CSE-506 (Spring 2018) Homework Assignment #2
(100 points, 17% of your overall grade)
Version 1 (03/01/2018)
Due Sunday 4/1/2018 @ 11:59pm

* PURPOSE:

To become familiar with the VFS layer of Linux, and especially with extensible file systems APIs. To build a useful file system using stacking technologies.

You will use the "wrapfs" stackable file system as a starting point for this assignment. You will modify wrapfs to add "secure garbage file system" (sgfs) support.

* TEAMING:

For HW2, you must work alone.

Note that being the second homework assignment in the class, it is on purpose made a bit more open ended. Not every little detail would be given to you as in HW1. Your goal would be to follow the spec below, but to also think about how to address all sorts of corner cases that I do not specifically list below. You will have more freedom here to design a suitable system and implement it. As time goes by this semester, I expect all of you to demonstrate an increased level of maturity and expertise in this class. Be sure to document everything you do that's not explicitly stated in this document, in your README, and justify your decisions

* RESOURCES

For this assignment, the following resources would be handy. Study them well.

(a) The Linux kernel source code, obviously. Pay special attention to the Documentation/ subdirectory, esp. files related to locking, filesystems/*.txt, vfs.txt, and others. There's a lot of good stuff there, which you'd have to find and read carefully.

(b) The Wrapfs kernel sources in the hw2-USER git repository (or the group repository) that each of you have, under fs/wrapfs. Note also the file Documentation/filesystems/wrapfs.txt. This Wrapfs file system is under 2,000 LoC, and hence is easier to study in its entirety.

(c) Assorted papers related to stackable file systems which were published here:

http://www.fsl.cs.sunysb.edu/project-fist.html

Especially useful would be the following:

"Tracefs: A File System to Trace Them All"
"A Stackable File System Interface for Linux"
"Extending File Systems Using Stackable Templates"
"FiST: A Language for Stackable File Systems"
"On Incremental File System Development"
"UnionFS: User- and Community-oriented Development of a Unification Filesystem"
"Versatility and Unix Semantics in Namespace Unification"
"I3FS: An In-Kernel Integrity Checker and Intrusion Detection File System"

(d) Browsable GIT-Web sources here, especially wrapfs-4.6:

http://git.fsl.cs.sunysb.edu/

Look at source code for other file systems. The stackable file system eCryptfs may be helpful.

* INTRODUCTION:

In a stackable file system, each VFS-based object at the stackable file system (e.g., in Wrapfs) has a link to one other object on the lower file system (sometimes called the "hidden" object). We identify this symbolically as X->X' where "X" is an object at the upper layer, and X' is an object on the lower layer. This form of stacking is a single-layer linear stacking.

* DETAILS (KERNEL CODE):

In Unix, when you delete/unlink a file, it is deleted permanently. New OSs nowadays offer a "trash" or "garbage" icon, so when you delete a file, it's not deleted right away, but instead is moved to the trash. Later on, you can purge your deleted files manually.

SGFS is intended to create such a secure garbage folder. When a user deletes a file, the file should instead be moved to a special "hidden" folder called ".sg/" at the top of the mount point.

When a file is moved, it should be renamed so you know when it was moved, for example a file "foo.txt" should be named "2018-03-01-20:07-foo.txt". This would also permit you to handle duplicate file names being deleted.

Because there is only one .sg folder for all users, it is important to securely protect files. Therefore, you should support a mount-time option "enc=KEY" to encrypt all files in the .sg folder, with key "KEY". (Yes, one key for all files; see EC below.). Files encrypted should further have an extension ".enc". So a deleted and encrypted file foo.txt would be named "2018-03-01-20:07-foo.txt.enc".

Users should be able to permanently delete files from the .sg folder. If a user performs an unlink on a file inside .sg, then permanently delete the file ONLY if the user owns the file.

To protect the .sg folder, no user other than root should be able to fully list the contents of the directory (readdir op). Moreover, if a user tries to lookup a file in .sg, you should return ENOENT unless the user owns the file. Users, however, should be able to list/view their own files in the secure garbage. (Hint: it might help to add the UID into the filename when it's moved into the .sg folder.)

Users may want to recover/restore a file deleted by mistake. Support an ioctl(2) to "undelete" a file. If the user owns the file, then you should move/copy the file to the user's cwd, decrypt it back, and rename it back to its original "foo.txt" name. Note: the order of ops (move, decrypt, rename) doesn't matter, but do it efficiently and think about partial failures and how to recover from them (e.g., what if copying/decrypting fails half-way?)

To implement sgfs, you should make a copy of the fs/wrapfs/* code; don't edit wrapfs source files, so you can refer to them later on (wrapfs is a fully functional stackable f/s). So make a copy of all wrapfs sources to fs/sgfs, and edit them there; you'll have to rename symbols with 'wrapfs' in them, as well as the f/s name, to 'sgfs' (some symbols are upper/mixed case, so be careful when you change them). Remember to git add/commit/push any new source files.

Mounting sgfs can be done as follows:

```
# mount -t sgfs /some/lower/path /mnt/sgfs
```

or

```
# mount -t sgfs -o key=MySecretPa55 /some/lower/path /mnt/sgfs

(Note that you may need to first run "insmod ./fs/sgfs/sgfs.ko" to
insert the new sgfs module; of course, you'll need to remove an older one
that's already loaded, if any.)

After mounting, you should be able to "cd" to /mnt/sgfs and issue normal
file system commands.  Those commands will cause the VFS to call methods in
sgfs.  You can stick printk lines in sgfs to see what gets called and
when.  Your actions in /mnt/sgfs will invoke sgfs methods, which you
will intercept, and write to the tfile.

* DETAILS (USER CODE):

You will need to write some user level code to issue ioctls to sgfs: to
undelete a file.  Usage of the program is:

$ ./sgctl [-u] FILE

FILE: the file's name to undelete
-u: option to "undelete"

* GIT REPOSITORY

For this assignment, we've created clean GIT kernel repositories for each of
you.  Do not use the one from HW1 for this second assignment; but you can
copy a good .config you've had in HW1 and use that for your HW2 kernel.  You
can clone your new GIT repo as follows, using similar instructions as in
HW1:

# git clone ssh://USER@scm.cs.stonybrook.edu:130/scm/cse506git-s18/hw2-USER

Note that if you don't have enough disk space on your VM, it'll be a good
idea to remove the older hw1-USER repo to make space.

If you want, you can use my kernel config as a starting point, and adapt it
as needed:

http://www3.cs.stonybrook.edu/~ezk/cse506-s18/cse506-s18-kernel.config

* SUBMISSION

Simply git-commit and git-push your changes to your cloned git repository; a
successful git-push will result in an email going to you, confirming the
commit and push.  Don't forget to include the README.HW2 file.  If for some
reason you decided to add other file(s) to your GIT repository, please
mention this in README.HW2 so we don't miss it during grading (and justify
why a new file was needed).

All new files should be added to the hw2-USER/hw2/ subdir in your repo
(e.g., user-land code, additional Makefile, etc.).

Your README.HW2 should detail your design for sgfs, the ioctl, mounting
code, user-level code, anything special/different you did, extra credit
design, etc.

Also note that we will just do a git clone of your final repository and run
make, make modules_install, and make install as usual.  You must not assume
that we will do ANY modification in your code.  Your code MUST compile and
run as it is.  You will lose all points in submission section IF your code
doesn't compile/run as checked out.

If you attempt any EXTRA CREDIT functionality, your README.HW2 MUST specify
exactly how to compile your code with extra credit enabled.  By default,
```

```
your code MUST NOT compile extra credit code.

* EXTRA CREDIT (OPTIONAL, MAX 20 pts, plus bug fixes)

If you do any of the extra credit work, then your EC code must be wrapped in

        #ifdef EXTRA_CREDIT
                // EC code here
        #else
                // base assignment code here
        #endif

A. [10 pts] per user keys

Instead of having one crypto key for all users, support per user keys.  You
will need to add several new ioctls (to be listed by non-root user):

- to list a user's key, if there is one
- to add a new user key, otherwise can't encrypt files
- to remove a user's key, if there's one
- to replace one key with another
- for root user only: to list all keys of all users

Then, each user's files being moved to .sg should use that user's crypto
key, if there is one.  If there's no key, consider it an error (and there's
no need to have a key=KEY mount option).

B. [10 pts] background .sg cleaning

Implement a kernel thread that wakes up periodically, and deletes files in
.sg that are older than time T.  Note: there's a lot of existing support in
Linux for ktreads, so just use what's already there.  Be careful to lock any
needed resource carefully, so the kthread and user un/delete actions don't
cause races/deadlocks.

Support a mount time option "maxage=T" where T is in seconds.  So for
example, if you mount sgfs with maxage=60, then files .sg who were moved to
.sg more than 60 seconds ago are permanently deleted.

C. [up to 10 pts] wrapfs bug fixes (optional)

If you find and fix true bugs in wrapfs in 4.6, you may receive special
extra credit, depending on the severity of the bug and the quality of the
fix.

Good luck.

* Copyright Statement

(c) 2018 Erez Zadok
(c) Stony Brook University

DO NOT POST ANY PART OF THIS ASSIGNMENT OR MATERIALS FROM THIS COURSE ONLINE
IN ANY PUBLIC FORUM, WEB SITE, BLOG, ETC.  DO NOT POST ANY OF YOUR CODE,
SOLUTIONS, NOTES, ETC.

* ChangeLog: a list of changes that this description had

v1: original version
```