Recall: How does the Processor 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, etc.

Processor accesses registers in two ways:
- Port-Mapped I/O: 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

Recall: Example Memory-Mapped Display Controller

Memory-Mapped:
- Hardware maps control registers and display memory into physical address space
  » Addresses set by HW jumpers or at boot time
- Simply writing to display memory (also called the "frame buffer") changes image on screen
  » Addr: 0x8000F000 — 0x8000FFFF
- Writing graphics description to cmd queue
  » Say enter a set of triangles describing 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

Simple interaction with DMA controller (from OSC book):

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 blocks to/from memory directly

Diagram showing a simple interaction with a DMA controller:
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 blocks to/from memory directly
- Simple interaction with DMA controller (from OSC book)

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
- 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
- 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

Kernel Device Structure

Recall: 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
Recall: Life Cycle of An I/O Request

The Goal of the I/O Subsystem

- Provide Uniform Interfaces, Despite Wide Range of Different Devices
  - This code works on many different devices:
    ```c
    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
  - Read may return nothing, write may write nothing
- Asynchronous Interface: "Tell Me Later"
  - When request data, take pointer to user’s buffer, return immediately; later kernel fills buffer and notifies user
  - When send data, take pointer to user’s buffer, return immediately; later kernel takes data and notifies user
**Administrivia**

Sorry about infrastructure disaster yesterday!
- We extended the HW4 deadline
HW 5 will have a Rust option!
- Choose one or the other
Project 2 still due Friday
Midterm 3 on April 25
- All topics up to previous Tuesday (4/23) are in scope
- Closed book, 3 pages, double-sided handwritten notes.

**Storage Devices**

- **Magnetic disks**
  - Storage that rarely becomes corrupted
  - Large capacity at low cost
  - Block level random access (except for SMR – later!)
  - Slow performance for random access
  - Better performance for sequential access

- **Flash memory**
  - Storage that rarely becomes corrupted
  - Capacity at intermediate cost (5-20x disk)
  - Block level random access
  - Good performance for reads; worse for random writes
  - Erasure requirement in large blocks
  - Wear patterns issue

**Hard Disk Drives (HDDs)**

IBM/Hitachi Microdrive

- 30 MB hard disk - $500
- 30-40ms seek time
- 0.7-1 MB/s (est.)

**The Amazing Magnetic Disk**

Unit of Transfer: **Sector** (512B or 4096B)
- Ring of sectors form a **track**
- Stack of tracks form a **cylinder**
- Heads position on **cylinders**

Disk Tracks ~ 1µm (micron) wide
- Wavelength of light is ~ 0.5µm
- Resolution of human eye: 50µm
- 100K tracks on a typical 2.5” disk

Separated by unused guard regions
- Reduces likelihood neighboring tracks are corrupted during writes (still a small non-zero chance)
The Amazing Magnetic Disk

Track length varies across disk
- Outside: More sectors per track, higher bandwidth
- Disk is organized into regions of tracks with same # of sectors/track
- Only outer half of radius is used
  » Most of the disk area in the outer regions of the disk

OS Unit of Transfer: Block
- Typically more than one Sector
- Example: 4KB, 16KB

Disks so big that some companies (like Google) reportedly only use part of disk for active data
- Rest is archival data

Shingled Magnetic Recording (SMR)

Overlapping tracks yields greater density, capacity
Restrictions on writing, complex DSP for reading

Magnetic Disk Performance

Cylinders: 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 cylinder
- Rotational latency: wait for desired sector to rotate under r/w head
- Transfer time: transfer a block of bits (sector) under r/w head

Disk Latency = Queueing Time + Controller time + Seek Time + Rotation Time + Xfer Time

Typical Numbers for Magnetic Disk

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Info/Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>A/Density</td>
<td>Space: 18TB (Seagate), 9 platters, in 3½ inch form factor!</td>
</tr>
<tr>
<td>Areal Density</td>
<td>≥ 1 Terabit/square inch! (PMR, Helium, ...)</td>
</tr>
<tr>
<td>Ge Seek Time</td>
<td>Typically 4-6 milliseconds</td>
</tr>
<tr>
<td>Ge Rotational Latency</td>
<td>Most laptop/desktop disks rotate at 3600-7200 RPM (16-8 ms/rotation). Server disks up to 15,000 RPM. Average latency is halfway around disk so 4-8 milliseconds</td>
</tr>
<tr>
<td>Xfer Time</td>
<td>Depends on controller hardware</td>
</tr>
<tr>
<td></td>
<td>Typically 50 to 270 MB/s. Depends on:</td>
</tr>
<tr>
<td></td>
<td>• Transfer size (usually a sector): 512B – 1KB per sector</td>
</tr>
<tr>
<td></td>
<td>• Rotation speed: 3600 RPM to 15000 RPM</td>
</tr>
<tr>
<td></td>
<td>• Recording density: bits per inch on a track</td>
</tr>
<tr>
<td></td>
<td>• Diameter: ranges from 1 in to 5.25 in</td>
</tr>
</tbody>
</table>

Used to drop by a factor of two every 1.5 years (or faster), now slowing do
Disk Performance Example

Assumptions:
- Ignoring queuing and controller times for now
- Avg seek time of 5ms
- 7200RPM \( \Rightarrow \) Time for rotation: \( \frac{60000 \text{ (ms/min)}}{7200 \text{ (rev/min)}} = 8 \text{ ms} \)
  
  Avg time to find block = \( \frac{1}{2} \times 8 \text{ ms} = 4 \text{ ms} \)
- Transfer rate of 50MByte/s, block size of 4Kbyte \( \Rightarrow \)
  
  \( \frac{4096 \text{ bytes}}{50 \times 10^6 \text{ (bytes/s)}} = 81.92 \times 10^{-6} \text{ sec} \cong 0.082 \text{ ms for 1 sector} \)

ead block from random place on disk:
- Seek (5ms) + Rot. Delay (4ms) + Transfer (0.082ms) = 9.082ms
- Approx 9ms to fetch/put data: \( \frac{4096 \text{ bytes}}{9.082 \times 10^{-3} \text{ s}} \cong 451 \text{ KB/s} \)

ead block from random place in same cylinder:
- Rot. Delay (4ms) + Transfer (0.082ms) = 4.082ms
- Approx 4ms to fetch/put data: \( \frac{4096 \text{ bytes}}{4.082 \times 10^{-3} \text{ s}} \cong 1.03 \text{ MB/s} \)

ead next block on same track:
- Transfer (0.082ms): \( \frac{4096 \text{ bytes}}{0.082 \times 10^{-3} \text{ s}} \cong 50 \text{ MB/sec} \)

way to using disk effectively is to minimize seek and rotational delays

Lots of Intelligence in the Controller

Sectors contain sophisticated error correcting codes
- Disk head magnet has a field wider than track
- Hide corruptions due to neighboring track writes

Sector sparing
- Remap bad sectors transparently to spare sectors on the same surface

Slip sparing
- Remap all sectors (when there is a bad sector) to preserve sequential behavior

Track skewing
- Sector numbers offset from one track to the next, to allow for disk head movement for sequential ops

When is Disk Performance Highest?

- When there are big sequential reads, or
- When there is so much work to do that they can be piggybacked (reordering queues—one moment)

- It is OK to be inefficient when things are mostly idle
- Bursts are both a threat and an opportunity
  
  <your idea for optimization goes here>
  
  – Waste space for speed?

- Other techniques:
  
  – Reduce overhead through user level drivers
  – Reduce the impact of I/O delays by doing other useful work in the meantime

Disk Scheduling (1/3)

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

  
  User Requests
  
  1
  2
  3
  4
  5
  6
  7
  8
  9
  10

  Head

  • FIFO Order
  
  – Fair among requesters, but order of arrival may be to random spots on the disk \( \Rightarrow \) 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
Disk Scheduling (2/3)

- Disk can do only one request at a time; What order do you choose to do queued requests?
- SCAN: Implements an Elevator Algorithm: take the closest request in the direction of travel
  – No starvation, but retains flavor of SSTF

Disk Scheduling (3/3)

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

Example of Current HDDs

- Seagate Exos X24 (2023)
  - 24 TB hard disk
  - 10 platters, 20 heads
  - 1.26 TB/in²
  - Helium filled: reduce friction and power
  - 4.16 ms average seek time
  - 4096 byte physical sectors
  - 7200 RPMs
  - Dual 6 Gbps SATA /12Gbps SAS Interface
  - 285MB/s MAX transfer rate
  - Cache size: 512MB
  - Price: $479 (~ $0.02/GB)

- IBM Personal Computer/AT (1986)
  - 30 MB hard disk
  - 30-40 ms average seek time
  - 0.7-1 MB/s (est.)
  - Price: $500 ($17K/GB)

Solid State Disks (SSDs)

- Replace rotating magnetic media with on-volatile memory (battery backed DRAM)
- Use NAND Multi-Level Cell (2 or 3-bit/cell) flash memory
  - Sector (4 KB page) addressable, but stores 4-64 “pages” per memory block
  - Trapped electrons distinguish between 1 and 0
  - No moving parts (no rotate/seek motors)
  - Eliminates seek and rotational delay (0.1-0.2ms access time)
  - Very low power and lightweight
  - Limited “write cycles”
- Rapid advances in capacity and cost ever since!
FLASH Memory

Like a normal transistor but:
- Has a floating gate that can hold charge
- To write: raise or lower wordline high enough to cause charges to tunnel
- To read: turn on wordline as if normal transistor
  » presence of charge changes threshold and thus measured current

Two varieties:
- NAND: denser, must be read and written in blocks
- NOR: much less dense, fast to read and write

V-NAND: 3D stacking (Samsung claims 1TB possible in 1 chip)

Samsung 2015: 512GB, NAND Flash

Flash Memory (Con't)

- Data read and written in page-sized chunks (e.g. 4K)
  » Cannot be addressed at byte level
  » Random access at block level for reads (no locality advantage)
  » Writing of new blocks handled in order (kinda like a log)
- Before writing, must be erased (256K block at a time)
  » Requires free-list management
  » CANNOT write over existing block (Copy-on-Write is normal case)

SSD Architecture – Reads

- ad 4 KB Page: ~25 usec
- No seek or rotational latency
- Transfer time: transfer a 4KB page
  » SATA: 300-600MB/s => ~4 x 10^3 b / 400 x 10^6 bps => 10 us
  » latency = Queuing Time + Controller time + Xfer Time
- Highest Bandwidth: Sequential OR Random reads

SSD Architecture – Writes

- Writing data to NAND Flash is complex!
- Can only write empty pages in a block (~ 200μs)
- Erasing a block takes ~1.5ms
- Controller maintains pool of empty blocks by coalescing used pages (read, erase, write), also reserves some % of capacity
- Rule of thumb: writes 10x reads, erasure 10x writes
- Ds provide same interface as HDDs: read I write chunk (4KB) at a time
  » not just erase and rewrite new version of the 256KB block?
- Erasure is very slow (milliseconds)
- Each block has a finite lifetime, can only be erased and rewritten about 10K times
- Heavily used blocks likely to wear out quickly

Solution – Two Systems Principles

- Layer of Indirection
  - Maintain a Flash Translation Layer (FTL) in SSD
  - Map virtual block numbers (which OS uses) to physical page numbers (which flash mem. controller uses)
  - Can now freely relocate data w/o OS knowing
- Copy on Write
  - Don’t overwrite a page when OS updates its data
  - Instead, write new version in a free page
  - Update FTL mapping to point to new location

Flash Translation Layer

- No need to erase and rewrite entire 256KB block when making small modifications
- SSD controller can assign mappings to spread workload across pages
  - Wear Levelling
- What to do with old versions of pages?
  - Garbage Collection in background
  - Erase blocks with old pages, add to free list

Some “Current” (large) 3.5 in SSDs

- Seagate Exos SSD: 15.36TB (2017)
  - Dual 12Gb/s interface
  - Seq reads 860MB/s
  - Seq writes 920MB/s
  - Random Reads (IOPS): 102K
  - Random Writes (IOPS): 15K
  - Price (Amazon): $5495 ($0.36/GB)
- Nimbus SSD: 100TB (2019)
  - Dual port: 12Gb/s interface
  - Seq reads/writes: 500MB/s
  - Random Read Ops (IOPS): 100K
  - Unlimited writes for 5 years!
  - Price: ~ $40K? ($0.4/GB)
    » However, 50TB drive costs $12500 ($0.25/GB)

HDD vs. SSD Comparison

SSD prices drop faster than HDD
Amusing calculation:
Is a full Kindle heavier than an empty one?

Actually, “Yes”, but not by much
Flash works by trapping electrons:
- So, erased state lower energy than written state
Assuming that:
- Kindle has 4GB flash
- ½ of all bits in full Kindle are in high-energy state
- High-energy state about $10^{-15}$ joules higher
- Then: Full Kindle is 1 attogram ($10^{-18}$ gram) heavier
  (Using $E = mc^2$)
Of course, this is less than most sensitive scale can measure (it can measure $10^{-9}$ grams)
Of course, this weight difference overwhelmed by battery discharge, weigh from getting warm, ….


SSD Summary

Pros (vs. hard 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), expensive (3-20x disk)
  » Hybrid alternative: combine small SSD with large HDD

Nano-Tube Memory (NANTERO)

Yet another possibility: Nanotube memory
- Nanotubes between two electrodes, slight conductivity difference between on and zeros
- No wearout!
Better than DRAM?
- Speed of DRAM, no wearout, non-volatile!
- Nantero promises 512Gb/dice for 8Tb/chip! (with 16 die stacking)
Conclusion (1/2)

Notification mechanisms
- Interrupts
- Polling: Report results through status register that processor looks at periodically

Device drivers interface to I/O devices
- Provide clean Read/Write interface to OS above
- Manipulate devices through PIO, DMA & interrupt handling
- Three types: block, character, and network

Direct Memory Access (DMA)
- Permit devices to directly access memory
- Free up processor from transferring every byte

Conclusion (2/2)

Disk Performance:
- Queuing time + Controller + Seek + Rotational + Transfer
- Rotational latency: on average ½ rotation
- Transfer time: spec of disk depends on rotation speed and bit storage density

Devices have complex interaction and performance characteristics
- Response time (Latency) = Queue + Overhead + Transfer
  » Effective BW = BW * T/(S+T)
- HDD: Queuing time + controller + seek + rotation + transfer
- SSD: Queuing time + controller + transfer (erasure & wear)

Systems (e.g., file system) designed to optimize performance and reliability
- Relative to performance characteristics of underlying device

Next time: Bursts & High Utilization introduce queuing delays

Next time: Queuing Latency:
- M/M/1 and M/G/1 queues: simplest to analyze
- As utilization approaches 100%, latency \( \rightarrow \infty \)
  \[ T_q = T_{ser} \cdot \frac{1}{2}(1+C) \cdot \frac{\rho}{(1 - \rho)} \]