Secure KeyValue style file system

Overview

When designing a "modern" file system, we care about performance of accessing from multiple clients as well as the security. In this project, we will implement a file system that "somewhat" fits the above goal using the fuse -- SKVFS (Secure KeyValue File System).

The SKVFS is different from conventional file systems in two ways:

First, to make the file system efficient for sharing among multiple clients, the file system uses a flat key-value structure as GoogleFS. There is no real directory supports. Every file, including directories, is translated into a 256-bit key. The file system maintains some data structure to map the key to a real location.

Second, to make the file system more secure, the file system uses MD5 to hash file names. As a result, anyone else cannot easily figure out what was the original filename, directory structure. Although MD5 can be brute-force attacked these days, it still take quite a while for an attacker to figure out.

You may use the VCL service maintained by NCSU through https://vcl.ncsu.edu/ . You may reserve one virtual machine and connect to this machine remotely by selecting reservations. We will use the "Ubuntu 14.04 Base" to test your kernel module. However, this virtual machine will reset once your reservation timeout. You should use https://github.ncsu.edu to control/maintain/backup your work.

You may work either alone or in a group of **3**. Groups do the same project as individuals. All members receive the same grade. Note that working in groups may or may not make the project easier, depending on how the group interactions work out. If collaboration issues arise, contact your instructor as soon as possible: flexibility in dealing with such issues decreases as the deadline approaches.

In this project, you will be given the prototype of the file system with a kvfs_functions.c file that only contains empty functions.

How to start

To begin, you need to make sure your linux install contains the following:

1. Fuse (2.9.7). You may download fuse 2.9.7 from https://github.com/libfuse/libfuse/releases/tag/fuse-2.9.7. Please following the instruction in their website and install the package into your linux install

2. OpenSSL library. We need this library for MD5 functions. In ubuntu, you can get this package installed by using the following command:

sudo apt-get install libssl-dev

Finally, you may use git to fork the project code from https://github.ncsu.edu/htseng3/KVFS and fetch it from your local repo. You need to navigate to the directory to the root of the repo on your local machine. Now, you should use the command "./configure" to generate the Makefile fits your machine setup. Then, you may navigate to the src subdirectory, and type "make". You will see the executable "kvfs" generated.

You can try use the kvfs to create a user space file system. For example, you can create an empty directory (highly recommended) under your home directory (e.g. /home/hungwei/kvfs_test), and then also create an empty directory under /mnt (e.g. /mnt/hungwei). You need to make sure that the directory that you created under /mnt direcotry gives you the full permissions (e.g. use chmod to change mode or chown to change ownership). Finally, you can "./kvfs /home/hungwei/kvfs_test /mnt/hungwei". If you haven't done anything to the project yet, you won't see this FS when you type "df" commands, although it's mounted. You need to "sudo umount" the mount point that you created to proceed your project.

Your task

In this project, you are responsible for completing the functions in the given kvfs_functions.c to support the required features that make the SKVFS file system work.

These functions are:

1. `int kvfs_open_impl(const char *path, struct fuse_file_info *fi)`
2. `int kvfs_read_impl(const char *path, char *buf, size_t size, off_t offset, struct fuse_file_info *fi);`
3. `int kvfs_write_impl(const char *path, const char *buf, size_t size, off_t offset, struct fuse_file_info *fi)`
4. `int kvfs_release_impl(const char *path, struct fuse_file_info *fi)`
5. `int kvfs_getattr_impl(const char *path, struct stat *statbuf)`
6. `int kvfs_readlink_impl(const char *path, char *link, size_t size)`
7. `int kvfs_mknod_impl(const char *path, mode_t mode, dev_t dev)`
8. `int kvfs_mkdir_impl(const char *path, mode_t mode)`
9. `int kvfs_unlink_impl(const char *path)`
10. `int kvfs_rmdir_impl(const char *path)`
11. `int kvfs_symlink_impl(const char *path, const char *link)`
12. `int kvfs_rename_impl(const char *path, const char *newpath)`
13. `int kvfs_link_impl(const char *path, const char *newpath)`
14. `int kvfs_chmod_impl(const char *path, mode_t mode)`
15. `int kvfs_chown_impl(const char *path, uid_t uid, gid_t gid)`
16. `int kvfs_truncate_impl(const char *path, off_t newsize)`

17. `int kvfs_utime_impl(const char *path, struct` utimbuf *ubuf)
18. `int kvfs_statfs_impl(const char *path, struct` statvfs *statv)
19. `int kvfs_fsync_impl(const char *path, int datasync, struct` fuse_file_info *fi)
20. `int kvfs_opendir_impl(const char *path, struct` fuse_file_info *fi)
21. `int kvfs_readdir_impl(const char *path, void *buf, fuse_fill_dir_t filler, off_t` offset, `struct` fuse_file_info *fi)
22. `int kvfs_releasedir_impl(const char *path, struct` fuse_file_info *fi)
23. `int kvfs_fsyncdir_impl(const char *path, int datasync, struct` fuse_file_info *fi)
24. `int kvfs_access_impl(const char *path, int` mask)
25. `int kvfs_ftruncate_impl(const char *path, off_t offset, struct` fuse_file_info *fi)
26. `int kvfs_fgetattr_impl(const char *path, struct stat *statbuf, struct fuse_file_info *fi)`

You can check out the comments inside the kvfs_functions.c file or go some online resource (e.g. Linux Assembly) for more information.

Turnins

In this project, you only need to turn in the kvfs_functions.c. Exactly 1 member of each group should submit the source code. All group members' names and Unity IDs should be easily found in a comment at the beginning of the code.