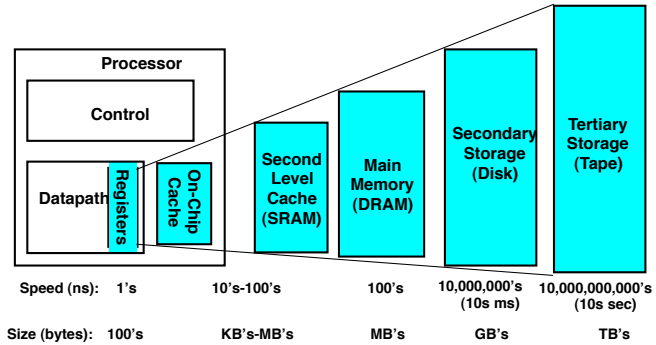
CS162
Operating Systems and Systems Programming
Lecture 15

Kernel/User, I/O, Disks

October 19, 2011
Anthony D. Joseph and Ion Stoica
http://inst.eecs.berkeley.edu/~cs162

Review: Memory Hierarchy of a Modern Computer System

- Take advantage of the principle of locality to:
  - Present as much memory as in the cheapest technology
  - Provide access at speed offered by the fastest technology



Goals for Today

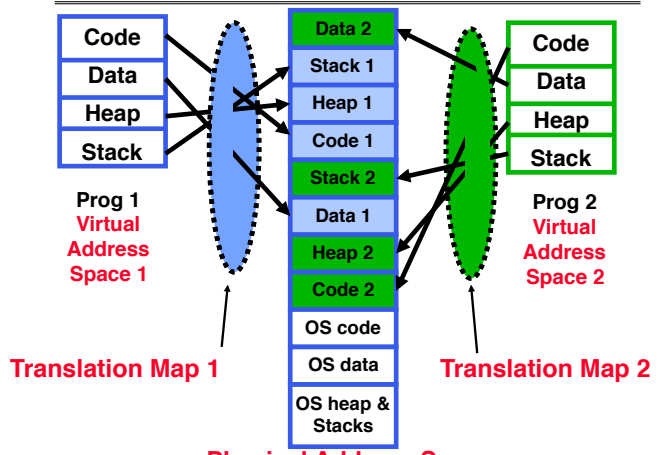
- Important System Properties
- Dual Mode Operation: Kernel versus User Mode
- I/O Systems
  - Hardware Access
  - Device Drivers
- Disk Performance
  - Hardware performance parameters

Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne. Many slides generated from my lecture notes by Kubiatowicz.

Important "ilities"

- Availability: the probability that the system can accept and process requests
  - Often measured in "nines" of probability. So, a 99.9% probability is considered "3-nines of availability"
  - Key idea here is independence of failures
- Durability: the ability of a system to recover data despite faults
  - This idea is fault tolerance applied to data
- *Durability doesn't imply Availability*
  - Information on pyramids was very durable, but could not be accessed until discovery of Rosetta Stone
- Reliability: the ability of a system or component to perform its required functions under stated conditions for a specified period of time (IEEE definition)
  - Usually stronger than simply availability: means that the system is not only "up", but also working correctly
  - Includes availability, security, fault tolerance/durability
  - Must make sure data survives system crashes, disk crashes, other problems

Review: Example of General Address Translation



Dual-Mode Operation

- Can an application modify its own translation maps?
  - If it could, could get access to all of physical memory
  - Has to be restricted somehow
- To assist with protection, hardware provides at least two modes (Dual-Mode Operation):
  - "Kernel" mode (or "supervisor" or "protected")
  - "User" mode (Normal program mode)
  - Mode set with bits in special control register only accessible in kernel-mode
- Intel processors actually have four "rings" of protection:
  - PL (Privilege Level) from 0 – 3
    - » PL0 has full access, PL3 has least
  - Typical OS kernels on Intel processors only use PL0 ("kernel") and PL3 ("user")

For Protection, Lock User-Programs in Asylum

- Idea: Lock user programs in padded cell with no exit or sharp objects
  - Cannot change mode to kernel mode
  - User cannot modify translation maps
  - Limited access to memory: cannot adversely effect other processes
    - » Side-effect: Limited access to memory-mapped I/O operations
  - What else needs to be protected?

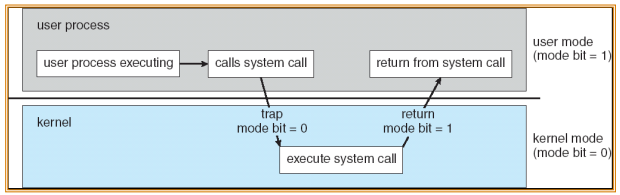
- A couple of issues
  - How to share CPU between kernel and user programs?
    - » Kinda like both the inmates and the warden in asylum are the same person. How do you manage this???
  - How does one switch between kernel and user modes?
    - » OS → user (kernel → user mode): getting into cell
    - » User→ OS (user → kernel mode): getting out of cell

How to get from Kernel→User

- What does the kernel do to create a new user process?
  - Allocate and initialize process control block
  - Read program off disk and store in memory
  - Allocate and initialize translation map
    - » Point at code in memory so program can execute
    - » Possibly point at statically initialized data
  - Run Program:
    - » Set machine registers
    - » Set hardware pointer to translation table
    - » Set processor status word for user mode
    - » Jump to start of program
- How does kernel switch between processes?
  - Same saving/restoring of registers as before
  - Save/restore hardware pointer to translation map

User→Kernel (System Call)

- Can't let inmate (user) get out of padded cell on own
  - Would defeat purpose of protection!
  - So, how does the user program get back into kernel?



- System call: Voluntary procedure call into kernel
  - Hardware for controlled User→Kernel transition
  - Can any kernel routine be called?
    - » No! Only specific ones.
  - System call ID encoded into system call instruction
    - » Index forces well-defined interface with kernel

System Call Continued

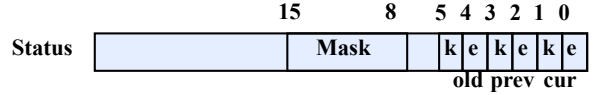
- What are some system calls?
  - I/O: open, close, read, write, lseek
  - Files: delete, mkdir, rmdir, truncate, chown, chgrp, ..
  - Process: fork, exit, wait (like join)
  - Network: socket create, set options
- Are system calls constant across operating systems?
  - Not entirely, but there are lots of commonalities
  - Also some standardization attempts (POSIX)
- What happens at beginning of system call?
  - On entry to kernel, sets system to kernel mode
  - Handler address fetched from table, and Handler started
- System Call argument passing:
  - In registers (not very much can be passed)
  - Write into user memory, kernel copies into kernel memory
    - » User addresses must be translated!
    - » Kernel has different view of memory than user
  - *Every argument must be explicitly checked!*

User→Kernel (Exceptions: Traps and Interrupts)

- A system call instruction causes a synchronous exception (or "trap")
  - In fact, often called a software "trap" instruction
- Other sources of *Synchronous Exceptions:*
  - Divide by zero, Illegal instruction, Bus error (bad address, e.g. unaligned access)
  - Segmentation Fault (address out of range)
  - Page Fault (for illusion of infinite-sized memory)
- Interrupts are *Asynchronous Exceptions*
  - Examples: timer, disk ready, network, etc….
  - Interrupts can be disabled, traps cannot!
- On system call, exception, or interrupt:
  - Hardware enters kernel mode with interrupts disabled
  - Saves PC, then jumps to appropriate handler in kernel
  - For some processors (x86), processor also saves registers, changes stack, etc.

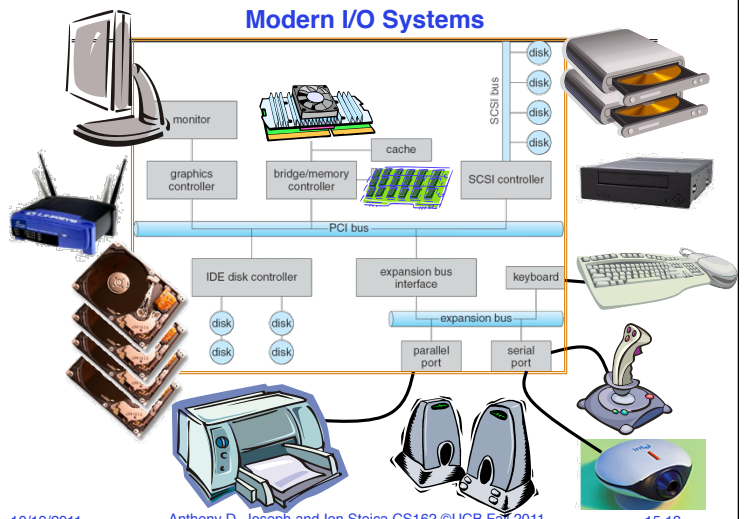
Additions to MIPS ISA to support Exceptions?

- Exception state is kept in "Coprocessor 0"
  - Use mfc0 to read contents of these registers:
    - » BadVAddr (register 8): contains memory address at which memory reference error occurred
    - » Status (register 12): interrupt mask and enable bits
    - » Cause (register 13): the cause of the exception
    - » EPC (register 14): address of the affected instruction



- Status Register fields:
  - Mask: Interrupt enable
    - » 1 bit for each of 5 hardware and 3 software interrupts
  - k = kernel/user: 0⇒kernel mode
  - e = interrupt enable: 0⇒interrupts disabled
  - Exception⇒6 LSB shifted left 2 bits, setting 2 LSB to 0:
    - » run in kernel mode with interrupts disabled

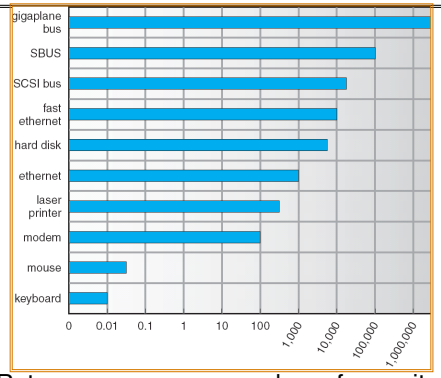
Modern I/O Systems



The Requirements of I/O

- What is the role of I/O?
  - Without I/O, computers are useless (disembodied brains?)
  - But… thousands of devices, each slightly different
    - » How can we standardize the interfaces to these devices?
  - Devices unreliable: media failures and transmission errors
    - » How can we make them reliable???
  - Devices unpredictable and/or slow
    - » How can we manage them if we don't know what they will do or how they will perform?
- Some operational parameters:
  - Byte/Block
    - » Some devices provide single byte at a time (*e.g.,* keyboard)
    - » Others provide whole blocks (*e.g.,* disks, networks, etc.)
  - Sequential/Random
    - » Some devices must be accessed sequentially (*e.g.,* tape)
    - » Others can be accessed randomly (*e.g.,* disk, cd, etc.)
  - Polling/Interrupts
    - » Some devices require continual monitoring
    - » Others generate interrupts when they need service

Example Device-Transfer Rates (Sun Enterprise 6000)



- Device Rates vary over many orders of magnitude
  - System better be able to handle this wide range
  - Better not have high overhead/byte for fast devices!
  - Better not waste time waiting for slow devices

The Goal of the I/O Subsystem

- Provide uniform interfaces, despite wide range of different devices
  - This code works on many different devices:

```
FILE fd = fopen("/dev/something","rw");
for (int i = 0; i < 10; i++) {
  fprintf(fd, "Count %d\n",i);
}
close(fd);
```

  - Why? Because code that controls devices ("device driver") implements standard interface.
- We will try to get a flavor for what is involved in actually controlling devices in rest of lecture
  - Can only scratch surface!

Want Standard Interfaces to Devices

- Block Devices: *e.g.,* disk drives, tape drives, DVD-ROM
  - Access blocks of data
  - Commands include `open()`, `read()`, `write()`, `seek()`
  - Raw I/O or file-system access
  - Memory-mapped file access possible
- Character Devices: *e.g.,* keyboards, mice, serial ports, some USB devices
  - Single characters at a time
  - Commands include `get()`, `put()`
  - Libraries layered on top allow line editing
- Network Devices: *e.g.,* Ethernet, Wireless, Bluetooth
  - Different enough from block/character to have own interface
  - Unix and Windows include socket interface
    - » Separates network protocol from network operation
    - » Includes `select()` functionality
  - Usage: pipes, FIFOs, streams, queues, mailboxes

How Does User Deal with Timing?

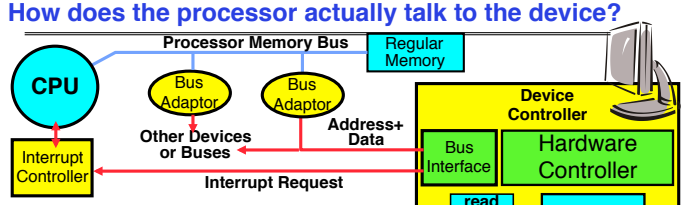
- Blocking Interface: "Wait"
  - When request data (*e.g.,* `read()` system call), put process to sleep until data is ready
  - When write data (*e.g.,* `write()` system call), put process to sleep until device is ready for data
- Non-blocking Interface: "Don't Wait"
  - Returns quickly from read or write request with count of bytes successfully transferred to kernel
  - Read may return nothing, write may write nothing
- Asynchronous Interface: "Tell Me Later"
  - When requesting data, take pointer to user's buffer, return immediately; later kernel fills buffer and notifies user
  - When sending data, take pointer to user's buffer, return immediately; later kernel takes data and notifies user

Administrivia

- Check our addition on your midterm exam!
- New lecture schedule posted: new order for topics
  - I/O, File Systems, Security, Address Translation, Capstones
- New project deadlines posted
  - Project 2 code/final design deadline moved later in term
  - Project 3 being merged with project 4
    - » Working to improve quality/experience
    - » Will focus on moving client and server to Amazon EC2 and adding a database for storing game history

5min Break

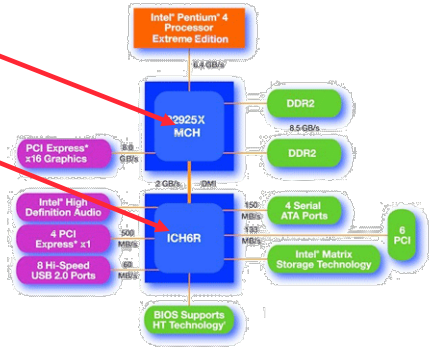
How does the processor actually talk to the device?



- CPU interacts with a *Controller*
  - Contains a set of *registers* that can be read and written
  - May contain memory for request queues or bit-mapped images
- Regardless of the complexity of the connections and buses, processor accesses registers in two ways:
  - I/O instructions: in/out instructions
    - » Example from the Intel architecture: out 0x21,AL
  - Memory mapped I/O: load/store instructions
    - » Registers/memory appear in physical address space
    - » I/O accomplished with load and store instructions

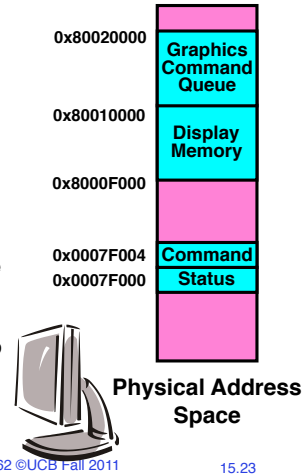
Main components of Intel Chipset: Pentium 4

- Northbridge:
  - Handles memory
  - Graphics
- Southbridge: I/O
  - PCI bus
  - Disk controllers
  - USB controllers
  - Audio
  - Serial I/O
  - Interrupt controller
  - Timers



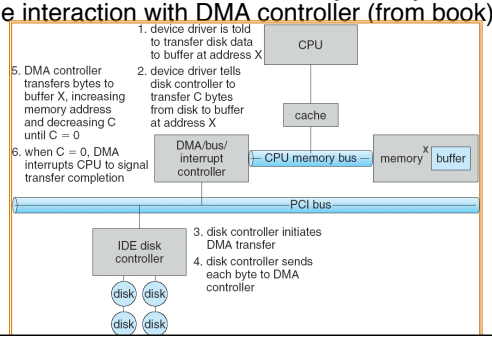
Example: Memory-Mapped Display Controller

- Memory-Mapped:
  - Hardware maps control registers and display memory into physical address space
    - » Addresses set by hardware jumpers or programming at boot time
  - Simply writing to display memory (also called the "frame buffer") changes image on screen
    - » Addr: 0x8000F000—0x8000FFFF
  - Writing graphics description to command-queue area
    - » Say enter a set of triangles that describe some scene
    - » Addr: 0x80010000—0x8001FFFF
  - Writing to the command register may cause on-board graphics hardware to do something
    - » Say render the above scene
    - » Addr: 0x0007F004
- Can protect with address translation

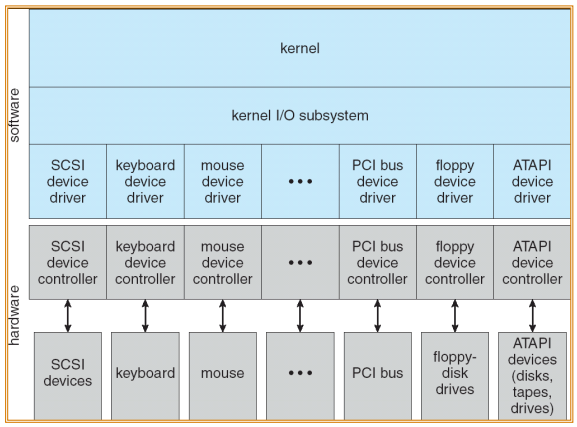


Transferring Data To/From Controller

- Programmed I/O:
  - Each byte transferred via processor in/out or load/store
  - Pro: Simple hardware, easy to program
  - Con: Consumes processor cycles proportional to data size
- Direct Memory Access:
  - Give controller access to memory bus
  - Ask it to transfer data to/from memory directly
- Sample interaction with DMA controller (from book):



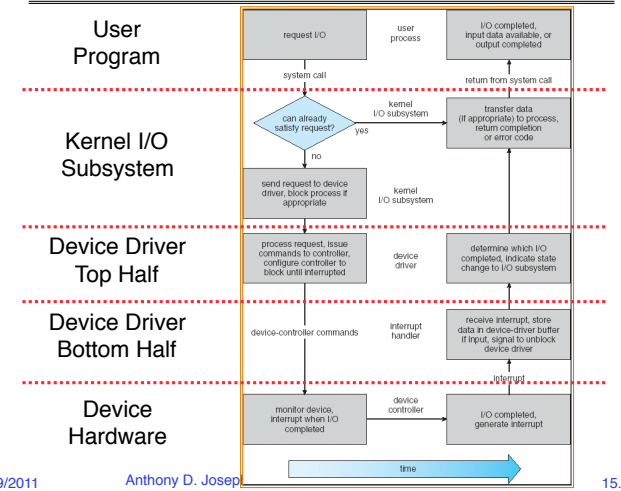
A Kernel I/O Structure



Device Drivers

- Device Driver: Device-specific code in the kernel that interacts directly with the device hardware
  - Supports a standard, internal interface
  - Same kernel I/O system can interact easily with different device drivers
  - Special device-specific configuration supported with the `ioctl()` system call
- Device Drivers typically divided into two pieces:
  - Top half: accessed in call path from system calls
    - » Implements a set of standard, cross-device calls like `open()`, `close()`, `read()`, `write()`, `ioctl()`, `strategy()`
    - » This is the kernel's interface to the device driver
    - » Top half will *start* I/O to device, may put thread to sleep until finished
  - Bottom half: run as interrupt routine
    - » Gets input or transfers next block of output
    - » May wake sleeping threads if I/O now complete

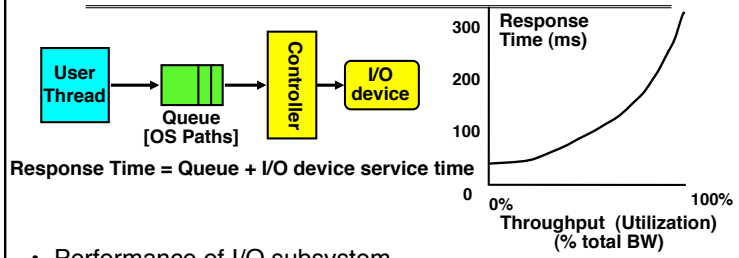
Life Cycle of An I/O Request



I/O Device Notifying the OS

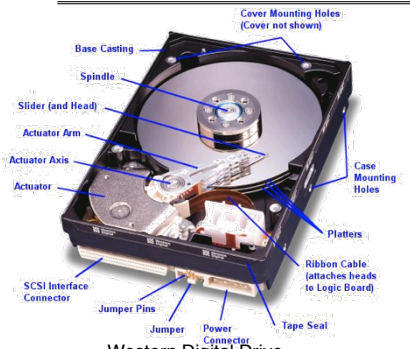
- The OS needs to know when:
  - The I/O device has completed an operation
  - The I/O operation has encountered an error
- I/O Interrupt:
  - Device generates an interrupt whenever it needs service
  - Handled in bottom half of device driver
    - » Often run on special kernel-level stack
  - Pro: handles unpredictable events well
  - Con: interrupts relatively high overhead
- Polling:
  - OS periodically checks a device-specific status register
    - » I/O device puts completion information in status register
    - » Could use timer to invoke lower half of drivers occasionally
  - Pro: low overhead
  - Con: may waste many cycles on polling if infrequent or unpredictable I/O operations
- Actual devices combine both polling and interrupts
  - For instance – High-bandwidth network adapter:
    - » Interrupt for first incoming packet
    - » Poll for following packets until hardware queues are empty

I/O Performance



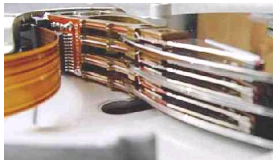
- Performance of I/O subsystem
  - Metrics: Response Time, Throughput
  - Contributing factors to latency:
    - » Software paths (can be loosely modeled by a queue)
    - » Hardware controller
    - » I/O device service time
- Queuing behavior:
  - Can lead to big increases of latency as utilization approaches 100%

Hard Disk Drives

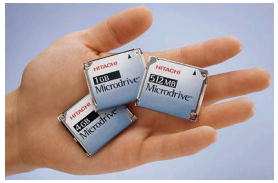


Western Digital Drive
http://www.storagereview.com/guide/

IBM Personal Computer/AT (1986)
30 MB hard disk - $500
30-40ms seek time
0.7-1 MB/s (est.)

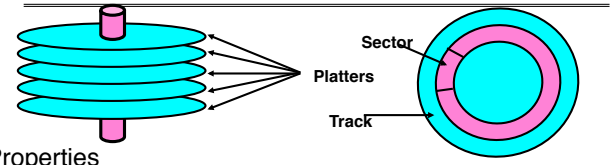


Read/Write Head Side View



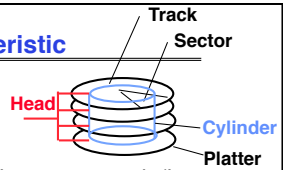
IBM/Hitachi Microdrive

Properties of a Magnetic Hard Disk

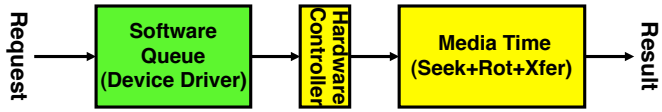


- Properties
  - Independently addressable element: sector
    - » OS always transfers groups of sectors together—"blocks"
  - A disk can access directly any given block of information it contains (random access). Can access any file either sequentially or randomly.
  - A disk can be rewritten in place: it is possible to read/modify/write a block from the disk
- Typical numbers (depending on the disk size):
  - 500 to more than 20,000 tracks per surface
  - 32 to 800 sectors per track
    - » A sector is the smallest unit that can be read or written
- Zoned bit recording
  - Constant bit density: more sectors on outer tracks
  - Speed varies with track location

Magnetic Disk Characteristic



- Cylinder: all the tracks under the head at a given point on all surfaces
- Read/write data is a three-stage process:
  - Seek time: position the head/arm over the proper track (into proper cylinder)
  - Rotational latency: wait for the desired sector to rotate under the read/write head
  - Transfer time: transfer a block of bits (sector) under the read-write head
- Disk Latency = Queuing Time + Controller time + Seek Time + Rotation Time + Xfer Time



- Highest Bandwidth:
  - Transfer large group of blocks sequentially from one track

Typical Numbers of a Magnetic Disk

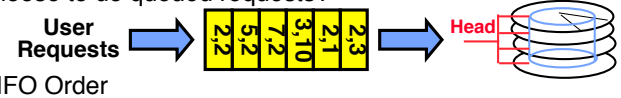
- Average seek time as reported by the industry:
  - Typically in the range of 8 ms to 12 ms
  - Due to locality of disk reference may only be 25% to 33% of the advertised number
- Rotational Latency:
  - *Most* disks rotate at 3,600 to 7200 RPM (Up to 15,000RPM or more)
  - Approximately 16 ms to 8 ms per revolution, respectively
  - An average latency to the desired information is halfway around the disk: 8 ms at 3600 RPM, 4 ms at 7200 RPM
- Transfer Time is a function of:
  - Transfer size (usually a sector): 512B – 1KB per sector
  - Rotation speed: 3600 RPM to 15000 RPM
  - Recording density: bits per inch on a track
  - Diameter: ranges from 1 in to 5.25 in
  - Typical values: 2 to 50 MB per second
- Controller time depends on controller hardware
- Cost drops by factor of two per year (since 1991)

Disk Performance Examples

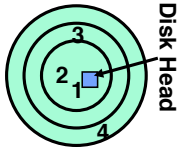
- Assumptions:
  - Ignoring queuing and controller times for now
  - Avg seek time of 5ms,
  - 7200RPM ⇒ Time for one rotation: 8ms
  - Transfer rate of 4MByte/s, sector size of 1 KByte
- Read sector from random place on disk:
  - Seek (5ms) + Rot. Delay (4ms) + Transfer (0.25ms)
  - Approx 10ms to fetch/put data: 100 KByte/sec
- Read sector from random place in same cylinder:
  - Rot. Delay (4ms) + Transfer (0.25ms)
  - Approx 5ms to fetch/put data: 200 KByte/sec
- Read next sector on same track:
  - Transfer (0.25ms): 4 MByte/sec
- Key to using disk effectively (esp. for filesystems) is to minimize seek and rotational delays

Disk Scheduling

- Disk can do only one request at a time; What order do you choose to do queued requests?

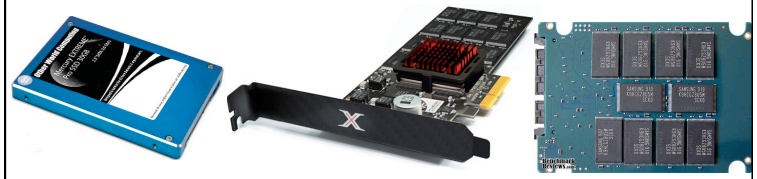


- FIFO Order
  - Fair among requesters, but order of arrival may be to random spots on the disk ⇒ Very long seeks
- SSTF: Shortest seek time first
  - Pick the request that's closest on the disk
  - Although called SSTF, today must include rotational delay in calculation, since rotation can be as long as seek
  - Con: SSTF good at reducing seeks, but may lead to starvation



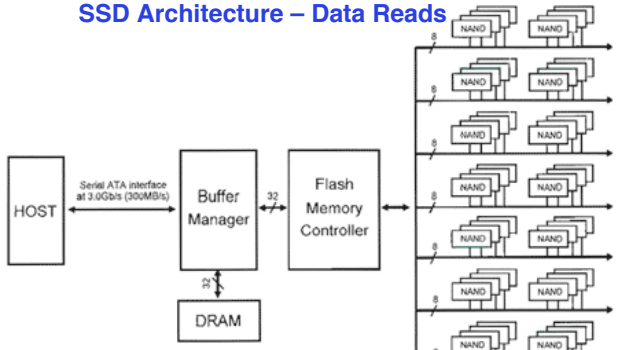
- SCAN: Implements an Elevator Algorithm: take the closest request in the direction of travel
  - No starvation, but retains flavor of SSTF
- C-SCAN: Circular-Scan: only goes in one direction
  - Skips any requests on the way back
  - Fairer than SCAN, not biased towards pages in middle

Solid State Disks (SSDs)



- 1995 – Replace rotating magnetic media with non-volatile memory (battery backed DRAM, since 2009 NAND Flash)
  - Single Level Cell (1-bit/cell), Multi-Level Cell (2-bit/cell)
- Sector addressable
  - Stores 4-64 "sectors" per memory page
- No moving parts (no rotate/seek motors)
  - Eliminates seek and rotational delay (0.1-0.2ms access time)
  - Very low power and lightweight

SSD Architecture – Data Reads



- Read data is similar to memory read (25µs)
  - No seek or rotational latency
  - Transfer time: transfer a block of bits (sector)
    - » Limited by controller and disk interface (SATA: 300-600MB/s)
  - Disk Latency = Queuing Time + Controller time + Xfer Time
  - Highest Bandwidth: Sequential OR Random reads

SSD Architecture – Data Writes

- Write data is complex! (~200µs – 1.7ms)
  - No seek or rotational latency
  - Transfer time: transfer a block of bits (sector)
- But, can only write empty pages (erase takes ~1.5ms!)
  - Controller maintains pool of empty pages by coalescing used sectors (read, erase, write), also reserve some % of capacity
- Steady state, when SSD full
  - One erase every 64 or 128 writes (depending on page size)
- Write and erase cycles require "high" voltage
  - Damages memory cells, limits SSD lifespan
  - Controller uses ECC, performs wear leveling
  - OS may provide TRIM information about "deleted" sector
- Result is very workload dependent performance
  - Disk Latency = Queuing Time + Controller time (Find Free Block) + Xfer Time

Rule of thumb: writes 10x more expensive than reads, and erases 10x more expensive than writes

Storage Performance & Price

	Bandwidth (sequential R/W)	Cost/GB	Size
HHD	50-100 MB/s	$0.05-0.1/GB	2-4 TB
SSD[1]	200-500 MB/s (SATA) 6 GB/s (PCI)	$2-5/GB	200GB-1TB
DRAM	10-16 GB/s	$12-13/GB	64GB-256GB

[1]http://www.fastestssd.com/featured/ssd-rankings-the-fastest-solid-state-drives/

BW: SSD up to x10 than HDD, DRAM > x10 than SSD
Price: HDD x20 less than SSD, SSD x5 less than DRAM

SSD Summary

- Pros (vs. magnetic disk drives):
  - Low latency, high throughput (eliminate seek/rotational delay)
  - No moving parts:
    - » Very light weight, low power, silent, very shock insensitive
  - Read at memory speeds (limited by controller and I/O bus)
- Cons
  - Small storage (0.1-0.5x disk), very expensive (20x disk)
    - » Hybrid alternative: combine small SSD with large HDD
  - Asymmetric block write performance: read pg/erase/write pg
    - » Controller GC algorithms have major effect on performance
    - » Sequential write performance may be worse than HDD
  - Limited drive lifetime (NOR is higher, more expensive)
    - » 50-100K writes/page for SLC, 1-10K writes/page for MLC

Summary

- Important system properties
  - Availability: how often is the resource available?
  - Durability: how well is data preserved against faults?
  - Reliability: how often is resource performing correctly?
- Dual-Mode
  - Kernel/User distinction: User restricted
  - User→Kernel: System calls, Traps, or Interrupts
  - Inter-process communication: shared memory, or through kernel (system calls)
- I/O Devices Types:
  - Many different speeds (0.1 bytes/sec to GBytes/sec)
  - Different Access Patterns: block, char, net devices
  - Different Access Timing: Non-/Blocking, Asynchronous

Summary

- I/O Controllers: Hardware that controls actual device
  - CPU accesses thru I/O insts, ld/st to special phy memory
  - Report results thru interrupts or a status register polling
- Device Driver: Device-specific code in kernel
- Magnetic Disk Performance:
  - Queuing time + Controller + Seek + Rotational + Transfer
  - Rotational latency: on average ½ rotation
  - Transfer time: depends on rotation speed and bit density
- SSD Performance:
  - Read: Queuing time + Controller + Transfer
  - Write: Queuing time + Controller (Find Free Block) + Transfer
  - Find Free Block time: depends on how full SSD is (available empty pages), write burst duration, …
  - Limited drive lifespan