
CS 422/522 Design & Implementation
of Operating Systems

Lecture 15: Storage Devices

Zhong Shao
Dept. of Computer Science
Yale University

1

The big picture

- ◆ Lectures before the fall break:
 - Management of CPU & concurrency
 - Management of main memory & virtual memory
- ◆ Current topics --- "Management of I/O devices"
 - Last lecture: I/O devices & device drivers
 - This lecture: **storage devices**
 - Next week: file systems
 - * File system structure
 - * Naming and directories
 - * Efficiency and performance
 - * Reliability and protection

2

Main points

- ◆ File systems
 - Useful abstractions on top of physical devices
- ◆ Storage hardware characteristics
 - Disks and flash memory
- ◆ File system usage patterns

3

File systems

- ◆ Abstraction on top of persistent storage
 - Magnetic disk
 - Flash memory (e.g., USB thumb drive)
- ◆ Devices provide
 - Storage that (usually) survives across machine crashes
 - Block level (random) access
 - Large capacity at low cost
 - Relatively slow performance
 - * Magnetic disk read takes 10-20M processor instructions

4

File system as illusionist: hide limitations of physical storage

- ◆ Persistence of data stored in file system:
 - Even if crash happens during an update
 - Even if disk block becomes corrupted
 - Even if flash memory wears out
- ◆ Naming:
 - Named data instead of disk block numbers
 - Directories instead of flat storage
 - Byte addressable data even though devices are block-oriented
- ◆ Performance:
 - Cached data
 - Data placement and data structure organization
- ◆ Controlled access to shared data

5

File system abstraction

- ◆ File system
 - Persistent, named data
 - Hierarchical organization (directories, subdirectories)
 - Access control on data
- ◆ File: named collection of data
 - Linear sequence of bytes (or a set of sequences)
 - Read/write or memory mapped
- ◆ Crash and storage error tolerance
 - Operating system crashes (and disk errors) leave file system in a valid state
- ◆ Performance
 - Achieve close to the hardware limit in the average case

6

Storage devices

◆ Magnetic disks

- Storage that rarely becomes corrupted
- Large capacity at low cost
- Block level random access
- Slow performance for random access
- Better performance for streaming access

◆ Flash memory

- Storage that rarely becomes corrupted
- Capacity at intermediate cost (50x disk)
- Block level random access
- Good performance for reads; worse for random writes

7

A typical disk controller

◆ External connection

- IDE / ATA, SATA
- SCSI, SCSI-2, Ultra SCSI, Ultra-160 SCSI, Ultra-320 SCSI
- Fibre channel (FC)

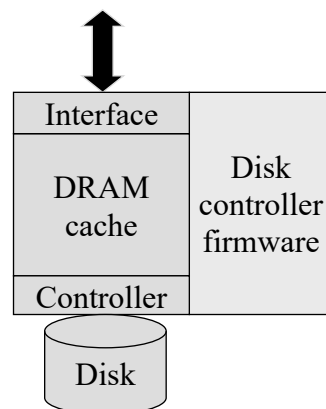
◆ Cache

- Buffer data between disk and the I/O bus

◆ Controller

- Details of read/write
- Cache replacement algorithm
- Failure detection and recovery

External connection



8

Caching inside a disk controller

◆ Method

- Disk controller has DRAM to cache recently accessed blocks
 - * Hitachi disk has 16MB
 - * Some of the RAM space stores “firmware” (an embedded OS)
- Blocks are replaced usually in an LRU order

◆ Pros

- Good for reads if accesses have locality

◆ Cons

- Expensive
- Need to deal with reliable writes

9

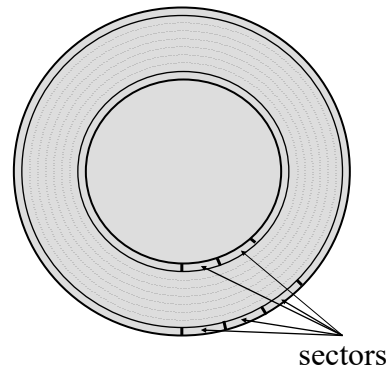
Magnetic disk



10

Disk organization

- ◆ Disk surface
 - Circular disk coated with magnetic material
- ◆ Tracks
 - Concentric rings around disk surface, bits laid out serially along each track
- ◆ Sectors
 - Each track is split into arc of track (min unit of transfer)



11

Disk tracks

- ◆ ~ 1 micron wide
 - Wavelength of light is ~ 0.5 micron
 - Resolution of human eye: 50 microns
 - 100K tracks on a typical 2.5" disk
- ◆ Separated by unused guard regions
 - Reduces likelihood neighboring tracks are corrupted during writes (still a small non-zero chance)
- ◆ Track length varies across disk
 - Outside: More sectors per track, higher bandwidth
 - Disk is organized into regions of tracks with same # of sectors/track
 - Only outer half of radius is used
 - * Most of the disk area in the outer regions of the disk

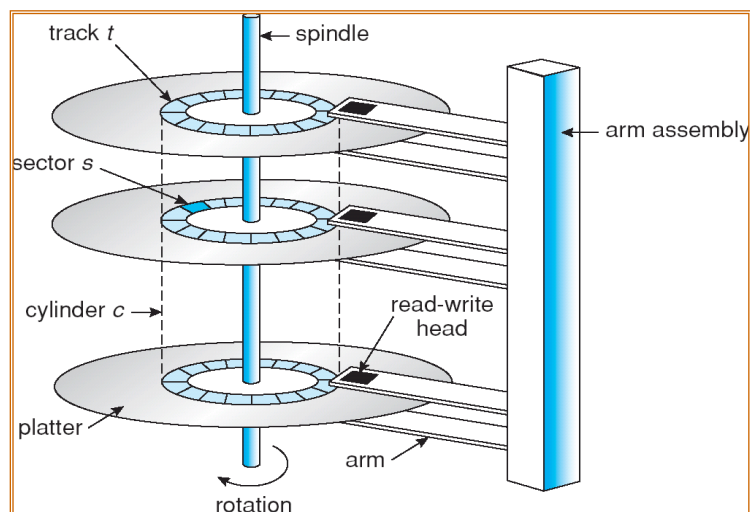
12

Sectors

- ◆ Sectors contain sophisticated error correcting codes
 - Disk head magnet has a field wider than track
 - Hide corruptions due to neighboring track writes
- ◆ Sector sparing
 - Remap bad sectors transparently to spare sectors on the same surface
- ◆ Slip sparing
 - Remap all sectors (when there is a bad sector) to preserve sequential behavior
- ◆ Track skewing
 - Sector numbers offset from one track to the next, to allow for disk head movement for sequential ops

13

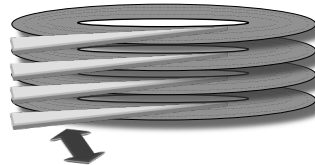
Moving-head disk mechanism



14

Disk cylinder and arm

- ◆ CD's and floppies come individually, but magnetic disks come organized in a disk pack
- ◆ Cylinder
 - Certain track of the platter
- ◆ Disk arm
 - A disk arm carries disk heads
- ◆ Read/write operation
 - Disk controller receives a command with <track#, sector#>
 - Seek the right cylinder (tracks)
 - Wait until the right sector comes
 - Perform read/write



seek a cylinder

15

Disk performance

Disk Latency =

Seek Time + Rotation Time + Transfer Time

Seek Time: time to move disk arm over track (1-20ms)

Fine-grained position adjustment necessary for head to "settle"

Head switch time ~ track switch time (on modern disks)

Rotation Time: time to wait for disk to rotate under disk head

Disk rotation: 4 - 15ms (depending on price of disk)

On average, only need to wait half a rotation

Transfer Time: time to transfer data onto/off of disk

Disk head transfer rate: 50-100MB/s (5-10 usec/sector)

Host transfer rate dependent on I/O connector (USB, SATA, ...)

16

Toshiba disk (2008)

Size	
Platters/Heads	2/4
Capacity	320 GB
Performance	
Spindle speed	7200 RPM
Average seek time read/write	10.5 ms/ 12.0 ms
Maximum seek time	19 ms
Track-to-track seek time	1 ms
Transfer rate (surface to buffer)	54–128 MB/s
Transfer rate (buffer to host)	375 MB/s
Buffer memory	16 MB
Power	
Typical	16.35 W
Idle	11.68 W

17

Question

- ♦ How long to complete 500 random disk reads, in FIFO order?

18

Question

- ◆ How long to complete 500 random disk reads, in FIFO order?
 - Seek: average 10.5 msec
 - Rotation: average 4.15 msec
 - Transfer: 5-10 usec
- ◆ $500 * (10.5 + 4.15 + 0.01)/1000 = 7.3 \text{ seconds}$

19

Question

- ◆ How long to complete 500 sequential disk reads?

20

Question

◆ How long to complete 500 sequential disk reads?

- Seek Time: 10.5 ms (to reach first sector)
- Rotation Time: 4.15 ms (to reach first sector)
- Transfer Time: (outer track)

$$500 \text{ sectors} * 512 \text{ bytes} / 128 \text{ MB/sec} = 2 \text{ ms}$$

Total: $10.5 + 4.15 + 2 = 16.7 \text{ ms}$

Might need an extra head or track switch (+1ms)

Track buffer may allow some sectors to be read off disk out of order (-2ms)

21

Question

◆ How large a transfer is needed to achieve 80% of the max disk transfer rate?

22

Question

- ◆ How large a transfer is needed to achieve 80% of the max disk transfer rate?

Assume x rotations are needed, then solve for x :

$$0.8 (10.5 \text{ ms} + (1\text{ms} + 8.5\text{ms}) x) = 8.5\text{ms } x$$

Total: $x = 9.1$ rotations, 9.8MB

23

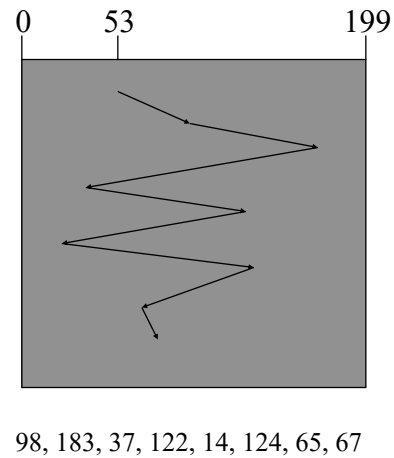
Disk scheduling

- ◆ FIFO
 - Schedule disk operations in order they arrive
 - Downsides?

24

FIFO (FCFS) order

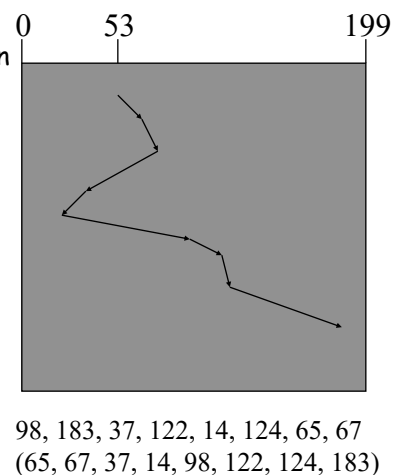
- ◆ Method
 - First come first serve
- ◆ Pros
 - Fairness among requests
 - In the order applications expect
- ◆ Cons
 - Arrival may be on random spots on the disk (long seeks)
 - Wild swing can happen



25

SSTF (Shortest Seek Time First)

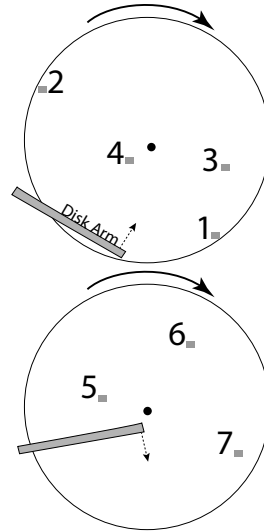
- ◆ Method
 - Pick the one closest on disk
 - Rotational delay is in calculation
- ◆ Pros
 - Try to minimize seek time
- ◆ Cons
 - Starvation
- ◆ Question
 - Is SSTF optimal?
 - Can we avoid the starvation?



26

Disk scheduling

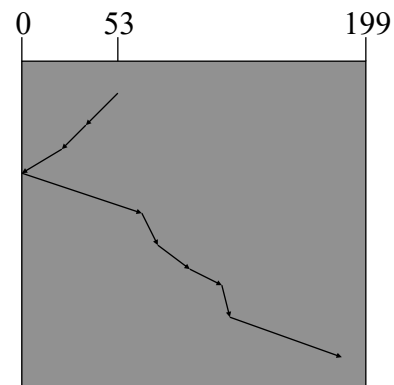
- ◆ **SCAN:** move disk arm in one direction, until all requests satisfied, then reverse direction
- ◆ Also called "elevator scheduling"



27

Elevator (SCAN)

- ◆ **Method**
 - Take the closest request in the direction of travel
 - Real implementations do not go to the end (called LOOK)
- ◆ **Pros**
 - Bounded time for each request
- ◆ **Cons**
 - Request at the other end will take a while

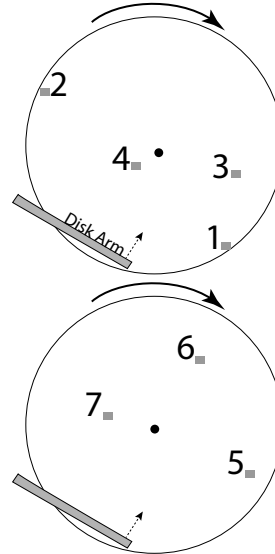


98, 183, 37, 122, 14, 124, 65, 67
 (37, 14, 65, 67, 98, 122, 124, 183)

28

Disk scheduling

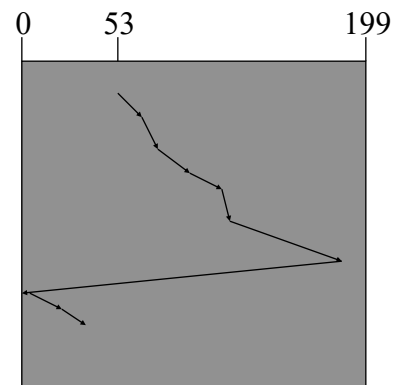
- ◆ **CSCAN:** move disk arm in one direction, until all requests satisfied, then start again from farthest request



29

C-SCAN (Circular SCAN)

- ◆ **Method**
 - Like SCAN
 - But, wrap around
 - Real implementation doesn't go to the end (**C-LOOK**)
- ◆ **Pros**
 - Uniform service time
- ◆ **Cons**
 - Do nothing on the return

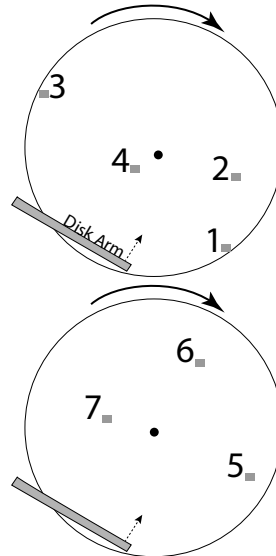


98, 183, 37, 122, 14, 124, 65, 67
 (65, 67, 98, 122, 124, 183, 14, 37)

30

Disk scheduling

- ◆ R-CSCAN: *CSCAN* but take into account that short track switch is $<$ rotational delay



31

Question

- ◆ How long to complete 500 random disk reads, in any order?

32

Question

- ◆ How long to complete 500 random disk reads, in any order?
 - Disk seek: 1ms (most will be short)
 - Rotation: 4.15ms
 - Transfer: 5-10usec
- ◆ Total: $500 * (1 + 4.15 + 0.01) = 2.2 \text{ seconds}$
 - Would be a bit shorter with R-CSCAN
 - vs. 7.3 seconds if FIFO order

33

Question

- ◆ How long to read all of the bytes off of a disk?

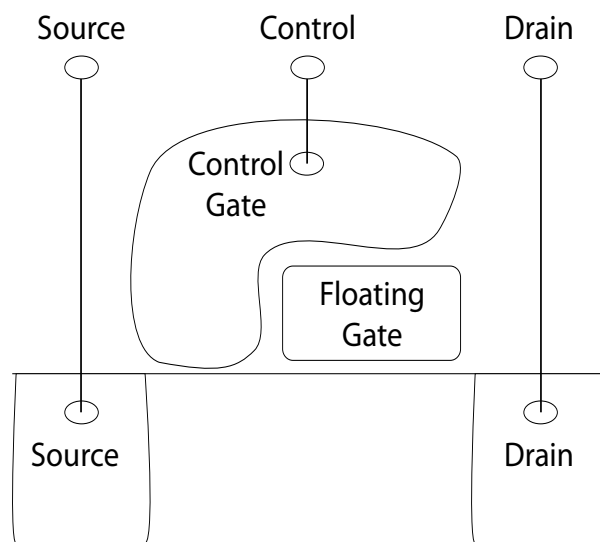
34

Question

- ◆ How long to read all of the bytes off of a disk?
 - Disk capacity: 320GB
 - Disk bandwidth: 54-128MB/s
- ◆ Transfer time =
Disk capacity / average disk bandwidth
~ 3500 seconds (1 hour)

35

Flash memory



36

Flash memory

- ◆ Writes must be to "clean" cells; no update in place
 - Large block erasure required before write
 - Erasure block: 128 - 512 KB
 - Erasure time: Several milliseconds
- ◆ Write/read page (2-4KB)
 - 50-100 usec

37

Flash drive (2011)

Size	
Capacity	300 GB
Page Size	4KB
Performance	
Bandwidth (Sequential Reads)	270 MB/s
Bandwidth (Sequential Writes)	210 MB/s
Read/Write Latency	75 μ s
Random Reads Per Second	38,500
Random Writes Per Second	2,000 (2,400 with 20% space reserve)
Interface	SATA 3 Gb/s
Endurance	
Endurance	1.1 PB (1.5 PB with 20% space reserve)
Power	
Power Consumption Active/Idle	3.7 W / 0.7 W

38

Question

- ◆ Why are random writes so slow?
 - Random write: 2000/sec
 - Random read: 38500/sec

39

Flash translation layer

- ◆ Flash device firmware maps logical page # to a physical location
 - Garbage collect erasure block by copying live pages to new location, then erase
 - * More efficient if blocks stored at same time are deleted at same time (e.g., keep blocks of a file together)
 - Wear-leveling: only write each physical page a limited number of times
 - Remap pages that no longer work (sector sparing)
- ◆ Transparent to the device user

40

File system - flash

- ◆ How does Flash device know which blocks are live?
 - Live blocks must be remapped to a new location during erasure
- ◆ TRIM command
 - File system tells device when blocks are no longer in use

41

File system workload

- ◆ File sizes
 - Are most files small or large?
 - Which accounts for more total storage: small or large files?

42

File system workload

◆ File sizes

- Are most files small or large?
 - * SMALL
- Which accounts for more total storage: small or large files?
 - * LARGE

43

File system workload

◆ File access

- Are most accesses to small or large files?
- Which accounts for more total I/O bytes: small or large files?

44

File system workload

◆ File access

- Are most accesses to small or large files?
 - * SMALL
- Which accounts for more total I/O bytes: small or large files?
 - * LARGE

45

File system workload

◆ How are files used?

- Most files are read/written sequentially
- Some files are read/written randomly
 - * Ex: database files, swap files
- Some files have a pre-defined size at creation
- Some files start small and grow over time
 - * Ex: program stdout, system logs

46

File system design

- ◆ For small files:
 - Small blocks for storage efficiency
 - Concurrent ops more efficient than sequential
 - Files used together should be stored together
- ◆ For large files:
 - Storage efficient (large blocks)
 - Contiguous allocation for sequential access
 - Efficient lookup for random access
- ◆ May not know at file creation
 - Whether file will become small or large
 - Whether file is persistent or temporary
 - Whether file will be used sequentially or randomly

47

File system abstraction

- ◆ Directory
 - Group of named files or subdirectories
 - Mapping from file name to file metadata location
- ◆ Path
 - String that uniquely identifies file or directory
 - Ex: /cse/www/education/courses/cse451/12au
- ◆ Links
 - Hard link: link from name to metadata location
 - Soft link: link from name to alternate name
- ◆ Mount
 - Mapping from name in one file system to root of another

48

UNIX file system API

- ◆ create, link, unlink, createdir, rmdir
 - Create file, link to file, remove link
 - Create directory, remove directory
- ◆ open, close, read, write, seek
 - Open/close a file for reading/writing
 - Seek resets current position
- ◆ fsync
 - File modifications can be cached
 - fsync forces modifications to disk (like a memory barrier)

49

File system interface

- ◆ UNIX file open is a Swiss Army knife:
 - Open the file, return file descriptor
 - Options:
 - * if file doesn't exist, return an error
 - * If file doesn't exist, create file and open it
 - * If file does exist, return an error
 - * If file does exist, open file
 - * If file exists but isn't empty, nix it then open
 - * If file exists but isn't empty, return an error
 - * ...

50

Interface design question

- ◆ Why not separate syscalls for open/create/exists?
 - Would be more modular!

```
if (!exists(name))  
    create(name); // can create fail?  
fd = open(name); // does the file exist?
```