Chapter 40: File System Implementation

1. Introduction

- This chapter delves into the implementation of a simple, yet functional, file system called **vsfs** (Very Simple File System).
- Understanding the on-disk structure and in-memory data structures is key to grasping how file systems work.

2. On-Disk File System Layout

- The disk is divided into several regions:
  - **Superblock:** Contains global information about the file system, such as its size, number of inodes, and a magic number to identify the file system type.
  - **Bitmaps:** Used to track free data blocks and inodes. A bit is set to 1 if the corresponding block/inode is in use, and 0 if it is free.
  - **Inodes:** A collection of on-disk structures that store metadata for each file.
  - **Data Blocks:** The majority of the disk space is dedicated to storing the actual file data.

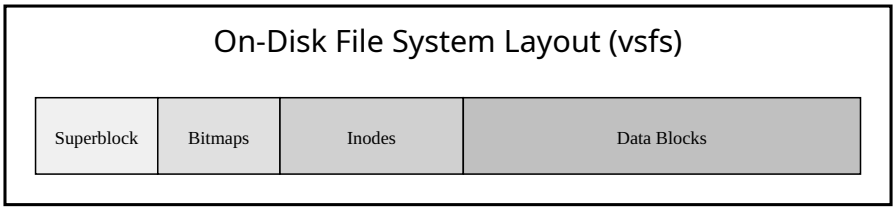

Figure 1: On-Disk Layout

3. The Inode

- The **inode** is the central data structure in a Unix-style file system. It stores all the metadata about a file, except for its name.
- Key fields in an inode:
  - **Type:** Indicates whether the inode represents a regular file, a directory, or another file type.
  - **Size:** The size of the file in bytes.
  - **Permissions:** Access control information (e.g., read, write, execute permissions).
  - **Timestamps:** Creation, modification, and last access times.
  - **Link Count:** The number of hard links to the file.
  - **Pointers to Data Blocks:** These pointers locate the actual data on the disk.

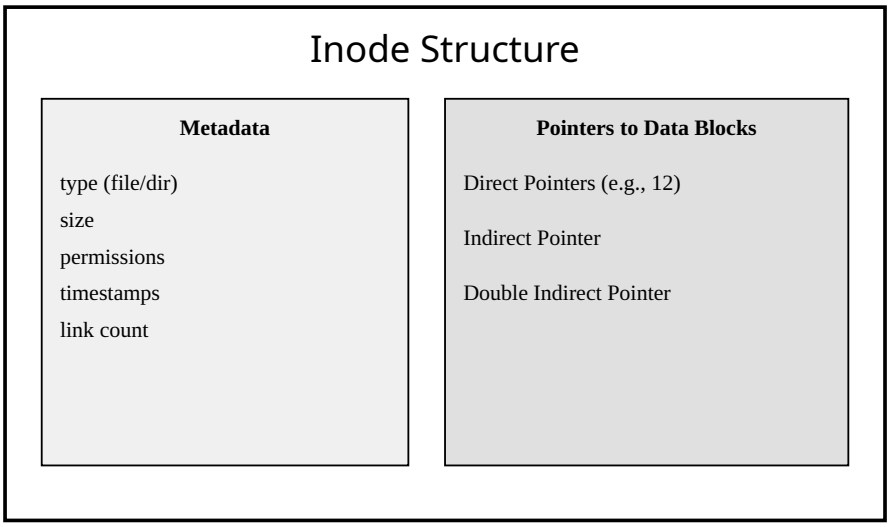

Figure 2: Inode Structure

Inode Code Example

```
typedef struct stat {
    int type;      // Type of file
    int dev;       // File system's disk device
    uint ino;      // Inode number
    short nlink;   // Number of links to file
    uint size;     // Size of file in bytes
} struct stat;
```

4. Multi-Level Index

- To support large files, inodes use a **multi-level index** to point to data blocks.
- **Direct Pointers:** A small number of direct pointers (e.g., 12) point directly to data blocks. This is efficient for small files.
- **Indirect Pointer:** An indirect pointer points to a block that contains an array of more direct pointers.
- **Double Indirect Pointer:** A double indirect pointer points to a block of indirect pointers, which in turn point to blocks of direct pointers. This allows for very large files.

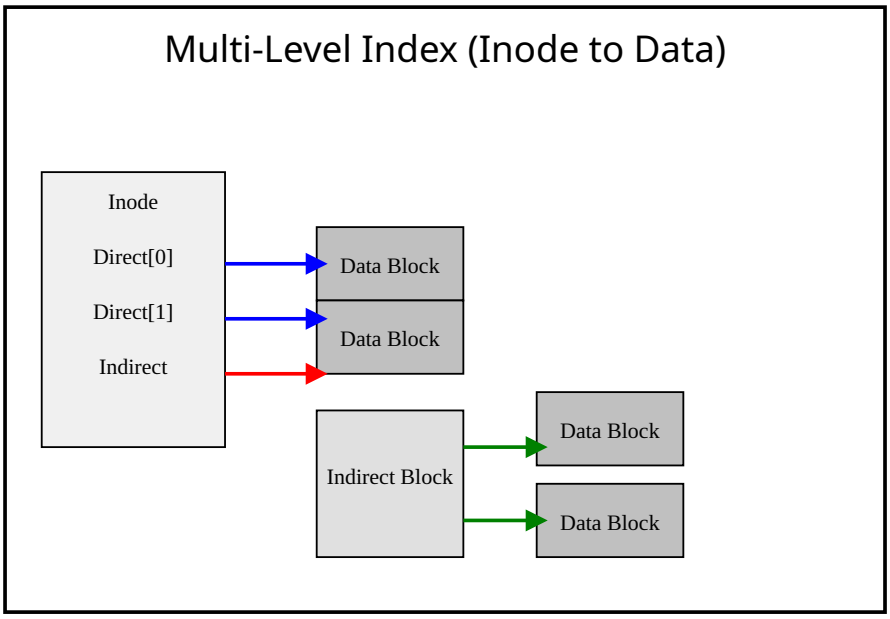

Figure 3: Multi-Level Index

5. Directory Organization

- A directory is essentially a file that contains a list of (name, inode number) pairs.
- When a file is opened, the file system traverses the directory tree, reading each directory to find the inode number of the next component in the path.

Directory Code Example

```
struct dirent {
    char name[28];
    int inum;
};
```

6. File System Operations in vsfs

- **Reading a file:**
  1. The file system reads the inode for the file.
  2. It then follows the pointers in the inode to find the data blocks.
  3. The data blocks are read from the disk into memory.
- **Creating a file:**
  1. The file system allocates an inode.

2. It updates the directory containing the new file to include a new entry with the file's name and inode number.

7. Caching and Buffering

- To improve performance, file systems make extensive use of **caching** and **buffering**.
- Frequently accessed blocks (such as the superblock, bitmaps, and inodes) are kept in memory to avoid slow disk I/O.
- This is crucial for making file system operations fast, as traversing the directory tree can involve many disk reads.