Denny Sabu

John Sweeney

Michael Parrilla

Assignment 3

Networked File Server

Creating the project:

As a starting point we began by referencing the hello.c file that was on the libfuse GitHub. This was useful as it allowed us to begin setting up a framework for the remainder of our project. We began by removing all of the code within the functions in hello.c and replacing them with simple print statements. The point of this was to see what was commands and actions called what functions. Once we had a general idea of what they were doing we began work on the actual project.

Then we setup a ‘skeleton’ version of the client and the server. The client began with basic functions, like open and read but were not functional. The server was able to listen on a port and accept incoming connections and spawn threads for each individual user. We also agreed on a protocol that all communication between the server follow. The server was able to perform different actions based on this protocol.

Once the framework was set up, we began to implement the actual functionality for the file system. We quickly noticed that getattr() was being called before and after every call, so we began with this. Initially we had planned to split the work and tackle multiple functions concurrently but realized that without getattr() we could not do much. In addition, we ran into massive problems finding documentation for FUSE. Eventually we were able to piece together a rough understanding of what the function needed. Once we were able to get getattr() done we were able to move onto the other functions.

Functions implemented in clientSNFS.c and serverSNFS.cpp

- getattr()
- readdir()
- opendir()
- open()
- read()
- mkdir()
- rmdir()
- write()
- create()
- release()
- unlink()

Control Flow:

The server side of our project is handled from the serverSNFS.cpp file, in which its main function takes input from the command line to set the port and mount directory path. After the port and mount directory have been set main calls the serverStart function to accept socket addresses and create socket file descriptors. New sockets are created via threads from which the whatToDo function is called. It is from the whatToDo function that all other functions are called including get attribute, create, open, read, write, close, truncate, read directory, and unlink.

The client file, titled clientSNFS.c, is what must be run for FUSE to be able to intercept system calls. The clientSNFS executable takes four command line arguments. The first is the port number the client will attempt to connect to. The next is the address of the machine that is hosting the server. After that, there is the local directory that the SNFS will be mounted to and lastly the "**-f**" flag keeps the client running in the foreground. Now with this client running, the user can open up another terminal on the same machine that is running the client program, navigate to the mounted folder and begin to interact with the server on the remote machine as if it was locally available.

How to run:

Note: *We recommend **51223** as a port number*

1) Extract the tar
2) Open two terminals, on two separate iLab Machines
3) On the one that you would like to be your server run "**mkdir /tmp/1436**"
4) On the one that you would like to be your client run "**mkdir /tmp/1436**"
5) Now, on one of the terminals, move into the directory from the extracted tar
6) Invoke the makefile, by running "**make**"
7) Now, on your server terminal type and run "**./serverSNFS [port_num] *tmp/1436***"
    a) For example: ***./serverSNFS 51223 /tmp/1436***
    b) Port_num argument must match on client and server command line arguments
8) Now, on your client terminal type and run "**./clientSNFS [port_num] [iLab_machine] /tmp/1436 -f**"
    a) For example: ***./serverSNFS 51223 /tmp/1436***
    b) Port_num argument must match on client and server command line arguments
9) Both terminals should now be running their respective programs.
10) Now, open a third terminal that is on the same machine the client is running on (this will be the testing terminal)
11) On this terminal, run "**cd /tmp/1436**"
12) You can type "ls" and nothing should appear.

Testing:

Note: *To see what functions are being called on the client and server, you may keep an eye on the terminals where they are running. They will print their status as they are called.*

`create, open, getattr, ls:`

1) To test create and open, from the /tmp/1436 directory, run "touch f1.txt"
    a) Warning: You will see a message pop up about setting utimes, you may disregard this.
2) Run "ls" again, f1.txt will now appear.
    a) f1.txt will now be an empty file.

`read, write, close:`

1) From the earlier step, we now have an empty file f1.txt.
2) To test write, run "**echo This is a test for write** >> **f1.txt**"
    a) This will test getattr(), open(), and write() and close().
3) Now, lets test read, run "**cat f1.txt**"
    a) You should see "This is a test for write" appear on the screen.
4) Now run "**echo Hello Zigzag marble** >> **f2.txt**"
5) Run "**ls**" and you will see f1.txt and f2.txt

`opendir, readdir, rmdir, mkdir:`

1) Now, lets test directories. Run "**mkdir dir1**"
    a) If you run "**ls**", you will now see f1.txt, f2.txt AND dir1.
2) Feel free to run "**ls dir1/**" to verify an empty directory has been created.
3) You can create another directory inside of dir1, run "**mkdir dir1/dir12**"
    a) Run "**ls dir1/**" to verify a new directory has been created.
4) You can now move into this new directory using "**cd dir1/**"
5) To test rmdir, you may run "**rmdir dir12**"
    a) Type "**ls**" to confirm dir12 has been removed.
    b) Run "**cd ..**" to move back to the home directory
    c) You should see f1.txt, f2.txt AND dir1 when you run "**ls"** here

Truncate:

1) Lets test the truncate function. From the **/tmp/1436** directory, run “**truncate -s 8 f2.txt**”
    a) Now run “**cat f2.txt**”
    b) You should see on the screen “Hello Zi” :)

Remove, unlink:

1) We no longer want f1.txt, lets, remove it. Run “**rm f1.txt**”
    a) Now run “**ls**”
    b) You will now see f2.txt and dir1/

Extra:

Due to implementing these functions extremely successfully, the use of Nano is supported on the SNFS. Type “**nano newFile.txt**”, when you do this, a Nano GUI should appear. Type a short sentence. You can now save and close the file. Run “**cat newFile.txt**” you should see your short sentence appear.

To verify that this is actually a networked file server:

1) On the testing terminal that you were running commands on, run ”**cd ..**” to exit it.
    a) You should now be in the /tmp directory
2) Change to the terminal running the clientSNFS program. Use CTRL + C to terminate the program.
3) Change to the terminal running the serverSNFS program. Use CTRL + C to terminate the program.
4) On the terminal that was running the server, run “**cd /tmp/1436**” and run “**ls**”.
    a) You will see all the files you made.
5) Now, go back to the testing terminal, from the /tmp directory run “**cd 1436/**”
    a) Run “**ls**”, none of the files will be present.