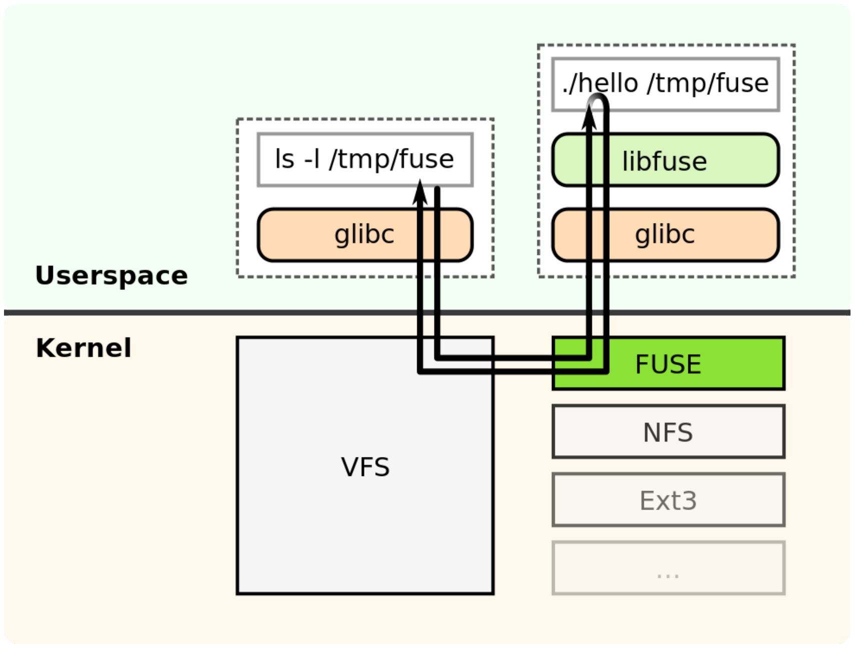
Filesystem in User Space (FUSE)

Filesystem in Userspace (**FUSE**) is a software interface for Unix-like computer operating systems that lets non-privileged users create their own file systems without editing kernel code. This is achieved by running file system code in user space while the FUSE module provides only a "bridge" to the actual kernel interfaces.

One of the difficulties in writing code in user-space is interacting with the kernel. In particular, a file system will likely have to interact with the kernel VFS (Virtual File System) at some point to access the hardware. So how does a user-space file system access the VFS?

For a long time, this was basically impossible. Any file system code had to live in kernel space (i.e. part of the kernel or a module) and that was the end of the story. But what do you do if, for example, you want to automatically encrypt/decrypt the data in a file system? Or if you want to compress/uncompress the data on a file system automatically (this is really deduplication) giving users the ability to see the data inside a tar file without untarring the data? Or perhaps you want to present the data in an SQL table as a directory with associated files? All of these things are great ideas but were unlikely to make it into the kernel. Ideally, these file systems should live in user-space but they still need to interact with the kernel.

Several developers decided they wanted to develop file systems or really, virtual file systems, such as those previously mentioned, in user space but needed some "help" from the kernel to get there. So they created something called FUSE (**F**ile System in **USE**rspace). The concept is to create a simple kernel module that interacts with the kernel, particularly the VFS, on behalf of non-privileged user applications and has an API that can be accessed from userspace. The figure below is the classic illustration of how this works.



The illustration corresponds to the “hello world” file system on the FUSE website. At a high level the “hello world” file system is compiled to create a binary called “hello”. This binary is executed in the upper right hand corner of the illustration with a file system mount point of /tmp/fuse. Then the user executes an ls -l command against the mount point (ls -l /tmp/fuse). This commands goes through *glibc* to the VFS. The VFS then goes to the FUSE module since the mount point corresponds to a FUSE based file system. The FUSE kernel module then goes through *glibc* and *libfuse* (libfuse is the FUSE library in user space) and contacts the actual file system binary (“hello”). The file system binary returns the results back down the stack to the FUSE kernel model, back through the VFS, and finally back to the ls –l command. There are some details I have glossed over of course, but the figure illustrates the general flow of operations and data.

Using the FUSE API you can pretty much write just about any type of file system you want with almost any features you want. Even better, you can use almost any language you want because there are many bindings between FUSE and other languages.

If you look around the web you will find various introductions to writing file systems using FUSE. Some of these are very simple and some are more complex. If you are interested in writing serious file systems with FUSE, I suggest you first understand file systems and their basic designs, before moving to FUSE. But at the same time, you can still write some very interesting FUSE file systems pretty easily.

There are many examples of file systems that use FUSE. Sometimes FUSE is used for prototyping or testing file systems or it is used as the file system itself. Some that you might recognize include:

- **SSHFS** This is a file system client that can mount and interact with directories and files on a remote system using sftp. Very handy file system for mounting remote file systems.

- **GmailFS** This FUSE based file system was written to use Google’s email storage as a file system. Originally it used the gmail web interface but this kept changing. The previous link takes you to a new version of GmailFS that uses IMAP to use the gmail email space as a file system. One of the interesting aspects of this file system is that it’s written in Python.

- **EncFS** This FUSE based file system provides an encrypted file system for Linux.

- **NTFS-3G** NTFS-3G gives you read/write access to a Windows NTFS file system. According to the website it works with Windows XP, Windows Server 2003, Windows 2000, Windows Vista, Windows Server 2008 and Windows 7 NTFS file systems.

- **archivemount** This file system allows you to mount archive files such as tar (.tar) or gzipped tar files (.tar.gz) to a mount point and interact with them including reading and writing. It’s a very cool way to check out the contents of a .tar.gz file before uncompressing and untarring it, especially if you only need one file from it. It also allows you to easily manipulate and create .tar.gz files.

- **ZFS-Fuse** This file system allows you to create, mount, use, and manage ZFS file systems under Linux. The licensing of ZFS is not compatible with GPL so interfacing ZFS with FUSE keeps ZFS as a user-space application which runs on Linux and doesn't violate any licensing. So if you want ZFS on Linux, this is your best option.

- **CloudStore** CloudStore is a distributed file system that is integrated with Hadoop and Hypertable.

- **MountableHDFS** There are several projects that allow you to mount a Hadoop file system (HDFS) and interact with it as you would a normal POSIX-style file system. For example, you can do an "ls" or a "cp" or a "mv" on HDFS using these FUSE based projects. This also means you can use POSIX conforming applications to read/write to HDFS without having to use the API.

- **GlusterFS** This is a high performance, distributed file system that uses a concept of "translators" that allow you to create file systems with various capabilities including mirroring and replication, striping, load-balancing, disk caching, read-ahead, write-behind, and self-healing. One of the strengths of GlusterFS is that it doesn't use metadata but rather relies on the knowledge of the file layout and the underlying file system.

- **MooseFS** MooseFS is a distributed fault-tolerant file system with several unique features: when you delete files MooseFS retains them for a period of time so they can be recovered; it also can create coherent snapshots of files even while the file is being accessed or written.

References

User Space File System. http://www.linux-mag.com/id/7814/

Filesystem in User Space. https://en.wikipedia.org/wiki/Filesystem_in_Userspace