CS 444 - Spring 2016 - Writing Assignment 5

Cody Malick

malickc@oregonstate.edu

Abstract

Understanding how an operating system handles its file systems and virtual file systems, and how that helps make the user's life easier by abstracting the physical system is an important topic for an aspiring kernel developer. In this report, we will cover how Linux, Windows, and FreeBSD handle the VFS in their respective kernels.

Contents

I. Introduction

As a general user, you don't want to have to deal directly with a computers input and output systems. It would be tiresome and tedious. To solve this problem, the file system and virtual file systems are wrappers around these arduous tasks that make it simple, and user friendly. The VFS is the unification of two ideas: making the user's life easy, while providing a generic interface that allows different types of file systems the same interfaces. Following is an examination of the FS and VFS implementations in Linux, Windows, and FreeBSD.

II. Linux

A. File Systems

A file system, simply, is an abstraction of the IO layer that allows ease of use for the user in tracking, reading, and writing to disk. Instead of forcing the user to remember where a file is, what its size is, what type of file it is, etc., the file system provides interfaces that make this a, relatively, easy task. There are many different types of file systems. They are differentiated by minimum and maximum file size, journaling capability, and maximum partition size.

1) Linux Default File Systems: There are a few default file systems available in Linux. A few of them are all from the `EXT` family, like `EXT2`, `EXT3`, and `EXT4`. `EXT3` is considered the standard FS for stable Linux builds, but `EXT4` is the latest and greatest release from that family. The biggest difference between `EXT4` and its previous version is the maximum file and partition size.[1] One the great things about Linux is the number of options for file systems. If there are features you're looking for, but don't have, then you can pick one that works better for your needs. If you want a bleeding edge system, you can also grab your favorite choice from the internet.

B. Virtual Filesystem

The VFS lies at the top layer of the OS input-output system. The virtual filesystem allows user-space programs to use standard unix systems calls such `read()` and `write()` regardless of storage medium or underlying file system. This is setup is only possible because Linux adds a layer of abstraction around the low-level filesystem. Although we take such functionality for granted, it is a very important system to have. The VFS is divide into two overall pieces: structures, and their associated operations. [2]

C. VFS Structures

The VFS provides an abstraction of the filesystem, or multiple filesystems through a few specific interfaces. Specifically, the VFS provides the following abstractions:

Files: an ordered string of bytes.

Dentries: Files that contain directory information. For example, '\this\is\a\path' has four dentries on the path to that directory. This is how the VFS contains directories as files.

Inodes: Otherwise known as an 'index node,' these files contain metadata about other files such as permissions, size, owner, etc.

Superblock: A file containing all the relevant information about a filesystem as a whole. This is a metadata file for an entire FS.

Each of these interfaces have sizable data structures, along with a corresponding operations struct that complements them, containing function pointers to give the VFS the great functionality it has.[2]

These interfaces are critical to how we use Linux every day. Without them, we would have to make manual calls to the different filesystems that managed different devices, in the protocols they require. Sounds like a mess!

III. Windows

Windows does things quite a bit different than Linux as far as file systems go. This comes as no surprise as Windows has a trend of doing its own thing. Because it is not originally a Unix based or Unix inspired operating system, we get to see a very different approach to solving similiar problems. In the following section, we will examine the Windows file system, `NTFS`, and its virtual file system layer.

A. NTFS

Windows has a few different file systems available by default. The native file system, however, is called `NTFS`, or New Technology File System. `NTFS` is fairly comperable to `EXT4` in maximum partition and file size. `NTFS` theoretically supports up to exabyte volume sizes, but Windows currently limits support to 256 terrabyte volume sizes with 64 kilobyte cluster sizes. `NTFS` has the normal array of features that standard file systems ship with, along with file and directory security, alternate data streams, file compression, symbolic and hard links, encryption, and transactional semantics. [3]

B. NTFS Driver

Windows implements the virtual file system layer using the file system module itself. In this, Windows doesn't have a distinct VFS layer. It implements the file system and the VFS in one module, the `NTFS` module. This is a hard contract between Linux and Windows. Linux seperates the VFS layer entirely, and then has the FS plug into the VFS and integrate. If you wanted to plug another file system in to Windows, it would use a distinctly different set of interfaces from another. So if you wanted to use a `FAT32` file system type, it would not easily integrate with a `NTFS` file system. [3]

Now that we've examined how Windows handles its filesystem layer, and saw how different its monolithic approach is Linux' abstracted layers approach, we will examine how FreeBSD handles these same concepts.

IV. FreeBSD

FreeBSD is very similar to Linux in how it handles many things. The file system is no exception to this rule. FreeBSD is very similiar to Linux in this department, and we examine the subtle differences it has.

A. The Fast File System

The Fast File System is FreeBSD's default file system. It is fairly standard as File systems come, and is well tested and stable. The FS has features like directory structure, file names, journaling, etc. The standard set of features. It does not do file and directory security, like Windows. Permissions are managed, but not access to specific files and directories.

B. Virtual File System

The VFS setup in FreeBSD has a few structs that are similiar to the Linux structs. Something interesting about the FreeBSD versus the Linux is that FreeBSD makes explicit statements about how a file's existence is defined. For example, a file in FreeBSD exists until no references or descriptors are open in the OS. Linux may have similiar functions, but the rules for these are not explicitly defined. Here are the basic objects that make up the FreeBSD basic IO system: [4]

> Files: A Linear array of bytes with at least one name. A file exists until all its names are explicitly deleted explicitly and no process holds a descriptor of that file. All IO devices are treated as files.
>
> Directory Entry: A file that contains information about itself and other files and directory entries.
>
> Pipes: Pipes are also a linear array of bytes, but are used exclusively for one direction data transfer through the use of an IO stream. If you need to read and write from a file at the same time, two pipes will need to be open.
>
> Socket: An object that is used for interprocess communication, it exists only as long as some process holds the descriptor for it.

This setup is almost identical to how Linux handles things, but with one big difference: other files sytems mounted inside of the virtual file system are treated simply as directories, not as whole units like Linux's superblocks. This simplifies the handling of the entire filesystem structure. A root directory of a filesystem is set so that the OS knows where everything starts.

V. Conclusion

Linux, Windows, and FreeBSD all have their own way of doing things. Although Linux and FreeBSD share a lot of the same ideas, they are unique. Windows is another beast entirely with its modules and driver structures. Understanding how all these systems work are important as a developer working in the each of their respective kernels.

VI. Appendix A - Linux Structs

All following structs have been pulled from the classroom text. [2]

```
for future code samples
```

VII. Appendix B - Windows Structs

```
for future code samples
```

VIII. Appendix C - FreeBSD Structs

```
for future code samples
```

sectionBibliography

References

[1] A. J. Simon. (2015, nov) Linux file systems explained. [Online]. Available: https://help.ubuntu.com/community/LinuxFilesystemsExplained

[2] R. Love, *Linux Kernel Development*, 3rd ed. Developer's Library, 2010.

[3] M. R. D. Solomon and A. Ionescu, *Windows Internals Part 2*, 6th ed. Microsoft Press, 2012.

[4] FreeBSD. (2016, jan) 2.6. io system. [Online]. Available: https://www.freebsd.org/doc/enUS.ISO8859-1/books/design-44bsd/overview-io-system.html