Assignment 3: SNFS: Simple Network File System

Authors:

Alex Silva
Netid: ars366
ars366@scarletmail.rutgers.edu
Section: 02

Kimberly Russo
Netid: krr89
kimberly.russo@rutgers.edu
Section: 01

Warren Ho
Netid: wgh18
wgh18@scarletmail.rutgers.edu
Section: 02

Synopsis:

- Create mount directories for both client and server

- Run make inside each serverSNFS and clientSNFS directories

- To run the server:
  - ./serverSNFS -port [port number] -mount [server mount directory]

- To run the client:
  - ./clientSNFS -port [port number] -hostname [hostname of server] -mount [client mount directory] -f

- Example:
  - mkdir /tmp/hello
  - make
  - ./serverSNFS -port 123462 -mount ./serverDir
  - ./clientSNFS -port 123462 -hostname localhost -mount /tmp/hello -f

Attention:

- **Please make sure mounted directories exist before running both executables.**

- **If multiple file systems are mounted and running do not interrupt and kill any of them until all tests are finished.**

Description:

Server: The server establishes connections with clients and maintains a local file system where all files, directories and operations are stored and performed. It uses getaddrinfo to get the information of the current machine and a particular port to create a socket and bind to it.Once the server is binded to a particular host and port number it will go into an infinite loop where it will wait for incoming connections and spawn worker threads for each file system that connects to it. Each thread will then await for operations sent from the client file system, to perform those operations on its local file system. The server will not terminate unless an error occurs that forces the server to shut down or a user exit command.

Client: The client mounts the file system that catches a users' operations on that particular file system. It starts by connecting to the server via a socket and mounts the file system in a particular. Once the client is connected to the server it mounts the file system on an existing directory selected by the client in his system. With the mounted file system, a user will be able to perform any operations, create files, directories… and these will be caught by the client and sent over to the server to be perform those operations on it's local directory. When the client receives a confirmation from the server of its actions, the client will then display those results back to the user.

User: The users perform normal file operations on the directories the file servers are mounted on.

Protocol:

All communications performed over the socket are accomplished via strings.

Client:

- Send to server the command inputted by the user
- Receive from server the validity of the command
- Send to server the file or directory path to perform the operation on
- Receive operation success or failure info from server

Server:

- Wait and receive from client user operations
- Tell client if operation exists or not and await user operation data
- Receive from client the operation data and perform the actions requested
- Send to client the status of the tasks performed

Test Cases:

All commands are being executed sequentially but can be ran independently as long as the files and directories exist. Screenshots of these test cases are located in the "test cases" folder

- Single Client:
  - touch file.txt
  - touch file2
  - mkdir newDir
  - echo writingToFile > file.txt
  - cat file.txt
  - cp file.txt ./newDir
  - cd ./newDir
  - ls
  - vim file.txt
  - mkdir dir1; cd dir1; mkdir dir2; cd dir2; touch file3; echo sendmetofile1 > file3; cat file3; cat file3 > ../../../file2; cd ../../../; cat file2
  - mkdir newDir/dir1/dir2/ newDir3; cp file2 ./newDir/dir1/dir2/newDir3/; cat ./newDir/dir1/dir2/newDir3/file2

- Multi Client:
  client1 | client2 | client3
  - mkdir multiDir
  - cd multiDir
  - cp file2 ./multiDir/
  - ls
  - cd multiDir/; mkdir new
  - cd new/; touch file4
  - cat file2 > ./multiDir/new/file5; cat ./multiDir/new/file5
  - echo newString > ./multiDir/new/file5
  - cat file5 > ../file2
  - cat file2

Design:

```
static int client_getattr(const char*, struct stat*);
```
and
```
int do_getattr(char*, int);
```

Client_getattr() will send the path in question over a socket. The path is then received from do_getattr() which then finds the file or directory specified by the path. The full path and the attributes are sent over the socket and is then given back to client_getattr() which then sets the attributes of the file.

```
static int client_readdir(const char*, void*, fuse_fill_dir_t, off_t, struct
fuse_file_info*);
```
and
```
void do_readdir(char*, int);
```

Client_readdir() will send the path over to the server which do_readdir() will receive. Do_readdir() will take this path, and check all files which appear in this path. The list of file/directory names found will be sent over a socket back to client_readdir(). Client_readdir() then adds each name to the filler along with the current and parent directories.

```
static int client_read(const char*, char*, size_t, off)t, struct
fuse_file_info*);
```
and
```
void do_read(char*, char*, size_t, off_t, int);
```

Client_read() will send a message containing the operation, size and offset over to do_read() through a socket. Do_read() will receive this message and perform a read with all the given information. The data that is read is then sent back over to client_read() and is copied into client_read()'s buffer.

```
static int client_write(const char*, const char*, size_t, off_t, struct
fuse_file_info*);
```
and
```
void do_write(char*, char*, size_t, off_t, int);
```

Client_write() will send a message containing the operation , the string to write, the size, and the offset over a socket to do_write(). Do_write will receive this message and perform the specified write. Bytes actually written will be sent back over to client_write().

```
static int client_open(const char*, struct fuse_file_info*);
```
and
```
void do_open(char*);
```

Open is a dummy function. They return zero.

```
static int client_release(const char*, struct fuse_file_info*);
```

Client_release() is a dummy function, server side implementation was not needed.

```
static int client_create(const char*, mode_t, struct fuse_file_info*);
```
and
```
void do_create(char*, mode_t, int);
```

Client_create() will send a message containing the path and the mode of the file to be created. Do_create() receives this and creates the file with the appropriate mode.

```
static int client_truncate(const char *, off_t);
```
and
```
void do_truncate(char*, off_t);
```

Client_truncate() sends a message containing the operation "t", the path, and the offset to the server. Do_truncate() receives it and performs truncate on the specified file.

```
static int client_mkdir(const char *, mode_t);
```
and
```
void do_mkdir(char*, mode_t, int);
```

Client_mkdir() sends a message with the operation "m" and the mode over to the server. Do_mkdir() receives this message and creates the directory if it does not exist. The return value from mkdir is sent back over from the socket.

Performance Evaluation:

The overall performance of the SNFS runs smoothly. Even with multiple clients having multiple requests, the server seems to have no problem keeping up. All requests are almost instantaneous. Unix file system commands look identical from what we have tested.

Difficulties Encountered:

The main difficulties we encountered with this project was learning how FUSE works. Researching took the majority of time although the project still required a lot of code and logic. Finding what should be returned to the FUSE functions and exactly how to implement them was a little tough because most of the documents found online seemed to be out of date or weren't very clear. Also we had to dig pretty deep to find resources online, it felt like there was not much information about FUSE on the web. Furthermore, we also encountered a lot of issues regarding segmentation faulting and overflows within implementing the multi client file systems.

Work Distribution:

- Alex: Socket and Threading Protocol
- Kimberly: Client and Fuse Operations
- Warren: Server and File System Operations