StockFS: A Pseudo-Filesystem for Stock Market Quotes

Professor: Paul Krzyzanowski
TA: William Katsak
TA: Nader Boushehrinejadmoradi

April 18, 2012

1 Introduction

In this project you will implement a pseudo-filesystem to provide stock market quotes, using FUSE (File System In Userspace), and UNIX sockets. This project will give you practice implementing a rudimentary FUSE filesystem, as well as using UNIX TCP sockets to connect to an HTTP server.

2 Description

You will implement a pseudo-filesystem that provides an easy interface to obtain stock market quotes without additional application programs. This file system must be implemented using FUSE and HTTP over Linux sockets.

2.1 Background

In the stock market, each publicly traded company has a "ticker symbol", a short sequence of letters that act as a shorthand name for the company. For example, Apple, Inc.'s ticker symbol is "AAPL", and Google Inc.'s ticker symbol is "GOOG".

2.2 Requirements

- The "files" in your pseudo file system will be named after a particular ticket symbol, and a file will contain the current price information for a particular stock.
- To describe the operation of the file system, let us assume that it is "mounted" at the path `/stockfs`.
- Your file system, when initially mounted, will present an empty directory to the user. For example, `ls /stockfs` will not show any files. However, if you try to obtain the contents of an arbitrary "file", for example `cat /stockfs/file`, the file system will take the file name ("file") to be a stock ticker symbol, contact a web service, obtain stock price information for this stock, and provide this data as the contents of the file.
- Additionally, the file system should support "bookmarks" of favorite stocks in the following manner: If the user creates a new file, for example: `touch /stockfs/aapl`, this file will then appear in a directory listing (e.g. a subsequent `ls /stockfs` will show this file). Access to the file will behave in the same manner as before.
- The following information is required for each stock, in the following format. Data is just for example. All of this information can be obtained by figuring out the proper f= argument to the web service URL. See the main discussion below.
  - Symbol: XXXX
  - Company Name: XXXXXXXX, Inc.
  - Last Trade: 999.99
  - Change: -99.99
  - Bid: 95.99

- Ask: 97.99
- Bid Size: 5
- Ask Size: 100

3 Implementation

Your file system should be implemented in the context of a FUSE module. FUSE modules have a very simple `main()` function, and are made up mostly of the implementations of the various possible FS operations. The operations that you must implement, at a minimum, are: `open()`, `release()`, `read()`, `getattr()`, `readdir()`, and `utime()`.

3.1 Control Flow

When a user executes your program, they will provide a mount point as a parameter, e.g. `stockfs ./mnt`. The program will immediately terminate, and the file system will be mounted on the directory that was specified. From this moment, the only entry points into your code will be via performing operations on the mounted file system. For example, if you try to open the file /mnt/aapl, your open() function will be called with parameters matching this file name.

3.2 Suggested Function Implementations

3.2.1 getattr()

`getattr()` will likely be the first function that gets called when a user tries to read a file. The OS will call this file, to see what the size and other attributes are. You must support this function for any potential ticker, but the size doesn't have to be correct (but does need to be larger than zero). You should report some fixed size like 4 kilobytes.

3.2.2 readdir()

`readdir()` must report the files ., .., and any files that the user has "touched".

3.2.3 open()

`open()` is where you must open a socket connection to the web service and pull the data for the stock. If the stock ticker turns out to be invalid, you must refuse to open the file, and report an error. Since `read()` will be called asynchronously, later, you must maintain a table (or linked list, etc.) of all currently open files and save the stock information that you have read as a string associated with this file; this string is what `read()` will provide to the user later.

3.2.4 release()

`release()` should destroy the information that you have saved for a file, and return.

3.2.5 read()

`read()` must read the stock information that you have obtained for a file in `open()` into the user's buffer. This is a straightforward process.

3.2.6 utime()

`utime()` should create an entry in a list for that file. `readdir()` should return any symbols in this list, so that the user can see them with ls. The reason that we use `utime()` is that touch opens a file for writing, and then updates its timestamp. We don't care what the timestamp is, but simply use the call to `utime()` as a marker that a file has been "touched". This replaces the call to `create()` or `mknod()` in previous versions of this text. Even though `open()` is called first, you will still need to perform validation here to make sure that the symbol is valid before you actually create the bookmark.

3.3 Thread-safe code

FUSE is multithreaded, so you must assume that there can be multiple calls to any function at any time. For this reason, you must synchronize access to any common lists or data structures.

3.4 Getting Stock Market Data

3.4.1 Yahoo Finance

You will be using the Yahoo Finance API. This API can be tested in your web browser by accessing the following URL:

```
http://download.finance.yahoo.com/d/quotes.csv?s=aapl&f=snl1
```

In this URL, the s parameter contains the symbol we are interested in, and f parameter represents which data should be returned. Of course, in this example, aapl is the symbol for Apple, Inc. and should be replace by the ticker of the company that you are interested in.

3.4.2 HTTP Client

This service is very easy to access in a web browser, but we must obtain this data inside our code. In order to do this, you must implement a VERY simply HTTP client function. A basic HTTP session involves opening a socket to a host on a TCP port (by default, port 80), and sending a HTTP request header. For example, to get information from the Yahoo Finance service, you must open a TCP socket to download.finance.yahoo.com on port 80, and send this string:

```
"GET /d/quotes.csv?s=aapl&f=snl1 HTTP/1.0\n\n''
```

After sending this string, you should immediately start reading the response, which will look something like this:

```
HTTP/1.1 200 OK
Date:  Fri, 06 Apr 2012 17:46:00 GMT
P3P: policyref="http://p3p.yahoo.com/w3c/p3p.xml", CP="CAO DSP COR CUR ADM DEV TAI PSA PSD IVAi IVDi
CONi TELo OTPi OUR DELi SAMi OTRi UNRi PUBi IND PHY ONL UNI PUR FIN COM NAV INT DEM CNT STA POL HEA
PRE GOV"
Cache-Control:  private
Connection:  close
Content-Type:  application/octet-stream

"AAPL","Apple Inc.",633.68
```

Hints:

- The final line contains the string that you must parse in order to get the stock information. The three values there correspond to the arguments to the f= parameter in the URL. Note that you can just search for two newlines in order to locate this last, most important line.
- Every line returned from the HTTP server is terminated not with just a newline but with a carriage return ('\r') followed by a newline. When parsing your input, ignore all carriage return characters.
- Prior to even touching any aspect of FUSE, make sure that you can write a program to connect to the server and get the data you need. Try it first with a command line using wget or curl. Then, use client.c in demo-03, the last downloadable demo in the socket programming demo as a basis to connect to the server.
- If you're more comfortable with fputs, fgets, and fprintf, realize that you cannot use those on a file descriptor (socket) directly. You'll have to "reopen" the file/socket within the stdio library via FILE *fp = fdopen(fd, "w+"); Be sure to check for errors, of course.

4 Restrictions

This project must be done with FUSE and Sockets, on Linux. We will grade on the iLab machines, so this is where you should do your development and testing.

5 Resources

- Yahoo Finance API Reference: `http://www.gummy-stuff.org/Yahoo-data.htm`
- Sockets Example: `http://www.cs.rutgers.edu/~pxk/rutgers/notes/sockets/index.html`
- FUSE Website: `http://fuse.sourceforge.net`

6 Compile and Run Instructions

6.1 Special Note

The very first line of your program must be:

```
#define FUSE_USE_VERSION 26
```

6.2 Compiling

To compile your FUSE module, use the following command line:

```
gcc sourcefile.c -o stockfs -lfuse -D_FILE_OFFSET_BITS=64
```

6.3 Running

To mount your FUSE file system, use the following command line:

```
./stockfs some_empty_directory_for_mount_point
```