I/O Systems

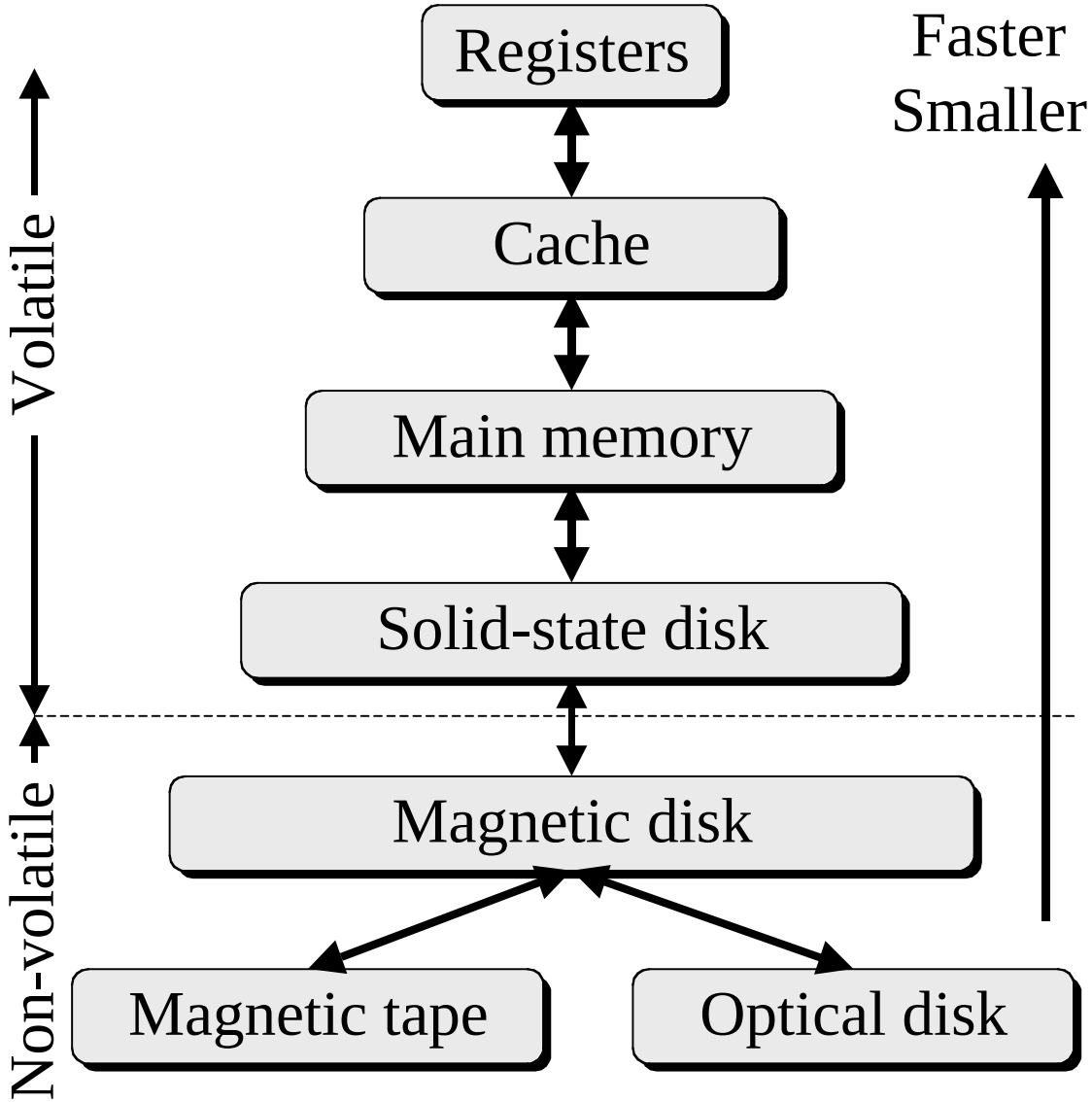
- What kinds of I/O devices are used by the operating system?
  - » How are they used to manipulate data?
  - » What kinds of operations can they do?
- How do applications make I/O requests?
- How does the kernel service I/O requests?
  - » How are user requests translated into OS requests?
  - » How does the kernel control I/O hardware?
- What are the performance issues?

Types of I/O Hardware

- Storage devices
  - » Hold data for short- or long-term
  - » Can be viewed as memory
  - » Examples
    - – Disk
    - – Tape
- Input devices
  - » Keyboard
  - » Mouse
- Output devices
  - » Video display
  - » Sound system
- Both input & output: network

Storage Device Hierarchy

- Main memory
  - » Low latency
  - » High bandwidth
  - » Volatile
- Magnetic disk
  - » Medium latency
  - » Medium bandwidth
  - » Non-volatile
- Magnetic tape
  - » Long latency
  - » Medium bandwidth
  - » Non-volatile
- CD-ROM

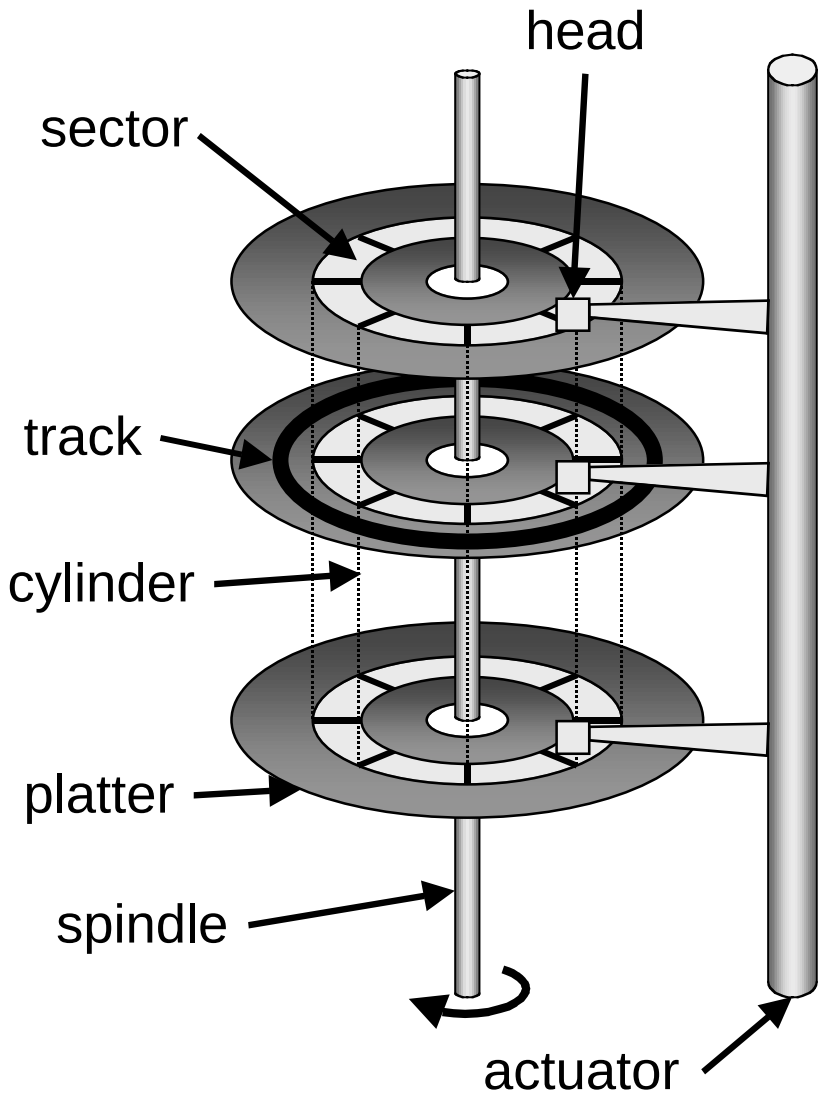


Data Movement in the Storage Hierarchy

- Data moves up the hierarchy
  - » Demand from a higher level
  - » Copy kept on both higher and lower levels (caching)
- Data moves down the hierarchy
  - » Lack of space in the higher level
  - » Volatility in the higher level
    - – Move the data for permanent storage
    - – Back up the data to guard against accidental loss
- Consistency & coherency
  - » Data on all levels should be the same
  - » Temporary differences may occur because of caching
  - » Differences must be kept tracked

Magnetic Disk (Hard Drive) Operation

- Data storage
  - Concentric circles (tracks)
  - Both sides of the platters
- Corresponding tracks on different surfaces make up a cylinder
- Minimum transfer unit is a sector
- All heads move (seek) together (fixed to actuator)
- To read or write data, disk
  - Seeks to correct track
  - Rotates to correct sector
  - Reads or write data as it passes under the appropriate head



Other Types of Disks

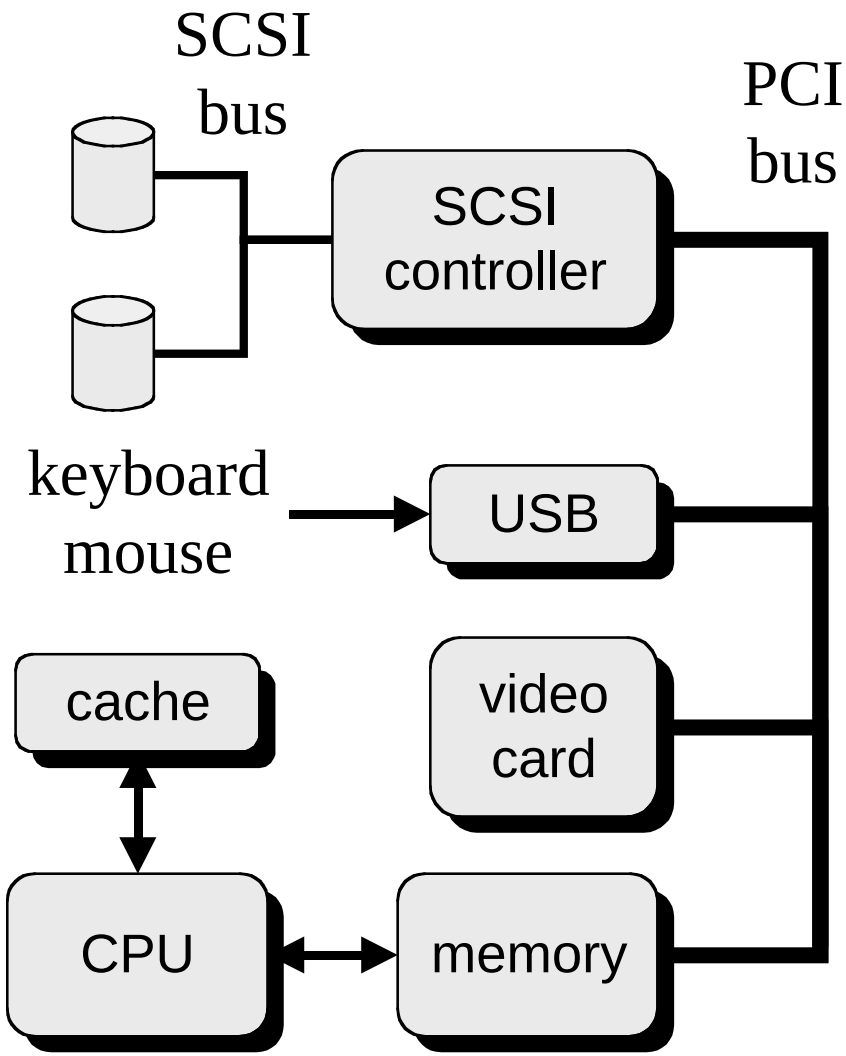
- Removable magnetic disk cartridges
  - » Regular 1.4 MB floppies & Zip disks
  - » Removable hard drives
  - » Range from slow to fast
  - » Very inexpensive
- CD-ROM / CD-R / DVD
  - » Write-once (or not at all)
  - » No risk of accidental erasure
  - » Read & written optically
- Magneto-optical
  - » Read & written magnetically with optical assistance
  - » Faster than optical (slightly)
  - » Less common today

Input & Output Devices

	Slow (low bandwidth)	Fast (high bandwidth)
Input	Keyboard Mouse Modem	Network Scanner
Output	Printer Modem Sound	Network Video

Connecting I/O Devices Together

- Many different I/O devices: use standard interface (bus)
  - » System bus (PCI)
  - » I/O bus (SCSI, IDE/ATA, USB, FireWire)
- Buses may be connected together by controller / adapter
  - » PCI card that speaks SCSI
- Bus has well-defined protocol controlled by
  - » Hardware (load / store or I/O commands)
  - » Software "commands"
- I/O port provides way for system to communicate with device



Controlling an I/O Device

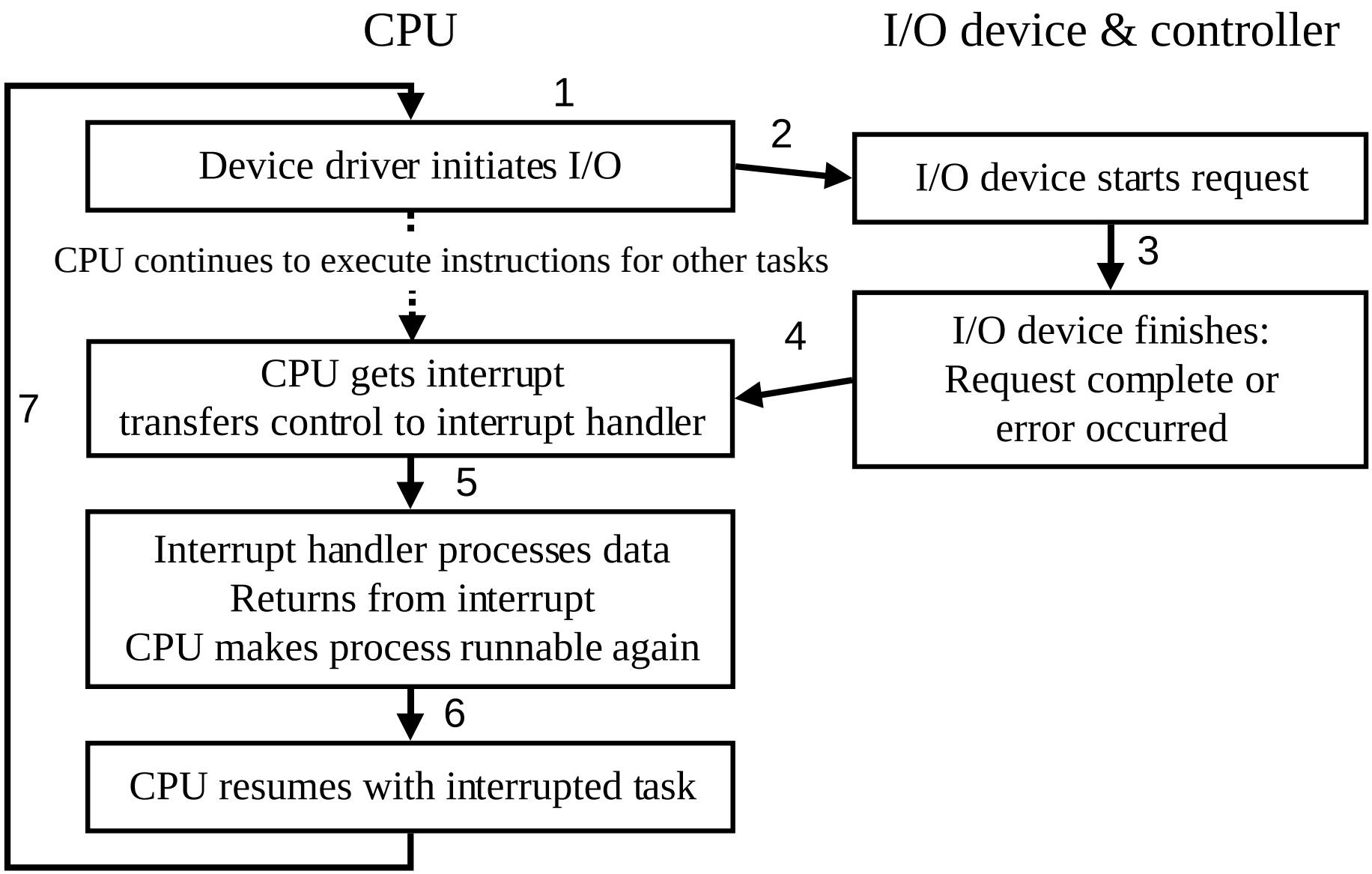
- Range of memory addresses assigned to each I/O device (memory mapping)
  - » Devices listen on bus for particular addresses
  - » Device control registers mapped to addresses
  - » Each device listens to a different range
- Simple register interface
  - » Status register (current condition of device)
  - » Control register (send instructions & control info to device)
  - » Data-in register (get data from the device)
  - » Data-out register (send data to the device)
- Host uses registers to communicate with the device
  - » Control the sending & receiving of data
  - » Discover device's status

Polling vs. Interrupts

- Polling repeatedly asks device for status
  - » Waits until device is ready
  - » Analogous to spin-lock
- Simple to implement
  - » Easy to program
  - » Cheap in hardware
- Inefficient
  - » Uses up CPU time
  - » Uses bus bandwidth

- Interrupt notifies CPU of any change in device status
  - » CPU can do other things in the meantime
  - » Analogous to sleep/wakeup
- Interrupt causes execution of "interrupt handler"
  - » Figures out which device caused interrupt
    - – Vector
    - – Polling
  - » Notifies waiting process
- Interrupt handler often handles exceptions as well

Interrupt-Driven I/O Cycle in OS



Direct Memory Access (DMA)

- Problem: many devices must transfer lots of data
  - » Polling (programmed I/O) sucks up CPU cycles
  - » Interrupts for each data word is worse!
- Solution: Direct Memory Access (DMA)
  - » CPU tells I/O device (controller) where to put data
  - » DMA Controller manages transfer, bypassing CPU
  - » Controller interrupts CPU when it's done
- DMA removes load from the CPU
  - » Steals memory cycles from the CPU: only one thing (CPU or DMA) can access memory in a given bus cycle
- DMA complicates OS programming
  - » Keep track of multiple outstanding requests
  - » Match completed requests with waiting processes

Steps in a DMA Request

- Application makes request to read *C* bytes from location *L* on disk to address *X* in memory
- Device driver tells disk controller to transfer *C* bytes from location *L* on disk to memory at address *X*
- Disk controller tells disk to seek to correct location
- Disk controller initiates DMA transfer and sends each word from disk to the DMA controller
- DMA controller transfers each word from to memory, keeping track of current address and count remaining
- When the count reaches zero, the DMA controller signals the CPU by causing an interrupt
- Device driver marks the request as complete and notifies application

I/O Interface for Applications

- Standard I/O system calls for common operations
  - » Read & write
  - » Open & close
- Device drivers hide differences among controllers and devices from the kernel
- Devices have many different characteristics
  - » Character vs. block-oriented
  - » Sequential vs. random-access
  - » Synchronous vs. asynchronous
  - » Sharable vs. dedicated (single-user)
  - » Slow vs. fast
  - » Read-only vs. write-only vs. read-write

Device Drivers

- Provide a standard interface to the kernel
  - » Read / write data to / from a buffer
  - » Provide ready / not-ready status
  - » Suspend process until I/O complete
- Manage the details of the particular device
  - » Keep device status
  - » Execute the specific instructions necessary to run the device
  - » Keep queues of buffers, requests, and processes
- Keep track of multiple devices of the same type
- Allow for easy addition of new devices
  - » Write new driver to standard interface
  - » Use devices interchangeably - interfaces are the same

Block and Character Devices

- Block devices
  - » Transfer data in big chunks
  - » Are typically memory-type devices
  - » Support read, write, and (usually) seek commands
  - » May support memory-mapped I/O: accesses look like memory reads & writes rather than like I/O
  - » Include disks, tapes, video displays
- Character devices
  - » Transfer data one (or a few) characters at a time
  - » Are usually relatively slow devices
  - » Support read (get) and write (put)
  - » Include keyboards, mice, serial ports, parallel ports

Networks: Neither Block nor Character

- Share some characteristics of both block and character devices
  - » Relatively high-speed
  - » Impossible to seek backwards (or anywhere else)
- Include a large amount of pre-processing
  - » Network protocol stack
  - » Separate processing for network I/O itself
- Not covered in CMSC 421 - take CMSC 481 instead...

I/O Devices Without Data?

- Timers
  - » Interrupt CPU after a particular interval has elapsed
  - » Keep track of total time since the timer was reset
- Clocks
  - » Provide real time and/or date
  - » Can be simulated with timer interrupts and software control
    - – Timer interrupts 10 times per second
    - – OS counts the number of timer interrupts
- Event counters
  - » Count CPU-specific events
  - » Useful for debugging & testing

Other I/O Operations

- Standard operations provided by OS
  - » Read & write
  - » Open & close
  - » Seek
- What about miscellaneous operations?
  - » All devices: status
  - » Tape: fast-forward & tape eject
  - » Printer: eject page
  - » Video display: set color tables, scan rate, etc.
- Solution: catch-all I/O command
  - » Specifics depend on the particular device & driver
  - » Allows the addition of new operations without changing the interface to the OS kernel

Error Handling for I/O Operations

- OS can usually recover from I/O errors
  - » Device error is simply reported to application
  - » Type of error is also available
    - – Transient error (out of paper, device busy, ...)
    - – Hard error (retrying won't help)
    - – Device unavailable or unknown
- Return error information to application
- Log the error in the system log for review by system administrator
  - » Identify failing devices
  - » Notify operator of needed changes (i.e., disk full)

I/O Calls and Process Waiting

- Question: does a process continue executing while waiting for an I/O request to complete?
- Blocking I/O: process waits until all data available
  - » Easy to implement and understand
  - » Less efficient: requires alternation of I/O and computation
- Non-blocking I/O: call returns as much data is available
  - » Returns quickly with count of data read
  - » Allows process to continue quickly with available data
- Asynchronous I/O: process runs while I/O proceeds
  - » Kernel signals process after the I/O is done
  - » Application can check I/O progress with system call
  - » Can be difficult to use
  - » May provide big gains in application speed (process chunk *n* while reading chunk *n*+1 and writing chunk *n*-1)

Generic I/O Modules in the Kernel

- Scheduling
  - » Provide queues and queue management for device drivers
  - » Allow drivers to suspend and wake up processes making I/O requests
- Buffering & caching
  - » Provide mechanisms to hold data in kernel memory
  - » Allow matching of device transfer rate to application rate
  - » Keep “copy semantics” of I/O - changes don’t affect previously written data
  - » Keep commonly used data in memory for fast access
- Device reservations
  - » Provide locking primitives
  - » Check for deadlocks (sometimes)

Kernel Data Structures

- Buffers & cache
  - » Description of information in the buffer
  - » Locking information for buffer
- Open file table
- Per-device information
  - » Status & other device-specific information
  - » Requests in queue or in progress
  - » Processes waiting for results
- Data structures can be complex and heavily interconnected
  - » Object-oriented methods can simplify matters a bit
  - » Message passing between components can also help

Sending an I/O Request to Hardware

- Determine which device is desired
- Send the request to the appropriate device driver
- Put the request on a queue maintained by the driver
- Allocate kernel buffers necessary to do the transfer, and mark them as busy
- Perform the I/O operation, copying data to or from the buffers
- Notify the device driver that the request is complete
- Copy the data from kernel buffers to user data space
- Remove the request from the device queue
- Wake up the waiting process and allow it to continue

UNIX Request: Read a Disk Sector

- Determine which disk & sector to read
  - » File system determines this information
  - » Use device number to identify a device driver
- Allocate buffer & put request in device queue
  - » Suspend process until request is complete
  - » Lock buffer so no other process will use it
- Send the request to the disk
  - » Device driver knows how to format the request properly
- Mark request complete upon receiving interrupt
- Copy the data from buffer to user process
  - » Keep a copy in kernel buffer (cache) for possible reuse
- Resume the user process, which can then use the newly read data

How a System Call Works

- Problem: user process wants to do an I/O request
  - » Operating system has to do the request
  - » Needs to be a controlled way of getting into the kernel
- Solution: use a “trap” instruction
  - » Pass parameters in registers
  - » Pass parameters on stack
    - – Must translate between user and system addresses
    - – Need to copy string-type data between user and kernel addresses
  - » Return value goes in a register
- Trap handled as an exception
  - » Interrupts turned off unless kernel explicitly reactivates them
  - » Turn on interrupts if call will take a long time

Performance Issues

- I/O is an important part of computer system performance
  - » Slow I/O means data moves in and out of computer slowly
  - » Faster processor doesn't necessarily help slow I/O
- I/O affects performance by:
  - » Unnecessary memory copies
  - » Bloated, inefficient driver code and other kernel code
  - » Overhead from interrupts and context switches
  - » Programmed I/O requiring CPU to do all the work
- File systems and networks are particularly important
  - » High-bandwidth I/O devices
  - » Frequently-used parts of the I/O system
  - » Potentially lots of copying because of layers of abstraction in the OS

Improving I/O Performance in the OS

- Reduce the amount of copying
  - » Do I/O directly to user buffers?
  - » Try to do data interpretation in-place (networks, file systems)
- Reduce the number of context switches
  - » Allow processes to wait for multiple I/O requests
  - » Handle I/O entirely in the kernel without assistance from user processes
- Cut down on interrupts
  - » Use devices that do more without CPU direction
  - » Use polling where appropriate
- Use DMA to shift load from the CPU to the DMA controller
- Balance CPU speed, memory speed, bus speed, and I/O device speed for highest throughput