

Team name: Clouds

Names & Student IDs:

Cristian Garcia & 920317889 ,

Francis Quang & 912679019,

Tsewang Sonam & 921911364,

Kurtis Chan & 918139175

GitHub: <https://github.com/CSC415-2022-Fall/csc415-filesystem-VtecFrancis>

## **1. A description of your file system**

Our file system stores files and directories on disk. For managing free space, our file system uses a bitmap. The bitmap is set to 0 to indicate if a block is used and the information is written. It is set to 1 if the block is free and nothing is written. Our directory entry is held in a typedef struct and it contains an identifier variable for the file system to use as an ID, a name variable, a protected variable that says whether the file is protected or not, a boolean variable that says if the item is a file or directory, a directory size variable, directory location variable, time last modified variable, and a last user modified variable.

The file system has the volume control block in a typedef struct and it contains the number of blocks needed in the block address, the number of free blocks, the number of free file control blocks, the number of free file control block pointers, a magic number that tells if the volume is initialized, the maximum number of bytes of data in each block, and the block where the free space starts. The main driver of our program initializes the volume control block, root directory, and free space. Our file system is able to perform the open, seek, read, and close I/O functions. Some commands that work with our file system include

## **2. Issues you had**

At the start of the files system project, we had a problem understanding and implementing the free space and root directory. After reviewing the concepts with group members we had a better understanding and solved the problem efficiently. So when we decided to divide the work for milestone two and then merge our work at the end to solve it together.

The problem with that approach was some team members had to work on part of a file system project that was harder than the rest. With everyone working on separate parts of the project, we experience communication issues. Looking back at it now, we should have focused on finishing milestone two completely and moving forward to milestone 3.

I, Tsewang, had many issues while working on functions like the free space or fs\_stat where I did not spend enough time understanding what needs to be done. I was stuck in a loop of errors and no progress. Later I realized the importance of understanding the concepts and use of manpage to know the details about the Linux functions is a key to making advancement in this project.

One issue particular to me, Sonam, is that although I clearly understand the concepts like directory, free space, LBA, etc after reading and reviewing materials about it. When it comes to coding it in C language it was a struggle. Half of the time I was solving syntax errors and random errors that are foreign to me. Although it was a nice trial and tribulation, it was for sure a big issue for me.

Another issue was that our idea of putting the divided work together at the end was not fruitful as not all the teammates were able to finish their work. So, even though most of the functions were done, we were not able to put the final product together on time and debug it. To resolve this, in the future we'd check in more often on our teammates' progress through meetings and offer more support. We put a lot of trust in one another and in the end, someone did not turn in any functions that they were assigned and that really limited us and prevented us from finishing our file system project.

An issue I, Kurtis, had with this project was working on the `parsePath` function and how to deal with relative paths since you may not have a starting point. At first I thought to search through each directory to find it, but realized that would not work because it will always go to the first instance of the first directory name. Then I looked into `getcwd()` and figured it would be easier to use that (used our `fs_getcwd()`) since the regular `getcwd()` is supposed to return the absolute pathname for the current working directory.

Another issue for me, Kurtis, was at the start with `initRoot` was wrapping my head around how it will make a directory every time and what the structure will be like. After talking about it with my team and expressing my idea it helped a lot organizing what I was thinking and trying to do.

### **3. Detail of how your driver program works**

The driver of our program is located in the `fsInit.c` file. It works by first initializing our volume control block (VCB). It first creates a VCB pointer by allocating a block size. Then `LBareads` block 0. The program then checks to make sure that the volume has not already been initialized by comparing a signature variable to our already-defined VCB magic number. If the volume isn't initialized then we make sure that the number of blocks requested can be held in the VCB. If so, then we initialize values in the VCB. These values are the number of blocks, the number of free blocks, the block size, the location the free space starts, and the signature. Afterward, we initialized our free space.

In the function to initialize our free space, we create variables that hold variables to hold the total number of bits needed for the VCB, the number of bytes needed for the VCB, and the number of blocks needed for the VCB. We also created variables that hold any remainder for the number of bytes, bits, and blocks. If there remainders, there is a loop that places those remainders back into their respective variables. We then malloc the free space map and furthermore, in the VCB block 0, allocate the first 6 bits as 0 or occupied and the remaining as 1 or free. Next, 5 blocks are written starting from block 1. Finally, the free space map is freed, set to null, and the starting block is returned.

Next, the root directory is initialized. We created a separate function in which this takes place. In that function, we created variables that set the maximum number of entries per directory, the size of the directory, the number of blocks we need, and a variable that checks if another block is needed to fit the data. Afterward, we created a directory entry pointer. Using this pointer we set the boolean variable that signaled if the block was free, to 1 which meant that they were all free. Next, we initialized the updated time variable, the starting block, directory name, identifier, the last modified, is directory checker, and size variables for both the "." and the

“.” entries. Then, we write the directory entry pointer. Finally, the directory entry pointer is freed, set to null, and the starting block is returned.

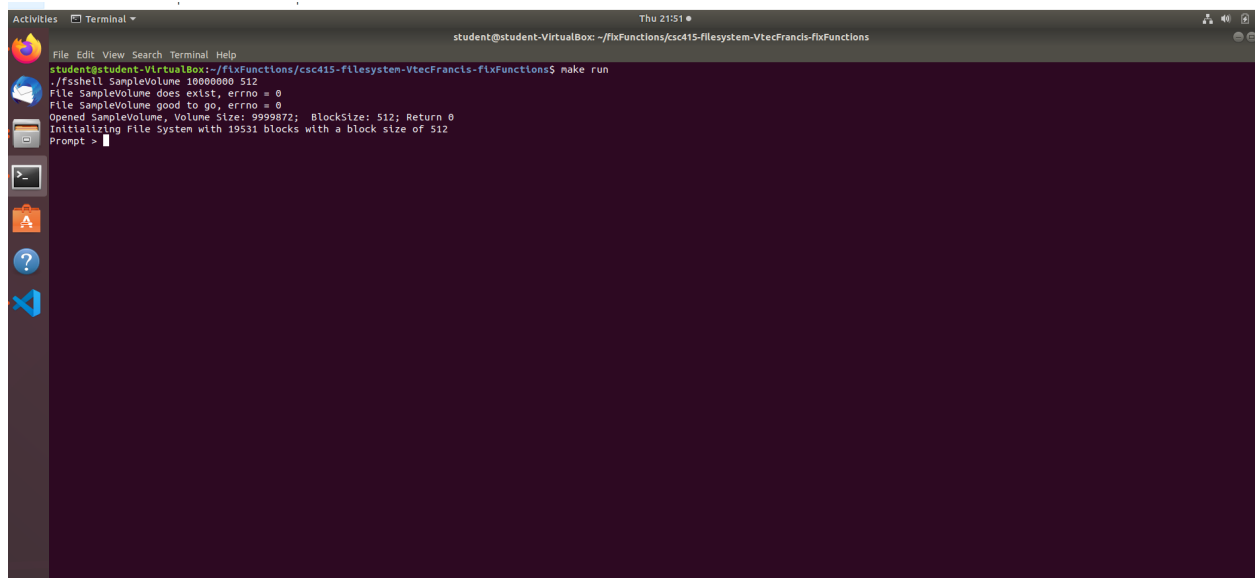
Finally, once the VCB, free space, and root directory are initialized, the VCB is written to block 0, also known as the disc. There's also a separate function that signals that the file system has been exited.

#### 4. Screenshots showing each of the commands listed in the readme

A big part of this project was working in a group and there is always something to learn from such experience. In our experience, one of our teammates had stopped communicating in the last critical days of putting together the final project. With the assurance that his part (functions such as `Fs_opendir`, `fs_readdir`, `fs_closedir`, and `b_write` ) are almost done and we were relieved that the file system project would be done.

Without those functions, we were unable to get the whole project working or debug it. It was too late in the final days for us to work on those functions.

In the screenshot below you can see that the shell has compiled and started running. Calling `help` to list out all the functions to use.



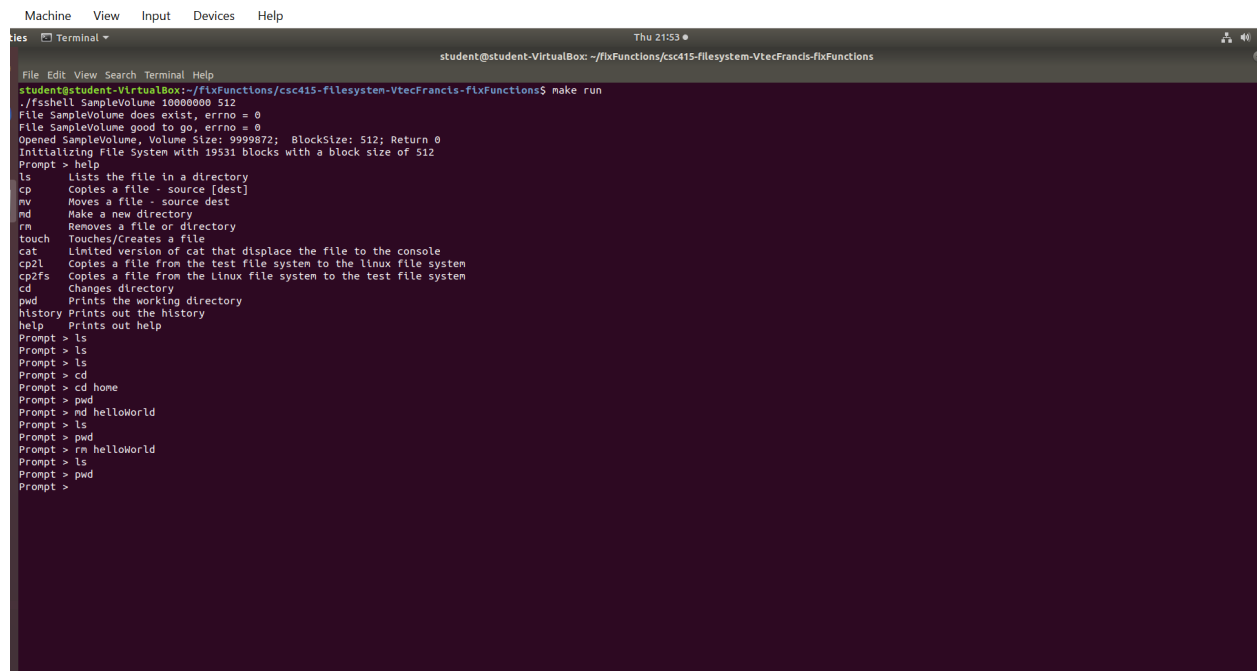
```
student@student-VirtualBox: ~/flxFunctions/csc415-filesystem-VtecFrancis-flxFunctions
student@student-VirtualBox:~/flxFunctions/csc415-filesystem-VtecFrancis-flxFunctions$ make run
./fshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872; BlockSize: 512; Return 0
Initializing File System with 19531 blocks with a block size of 512
Prompt >
```

```
student@student-VirtualBox: ~/fixFunctions/csc415-fileSystem-VtecFrancis-flxFunctions$ 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
Prompt > help
ls      Lists the file in a directory
cp      Copies a file - source [dest]
mv      Moves a file - source dest
md      Make a new directory
rm      Removes a file or directory
touch   Touches/Creates a file
cat     Limited version of cat that displace the file to the console
cp2l    Copies a file from the test file system to the linux file system
cp2fs   Copies a file from the Linux file system to the test file system
cd      Changes directory
pwd     Prints the working directory
history Prints out the history
help    Prints out help
Prompt >
```

In the screenshot below, there is no return for the commands 'ls' or 'cd'

```
student@student-VirtualBox: ~/fixFunctions/csc415-fileSystem-VtecFrancis-flxFunctions$ 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
Prompt > help
ls      Lists the file in a directory
cp      Copies a file - source [dest]
mv      Moves a file - source dest
md      Make a new directory
rm      Removes a file or directory
touch   Touches/Creates a file
cat     Limited version of cat that displace the file to the console
cp2l    Copies a file from the test file system to the linux file system
cp2fs   Copies a file from the Linux file system to the test file system
cd      Changes directory
pwd     Prints the working directory
history Prints out the history
help    Prints out help
Prompt > ls
Prompt > ls
Prompt > cd
Prompt > cd home
Prompt >
```

In this screenshot we attempt to make a directory display it and remove it, but there is no output. We believe it is a result of not all our functions being completed.



```
Machine View Input Devices Help
Terminal
student@student-VirtualBox: ~/fixFunctions/csc415-filesystem-VtecFrancis-fixFunctions
student@student-VirtualBox:~/fixFunctions/csc415-filesystem-VtecFrancis-fixFunctions$ 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
Prompt > help
ls      Lists the file in a directory
cp      Copies a file - source [dest]
mv      Moves a file - source dest
md      Make a new directory
rm      Removes a file or directory
touch   Touches/Creates a file
cat     Limited version of cat that displace the file to the console
cp2l    Copies a file from the test file system to the linux file system
cp2fs   Copies a file from the Linux file system to the test file system
cd      Changes directory
pwd     Prints the working directory
history Prints out the history
help    Prints out help
Prompt > ls
Prompt > ls
Prompt > ls
Prompt > cd
Prompt > cd hone
Prompt > pwd
Prompt > md helloWorld
Prompt > ls
Prompt > pwd
Prompt > rm helloWorld
Prompt > ls
Prompt > pwd
Prompt >
```

- ls** - Lists the file in a directory
- cp** - Copies a file - source [dest]
- mv** - Moves a file - source dest
- md** - Make a new directory
- rm** - Removes a file or directory
- touch** - creates a file
- cat** - (limited functionality) displays the contents of a file
- cp2l** - Copies a file from the test file system to the linux file system
- cp2fs** - Copies a file from the Linux file system to the test file system
- cd** - Changes directory
- pwd** - Prints the working directory
- history** - Prints out the history
- help** - Prints out help

**A table of who worked on which components:**

✓ = Worked On

Action Items	Cristian Garcia	Tsewang Sonam	Kurtis Chan	Francis Quang
Volume Control Block (block 0) is written	✓			
Free Space management is written/initialized		✓		
Root Directory is written and initialized			✓	✓
fs_mkdir()		✓		
fs_stat()		✓		
fs_isDir()		✓		
b_open()		✓		
b_close()		✓		
fs_delete()			✓	
fs_rmdir()			✓	
parsePath()			✓	
b_read()			✓	
b_seek()	✓			

fs_setcwd()	✓			
fs_getcwd()	✓			
fs_isFile()	✓			
b_write()				Assigned but didn't complete
fs_opendir()				Assigned but didn't complete
fs_readdir()				Assigned but didn't complete
fs_closedir()				Assigned but didn't complete
Writeup	✓	✓	✓	