Operating Systems (CS-323): Assignment #4
Simple Versioning File System

NSL

1 Objectives

1. Learn basics of VFS APIs
2. Getting familiar with FUSE (the Filesystem in USErspace) by implementing a Simple Versioning File system (SVFS)

2 Introduction

In this assignment, you will have to develop a Simple Versioning File System (SVFS). SVFS will keep backups of every changed file for the next N (say 10) minutes. Hence, all files that are saved with unwanted changes, or accidentally deleted for instance, can be easily retrieved.

You are strongly encouraged to use The FUSE (Filesystem in USErspace) for this assignment. There exist as many file system APIs (the so-called VFS API) as there are different operating system kernels. The FUSE API is another file system API, however it allows the development and execution of file system drivers in userland instead. The advantage of this approach is that less code is run in the kernel, which reduces the risk of crashes or exploits. Since FUSE presents a simple, abstracted file system interface, it is easily possible to use FUSE file system applications on different operating systems, such as Linux, FreeBSD, Mac OS, OpenSolaris or even Windows. Running in userland also enables the use of various language bindings, allowing the implementation of FUSE file systems in different programming languages such as C, C++, Java, C#, Python, Perl, OCaml, etc.

3 Assignment

Your goal is to write the SVFS. SVFS easily retrieves files that are saved with unwanted changes, or accidentally deleted by simply rewriting the current file with the previously created backup. Your implementation should not

focus on the standard file system operations that store/retrieve information to/from disks, but on the versioning part - creating, maintaining and deleting backups. Indeed, FUSE allows you to use existing filesystem operations and we provide the skeleton implementation on the Moodle page that represents a simple files system, without any backup functionality.

Your implementation should focus on 2 essential operations:

1. Creating backup files. Each time a file is about to be changed, you should create a backup copy of the file, with a suffix of the file (e.g. filename.BACKUP.1). Hence, the file operations such as $open$ with write access rights, $trancate$, etc. should be modified.

2. Deleting backup files. You have to implement the "garbage collector" that will delete backup copies of a file, N minutes after the last change is performed on the file. Further, all backup files should be deleted after the user unmounts the file system.

You are allowed to make simplifying assumptions as follows:

1. You can consider that a single file system instance runs at a time (the implementation is easier because you can use global variables); otherwise, it becomes more complicated because you would need to define a context per mount point.

2. The number of backup files is bounded by a constant, otherwise you would need to implement a linked list to handle an unknown number of backups.

Finally, you should make the parameter N dependent on number of writes performed on a file. Intuitively, heavily modified files should have a backup copy of the file for a longer time than files that are subject to only one or two write operations. If N is kept constant, we will consider this as a minor lack of functionality and deduct 10 points.

You can make use of the skeleton implementation provided on the Moodle page, but you are not required to use it. You are also allowed to use a programming language different than C, but you *must* make sure that we can compile and run your submission (see below).

4 Running and testing

Development environment

You are encouraged to use your own computer to develop and test your implementation. If you do not have a means to perform development, you can use the machine `TBA.epfl.ch`, with your GASPAR user and password. Your `icfiler` home directory will be available to you.

If you want to develop in a programming language different than C, make sure that your submission can be compiled and run on `TBA.epfl.ch`. If you think the system is missing features for your language of choice, please contact us via Moodle well before the submission deadline.

Mounting

The supplied skeleton already performs basic file system operations (without backup functionality), using the existing file system. For this purpose, we pass another argument to SVFS (called *rootdir*). Essentially, whenever a file operations is called on the *mountpoint*, the FUSE callback will read/write from the root directory using standard linux fs operations. Hence, to mount the file system, run the application with the root directory and destination mount point as arguments. However, before mounting, you might need to install FUSE-related libraries:

```
% //first, install FUSE
% apt-get install fuse-utils libfuse-dev libfuse2
% //then, perform mounting as explained above
% //rootdir and mountdir are already created for you
% svfs -s example/rootdir example/mountdir
% ls -l example/mountdir
```

The argument `-s` specifies single-threaded operation. Optionally, `-f` instructs FUSE to stay in the foreground and not to place itself in the background. These settings are useful for debugging.

Now if you cd to example/mountdir and do your operations there it will use the SVFS callbacks.

```
% cd example/mountdir
% echo 'aaa' > foo
% echo 'bbbbbbbbb' > foo
% ls -la
% -rw-r--r-- 1 sana sana   10 2012-05-02 08:10 foo
% -rw-r--r-- 1 sana sana    4 2012-05-02 08:10 foo.backup.1
% cat foo.backup.1
% aaa
% cat foo
% bbbbbbbbb
```

After N seconds the backup files should disappear:

```
% ls -la
% -rw-r--r-- 1 sana sana   10 2011-05-02 08:10 foo
```

For further tests, consider using commands such as *cat*, *cp*, and shell operators like | and >>. For example:

```
% ls -la | wc -l | cat > bar
% echo 'qwerty' >> foo
```

Unmounting

In case your file system does not shut down properly, you might experience an error message, such as `ls: cannot access dest: Transport endpoint is not connected`. In this case you will have to unmount the file system manually:

```
% fusermount -u -z example/mountdir
```

5 References

FUSE http://fuse.sourceforge.net/

FUSE file system operations http://fuse.sourceforge.net/doxygen/structfuse__operations.html

6 Deliverables

Deliverables for this project are:

- Short project report about FUSE, SVFS and your experience
- Full source code for the assignment