

Progress Report: Secure Isolated Copy-on-Write File System

Fareha Shafique
October 17, 2006

Introduction

The proposed project involved developing a copy-on-write user-level file system using FUSE. A copy-on-write (CoW) file system allows reads to execute normally, but creates a copy of the file in its own space when any modification is done. Thereafter, all reads and writes to this file are redirected to the new copy. In this way, the underlying file system remains unchanged by all the modifications and it is possible to either revert to it by discarding everything in the CoW file system or to merge the changes. This is very similar to 'One-way Isolation' and 'Alcatraz' by Liang *et. al.* [4,5], however, the implementation as well as the intended uses of this CoW file system are significantly different from those of Alcatraz.

As described above, the system will consist of a base layer and a copy-on-write layer. It is part of a larger project to provide mandatory access control for files and a controlled environment for replaying applications starting from a snapshot of a file system and reverting back to this snapshot at the end of the replay. It will also allow a user to commit changes to the base layer (which may itself have changed resulting in conflicts) according to a set of policies to be developed as a future project, and this can be used for Web management, for example, on-line photo galleries.

This report outlines the progress to date. Most work done thus far focused on understanding FUSE, which is the infrastructures for the CoW file system, and hence this report first provides a description of FUSE.

FUSE

Background

Filesystem in User Space (FUSE) is a kernel module that allows development of a fully functional file system that has a simple API library, can be accessed by non-privileged users, and provides a secure and stable implementation. FUSE provides an efficient userspace-kernel interface. Furthermore, file systems can be developed as executable binaries that are linked to the FUSE library and, therefore, a programmer does not need to understand in detail the file system internals or kernel module programming [1].

The first file system to be developed using FUSE, and for which FUSE was originally created, was AVFS (a virtual file system that allows all users to look inside archived or compressed files, or access remote files [2]). Several other file systems have been implemented now [3]:

1. Wayback: user-level versioning file system.
2. BTSlave: bit torrent file system.
3. LoggedFS: a logging file system through which every operation in a file system can be seen.
4. unionsfs-fuse: an implementation of Union file system using FUSE.
5. CryptoFS: an encryption file system (all files will be stored encrypted, both filename and data).

Overview

FUSE consists of a FUSE library and a kernel module that communicate via a special file descriptor.

1. FUSE library consists of the main function called by the user-level file system code. It is responsible for mounting the file system and setting up the environment, including obtaining a file handle for `/proc/fs/fuse/dev` (used to communicate between the two FUSE components) and

initializing the data structures. It also reads the file system calls from /proc/fs/fuse/dev, calls the appropriate user-level function and writes the result back to this file.

2. FUSE kernel module consists of the system calls filesystem component to deal with the file system calls by queuing them appropriately and the proc filesystem component that responds to file io requests to the /proc/fs/fuse/dev file.

FUSE allows a user (developer of a file system) to mount a directory onto the new file system. It then works by intercepting file system related system calls at the virtual file system (VFS) layer, and redirecting execution, with the appropriate arguments, to the user level code that implements the new file system. The response then follows the same path back as shown in Figure 1. Note that any file system call in the file system's user-level code will not be intercepted, but will execute as normal. Therefore, implementation of the proposed file system involves writing wrappers, for those calls that are supported by FUSE, to carry out the copy-on-write and isolation functionality.

Table 1 shows a list of system calls that are supported by the latest FUSE version (2.5.3).

getattr	access	readlink	readdir	mknod
mkdir	rmdir	symlink	unlink	link
rename	chmod	chown	truncate	utime
open	read	write	statfs	release
fsync	setxattr	getxattr	listxattr	removexattr

Table 1: File System calls supported by FUSE

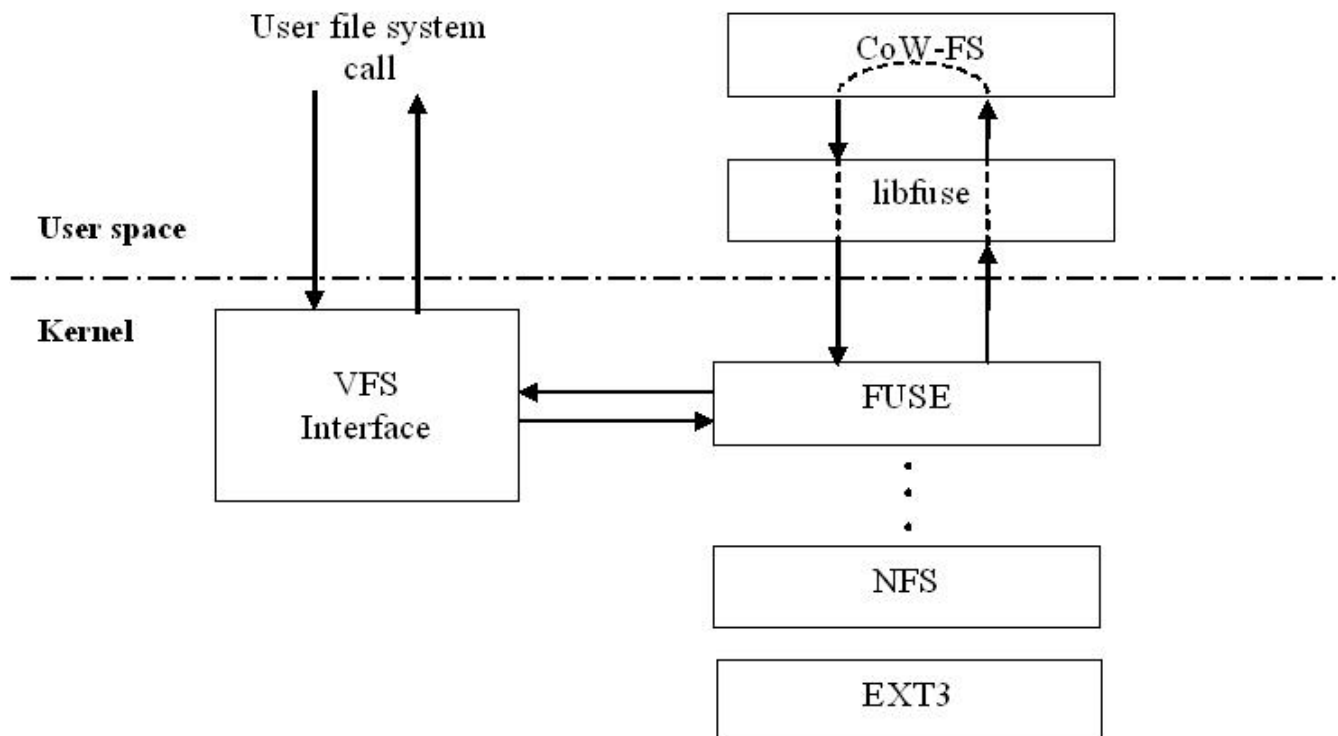


Figure 1 Path of a file system call for a file system mounted on FUSE and using CoWFS

Progress

I have so far read up on FUSE, worked with the examples provided in the package and worked in detail with Wayback (the versioning filesystem) to understand how to use FUSE for the purposes of developing the proposed Copy-on-Write file system.

I have also thought about how to implement some of the file system calls. The most challenging in my opinion is the open system call, since the lookup involved in opening a file is where all the redirection and copying of files will take place. I am currently implementing this system call, as well as designing the read, write, mknod, mkdir and rmdir system calls.

By the end of the project I hope to have implemented those system calls, that are shown in Table 1 as bold text and possibly provide an evaluation of the performance and effectiveness of the CoWFS in isolating all changes made to the mounted directory in a separate area.

References

- [1] <http://www-128.ibm.com/developerworks/linux/library/l-fuse/index.html>
- [2] <http://avfs.sourceforge.net>
- [3] <http://fuse.sourceforge.net>
- [4] W. Sun, Z. Liang, R. Sekar, V.N. Venkatakrisnan: One Way Isolation: An Effective Approach for Realizing Safe Execution Environments. In *Proceeding of Network and Ditsributed System Security Symposium, 2005*.
- [5] Z. Liang, V.N. Venkatakrisnan, R. Sekar: Isolated Program Execution: An Application Transparent Approach for Executing Untrusted Programs. In *Proceedings of the 19th Annual Computer Security Applications Conference, 2003*.