Project 3 - Simple Network File System

Your report should at least describe 1) the work you've done, 2) the difficulties you faced, and 3) all your working operation examples of your file system. Please also include the responsibilities of the members of the group.

Samyak Gupta, Daniel Pattathil, Michael Wu, Katie Dickerson

The primary goal of this project was to create a file system that works on a client-server model in which the client, or multiple clients, makes calls in a local process that packages system calls as requests to be sent to the server. The project is laid into the three major parts which are the server, the client (with FUSE), and the Remote Procedure Calls (RPC).

Server:

Our server application is meant to perform as a local file system dependent on calls made from the client(s). The information shared between the server and client is completed through packages of information after a connection is made between the different sockets. By predetermining ports, we verify that the connection will be made in order for send and receive communication between each client and the server.

The server accepts client connections until there is a timeout, and then, after all the requests are processed (all the threads have completed), the server will close it's own socket. The assumption in the processing of the threads is that requests should not interfere with each other, and as such, we do not want to inefficiently set locks for read or write. We are designing the server in a way that it can receive multiple clients' requests by setting calls to be addressed through a "connection_handler" method which addresses each client call at a time. Our server will receive "messages" which should follow our SFNS.h structs to hold the enum for the particular procedure (1 - CREATE, 2 - OPEN, etc.). Then, within the "connection_handler" method, the particular message is sent to a procedure-specific method in order to send the correct response. Those procedure-specific methods would essentially run the equivalent procedures on the server's local directory and return the output for the client(s) to display.

Client:

Our client application is meant to allow for calls to be packaged and sent to the server application. The client application will simulate the file system as if the server's directory is what the actions are being performed on. This is done using the FUSE tool, which allows unique implementation of user-level file system procedures. For our client, the actual file system calls are running normally for the most part, simply some calls such as open() or write() have been modified so that they would be sent as calls to the server to execute and wait for a response rather than attempt to perform them locally. By using FUSE and making sure to set a mount

point, the calls are primarily dealt with on the server end, because FUSE is able to redirect the particular calls for the specific directory that was mounted.

The actual functionality of the client is to first set up the socket and attempt to connect to the server credentials provided when launching the program. Afterwards, the program will allow for the input of file system commands that should normally work on a local level. This lets the general functionality work for each test command, and only intercepts for situations where the mounted folder is accessed.

This is the place in our project where we set up our procedure specific methods to send the request to the server for each of create(), open(), write(), etc. Each of these methods will take in parameters surrounding the request, send them in a packaged struct, and then, receive the required output from the server in order to display.

RPC:

Remote Procedure Calls are to be used for communication between the server and the client. We have the specifications of ours in SNFS.h, where we have specified rpcCall and rpcRecv structs. Inside of the call struct, we are sending required information such as the procedure enum and the file path that is being referenced, as those are needed for the server side. In the return, we are packaging whatever outgoing information is needed for the client end to complete.

Difficulties:

We initially had a learning curve with setting up the FUSE system …
We didn't know how to approach locks and mutual exclusion initially …
We had issues with sending information over sockets and maintaining structs …

Working Examples:

touch /tmp/fuse/test_create
cat /tests/test1 > /tmp/fuse/test_create
mkdir /tmp/fuse/test_folder
ls -l /tmp/fuse/
Running a test binary file that performs some actions?