UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 537
Intro to Operating Systems

A. Arpaci-Dusseau
Spring 2000

Allocation of File Data to Disk Blocks

Questions answered in these notes

- What are typical file access patterns?
- How should disk blocks be allocated?
- What are the advantages and disadvantages of each approach?
- How should requests for blocks be scheduled?

Reading for this topic: Chapters 11 and 13

Access Patterns

Sequential Access

- Information is read (or written) in order
  - Most common access pattern
  - Editor writes out new file, compiler reads entire file, etc.
- Optimizes well (OS can prefetch, peak transfer rate from disk)

Random Access (Direct Access)

- Address any block directly without referencing predecessors
  - Data set for demand paging
- Difficult to optimize (Seek and rotational delays)

Keyed index (Associative)

- Return data associated with particular key value
  - Hash table
  - Database
- Usually not provided by OS (build on Random Access)

Workloads

Workloads determine design of file system

File characteristics (from measurements of UNIX)

- Most files are small
- Much of the disk is allocated to large files

Goals

- Per-file overhead low
- Access large files with good bandwidth

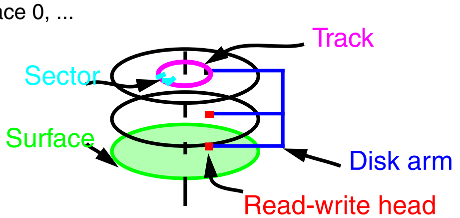
Common operations

- Access meta-data when access file data
- Locality: Access files in same directory
  - Need to know size of all files (ls -l)

Disk Management

Need to track where file is on disk

- How should disk sectors be used to represent blocks of file?
- Order disk sectors to minimize seek time
  - Track 0, surface 0, sector 0 --> sector 1, sector 2, sector 3, ...
  - Track 0, surface 1, sector 0, ...
  - Track 1, surface 0, ...



Track free versus allocated areas of disk

Keep track of allocation in meta-data associated with file

- Meta-data stored on disk

Finding Free Space

Track free blocks with bitmap (Unix, MacOS)

- Array of bits, one per block
- Usually keep entire bitmap in memory

Try to allocate next block of file close to previous block of file

- If disk is almost empty, this is easy
- As disk becomes full
  - Search becomes VERY expensive
  - Difficult to find adjacent blocks

Solution: Keep a reserve (e.g. 10% of disk) of disk always free

- Do not tell users about the reserve! (`df` --> 110% full)
- Multiple surfaces --> multiple adjacent blocks
  - With 10% of disk free, can almost always find one of them free

Allocation Strategies

Progression of different approaches (similar to memory allocation)

- Contiguous
- Extent-based
- Linked
- FAT tables
- Linked
- Multi-level Linked

Questions

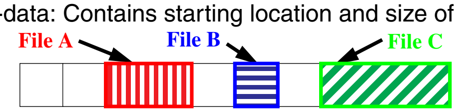
- Amount of fragmentation (internal and external)?
- Ability to grow file over time (after initial creation)?
- Speed to find data blocks for sequential and random accesses?
- Seek cost for sequential accesses?
- Wasted space for pointers to data blocks?

Determine advantages and disadvantages of each approach

Contiguous Allocation

Allocate files like segmented memory (base & bounds)

- User specifies length, file system allocates space all at once
- Find disk space by examining bitmap (apply first-fit or best-fit)
- Meta-data: Contains starting location and size of file



Example: IBM OS/360

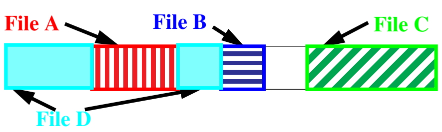
Advantages

Drawbacks

Extent-Based Allocation

Multiple contiguous regions per file

- Small array designating contiguous regions (2 to 6 entries)
  - Each entry: Starting location and size of region
- More common approach



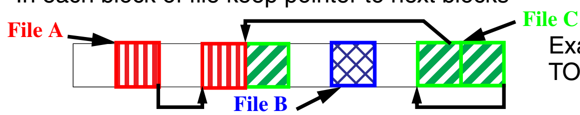
Advantages

Disadvantages

Linked Allocation

Meta-data: Pointer to first (fixed-size) block of file

- In each block of file keep pointer to next blocks



Examples:
TOPS-10, Alto

Advantages

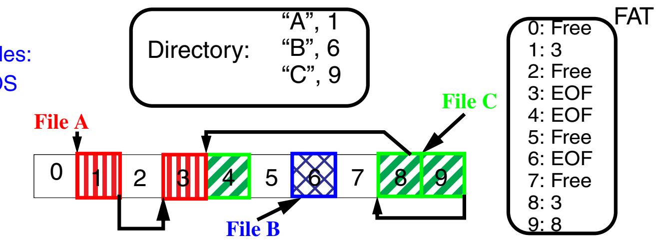
Drawbacks

Variation: FAT Table

Beginning of disk contains File-Allocation Table for all files

- One entry for each block of disk
- Linked-list of entries for each file

Examples:
MS-DOS
OS/2



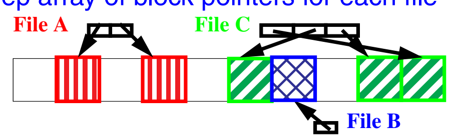
Disadvantages

Partial solution: Cache FAT in main memory

- Simplifies Random Access pattern

Indexed Allocation

Meta-data: Keep array of block pointers for each file



- All block pointers contiguously in meta-data
  - Maximum length declared when file created
  - Allocate pointers at creation, allocate disk blocks on demand
  - **Disadvantages:** Waste disk space for pointers until needed
    Need to know size of file at creation
- Maintain multiple lists of block pointers
  - Last entry points to next block of pointers
  - **Disadvantage:** Random access on large files traverses meta-data

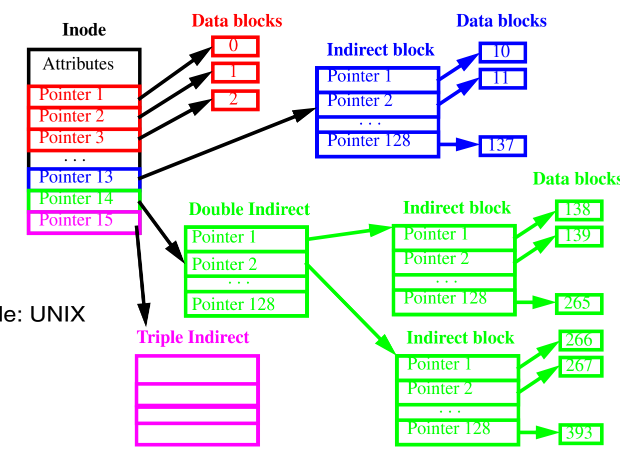
Advantage:

Disadvantage:

Multi-Level Indexed Files

Meta-data: 15 pointers to data and pointers to pointers

- 512-byte blocks



Example: UNIX

Multi-Level Indexing (Continued)

Advantages

- Incremental expansion
- Descriptor space not allocated until needed
- Fast access for small files

Drawbacks

- Indirect blocks not efficient for large files
  - Multiple lookups on disk for each real data access
- File and pointers may not be contiguously allocated on disk

File Cache

Cache disk blocks in main memory

- A pool of recently-accessed blocks kept in main memory
- More main memory --> larger file cache

Advantage

- If same blocks referenced again (e.g., indirect blocks), no need to re-read from disk
- Faster accesses if locality

Disadvantage

- Large files may flush useful data from cache
- File cache contends with virtual memory system for physical memory

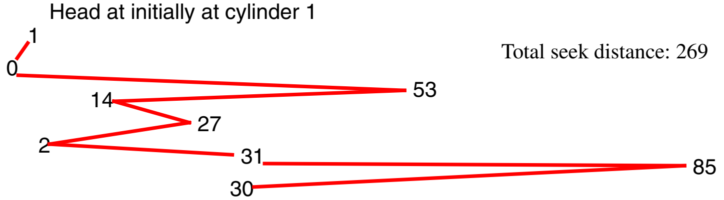
Disk Scheduling

Goal: Minimize time for disk seeks

- Observation for multiprogrammed systems:
  May have several requests queued at disk simultaneously

First come first served (FCFS, FIFO)

- Schedule disk requests in order received
  - Example: Read from cylinder numbers 0, 53, 14, 27, 2, 31, 85, 30
  - Head at initially at cylinder 1



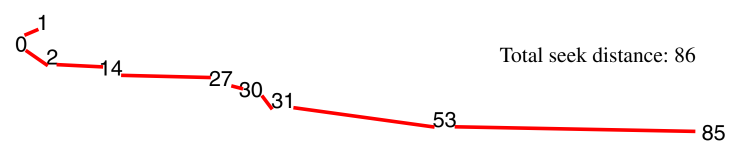
Advantage:Fair

Disadvantage: May result in unnecessary disk arm motion

Disk Scheduling (Continued)

Shortest seek time first (SSTF)

Handle nearest request first



Advantage

- Can reduce arm movement and result in better efficiency

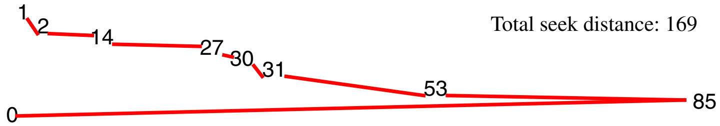
Disadvantage

- Unfair, some requests may have to wait a long time
  - Starve some requests if keep receiving closer requests

Disk Scheduling (Continued)

Scan (elevator algorithm)

- Move arm back and forth, handling requests underneath



Advantages

More fair to requests scattered throughout disk

Similar performance to SSTF

Disadvantage: Most efficient when disk is heavily loaded

Variation: Circular-Scan (C-Scan)

- Observation: Requests at other end of disk have been waiting longest
- Restart at beginning of disk when reach end

Current Approach: Take into account rotational delays as well