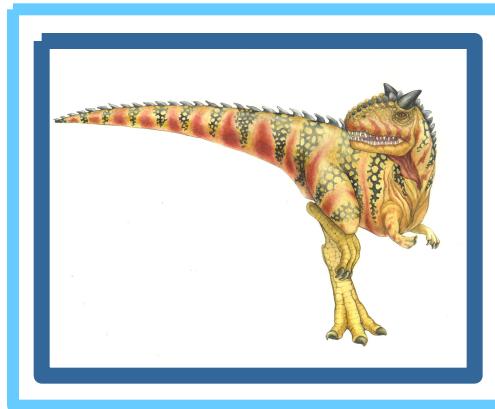


# Chapter 13: I/O Systems



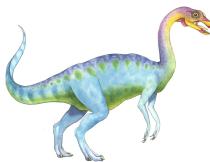


# Chapter 13: I/O Systems

---

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





# Objectives

---

- n Explore the structure of an operating system's I/O subsystem
- n Discuss the principles of I/O hardware and its complexity
- n Provide details of the performance aspects of I/O hardware and software





# Overview

---

- n 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
- n Ports, busses, device controllers connect to various devices
- n **Device drivers** encapsulate device details
  - | Present uniform device-access interface to I/O subsystem





# I/O Hardware

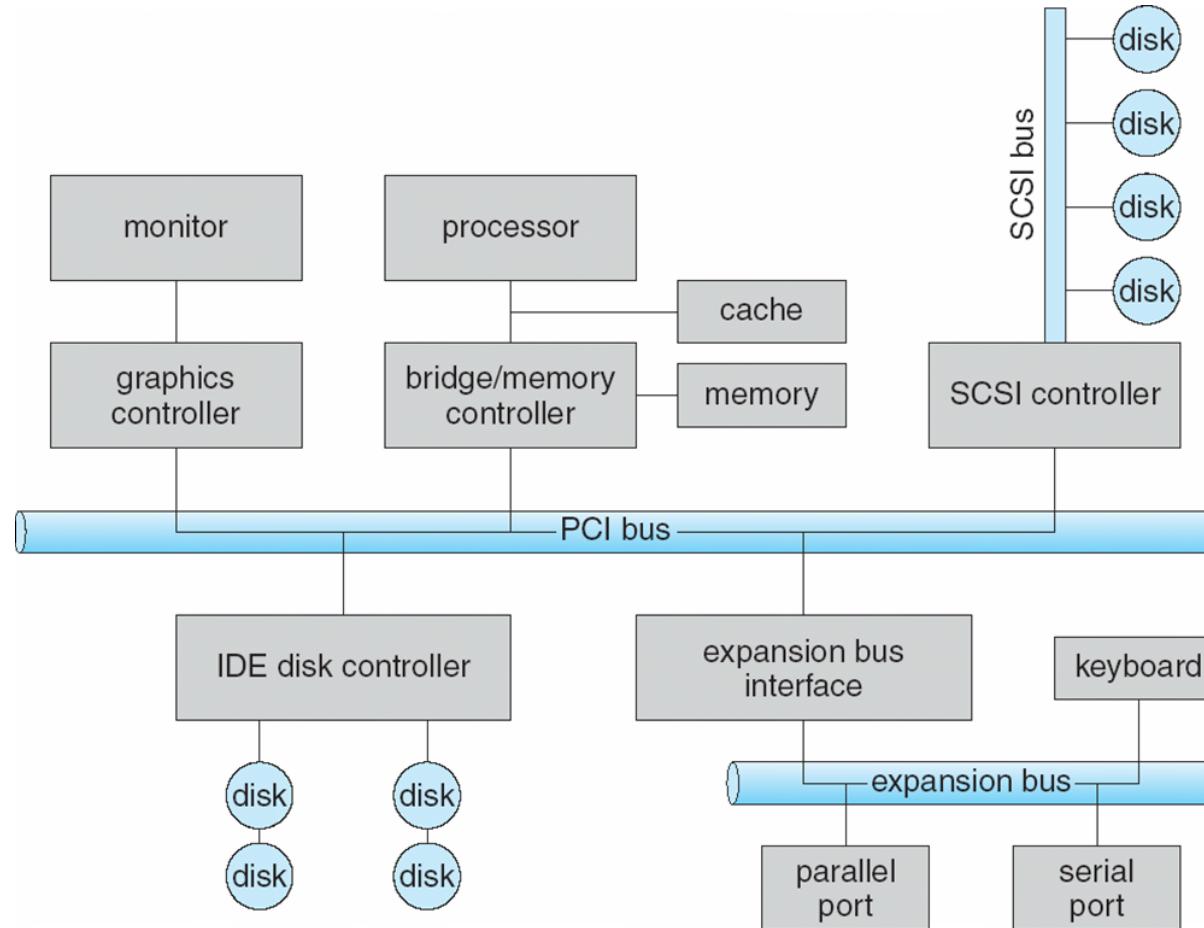
---

- n Incredible variety of I/O devices
  - | Storage
  - | Transmission
  - | Human-interface
- n Common concepts – signals from I/O devices interface with computer
  - | **Port** – connection point for device
  - | **Bus - daisy chain** or shared direct access
    - ▶ **PCI** bus common in PCs and servers, PCI Express (**PCle**)
    - ▶ **expansion bus** connects relatively slow devices
  - | **Controller (host adapter)** – electronics that operate port, bus, device
    - ▶ Sometimes integrated
    - ▶ Sometimes separate circuit board (host adapter)
    - ▶ Contains processor, microcode, private memory, bus controller, etc
      - Some talk to per-device controller with bus controller, microcode, memory, etc





# A Typical PC Bus Structure





# I/O Hardware (Cont.)

- n I/O instructions control devices
- n 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
- n 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)





# Polling

---

- n 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
- n 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 overwritten / lost





# Interrupts

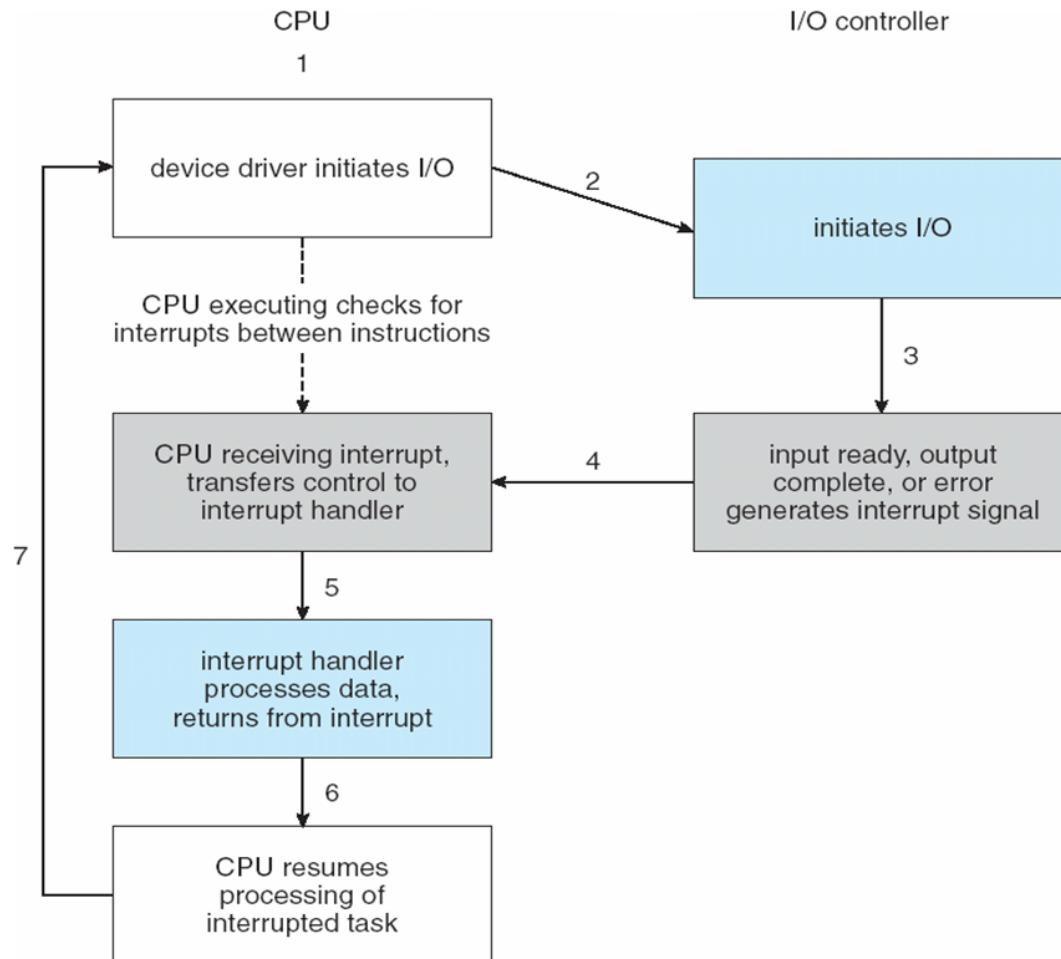
---

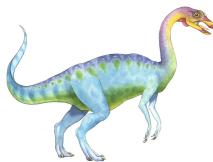
- n Polling can happen in 3 instruction cycles
  - | Read status, logical-and to extract status bit, branch if not zero
  - | How to be more efficient if non-zero infrequently?
- n CPU **Interrupt-request line** triggered by I/O device
  - | Checked by processor after each instruction
- n **Interrupt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





# Interrupt-Driven I/O Cycle





# Intel Pentium Processor Event-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.)

---

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



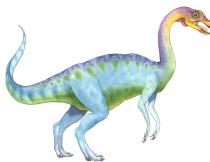


# Direct Memory Access

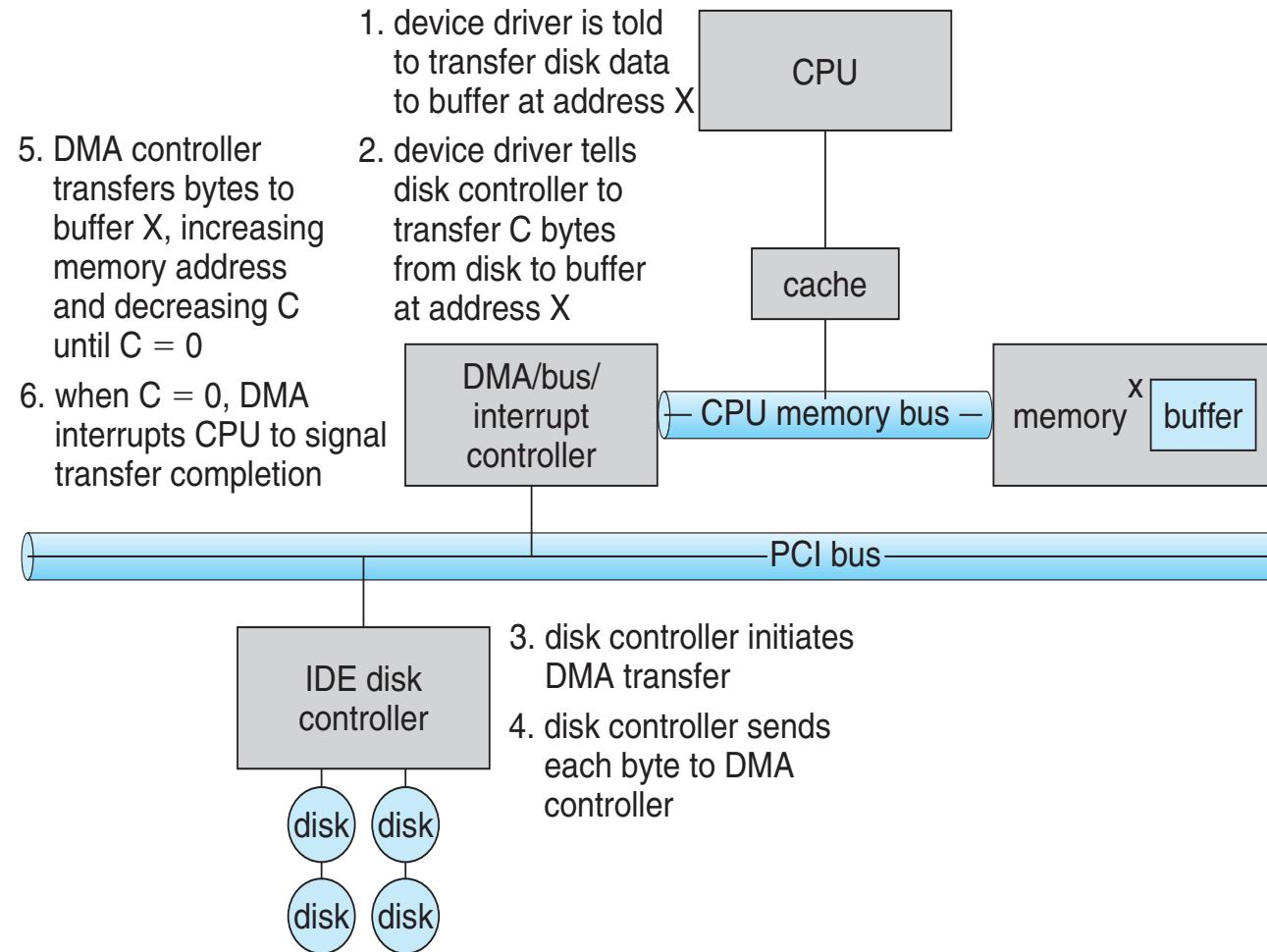
---

- n Used to avoid **programmed I/O** (one byte at a time) for large data movement
- n Requires **DMA** controller
- n Bypasses CPU to transfer data directly between I/O device and memory
- n OS writes DMA command block into memory
  - | Source and destination addresses
  - | Read or write mode
  - | Count of bytes
  - | Writes location of command block to DMA controller
  - | Bus mastering of DMA controller – grabs bus from CPU
    - ▶ **Cycle stealing** from CPU but still much more efficient
  - | When done, interrupts to signal completion
- n Version that is aware of virtual addresses can be even more efficient - **DVMA**





# Six Step Process to Perform DMA Transfer





# Application I/O Interface

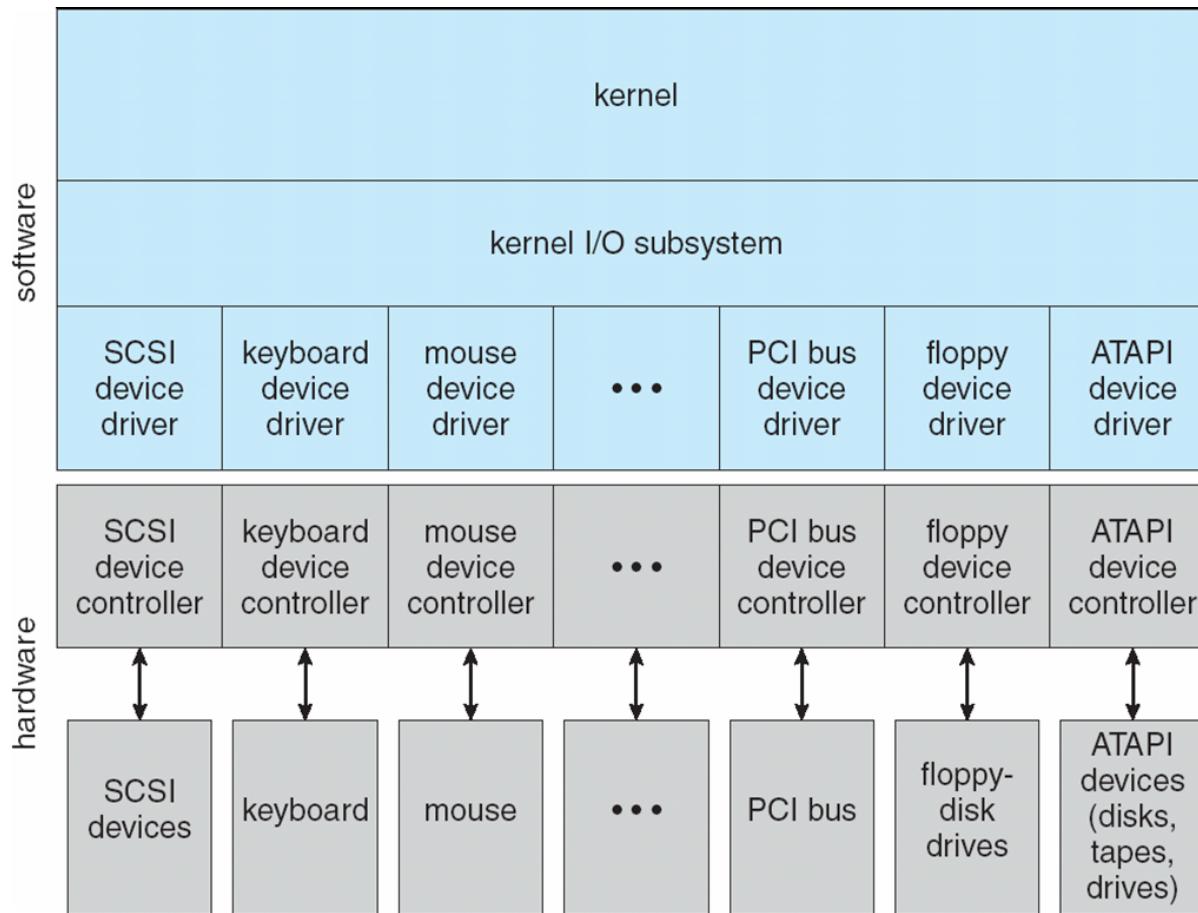
---

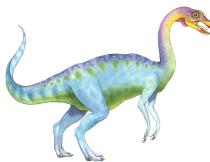
- n I/O system calls encapsulate device behaviors in generic classes
- n Device-driver layer hides differences among I/O controllers from kernel
- n New devices talking already-implemented protocols need no extra work
- n Each OS has its own I/O subsystem structures and device driver frameworks
- n Devices vary in many dimensions
  - | **Character-stream** or **block**
  - | **Sequential** or **random-access**
  - | **Synchronous** or **asynchronous** (or both)
  - | **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.)

---

- n Subtleties of devices handled by device drivers
- n Broadly I/O devices can be grouped by the OS into
  - | Block I/O
  - | Character I/O (Stream)
  - | Memory-mapped file access
  - | Network sockets
- n 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

---

- n 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
- n Character devices include keyboards, mice, serial ports
  - | Commands include `get()`, `put()`
  - | Libraries layered on top allow line editing



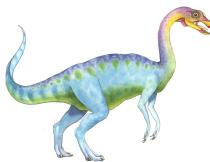


# Network Devices

---

- n Varying enough from block and character to have own interface
- n Linux, Unix, Windows and many others include **socket** interface
  - | Separates network protocol from network operation
  - | Includes **select()** functionality
- n Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)



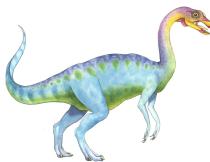


# Clocks and Timers

---

- n Provide current time, elapsed time, timer
- n Normal resolution about 1/60 second
- n Some systems provide higher-resolution timers
- n **Programmable interval timer** used for timings, periodic interrupts
- n **ioctl()** (on UNIX) covers odd aspects of I/O such as clocks and timers



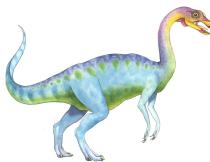


# Nonblocking and Asynchronous I/O

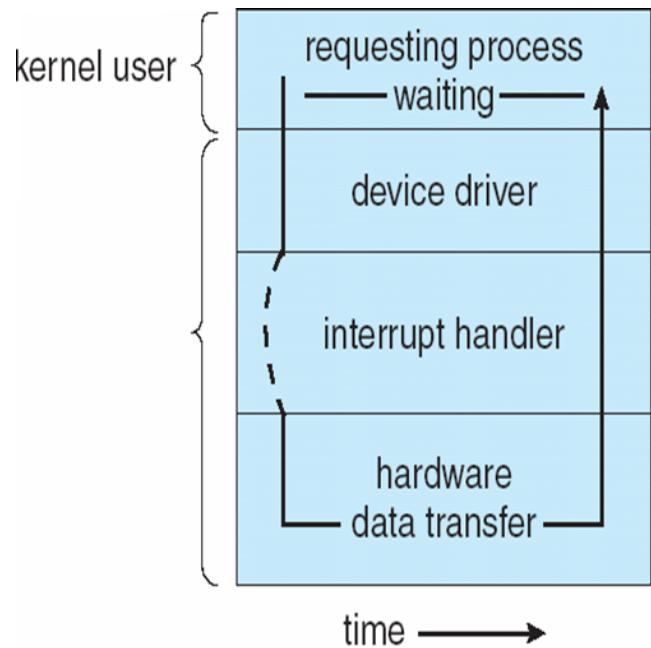
---

- n **Blocking** - process suspended until I/O completed
  - | Easy to use and understand
  - | Insufficient for some needs
- n **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
- n **Asynchronous** - process runs while I/O executes
  - | Difficult to use
  - | I/O subsystem signals process when I/O completed



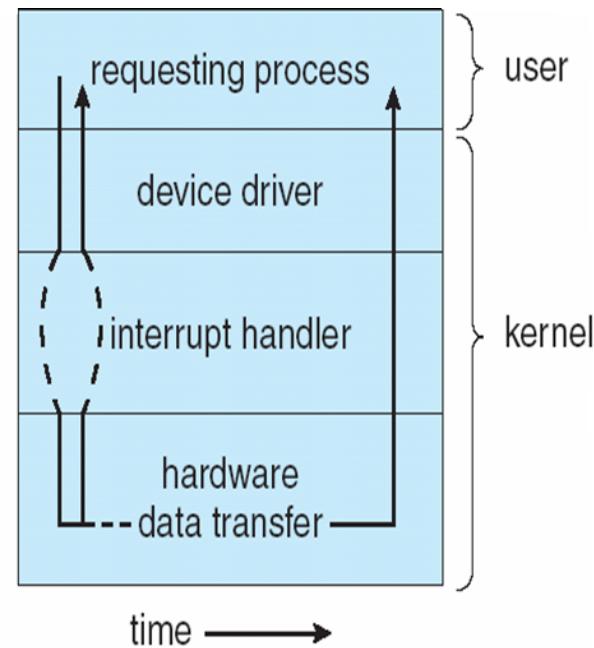


# Two I/O Methods



(a)

Synchronous



(b)

Asynchronous





# Vectored I/O

- n Vectored I/O allows one system call to perform multiple I/O operations
- n For example, Unix `readve()` accepts a vector of multiple buffers to read into or write from
- n This scatter-gather method better than multiple individual I/O calls
  - | Decreases context switching and system call overhead
  - | Some versions provide atomicity
    - ▶ Avoid for example worry about multiple threads changing data as reads / writes occurring





# Kernel I/O Subsystem

---

## n Scheduling

- | Some I/O request ordering via per-device queue
- | Some OSs try fairness
- | Some implement Quality Of Service (i.e. IPQOS)

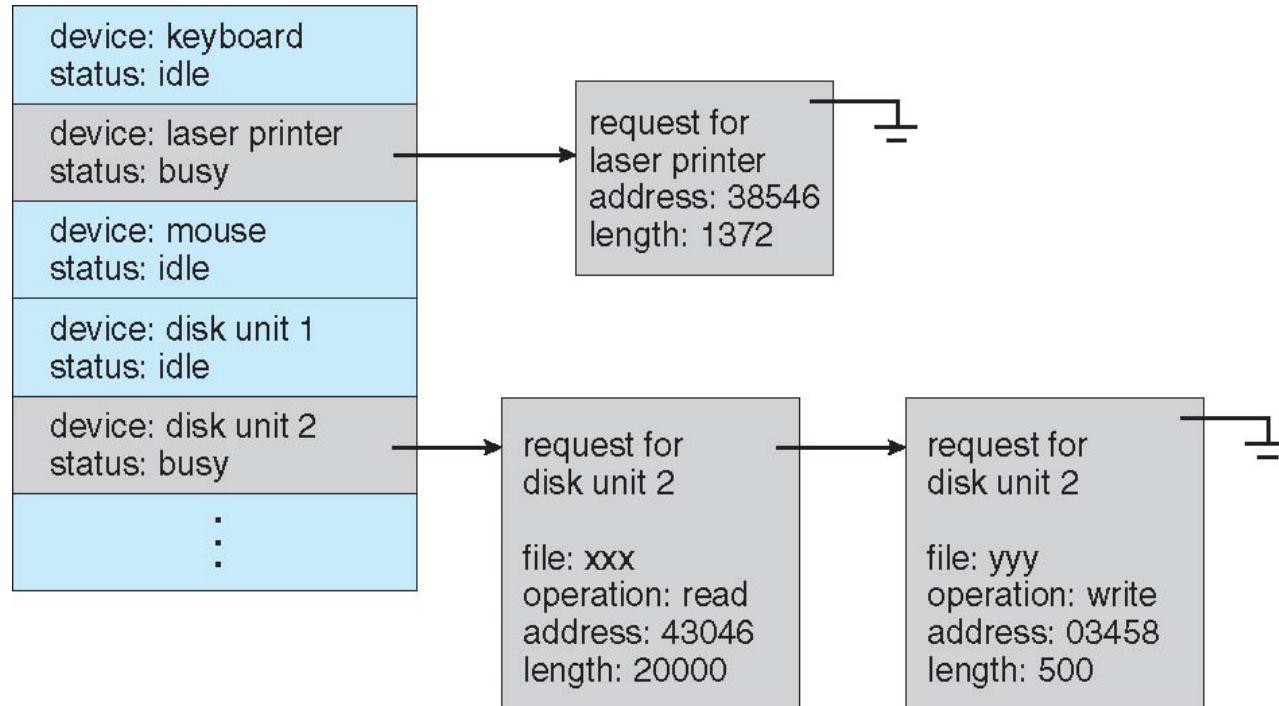
## n **Buffering** - store data in memory while transferring between devices

- | To cope with device speed mismatch
- | To cope with device transfer size mismatch
- | To maintain “copy semantics”
- | **Double buffering** – two copies of the data
  - ▶ Kernel and user
  - ▶ Varying sizes
  - ▶ Full / being processed and not-full / being used
  - ▶ Copy-on-write can be used for efficiency in some cases



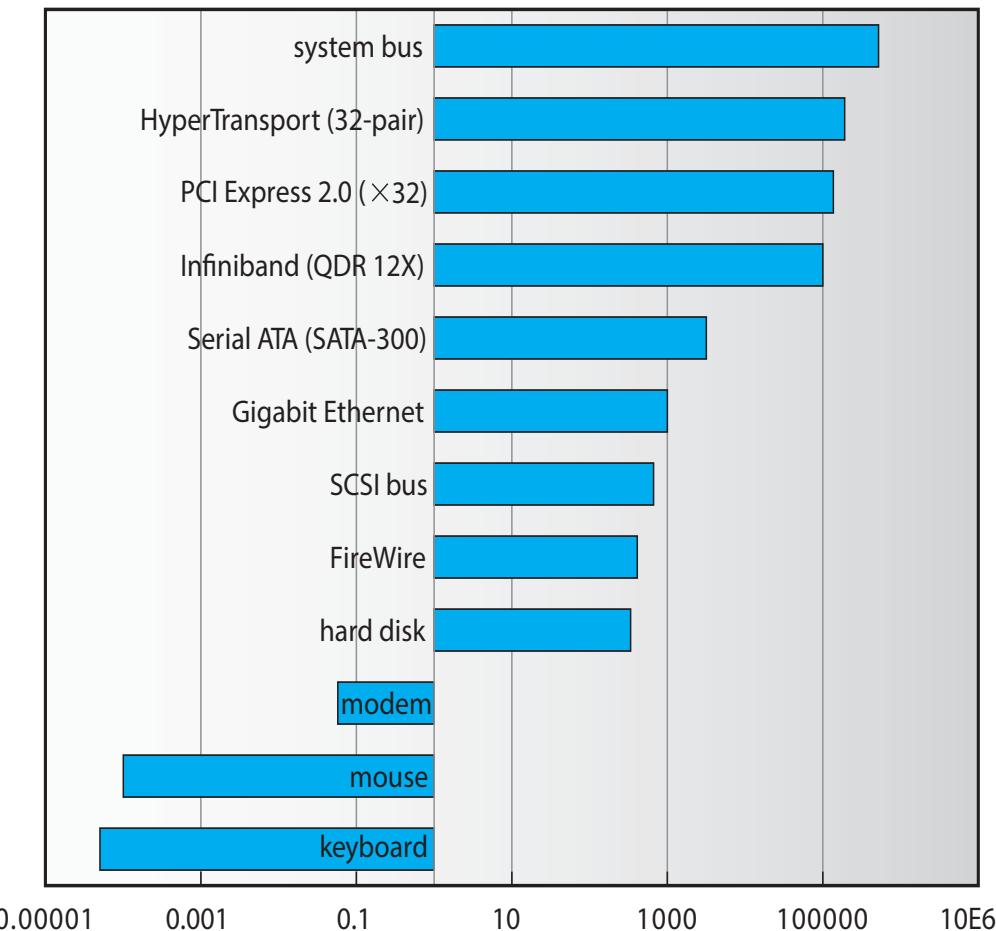


# Device-status Table





# Sun Enterprise 6000 Device-Transfer Rates





# Kernel I/O Subsystem

---

- n **Caching** - faster device holding copy of data
  - | Always just a copy
  - | Key to performance
  - | Sometimes combined with buffering
- n **Spooling** - hold output for a device
  - | If device can serve only one request at a time
  - | i.e., Printing
- n **Device reservation** - provides exclusive access to a device
  - | System calls for allocation and de-allocation
  - | Watch out for deadlock

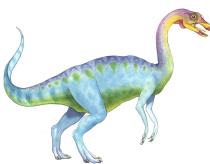




# Error Handling

- n OS can recover from disk read, device unavailable, transient write failures
  - | Retry a read or write, for example
  - | Some systems more advanced – Solaris FMA, AIX
    - ▶ Track error frequencies, stop using device with increasing frequency of retry-able errors
- n Most return an error number or code when I/O request fails
- n System error logs hold problem reports





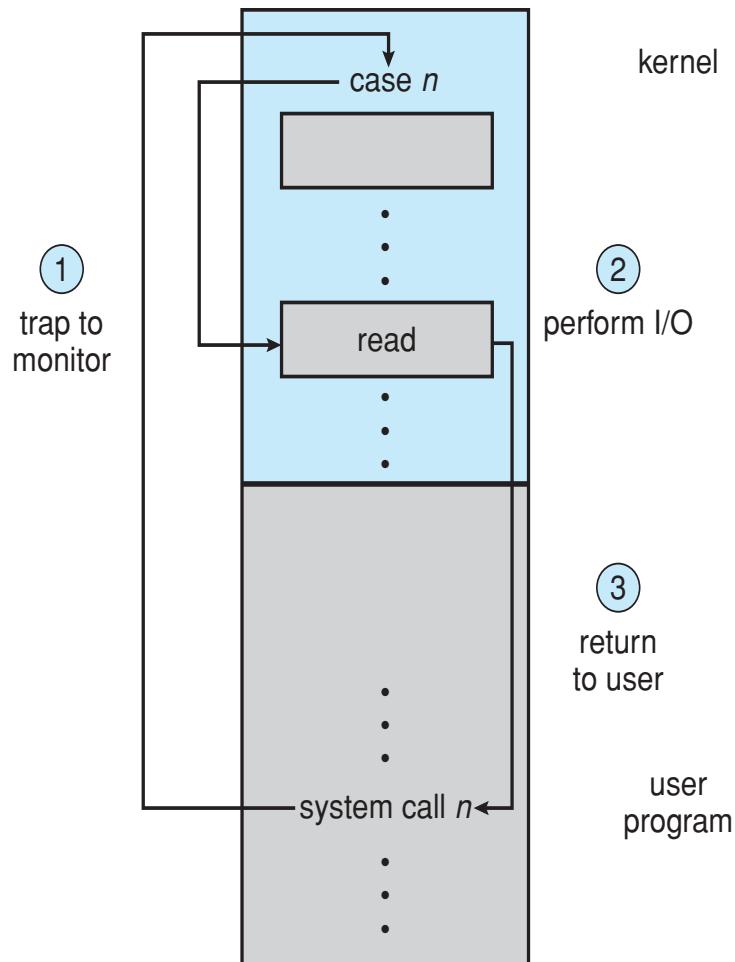
# I/O Protection

- n User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions
  - | All I/O instructions defined to be privileged
  - | 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 Perform I/O





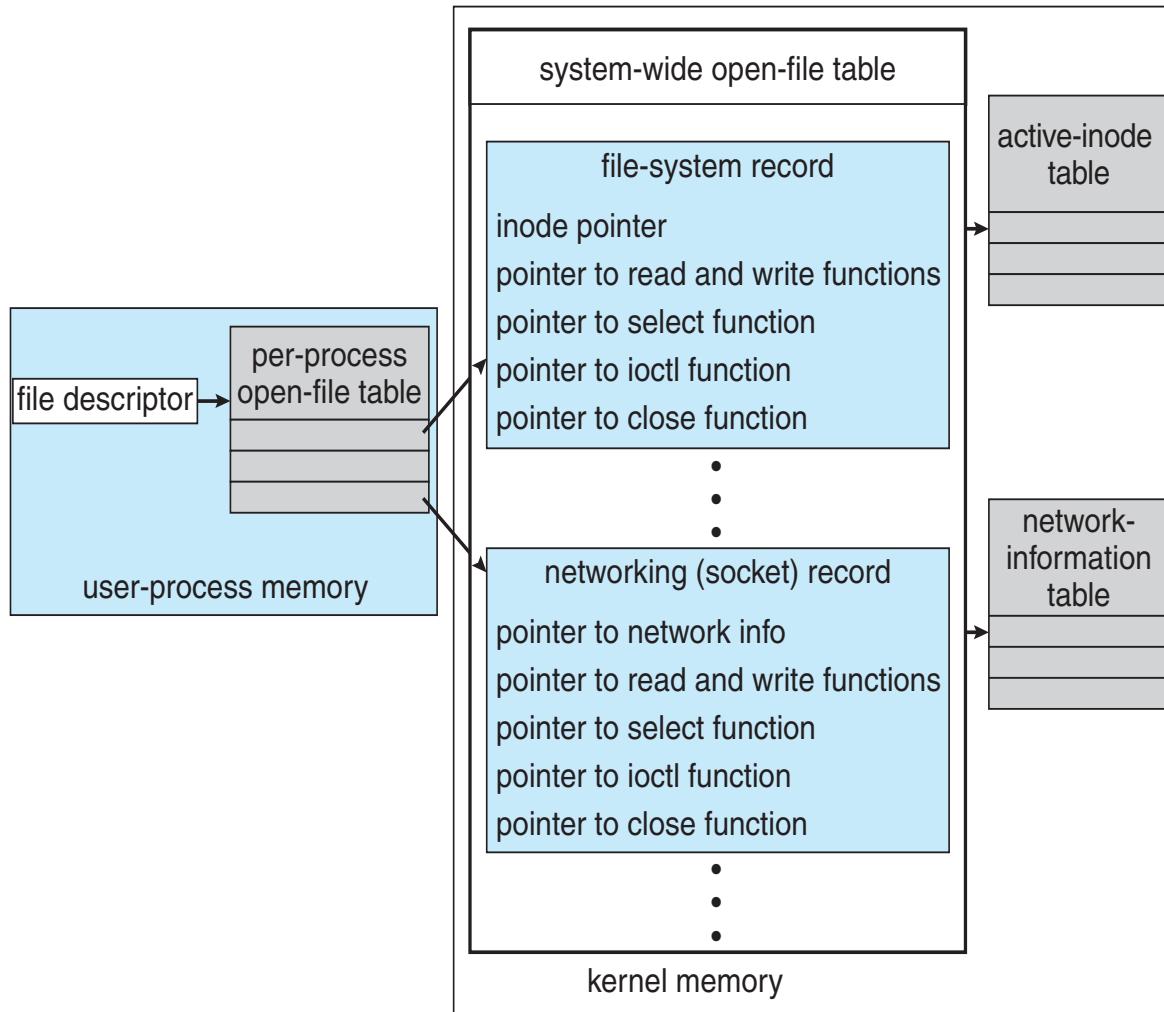
# Kernel Data Structures

- n Kernel keeps state info for I/O components, including open file tables, network connections, character device state
- n Many, many complex data structures to track buffers, memory allocation, “dirty” blocks
- n Some use object-oriented methods and message passing to implement I/O
  - | Windows uses message passing
    - ▶ Message with I/O information passed from user mode into kernel
    - ▶ Message modified as it flows through to device driver and back to process
    - ▶ Pros / cons?





# UNIX I/O Kernel Structure

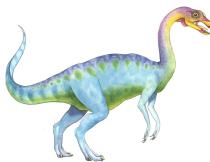




# Power Management

- n Not strictly domain of I/O, but much is I/O related
- n Computers and devices use electricity, generate heat, frequently require cooling
- n OSes can help manage and improve use
  - | Cloud computing environments move virtual machines between servers
    - ▶ Can end up evacuating whole systems and shutting them down
- n Mobile computing has power management as first class OS aspect





# Power Management (Cont.)

- n For example, Android implements
  - | Component-level power management
    - ▶ Understands relationship between components
    - ▶ Build device tree representing physical device topology
    - ▶ System bus -> I/O subsystem -> {flash, USB storage}
    - ▶ Device driver tracks state of device, whether in use
    - ▶ Unused component – turn it off
    - ▶ All devices in tree branch unused – turn off branch
  - | Wake locks – like other locks but prevent sleep of device when lock is held
  - | Power collapse – put a device into very deep sleep
    - ▶ Marginal power use
    - ▶ Only awake enough to respond to external stimuli (button press, incoming call)





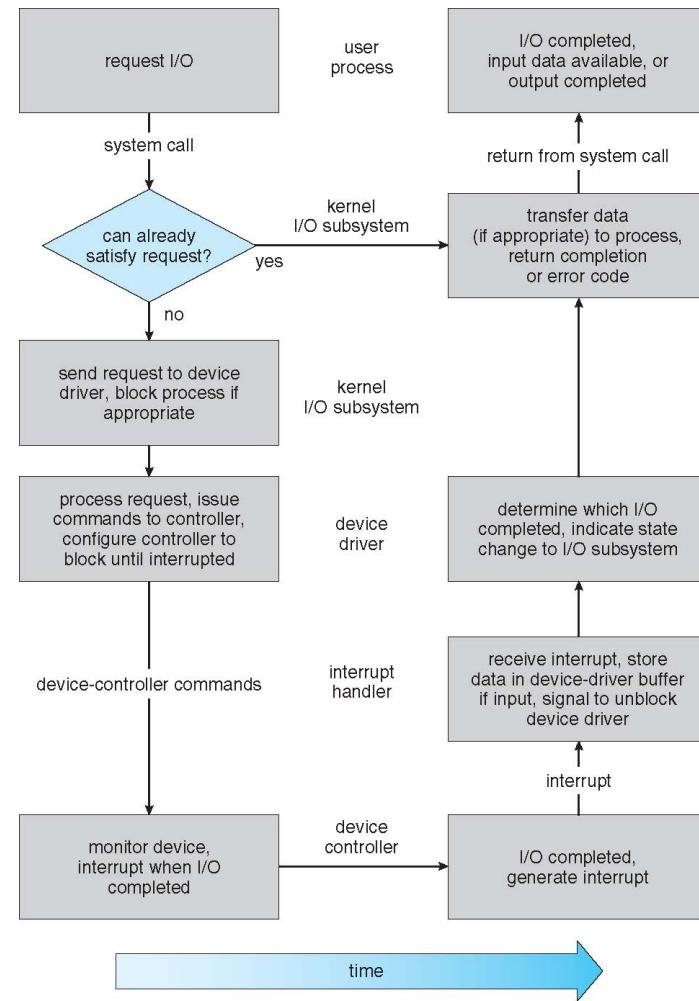
# I/O Requests to Hardware Operations

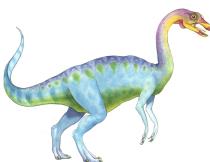
- n Consider reading a file from disk for a process:
  - | Determine device holding file
  - | Translate name to device representation
  - | Physically read data from disk into buffer
  - | Make data available to requesting process
  - | Return control to process





# Life Cycle of An I/O Request





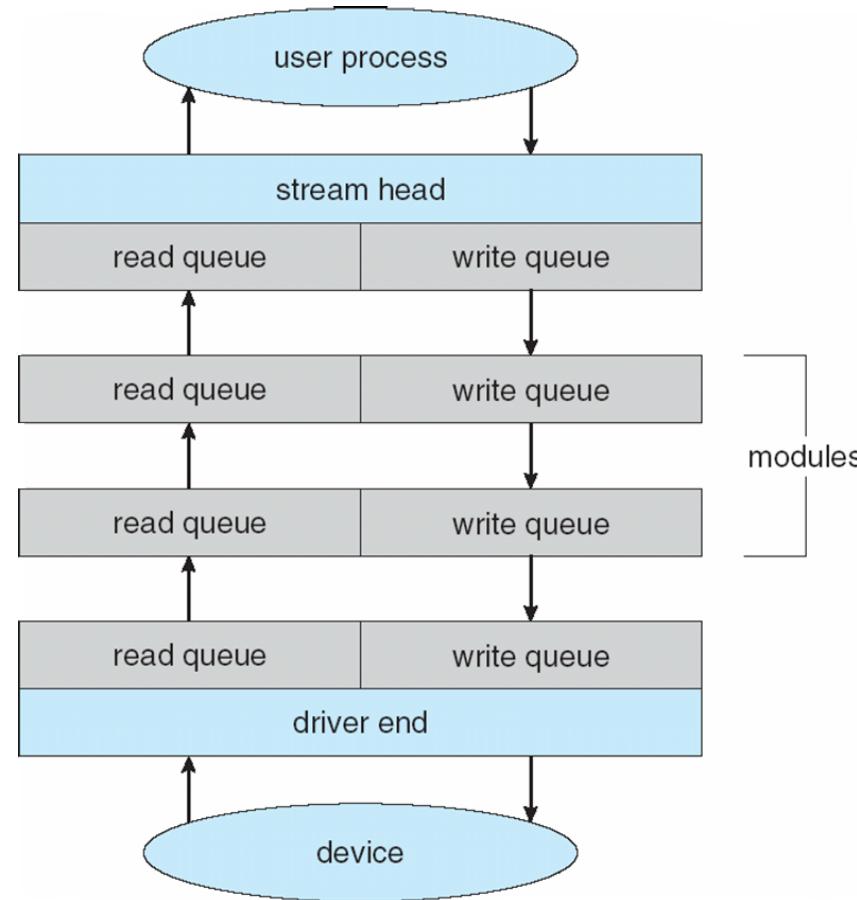
# STREAMS

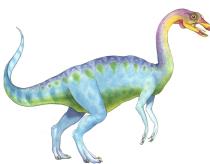
- n **STREAM** – a full-duplex communication channel between a user-level process and a device in Unix System V and beyond
- n A STREAM consists of:
  - | STREAM head interfaces with the user process
  - | driver end interfaces with the device
  - | zero or more STREAM modules between them
- n Each module contains a **read queue** and a **write queue**
- n Message passing is used to communicate between queues
  - | **Flow control** option to indicate available or busy
- n Asynchronous internally, synchronous where user process communicates with stream head





# The STREAMS Structure



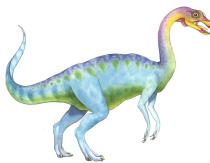


# Performance

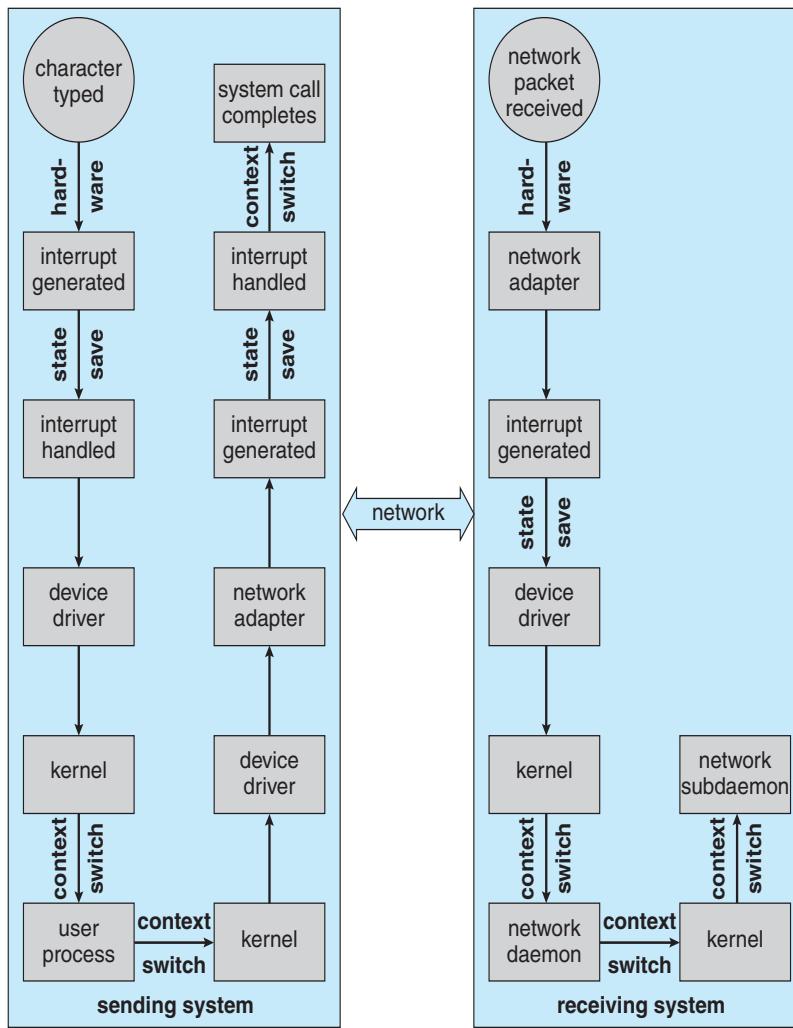
---

- n I/O a major factor in system performance:
  - | Demands CPU to execute device driver, kernel I/O code
  - | Context switches due to interrupts
  - | Data copying
  - | Network traffic especially stressful





# Intercomputer Communications





# Improving Performance

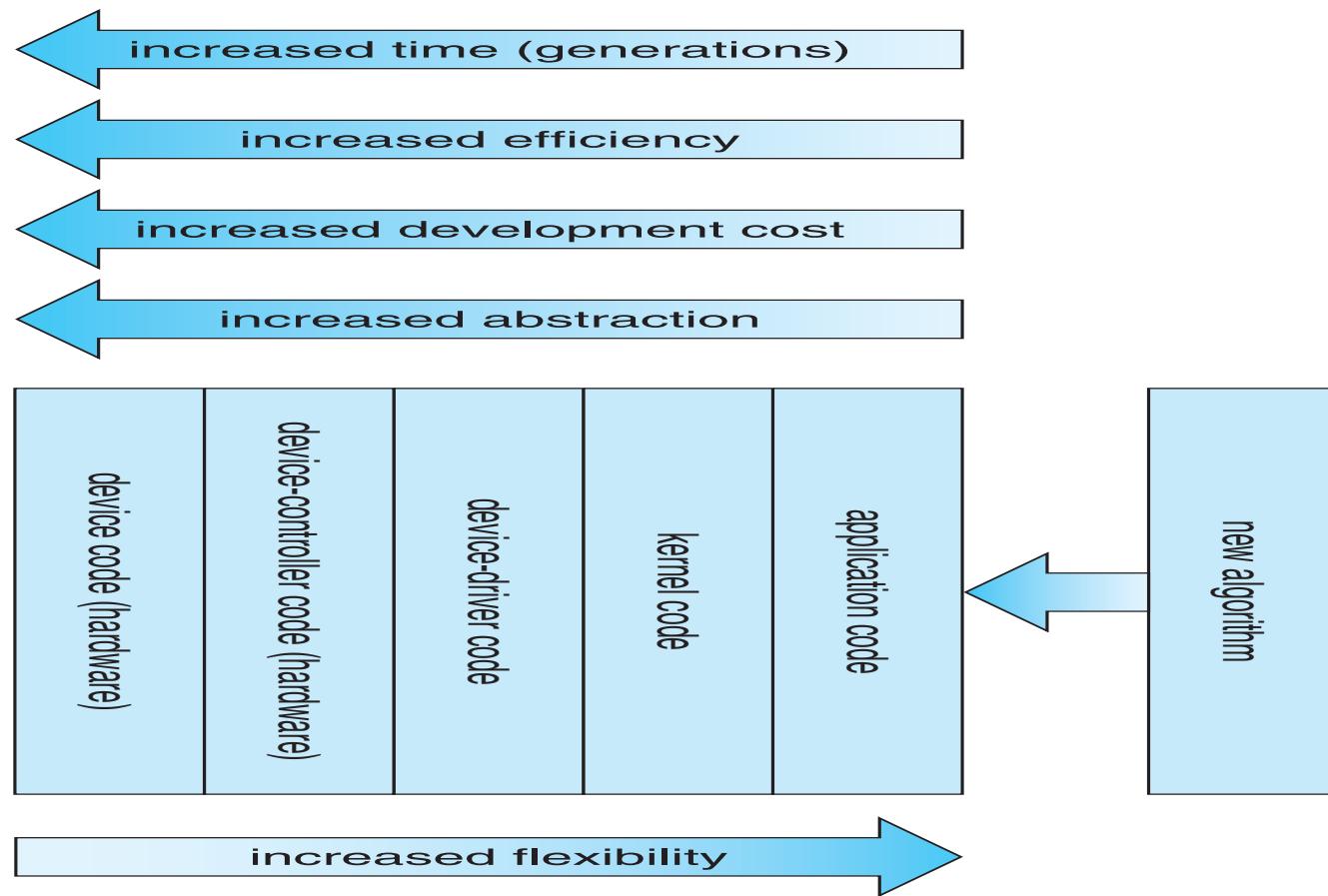
---

- n Reduce number of context switches
- n Reduce data copying
- n Reduce interrupts by using large transfers, smart controllers, polling
- n Use DMA
- n Use smarter hardware devices
- n Balance CPU, memory, bus, and I/O performance for highest throughput
- n Move user-mode processes / daemons to kernel threads





# Device-Functionality Progression



# End of Chapter 13

