File Systems
Ch 4.

File Systems

Manage and organize disk space.

Create and manage files.

Create and manage directories.

Manage free space.

Recover from errors.

File Systems

Complex data structure.

Provide an abstraction to the user.

Abstraction should be useful.

Implementation should be efficient.

Meta-Data

Data about the file.
Not the contents

Name.

Size.

Modification time.

Ownership and permissions.

etc.

File Names

Case-sensitive or not.

May include a type extension.

Windows OS uses extensions.

On Unix, some applications use them, but not the OS itself.

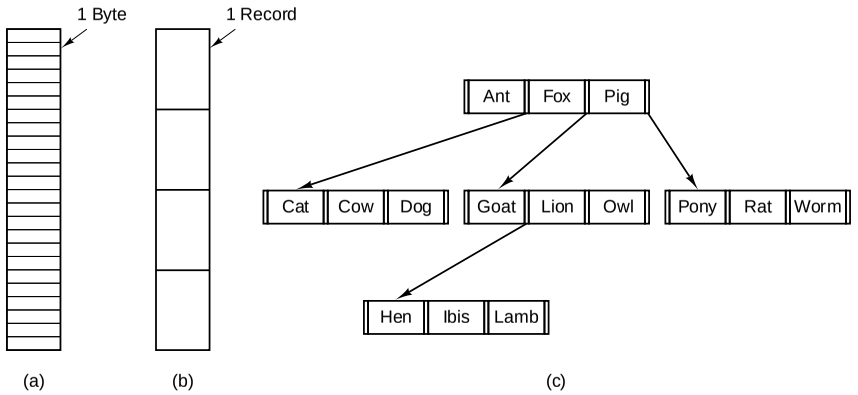
File Content Types

May record content type along with other *metadata*

May use an extension to indicate content type.

May just not record it.

Structures



Structures

A file is just a stream of bytes.
Most common; Unix and Windows.

A series of records.
Each read or write transfers one more records.

Tree-Structured.
Essentially a dictionary.

File Types

Regular files

Directories

Character Special Files
Block Special Files
Unix

O/S must recognize its own executable format.
May recognize other formats.

Generally, file contents are arranged by applications as desired.

Access Types

Sequential Access.

Random Access.

Attributes

Information beyond name and data.

Size Ownership Permissions

Create time Access time

Read-Only Archive info

Many possibilities.

Operations

Create Delete Open Close

Read Write Append Seek

Get attribs Set attribs

Rename

Unix File Operations

`open` `read` `write` `close`

Not to be confused with...

`fopen` `fclose` `printf` `. . .`

Memory-Mapped Files

A file can be added to the virtual memory.

Stores and fetches by the program change the file.

Unix call: `mmap`

Directory Structure

Single-Level

All files in one directory.
Not much used anymore.

One-Per User

Directory for each user, but no subdirectories.

General Tree

Pretty Much The Way To Do It.

Path Names

Absolute: `/usr/local/bin/hogwash`
Walk down the tree from the root.

Relative: `a.out` `public_html/index.html`
Relative to the current directory.

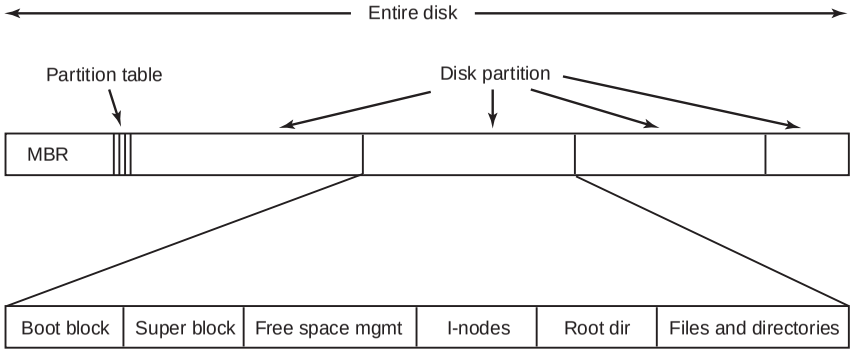
Separator character varies by O/S.

Directory Operations

create delete opendir readdir closedir

rename link unlink

Disk Layout



File System Layout

How are files organized.

Reading Performance.

Writing Performance.

Limits (or their lack).

Contiguous Files

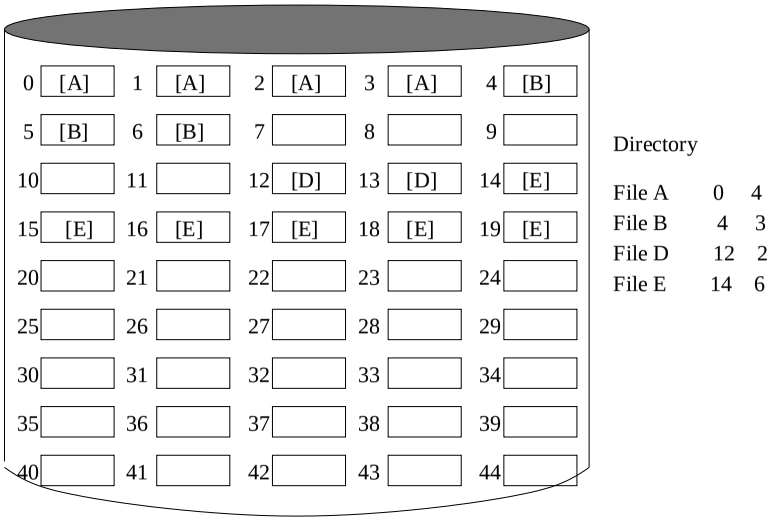
Each file is given a contiguous range of blocks.

New files added at the end until full.
Then have to find space among the holes.

Must know final size to allocate a file.

Once used on magnetic disks.
Now standard for CD-ROMS.

Contiguous Files



Directory

File A	0	4
File B	4	3
File D	12	2
File E	14	6

Linked Files

A file's blocks are organized as a linked list.

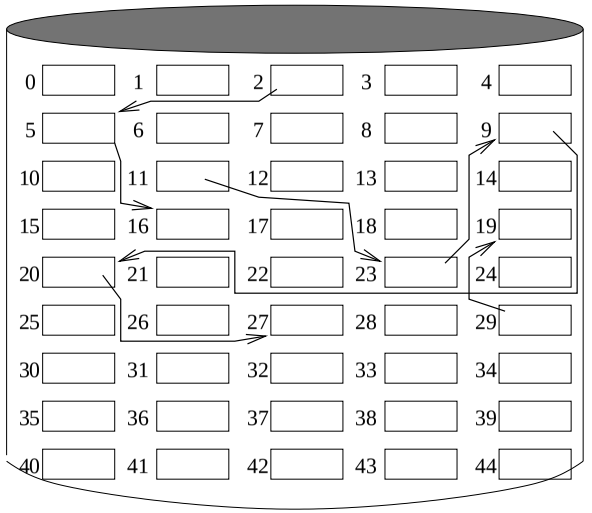
Sequential reading is fine.

Seeking is very expensive.
Requires reading all the blocks to follow the links.

File blocks are no longer powers of two.
Reduced by the size of the link.

Many programs read power-of-two blocks.
Having blocks a bit too small is inefficient.

Linked Files



Directory

File A	2
File B	11
File C	29

Linked With Separate FAT

Reserve small a portion of the disk as a File Allocation Table.

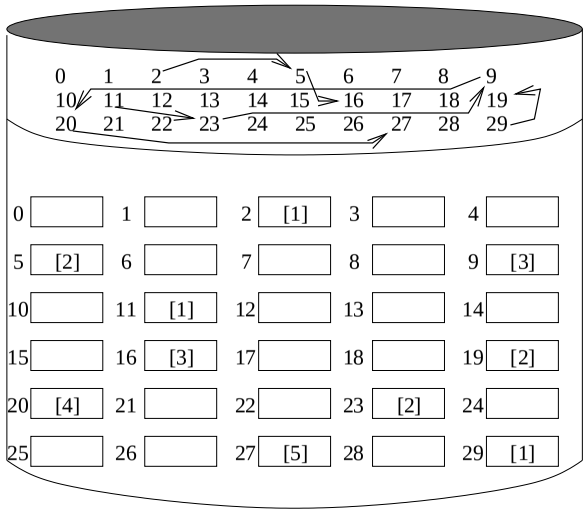
Divide this area up into links. Each is numbered from zero.

Number the remaining sectors from zero.

Each link in the FAT belongs logically to the sector of the same number.

The linked list is built in the FAT, not the sectors themselves.

FAT Organization



FAT Properties

FAT is copied from disk to main memory when the volume is mounted.

Links can be followed without reading disk.

Fixes the seek and power-of-two problems.

FAT grows as the volume grows.
And must be loaded into memory

Index Nodes

An index node is a disk sector used as an array of disk locations.
Part of the structure; no user data.

Each file has an index node containing the locations of its data blocks.

The last position is a link to another index block.

The index is copied into memory when the file is opened.

Unix uses a variation of this scheme.

Seeking

Reading should minimize file seeking.

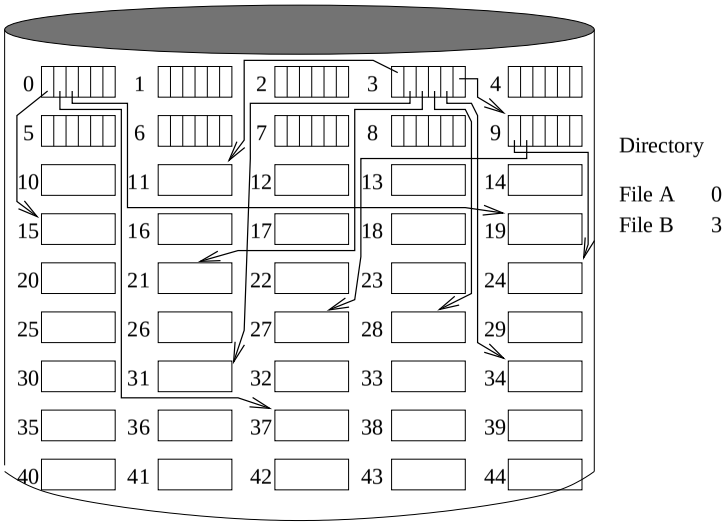
Continuous is good.

DOS FAT tends to fragment.
That's why you need to defrag them.

Most Unixes use cylinder blocks.
Tries to allocate a file's blocks near each other.

The cylinder blocks technique could be applied to linked structures also.

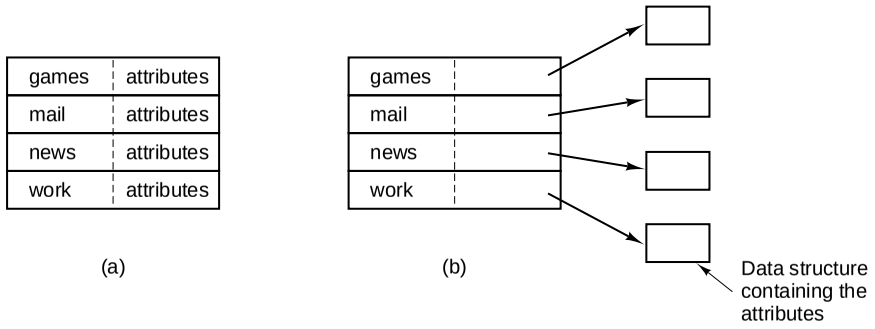
Index Node Organization



Directories

A list associating file names with files.

Attributes may live in the directory or in the index node.



Storing The File Name

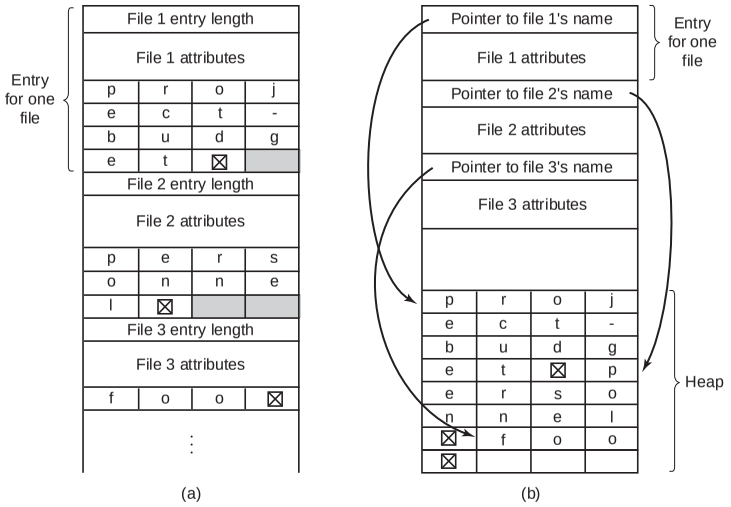
May use fixed-size directory entries, and limit the size of file names.
DOS *Many older systems*

May let the directory entry size vary.
Linux

May use part of the directory space to hold strings, and place pointers in the directory entries.

Both variable forms must manage holes
Searching is usually faster when they're in the string area.

File Name Alternatives



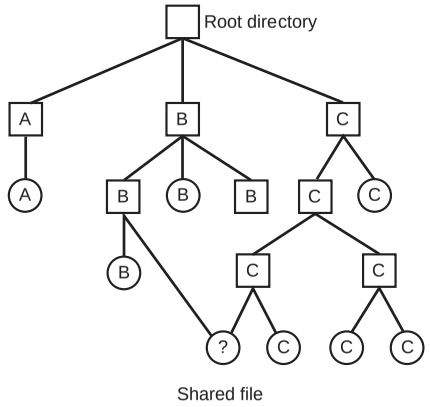
Directories

Usually use linear search.

A hash table is also possible.

Results are often cached.

Shared Files

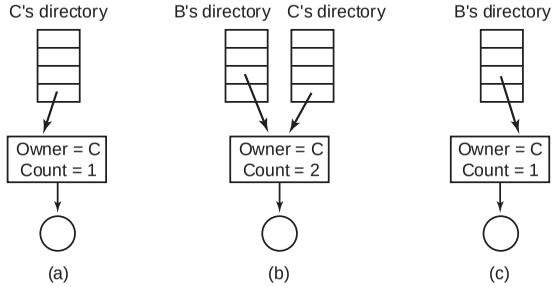


Shared Files

Can be useful for users working together.

A fix for when two cranky pieces of software have different notions about where things are.

Deletion can be problematic.



Approaches

Hard Links

Multiple directories refer to the same file.

Symbolic Links

A special file contains a file name.

Sort of like a forwarding address.

Block Size

Files need not be allocated by sector.
Sectors may be grouped into continuous blocks.

Larger Blocks

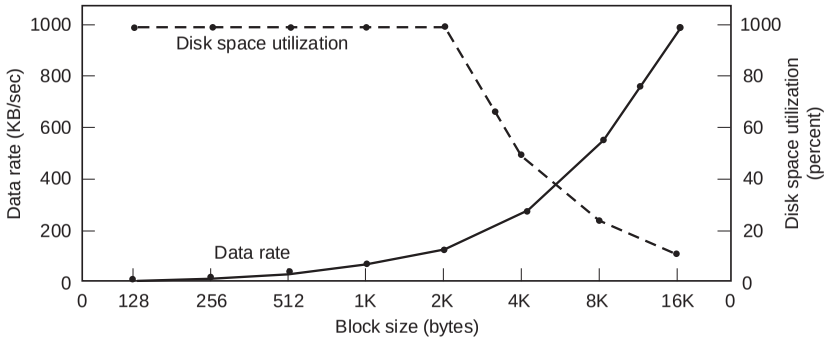
Reduce table sizes.

Reduce seek times.
Entire block can be read without seeks between sectors.

Increase waste due to fragmentation.

Block Size

Assuming all files are 2K.



Recording Free Blocks

Linked List

A linked list of blocks.

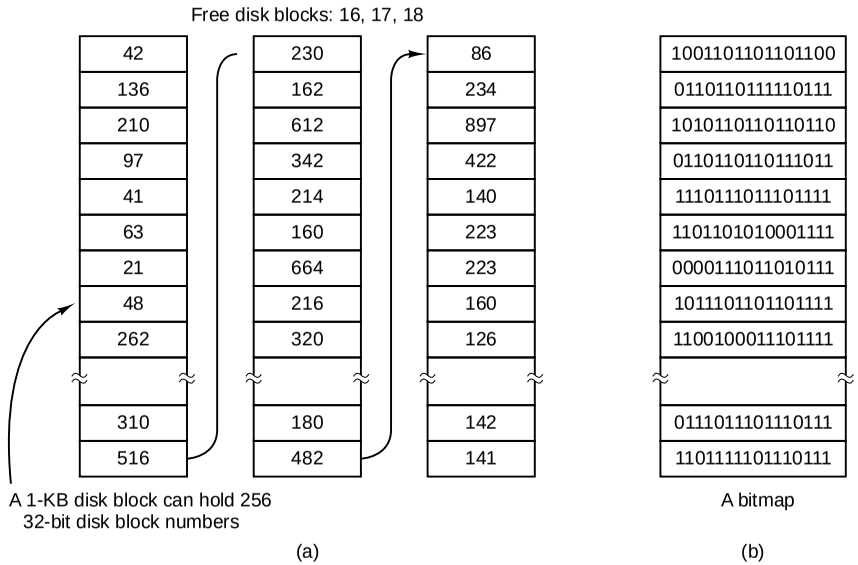
Each node is a disk block containing an
array of block numbers.

Bitmap

An array of bits on disk.
One bit for each managed block.

A one means the corresponding block is free.

Recording Free Blocks



Free List Advantages

Larger when the disk is emptier.
Can borrow the free blocks to hold the list.

Keep only the ending block in memory.

Add or remove from that block.

Write to memory if it fulls.

Read another if it empties.

Moving a half block at a time can avoid I/Os.

Bit Map Advantages

Can keep one block in memory for allocation.
Tends to allocate blocks close to each other.

Whole map can be paged.

Quotas

Keep a table of usage and limits for every user.

When a file is opened, keep a pointer to the owners quota.

Charge changes in file size to the quota.

Check quota when user logs in.

Hard and soft.

Backups

Disks fail. It's no fun.

People make mistakes. That's no fun either.

Backups help recover from either problem.

How To Spend Less Time Dumping

Don't Dump Everything.

Stuff that can be reinstalled from the OS installation media.

Temporary files.

Stuff that hasn't changed since last time.
Incremental dumps.

Compress it.
But don't let one bad block ruin the batch.

Active Systems

Usually better to shut down the system.

If files are being changed, the backup may miss something.
Also hard to tell "when" the backup was for a restore.

Algorithms for copying FS structures first
to capture its state at a single time.

Security Issues

Swiping a dump tape bypasses the best OS file security.

Off-site storage is wise.
And a security risk.

Dump Types

Physical Dump

Copy each physical disk block to a backup.

Doesn't know about the file system.

Useless copying of free blocks.
May copy bad blocks if visible to OS

Logical Dump

Copies by file; usually recursive directory copies.

Can do incrementals based on file write dates.

Creating a File in Unix

Remove a free inode from the free inode list.

Update the node.

Add a link to the inode to a directory.

Remove a data block from the free data block list.

Add a link from the inode to the data block

Other systems are of similar complexity.

What happens if the system crashes half way through all this?

File System Consistency

A file system must be *consistent.*

All the data blocks should be either free or part of a file.

No data block should be both free and part of a file.

No data block should be part of two files.

Etc.

Keeping It Okay

Operations are performed in a particular order
to minimize negative consequences.

Remove from the free list before adding to a file.

Requires disk operations ordered by structure, not seek time.

Requires synchronous disk operations.

File Consistency Checking

On boot after a crash, run a consistency checker.
scandisk *fsck*

Allocate used and free counters for each disk block.

Scan the file system and count the number of times each block
appears in some file or in the free list.

Every block should should be somewhere exactly once.

If not, make changes to render the system consistent.

Add orphan blocks to the free list.

A block both free allocated is removed from the free list.

Meta-data Logging

Standard on current file systems.

Keep a log of each change made the file system structure.

The log file is written frequently and in order.

After a crash, the log is used to return to a consistent state.
Much faster than a consistency check.

Optimizations: Caching

Blocks are kept in memory.

On a read, check for presence in the cache.
Usually use a hash for searching.

When a new block is read, an old one is removed, LRU.

Blocks not likely to be needed may be added near the front.

Modified blocks essential to the structure are
generally written immediately.

Writing Data Blocks

Modified data blocks may be written immediately:
Write-through cache.
DOS does this.

Newer systems write periodically.
Memory is efficient, but you want your data written.

Read Ahead

Most files are read sequentially.

Keep a sequentially flag for open files. Initially true.

After a block is read for a sequential file, read the next.

In case of a seek, turn off the sequential flag.

After several sequential reads, turn it back on.

Reducing Fragmentation

Allocate with larger blocks than other operations.

Cylinder Groups

Areas on disk with separate free list and inodes (if used).

Files grow within the group, if possible.

Keeps parts closer together.

Sources

Tanenbaum, *Modern Operating Systems*
(Course textbook.)