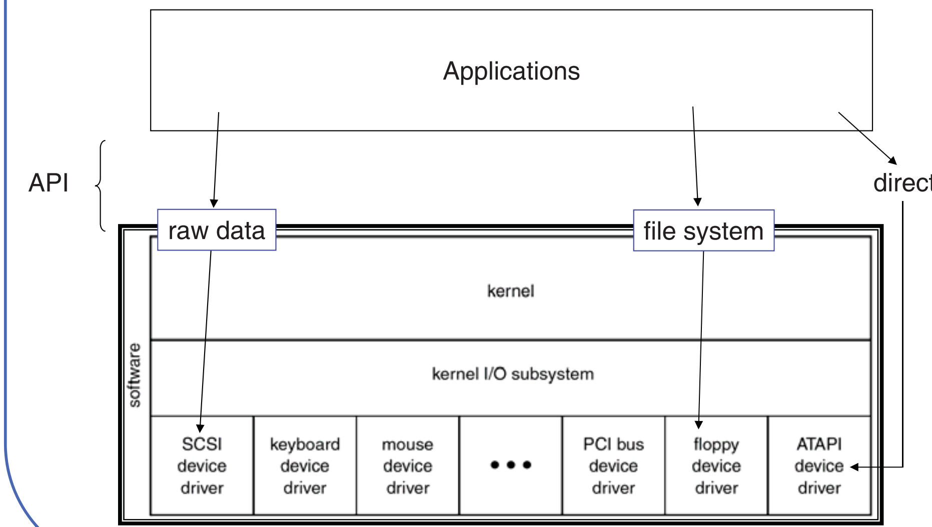
Application I/O Interface

- All application I/O has to be done via the O/S
- O/S provides abstraction and encapsulation of I/O operations
  - For applications, I/O system calls encapsulate device behaviors in generic device categories
    - hides differences between devices
    - provides "standard" API for I/O
  - For the O/S, device-driver layer hides differences among I/O controllers from kernel
- Devices vary in many dimensions
  - Character-stream or block
  - Sequential or random-access
  - Sharable or dedicated
  - Speed of operation
  - read-write, read only, or write only

A Kernel I/O Structure



Application I/O Interface via API



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'd)

- Differences between I/O devices handled by device drivers
- 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
  - Direct 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 allow line editing

Network Devices

- Varying enough from block and character to have own interface
- Unix (Linux) and Windows include socket interface
  - Separates network protocol implementation from network operation
  - Includes `select` functionality
    - Manage groups of sockets for server apps
    - Pick a socket with traffic waiting from group with no busy wait (using interrupts)
- Approaches to access network communications vary widely (pipes, FIFOs, STREAMS, queues, mailboxes)

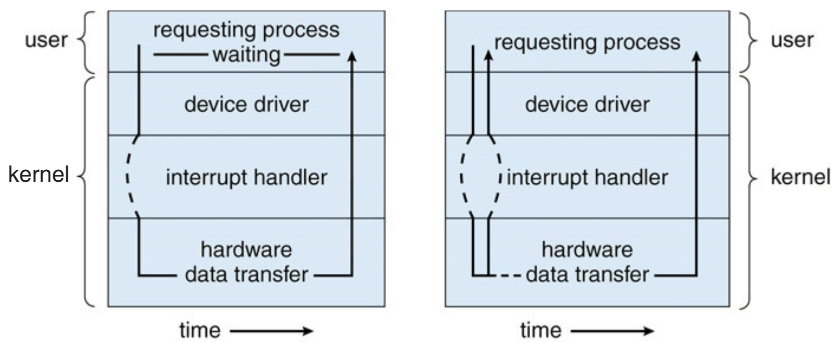
Clocks and Timers

- Hardware that provides current time, elapsed time, timed events
- Programmable interval time used for timings, periodic interrupts
  - no standard I/O abstractions across different O/S
  - example: `ioctl` (on UNIX) covers odd aspects of I/O such as clocks and timers by providing direct access to I/O device
- Interrupt schemes usually used

Blocking and Nonblocking I/O

- Blocking - process suspended until I/O completed
  - Also called synchronous I/O
  - Implemented via system calls
  - Easy to use and understand (from process' point of view)
  - Insufficient for some needs
- Nonblocking - I/O call immediately returns as much as available
  - User interface, data copy (buffered I/O)
  - Implemented via multi-threading
  - Examples – keyboard, mouse, socket select
- Asynchronous – submits I/O request and then process runs while I/O executes
  - I/O subsystem signals process when I/O completed
  - alternative to non-blocking I/O
  - no data available until I/O completes
  - More difficult to use

I/O Methods



Synchronous

Asynchronous

Kernel I/O Subsystem Services

- Improve efficiency by scheduling waiting I/O requests
  - May provide I/O request ordering via per-device queue
  - Some O/S try fairness (equal priority), while others apply some sort of priorities to I/O
- Improve data transfer efficiency by buffering in O/S protected storage area of main memory
  - A buffer holds the only copy of data temporarily between the source of data and its destination
  - Used to cope with device speed mismatch
  - Used to cope with device transfer size mismatch

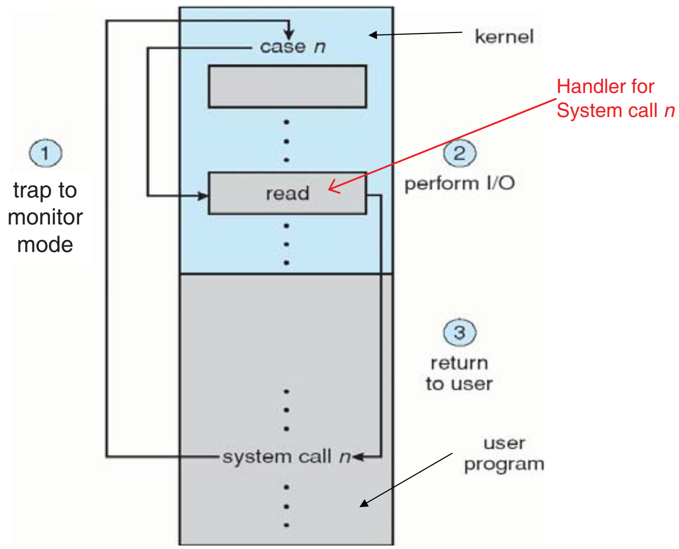
Kernel I/O Subsystem Services (cont'd)

- Caching – holding a working copy of data in a storage area of faster access than where the data is permanently stored
  - Caching used to increase performance
  - Different than buffering discussed on previous slide
  - Buffering and caching are key to improving I/O overall performance
- Spooling – term for buffer holding output data for an I/O device
  - If device can serve only one request at a time
  - i.e., Printing
- Device reservation - provides exclusive access to a device
  - System calls for allocation and deallocation
  - Watch out for deadlock

I/O Protection

- User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions
  - All I/O device instructions defined to be privileged
    - cannot be executed in user mode
  - User process I/O must be performed via system calls
    - Memory-mapped and I/O port memory locations must be protected too

Use of a System Call to Protect I/O



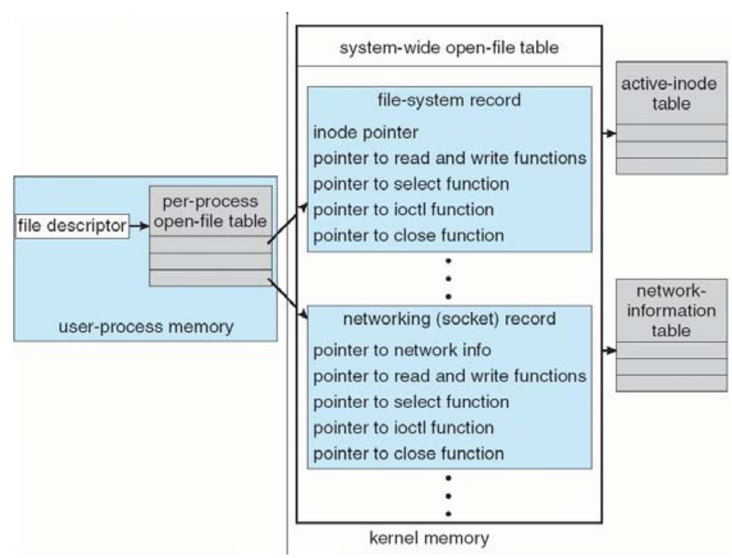
Error Handling

- OS can try to recover from storage or network read failure, device unavailable, transient write failures
  - data read/write retry
  - network resend
- Return an error number or code when I/O request fails and recovery procedure fails
  - notify user
- System error logs hold problem reports

Kernel Data Structures

- Kernel keeps state information for I/O components, including open file tables, network connections, character device state
- Many complex data structures needed to track buffers, memory allocation, “dirty” blocks
- Data structures can be encapsulated in “objects” associated with I/O operations
  - allows uniform handling of the data structures
  - Some use object-oriented methods to encapsulate I/O requests in a message passed between kernel and I/O drivers (Windows NT/2000/XP)

UNIX I/O Kernel Structures



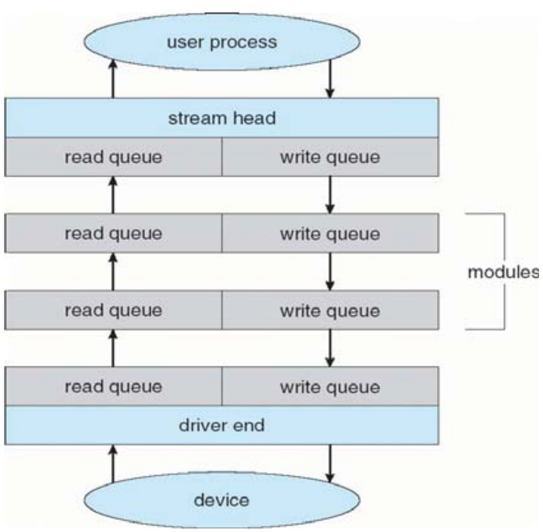
Custom I/O Handling – Unix STREAMs

- **STREAM** – a full-duplex communication channel between a user-level process and a device in Unix System V and later versions
- A STREAM can be configured dynamically by chaining together I/O modules that process data in the stream
- A STREAM consists of:
  - STREAM head interfaces with the user process
  - driver end interfaces with the device
  - zero or more STREAM modules between them.

STREAMs

- Each module contains a **read queue** and a **write queue**
  - can use flow control to prevent buffer overruns
- Message passing is used to communicate between queues
- Using STREAMs, the interface to specific I/O devices can be customized
  - somewhat like building a device driver from plug-together pieces
  - similar to Java language I/O streams

The STREAMs Structure

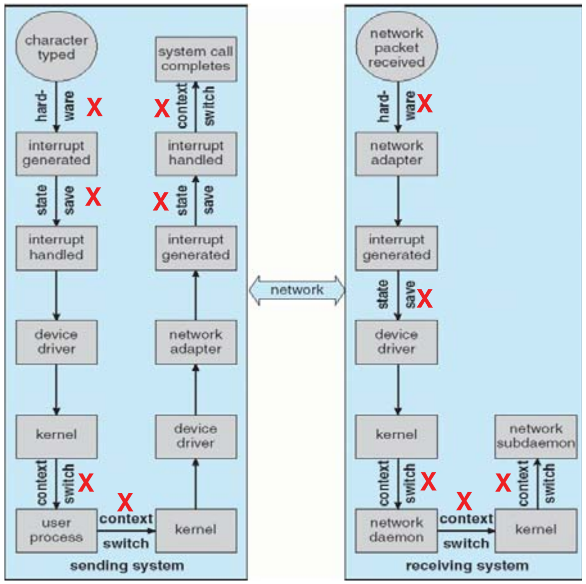


Performance

- I/O a major factor in system performance for many applications
- Can be significant system overhead to perform I/O
  - Demands on CPU to execute device driver, kernel I/O code
  - Context switches due to interrupts
  - Servicing multiple requests
  - Data copying
  - Network traffic especially stressful (example next slide)

Intercomputer communications

- Example of how network traffic can stress performance
- diagram shows full duplex operation of a terminal (telnet) application
  - each character typed
- X marks location of significant overhead operations



Improving Performance

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling (when it is more efficient)
- Use DMA
- Balance CPU, memory, bus, and I/O performance for highest throughput

Device-Functionality Progression