Operating Systems and Concurrency

File Systems 1

COMP2007

Geert De Maere
(Dan Marsden)
{Geert.DeMaere,Dan.Marsden}@Nottingham.ac.uk

University Of Nottingham
United Kingdom

2023

Goals for Today

Overview

- Construction **rotational** and **solid state** drives
- **Access times** for hard drives
- **Disk scheduling**

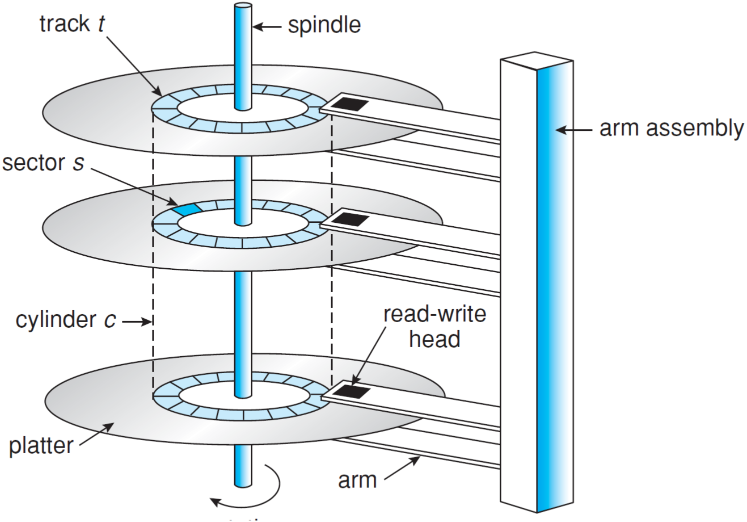
Rotational Hard Drives

Construction

- Rotational hard drives are made of aluminium/glass **platters** covered with **magnetisable material**
  - **Read/write heads** fly just above the surface (0.2 – 0.07mm) and are connected to a single **disk arm** controlled by a single **actuator**
  - **Data** is stored on both sides
  - Common **diameters** range from 1.8 to 3.5 inches
  - They **rotate** at a **constant speed** (speed near the spindle is less than on the outside)
- A **disk controller** abstracts the low level interface
- **Rotational hard drives** are approx. **4 orders of magnitude slower than main memory**

Rotational Hard Drives

Construction



Rotational Hard Drives

Low Level Format

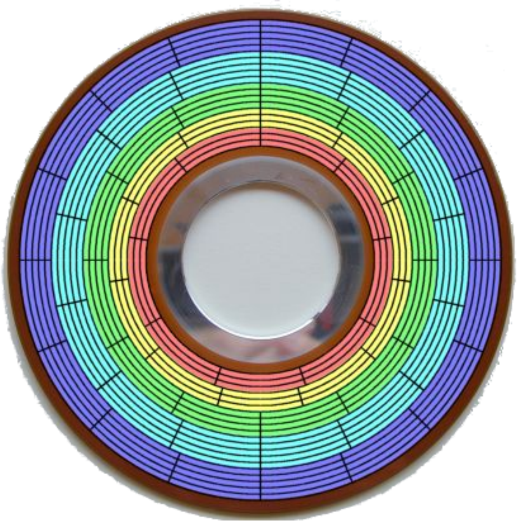
- Disks are organised in:
  - **Cylinders**: all tracks in the same position relative to the spindle
  - **Tracks**: a concentric circle on a single platter side
  - **Sectors**: segments of a track (usually 512B or 4KB in size)
- **Sectors** usually have an **equal number of bytes** in them (**preamble**, **data**, **error correcting code - ECC**)
- The **number of sectors increases** from the inner side of the disk to the outside

Preamble	Data	ECC

Figure: Disk Sector

Rotational Hard Drives

Organisation of Rotational Drives



Rotational Hard Drives

Organisation

- **Cylinder skew** is an **offset** that is added to sectors in adjacent tracks to account for the seek time
- In the past, consecutive **disk sectors were interleaved** to account for transfer time
- **Disk capacity** is reduced due to **low level formatting** (preamble, ECC, etc.)

Rotational Hard Drives

Access Times

- **Access time** = seek time + rotational delay + transfer time
  - **Seek time**: time needed to move the arm to the cylinder (dominant)
  - **Rotational latency**: time before the sector appears under the head
  - **Transfer time**: time to transfer the data

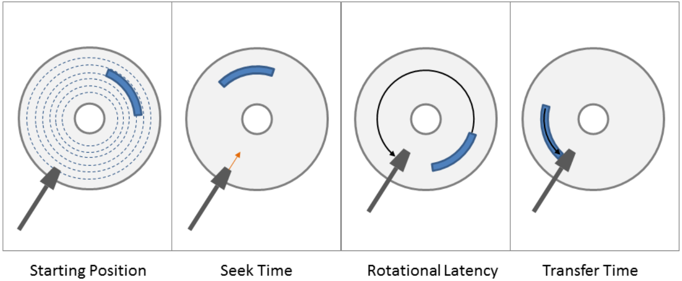
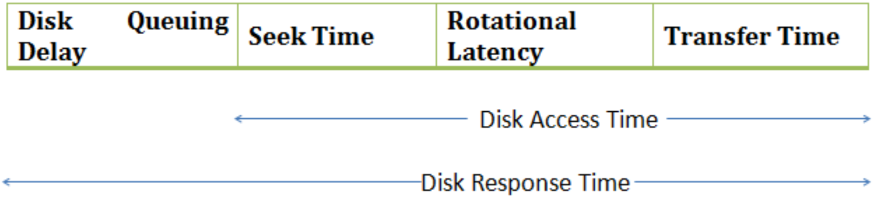


Figure: Access time to Disk (Source: www.studiodaily.com/)

Rotational Hard Drives

Access Times

- Multiple requests may be happening at the same time (concurrently). Thus, access time may be increased by a **queueing time**
- Dominance of seek time leaves room for **optimisation** by carefully considering the order of read operations



Rotational Hard Drives

Access Times

- The **estimated seek time** T_s to move the arm from one track to another is approximated by:

$$T_s = n \times m + s \tag{1}$$

- In which:
  - n the **number of tracks** to be crossed
  - m the **crossing time per track**
  - s any **additional startup delay**

Rotational Hard Drives

Access Times

- Assume a disk that **rotates at 3600rpm** (common rotation speeds are between 3600 and 15000rpm)
  - One rotation takes approx. 16.7ms ($\frac{60000}{3600}$)
  - The average **rotational latency** (T_r) is half a rotation on average ($\frac{16.7}{2} \approx 8.3ms$)
- Let b denote the **number of bytes transferred**, N the **number of bytes per track**, and rpm the **rotation speed** in revolutions per minute
- The **transfer time** T_t is given by:
  - N bytes take 1 revolution (16.7ms)
  - b contiguous bytes takes $\frac{b}{N}$ revolutions

$$T_t = \frac{b}{N} \times ms\ per\ revolution \qquad (2)$$

Rotational Hard Drives

Access Times: Example

- To read a file of **size 256 sectors** with:
  - T_s = 20 ms (average seek time)
  - 32 sectors/track
- If the file is stored **contiguously**:
  - The first track: $20 + 8.3 + 16.7 = 45ms$ (seek + rotational delay + transfer time)
  - The remaining tracks (assuming no cylinder skew and negligible seeks time between neighbouring tracks): $8.3 + 16.7 = 25ms$ (rotational delay + transfer time)
- The total time is then $45 + 7 \times 25 = 220ms$ = 0.22s

Rotational Hard Drives

Access Times: Example

- In case the access is not sequential but at **random for the sectors**, we get:
  - Time per sector = $T_s + T_r + T_t = 20 + \frac{16.7}{2} + \frac{16.7}{32} = 28.8ms$
  - Total time for 256 sectors = $256 \times 28.8ms = 7.37s$
- Observation: **sectors** must be **positioned carefully** and avoid **disk fragmentation**

Disk Scheduling

Concepts

- The OS/hardware must:
  - **Position/organise** files and sectors strategically
  - Optimise **disk requests** to **minimise ovehead** from seek time and rotational delays
- **I/O requests happen over time** and go through a **system calls** and are **queued**:
  - They are kept in a **table of requested sectors per cylinder**
  - This allows the operating system to **intercept** and **re-sequence** them

Disk Scheduling

Concepts

- **Disk scheduling algorithms** determine the order in which disk requests are processed to **minimise overhead**
  - They commonly use **heuristic approaches**
  - That is, **none** of the algorithms discussed here are **optimal algorithms**
- Assume a disk with 36 cylinders, numbered 1 to 36

Disk Scheduling

First-Come, First-Served

- **First come first served**: process the requests in the order that they arrive
- Consider the following sequence of disk requests (cylinder locations):
  `11 1 36 16 34 9 12`
- In the order of arrival (FCFS) the total length is:
  `|11-1|+|1-36|+|36-16|+|16-34|+|34-9|+|9-12|=111`

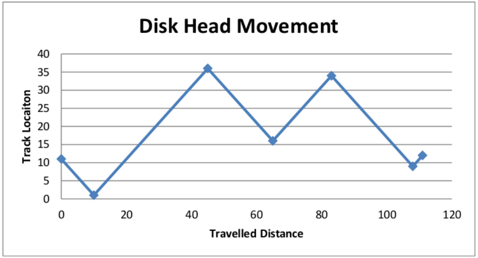


Figure: Head movement for FCFS

Disk Scheduling

Shortest Seek Time First

- **Shortest seek time first** selects the request that is **closest to the current head position**
- In the order "**shortest seek time first**" (SSTF) we gain approx. **50%**:

  `11 1 36 16 34 9 12:`

  `|11−12|+|12−9|+|9−16|+|16−1|+|1−34|+|34−36|=61`

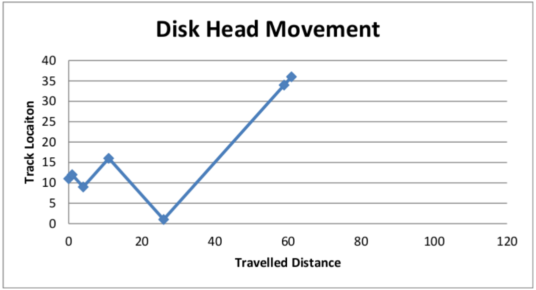


Figure: Head movement for shortest seek time

Disk Scheduling

Shortest Seek Time First

- Shortest seek time first could result in **starvation**:
  - The **arm stays in the middle of the disk** in case of heavy load (edge cylinders are poorly served)
  - Continuously arriving requests for the same location could **starve** other regions

Disk Scheduling

SCAN

- "Lift algorithm, **SCAN**": **keep moving in the same direction** until end is reached (start upwards):
  - It continues in the current direction, **servicing all pending requests** as it passes over them
  - When it gets to the **last cylinder**, it **reverses direction** and services all the pending requests (until it reaches the first cylinder)
- (Dis-)advantages include:
  - The **upper limit** on the "waiting time" is 2 $\times$ number of cylinders, i.e. **no starvation occurs**
  - The **middle cylinders are favoured** if the disk is heavily used (max. wait time is N tracks, 2N for the cylinders on the edge)

Disk Scheduling

SCAN

- "Lift algorithm, SCAN":

  11 1 36 16 34 9 12:

  |11−12|+|12−16|+|16−34|+|34−36|+|36−9|+|9−1|=60

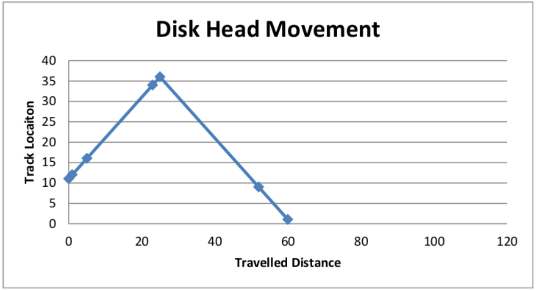


Figure: Head movement for SCAN

Disk Scheduling

Other SCAN variations: LOOK-SCAN, N-step-SCAN

- Look-SCAN moves to the cylinder containing **the first/last request** (as opposed to the first/last cylinder on the disk with SCAN)
- Seeks are **cylinder by cylinder** and one cylinder contains multiple tracks
- The arm can **stick to a cylinder**
- **N-step-SCAN** only services *N* requests every single sweep.

Disk Scheduling

C-SCAN

- Once the outer/inner side of the disk is reached, the **requests at the other end of the disk have been waiting longest**
- SCAN can be improved by using a **circular scan** approach ⇒ C-SCAN
  - The disk arm moves in **one direction** servicing requests until the **last cylinder** is reached
  - It **reverses direction** but **does not service requests** when returning
  - Once it gets back to the **first cylinder** it reverses direction and **services requests**
  - It is **fairer** and **equalises response times** across a disk
- The C-SCAN algorithm (for `11 1 36 16 34 9 12`):
  `|11-12|+|12-16|+|16-34|+|34-36|+|36-1|+|1-9|=68`

Disk Scheduling

Observations

- Look-SCAN and variations are reasonable choices for the algorithms
- Performance of the **algorithms is dependent on the requests/load of the disk**
  - One request at a time $\Rightarrow$ FCFS will perform equally well as any other algorithm
- **Optimal algorithms** are difficult to achieve if requests arrive over time (they need perfect knowledge of information)

Disk scheduling in Unix/Linux

Modifying the disk scheduler

- In Linux, we can modify the disk scheduler by modifying the file: /sys/block/sda/queue/scheduler
- We have got three policies:
  - noop: this is FCFS
  - deadline: N-step-SCAN
  - cfq: Complete Fairness Queueing from Linux.
- The one between brackets is the current policy.

```
pszgd@severn:~$ cat /sys/block/sda/queue/scheduler
noop [deadline] cfq
```

Rotational Hard Drives

Driver Caching

- For most current drives, **the time required to seek** a new cylinder is **more than the rotational time** (remember **pre-paging** in this context!)
- It makes sense, therefore, to **read more sectors than actually required**
    - **Read** sectors during the **rotational delay** (i.e. that accidentally pass by)
    - **Modern controllers read multiple sectors** when asked for the data from one sector: track-at-a-time caching

SSD drives

Architecture

- Solid State Drives (SSDs):
  - Have **no moving parts**, store data using **single level** (SLC), **multiple level** (MLC), **triple level** (TLC) electrical circuits, and suffer from **wear out** and **disturbance**
  - Are organised into **banks**, **blocks**, **pages** and have some **volatile cache memory** (buffering, mapping tables)
  - Often use multiple **banks in parallel** to improve performance

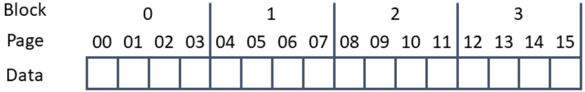


Figure: Layout of an SSD

SSD drives

Architecture

- Solid State Drives (SSDs):
  - Have **no moving parts**, store data using **single level** (SLC), **multiple level** (MLC), **triple level** (TLC) electrical circuits, and suffer from **wear out** and **disturbance**
  - Are organised into **banks**, **blocks**, **pages** and have some **volatile cache memory** (buffering, mapping tables)
  - Often use multiple **banks in parallel** to improve performance

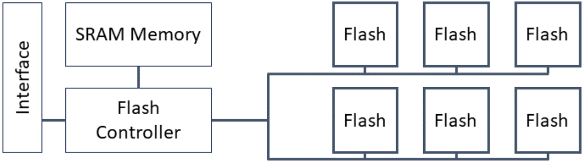


Figure: Layout of an SSD

SSD drives

Reads/Writes

- The **Flash Translation Layer** that maps **logical blocks** onto **physical pages**
- The following operations are supported:
  - Read: uniformly fast random access to any **page** to any location (10s of microseconds)
  - Erase: entire **blocks** containing multiple pages (milliseconds magnitude)
  - Program: write a **page** (100s of microseconds, block must be erased first)

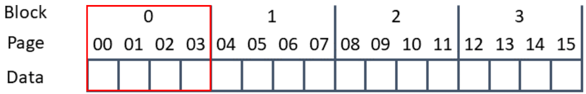


Figure: SSD Write Operation

SSD drives

Reads/Writes

- The **Flash Translation Layer** that maps **logical blocks** onto **physical pages**
- The following operations are supported:
  - Read: uniformly fast random access to any **page** to any location (10s of microseconds)
  - Erase: entire **blocks** containing multiple pages (milliseconds magnitude)
  - Program: write a **page** (100s of microseconds, block must be erased first)

Block	0			
Page	00	01	10	11
Data	00101001	10100101	11101011	00001010

Figure: SSD Write Operation

SSD drives

Reads/Writes

- The **Flash Translation Layer** that maps **logical blocks** onto **physical pages**
- The following operations are supported:
  - Read: uniformly fast random access to any **page** to any location (10s of microseconds)
  - Erase: entire **blocks** containing multiple pages (milliseconds magnitude)
  - Program: write a **page** (100s of microseconds, block must be erased first)

Block	0			
Page	00	01	10	11
Data	11111111	11111111	11111111	11111111

Figure: SSD Write Operation

SSD drives

Reads/Writes

- The **Flash Translation Layer** that maps **logical blocks** onto **physical pages**
- The following operations are supported:
  - Read: uniformly fast random access to any **page** to any location (10s of microseconds)
  - Erase: entire **blocks** containing multiple pages (milliseconds magnitude)
  - Program: write a **page** (100s of microseconds, block must be erased first)

Block	0			
Page	00	01	10	11
Data	10101010	11111111	11111111	11111111

Figure: SSD Write Operation

SSD drives

Direct Mapping

- **Logical pages** (seen by the OS) are **directly mapped** on to **physical pages**
  1. Read the entire block for the given page
  2. Erase the entire block
  3. Write the new page and remaining old pages back
- **Write performance is bad** (write amplification) and **wear is increased** (some blocks are used more than others) $\Rightarrow$ a different **log structured approach** is needed

Device	Read (μs)	Program (μs)	Erase (μs)
SLC	25	200-300	1500-2000
MLC	50	600-900	~3000
TLC	~75	~900-1350	~4500

Figure: SSD Performance (from Arpaci-Dusseau)

SSD drives

Direct Mapping

Device	Random Reads (MB/s)	Random Writes (MB/s)	Sequential Reads (MB/s)	Sequential Writes (MB/s)
Samsung 840 Pro SSD	103	287	421	384
Seagate 600 SSD	84	252	424	374
Intel SSD 335 SSD	39	222	344	354
Seagate Savvio 15K.3 HDD	2	2	223	223

Figure: SSD / HDD Comparison (from Arpaci-Dusseau)

Test Your Understanding

Problem (from Tanenbaum)

Disk Access Times

Disk requests come in to the disk driver for cylinders 10, 22, 20, 2, 40, 6 and 38, in that order.

- A seek takes 6ms per cylinder.
- How much seek time is needed for: FCFS, SSTF and Look-SCAN (initially moving upward)
- In all cases, the arm is initially at cylinder 20.

Summary

Take-Home Message

- Construction and organisation of **rotational hard drives**
- **Access times** of rotational hard drives
- **Disk scheduling** & caching
- **Solid state drives**