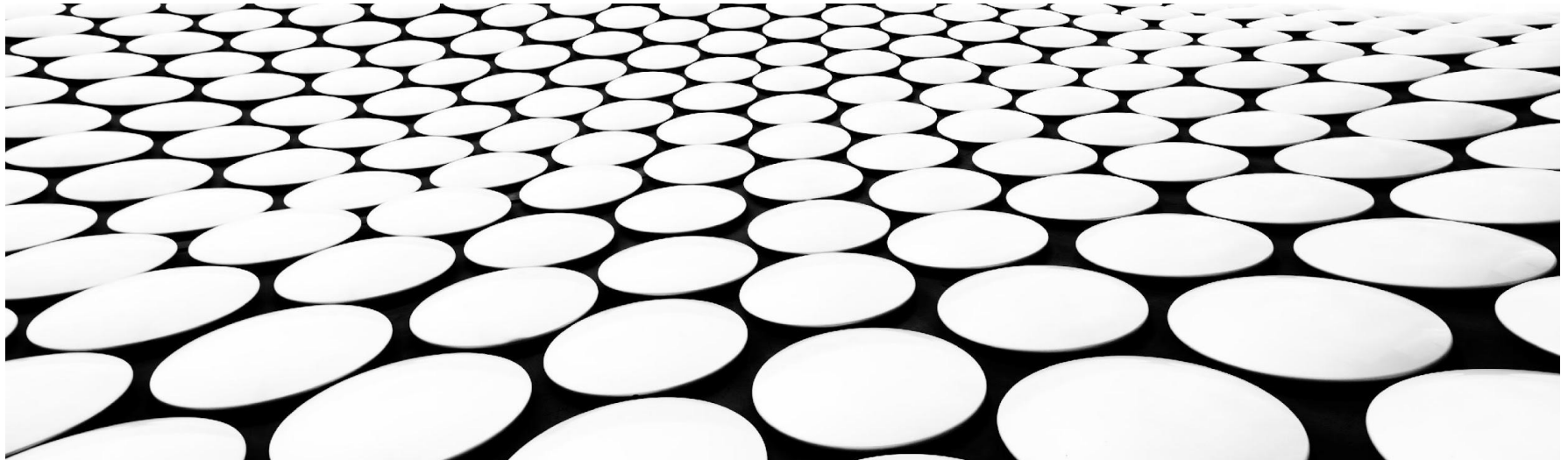


LECTURE 20: I/O Systems

DR. ARIJIT ROY

COMPUTER SCIENCE AND ENGINEERING GROUP

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY



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

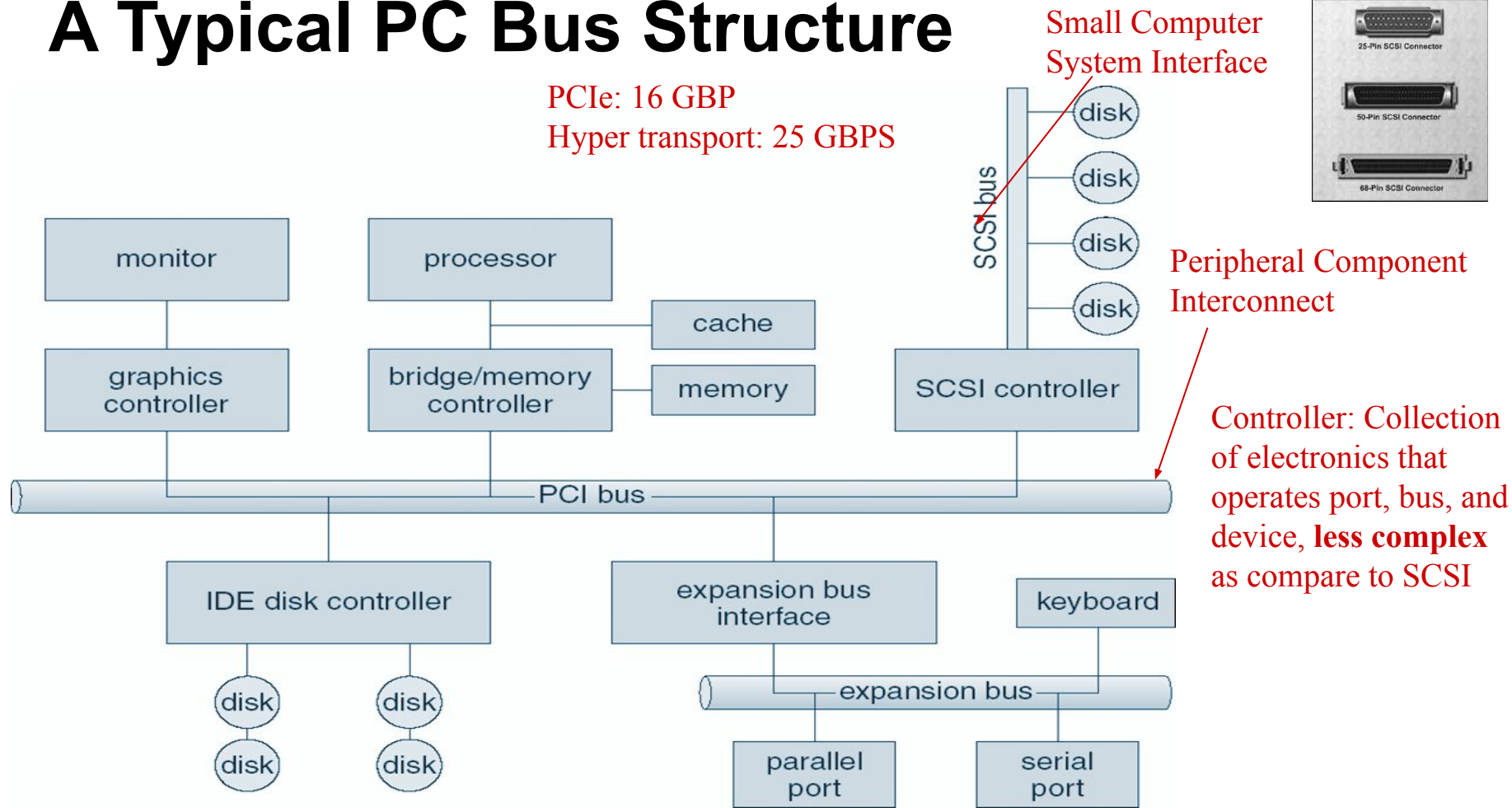
I/O Hardware

- Incredible variety of I/O devices
 - Storage
 - Transmission
 - Human-interface
 - 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 (**PCIe**)
 - **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
- Port: Communicate with a machine via a connection point
 - Bus: A set of wires
 - Daisy chain: A plug into B, B plug into C, C plug into a device
 - PCI: Peripheral Component Interconnect

A Typical PC Bus Structure

PCIe: 16 GBP

Hyper transport: 25 GBPS



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)

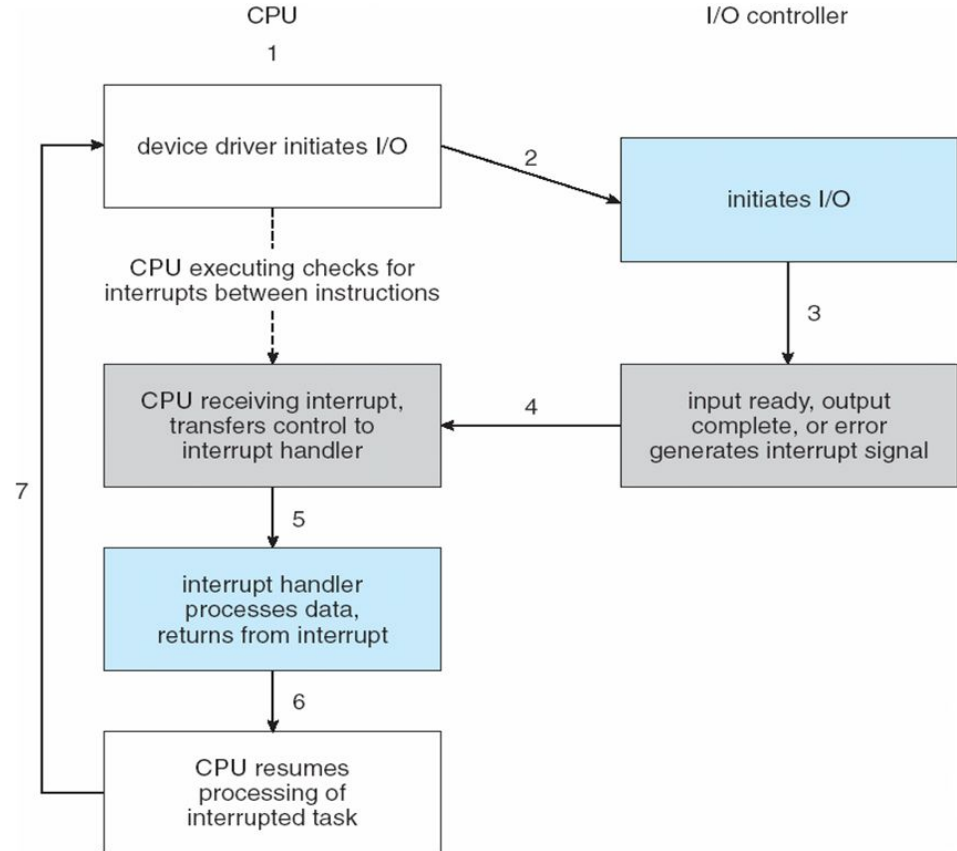
Polling

- For each byte of I/O
 1. Read busy bit from status register until 0
 2. Host sets write bit in the command register and write a byte 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 overwritten / lost
- To complete protocol for interaction between the host and a controller, a handshaking is required.
- Example: Producer-consumer relationship between controller and host
 - Controller: busy bit
 - Host: Command ready bit

Interrupts

- 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?
- 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
 - Based on priority
 - Some **nonmaskable**
 - Interrupt chaining if more than one device at same interrupt number
- Interrupt-request line is a **wire**, if it asserted a signal, CPU performs a state save and jump to the **interrupt handler routine**
- Typically, a CPU have **two** interrupt request lines
 - **Nonmaskable** - reserved for events such as unrecoverable memory errors
 - **Maskable** - Can be turned off by the CPU before executing uninterrupted instructions
- Interrupt vector holds address of specialized interrupt handlers

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

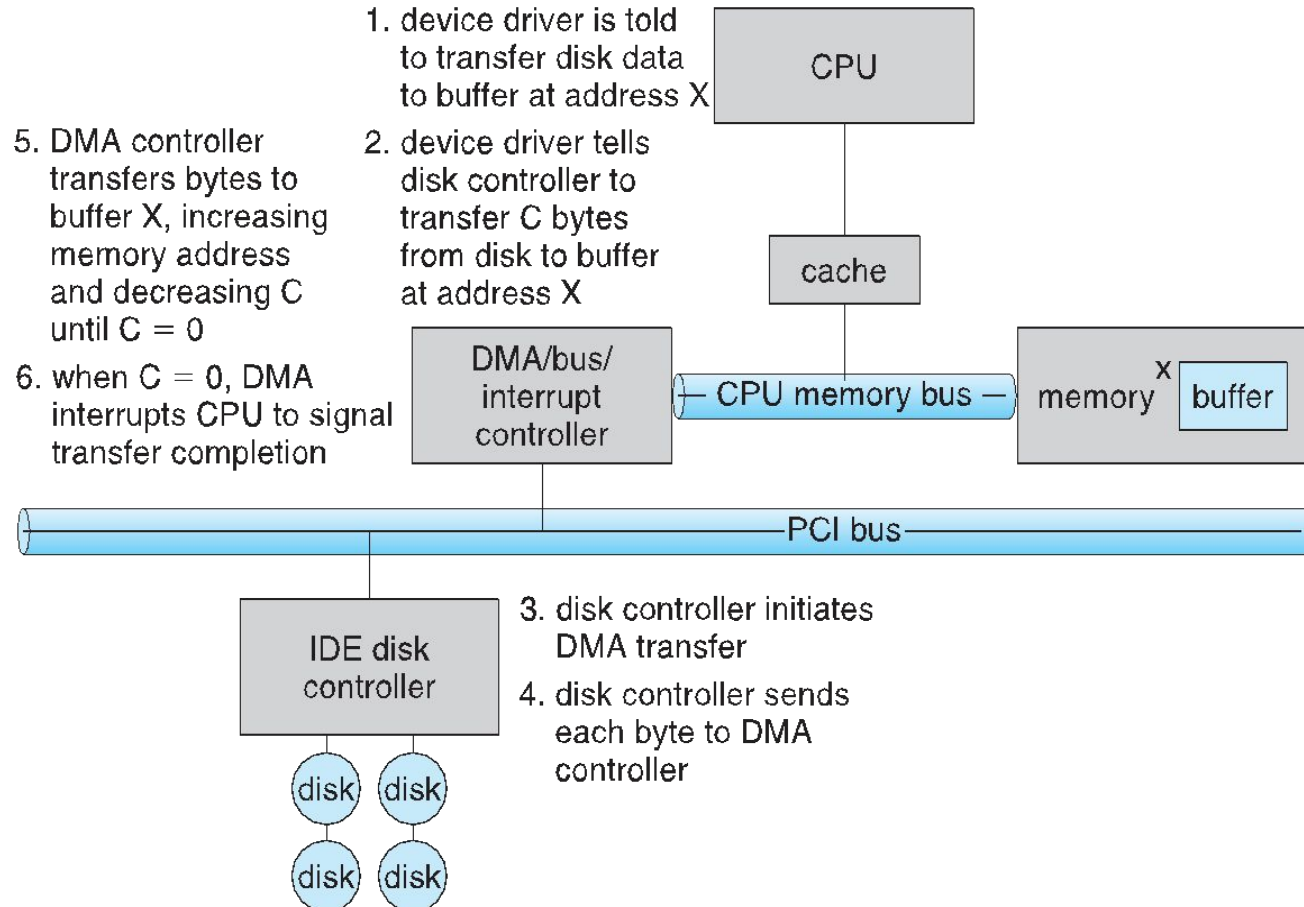
Interrupts (Cont.)

- 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
- Requires **DMA** controller
- Bypasses CPU to transfer data directly between I/O device and memory
- 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
- Version that is aware of virtual addresses can be even more efficient - **DVMA**

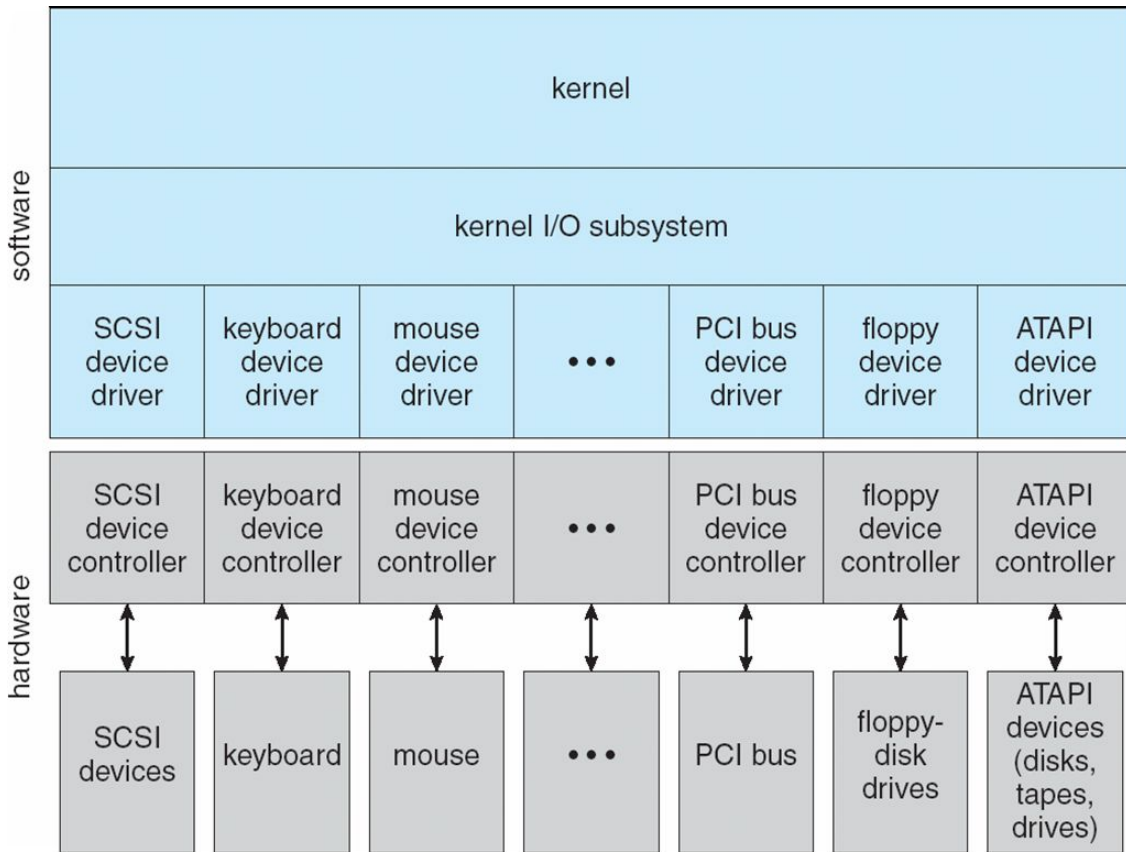
Six Step Process to Perform DMA Transfer



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)
 - **Sharable** or **dedicated**
 - **Speed of operation**
 - **read-write, read only, or write only**
- Abstraction, encapsulation, and software layering
- The detailed differences among the I/O devices are abstracted
- The differences are encapsulated in kernel module in the form of a device driver

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

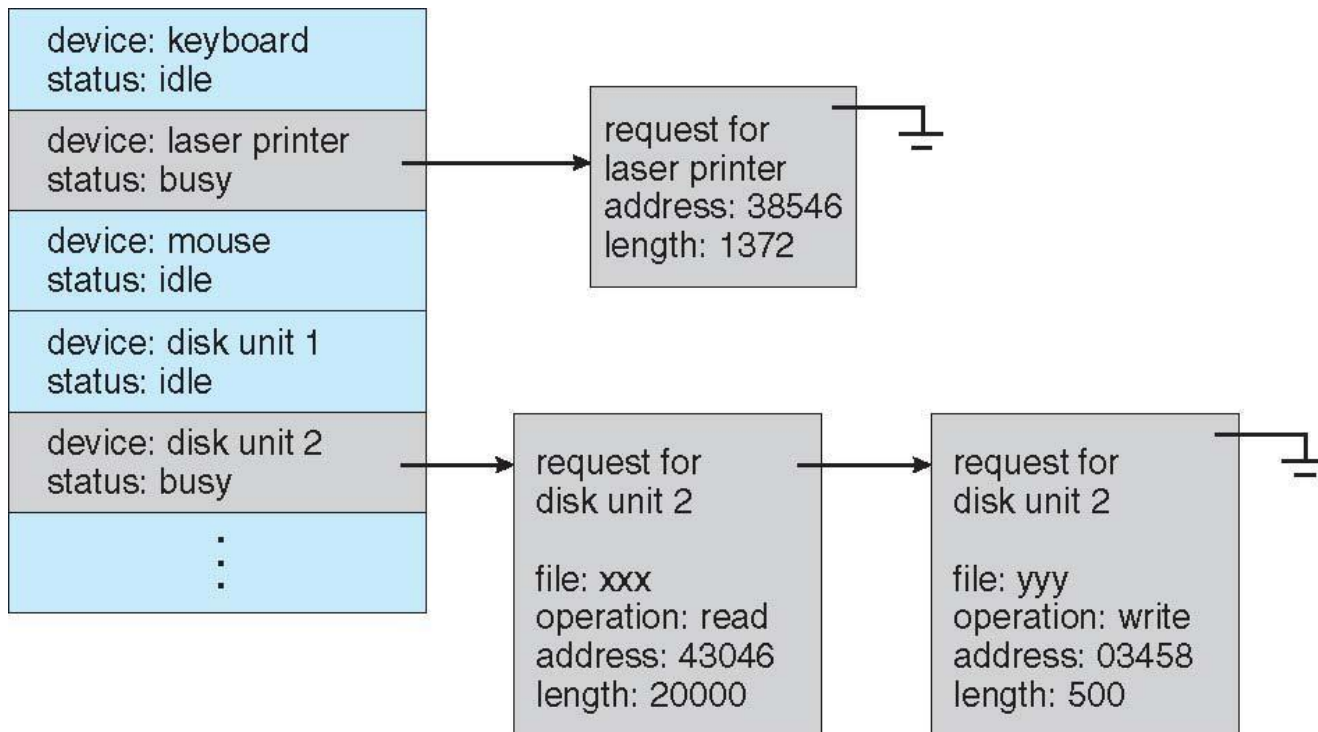
Transparently passes arbitrary commands from an application to a device driver



Kernel I/O Subsystem

- Scheduling
 - Some I/O request ordering via per-device queue
 - Some OSs try fairness
 - Some implement Quality Of Service (i.e. IPQOS)
- **Buffering** - store data in memory while transferring between devices/ or between a device and an application
 - 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



Kernel I/O Subsystem

- **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
 - i.e., Printing
- **Device reservation** - provides exclusive access to a device
 - System calls for allocation and de-allocation
 - Watch out for deadlock

Buffer may hold only existing copy of a data item, whereas a cache holds a copy on a faster storage of an item that resides elsewhere

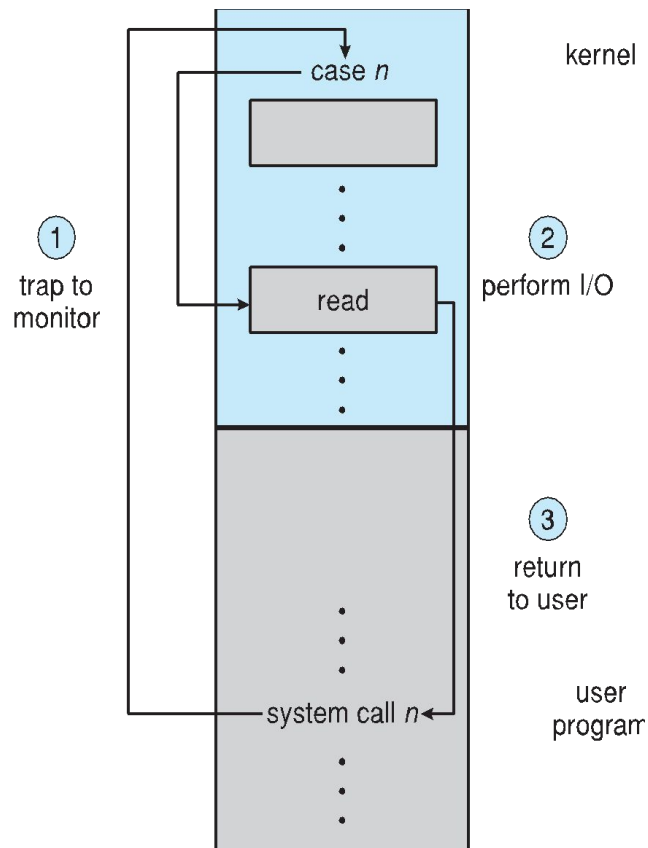
Error Handling

- 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
- Most return an error number or code when I/O request fails
- System error logs hold problem reports

I/O Protection

- 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

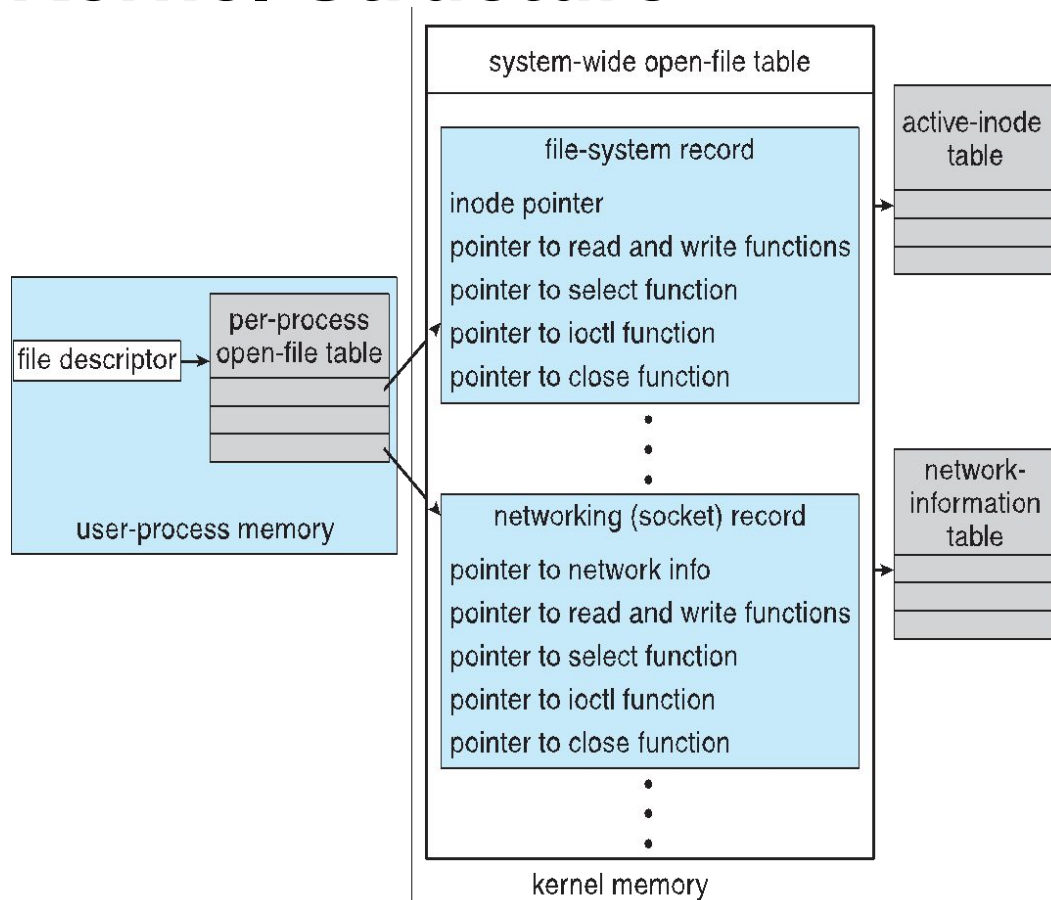
Use of a System Call to Perform I/O



Kernel Data Structures

- Kernel keeps state info for I/O components, including open file tables, network connections, character device state
- Many, many complex data structures to track buffers, memory allocation, “dirty” blocks
- 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?

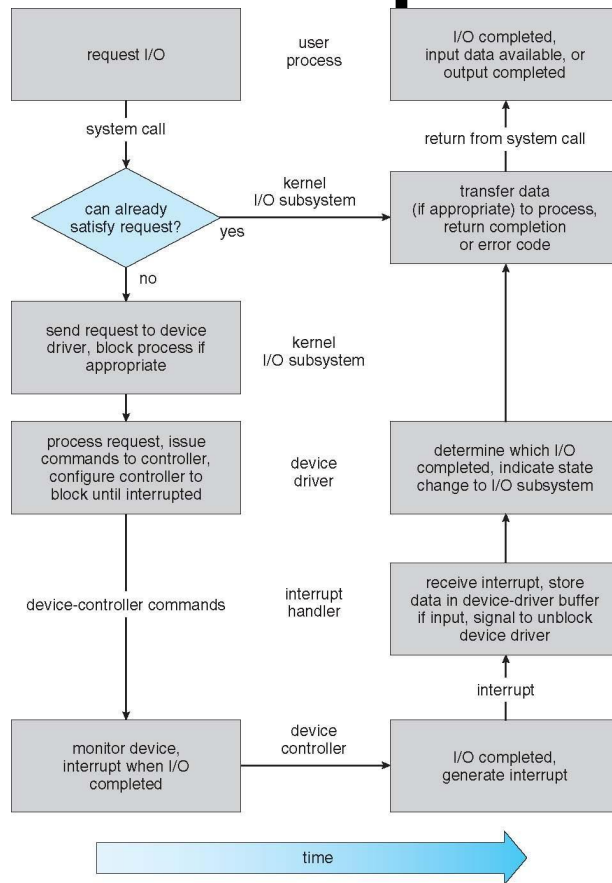
UNIX I/O Kernel Structure



I/O Requests to Hardware Operations

- 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



THANK YOU!

