FILE SYSTEM IMPLEMENTATION

FILE SYSTEM STRUCTURE:

When talking about “the file system”, you are making a statement about both the rules used for file access, and about the algorithms used to implement those rules. Here’s a breakdown of those algorithmic pieces.

Application Programs	The code that's making a file request.
Logical File System	This is the highest level in the OS; it does protection, and security. Uses the directory structure to do name resolution.
File-organization Module	Here we read the **file control block** maintained in the directory so we know about files and the logical blocks where information about that file is located.
Basic File System	Knowing specific blocks to access, we can now make generic requests to the appropriate device driver.
IO Control	These are device drivers and interrupt handlers. They cause the device to transfer information between that device and CPU memory.
Devices	The disks / tapes / etc.

ALLOCATION METHODS

CONTIGUOUS ALLOCATION

- Method: Lay down the entire file on contiguous sectors of the disk. Define by a dyad <first block location, length >.

- Accessing the file requires a minimum of head movement. **<<< FIGURE 11.3>>>**

- Easy to calculate block location: block i of a file, starting at disk address **b**, is simply **b + i.**

- Difficulty is in finding the contiguous space, especially for a large file. Problem is one of dynamic allocation (first fit, best fit, etc.) which has external fragmentation. If many files are created/deleted, compaction will be necessary.

- It's hard to estimate at create time what the size of the file will ultimately be. What happens when we want to extend the file --- we must either term the owner of the file, or try to find a bigger hole.

LINKED ALLOCATION

- Each file is a linked list of disk blocks, scattered anywhere on the disk. **<<< FIGURE 11.4 >>>**

- At file creation time, simply tell the directory about the file. When writing, get a free block and write to it, enqueueing it to the file header.

- There's no external fragmentation since each request is for one block.

- Method can only be effectively used for sequential files.

- Pointers use up space in each block. Reliability is not high because any loss of a pointer loses the rest of the file.

- A File Allocation Table is a variation of this.

  It uses a separate disk area to hold the links. See **<<<FIGURE 11.5>>>**

  This method doesn't use space in data blocks. Many pointers may remain in memory.

  A FAT file system is used by MS-DOS.

INDEXED ALLOCATION

- Each file uses an index block on disk to contain addresses of other disk blocks used by the file. **<<< FIGURE 11.6 >>>**

- When the **i th** block is written, the address of a free block is placed at the **i th** position in the index block.

- Method suffers from wasted space since, for small files, most of the index block is wasted. What is the optimum size of an index block?

- If the index block is too small, we can:

  a) Link several together

  b) Use a multilevel index

  c) UNIX keeps 12 pointers to blocks in its header. If a file is longer than this, then it uses pointers to single, double, and triple level index blocks.

PERFORMANCE ISSUES FOR THESE METHODS

It's difficult to compare mechanisms because usage is different. Let's calculate, for each method, the number of disk accesses to read block i from a file:

contiguous: **1** access from location **start + i.**
linked: **i + 1** accesses, reading each block in turn. (is this a fair example?)
index: **2** accesses, 1 for index, 1 for data.

FREE SPACE MANAGEMENT

We need a way to keep track of space currently free. This information is needed when we want to create or add (allocate) to a file. When a file is deleted, we need to show what space is freed up.

BIT VECTOR METHOD

- Each block is represented by a bit

  1 1 0 0 1 1 0 means blocks 2, 3, 6 are free.

- This method allows an easy way of finding contiguous free blocks. Requires the overhead of disk space to hold the bitmap.

- A block is not REALLY allocated on the disk unless the bitmap is updated.

- What operations (disk requests) are required to create and allocate a file using this implementation?

FREE LIST METHOD

- Free blocks are chained together, each holding a pointer to the next one free.

- This is very inefficient since a disk access is required to look at each sector.

GROUPING METHOD

- In one free block, put lots of pointers to other free blocks. Include a pointer to the next block of pointers.

COUNTING METHOD

- Since many free blocks are contiguous, keep a list of dyads holding the starting address of a "chunk", and the number of blocks in that chunk.

  Format < disk address, number of free blocks >

DIRECTORY IMPLEMENTATION:

- The issue here is how to be able to search for information about a file in a directory given its name.
- Could have **linear list** of file names with pointers to the data blocks. This is:

  simple to program BUT time consuming to search.

- Could use hash table - a linear list with hash data structure.

  a) Use the filename to produce a value that's used as entry to hash table.

  b) Hash table contains where in the list the file data is located.

  - ✓ This decreases the directory search time (file creation and deletion are faster.)
  - ✓ Must contend with collisions - where two names hash to the same location.
  - ✓ The number of hashes generally can't be expanded on the fly.

GAINING CONSISTENCY:

Consistency checker - compares data in the directory structure with data blocks on disk and tries to fix inconsistencies. For example, What if a file has a pointer to a block, but the bit map for the free-space-management says that block isn't allocated.

Back-up - provides consistency by copying data to a "safe" place.

Recovery - occurs when lost data is retrieved from backup.

THE DISK CACHE MECHANISM

- This is an essential part of any well-performing Operating System.
- The goal of this subsystem is to ensure that the disk is accessed as seldom as possible.
- Algorithms are designed to keep previously read data in memory so that it might be read again.
- They also hold on to written data, hoping to aggregate several writes from a process before the block of data is written to the disk.
- Can also be “smart” and do things like read-ahead. Anticipate what will be needed.