1119 Eviction Wrap-up & Storage devices

LRU ⇒ Clock

Clode doesn't consider the cost of existing frames.

⇒ Clean copy on disk (code page, unchanged data)

Second Chance / Enhanced Clock

⇒ Access bit, dirty bit (1 if written to)

⇒ A, D

0 0 (it has't been accessed recently, it's a clean page)
1 0 (clear the access bit)
0 1 (clear the dirty bit, move on)
1 1 (dirty page list)
(keep access bit & move on)
Storage Devices (I/O Devices)

→ persistent (nonvolatile)

→ hard drive / Spinning disk (HDD)
  • low cost ($10-20 per TB), large capacity
  • physical moving parts, slow access (10-20ms)

→ solid state drive (SSD)
  • higher cost ($3 per disk, $60-100 per TB), large capacity
  • no physical moving parts, ms scale access latency (50-100ms)
Disk Anatomy
→ sector addressable (unit of read/write)
512 bytes
→ has error correcting code.

Disk read steps:
① kernel send the request (ide.c, iderw)
② disk find right platter & surface
③ move arm to track, wait for desired sector to be under the head
④ disk head reads & transfers data back to the kernel.
Disk Performance

\[ \text{total time for a request} = \text{seek time} + \text{rotation time} + \text{transfer time} \]

(arm to track) (sector under disk head) (data read/write)

1. Seek time: 1-29 ms depending on how far to seek (let's say 10 ms)

2. Rotation time: specified as RPM, eg. 7200 RPM
   (assume it takes \( \frac{1}{2} \) rotation for the desired sector to be in the right place, 4 ms)
   \( \frac{\text{sector}}{\text{\degree}} = 0.005 \text{ ms} \)

3. Transfer time: specified as disk bandwidth, eg. 100 MB/s = 100 MB/s = 5 \( \mu \text{s} \) per sector
   \( \text{sector} \times 0.005 \text{ ms} \)

---

Sequential vs. Random access

<table>
<thead>
<tr>
<th>Access 10 consecutive sectors</th>
<th>Access 10 random sectors</th>
</tr>
</thead>
<tbody>
<tr>
<td># of seeks? 1</td>
<td># of seeks? 10</td>
</tr>
<tr>
<td>10 ms + 4 ms + 0.005 x 10 = 14.05 ms</td>
<td>10 x 10 + 4 x 10 + 0.5 x 10 = 140.05 ms</td>
</tr>
</tbody>
</table>

Access pattern matters!
Disk Scheduling

- Reorder I/O requests for better performance
- Shortest seek time first
  - Serve the request with the shortest seek-time next (closest)
  - Starvation!
- Elevator/Scan family (SCAN, CSCAN, R-CSCAN)
  - SCAN: arm moves from innermost to outermost, the outermost to innermost, serving request along the way.
  - CSCAN: arm moves from innermost to outermost, once it reaches the end, moves back to innermost & start again
  - R-CSCAN: takes rotation delay into account