# Operating Systems and Concurrency

## Lecture 18: File Systems I - Disk Scheduling

G52OSC



Geert De Maere and Isaac Triguero
{Geert.DeMaere,Isaac.Triguero}@Nottingham.ac.uk

University Of Nottingham
United Kingdom



2018

# Goals for Today

## Overview[1]



- Construction of hard disks
- Accessing hard disks
- Disk scheduling



[1] Slides partially based on slides by Colin Higgins and Jon Garibaldi

# Hard Disks

## Construction of Hard Disks



- Disks are constructed as multiple 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
  - Hard disks **rotate** at a **constant speed** (i.e., speed on the inside less than on the outside $\Leftrightarrow$ CD-ROMS)
- A **disk controller** sits between the CPU and the drive
- Hard disks are currently about **4 orders of magnitude slower than main memory** $\Rightarrow$ how can we reduce the impact of this?

# Hard Disks

## Construction of Hard Disks

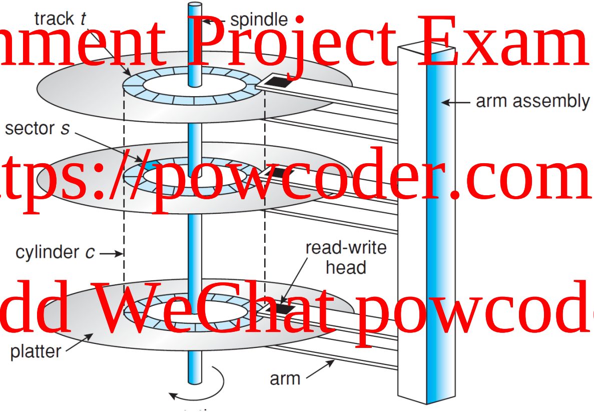





Figure: Construction of a Hard Disk (Silberschatz)

# Hard Disks
## Low Level Format



- Disks are organised in:
  - **Cylinders**: a collection of tracks in the same relative position 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, consisting of a **preamble**, **data**, and an **error correcting code**
- The **number of sectors increases** from the inner side of the disk to the outside




Figure: Disk Sector

# Hard Disks

## Organisation of Hard Disks

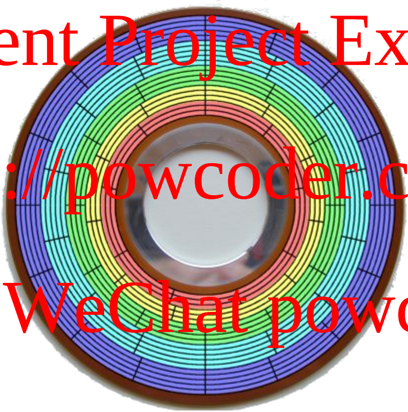




Figure: Disk Layout: Cylinder skew (Source: www.pcguide.com/)

# Hard Disks

## Organisation of Hard Disks



- Disks usually have a **cylinder skew**: i.e., an **offset** is added to sector 0 in adjecent tracks to account for the seek time



- In the past, consecutive **disk sectors were interleaved** to account for transfer time
- Note that as a result of this low-level formatting, disk capacity is reduced (size of preamble, ECC, etc)

# Hard Disks

## 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 (on average half the rotation time)
  - **Transfer time** = time to transfer the data



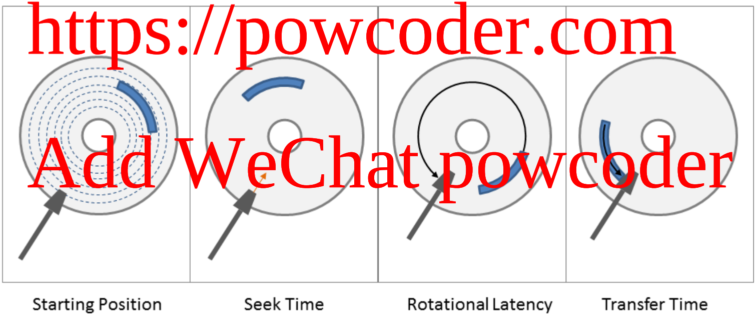
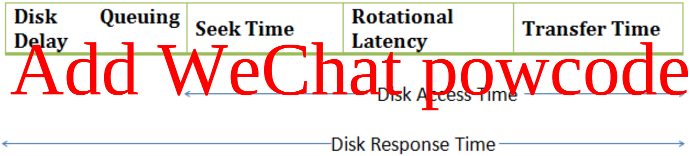



Figure: Access time to Disk (Source: www.studiodaily.com/)

# Hard Disks
## Access Times

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





# Hard Disks
## Access Times



- The **estimated seek time** (i.e., to move the arm from one track to another) is approximated by:



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

- In which $T_s$ denotes the estimated seek time, $n$ the **number of tracks** to be crossed, $m$ the **crossing time per track**, and $s$ any **additional startup delay**

# Hard Disks
## Access Times



- Let us assume a disk that **rotates at 3600 rpm** (common rotation speeds are between 3600 and 15000 rpm)
    - One rotation takes approx. 16.7ms ($\Rightarrow \frac{1}{x} = \frac{3600}{60 \times 1000} \Leftrightarrow x = \frac{60000}{3600}$)
    - The average **rotational latency** ($T_r$) is then $\frac{16.7}{2} \approx 8.3ms$



- Let $b$ denote the **number of bytes transferred**, $N$ the **number of bytes per track**, and $rpm$ the **rotation speed** in rotations per minute, the **transfer time**, $T_t$, is then given by:
    - $N$ bytes take 1 revolution $\Rightarrow \frac{60000}{3600} ms = \frac{ms\_per\_minute}{rpm}$
    - $b$ contiguous bytes takes $\frac{b}{N}$ revolutions



$$T_t = \frac{b}{N} \times \frac{ms\_per\_minute}{rpm} \tag{2}$$

# Hard Disks

## Access Times: Example



- Read a file of **size 256 sectors** with:
  - $T_s$ = 20 ms (average seek time)
  - 32 sectors/track



- Suppose the file is stored **as compact as possible - contiguous**, i.e., all sectors on 8 consecutive tracks of 32 sectors each (sequential storage)
  - The first track takes: seek + rotational delay + transfer time $\rightarrow 20 + 8.3 + 16.7 = 45ms$
  - Assuming no cylinder skew, and neglecting small seeks between tracks - We only need to account for rotational delay + transfer time: $8.3 + 16.7 = 25ms$



- The total time is then $45 + 7 \times 25 = 220ms$ = 0.22 s

# Hard Disks

## Access Times: Example



- In case the access is not sequential but at **random for the sectors**, we get:
  - Time per sector = $T_s + T_l + T_t = 20 + 8.3 + 0.5 = 28.8ms$
    $T_t = 16.7 \times \frac{1}{32} = 0.5$
  - Total time 256 sectors = $256 \times 28.8ms = 7.37s$
- It is important to **position the sectors carefully** and avoid **disk fragmentation**

# Disk Scheduling

## Concepts



- The OS must use the hardware efficiently:
  - The file system can **position/organise files strategically**
  - Having multiple disk requests in a queue allows us to **minimise the arm movement**



- Note that every I/O operation goes through a system call, allowing the **operating system to intercept the request** and **resequence it**
- If the drive (or the controller) **is free**, the request can be serviced immediately, if not, the request will be **queued**

# Disk Scheduling
## Concepts



- In a dynamic situation, several I/O requests will be **made over time** that are kept in a **table of requested sectors per cylinder**

- **Disk scheduling algorithms** determine the order in which disk events are processed
- **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



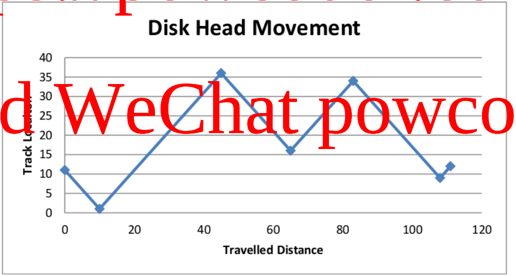





Figure: Head movement for FCFS

# Disk Scheduling
## Shortest Seek Time First

- **Shortest seek time first** selects the request that is closest to the current head position to reduce head movement
- In the order "shortest seek time first, SSTF" (cir shortest job first) **we gain approx. 50%** (for `11 1 36 16 34 9 12`): |11-12|+|12-9|+|9-16|+|16-1|+|1-34|+|34-36|=61



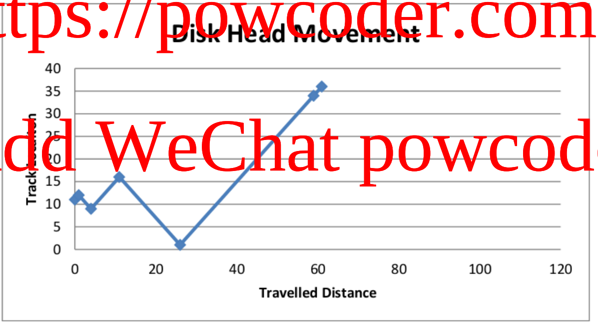



Figure: Head movement for shortest seek time

# 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, the strategy is unfair
  - 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" (for `11 1 36 16 34 9 12`):
  |11-12|+|12-16|+|16-34|+|34-36|+|36-9|+|9-1|=60



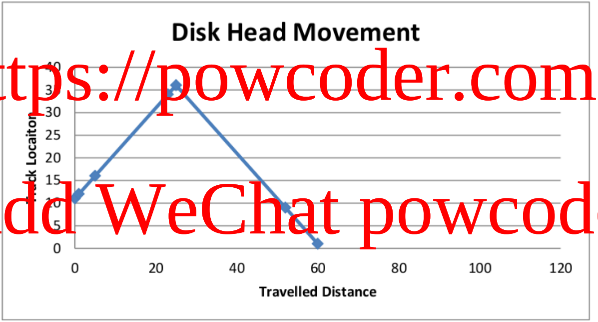





Figure: Head movement for SCAN

# Disk Scheduling
## 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 $\Rightarrow$ C-SCAN



  - The disk arm moves in one direction servicing requests
  - When it gets to the last cylinder of the disk, it **reverses direction** but it **does not service requests** on the return journey
  - Once it gets back to the first cylinder it reverses direction, and again 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

## 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)
- However, seeks are **cylinder by cylinder**, and one cylinder contains multiple tracks.
- It may happen that the arm "sticks" to a cylinder.
- N-step-SCAN only services $N$ requests every single sweep.

# Disk Scheduling
## Observations



- Look-SCAN and variations seem to be 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!

# 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.





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

# Hard Disks

## 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

# Disk scheduling SSD drives

## Do we have to do any scheduling at all?



- Solid State Drives (SSDs) have no moving parts and store data using electrical circuits.
  - They don't have $T_{seek}$ or rotational delay!
  - FCFS algorithm is useful in general purposes systems
  - SSTF, SCAN, LOOK-SCAN may reduce performance (no heads to move)

# Summary

## Take-Home Message[2]



- Construction and organisation of hard disks
- Access times of hard disks
- Disk scheduling
- Disk caching





[2] Tanenbaum Section 5.4.1 (excluding raid), 5.4.2, 5.4.3

# Disk Scheduling

## Problem (From Tanenbaum)



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.



Submit your answer at:

`https://b.socrative.com/login/student/`

**Room name: G52OSC**