Today: File System Implementation

- How to organize data on to disks.
- Brief review of how disks work.

Disk management

```
Hardware

<table>
<thead>
<tr>
<th>Device Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tracks</td>
</tr>
<tr>
<td>Sectors</td>
</tr>
<tr>
<td>Writeblock()</td>
</tr>
<tr>
<td>Readblock()</td>
</tr>
<tr>
<td>Seek()</td>
</tr>
</tbody>
</table>

Independent Device Interface

<table>
<thead>
<tr>
<th>Programmer Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications</td>
</tr>
<tr>
<td>Daemons</td>
</tr>
<tr>
<td>Servers</td>
</tr>
<tr>
<td>Shell</td>
</tr>
</tbody>
</table>
```

Last Class: File System Abstraction
Bandwidth: once a transfer is initiated, the rate of I/O transfer.

- Rotational time: the time for the correct sector to rotate under the head.
- Seek time: time to position the head over the correct cylinder.

Latency: the time to initiate a disk transfer of 1 byte to memory.

Overhead: time the CPU takes to start a disk operation.

**Disk Overheads**

Each track is split into sectors or blocks. The minimum unit of transfer from the disk.

Tracks are concentric rings on disk with bits laid out sequentially on tracks.

The disk surface is circular and is coated with a magnetic material. The disk is always spinning (like a CD).

**How Disks Work**
3. How do we lay out the files on the physical disk?

2. What is the right data structure in which to maintain file location?

I. We need to support sequential and random access.

Key performance issues:

... 

File Organization on Disk

The information we need:

File Organization on Disk

I. Select and transfer the correct sector as it spins by.

2. Move arm to correct track, waiting for the disk to rotate under the head.

Disk operations are in terms of radial coordinates.

• Cylinders are matching sectors on each surface.

• Comb has 2 read/write heads assembled at the end of each arm.

• Disk packs use both sides of the platters, except on the ends.

• Platters.

• CD's come individually, but disks come organized in disk pack consisting of a stack of

How disks work
Examples: IBM OS/360, write-only disks, early personal computers

- Fragmentation? Disk management?
- Changing file sizes

Disadvantages

- Access time? Number of seeks?
- Simple?

Advantages

- Need to store only the start location and size in the file descriptor
- OS allocates a contiguous chunk of free blocks when it creates a file.
- OS maintains an ordered list of free disk blocks

Continuous Allocation

Performance:

- The per-file cost must be low, but large files must also have good
- 3. I/O operations target both small and large files.
- 2. Most disk space is taken up by large files.
- 1. Most files are small.

Most systems fit the following profile:

- To be stored on disks just like files.
- Attributes of the file is the file descriptor (fdes). File descriptors have
- The structure used to describe where the file is on the disk and the

File Organization: On-Disk Data Structures
Examples: MS-DOS

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

Disadvantages:

- Efficiency supports which type of access?
- File size change?
- Fragmentation?

Advantages:

Linked files

In each sector, keep a pointer to the next sector.

In the file descriptor, keep a pointer to the first sector/block.

Keep a list of all the free sectors/blocks.
Examples: NACOS

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

Disadvantages

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

Advantages

Index files

Descriptor

File

OS files in the pointers as it allocates blocks.

OS creates the file, but allocates the blocks only on demand.

OS allocates an array to hold the pointers to all the blocks when it is created.

The user or OS must declare the maximum length of the file when it is created.

OS keeps an array of block pointers for each file.
What could the OS do to get more continuous access and fewer seeks?

- Is the file size bounded?
  - Lots of seeks because data is not contiguous.
  - Indirect access is inefficient for random access to very large files.

Disadvantages

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

Advantages

Multilevel indexed files: BSD UNIX 4.3
an alternative implementation is to link together the free blocks.

- Binary
- If most of the disk is in use, it will be expensive to find free blocks with a
2 GB disk with 512 byte sectors requires a bitmap with 4,000,000 entries.

Problem: Bitmap might be too big to keep in memory for a large disk.

Free-Space Management

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

1100000100011111110

operations to find an empty block.
word to 0. If it is 0, all the pages are in use. Otherwise, you can use bit
Can quickly determine if any page in the next 32 is free, by comparing the
- If the bit is 1, the block is free. If the bit is 0, the block is allocated.
- The bitmap has one bit for each block on the disk.

Need to be able to find free space quickly and release space quickly.
(We need a free-space list for main memory)

Need a free-space list to keep track of which disk blocks are free (just as

Free-Space Management
– Free space can be managed using a bitmap or a linked list.
– To physical disk blocks.
– Indexed allocation is very similar to page tables. A table maps from logical file blocks.
– Compaction, and the need to move files as they grow.
– Continuous allocation is simpler, but suffers from external fragmentation, the need for
  implementations are similar to those of virtual memory implementations.

Many of the concerns and implementations of the system