Goals for Today

- **Learning Objective:**
  - Understand and evaluate disk scheduling algorithms

- **Announcements, etc:**
  - Midterm scores and debrief this week (sorry!)
  - MP2 extension: **now due on March 25th (UTC-11)**
  - MP3 released March 27th
  - MP2.5 (Extra Credit) release on March 27th also

Reminder: Please put away devices at the start of class
CS 423
Operating System Design:
Disk Scheduling Algorithms

Professor Adam Bates
Results for Quantitative Questions on the IEF form:

- Lectures are helpful always? 3.75 (stdev=1.1)
- Textbook is helpful always? 2.58 (stdev=1.77)
- Piazza is helpful always? 2.89 (stdev=1.1)
- Pace of the course is too easy? 3.06 (stdev=0.82)
- MPs are too easy? 2.90 (stdev=0.74)
- C4 is too easy? 2.44 (stdev=1.01)
I did some quick and dirty coding of the open responses for **Best Part** and **Worst Part** of the class:

**Best Things About the Course**

- Lectures

**Worst Things About the Course**

- Lectures Too Imprecise/Undetailed/Lack Examples
Informal Early Feedback

Here are a few suggestions for improving the course *this semester* that I liked:

- Lecture on OS support for Containers, Namespaces
- Lecture on Device Drivers (MP isn’t practical)
- Improved feedback for MPs
- Increase real-world examples and precision in lectures
- Post (drafts of) lecture slides in advance of class

Some suggestions I will look into for future semesters:

- Changing course to 2 days/week
- Device Drivers MP
- Simplify MP submission
- Better integrate course textbook
- Reduce 241 overlap

Thank you!
Why Files?

- Physical reality
  - Block oriented
  - Physical sector #s
  - No protection among users of the system
  - Data might be corrupted if machine crashes

- Filesystem model
  - Byte oriented
  - Named files
  - Users protected from each other
  - Robust to machine failures
Question

- What functions should file systems provide?
File System Requirements

- Users must be able to:
  - create and delete files at will.
  - read, write, and modify file contents with a minimum of fuss about blocking, buffering, etc.
  - share each other's files with proper authorization
  - refer to files by symbolic names.
  - see a logical view of files without concern for how they are stored.
  - retrieve backup copies of files lost through accident or malicious destruction.
Which disk request is serviced first?

- FCFS
- Shortest seek time first
- SCAN (Elevator)
- C-SCAN (Circular SCAN)

A: Track.
B: Sector.
C: Sector of Track.
D: File

**Disk Scheduling Decision** — Given a series of access requests, on which track should the disk arm be placed next to maximize fairness, throughput, etc?
FIFO (FCFS) Order

- **Method**
  - First come first serve

- **Pros?**
  - Fairness among requests
  - In the order applications expect

- **Cons?**
  - Arrival may be on random spots on the disk (long seeks)
  - Wild swings can happen

- **Analogy:**
  - FCFS elevator scheduling?
SSTF (Shortest Seek Time First)

- **Method**
  - Pick the one closest on disk

- **Pros?**
  - Tries to minimize seek time

- **Cons?**
  - Starvation

- **Questions**
  - Is SSTF optimal?
  - Is this fair to all disk accesses?
  - Are we worried about sorting overhead?
  - Can we avoid starvation?

98, 183, 37, 122, 14, 124, 65, 67
(65, 67, 37, 14, 98, 122, 124, 183)
SCAN (Elevator)

- Method
  - Take the closest request in the direction of travel

- Pros
  - Bounded time for each request

- Cons
  - Request at the other end will take a while

- Question
  - Is this fair to all disk accesses?
  - How to fix?

```
98, 183, 37, 122, 14, 124, 65, 67
(37, 14, 65, 67, 98, 122, 124, 183)
```
C-SCAN (Circular SCAN)

- **Method**
  - Like SCAN
  - But, wrap around

- **Pros**
  - Uniform service time

- **Cons**
  - Do nothing on the return (i.e., higher overhead)

![Diagram of disk tracks and time]
## Scheduling Algorithms

<table>
<thead>
<tr>
<th>Algorithm Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCFS</td>
<td>First-come first-served</td>
</tr>
<tr>
<td>SSTF</td>
<td>Shortest seek time first; process the request that reduces next seek time</td>
</tr>
<tr>
<td>SCAN (aka Elevator)</td>
<td>Move head from end to end (has a current direction)</td>
</tr>
<tr>
<td>C-SCAN</td>
<td>Only service requests in one direction (circular SCAN)</td>
</tr>
<tr>
<td>LOOK</td>
<td>Similar to SCAN, but do not go all the way to the end of the disk.</td>
</tr>
<tr>
<td>C-LOOK</td>
<td>Circular LOOK. Similar to C-SCAN, but do not go all the way to the end of the disk.</td>
</tr>
</tbody>
</table>
What factors impact disk performance?
- Seek Time: Time taken to move disk arm to a specified track
- Rotational Latency: Time taken to rotate desired sector into position
- Transfer Time: Time to read/write data. Depends on rotation speed of disk and transfer amount.

Disk Access Time = Seek Time + Rotational Latency + Transfer Time (+ Controller Latency)
Disk Access Time Example

- **Disk Parameters**
  - Transfer Size is 8K bytes
  - Advertised average seek time is 12 ms
  - Disk spins at 7200 RPM
  - Transfer rate is 4 MB/sec
  - Controller Overhead is 2 ms

- **Assume idle disk (i.e., no queuing delay)**

  Disk Access Time = 12 ms
  + \( \frac{0.5}{(7200 \text{ RPM} / 60)} \)
  + \( \frac{8 \text{ KB}}{4 \text{ MB per sec}} \)
  + 2 ms
Disk Access Time Example

- Disk Parameters
  - Transfer Size is 8K bytes
  - Advertised average seek time is 12 ms
  - Disk spins at 7200 RPM
  - Transfer rate is 4 MB/sec
  - Controller Overhead is 2 ms

- Assume idle disk (i.e., no queuing delay)

\[
\text{Disk Access Time} = 12 \text{ ms} + 4.15 \text{ ms} + 2 \text{ ms} + 2 \text{ ms} = 20 \text{ ms}
\]
Linux I/O Schedulers

• What disk (I/O) schedulers are available in Linux?

```sh
$ cat /sys/block/sda/queue/scheduler
noop deadline [cfq]
```

^ scheduler enabled on our VMs

• As of Linux 2.6.10, it is possible to change the I/O scheduler for a given block device on the fly!

• How to enable a specific scheduler?

```sh
$ echo SCHEDNAME > /sys/block/DEV/queue/scheduler
```

• SCHEDNAME = Desired I/O scheduler
• DEV = device name (e.g., sda)
Linux NOOP Scheduler

- Insert all incoming I/O requests into a simple FIFO
- Merges duplicate requests (results can be cached)
- When would this be useful?
• Insert all incoming I/O requests into a simple FIFO
• Merges duplicate requests (results can be cached)
• When would this be useful?
  • Solid State Drives! Avoids scheduling overhead
  • Scheduling is handled at a lower layer of the I/O stack (e.g., RAID Controller, Network-Attached)
  • Host doesn’t actually know details of sector positions (e.g., RAID controller)
• Imposes a deadline on all I/O operations to prevent starvation of requests

• Maintains 4 queues:
  • 2 Sorted Queues (R, W), order by Sector
  • 2 Deadline Queues (R, W), order by Exp Time

• Scheduling Decision:
  • Check if 1st request in deadline queue has expired.
  • Otherwise, serve request(s) from Sorted Queue.
  • Prioritizes reads (DL=500ms) over writes (DL=5s). Why?
Linux CFQ Scheduler

- CFQ = Completely Fair Queueing!
- Maintain per-process queues.
- Allocate time slices for each queue to access the disk
- I/O Priority dictates time slice, # requests per queue
- Asynchronous requests handled separately — batched together in priority queues
- CFQ is often the default scheduler
• **Deceptive Idleness:** A process appears to be finished reading from disk, but is actually processing data. Another (nearby) request is coming soon!

• Bad for synchronous read workloads because seek time is increased.

• **Anticipatory Scheduling:** Idle for a few milliseconds after a read operation in *anticipation* of another close-by read request.

• Deprecated — CFQ can approximate.