File System Implementations

1

Overview

- The Low Level
  - disks
  - caching
  - RAM disks
  - RAIDs
  - disk head scheduling
- The higher level
  - file systems
  - directories
  - sharing and access control

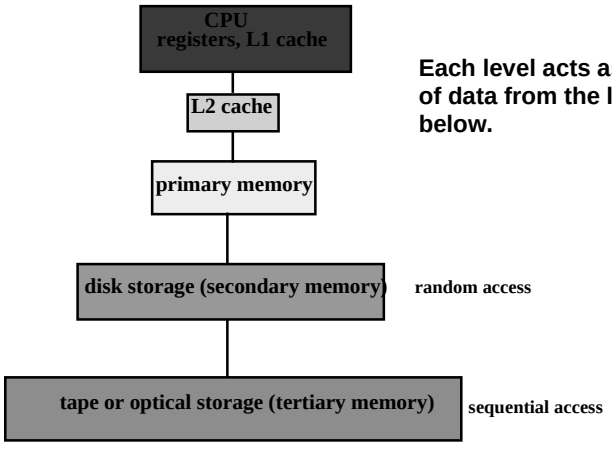
2

File Systems and Secondary Storage

- Secondary Storage is usually:
  - anything outside of “primary memory”
  - storage that does not permit direct instruction execution or data fetch by load/store instructions
- Characteristics of secondary storage
  - it’s large
    - hundreds of megabytes to gigabytes
  - it’s cheap
    - 1GB of disk now costs 300 dollars.
  - it’s non-volatile
    - when the power goes down, the data comes back.
  - it’s slow
    - lots of milliseconds to access

3

The Memory Hierarchy



Each level acts as a cache of data from the level below.

4

Types of Secondary Storage Devices

- Magnetic disks
  - fixed
  - removable
- Optical disks
  - write-once, read-many
  - write-many

5

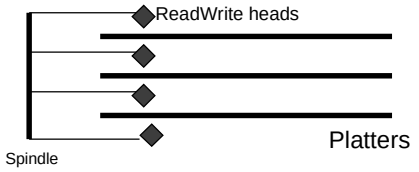
Disks and the Operating System

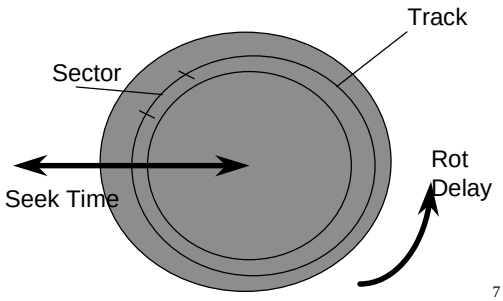
- The OS must deal with the mess of physical devices:
  - errors
  - bad blocks
  - missed seeks
- The job of the OS is to hide this mess from higher levels by:
  - providing low-level device control
  - providing higher-level abstractions:
    - files, database, ...
- The OS may provide different levels of disk access to different clients
  - physical disk block (surface, cylinder, sector)
  - disk logical block (disk block #)
  - file logical (file block, record, or byte #)

6

Physical Disk Structure

- Disk are made of thin metallic platters with a read/write head flying over it.
- To read from disk, we must specify:
  - cylinder #
  - surface #
  - sector #
  - transfer size
  - memory address
- Transfer time includes: seek, latency, and transfer time

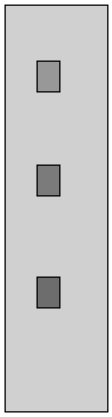




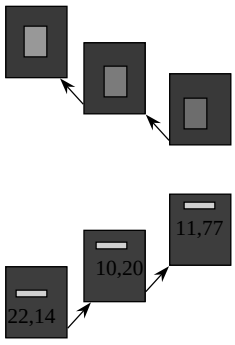
7

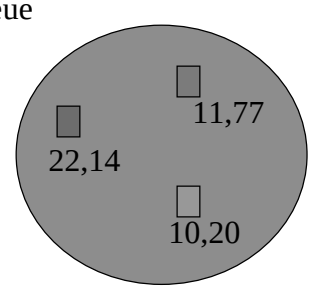
The Operating System's View of the Disk

Memory



The disk completed queue





The disk

The disk request queue

8

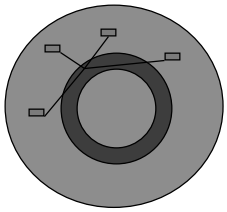
Some Typical Numbers

- Sector size: 4k
- Sectors per track: 32
- Cylinders per disk (tracks per platter): 1000
- Platters: 4-20
- Rotational speed: 3600 RPM
- Storage size: .5--2GB
- Seek time: 5-100ms
- Latency: 0--15Ms
- Xmission time: 1usec/byte

9

Disk Structure

- There is no structure to a disk except cylinders and sectors, anything else is up to the OS.
- The OS imposes some structure on disks.
- Each disk contains:
  - 1. data: e.g., user files
  - 2. meta-data: OS info describing the disk structure
- For example, the free list is a data structure indicating which disk blocks are free. It is stored on disk (usually) as a bit map: each bit corresponds to one disk block.
- The OS may keep the free list bit map in memory and write it back to disk from time to time.



10

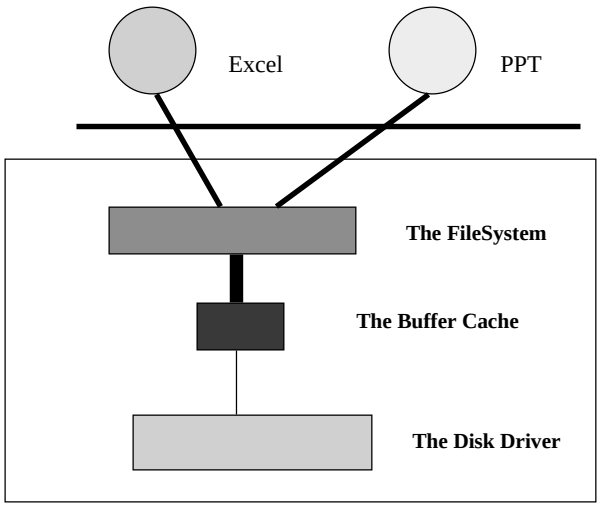
Dealing with Mechanical Latencies

- Caches
  - locality in file access
- RAM disk
  - cheap, slow, big memory that looks like disk
- RAID
  - parallelism
- Clever layouts and scheduling algorithms
  - head scheduling
  - meta-information layout

11

Disk Caching

- The idea is that data you (or someone) accessed recently is likely to be data that you (or someone) needs again.



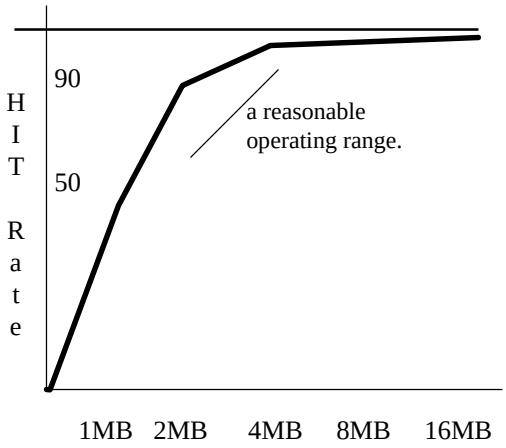
12

Disk Locality Works Quite Well

Most studies have shown that a disk cache on the order of 2-4 megabytes captures 90% of the read traffic for many kinds of applications

- your mileage will definitely vary
- but the impact can be enormous
  - 90% hit rate
  - hit time of .1 ms
  - miss time of 10ms
  - average access time

    (.9 * .1) + (.1 * 10) = 1.09 ms
- No cache increases disk access time by almost 1000%.



13

The Buffer Cache

- The buffer cache has only a finite number of blocks in it.
- On a cache miss, one block must be discarded in favor of the fresh block that is being brought in.
- Typical systems operate in an LRU order
  - the least recently used block is the one that should be discarded next.
  - favors “busier” portions of the filesystem
  - can hurt “the quiet” client.
- Some workloads LRU is really bad for
  - sequential scans
    - video, audio
  - random access
    - large database

14

Read Ahead

- Many file systems will implement “read ahead” whereby the file system will try to predict what the application is going to need next, go to disk, and read it in.
- The goal is to beat the application to the punch.
- For sequentially accessed files, this can be a big win.
  - as long as reads are spaced in time, the OS can schedule the next IO during the current gap.
  - enabling read ahead can give applications a 100% speedup.
    - suppose that it takes 10ms to read a block.
    - the application requests the next block in sequence every 10 ms.
    - with read ahead, wait time is 0 ms per block and execution time is
      - 10 * number of blocks read.
    - without read ahead, wait time is 20 ms per block and execution time is
      - 20 * number of blocks read.

15

Caching works for reads, what about writes?

- On write, it is necessary to ensure that the data makes it through the buffer cache and onto the disk.
- Consequently, writes, even with caching, can be slow.
- Systems do several things to compensate for this
  - “write-behind”
    - maintain a queue of uncommitted blocks
    - periodically flush the queue to disk
    - unreliable
  - battery backed up RAM
    - as with write-behind, but maintain the queue in battery backed up memory
    - expensive
  - log structured filed system
    - always write the next block on disk the one past where the last block was written
    - complicated

16

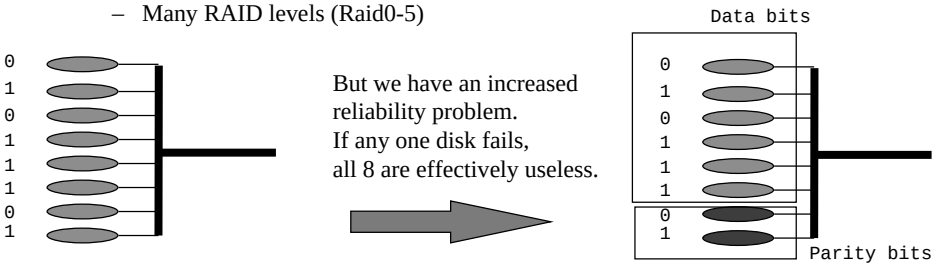
RAM Disk

- Dedicate a large chunk of memory to act as a file system.
  - Note that most disks now have a small amount of built in RAM already on them.
- Extremely simple and at times effective.
- Not so good for reliability
  - used to maintain temporary, or “expendable” files.
- Considering hit rates of caches, not clear that this is a great win.
- FLASH RAM is one kind of RAM disk that does not have a problem with power outage.
  - faster than disk, slower than memory
  - 10K writes, and you throw it away.

17

RAID

- Caching, RAM disks deal with the latency issue.
- DISKS can also be used in PARALLEL
- This is the idea behind RAIDs
  - Redundant Array of Inexpensive Disks
  - Many RAID levels (Raid0-5)



An array of inexpensive disks (Can read 8 tracks at once)

A redundant array of inexpensive disks.

18

Different RAID levels

- Raid 0: Striping
  - reduce disk queue length, good for random access. no reliability
- Raid 1: Mirroring
  - write data to both disks, expensive
- Raid 2: ECC
  - stripe at bit level, multiple parity bits to determine failed bit. expensive (eg, 10 data disks require 4 ECC disks), read/write 1 bit, means read/write all in a parity stripe!
- Raid 3: PARITY only.
  - disks say they break. Only need to reconstruct. Byte stripe. Single parity drive. All disks involved in a single read. Bottleneck writes.
- Raid 4: PARITY w/larger stripe size
  - can do multiple reads in parallel. Single parity drive. Bottleneck writes.
- Raid 5: Floating parity
  - assign parity bits for different stripes to different disks. Still have write bottleneck, but distributed. One write requires a read/write.

Disk Scheduling

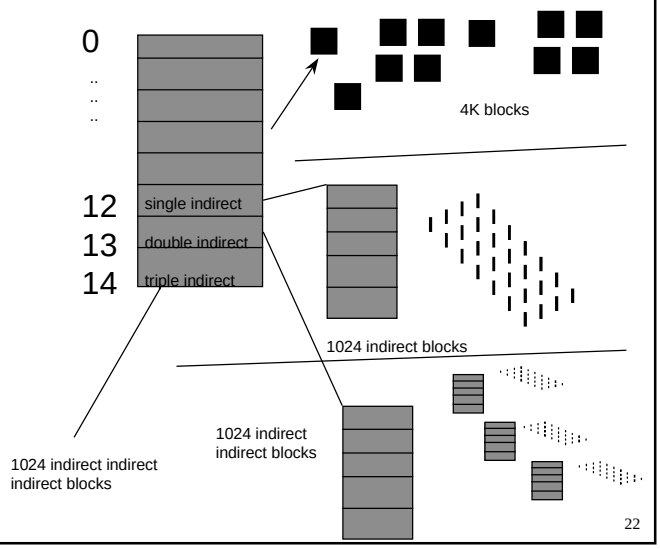
- Because disks are slow and seeks are long and depend on distance, we can schedule disk accesses, e.g.:
  - FCFS (do nothing)
    - ok when load is low
    - long waiting times for long request queue
  - SSTF (shortest seek time first)
    - always minimize arm movement. maximize throughput.
    - favors middle blocks
  - SCAN (elevator) -- continue in same direction until done, then reverse direction and service in that order
  - C-SCAN -- like scan, but go back to 0 at end
- In general, unless there are request queues, it doesn't matter
  - explains why some single user systems do badly under heavy load.
- The OS (or database system) may locate files strategically for performance reasons.

Disk Layout Strategies

- Files can be allocated on disk in different ways, e.g.:
  - 1. contiguous allocation
    - like memory
    - fast and simplifies directory access
    - inflexible, causes fragmentation, needs compaction
  - 2. linked structure
    - each block points to next block, directory points to first
    - good for sequential access (bad otherwise)
  - 3. indexed structure
    - an “index block” contains pointers to many other blocks
    - better for random access
    - may need multiple index blocks (linked together)

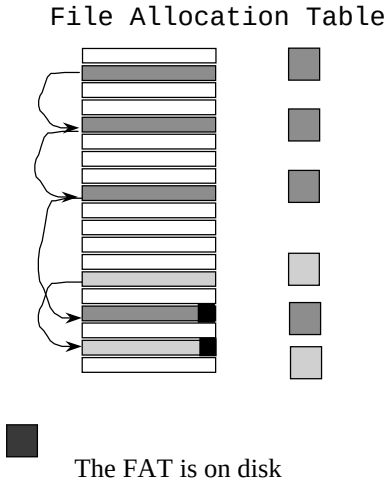
Eg, UNIX Inodes

- A UNIX inode is the metainformation for UNIX files.
- Contains control and allocation information.
- Each inode contains 15 block pointers.
  - first 11 are direct blocks
  - then single, double, and triple indirect



DOS FAT

- One file allocation table describes the layout of the entire disk.
- Each entry in the table refers to a specific cluster within a file.
  - zero says cluster not used.
  - not zero says where the next FAT entry for the file is.
- A file's directory entry points to the first FAT entry for the file.



The FAT is on disk in a special sequence blocks.

23

Limitations of FAT

- FAT index is 16 bits.
- 1 disk can have up 64K clusters.
- As disks get bigger, cluster size must increase.
  - eg, 16KB cluster for a 640MB disk.
- Big clusters yield internal fragmentation.
  - 10 to 20% wastage for 16KB clusters not uncommon.
- Minimum of one file per cluster.
  - limitation to 64K files.
- The FAT itself is a critical resource.
  - You lose the FAT on disk, you've lost the whole disk.
- And of course, the 8.3 limit.
  - Directory entries contained
    - 11 characters of name and 16 bits of FAT.

24

Summary

- Latency and bandwidth are a big problem in file systems
- Caching and readahead can help a tremendous amount
  - a little bit of mechanism can go a long way.
  - partly the reason why newer versions of an operating system can run so much faster.
  - partly also why your competitor's may be running so much slower.
- Many of the "interesting" issues in file systems are at the implementation level
  - within the operating system
- APIs are relatively similar, even across operating systems.