COMPUTER SYSTEMS
PROJECT 6
Making a FUSE file system

1 User level file systems

Our goal for this assignment is to create our own *virtual file system (VFS)* using *FUSE*—a tool that allows us to run file system code as a regular program (a.k.a. *in userspace*). When a FUSE program is run, it creates a file system at a *mount point* provided by the user, and then makes its virtual file system appear within that mount point. All operations performed within the subdirectories and files within that mount point are passed, by the kernel, onto the FUSE program to handle.

There are a number of more arcane operations that a file system performs, and that a FUSE program must address, but for this assignment, we will focus only on the `READ` and `WRITE` operations, since their intent is clear and simple.

2 Stuff to do

2.1 Getting started

Login to `vega.cs.amherst.edu`,[1] make and change into a directory for this project, and then grab some starting source code.

```
$ wget -nv -i https://bit.ly/COSC-171-project-6-source
```

Before bothering to look at the code, we should first try using it to see how FUSE file systems behave. The code file `amhfs.c` is a complete FUSE program that provides a *passthrough file system*. That is, it mirrors one part of the file system—a *source directory* and all its contents—at another part of the file system—at the *mount point*. Every file seen in the source directory is visible at the identical (relative) location within the mount point, and vice versa.

Seeing it in action will likely be illuminating. To do so, first compile the code:

```
$ gcc -Wall amhfs.c `pkg-config fuse --cflags --libs` -o amhfs
```

Once compiled, make yourself a source directory and a mount point, and then start the FUSE file system:

```
$ mkdir source-dir mount-point
$ ./amhfs $cwd/source-dir $cwd/mount-point
```

You should get the prompt back with no errors or warnings. Note that `$cwd` is the environment (shell) variable that stores the *current working directory*. You can see it with the command:

```
$ echo $cwd
```

[1] This server is the only one configured for FUSE to work properly.

Notice that it provides the *absolute pathname* of the current directory—a single, complete name that begins from the root directory (the leading `/`). FUSE needs absolute pathnames, and so we provide the source directory and mount point in that form.

You can also check that the mount was successful:

```
$ mount | grep amhfs | grep sfkaplan
amhfs on /home/staff/sfkaplan/project-6/mount-point type
fuse.amhfs (rw,nosuid,nodev,user=sfkaplan)
```

Now that our file system is running, let's use it a little:

```
$ cd mount-point
$ printf "Some stuff\n" > foo.txt
$ cat foo.txt
Some stuff
$ cd ../source-dir
$ cat foo.txt
Some stuff
```

We made a small text file in the `mount-point` directory, and then verified that the file's contents are there. Then we switched to `source-dir` and, lo and behold, the same `foo.txt` file is there, and with the same contents! Try this trick with another file in reverse: make the file in `source-dir`, and then switch to `mount-point` and find it there.

Go back to your project directory (that is, move out of `mount-point` if you are in it), and then *unmount* our virtual file system like so:

```
$ fusermount -u $cwd/mount-point
```

You can then use the `mount` and `grep` commands again to verify that your file system is no longer listed.

2.2 Groking the code (at least a little)

Here you will see some semi-gnarly C code. It's not too bad, but it's not much commented, and so it's hard to follow if you don't know what you're looking at. What matters, for this assignment, is that each of the operations that a file system must support (e.g., `READ`, `WRITE`, `OPEN`. `CLOSE`) is handled by a corresponding FUSE function (e.g., `amh_read()`, `amh_write()`, `amh_open()`, `amh_close()`). You will see that most of this file is composed of these functions.

Search, specifically, for `amh_read()` and `amh_write()` (they are contiguous in the source code). You will find, in these functions, code where I have simply copied what is read and written, unchanged. You can see the code that opens the file in the source directory and then reads/writes exactly as requested into the FUSE-managed file.

3 Your assignment options

3.1 Option B: An encrypting VFS

***Encrypt* the data stored in the `amhfs` file system.** The encryption should be simple—something like a Caesar cipher. Thus, the bytes being written should be *encrypted* before they are stored in the source directory; likewise, the bytes read from the source directory should be *decrypted* before being placed in the buffer passed into the read function.

3.2 Option A: A versioning VFS

Normally, when a file is saved, the previous copy of the file is replaced by the newly updated copy. If a previous revision is desired, then either the user must manually make a copy, or use *version control software* (e.g., CVS, Subversion, git) to keep a history of previous versions. Meanwhile, *cloud storage services* (e.g., Google Drive, Dropbox), maintain some history of previous versions all on their own.

What we would like, however, is to have the preservation of multiple versions—a complete history of each file at each moment it was saved—performed by the computer on which we are working. Some systems have provided various forms of such *versioning file systems* (e.g., OpenVMS, LMFS). Therefore, our goal is to **create a virtual versioning file system** using FUSE.

Create a versioning file system that does the following:

1. No version of a file is lost. All versions are somehow represented in the storage directory. A *write* operation on a file adds a new version. Additionally, a *trunc* operation creates a new, shorter version of the file.

2. The files appear, in the mount-point, as normal files that, when *read*, yield the most recent version available.

3. A mechanism will be provided to read a specific version of a file, where the files are numbered from 1 to n (assuming n versions of the file have been created). Specifically, the 3^{rd} version of the file `foo.txt` could be read by specifying the file name `foo.txt,3`.

4 How to submit your work

Submit your new, enhanced `amhfs.c` file. You may use either of the following two methods to use the CS submission system:

- **Web-based:** Visit the submission system web page.
- **Command-line based:** Use the `cssubmit` command at the shell prompt on `vega.cs`.

This assignment is due on Wednesday, Dec-12, 11:59 pm.