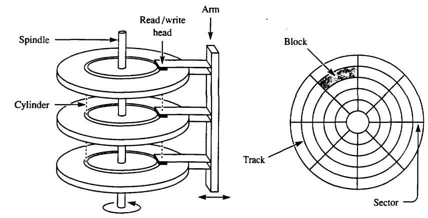
Lecture 16: File system interface/ Disk Management

- disk configuration and typical access times
- evolution of UNIX file system
- improving disk performance
  - using caching
  - using head scheduling

On previous lecture



- *Seek time* — time required to position heads over the track / cylinder
  - Typically 10 ms to cross entire disk
- *Rotational delay* — time required for sector to rotate underneath the head
  - 120 rotations / second = 8 ms / rotation

Disk access times

- Typically on a disk:
  - 32-64 sectors per track
  - 1K bytes per sector
- *Data transfer rate* is number of bytes rotating under the head per second
  - 1 KB / sector * 32 sectors / rotation * 120 rotations / second = 4 MB / s
- *Disk I/O time* = seek + rotational delay + transfer
  - If head is at a random place on the disk
    - Avg. seek time is 5 ms
    - Avg. rotational delay is 4 ms
    - Data transfer rate for a 1KB is 0.25 ms
    - I/O time = 9.25 ms for 1KB
    - Real transfer rate is roughly 100 KB / s
  - In contrast, memory access may be 20 MB / s (200 times faster)

Disk hardware (cont.)

- Typical disk today
  (Compaq 40GB Ultra ATA 100 7200RPM hard disk = $369):
  - 16383 cylinders, 16 heads, 63 sectors/track
  - 16 platters * 16383 tracks/platter * 63 sectors/track * 4048 bytes/sector * 1/1024^3 GB/byte = 63GB unformatted
  - 7200 rpm spindle speed, 8 ms average seek time, 100 MBps data transfer rate
- Trends in disk technology
  - Disks get smaller, for similar capacity
    - Faster data transfer, lighter weight
  - Disk are storing data more densely
    - Faster data transfer
    - Density improving faster than mechanical limitations (seek time, rotational delay)
  - Disks are getting cheaper (factor of 2 per year since 1991)

Selecting sector size

- The read / write head needs to synchronize with the track as it rotates
  - Need 100-1000 bits between each sector to measure how fast disk is spinning
- If sector size is 1 byte
  - Only 1% of disk holds useful data
  - 1/1000 transfer rate as before = 100 B / s
- If sector size is 1 KB
  - 90% of disk holds useful data
  - Transfer rate is 100 KB / s
- If sector size is 1 MB
  - Almost all of disk holds useful data
  - Transfer rate is 4 MB / s (full disk transfer rate — seek and rotational latency usually won't matter anymore)
  - space is wasted for small files!

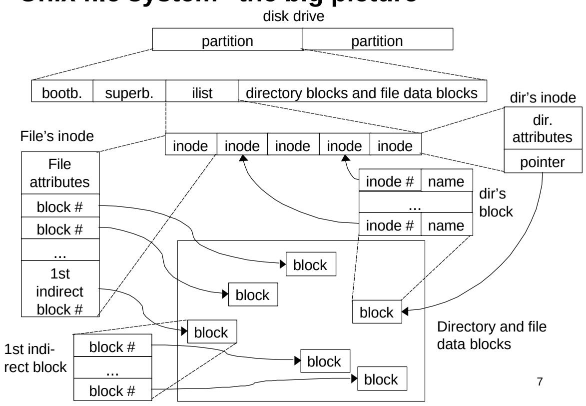
Selecting sector size (cont.)

- What about making the blocks bigger?
  - Causes internal fragmentation
  - Most files are small, maybe one block
- Some measurements from a file system at UC Berkeley:

Organization	Space used	Waste
Data only	775.2	0%
+inodes, 512B block	828.7	6.9%
+inodes, 1KB block	866.5	11.8%
+inodes, 2KB block	948.5	22.4%
+inodes, 4KB block	1128.3	45.6%

- The presence of small files kills the performance for large files!
  - Want big blocks to reduce the seek overhead for big files
  - But… big blocks increase fragmentation for small files

Unix file system “the big picture”



Traditional Unix File System

- In traditional UNIX (System V FS), and Berkeley BSD 3.0 UNIX
  - Disk lock size was 512 bytes
  - i-list follows superblock, has limited size determined at formatting (limits the number of files on system
  - directory contains fixed size records 16 bytes each (first two - i-node number, the rest - file name)
  - free blocks maintained in a linked list, superblock contains pointer to first
- problems with System V FS:
  - one superblock - becomes corrupted - filesystem unusable
  - all I-nodes at the beginning of disk - reading files requires accessing I-nodes - random disk access pattern
  - files blocks are allocated at random
  - practical measurements: when file system was first created
    - Free list was ordered, and they - transfer rates up to 175 KB / s
    - After a few weeks data and free blocks got so randomized - to 30 KB / s; less than 4% of the maximum transfer rate!
  - 14 character names insufficient

Unix Fast-File System

- In Berkeley BSD 4.2 UNIX:
  -
  - Introduced concept of a cylinder group
    - A cylinder is the set of corresponding tracks on all the disk surfaces
    - in the cylinder as it is to access any other
    - A cylinder group is a set of adjacent cylinders
  - Each cylinder group has a copy of super block, bit map of free blocks, ilist, and blocks for storing directories and files
  - The OS tries to put related information together into the same cylinder group
    - Try to put all i-nodes in a directory in the same cylinder group
    - try to put i-node and file blocks in the same cylinder group
    - Try to put blocks for one file contiguously in the same cylinder group: bitmap of free blocks makes this easy
    - For long files, redirect each megabyte to a new cylinder group

Unix FFS (cont.)

- In Berkeley BSD 4.2 UNIX: (cont.)
- Block size was changed to 4096 bytes
  - Reduced fragmentation as follows:
    - Each disk block can be used in its entirety, or can be broken up into 2, 4, or 8 fragments
    - For most of the blocks in the file, use the full block
    - For the last block in the file, use as small a fragment as possible
    - Can get as many as 8 very small files in one disk block
  - This change resulted in
    - Only as much fragmentation as a 1KB block size (w/ 4 fragments)
    - Data transfer rates that were 47% of the maximum rate
- Other improvements:
  - Bit map instead of unordered free list - each bit corresponds to a fragment
  - Variable length file names, symbolic links
  - File locking, disk quotas

Extent-based allocation, journalling

- Modern filesystems further improve on filesystem desing
- two improvements in Veritas file system (VxFS) from Veritas Software (see white paper reference on webpage)
- extent-based allocation
  - rather than refer to individual data blocks the index blocs specifies the beginning of an *extent* of continuously allocated blocks and the number of blocks in the extent
    - advantages - faster disk access, fewer indirections (combines the advantages of continuous and indexed allocation)
    - disadvantages - hard to select extent size
- journalling (also in NTFS and UFS in modern Unices)
  - updating data entails multiple operations in several places:
    - slow, not robust in case of a crash
  - *metadata* (directories, pointers, free list, etc.) needs to be updated
  - improvement: synchronously write changes to a file (called log or journal) and then asynchronously to all needed places on disk
    - advantage: sequential synchronous write instead of distributed asynchronous one

Improving performance with good block management

- OS usually keeps track of free blocks on the disk using a bit map
  - A bit map is just an array of bits
    - 1 means the block is free,
    - 0 means the block is allocated to a file
  - For a 1.2 GB drive, there are about 307,000 4KB blocks, so a bit map takes up 38.4 KB (usually kept in memory)
- Try to allocate the next block of the file close to the previous block
  - Works well if disk isn’t full
  - If disk is full, this is doesn’t work well
    - Solution — keep some space (about 10% of the disk) in reserve, and don’t tell users; never let disk get more than 90% full
    - With multiple platters / surfaces, there are many possibilities (one surface is as good as another), so the block can usually be allocated close to the previous one

Improving performance using disk cache

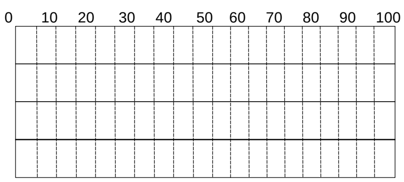
- Have OS (not hardware) manage a disk block cache to improve performance
  - Use part of main memory as a cache
  - When OS reads a file from disk, it copies those blocks into the cache
  - Before OS reads a file from disk, it first checks the cache to see if any of the blocks are there (if so, uses cached copy)
- *page cache* (Solaris, new Linux, NT)
  - storing files info as pages is more efficient than as blocks – can apply virtual memory techniques, if so – no reason to differentiate
  - *unified buffer cache* – combined (process and file I/O) paging what page replacement to use?
    - a variant of LRU seems good
    - optimization for files for sequential access
      - *free behind* – discards page as soon as it is read
      - *read ahead* – pages are read in advance

Improving performance with disk head scheduling

- Permute the order of the disk requests
  - From the order that they arrive in
  - Into an order that reduces the distance of seeks
- Examples:
  - Head just moved from lower-numbered track to get to track 30
  - Request queue: 61, 40, 18, 78
- Algorithms:
  - First-come first-served (FCFS)
  - Shortest Seek Time First (SSTF)
  - SCAN (0 to 100, 100 to 0, …)
  - C-SCAN (0 to 100, 0 to 100, …)
  - LOOK (lowest-highest, highest-lowest)
  - C-LOOK (lowest-highest, lowest-highest)

Disk head scheduling (cont.)

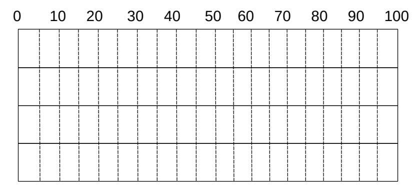
FCFS - handle in the order of arrival



- Advantages: simple, fair
- Disadvantages: can use disk inefficiently (if one person is using file on outer track, and other person is using file on inner track, will be many long seeks)

Disk head scheduling (cont.)

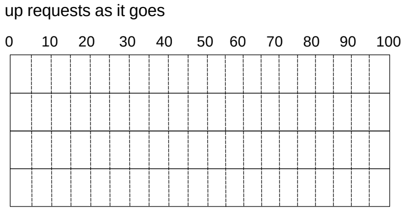
SSTF - select the request that requires the smallest seek from current track



- Advantages: reduces arm movement, uses the disk rather efficiently
- Disadvantages:
  - Fairness: disk can stay in one area for a long time (result = starvation)
  - Only accounts for seek time, not rotational delay (which is similar to seek time), so isn't a very good overall measure of time to access next block

Disk head scheduling (cont.)

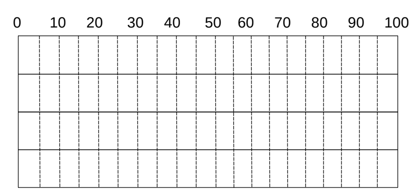
SCAN (elevator algorithm) - Move the head 0 to 100, 100 to 0, picking up requests as it goes



- Advantages: better fairness (no starvation), but not perfect
  - Request on edge of disk just behind in direction traveling can wait a long time to be serviced (twice disk length)
  - Even request in middle waits long time
- ek

Disk head scheduling (cont.)

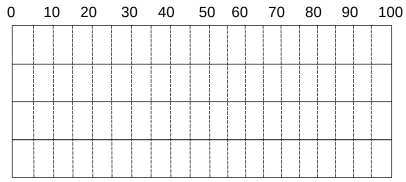
LOOK (variant of SCAN) - don't go to edges if there are no requests there



- Advantages: less wasted movement than SCAN

Disk head scheduling (cont.)

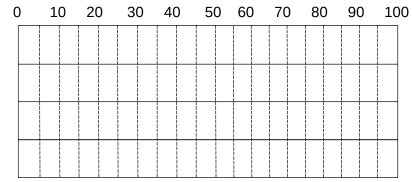
C-SCAN -Move the head 0 to 100, picking up requests as it goes, then big seek to 0



- Advantage: fairer than SCAN
- Disadvantage: big seek is just wasted time

Disk head scheduling (cont.)

C-LOOK -same as C-SCAN, don't go to edge if not necessary



- Used for head positioning, also used for rotation scheduling

Summary: improving disk performance

- Keep some structures in memory
  - Active inodes, file tables
- Efficient free space management
  - Bitmaps
- Careful allocation of disk blocks
  - Contiguous allocation where possible
  - Direct / indirect blocks
  - Good choice of block size
  - Cylinder groups
  - Keep some disk space in reserve
- Disk management
  - Cache of disk blocks
  - Disk scheduling

Disk management

- Disk formatting
  - Physical formatting — dividing disk into sectors: header, data area, trailer
  - Most disks are preformatted, although special utilities can reformat them
  - After formatting, must partition the disk, then write the data structures for the file system (logical formatting)
- *Boot block* contains the “bootstrap” program for the computer
  - System also contains a ROM with a bootstrap loader that loads this program
- Disk system should ignore bad blocks
  - When disk is formatted, a scan detects bad blocks and tells disk system not to assign those blocks to files
  - blocks may go bad as disk is used

Disk management (cont.)

- Disk reliability RAIDs
  - Data normally assumed to be persistent
  - Disk striping — data broken into blocks, successive blocks stored on separate drives
  - Mirroring — keep a “shadow” or “mirror” copy of the entire disk
  - Stable storage — data is never lost during an update — maintain two physical blocks for each logical block, and both must be same for a write to be successful
  - RAID5 - use parity disk