[537] I/O Devices/Disks

Tyler Harter
I/O Devices
Motivation

What good is a computer without any I/O devices?
- keyboard, display, disks

We want:
- **H/W** that will let us plug in different devices
- **OS** that can interact with different combinations
Motivation

What good is a computer without any I/O devices?
- keyboard, display, disks

We want:
- **H/W** that will let us plug in different devices
- **OS** that can interact with different combinations

*Largely a communication problem...*
CPU

RAM

Graphics

Memory Bus

General I/O Bus (e.g., PCI)

Peripheral I/O Bus (e.g., SCSI, SATA, USB)
Why use hierarchical buses?

- Memory Bus
- General I/O Bus (e.g., PCI)
- Peripheral I/O Bus (e.g., SCSI, SATA, USB)
Canonical Device

Device Registers:  

<table>
<thead>
<tr>
<th>Status</th>
<th>COMMAND</th>
<th>DATA</th>
</tr>
</thead>
</table>
OS reads/writes to these device registers:

<table>
<thead>
<tr>
<th>Status</th>
<th>COMMAND</th>
<th>DATA</th>
</tr>
</thead>
</table>
Canonical Device

Device Registers:

<table>
<thead>
<tr>
<th>Status</th>
<th>COMMAND</th>
<th>DATA</th>
</tr>
</thead>
</table>

Hidden Internals:

???
Canonical Device

Device Registers:

<table>
<thead>
<tr>
<th>Status</th>
<th>COMMAND</th>
<th>DATA</th>
</tr>
</thead>
</table>

Hidden Internals:

- Microcontroller (CPU+RAM)
- Extra RAM
- Other special-purpose chips

OS reads/writes to these
Example Protocol

while (STATUS == BUSY)
    ; // spin
Write data to DATA register
Write command to COMMAND register
while (STATUS == BUSY)
    ; // spin
CPU:

Disk:

while (STATUS == BUSY) // 1
;
Write data to DATA register // 2
Write command to COMMAND register // 3
while (STATUS == BUSY) // 4
;
while (STATUS == BUSY)  // 1
;
Write data to DATA register  // 2
Write command to COMMAND register  // 3
while (STATUS == BUSY)  // 4
;

CPU:  A

Disk:  C
While (STATUS == BUSY) // 1
;
Write data to DATA register // 2
Write command to COMMAND register // 3
While (STATUS == BUSY) // 4
;
while (STATUS == BUSY) // 1
;
Write data to DATA register // 2
Write command to COMMAND register // 3
while (STATUS == BUSY) // 4
;
while (STATUS == BUSY) // 1
;
Write data to DATA register // 2
Write command to COMMAND register // 3
while (STATUS == BUSY) // 4
;
while (STATUS == BUSY) // 1
;
Write data to DATA register // 2
Write command to COMMAND register // 3
while (STATUS == BUSY) // 4
;
while (STATUS == BUSY);  // 1

;  // 2

Write data to DATA register

Write command to COMMAND register  // 3

while (STATUS == BUSY);  // 4

;  // 5

CPU:

Disk:

C A
while (STATUS == BUSY) // 1
;
Write data to DATA register // 2
Write command to COMMAND register // 3
while (STATUS == BUSY) // 4
;
while (STATUS == BUSY) // 1
;
Write data to DATA register // 2
Write command to COMMAND register // 3
while (STATUS == BUSY) // 4
;
while (STATUS == BUSY) // 1
;
Write data to DATA register // 2
Write command to COMMAND register // 3
while (STATUS == BUSY) // 4
;
while (STATUS == BUSY) // 1
;
Write data to DATA register  // 2
Write command to COMMAND register // 3
while (STATUS == BUSY)  // 4
;

how to avoid spinning?
interrupts!
how to avoid spinning?
interrupts!

while (STATUS == BUSY)     // 1
    wait for interrupt;
Write data to DATA register  // 2
Write command to COMMAND register // 3
while (STATUS == BUSY)     // 4
    wait for interrupt;
while (STATUS == BUSY) // 1
    wait for interrupt;
Write data to DATA register // 2
Write command to COMMAND register // 3
while (STATUS == BUSY) // 4
    wait for interrupt;

how to avoid spinning? interrupts!

CPU: A B A B A B

Disk: C A

1 2 3 4
Interrupts vs. Polling

Discuss: are interrupts ever worse?
Interrupts vs. Polling

Discuss: are interrupts ever worse?

Interrupts can sometimes lead to *livelock*
- e.g., flood of network packets
Interrupts vs. Polling

Discuss: are interrupts ever worse?

Interrupts can sometimes lead to **livelock**
- e.g., flood of network packets

Techniques:
- hybrid approach
- interrupt **coalescing**
Protocol Variants

Status checks: polling vs. interrupts

Data: PIO vs. DMA

Control: special instructions vs. memory-mapped I/O
while (STATUS == BUSY) // 1
    wait for interrupt;
Write data to DATA register // 2
Write command to COMMAND register // 3
while (STATUS == BUSY) // 4
    wait for interrupt;
while (STATUS == BUSY) // 1
    wait for interrupt;
Write data to DATA register // 2
Write command to COMMAND register // 3
while (STATUS == BUSY) // 4
    wait for interrupt;

what else can we optimize?
while (STATUS == BUSY)             // 1
    wait for interrupt;
Write data to DATA register        // 2
Write command to COMMAND register  // 3
while (STATUS == BUSY)             // 4
    wait for interrupt;

what else can we optimize?
data transfer!
Programmed I/O vs. Direct Memory Access

PIO (Programmed I/O):
- CPU directly tells device what data is

DMA (Direct Memory Access):
- CPU leaves data in memory
- DMA device does copy
PIO Flow

- CPU
- RAM
- Disk
DMA Flow

CPU  RAM

Disk
while (STATUS == BUSY)             // 1
  wait for interrupt;
Write data to DATA register        // 2
Write command to COMMAND register  // 3
while (STATUS == BUSY)             // 4
  wait for interrupt;
while (STATUS == BUSY) // 1
    wait for interrupt;
initiate DMA transfer // 2a
wait for interrupt // 2b
Write command to COMMAND register // 3
while (STATUS == BUSY) // 4
    wait for interrupt;
Protocol Variants

Status checks: polling vs. interrupts

Data: PIO vs. DMA

Control: special instructions vs. memory-mapped I/O
while (STATUS == BUSY) // 1
    wait for interrupt;
Write data to DATA register // 2
Write command to COMMAND register // 3
while (STATUS == BUSY) // 4
    wait for interrupt;
while (STATUS == BUSY) // 1
    wait for interrupt;
Write data to DATA register // 2
Write command to COMMAND register // 3
while (STATUS == BUSY) // 4
    wait for interrupt;
how does OS read and write registers?
Special Instructions vs. Mem-Mapped I/O

Special instructions
- each device has a port
- `in/out` instructions (x86) communicate with device

Memory-Mapped I/O
- H/W maps registers into address space
- `loads/stores` sent to device

Doesn’t matter much (both are used).
static inline void
outb(ushort port, uchar data)
{
    asm volatile("out %0,%1" : : "a" (data), "d" (port));
}

// Start the request for b. Caller must hold idelock.
static void idestart(struct buf *b) {
    ...  
    outb(0x3f6, 0);  // generate interrupt  
    outb(0x1f2, 1);  // number of sectors  
    ...
}
int fd = open("/dev/mem", O_RDWR | O_SYNC);  
assert(fd >= 0);  
gpio = (uint32_t *)mmap(0, getpagesize(), ..., fd, 0x20200000);  
assert(gpio != NULL)  

//set gpio17 as an output  
*(gpio + 1) = (*(gpio + 1) & ~(7 << 21)) | (1 << 21);
Protocol Variants

**Status checks:** polling vs. interrupts

**Data:** PIO vs. DMA

**Control:** special instructions vs. memory-mapped I/O
Variety is a Challenge

Problem:
- many, many devices
- each has its own protocol

How can we avoid writing a slightly different OS for each H/W combination?
Solution

Encapsulation!

Write driver for each device.

Drivers are 70% of Linux source code.
Solution

Encapsulation!

Write driver for each device.

Drivers are 70% of Linux source code.

Encapsulation also enables us to mix-and-match devices, schedulers, and file systems.
Storage Stack

- application
- file system
- scheduler
- driver
- hard drive

build common interface on top of all HDDs
Storage Stack

- application
- file system
- scheduler
- driver
- hard drive

build common interface on top of all HDDs

what about special capabilities?
Hard Disks
Basic Interface

Disk has a sector-addressable address space (so a disk is like an array of sectors).

Sectors are typically 512 bytes or 4096 bytes.

Main operations: reads + writes to sectors (blocks).
Platter

Disk Internals
Platter is covered with a magnetic film.
Spindle
Many platters may be bound to the spindle.
Each surface is divided into rings called tracks. A stack of tracks (across platters) is called a cylinder.
The tracks are divided into numbered sectors.
Heads on a moving arm can read from each surface.
Spindle/platters rapidly spin.
Don’t try this at home!

http://youtu.be/9eMWG3fwiEU?t=30s
Let’s Read 12!
Seek to right track.
Seek to right track.
Seek to right track.
Wait for rotation.
Wait for rotation.
Wait for rotation.
Wait for rotation.
Wait for rotation.
Wait for rotation.
Transfer data.
Transfer data.
Transfer data.
Yay!
Seek, Rotate, Transfer

Must accelerate, coast, decelerate, settle

Seeks often take several milliseconds!

Settling alone can take 0.5 - 2 ms.

Entire seek often takes 4 - 10 ms.
Seek, Rotate, Transfer

Depends on rotations per minute (RPM).
- 7200 RPM is common, 15000 RPM is high end.

\[
\frac{1}{7200 \text{ RPM}} = \\
1 \text{ minute} / 7200 \text{ rotations} = \\
1 \text{ second} / 120 \text{ rotations} = \\
12 \text{ ms} / \text{ rotation}
\]
Seek, **Rotate**, Transfer

Depends on rotations per minute (RPM).
- **7200 RPM** is common, 15000 RPM is high end.

\[
\frac{1}{7200 \text{ RPM}} = \frac{1 \text{ minute}}{7200 \text{ rotations}} = \frac{1 \text{ second}}{120 \text{ rotations}} = \frac{12 \text{ ms}}{\text{rotation}}
\]

so it may take **6 ms** on avg to rotate to target \((0.5 \times 12 \text{ ms})\).
Seek, Rotate, Transfer

Pretty fast — depends on RPM and sector density.

100+ MB/s is typical.

$\frac{1 \text{ s}}{100 \text{ MB}} = 10 \text{ ms} / \text{MB} = 4.9 \text{ us} / \text{sector}$

(assuming 512-byte sector)
Workload

So...
- seeks are slow
- rotations are slow
- transfers are fast

What kind of workload is fastest for disks?
Workload

So…
- seeks are slow
- rotations are slow
- transfers are fast

What kind of workload is fastest for disks?
*Sequential*: access sectors in order (transfer dominated)
*Random*: access sectors arbitrarily (seek+rotation dominated)

Demos: example-rand.csh and example-seq.csh
# Disk Spec

<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>300 GB</td>
<td>1 TB</td>
</tr>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
<tr>
<td>Platters</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Cache</td>
<td>16 MB</td>
<td>32 MB</td>
</tr>
</tbody>
</table>
## Disk Spec

<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>300 GB</td>
<td>1 TB</td>
</tr>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
<tr>
<td>Platters</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Cache</td>
<td>16 MB</td>
<td>32 MB</td>
</tr>
</tbody>
</table>

Sequential workload: what is throughput for each?
## Disk Spec

<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>300 GB</td>
<td>1 TB</td>
</tr>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td><strong>125 MB/s</strong></td>
<td><strong>105 MB/s</strong></td>
</tr>
<tr>
<td>Platters</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Cache</td>
<td>16 MB</td>
<td>32 MB</td>
</tr>
</tbody>
</table>

Cheeta: 125 MB/s.
Barracuda: 105 MB/s.
## Disk Spec

<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>300 GB</td>
<td>1 TB</td>
</tr>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
<tr>
<td>Platters</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Cache</td>
<td>16 MB</td>
<td>32 MB</td>
</tr>
</tbody>
</table>

Random workload: what is throughput for each? (what else do you need to know?)
## Disk Spec

<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Capacity</strong></td>
<td>300 GB</td>
<td>1 TB</td>
</tr>
<tr>
<td><strong>RPM</strong></td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td><strong>Avg Seek</strong></td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td><strong>Max Transfer</strong></td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
<tr>
<td><strong>Platters</strong></td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td><strong>Cache</strong></td>
<td>16 MB</td>
<td>32 MB</td>
</tr>
</tbody>
</table>

Random workload: what is throughput for each? Assume 16-KB reads.
## Disk Spec

<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>300 GB</td>
<td>1 TB</td>
</tr>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
<tr>
<td>Platters</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Cache</td>
<td>16 MB</td>
<td>32 MB</td>
</tr>
</tbody>
</table>

Random workload: what is throughput for each? Assume 16-KB reads.
<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
</tbody>
</table>

How long does an average 16-KB read take w/ Cheetah?
<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
</tbody>
</table>

How long does an average 16-KB read take w/ Cheetah?

\[
\text{avg rotation} = \frac{1}{2} \times \frac{1 \text{ min}}{15000}
\]
<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
</tbody>
</table>

How long does an average 16-KB read take w/ Cheetah?

$$\text{avg rotation} = \frac{1}{2} \times \frac{1 \text{ min}}{15000} \times \frac{60 \text{ sec}}{1 \text{ min}} \times \frac{1000 \text{ ms}}{1 \text{ sec}}$$
<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RPM</strong></td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td><strong>Avg Seek</strong></td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td><strong>Max Transfer</strong></td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
</tbody>
</table>

How long does an average 16-KB read take w/ Cheetah?

\[
\text{avg rotation} = \frac{1}{2} \times \frac{1 \text{ min}}{15000} \times \frac{60 \text{ sec}}{1 \text{ min}} \times \frac{1000 \text{ ms}}{1 \text{ sec}} = 2 \text{ ms}
\]
How long does an average 16-KB read take with Cheetah?

Transfer = \( \frac{1 \text{ sec}}{125 \text{ MB}} \times 16 \text{ KB} \)
How long does an average 16-KB read take with Cheetah?

\[
\text{transfer} = \frac{1 \text{ sec}}{125 \text{ MB}} \times 16 \text{ KB} \times \frac{1,000,000 \text{ us}}{1 \text{ sec}}
\]
Cheetah Barracuda
RPM 15,000 7,200
Avg Seek 4 ms 9 ms
Max Transfer 125 MB/s 105 MB/s

How long does an average 16-KB read take w/ Cheetah?

\[
\text{transfer} = \frac{1 \text{ sec}}{125 \text{ MB}} \times 16 \text{ KB} \times \frac{1,000,000 \text{ us}}{1 \text{ sec}} = 125 \text{ us}
\]
How long does an average 16-KB read take with Cheetah?

Cheetah time = 4ms + 2ms + 125us = 6.1ms
<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
</tbody>
</table>

How long does an average 16-KB read take with Cheetah?

Cheetah time = 4ms + 2ms + 125us = 6.1ms

throughput = \( \frac{16 \text{ KB}}{6.1\text{ms}} \)
<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
</tbody>
</table>

How long does an average 16-KB read take w/ Cheetah?

Cheetah time = 4ms + 2ms + 125us = 6.1ms

throughput = \[
\frac{16 \text{ KB}}{6.1\text{ms}} \times \frac{1 \text{ MB}}{1024 \text{ KB}} \times \frac{100 \text{ ms}}{1 \text{ sec}}
\]
<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
</tbody>
</table>

How long does an average 16-KB read take with Cheetah?

Cheetah time = 4ms + 2ms + 125us = 6.1ms

throughput = \[
\frac{16 \text{ KB}}{6.1\text{ ms}} \times \frac{1 \text{ MB}}{1024 \text{ KB}} \times \frac{100 \text{ ms}}{1 \text{ sec}} = 2.5 \text{ MB/s}
\]
<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RPM</strong></td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td><strong>Avg Seek</strong></td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td><strong>Max Transfer</strong></td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
</tbody>
</table>

How long does an average 16-KB read take w/ Barracuda?
<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
</tbody>
</table>

How long does an average 16-KB read take w/ Barracuda?

avg rotation = \( \frac{1}{2} \times \frac{1 \text{ min}}{7200} \)
<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RPM</strong></td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td><strong>Avg Seek</strong></td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td><strong>Max Transfer</strong></td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
</tbody>
</table>

How long does an average 16-KB read take w/ Barracuda?

\[
\text{avg rotation} = \frac{1}{2} \times \frac{1 \text{ min}}{7200} \times \frac{60 \text{ sec}}{1 \text{ min}} \times \frac{1000 \text{ ms}}{1 \text{ sec}}
\]
<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
</tbody>
</table>

How long does an average 16-KB read take w/ Barracuda?

\[
\text{avg rotation} = \frac{1}{2} \times \frac{1 \text{ min}}{7200} \times \frac{60 \text{ sec}}{1 \text{ min}} \times \frac{1000 \text{ ms}}{1 \text{ sec}} = 4.1 \text{ ms}
\]
<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
</tbody>
</table>

How long does an average 16-KB read take w/ Barracuda?

\[
\text{transfer} = \frac{1 \text{ sec}}{105 \text{ MB}} \times 16 \text{ KB}
\]
<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
</tbody>
</table>

How long does an average 16-KB read take w/ Barracuda?

\[
\text{transfer} = \frac{1 \text{ sec}}{105 \text{ MB}} \times 16 \text{ KB} \times \frac{1,000,000 \text{ us}}{1 \text{ sec}}
\]
<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
</tbody>
</table>

How long does an average 16-KB read take with Barracuda?

\[
\text{transfer} = \frac{1 \text{ sec}}{105 \text{ MB}} \times 16 \text{ KB} \times \frac{1,000,000 \text{ us}}{1 \text{ sec}} = 149 \text{ us}
\]
### Comparison Table

<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
</tbody>
</table>

How long does an average 16-KB read take w/ Barracuda?

Barracuda time = 9ms + 4.1ms + 149us = 13.2ms
### Comparison of Cheetah and Barracuda

<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
</tbody>
</table>

How long does an average 16-KB read take with Barracuda?

Barracuda time = 9ms + 4.1ms + 149us = 13.2ms

![Throughput Calculation](https://via.placeholder.com/150)

\[
\text{throughput} = \frac{16 \text{ KB}}{13.2 \text{ms}}
\]
<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
</tbody>
</table>

How long does an average 16-KB read take w/ Barracuda?

Barracuda time = 9ms + 4.1ms + 149us = 13.2ms

throughput = \[\frac{16 \text{ KB}}{13.2\text{ms}} \times \frac{1 \text{ MB}}{1024 \text{ KB}} \times \frac{1000 \text{ ms}}{1 \text{ sec}}\]
<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
</tbody>
</table>

How long does an average 16-KB read take w/ Barracuda?

Barracuda time = 9ms + 4.1ms + 149us = 13.2ms

Throughput = \[\left(\frac{16\ \text{KB}}{13.2\ \text{ms}}\right) \times \left(\frac{1\ \text{MB}}{1024\ \text{KB}}\right) \times \left(\frac{1000\ \text{ms}}{1\ \text{sec}}\right) = 1.2\ \text{MB/s}\]
<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
<tr>
<td>Random</td>
<td>2.5 MB/s</td>
<td>1.2 MB/s</td>
</tr>
</tbody>
</table>
Other Improvements

Track Skew

Zones

Cache
Other Improvements

Track Skew

Zones

Cache
When reading 16 after 15, the head won’t settle quick enough, so we need to do a rotation.
When reading 16 after 15, the head won’t settle quick enough, so we need to do a rotation.

Demo: example-skew.csh
enough time to settle now
enough time to settle now

Demo: example-skew-fixed.csh
Other Improvements

Track Skew

Zones

Cache
Performance implications?
Demo: example-zones-outer.csh
Other Improvements

Track Skew

Zones

Cache
Drive Cache

Drives may cache both reads and writes.

OS does this to.

What advantage does drive have for reads?

What advantage does drive have for writes?
Schedulers
Schedulers

Given a stream of requests, in what order should they be served?
FCFS (First-Come-First-Serve)

Assume seek+rotate = \textbf{10 ms} on average.
Assume transfer = \textbf{100 MB/s}.

How long (roughly) does the below workload take?
The integers are sector numbers.

\textbf{300001, 700001, 300002, 700002, 300003, 700003}
FCFS (First-Come-First-Serve)

Assume seek+rotate = 10 ms on average.
Assume transfer = 100 MB/s.

How long (roughly) does the below workload take? The integers are sector numbers.

300001, 700001, 300002, 700002, 300003, 700003 (~60ms)
FCFS (First-Come-First-Serve)

Assume seek+rotate = 10 ms on average.
Assume transfer = 100 MB/s.

How long (roughly) do the below workloads take?
The integers are sector numbers.

300001, 700001, 300002, 700002, 300003, 700003 (~60ms)
300001, 300002, 300003, 700001, 700002, 700003
FCFS (First-Come-First-Serve)

Assume seek+rotate = 10 ms on average. Assume transfer = 100 MB/s.

How long (roughly) do the below workloads take? The integers are sector numbers.

300001, 700001, 300002, 700002, 300003, 700003 (~60ms)
300001, 300002, 300003, 700001, 700002, 700003 (~20ms)
Schedulers

os

Disk
Schedulers

Where should the scheduler go?
SPTF (Shortest Positioning Time First)

**Strategy**: always choose the request that will take the least time for seeking and rotating.

How to implement in disk?
How to implement in OS?
SPTF (Shortest Positioning Time First)

**Strategy**: always choose the request that will take the least time for seeking and rotating.

How to implement in disk?
How to implement in OS?

Disadvantages?
SPTF (Shortest Positioning Time First)

**Strategy**: always choose the request that will take the least time for seeking and rotating.

How to implement in disk?
How to implement in OS?

Disadvantages?

**Starvation**: example-starve.csh, example-starve-bsptf.csh
**Greedy**: example-greedy-sptf.csh, example-greedy-optimal.csh
SCAN

Sweep back and forth, from one end of disk to the other, serving requests as you go.

Pros/Cons?
SCAN

Sweep back and forth, from one end of disk to the other, serving requests as you go.

Pros/Cons?

Better: C-SCAN (circular scan)
  - only sweep in one direction
What happens?

Assume 2 processes, and C-SCAN.

```c
void reader(int fd) {
    char buf[1024];
    int rv;
    while((rv = read(buf)) != 0) {
        assert(rv);
        // takes short time, e.g., 1ms
        process(buf, rv);
    }
}
```
Work Conservation

Work conserving schedulers always try to do I/O if there’s I/O to be done.

Sometimes, it’s better to wait instead if you anticipate another request will appear nearby.

Such non-work-conserving schedulers are called anticipatory schedulers.
CFQ (Linux Default)

Completely Fair Queueing.

Queue for each process.

Do weighted round-robin between queues, with slice time proportional to priority.

Optimize order within queue.

Yield slice only if idle for a given time (anticipation).
Summary

Overlap I/O and CPU whenever possible!
- use interrupts, DMA

Never do random I/O unless you must!
- e.g., Quicksort is a terrible algorithm on disk