Project Members:

Nicholas Heah, Seong Kon Kim, Andrea Wu
Github handles: Andrea-Wu. Nickheah07, gon93

Contributions:

Nicholas Heah: getattr, create, open, read, flush
Andrea Wu: mkdir, opendir, releasedir, readdir
Seong Kon Kim: implemented the socket connections

Socket Setup:

Client Side: The function connectionForClientRequests returns a socket descriptor that will connect to the server. It calls connect() to connect to the server. Each file operations function calls connectionForClientRequests once.
Server Side: A socket is set up to recieve connections from the client. Then, a while loop waits for incoming connections from the client.

Server and Client RPC:

In each method, the client would send all of the necessary information to the server through a socket, making sure that it sent the message successfully each time. The server would receive them and do the operation required for each function. The server would return the result it would get, and if there was an error, it would send the errno back to the client. When necessary, the server and client would call htonl() or ntohl() on integers to make sure they were passed on or received correctly. This back and forth messaging required custom protocols for each function, as different functions would send and receive different data types, and some functions would send back and forth more or less times compared to another.

File Operations (how to handle on server side):

On the client side, we send all of the necessary data for running system calls for each request. After the system call is run, we send the result back to the client, and possibly send an error code.

Getattr: the server runs getattr() on the given filename.
Create: the server runs creat() with the given filename and flags
Open: the server runs open on the given filename, stores the file descriptor in packet struct
Read: runs read and sends the read data back to the client
Write: writes a string that the client gives to the correct file
Flush: on the server side, fsync() is called on the given file descriptor
Release: on the server side, the file descriptor given to the server is closed, and its entry is removed from the file linked list.
Truncate: the server recieves a filename and an offset, truncates the file to that length
Opendir: server opens a DIR* stream and stores its data in a linked list

Readdir server gets the DIR* stream corresponding to the filename that the client sends over. Readdir is run, and each directory name is placed into a buffer and sent back to the client. The client takes the buffer of filenames, and tokenizes it to separate the filenames. The “filler” function is run on each of the filenames.
Mkdir: A directory with the given name is created on the server side if possible
Releasedir: server releases the directory stream corresponding to the directory name that the client sends. The node belonging to that filename is removed from the linked list.

Working Operation Examples:

- After calling make, run the server executable on a machine, for example facade.cs.rutgers.edu. Give it a port number and a mounting point. Make sure that if there are files in the mounting point, you must have permission to read, write and execute them.
  Ex: ./serverSNFS -port 62798 -mount /tmp/mount

  On a different machine, run the client executable, and give it the same port and machine that the server is running on, as well as the mount point, which should exist.
  Ex: ./myClient --port 62798 --address facade.cs.rutgers.edu -f /tmp/fuse

  Then on the client’s machine, cd to the mount point

- Call ls, which will call getattr(), opendir(), readdir(), and releasedir()
- Call cat on a file, which will call getattr(), open(), read(), flush(), and release()
  Ex: cat hello.txt
- Echo any string into a file that **does not** exist, which will call create() in addition to some other calls seen before
  Ex: echo “hello” > test.txt
- Echo any string into a file that **does** exist, which will call truncate() in addition to some other calls seen before
  Ex: echo “world” > test.txt
- Call mkdir to make any directory, which will call mkdir() in addition to some other calls seen before
  Ex: mkdir test_dir

Problems we encountered:

- It was very hard to start testing, as we had to implement all of the functions for the fuse file operations *correctly* in order to see any working result. If we left one operation incomplete, like getattr() or readdir(), we would not be able to run even the simplest of commands
- In our RPC protocol, we used something similar to a TCP server, which made synchronizing the sent and received messages very challenging. We had to include many sleep statements that slowed down the operation time but guaranteed durability

and robustness of the messages being sent and received. We understand that it may take a couple of seconds for the command line call to finish, but it was necessary.

- If we closed the server and the client and reran the executables with the same mounting point for the client, fuse wouldn't let us run any commands, so before we could rerun our code, we would have to rmdir the mount point and mkdir it again in /tmp