CSCE 451/851
Operating Systems Principles

Secondary Storage Management

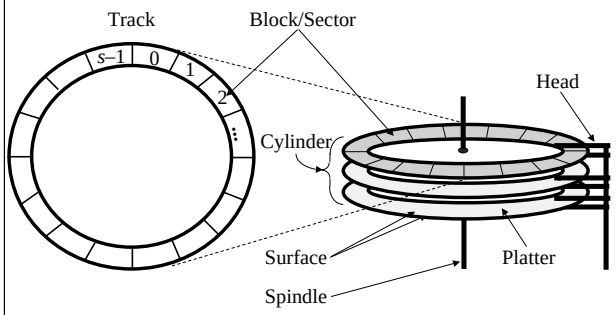
Steve Goddard
goddard@cse.unl.edu

http://www.cse.unl.edu/~goddard/Courses/CSCE451

Secondary Storage Management
Disks — just like memory, only different

- Why have disks?
  - » We'll never have enough memory
    - memory — 341 KB/dollar or $3/MB
    - disks — 33 MB/dollar or $0.03/MB
  - ⇨ disks as short term storage
    - swap space for virtual memory system
  - » Memory is volatile
  - ⇨ disks as long term storage
    - files

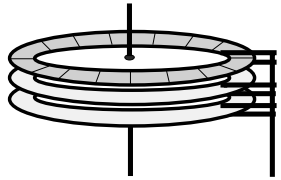
Anatomy of a Disk
Basic components



3

Anatomy of a Disk
Example: Seagate 9GB Fast/Wide/Differential SCSI disk

- Specs:
  - » 12 platters
  - » 22 heads
  - » variable # of sectors/track
  - » 7,200 RPM
    - ❖ average latency: 4.2 ms.
  - » Seek times
    - ❖ track-to-track: 1 ms
    - ❖ average: 7.9 ms
  - » 40MB/s peak transfer rate
  - » 11 arms
  - » 4,735 tracks
  - » 512 bytes/sector



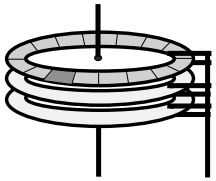
4

Disk Operations

Data transfer in units of sectors

Random access devices with non-uniform access times

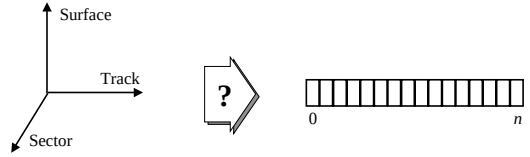
- Present disk with a sector address
  - » *DA* = (*drive, surface, track, sector*)
- Head moved to appropriate track
  - » "seek time"
- The appropriate head is enabled
- Wait for the sector to appear under the head
  - » "rotational latency"
- Read/write the sector
  - » "transfer time"



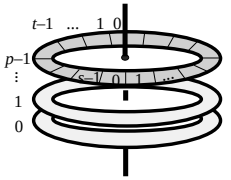
5

Disk Addressing

Mapping a 3-D structure to a 1-D structure



- Mapping criteria
  - » block *n*+1 should be a close as possible to block *n*



6

Disk Addressing

The impact of a good/bad file mapping

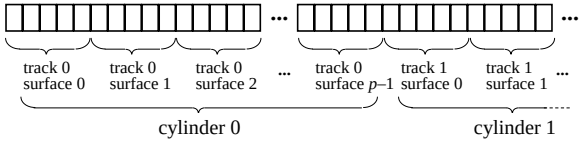
- Consider the time required to read/write a 1 MB file stored in contiguous locations in the OS file block array
  - » File requires 2,048 sectors
- Array elements map to contiguous sectors on disk
  - » Middle of the disk
    - ❖ $7.9 + 4.2 + 8.3\frac{2,048}{3,712} = 12.1 + 4.6 = 16.7$ ms

      Transfer time = (time per rev) × (# of revs required)
  - » Outside of disk
    - ❖ $7.9 + 4.2 + 8.3\frac{2,048}{14,848} = 12.1 + 1.1 = 13.3$ ms
  - » Center of disk
    - ❖ $7.9 + 4.2 + 8.3\frac{2,048}{928} = 12.1 + 18.3 = 30.4$ ms
- Array elements map to random sectors on disk
  - » 2,048× (7.9 + 4.2) = 24.8 secs!

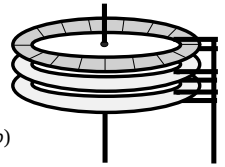
7

Disk Addressing

Cylinder-based mappings



- Sector address to block list mapping
  - » *s* sectors
  - » *p* platters



(*surface j*, *track i*, *sector k*) = $k + s(j + ip)$

8

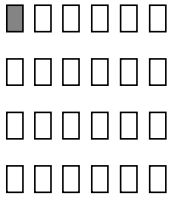
Disk Space Management
Device Directory

- A block(s) on disk containing...
  - » data structures storing names, locations, lengths, owner, *etc*. of all files on disk
    - ❖ a symbol table
  - » data structures storing free block list
- Stored at a fixed location on disk
- Directory operations
  - » search (find a file)
    - ❖ linear search
    - ❖ binary search
    - ❖ hash table
  - » Create a file
  - » Delete a file
  - » List contents of a directory
  - » Backup

9

Device Directory
Allocated space representations

- Single-level directory
  - » easy to build
  - » *all* file names must be unique
- Two-level directory
  - » a separate directory for each user/project/group
- Trees
- Acyclic graphs
  - » problems with...
    - ❖ aliases
    - ❖ file deletion
    - ❖ backups



10

Device Directory
Free list representation

- Bit vector: 1111111111111110011101010111011111...
  - » If bit *i* = 0 then block *i* is *free*, if *i* = 1 then it is *allocated*
  - » Simple to use but this can be a big vector:
    - 17.5 million elements for a 9 GB disk (2.2 MB worth of bits)
  - » However, if free sectors are uniformly distributed across the disk then the expected number of bits that must be scanned before finding a "0" is

$$n/r$$

where

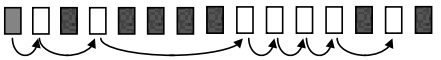
n = total number of blocks on the disk,
r = number of free blocks

- If a disk is 90% full, then the average number of bits to be scanned is 10, independent of the size of the disk

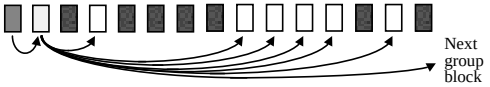
11

Device Directory
Other free list representation schemes

- In-situ linked lists



- Grouped lists



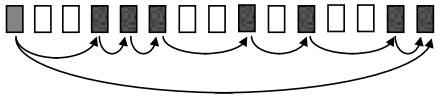
12

File Allocation Methods
Contiguous allocation

- Directory entry specifies starting block & length
- Placement
  - » First-fit, best-fit, ...
- Pluses
  - » Best file read/write performance
  - » Efficient sequential & random access
- Minuses
  - » Fragmentation!
  - » Problems with file growth
    - ❖ Pre-allocation?
    - ❖ On-demand allocation?

13

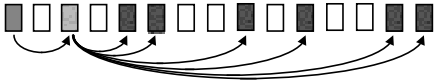
File Allocation Methods
Linked allocation



- Files stored as a linked list of blocks
- Directory entry is a pointer to the first & last file blocks
- Pluses
  - » Easy to create, grow & shrink files
  - » No fragmentation
- Minuses
  - » Impossible to do true random access
  - » Reliability
    - ❖ Break one link in the chain and...

14

File Allocation Methods
Indexed allocation



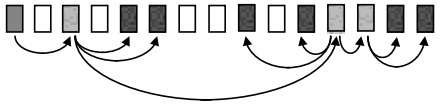
- Create a non-data block for each file called the *index block*
  - » A list of pointers to file blocks
- Directory entry is a pointer to the index block

- Pluses
  - » Easy to create, grow & shrink files
  - » No fragmentation
  - » Supports direct access
- Minuses
  - » Overhead of storing index when files are small
  - » How to handle large files?

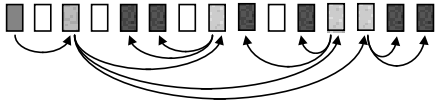
15

Indexed Allocation
Handling large files

- Linked index blocks



- Multilevel index blocks

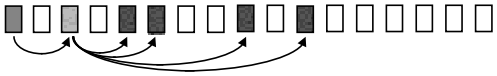


16

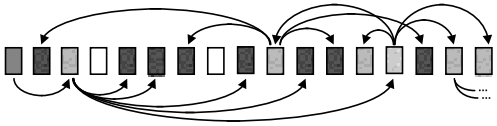
Indexed Allocation

Indexed allocation in UNIX

- Small files
  - » A single index block

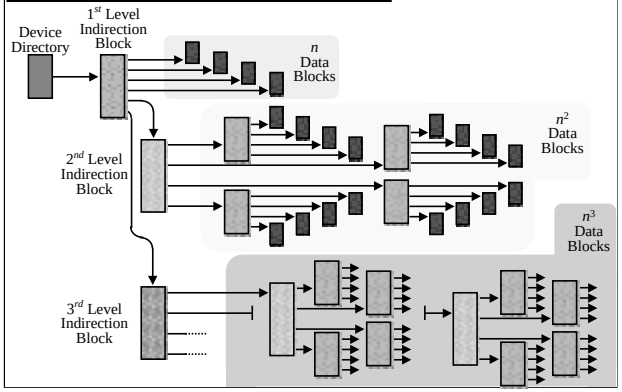


- Medium size files
  - » An indirection block



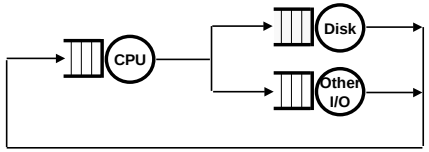
Indexed Allocation in UNIX

Multilevel, indirection, index blocks



Disk Head Scheduling
Maximizing disk throughput

- In a multiprogramming/timesharing environment, disk I/O requests are queued up

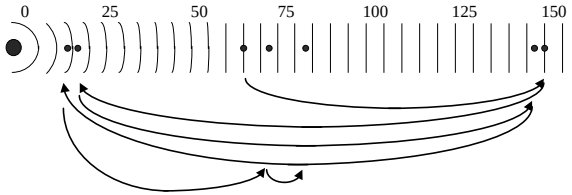


- The OS maximizes disk I/O throughput by minimizing head movement through *disk head scheduling*

19

Disk Head Scheduling
Examples

- Assume a queue of requests exists to read/write tracks:
  - » 150 16 147 14 72 83 and the head is on track 65

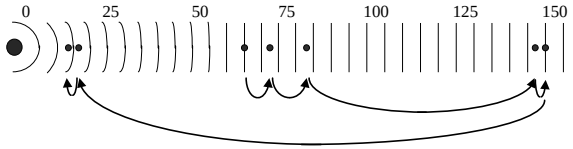


- FCFS scheduling results in the head moving 550 tracks
  - » Can we do better?

20

Disk Head Scheduling
Minimizing head movement

- Greedy scheduling: *shortest seek time first*
  - » Rearrange queue from: 150 16 147 14 72 83
    To: 72 83 147 150 16 14

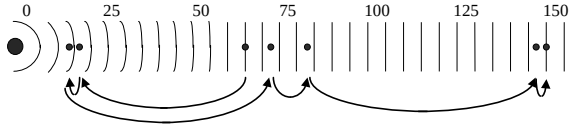


- SSTF results in the head moving 221 tracks
  - » Can we do better?

21

Disk Head Scheduling
Optimal scheduling

- Rearrange queue from: 150 16 147 14 72 83
  To: 16 14 72 83 147 150

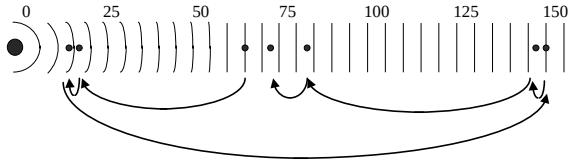


- SCAN scheduling
  - » Move the head in one direction until all requests have been serviced and then reverse
  - » Results in the head moving 187 tracks

22

Disk Head Scheduling
Other variations

- C-SCAN scheduling ("Circular"-SCAN)
  - » Move the head in one direction *until an edge of the disk is reached* and then *reset to the opposite edge*

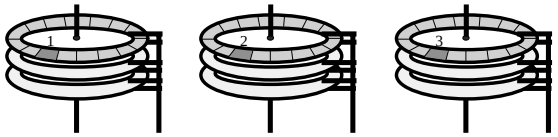


- LOOK scheduling
  - » C-SCAN except the head is reset when no more requests exist between the current head position and the approaching edge of the disk

23

Speeding Up Disk I/O
Disk architectures

- Disk striping
  - » Blocks broken into sub-blocks that are stored on separate disks
    - ❖ similar to memory inter-leaving
  - » Provides for higher disk bandwidth through a larger effective block size

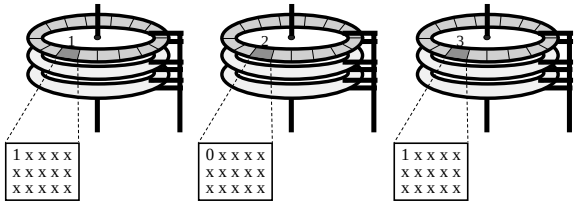


24

Speeding Up Disk I/O

Disk architectures

- RAID (*redundant array of inexpensive disks*) disks
  - » Bit-wise striping of the disks (RAID-3) or
  - » Block-wise striping of the disks (RAID-5)
  - » Provides better performance & reliability
- Example: storing the bit-string 101



25

RAID Disks

Improving reliability & availability

- Block interleaved parity striping
  - » Allows one to recover from the crash of any one disk
  - » Example: storing 8, 9, 10, 11, 12, 13, 14, 15, 0, 1, 2, 3

Layout on a non-RAID disk:

Block 1	Block 2	Block 3	Block 4
1 0 0 0 1 0 0 1 1 0 1 0	1 0 1 1 1 1 0 0 1 1 0 1	1 1 1 0 1 1 1 1 0 0 0 0	0 0 0 1 0 0 1 0 0 0 1 1

RAID layout:

Block 1 Disk 1	Block 1 Disk 2	Block 1 Disk 3	Block 1 Disk 4	Block 1 Disk 5
1 1 1 1 1 1 1 1 0 0 0 0	0 0 0 0 1 1 1 1 0 0 0 0	0 0 1 1 0 0 1 1 0 0 1 1	0 1 0 1 0 1 0 1 0 1 0 1	1 0 0 1 0 1 1 0 0 1 1 0

26