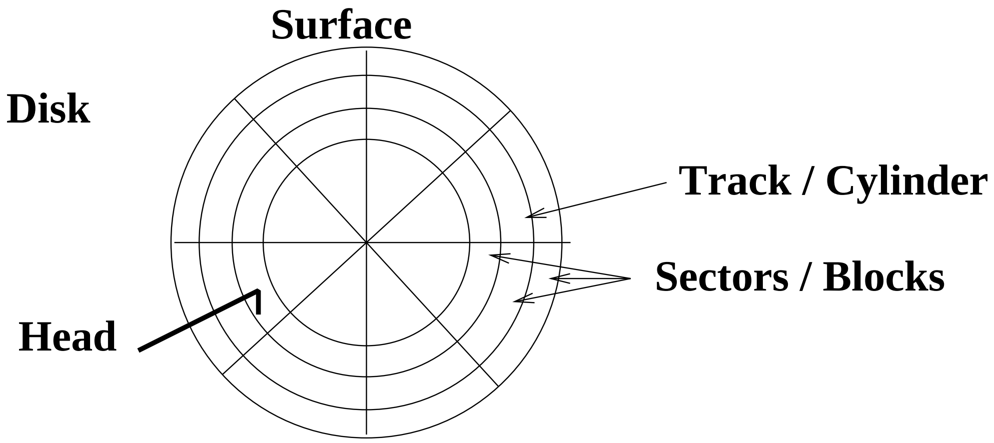
How Disks Work



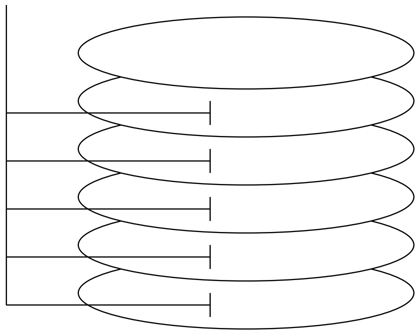
- The disk surface is circular and is coated with a magnetic material. The disk is always spinning, at constant angular velocity.
- Tracks are concentric rings on disk with bits laid out serially on tracks.
- Each track is split into *sectors* or *blocks*, the minimum unit of transfer from the disk.

Overhead: time the CPU takes to start a disk operation

Latency: the time to initiate a disk transfer of 1 byte to memory. Latency includes time to position the head over the correct cylinder (*seek time*), and the time for the correct sector to rotate under the head (*rotational delay*).

Bandwidth: once a transfer is initiated, the rate of I/O transfer.

How Disks Work



- Disks are organized in *disk pack* consisting of a stack of platters. Disk packs use both sides of the platters, except on the ends.
- Comb has 2 read/write head assemblies at the end of each arm.
- *Cylinders* are matching sectors on each surface.
- Disk operations are in terms of radial coordinates.
  1. Move arm to correct track, waiting for the disk to rotate under the head.
  2. Select and transfer the correct sector as it spins by.

How do we find and organize files on the disk?

The information we need:

fileID 0, Block 0 $\rightarrow$ Platter 0, cylinder 0, sector 0
fileID 0, Block 1 $\rightarrow$ Platter 4, cylinder 3, sector 8
...

Key performance issues:

1. We need to support sequential and random access.
2. What is the right data structure in which to maintain file location information?
3. How do we lay out the files on the physical disk?

Disk data structures:

- The structure used to describe where the file is on the disk and the attributes of the file is the *file descriptor (FileDesc)*. File descriptors have to be stored on disks just like files.
- Most systems fit the following profile:
  1. Most files are small.
  2. Most disk space is taken up by large files.
  3. I/O operations target both small and large files.

  $\Longrightarrow$ The per-file cost must be low, but large files must also have good performance.

How do we find and organize files on the disk?

Approaches:

1. Contiguous allocation
2. Linked files
3. Indexed files
4. Multilevel indexed files

Contiguous allocation:

- OS maintains an ordered list of free disk blocks
- OS allocates a contiguous chunk of free blocks when it creates a file.
- The location information in the file descriptor need only contain the start location and size.

Advantages:

- Simple
- Access time?
- Number of seeks?

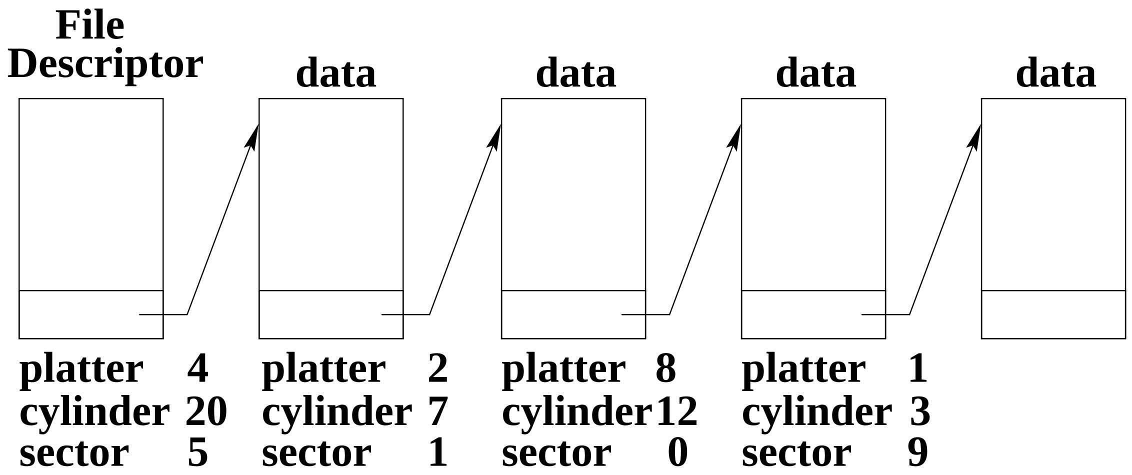
Disadvantages:

- Changing file sizes?
- Fragmentation?
- Disk management?

How do we find and organize files on the disk?

Linked files:

- Keep a list of all the free sectors/blocks.
- In the file descriptor, keep a pointer to the first sector/block.
- In each sector, keep a pointer to the next sector.



Advantages:

- Fragmentation?
- File size changes?
- Efficiently supports which type of access?

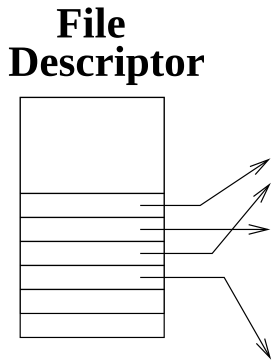
Disadvantages:

- Does not support which type of access? Why?
- Number of seeks?

How do we find and organize files on the disk?

Indexed files:

- OS keeps an array of block pointers for each file.
- The user or OS must declare the maximum length of the file when it is created.
- OS allocates an array to hold the pointers to all the blocks when it creates the file, but allocates the blocks only on demand.
- OS fills in the pointers as it allocates blocks.



Advantages:

- Not much wasted space.
- Both sequential and random accesses are easy.

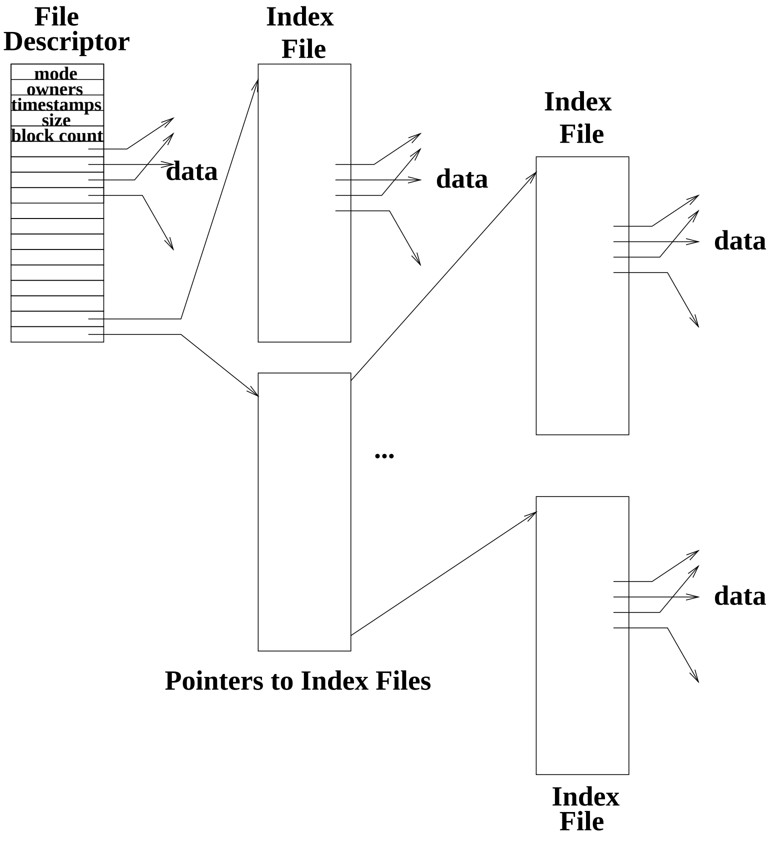
Disadvantages:

- Sets a maximum file size.
- Lots of seeks because data is not contiguous.

How do we find and organize files on the disk?

Multilevel indexed files: BSD UNIX 4.3

- Each file descriptor contains 14 block pointers.
- First 12 pointers point to data blocks.
- 13^{th} pointer points to a block of 1024 pointers to 1024 more data blocks. (One indirection.)
- 14^{th} pointer points to a block of pointers to indirect blocks. (Two indirections.)



Multilevel indexed files: BSD UNIX 4.3

Advantages:

- Simple to implement
- Supports incremental file growth
- Small files?

Disadvantages:

- Indirect access is inefficient for random access to very large files.
- Lots of seeks because data is not contiguous.

- Is the file size bounded?
- What could the OS do to get more contiguous access and fewer seeks?

Free-Space Management

- Need a free-space list to keep track of which disk blocks are free (much as we need a free-space list for main memory)
- Need to be able to find free space quickly and release space quickly.
- A common implementation is a *bitmap* data structure.
- A bitmap is a long array of bits, with one bit for each block on the disk.
- If the bit is 1, the block is free. If the bit is 0, the block is allocated.
- Can quickly determine if any page in the next 32 is free, by comparing the word to 0. If it is 0, all the pages are in use. Otherwise, you can use bit operations to find an empty block.

  110000100100011111110...

- Marking a block as freed is simple since the block number can be used to index into the bitmap to set a single bit.

Free-Space Management

- **Problem:** Bitmap might be too big to keep in memory for a large disk. A 2 GB disk with 512 byte sectors requires a bitmap with 4,000,000 entries (500,000 bytes).
- If most of the disk is in use, it will be expensive to find free blocks with a bitmap.
- An alternative implementation is to link together the free blocks. The head of the list is cached in kernel memory. Each block contains a pointer to the next free block.
  - How expensive is it to allocate a block?
  - How expensive is it to free a block?
  - How expensive is it to allocate consecutive blocks?

Summary

- Many of the concerns and implementations of file system implementations are similar to those of virtual memory implementations.
  - Contiguous allocation is simple, but suffers from external fragmentation, the need for compaction, and the need to move files as they grow.
  - Indexed allocation is very similar to page tables. A table maps from logical file blocks to physical disk blocks.
  - Free space can be managed using a bitmap or a linked list.