Disks

OPS Lecture 12, G53OPS/G52OSC

Geert De Maere
(Jason Atkin – OSC)
Geert.DeMaere@Nottingham.ac.uk

University Of Nottingham
United Kingdom

2015

- Assume **all available pages are in active use** and a new page needs to be loaded:
  - The page that will be **evicted** will have to be **reloaded soon afterwards**, i.e., it is still active
- **Trashing** occurs when pieces are swapped out and loaded again immediately

Trashing

A Vicious Circle?

- CPU utilisation is too low $\Rightarrow$ scheduler **increases degree of multi-programming**
  - $\Rightarrow$ Pages are allocated to new processes and **taken away from existing processes**
    - $\Rightarrow$ I/O **requests are queued** up as a consequence of page faults
- CPU **utilisation drops further** $\Rightarrow$ scheduler increases degree of multi-programming

Trashing
Causes/Solutions

- **Causes** of trashing include:
  - The degree of multi-programming is too high, i.e., the total **demand** (i.e., the sum of all **working set** sizes) **exceeds supply** (i.e. the available frames)
  - An individual process is allocated **too few pages**
- This can be **prevented** by, e.g., using good **page replacement policies** and reducing the **degree of multi-programming** (medium term scheduler)

Trashing
Causes/Solutions

- **Causes** of trashing include:
    - The degree of multi-programming is too high, i.e., the total **demand** (i.e., the sum of all **working set** sizes) **exceeds supply** (i.e. the available frames)
    - An individual process is allocated **too few pages**
- This can be **prevented** by, e.g., using good **page replacement policies** and reducing the **degree of multi-programming** (medium term scheduler)
- Solve the **root cause** $\Rightarrow$ add **more memory**

Content
Remember The Course Structure

Subject	#Lectures
Introduction to operating systems/computer design	1-2
Processes, process scheduling, threading, ...	3-4
Memory management, swapping, virtual memory, ...	3-4
File Systems, file structures, management, ...	3-4
Case study: Linux	2
Revision	2

Table: Preliminary course structure

Goals for Today

Overview[1]

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

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

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 ⇔ CD-ROMS)
- A **disk controller** sits between the CPU and the drive
- Disks are currently about **4 orders of magnitude slower than main memory** ⇒ how can we reduce the impact of this?

Disks

Construction of Hard Disks

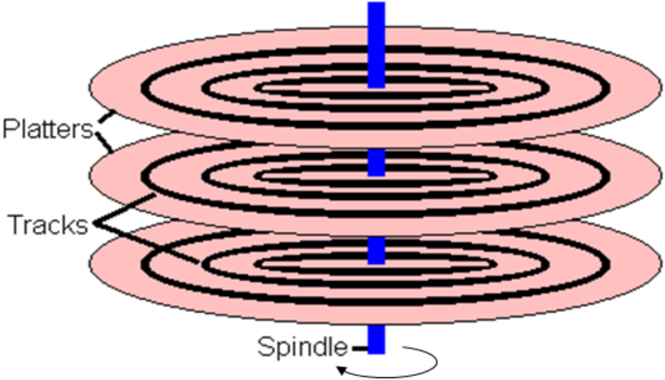

Figure: Construction of a Hard Disk

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 4k in size)
- **Sectors** usually have an **equal number of bytes** in them, consist 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

Disks
Organisation of Hard Disks

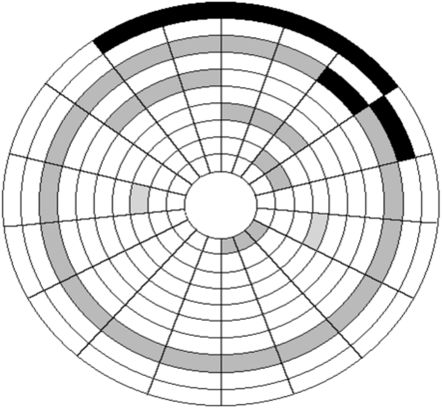

Figure: Disk Layout

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
- Consecutive **disks sectors may be interleaved** to account for transfer time

Disks
Access Times

- **Access time** = seek time + rotational delay + transfer time
  - **Seek time** = time needed to move the arm to the cylinder (dominant)
  - **Rotational delay** = time before the sector appears under the head (on average half the rotation time)
  - **Transfer time** = time to transfer the data
- Dominance of seek time leaves room for **optimisation** by carefully considering the order of read operations
- Note that the access time may be increased by **queueing time** when multiple concurrent requests are queued

Disks
Access Times

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

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

- 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**

Disks
Access Times

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

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

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 $20 + 8.3 + 16.7 = 45ms$ (seek + rotational delay + transfer time)
  - The remaining tracks do not need seek time, hence, per track we need $8.3 + 16.7 = 25ms$ (assuming no cylinder skew, neglect small seeks)
- The total time is then $45 + 7 \times 25 = 220ms$ = 0.22 s

Disks
Access Times: Example

- In case the access is not sequential but at **random for the sectors**, we get:
  - Time per sector = $20 + 8.3 + 0.5 (= \frac{16.7}{32}) = 28.8ms$
  - Total time 256 sectors = $256 \times 28.8 = 7.37s$
- 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**
  - Heavy loaded disk queue allows to **minimize the arm movement**
- If the drive (or the controller) is free, the request can be serviced immediately, if not, the request will be **queued**
- Note that every I/O operation goes through a system call, allowing the **operating system to intercept the request** and **resequence it**

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

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

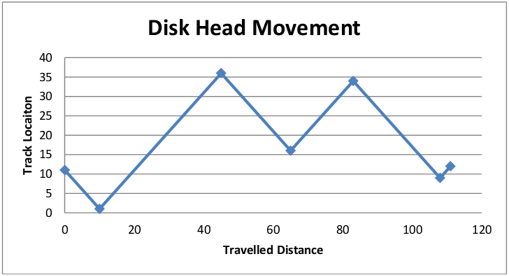

Figure: Head movement for FCFS

Disk Scheduling
Shortest Seek Time First

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

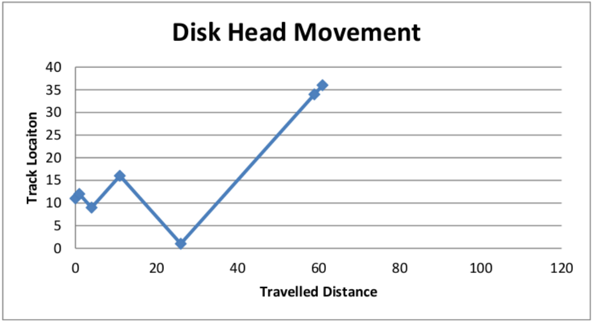

Figure: Head movement for shortest seek time

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
- Shortest seek time first is **not an optimal algorithm**

Disk Scheduling
SCAN

- "Lift algorithm, **SCAN**" : **keep moving in the same direction** until end is reach (start upwards):
  - It continues in the current direction, **servicing all pending requests** as it passes over them
  - When it gets to the **last pending request** in the direction it is going, it **reverses direction** and services all the pending requests
- (Dis-)advantages include:
  - The **upper limit** on "wait time" 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” :
  |11-12|+|12-16|+|16-34|+|34-36|+|36-9|+|9-1|=60

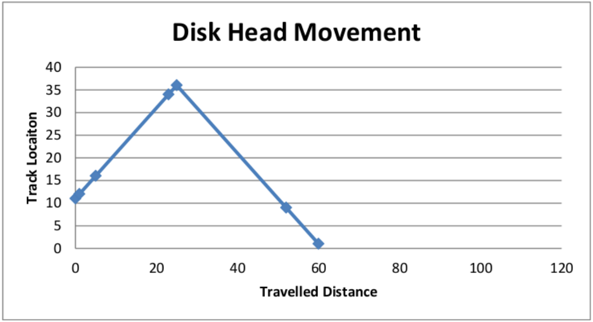

Figure: Head movement for SCAN

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 end of the disk, it **reverses direction** but it **does not service requests** on the return journey
  - Once it gets back to the beginning it reverses direction, and again services requests
  - It is **fairer** and **equalises response times** across a disk
- **Smaller variance** is obtained by moving the arm in one direction, always returning to the lowest number at the end of the road:
- The C-SCAN algorithm:
  |11-12|+|12-16|+|16-34|+|34-36|+|36-1|+|1-9|=68

Disk Scheduling

LOOK-SCAN, N-step-SCAN and F-SCAN

- Look SCAN moves to the **location of the first/last request**
- Seeks are **cylinder by cylinder** (one cylinder contains multiple tracks)
- It may happen that the arms sticks to a cylinder:
  - N-step-SCAN: **divide the queue** into segments of N requests which are handled using SCAN
  - F-SCAN: **use two queues**, while one is handled with SCAN, the other one is filled (swapped over when all are processed)

Disk Scheduling
Observations

- LOOK and SSTF are reasonable choices for the algorithms
- Performance of the **algorithms is dependent on the requests/load of the disk**
  - One outstanding request at a time $\Rightarrow$ FCFS will perform equally well as any other algorithm
- **Optimal algorithms** are difficult to achieve!

Disk Scheduling
Observations

- The **layout of the file system** and the **file allocation method** used by the operating system **heavily influences the seek movements**
  - Contiguous files will result in many short head movements
- Whilst disk scheduling could be **implemented in the controller**, carrying it out in the OS means that the **OS can prioritise/order requests** as appropriate

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** (that accidentally pass by)
  - **Modern controllers read the multiple sectors** when asked for the data from one sector: track-at-a-time caching

Summary
Take-Home Message

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