I/O MANAGEMENT

Ken Wong
Washington University, St. Louis

kenw@arl.wustl.edu
www.arl.wustl.edu/~kenw

EVOLUTION OF I/O FUNCTION

<table>
<thead>
<tr>
<th>Controller</th>
<th>Control/Program</th>
<th>DMA</th>
<th>Interrupt</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>CPU/Main Memory</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>CPU/Main Memory</td>
<td>No</td>
<td>No</td>
<td>Prog. I/O</td>
</tr>
<tr>
<td>Yes</td>
<td>CPU/Main Memory</td>
<td>No</td>
<td>After Word</td>
<td>Prog. I/O</td>
</tr>
<tr>
<td>Yes</td>
<td>CPU/Main Memory</td>
<td>Yes</td>
<td>After Block</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>Cntrlr/Main Memory</td>
<td>Yes</td>
<td>After Block</td>
<td>I/O Channel</td>
</tr>
<tr>
<td>Yes</td>
<td>Cntrlr/Cntrlr Mem.</td>
<td>Yes</td>
<td>After Block</td>
<td>I/O Processor</td>
</tr>
</tbody>
</table>

DMA CONFIGURATIONS

I/O OPERATION OVERHEAD

- Example: Transfer 1K (1,024) words
- Programmed I/O
  - 1,024 loads, 1,024 stores, 1,024 busy-wait loop executions
- Interrupts
  - 1,024 loads, 1,024 stores, 1,024 interrupts
- DMA
  - 1 interrupt (when transfer is done)
  - DMA controller steals bus cycles from CPU
**Layered I/O Architecture**

- **User Process**
- **Directory Structure Path Name**
- **File System File Name/Offset**
- **Physical Organization Partition/Block Number**
- **Device I/O Absolute Block Number**
- **Hardware Cylinder/Head/Sector**

**Disk Geometry**

- **Areal Density** = BPI x TPI
- 2.5” or 3.5” platters
- 7200, 10000 RPM
- 3600, 5400, 5600, 6400, 7200, 10000 RPM

- Areal density (and capacity) has been doubling every 2-3 years

**Physical Disk Management**

- **Physical Formatting**
  - Create (on the disk surface) the electronic patterns that define the smallest data transfer unit (sector)
  - Header: Sector number, bad sector information
  - Body: Data (sector contents)
  - Trailer: Error detection/correction

- Create an initial disk label
  - Define partitions (cylinder groupings)
  - Record disk geometry (# cylinders, # heads, # sectors/track)

- **Bad sector** (a sector with at least 1 bad bit)
  - **Sector Forwarding**: Chain bad sector to a sector in extra cylinder
  - **Sector Slipping**: Forward shift header information on a track

**Disk Label**

- `sd0`: SCSI OCS controller at addr f0800000, unit # 24
  - 2036 cylinders 14 heads 72 sectors/track
  - a: 66528 sectors (66 cyls)
    - starting cylinder 0
  - b: 266112 sectors (264 cyls)
    - starting cylinder 66
  - c: 2052288 sectors (2036 cyls)
    - starting cylinder 0
  - g: 392112 sectors (389 cyls)
    - starting cylinder 330
  - h: 1327536 sectors (1317 cyls)
    - starting cylinder 719
**DISK REQUEST TIME**

- **Seek Time**
  
  \[ T_{\text{seek}} = k_0 + N_{\text{cylinders}} k_1 \]

- **Rotational Latency (Delay)**
  
  \[ T_{\text{rotation}} \sim \text{Uniform} \left( 0, T_{\text{rotation}} \right) \]

- **Transfer Time**
  
  \[ T_{\text{transfer}} = \frac{L_{\text{block}}}{C_{\text{track}}} \times T_{\text{rotation}} = \frac{L_{\text{block}}}{C_{\text{track}}} \times \frac{1}{S_{\text{rotation}}} = \frac{L_{\text{block}}}{S_{\text{rotation}} \times C_{\text{track}}} \]

- **Average Access Time**
  
  \[ T_{\text{access}} = T_{\text{seek}} + \frac{1}{S_{\text{rotation}}} + \frac{L_{\text{block}}}{S_{\text{rotation}} \times C_{\text{track}}} \]

**DISK SCHEDULING POLICIES**

- **Example** (Starting Cylinder = 100; Largest Cylinder Number = 199)
  
  - Cylinder Numbers in Request Queue: 55, 58, 39, 18, 90, 160, 150, 38
  - **FIFO (First-In-First-Out):** 55, 58, 39, 18, 90, 160, 150, 38
  - **SSTF (Shortest-Seek-Time-First):** 90, 58, 50, 39, 38, 18, 150, 160
  - **SCAN (Elevator Algorithm):** (right first) 150, 160, 199, 90, 55, 39, 38, 18, 0
    
    - Bidirectional: Increasing, goto max, decreasing, goto 0, ...
  - **C-SCAN (Circular SCAN):** (right first) 150, 160, 199, 0, 18, 38, 39, 55, 58, 90
    
    - Unidirectional: Increasing, goto max, goto 0, increasing, ...
  - **LOOK:** (right first) 150, 160, 90, 58, 55, 39, 38, 18
    
    - Don’t go to extreme cylinders unless necessary
  - **C-LOOK:** (right first) 150, 160, 18, 38, 39, 55, 58, 90

**DISK REQUEST TIME EXAMPLES**

- **Disk Parameters**
  
  - Average Seek Time: 20 ms
  - Rotation Speed: 7200 RPM (8.35 ms rotation time)
  - 512-Byte sectors
  - 64 sectors per track

- **Average time to read 128K bytes (Read Consecutive Sectors)**
  
  - Average Time (ms): \( 4 \times (4.18 + 8.35) = 50.12 \) ms

- **Average time to read 128K bytes (Read Random Sectors)**
  
  - Average Time (ms): \( 256 \times (20 + 4.18 + 0.13) = 6223.36 \) ms = 6.2 sec

- **Significant Effect: Location of the sectors**

**RAID**

(Redundant Array of Inexpensive Disks)

- Physical Disks
  
  - Disk 0
  - Disk 1
  - Disk 2
  - Disk m-1

- Logical Disk
  
  - Array Management Software and Controller
**THE RAID IDEA**

- **Redundant Array of Inexpensive (Independent) Disks**
  - Set of physical disk drives seen by OS as a single logical drive
  - Data distributed across physical drives of an array
  - Redundant disk space is used to store parity information which guarantees data recoverability in case of disk failure

- **Motivation**
  - Replace large, expensive disk drives with multiple, less-expensive, smaller-capacity drives

- **Tradeoff**
  - Increase *availability* at the cost of increased *unreliability* (probability of a single failure)

---

**RAID 0 (STRIPING)**

<table>
<thead>
<tr>
<th>Disk 0</th>
<th>Disk 1</th>
<th>Disk 2</th>
<th>Disk 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strip 0</td>
<td>Strip 1</td>
<td>Strip 2</td>
<td>Strip 3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>4n-4</td>
<td>4n-3</td>
<td>4n-2</td>
<td>4n-1</td>
</tr>
</tbody>
</table>

Read Request

- **Strips 0,1,2,3 = 1 Stripe**

- **Small versus large strip size?** Concurrent transfer versus concurrent I/O requests.

---

**RAID LEVELS**

<table>
<thead>
<tr>
<th>Level</th>
<th>Category</th>
<th>Strip Size</th>
<th>Redundancy</th>
<th>Commercial</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Striping</td>
<td>Var.</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>1</td>
<td>Mirroring</td>
<td>Var.</td>
<td>Duplication</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>Par. Access</td>
<td>Small</td>
<td>Hamming Code</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>Par. Access</td>
<td>Small</td>
<td>Bit-Interleaved Parity</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>Ind. Access</td>
<td>Large</td>
<td>Block-Interleaved Parity</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>Ind. Access</td>
<td>Large</td>
<td>Distributed RAID 4</td>
<td>Yes</td>
</tr>
</tbody>
</table>

---

**RAID 0**

- A **strip** is any convenient storage unit (e.g., N sectors)
- Data are **striped** across the disk drives
  - Example: Strips 0, 1, 2, 3 = Stripe 0
- Performance is a function of the request pattern and data layout

- **High Data Transfer Capacity**
  - Need high transfer capacity from drive to host memory
  - Need requests to be for large amounts of logically contiguous data
    - Overlap seek and transfer times of all disk drives
    - Compare to 1 seek followed by large transfer while other drives are idle

- **High I/O Request Rate**
  - Initiate concurrent, independent requests (each reading 1 strip)

- **Disadvantage**: One drive failure ⇒ Whole array is inaccessible
RAID 1 (MIRRORING)

- **Approach**
  - Distribute data in single bits over drives
  - Use Hamming Code to generate ECC (Error Correction Code) checksum bits.

- **Cons**
  - Need large number of ECC drives (4 ECC drives for 10 data drives)
  - Each sector of modern disks now have ECC
  - All the disadvantages of RAID 3 without any of its advantages

RAID 2 (STRIPING WITH ECC)

RAID 3 (BIT-INTERLEAVED PARITY)

- **Stripe bits or bytes over disks and compute parity over stripe**

- **Simple parity bit** computed for same bit position on all drives
  - For m=4 drives, parity bit i on drive 4 (the parity drive):
    \[ X4(i) = X0(i) \oplus X1(i) \oplus X2(i) \oplus X3(i) \]
  - 1 parity drive, no matter how large the disk array
  - Parallel access with data distributed in *small strips*

- **Recovery From Failure**
  - Reconstruct data from remaining drives until failed drive is replaced
  - Example: Drive X1 fails when m=4 drives
    \[ X4(i) \oplus X4(i) \oplus X1(i) = X0(i) \oplus X1(i) \oplus X2(i) \oplus X3(i) \oplus X4(i) \oplus X1(i) \]
    \[ X1(i) = X0(i) \oplus X2(i) \oplus X3(i) \oplus X4(i) \]
**RAID 3**

- **Write Request**
  - Example: $m=4$ drives and update a strip on drive $X1$
    
    \[
    X4'(i) = X0(i) \oplus X1'(i) \oplus X2(i) \oplus X3(i)
    \]
    
    \[
    x4'(i) = X0(i) \oplus X1'(i) \oplus X2(i) \oplus X3(i) \oplus X1(i) \oplus X1(i)
    \]
    
    \[
    X4'(i) = X4(i) \oplus X1(i) \oplus X1'(i)
    \]
  - 2 Reads and 2 Writes: Read $X4$; Read $X1$; Write $X4$; Write $X1$
  - Con:
    - Every read or write needs to access all drives of a set $\Rightarrow$ Only 1 pending request per disk set

**RAID 4 AND 5 (BLOCK-LEVEL PARITY)**

- **RAID 4**: Parity Disk $D_i = f(D_i, \text{all data disks})$
  - Stripe blocks over disks and compute block parity over stripe of blocks
  - Can read a single block in a stripe
  - Write requires waiting for parity block(s) to be written
    - Parity drive becomes a bottleneck
- **RAID 5**: Distribute parity blocks among all disks (Avoid parity disk bottleneck)
  - The most popular type today
  - Very good read performance
  - Write performance is better than RAID 4, but is still slow because of parity block.