I/O Management

- Categories of I/O devices and their integration with processor and bus
- Design of I/O subsystems
- I/O buffering
- Disk scheduling
- RAID

Disk Scheduling

- Disk performance is critical for system performance
- Management and ordering of disk access requests have strong influence on
  - access time
  - bandwidth
- Important to optimise because:
  - huge speed gap between memory and disk
  - disk throughput extremely sensitive to
    - request order ⇒ disk scheduling
    - placement of data on disk ⇒ file system design
- Request scheduler must be aware of disk geometry

Disk scheduling parameters:

- Disk is moving device ⇒ must position correctly for I/O
- Execution of a disk operation involves:
  - Wait time: the process waits to be granted device access
    - Wait for device: time the request spends in a wait queue
    - Wait for channel: time until a shared I/O channel is available
  - Access time: time the hardware needs to position the head
    - Seek time: position the head at the desired track
    - Rotational delay (latency): spin disk to the desired sector
  - Transfer time: sectors to be read/written rotate below the head
**Performance Parameters**

- **Seek time** $T_s$: Moving the head to the required track
  - Not linear in the number of tracks to traverse:
    - Startup and settling time
  - Typical average seek time: a few milliseconds

- **Rotational delay**:
  - Rotational speed, $r$, of 5,000 to 10,000 rpm
  - At 10,000 rpm, one revolution per 6 ms $\Rightarrow$ average delay 3 ms

- **Transfer time**:
  - To transfer $b$ bytes, with $N$ bytes per track:
    $$ T = \frac{b}{rN} $$
  - Total average access time:
    $$ T_a = T_s + \frac{1}{2r} + \frac{b}{rN} $$

**Disk Scheduling Policy**

**Observation from the calculation:**

- **Seek time** is the reason for differences in performance
- For a single disk there will be a number of I/O requests
- Processing in random order leads to worst possible performance
- We need better strategies

**A Timing Comparison:**

- $T_s = 2$ ms, $r = 10,000$ rpm, 512B sect, 320 sect/track
- Read a file with 2560 sectors ($= 1.3$ MB)
- File stored compactly (6 adjacent tracks):
  - Read first track
    - Average seek 2 ms
    - Rot. delay 3 ms
    - Read 320 sectors 6 ms
    - Total 11 ms
    - All sectors: $11 + 7 \times 9 = 74$ ms

- Sectors distributed randomly over the disk:
  - Read any sector
    - Average seek 2 ms
    - Rot. delay 3 ms
    - Read 1 sector 0.01875 ms
    - Total 5.01875 ms
    - All: $2560 \times 5.01875 = 12,848$ ms
First-in, first-out (FIFO):
- Process requests as they come in

Request tracks: 55, 58, 39, 18, 90, 160, 150, 38, 184

- Fair (no starvation)
- Good for few processes with clustered requests
- Deteriorates to random if there are many processes

Shortest Service Time First (SSTF):
- Select the request that minimises seek time

Request tracks: 55, 58, 39, 18, 90, 160, 150, 38, 184

- Service order: 90, 58, 55, 39, 18, 150, 160, 184
- Minimising locally may not lead to overall minimum!
- Can lead to starvation

SCAN (Elevator):
- Move head in one direction
  - Serves requests in track order until it reaches last track, then reverse direction

Request tracks: 55, 58, 39, 18, 90, 160, 150, 38, 184

- Service order: 150, 160, 184, (200), 90, 58, 55, 39, 18
- Similar to SSTF, but avoids starvation
- LOOK: variant of SCAN, moves head only to last request of one direction: 150, 160, 184, 90, 58, 55, 39, 18
- SCAN/LOOK are biased against region most recently traversed
- Favour innermost and outermost tracks
- Makes poor use of sequential reads (on down-scan)

Circular SCAN (C-SCAN):
- Like SCAN, but scanning to one direction only
  - When reaching last track, go back to first non-stop

Request tracks: 55, 58, 39, 18, 90, 160, 150, 38, 184

- Better use of locality (sequential reads)
- Better use of disk controller’s read-ahead cache
- Reduces the maximum delay compared to SCAN
N-step-SCAN:
- SSTF, SCAN & C-SCAN allow device monopolisation
  - process issues many requests to same track
- N-step-SCAN segments request queue:
  - subqueues of length \( N \)
  - process one queue at a time, using SCAN
  - added new requests to other queue

Slide 13

Disk scheduling algorithms:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSS</td>
<td>Random scheduling</td>
<td>For analysis and simulation</td>
</tr>
<tr>
<td>FIFO</td>
<td>First in, first out</td>
<td>Fairest</td>
</tr>
<tr>
<td>PRI</td>
<td>By process priority</td>
<td>Control outside disk mgmt</td>
</tr>
<tr>
<td>LIFO</td>
<td>Last in, first out</td>
<td>Maximise locality &amp; utilisation</td>
</tr>
</tbody>
</table>

Selection according to requestor

- SSTF Shortest seek time first
- SCAN Back and forth over disk
- C-SCAN One-way with fast return
- N-SCAN SCAN of N recs at once
- FSCAN N-SCAN (N=init. queue)

Selection according to requested item

- High utilisation, small queues
- Better service distribution
- Better worst-case time
- Service guarantee
- Load sensitive

Slide 15

Disk Scheduling

- Modern disks:
  - seek and rotational delay dominate performance
  - not efficient to read only few sectors
  - cache contains substantial part of currently read track
  - assume real disk geometry is same as virtual geometry
  - if not, controller can use scheduling algorithm internally

So, does OS disk scheduling make any difference at all?

Slide 16
**Linux 2.4.**

- Used a version of C-SCAN
- No real-time support
- Write and read handled in the same way — read requests have to be prioritised

**Slide 17**

**Linux 2.6.**

**Deadline I/O scheduler:**

- Two additional queues: FIFO read queue with deadline of 5ms, FIFO write with deadline of 500ms
- Request submitted to both queues
- If request expires, scheduler dispatches from FIFO queue

**Performance:**

- ✔ Seeks minimised
- ✔ Requests not starved
- ✔ Read requests handled faster
- ✘ Can result in seek storm, everything read from FIFO queues

**Slide 18**

**Anticipatory Scheduling:**

- Same, but anticipates dependent read requests
- After read request: waits for a few ms

**Performance**

- ✔ Can dramatically reduce the number of seek operations
- ✘ If no requests follow, time is wasted

**Slide 19**

**Performance**

- **Writes**
  - Similar for writes
  - Deadline scheduler slightly better than AS
- **Reads**
  - Deadline: About 10 times faster for reads
  - As: 100 times faster for streaming reads

**Slide 20**
Disk Cache

- Buffer in main memory for disk sectors
- Contains a copy of some of the sectors on the disk

Design Considerations:
- Transfer of data from cache to process memory
- Using shared memory approach to map memory area into process memory

Least Recently Used:
- The block that has been in the cache the longest with no reference to it is replaced
- The cache consists of a stack of blocks
- Most recently referenced block is on the top of the stack
- When a block is referenced or brought into the cache, it is placed on the top of the stack
- The block on the bottom of the stack is removed when a new block is brought in
- Blocks don’t actually move around in main memory
- A stack of pointers is used

Least Frequently Used:
- The block that has experienced the fewest references is replaced
- A counter is associated with each block
- Counter is incremented each time block accessed
- Block with smallest count is selected for replacement
- Some blocks may be referenced many times in a short period of time and then not needed any more

UNIX Buffer Cache:
- Three lists maintained to manage buffer:
  - Free list: free slots in the cache (LRU)
  - Device list: all buffers associated with each disk
  - Driver I/O queue: list of all buffers waiting for the completion of an I/O request
Slide 25

Device List
Hash Table
Buffer Cache
Free List Pointers

Slide 26

RAID
→ CPU performance has improved exponentially
→ disk performance only by a factor of 5 to 10
→ huge gap between CPU and disk performance

Parallel processing used to improve CPU performance.

Question: can parallel I/O be used to speed up and improve reliability of I/O?

Slide 27

RAID: Redundant Array of Inexpensive/Independent Disks

Multiple disks for improved performance or reliability:
→ Set of physical disks
→ Treated as a single logical drive by OS
→ Data is distributed over a number of physical disks
→ Redundancy used to recover from disk failure (exception: RAID 0)
→ There is a range of standard configurations
  - numbered 0 to 6
  - various redundancy and distribution arrangements

Slide 28

RAID 0 (striped, non-redundant):

→ controller translates single request into separate requests to single disks
→ requests can be processed in parallel
→ simple, works well for large requests
→ does not improve on reliability, no redundancy
Data mapping for RAID 0:

```
Physical Disk 0
strip 0
strip 4
strip 8
strip 12

Physical Disk 1
strip 1
strip 5
strip 9
strip 13

Physical Disk 2
strip 2
strip 6
strip 10
strip 14

Physical Disk 3
strip 3
strip 7
strip 11
strip 15
```

RAID 1 (mirrored, 2x redundancy):

```
strip 12
strip 8
strip 4
strip 0

strip 13
strip 9
strip 5
strip 1

strip 14
strip 10
strip 6
strip 2

strip 15
strip 11
strip 7
strip 3
```

RAID 2 (redundancy through Hamming code):

```
strip 15
strip 11
strip 7
strip 3
```

- strips are very small (single byte or word)
- error correction code across corresponding bit positions
- for n disks, \( \log_2 n \) redundancy
- expensive
- high data rate, but only single request

**Error Correction and Redundancy**

Just keeping two copies doesn’t necessarily help to correct the error:

Example:

```
Original: 1 0 0 1 0 1 1 1
Copy: 1 0 0 1 0 0 1 1
```

```
1 0 0 1 0 1 1 1
```

```
1 0 0 1 0 0 1 1
```

- it is not clear if the error occurred in the copy or the original
- no error correction possible
Hamming Distance between two bit-strings: The number of bits in which they differ.

→ One-bit error detection could be achieved much cheaper (parity bit)
→ How much redundancy is necessary for a one bit error correction?

Hamming Code
For every four bits of data, three parity bits
1. Parity (3,5,7)
2. Parity (3,6,7)
3. Data
4. Parity (5,6,7)
5. Data
6. Data
7. Data

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
</table>
| D | D | D |   | Parity bit 1
| D |   |   | D | Parity bit 2
| D | D |   | D | Parity bit 4
| 1 | 1 | 0 | 1 | 1 | 0 | Data |
Slide 37

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Error Correction:
- bit 111 (i.e., 7) is corrupted
- two bit errors can be detected, but not corrected

Slide 38

RAID 3 (bit-interleaved parity):
- strips are very small (single byte or word)
- simple parity bit based redundancy
- error detection
- partial error correction (if offender is known)

Slide 39

RAID 4 (block-level parity):

Slide 40

RAID 5 (block-level distributed parity):

RAID 6 (dual redundancy):

- Block 0
- Block 4
- Block 8
- Block 12
- P(12-15)
- Block 1
- Block 5
- Block 9
- Block 13
- Q(12-15)
- Block 2
- Block 6
- Block 10
- Block 14
- Block 3
- Block 7
- Block 11
- Block 15
- P(12-15)
- P(8-11)
- Q(4-7)
- P(4-7)
- Q(8-11)
- Block 16
- Block 17
- Block 18
- Block 19