DFS: Part 3: Reading, Writing

Due: 11:59PM Saturday, December 11, 2013

1 Introduction

In this lab, you will continue to add functionality to YFS supporting SETATTR, OPEN, WRITE and READ operations.

2 Getting started

Add the testscript in handout to your existing project directory. Make sure everything compiles before you change any code.

When you're done with Part 3, you should be able to read and write files in your file system. For example, use the start.sh script to start up two separate yfs_clients with mountpoints at "./yfs1" and "./yfs2", which both use the same extent server for organizing its data:

```
% ./start.sh
```

Then, you can write files to one directory and read them from the other:

```
% ls -l yfs1 yfs2
yfs1:
total 0

yfs2:
total 0
% echo "Hello world" > yfs1/hello.txt
% ls -l yfs1 yfs2
yfs1:
total 0
-rw-rw-rw- 1 root root 12 Sep  6 20:14 hello.txt

yfs2:
total 0
-rw-rw-rw- 1 root root 12 Sep  6 20:14 hello.txt
% cat yfs2/hello.txt
Hello world
```

Afterwards, be sure to stop your all the processes:

```
% ./stop.sh
```

If you try the above commands without implementing the required SETATTR, WRITE and READ operations, you will get an error.

3 Your Job

Your job is to implement SETATTR, OPEN, WRITE, and READ FUSE operations in fuse.cc. You must store the file system's contents in the extent server, so that you can share one file system among multiple yfs_clients. If your server passes the tester test-lab-3.pl, then you are done. If you have questions about whether you have to implement specific pieces of FUSE functionality, then you should be guided by the tester: if you can pass the tests without implementing something, then don't bother implementing it. Please don't modify the test program or the RPC library (beyond your Part 1 changes) in any way; we will be using our own versions of these files during grading.

The test-lab-3.pl script tests reading, writing, and appending to files, as well as testing whether files written on one yfs_client instance can be read through a second one. To run the tester, first start two yfs_clients using the start.sh script. It runs two yfs_client processes each mounting the same file system under a different name (yfs1 or yfs2).

```
% ./start.sh
```

Now run test-lab-3.pl by passing the yfs1 and yfs2 mountpoints. Since the script tests each yfs_client sequentially, we do not need to worry about locking for this lab.

```
% ./test-lab-3.pl ./yfs1 ./yfs2
Write and read one file: OK
Write and read a second file: OK
Overwrite an existing file: OK
Append to an existing file: OK
Write into the middle of an existing file: OK
Check that one cannot open non-existant file: OK
Check directory listing: OK
Read files via second server: OK
Check directory listing on second server: OK
Passed all tests
% ./stop.sh
```

If test-lab-3.pl exits without printing "Passed all tests!" or hangs indefinitely, then something is wrong with your file server.

4 Detailed Guidance

Begin by uncommenting the relevant lines at the bottom of fuse.cc::main (such as fuseserver_oper.open = fuseserver_open;) so that you point FUSE to call the appropriate functions that you will fill in. From there:

- Implementing SETATTR The operating system can tell your file system to set one or more attributes via the FUSE SETATTR operation. As always, see the FUSE lowlevel header file for the necessary function specifications. The to_set argument to your SETATTR handler is a mask that informs the method which attributes should be set. There is really only one attribute (the file size attribute) you need to pay attention to (but feel free to implement the others if you like), which has the corresponding bitmask, FUSE_SET_ATTR_SIZE. We have already done an AND (i.e., &) of the to_set mask value with an attribute's bitmask to see if the attribute is to be set. The new value for the attribute to be set is in the attr parameter passed to your SETATTR handler.

  Note that setting the size attribute of a file can correspond to truncating it completely to zero bytes, truncating it to a subset of its current length, or even padding bytes on to the file to make it bigger. Your system should handle all these cases correctly.

- Implementing OPEN/READ/WRITE: You are free to store the file contents however you like with the extent_server. You may, for example, treat a file's contents as a std::string. (If so, how would you support very large files? We would not test your FS for large files, but it's a good design exercise.) You will need to extend your yfs_client, yfs_server, and yfs_protocol to handle more types of messages than just the put/get/getaddr/remove commands you've already implemented.

  READ/WRITE operations are straightforward. A non-obvious situation may arise if the client tries to write at a file offset that's past the current end of the file. Linux expects the file system to return for any reads of unwritten bytes in these "holes". See the manpage for lseek(2) for details. This will be tested for, so make sure you handle this.

  OPEN should simply check if the file exists and return fuse_reply_open(req, fi), or an error otherwise. It need not fill in any of the parameters in the fuse_file_info struct.

5 Handin

Please submit all the files necessary for running Part 3, including the Makefile to /tags/pa4/part3.

6 C++ Tutorials and Resources

- C++ Tutorial
  `http://www.cplusplus.com/doc/tutorial/`

- C++ Reference
  `http://www.cppreference.com/wiki/start`

7 Grading

20% Code quality, style

80% Code functionality, robustness, scalability