

I/O Systems

Arvind Krishnamurthy
Spring 2004



I/O Devices

- Typically, IO devices generate an interrupt when something happens
- Three alternatives:
 - Raise an interrupt for every byte transferred from/to the I/O device
 - Do block transfers to memory (direct memory access or DMA) and raise interrupt when transfer is done
 - Bitmap display – CPU writes to video memory what should be displayed, display reads video memory, illuminating pixels
 - Done at user-level without kernel intervention



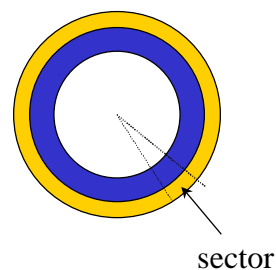
Performance issues

- Terminals, modems are connected to computers via serial lines
 - Raise an interrupt for every character
 - CPU could get swamped due to “overhead” of handling each byte
 - Example: 10 terminals connect at 9600 baud (about 1KB/s)
 - 10 thousand interrupts per second
 - If each interrupt takes 1 us to handle, load = 0.01 secs = 1%
 - If each interrupt takes 100 us to handle, CPU is loaded
- Block device performance depends
 - Overhead to initiate a request
 - Latency of performing a 1 byte transfer (dominates cost of small I/Os)
 - Bandwidth of I/O transfer once started (dominates cost of large transfers)



Disk organization

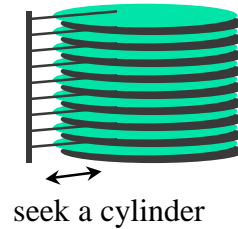
- Disk surface
 - Circular disk coated with magnetic material
- Tracks
 - Concentric rings around disk surface, bits laid out serially along each track
- Sectors
 - Each track is split into arc of track (min unit of transfer)





More on disks

- Magnetic disks come organized in a disk pack
- Cylinder
 - Certain track of the platter
- Disk arm
 - Seek to the right cylinder
- Disk is constantly spinning
- Disk operation is in radial coordinates (track #, sector #)



Disk performance

- Seek
 - Max seek cost is typically 10 ms
 - Average seek cost is typically 3-4 ms
- Rotational delay
 - Wait for a sector to rotate underneath the heads
 - Typically 8.3 – 5.0 ms (7,200 – 12,000RPM) or $\frac{1}{2}$ rotation takes **4.15-2.5ms**
- Transfer bytes
 - Average transfer bandwidth (**15-50 Mbytes/sec**)
- Performance of transferring 1 Kbytes
 - Seek (4 ms) + half rotational delay (3 ms) + transfer (0.04 ms)
 - Total time is 7.04 ms or **140 Kbytes/sec!**



Disk Performance


- Depends on locality of access
 - If reading next sector *immediately* after reading previous sector, no need to wait
 - If reading next sector after a small delay, need to wait for disk to finish the current rotation (assuming no **track buffers**)
 - If track buffers, next access within the same track is fast
- If random place in same cylinder
 - No seek needed
 - Just rotational delay (3ms)
 - Transfer (0.04 ms)
 - Total time is 3.04ms or about 330 KB/sec
- If random place in same cylinder, but read 10KB instead
 - Rotational delay (3ms) + Transfer (0.4ms)
 - Total time is 3.4ms or about 3MB/sec



Bandwidth w/ seek & rotation

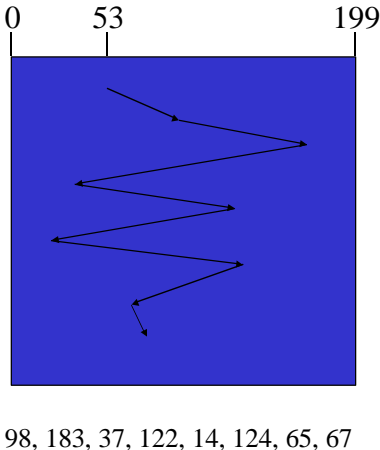
Block Size (Kbytes)	% of Disk Transfer Bandwidth
1Kbytes	0.5%
8Kbytes	3.7%
256Kbytes	55%
1Mbytes	83%
2Mbytes	90%

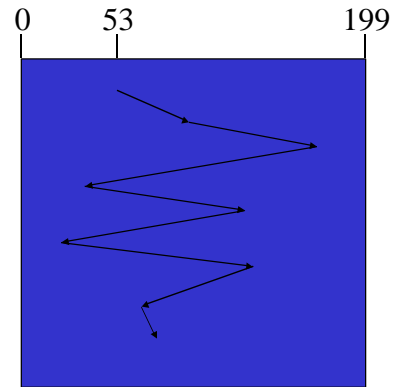
- Seek time and rotational latency dominates the cost of small reads
 - A lot of disk transfer bandwidth is wasted
 - Need algorithms to reduce seek time
- There are more sectors on outer tracks than inner tracks
 - For example, reading outer tracks could be 40MB/sec, while reading inner tracks could be 25MB/sec



Disk Performance Optimizations

- Minimize seek and rotational latency by putting related data together
- Always read/write large amounts of data
- If there are multiple requests, adaptively schedule the requests
- Use multiple arms? Use multiple disks?
 - Can we improve both bandwidth and latency?

- # Scheduling Disk Requests
- Method FIFO
 - 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)
 - Consider files accessed at a server – merging accesses from different users
 - Wild swings can happen
- 
- 98, 183, 37, 122, 14, 124, 65, 67

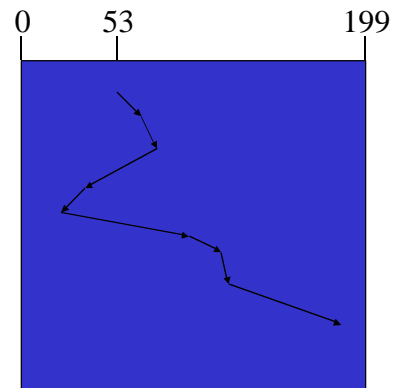


98, 183, 37, 122, 14, 124, 65, 67



SSTF (Shortest Seek Time First)

- Method
 - Pick the one closest on disk
- Pros
 - Try to minimize seek time
- Cons
 - Starvation
- Extension:
 - SATF – shortest access time first
 - Include rotational cost
 - Can we implement this??

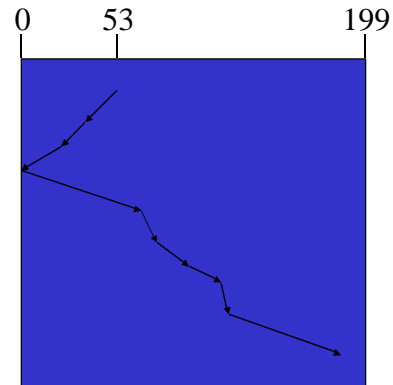


98, 183, 37, 122, 14, 124, 65, 67
(65, 67, 37, 14, 98, 122, 124, 183)



Elevator (SCAN)

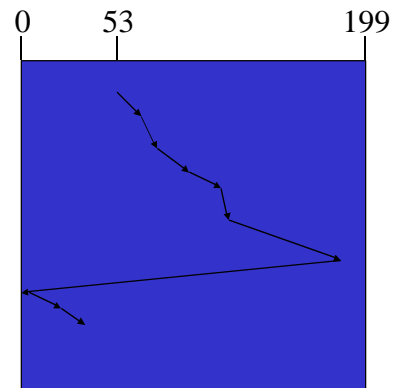
- Method
 - Take the closest request in the direction of travel
 - Real implementations do not go to the end
- Pros
 - Bounded time for each request
- Cons
 - Request at the other end will take a while



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

C-SCAN (Circular SCAN)

- Method
 - Like SCAN
 - But, wrap around
 - Real implementation doesn't go to the end
- 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)

Disk caching

- Method
 - Disk controller has a piece of RAM to cache recently accessed blocks
 - IBM Ultra160 SCSI has 16MB
 - Some of the RAM space stores "firmware" (an OS)
 - Blocks are replaced usually in LRU order
- Pros
 - Good for reads if you have locality
- Cons
 - Expensive
 - Need to deal with reliable writes



Disk Technology Trends

- Disks are getting cheaper (\$/MB)
 - About a factor of 2 per year since 1991
- Disk data is getting denser
 - More bits/square inch
 - Tracks are closer together
 - Doubles density every 18 months
 - Head close to surface
- Disks are getting smaller for similar capacity
 - Spin faster, less rotational delay, higher bandwidth
 - Less distance for head to travel (faster seeks)
 - Lighter weight (for portables)



Announcements

- Paper reviews for the week after spring break:
 - A fast file system for Unix (Monday: 3/22)
 - Log Structured File Systems (Wednesday: 3/24)



Upcoming lectures

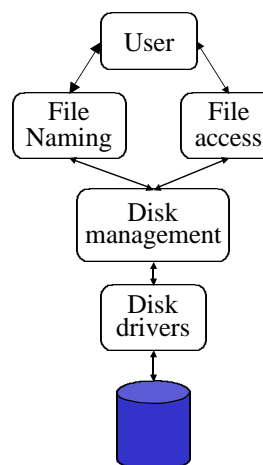
- Implementing file system abstraction

Physical Reality (Disks)	File System Abstraction
sector oriented	byte oriented
physical sector #'s	named files
no protection	users protected from each other
data might be corrupted if machine crashes	robust to machine failures



File System Components

- Disk management
 - Arrange collection of disk sectors into files
- Naming
 - User gives file name, not track or sector number, to locate data
- Protection
 - Keep information secure
- Reliability/durability
 - When system crashes, lose stuff in memory, but want files to be durable





File usage patterns

- How do users access files?
 - Sequential: bytes read in order
 - Random: read/write element out of middle of arrays
 - Content-based access: find me next byte starting with "CS422"
- How are files used?
 - Most files are small
 - Large files account for most of the bytes transferred
 - Large files use up most of the disk space
- Bad news
 - Need everything to be efficient



Data structures for Disks

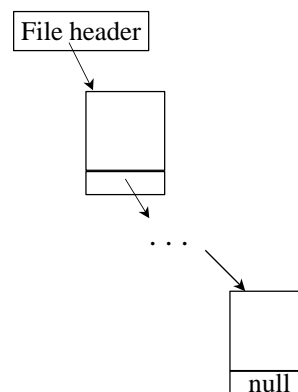
- A file header for each file (part of the file meta-data)
 - Disk sectors associated with each file
- A data structure to represent free space on disk
 - Bit map
 - 1 bit per block (or sector)
 - blocks numbered in cylinder-major order, why?
 - How much space does a bitmap need for a 4G disk?
 - Linked list
- Caching: keep most recently accessed blocks in memory

Approach 1: Contiguous allocation

- Request in advance for the size of the file
- Search bit map or linked list to locate a space
- File header
 - first sector in file
 - number of sectors
- Pros
 - Fast sequential access
 - Easy random access
- Cons
 - External fragmentation
 - Hard to grow files

Approach 2: Linked files

- File header points to 1st block on disk
- A block points to the next
- Pros
 - Can grow files dynamically
 - Free list is similar to a file
 - No waste of space
- Cons
 - Sequential access: slow
 - random access: horrible
 - unreliable: losing a block means losing the rest



Approach 3: File Allocation Table

- FAT Approach (used by MSDOS)

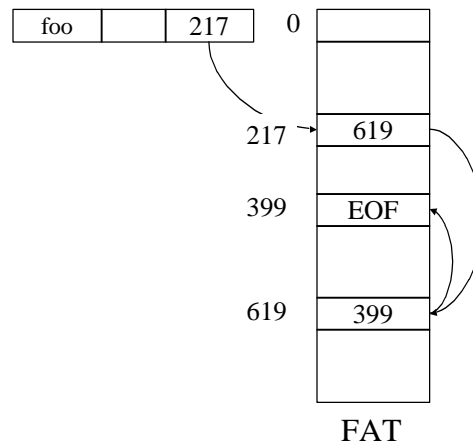
- A section of disk for each partition is reserved
- One entry for each block
- A file is a linked list of blocks
- A directory entry points to the 1st block of the file

- Pros

- Simple
- Improved random access

- Cons

- Always go to FAT



Approach 4: Indexed files

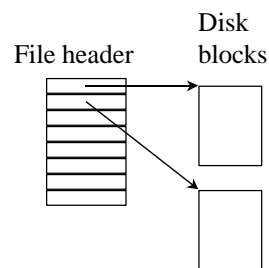
- Used in Nachos
- User declares max size
- A file header holds an array of pointers to point to disk blocks

- Pros

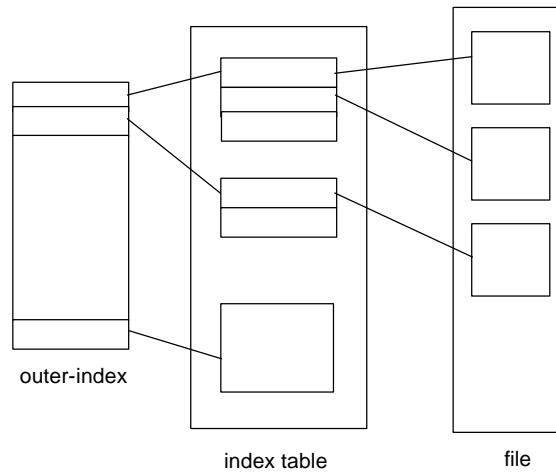
- Can grow up to a limit
- Random access is faster
- No external fragmentation

- Cons

- Clumsy to grow beyond the limit
- File headers typically need to be of fixed size



Multi-level indexed files



Combined Scheme (Unix 4.1)

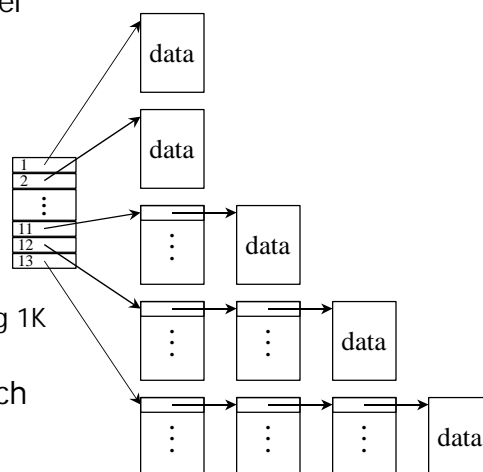
- 13 Pointers in a header

- 10 direct pointers
- 11: 1-level indirect
- 12: 2-level indirect
- 13: 3-level indirect

- Pros & Cons

- In favor of small files
- Can grow
- Limit is 16G (assuming 1K blocks)

- What happens to reach block 23, 5, 340?





File header storage

- Where is file header stored on disk?
 - In (early) Unix & DOS FAT file systems, special array in outermost cylinders
- Unix refers to file by index into array --- tells it where to find the file header
 - "i-node" --- file header; "i-number" --- index into the array
- Original Unix file header organization:
 - header not anywhere near the data blocks. To read a small file, seek to get header, seek back to data.
 - fixed size, set when disk is formatted.



File header storage (cont'd)

- Why not put headers near data?
 - Reliability: whatever happens to the data, you can find all of the files.
 - Unix BSD 4.2 puts portion of the file header array on each cylinder. For small directories, can fit all data, file headers, etc. in same cylinder → no seeks!
 - File headers are much smaller than a whole block (a few hundred bytes), so multiple file headers fetched from disk at same time.
- Q: do you ever look at a file header without reading the file?
 - Yes! Reading the header is 4 times more common than reading the file (e.g., ls, make).

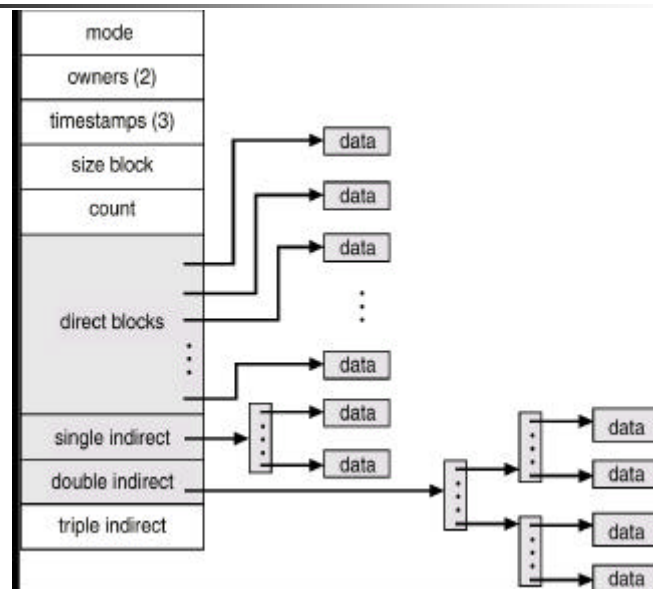


File attributes

- **Name**
- **Type** – needed for systems that support different types.
- **Location** – pointer to file blocks on device.
- **Size** – current file size.
- **Protection** – controls who can do reading, writing, executing.
- **Time, date, and user identification** – data for protection, security, and usage monitoring.
- Information about files are stored at:
 - Directory (typically contains just the name)
 - File-headers (typically contains all other file attributes)



Unix file header (I-node)





Disk Layout

Boot block	Super block	File descriptors (i-node in Unix)	File data blocks
---------------	----------------	--------------------------------------	------------------

- Superblock defines a file system
 - size of the file system
 - size of the file descriptor area
 - free list pointer, or pointer to bitmap
 - location of the file descriptor of the root directory
 - other meta-data such as permission and various times