

# **San Francisco State University**

**CSC 415 - Summer 2023**

**File System**

**Team Drivers**

**Github name: rorymcginnis1**

**Github repository link:**

*<https://github.com/CSC415-2023-Summer/csc415-filesystem-rorymcginnis1.git>*

**Kaung Nay Htet      |      922292784**

**Himal Shrestha      |      922399514**

**Rory McGinnis      |      921337245**

**James Donnelly      |      917703805**

## 1. The github link for your group submission.

- <https://github.com/CSC415-2023-Summer/csc415-filesystem-rorymcginnis1>

## 2. A description of your file system

- Our file system is a basic in-memory file system designed to simulate a simple file management system. It consists of several key components: ``extents``, ``fsInit``, and ``mfs`` modules, along with supporting functions in ``b_io``. The ``extents`` module is responsible for managing the allocation of data blocks in our file system. It uses a extent data structure to keep track of used and free data blocks. The ``extents.h`` header file defines necessary constants and function declarations. The ``extents.c`` file contains implementations for functions like initializing the extent, finding and allocating free blocks, and freeing used blocks.

Second, the ``fsInit`` module deals with the initialization and setup of the file system. It includes ``fsInit.h`` and ``fsInit.c`` files. This module contains functions to create the necessary data structures and metadata for the file system, such as the root directory and directory entry structures. The ``fsInit`` module lays the foundation for other components and ensures the file system starts in a consistent state.

Next, the core of our file system is the ``mfs`` module which includes ``mfs.h`` and ``mfs.c`` files.

This module handles various file and directory operations, such as creating and deleting directories, navigating through directories, reading directory entries, and obtaining file statistics.

It implements functions for opening, reading, and closing directories, as well as setting the current working directory. The ``mfs`` module interacts with other modules and data structures to provide a functional file system experience.

Last but not least, includes the ``b_io`` module, which handles the essential input/output operations. It allows us to open, read, write, seek, and close data blocks, storing and retrieving data from the underlying storage, which can be a disk or any other memory storage. This module

ensures data is preserved across different sessions, making our file system reliable and usable over time.

### **3. Issues we had**

#### **General Issues about the project**

One issue we had was deciding how to start this. Even though you gave steps this was a project which needs a lot of planning which we are not used to. We had a long talk and figured that we needed to plan. Another issue that we had with this project was how to divide up the work as we did not know how difficult each part would be since we have never done this before. We ended up assigning pieces to do and if we finished without part we helped others with theirs. In addition we checked each other's code though we do not know how helpful this may have been. We tried implementing a bitmap which was unsuccessful. No matter what we tried to make it work, we could not figure it out. We tried looking over the code over and over, multiple little revisions that we shared with each other but still nothing. So, oftentimes we had to go back to class recorded lecture videos and get some hints as well as lots of research. The other helpful thing was that we were allowed to use the professor's freeSpace header file as reference which gave us some general idea about what our freeSpace should look like. After that we created our version of the freeSpace file called “extents.h” and “extents.c” and gradually we were able to make progress on that.

#### **Issue 1:**

While working on implementing the `parsePath` function in `mfs.c`, our group encountered a few challenges. The main issue was related to correctly tokenizing the

input ``pathname`` and extracting individual directory names from it. We needed to break down the given path into separate components to identify the directories and navigate through the file system accordingly. Another challenge was handling edge cases and ensuring the function worked effectively for various inputs, including absolute and relative paths.

### **Solutions:**

To address the tokenization problem, we used the ``strtok`` function, which allowed us to split the input ``pathname`` using the delimiter ``/``. This way, we could extract each directory name from the given path and store them in an array for further processing. We also ensured to properly terminate the array to avoid any unexpected behavior. After the array of characters were tokenized, they were then stored into an array of tokens. The list present within the array presents the file's path in a sequence.

Furthermore, to handle edge cases and ensure the function's correctness, we decided to differentiate between parsing for the parent directory and parsing for changing the current working directory (CWD). By using a parameter ``parentParse``, we could switch between these modes and tailor the function accordingly. In parent parsing mode, we could find the parent directory and return its index. In CWD parsing mode, we could reconstruct the path and retrieve the index of the last entry, representing the target directory. This approach allowed us to handle different scenarios and make the ``parsePath`` function more versatile and reliable. Additionally, we set global variables for ``cwd`` and ``tmpcwd`` to keep track of the current working directory and intermediate path during the parsing process.

### **Issue 2:**

``fs_setcwd()`` Issue: The main problem with ``fs_setcwd()`` was verifying the validity of the provided pathname and ensuring it represented a valid directory in the file system. Failure to handle this properly could lead to setting an invalid current working directory and subsequent errors in file system operations.

**Solutions:**

``fs_setcwd()`` Solution: To tackle the pathname validity problem, we integrated the ``parsePath()`` function within ``fs_setcwd()``. This allowed us to verify if the provided pathname was a valid directory in the file system or not. If the pathname was invalid, we returned an error code.

**Issue 3:**

While working on the ``seek()`` function in the ``b_io.c`` file, we encountered an issue related to the management of the file pointer position. The primary problem was ensuring that the file pointer stayed within the valid range of the file and did not exceed the file size.

**Solutions:**

To address this issue, we performed range checking to ensure that the position specified by the ``whence`` and ``offset`` parameters falls within the valid range of the file. If the calculated position was negative or beyond the file size, we returned an error code, indicating an invalid seek operation. Next, after doing little bit of research, we found out that the ``whence`` parameter in the ``seek()`` function can take different values like ``SEEK_SET``, ``SEEK_CUR``, or ``SEEK_END``, representing different ways to interpret the ``offset``. We carefully handled each value and calculated the correct position based on the current file pointer, offset, and the specified ``whence``. Last but not least, after calculating the new position, we updated the file pointer to the correct position, ensuring that the subsequent read and write operations would start from the appropriate location.

**Issue 4:**

While working on `fs_mkdir()` one big issue we faced was writing it to memory. Whenever we tried it did not work. We checked the memory allocation errors but could not find any. Is that to say we have no errors there? Unlikely but he had no luck. We consistently faced an issue where it didn't write data to the disk as expected. We carefully analyzed the code, checked for problems with writing data to the storage, and reviewed the interaction with the file system's extent management system but nothing. Despite our efforts and limited by our timeframe, we couldn't identify any clear errors or issues that would explain the failure to write data to the disk. The directory creation process, which should involve saving data to the file system, did not function as intended. Maybe we should have focused more on this but our solution for this step was to work around it for the time being and come back to it if time allowed.

**Issue 5:**

Another issue we had was with `ls`. When we would run the `ls` function it would say that the directory we are trying to view was invalid but it was not invalid as it was just made. It had to do with the `fdDir *fs_opendir()` function. The `dir` structure was attempting to allocate before it was defined in the function. This was a simple fix nonetheless. All we had to do was move the definition of the `dir` before memory for `dir` was being allocated so this function would work properly. Also we changed the return value from `0` to `NULL` since it is a pointer.

```

parallels@ubuntu-linux-22-04-desktop:~/415/csc415-filesystem-rorymcginnis1$ make run
gcc -c -o mfs.o mfs.c -g -I.
gcc -o fsshell fsshell.o fsInit.o mfs.o b_io.o fsLowM1.o -g -I. -lm -l readline -l pthread
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872; BlockSize: 512; Return 0
Initializing File System with 19531 blocks with a block size of 512

|-----|
|----- Command -----| - Status -|
| ls                |      ON      |
| cd                |      ON      |
| md                |      ON      |
| pwd               |      ON      |
| touch             |      ON      |
| cat               |      ON      |
| rm                |      ON      |
| cp                |      ON      |
| mv                |      OFF     |
| cp2fs             |      ON      |
| cp2l              |      ON      |
|-----|

Prompt > md taco
Directory 'taco' created successfully.
Prompt > cd taco
Prompt > ls taco
Error memory allocate directory to store.
Directory to be opened is invalid.
Prompt > █

```

### Issue 6:

Another issue we had also related to the ls function. It was now accepting the directory to be opened as valid but would run into a segmentation fault right after. We went about to look at code related to this. We found `fs_readdir()` was causing the trouble. Despite the directory entry position appearing valid, the function didn't actually check its validity, which potentially led to undefined behavior. We also ran into a stumbling block with the result pointer. We set it to NULL, but did not allocate any memory for it. So, when we tried to use result later on, we ran head into errors. To add to our problems, we didn't put in any protection in case the memory allocation failed. So, when it did, our program would crash instantly bam segmentation fault. Finally, our function didn't handle situations where we'd reached the end of the directory entries, nor did it advance to the next entry after finishing with the current one.

To solve these issues involved, we made a few critical changes. First, we put in a validation check for the directory entry position to ensure it fell within a legitimate range. We also fixed the result pointer by

allocating memory for it using malloc(). Then, we checked if this allocation was successful, making sure the result wasn't NULL. We handled the end-of-directory entries situation by introducing a condition that freed up the result memory and returned NULL if we hit the last entry. Lastly, we tweaked the function to increment the position and move to the next entry after processing the current one.

```
parallels@ubuntu-linux-22-04-desktop:~/415/csc415-filesystem-rorymcginnis1$ ^C
parallels@ubuntu-linux-22-04-desktop:~/415/csc415-filesystem-rorymcginnis1$ make run
gcc -c -o mfs.o mfs.c -g -I.
gcc -o fsshell fsshell.o fsInit.o mfs.o b_io.o fsLowM1.o -g -I. -lm -l readline -l pthread
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872; BlockSize: 512; Return 0
Initializing File System with 19531 blocks with a block size of 512

|-----|
|----- Command -----| Status |
| ls          |      ON |
| cd          |      ON |
| md          |      ON |
| pwd         |      ON |
| touch       |      ON |
| cat         |      ON |
| rm          |      ON |
| cp          |      ON |
| mv          |     OFF |
| cp2fs       |      ON |
| cp2l        |      ON |
|-----|
Prompt > md taco
Directory 'taco' created successfully.
Prompt > cd taco
Prompt > ls
Directory to be opened is valid.
make: *** [Makefile:67: run] Segmentation fault (core dumped)
parallels@ubuntu-linux-22-04-desktop:~/415/csc415-filesystem-rorymcginnis1$
```



## Issue 7:

```
parallels@ubuntu-linux-22-04-desktop: ~/Documents/test/cs...
b_io.c:131:5: warning: implicit declaration of function 'LBAwrite' [-Wimplicit-f
unction-declaration]
  131 |     LBAwrite(fd, buffer, block_number, bytes_to_write_in_block);
      |     ^~~~~~
b_io.c: In function 'b_read':
b_io.c:241:29: warning: implicit declaration of function 'LBAread' [-Wimplicit-f
unction-declaration]
  241 |         bytesRead = LBAread (buffer+ part1, numberOfBlocks, fcbA
rray[fd].currentBlk + fcbArray[fd].fi->fileLocation);
      |                             ^~~~~~
/usr/bin/ld: /usr/lib/gcc/aarch64-linux-gnu/11/../../../../aarch64-linux-gnu/Scrt1.
o: in function `_start':
(.text+0x1c): undefined reference to `main'
/usr/bin/ld: (.text+0x20): undefined reference to `main'
/usr/bin/ld: /tmp/ccgs6LgW.o: in function `b_write':
b_io.c:(.text+0x348): undefined reference to `LBAwrite'
/usr/bin/ld: b_io.c:(.text+0x434): undefined reference to `LBAwrite'
/usr/bin/ld: b_io.c:(.text+0x50c): undefined reference to `LBAwrite'
/usr/bin/ld: /tmp/ccgs6LgW.o: in function `b_read':
b_io.c:(.text+0x880): undefined reference to `LBAread'
/usr/bin/ld: b_io.c:(.text+0x970): undefined reference to `LBAread'
collect2: error: ld returned 1 exit status
parallels@ubuntu-linux-22-04-desktop:~/Documents/test/csc415-filesystem-rorymcgi
nnis15
```

An issue we had was we were not able to use LBAread and LBAwrite in our function. This problem gave us issues for a while when we were trying to compile. We could not figure it out,

ultimately we realized that we did not have the correct files included and

```
1 /*****
2 * Class:  CSC-415-0# Fall 2021
3 * Names:
4 * Student IDs:
5 * GitHub Name:
5 * Group Name:
7 * Project: Basic File System
3 *
3 * File: b_io.c
3 *
1 * Description: Basic File System - Key File I/O Operations
2 *
3 *****/
4
5 #include <stdio.h>
5 #include <unistd.h>
7 #include <stdlib.h>           // for malloc
3 #include <string.h>          // for memcpy
3 #include <sys/types.h>
3 #include <sys/stat.h>
1 #include <fcntl.h>
2 #include "b_io.h"
3 #include "fsInit.h"
4
5 #define MAXFCBS 20
5 #define B_CHUNK_SIZE 512
```

After the code above was implemented the code compiled.

## Issue 8

Another issue we had was misreading some of the functions. LBAread we misread as having four arguments.

```
parallels@ubuntu-linux-22-04-desktop: ~/Documents/test/cs...
b_io.c:158:33: warning: passing argument 2 of 'LBAwrite' makes integer from pointer without a cast [-Wint-conversion]
  158 |         LBAwrite(fd, buffer + fcbArray[fd].index, block_number, count);
      |                                ^
      |                                |
      |                                char *
In file included from b_io.c:24:
fsLow.h:61:44: note: expected 'uint64_t' {aka 'long unsigned int'} but argument is of type 'char *'
   61 | uint64_t LBAwrite (void * buffer, uint64_t lbaCount, uint64_t lbaPosition);
      |
b_io.c:158:13: error: too many arguments to function 'LBAwrite'
  158 |         LBAwrite(fd, buffer + fcbArray[fd].index, block_number, count);
      |         ~~~~~^~~~~~
In file included from b_io.c:24:
fsLow.h:61:10: note: declared here
   61 | uint64_t LBAwrite (void * buffer, uint64_t lbaCount, uint64_t lbaPosition);
      |
parallels@ubuntu-linux-22-04-desktop:~/Documents/test/csc415-filesystem-rorymcginnis1$
```

The issue was for some reason I blanked and thought I had to include the fd when referencing fd. Obviously this did not compile, and once I changed it so that fd was not part of there function it worked.

## Issue 9

```
parallels@ubuntu-linux-22-04-desktop:~/Documents/csc415-assignment-5-rorymcginni
s1$ make run
gcc -c -o b_io.o b_io.c -g -I.
b_io.c: In function 'b_open':
b_io.c:116:21: error: 'b_fcb' has no member named 'location'
  116 |         fcbArray[fd].location=info->location;
      |                        ^
b_io.c: In function 'b_read':
b_io.c:189:73: error: 'b_fcb' has no member named 'location'
  189 |         uint64_t byteLocation= B_CHUNK_SIZE*fcbArray[fd].location;
      |                                                                ^
b_io.c: In function 'b_close':
b_io.c:240:17: error: 'b_fcb' has no member named 'location'
  240 |         fcbArray[fd].location=0;
      |                 ^
make: *** [Makefile:61: b_io.o] Error 1
parallels@ubuntu-linux-22-04-desktop:~/Documents/csc415-assignment-5-rorymcginni
s1$
```

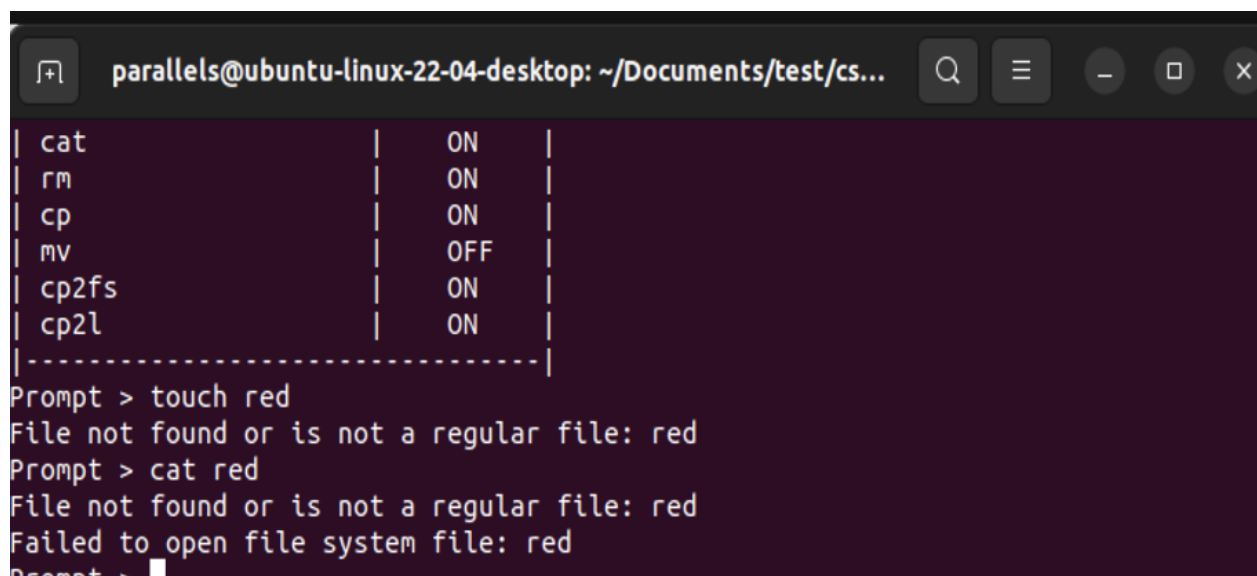
Another issue is just the misnaming of names, issues like this often occurred when working late into the night with the team, this was a fairly simple issue to rectify once I read the error message, but originally it was shocking that it caused compilation errors.

## Issue 10

```
mfs.c:(.text+0x24): undefined reference to `newD'
/usr/bin/ld: mfs.c:(.text+0x28): undefined reference to `newD'
/usr/bin/ld: mfs.c:(.text+0x64): undefined reference to `newD'
/usr/bin/ld: mfs.c:(.text+0x68): undefined reference to `newD'
/usr/bin/ld: mfs.c:(.text+0xac): undefined reference to `newD'
/usr/bin/ld: /tmp/ccHwIJCe.o:mfs.c:(.text+0xb0): more undefined references to `newD'
follow
collect2: error: ld returned 1 exit status
parallels@ubuntu-linux-22-04-desktop:~/Documents/test/csc415-filesystem-rorymcginnis1
$ gcc mfs.c -o mfs.o
/usr/bin/ld: /usr/lib/gcc/aarch64-linux-gnu/11/../../../../aarch64-linux-gnu/Scrt1.o: in
function `_start':
(.text+0x1c): undefined reference to `main'
/usr/bin/ld: (.text+0x20): undefined reference to `main'
/usr/bin/ld: /tmp/ccW9cfkM.o: in function `fs_stat':
mfs.c:(.text+0x24): undefined reference to `newD'
/usr/bin/ld: mfs.c:(.text+0x28): undefined reference to `newD'
/usr/bin/ld: mfs.c:(.text+0x64): undefined reference to `newD'
/usr/bin/ld: mfs.c:(.text+0x68): undefined reference to `newD'
/usr/bin/ld: mfs.c:(.text+0xac): undefined reference to `newD'
/usr/bin/ld: /tmp/ccW9cfkM.o:mfs.c:(.text+0xb0): more undefined references to `newD'
follow
collect2: error: ld returned 1 exit status
parallels@ubuntu-linux-22-04-desktop:~/Documents/test/csc415-filesystem-rorymcginnis1
$ make clean
rm fsshell.o fsInit.o fsshell
rm: cannot remove 'fsshell.o': No such file or directory
rm: cannot remove 'fsInit.o': No such file or directory
rm: cannot remove 'fsshell': No such file or directory
```

We forgot to create a global reference and instead were trying to reference our newD (the struct that holds our directory entries) in a different local environment. Obviously this caused an issue and we had to create a global variable for the directory entries. This seems like an obvious solution, but we could not figure it out as we were referencing it and including the file it was in. Ultimately it required us to just create a struct named GlobalDirEntries.

## Issue 11



```
parallels@ubuntu-linux-22-04-desktop: ~/Documents/test/cs...  
| cat           |      ON      |  
| rm            |      ON      |  
| cp            |      ON      |  
| mv            |     OFF     |  
| cp2fs         |      ON      |  
| cp2l          |      ON      |  
|-----|  
Prompt > touch red  
File not found or is not a regular file: red  
Prompt > cat red  
File not found or is not a regular file: red  
Failed to open file system file: red  
Prompt >
```

After creating open, I was able to create a file using touch, but cat would not work. In the code I originally had, cat was never reachable, it would always first go through and create a file, and if the file already existed it would just exit. So the cat code was completely skipped. As a result I had to make the cat code reachable, then it ultimately worked.

## Detail of how your driver program works

- Our main driver program, 'fsshell.c' file servers as the entry point to interact with the file system. It utilizes the functions from various headers and source files that we worked on

such as, fsLow.h, fsLow.c, mfs.h, mfs.c, b\_io.h, b\_io.c etc to interact with the file system's operations. It basically has the main function to run the file systems as well as the command functions similar to normal file systems. Upon execution, the main function sets up the file system by opening a specific volume file and initializing the file system with the given volume size and block size.

- One of the primary groups of functions that are in place to initialize the file system lies within the 'fsInit.c' file. This file contains the following functions:

initialize\_root\_directory, initFileSystem, exitFileSystem. The first function, "initialize root directory" is capable of initializing the root directory. The first two directory entries within the file system are "." and its child "..". Other accompanying metadata such as fileLocatin and fileSize are also populated appropriately. The second function, initFileSystem, is capable of initializing the file system by initializing the volume control block populating the data within. In this process, the free space map is also processed. Finally, the "exitFileSystem" frees the memory of the allocated data and exits the file system.

- Another primary group of functions is the "mfs.c" file. This file contains the directory entries of the file system. Firstly present is the parsePath function. This function is able to parse and tokenize the address of the file or the current working directory. Once parsed, it is returned the index of the desired directory, which is prompted as the child or the parent. With the use of parsepath, we can also use the three functions of opendir, readdir, and closedir. The opendir makes a temporary allocation of a directory entry that is to be accessed, which disables the file system from accessing the directory entries themselves and is able to access the data within them one by one. The readdir function is able to

process the data derived from the opendir function, while the closedir function clears the memory allocation used by the opendir process. Followed within the file are more functions that provide utility to the file system. The fs\_stat function derives the data from a given path and returns it to the buffer that was fed. The fs\_delete function takes in the parameter of a file name and once matched with a file, the entry is removed. The fs\_rmdir serves the same process as fs\_delete except that it applies to directory entries. There are two helper functions, isDir and isFile, which is used to identify whether the directory entry in question is a file or a directory by deriving the data from the entry. The function fs\_mkdir is capable of making more directories, whether it be a file or a directory, based on the given mode. The fs\_getcwd is the function that returns the current working directory, while the fs\_setcwd is the function that sets the current working directory.

**a. Screenshot of compilation**

```

parallels@ubuntu-linux-22-04-desktop:~/Desktop/SummerCSC415/csc415-filesystem-rorymcginnis1$ make
gcc -c -o fsshell.o fsshell.c -g -I.
gcc -c -o fsInit.o fsInit.c -g -I.
gcc -c -o mfs.o mfs.c -g -I.
gcc -c -o b_io.o b_io.c -g -I.
b_io.c: In function 'b_close':
b_io.c:352:18: warning: passing argument 1 of 'LBAwrite' makes pointer from integer without a cast [-Wint-conversion]
   352 |         LBAwrite(fd, fcbArray[fd].buf, fcbArray[fd].buflen);
       |                  ^~
       |                  |
       |                b_io fd {aka int}
In file included from b_io.c:24:
fsLow.h:61:27: note: expected 'void *' but argument is of type 'b_io fd' {aka 'int'}
   61 |     uint64_t LBAwrite (void * buffer, uint64_t lbaCount, uint64_t lbaPosition);
       |                      ~~~~~~
b_io.c:352:34: warning: passing argument 2 of 'LBAwrite' makes integer from pointer without a cast [-Wint-conversion]
   352 |         LBAwrite(fd, fcbArray[fd].buf, fcbArray[fd].buflen);
       |                      ~~~~~~^~
       |                      |
       |                    char *
In file included from b_io.c:24:
fsLow.h:61:44: note: expected 'uint64_t' {aka 'long unsigned int'} but argument is of type 'char *'
   61 |     uint64_t LBAwrite (void * buffer, uint64_t lbaCount, uint64_t lbaPosition);
       |                      ~~~~~~^~
gcc -o fsshell fsshell.o fsInit.o mfs.o b_io.o fsLowM1.o -g -I. -lm -l readline -l pthread
parallels@ubuntu-linux-22-04-desktop:~/Desktop/SummerCSC415/csc415-filesystem-rorymcginnis1$

```

**b. Execution of program:**

```
parallel@ubuntu-linux-22-04-desktop:~/415/csc415-file-system-rorymcginnis$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872; BlockSize: 512; Return 0
Initializing File System with 19531 blocks with a block size of 512
```

Command	Status
ls	ON
cd	ON
md	ON
pwd	ON
touch	ON
cat	ON
rm	ON
cp	ON
mv	OFF
cp2fs	ON
cp2l	ON

```
Prompt > pwd
./.../
```



c. 'md' Command

```
Prompt > md taco  
Directory 'taco' created successfully.
```

d. 'cd' & 'pwd' Command

```
Prompt > cd taco  
Prompt > pwd  
taco/
```

e. 'ls' command

```
Prompt > ls  
Directory to be opened is valid.  
  
taco
```

f. 'touch' & 'remove' commands

```
Prompt > touch bubble
Prompt > rm bubble
File 'bubble' removed successfully.
Prompt > rm taco
Directory 'taco' removed successfully.
Prompt > █
```

g. 'cat' command

```
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872; BlockSize: 512; Return 0
Initializing File System with 19531 blocks with a block size of 512
```

Command	Status
ls	ON
cd	ON
md	ON
pwd	ON
touch	ON
cat	ON
rm	ON
cp	ON
mv	OFF
cp2fs	ON
cp2l	ON

```
Prompt > touch red
Prompt > cat red
Prompt > cat re
File not found: re
Failed to open file system file: re
Prompt > █
```

≡ idekbro

```
1  hey there bierman
```

**h. 'cp2l' & cp2fs commands**

```
Prompt > cp2fs idebro bubble  
Prompt > cp2fs idekbro bubble  
Prompt > cp2l bubble idekbro2  
Prompt > 
```

```
≡ idekbro
```

```
1
```

**i. 'mv' command doesn't work**