Chap 40: File System Implementation

**Very Simple File System** **(vsfs)** is a simplified version of a typical UNIX file system and thus serves to introduce some of the basic on-disk structures, access methods, and various policies that you will find in many file systems today.

**40.1 The Way To Think**

We will consider two aspects. The first is the **data structures** of the file system. In other words, what types of on-disk structures are utilized by the file system to organize its data and metadata? The second aspect of a file system is its **access methods**. How does it map the calls made by a process, such as open(), read(), write(), etc., onto its structures?

**40.2 Overall Organization**

The first thing we’ll need to do is divide the disk into **blocks**. Simple file systems use just one block size, and that’s exactly what we’ll do here. Let’s choose a commonly used size of 4 KB. Lets say that there are N blocks, from 0 to N-1, and there are only 64 blocks.

To store these blocks to build a file system, we first consider user data. Most of the space in any file system is user data. Lets call this region **data region** and let this region take 56 blocks:

Text

Description automatically generated

The file system has to track information about each file. This information is a key piece of **metadata**, and tracks things like which data blocks (in the data region) comprise a file, the size of the file, its owner and access rights, access and modify times, and other similar kinds of information. To store this information, the file systems usually have a structure called an **inode**.

To accommodate inodes, we’ll need to reserve some space on the disk for them as well. Let’s call this portion of the disk the **inode table**, which simply holds an array of on-disk inodes. We will use 5 blocks for inodes:

Text

Description automatically generated

The inodes are not big, about 128 or 256 bytes.

We also need a way to track whether inodes or data blocks are free or allocated, or **allocation structures**. We can use a **free list**, but in this case, we will use a structure called **bitmap**, one for the data region (the **data bitmap**), and one for the inode table (the **inode bitmap**). Bitmap is a simple structure to indicate if a corresponding block/object is free(0) or in-use(1). Therefore, with an inode bitmap (i) and data bitmap(d), we have:

Text

Description automatically generated

The one block left is served as **superblock**, which contains information about this particular file system, including, for example, how many inodes and data blocks are in the file system, where the inode table begins and magic number to identify the file system type.

Thus, when mounting a file system, the operating system will read the superblock first, to initialize various parameters, and then attach the volume to the file-system tree. When files within the volume are accessed, the system will thus know exactly where to look for the needed on-disk structures.

**40.3 File Organization: The Inode**

Inode is short for **index node**. Each inode is implicitly referred to by a number (called the **i-number**), which we’ve earlier called the **low-level name** of the file. In vsfs, we should be able to calculate where on the disk the corresponding inode is located based on an i-number. Assuming that each inode is 256 bytes and starting at 12KB. We will have the following inode table:

Table

Description automatically generated

The sector address of the inode block can be calculated as follows:



Inside each inode is virtually all of the information you need about a file, including its type, size, the number of blocks allocated to it, protection information (who owns the file and who can access it), some **time information** and where its data blocks reside in disk. We call all such information as **metadata**.

One of the most important decisions in the design of the inode is how it refers to where data blocks are. One simple approach would be to have one or more direct pointers (disk addresses) inside the inode; each pointer refers to one disk block that belongs to the file.