JCJC File System Project Write Up

Team Github Link:
https://github.com/CSC415-2022-Fall/csc415-filesystem-AkiZhao614

Team Name: JCJC

Group Members:

Jiayi Gu
Student ID# 920024739
GitHub name:Jiayi079

Carmelo De Guzman
Student ID# 918749005
GitHub name: carmelodz

Jiaming Zhao
Student ID# 921891383
GitHub name:akizhao614

Congcheng Zeng
Student ID# 918327792
GitHub name:Congchengz

Command Line Table:

ls	can work
cp	can work
mv	can work
md	can work
rm	can work
touch	can work
cat	haven't tested yet
cp2l	haven't tested yet
cp2fs	haven't tested yet
cd	can work
pwd	can work
history	can work
help	can work

Description of our File System:

In this project, we designed a file system that essentially encompasses all the components we have learned about the Linux File System in order to make our own implementation. While not as robust and polished like the Linux file system, our file system can manage files and directories while also being able to display information about a file or directory. At the minimum, we were able to implement the basic functionalities of a file system that most operating system's file systems have today.

As for the three essential components, our file system consists of a volume control block (VCB), free space Bitmap, and a structure of our Directory Entry. In our VCB, we have variables that manages the blocks stored in the volume control block such as the number of blocks available, the number of freespace blocks available, as well as the magic number that is needed to initialize and format our "hard drive".

As for our free space tracking algorithm, we decided to use a Bitmap as the most effective and efficient way in managing storage space for files and directories. In a Bitmap, this stores a binary value in each block of either a 0 or 1, with 0 implying that this block space is free and can be allocated with data while a value of 1, implies that this block space is already in-use and cannot be allocated with data.

In terms of our Directory Entry Structure, we have information or metadata about each of the directory entries such as the file name, file size, file type, as well as the last access date and created-at date along with a short description of the file. These variables are the things needed to be able to access and read files in each directory entry.

In terms of the other components in our file system, we have several other files: b_io.c, fsInit.c, fsshell.c, mfs.c, fslow.h, and helperFunctions.c. The b_io.c file is used to manage and transfer individual files from the local file system (Linux) to our file system, and vice versa. The fsInit.c file contains two functions, *initFileSystem()* and *exitFileSystem()*, which are responsbile for starting/loading up our file system and closing it when the user exits out of the program. These two functions are called in the fsshell.c file. The fsshell.c file is the main driver program for the file system, where it creates an interactive user interface that can take command inputs from the user. This is essentially the "controller" that bridges the gap between the file system program and the user. As for the commands themselves, these functions come from the mfs.c file and the b_io.c file to perform any directory-based or file-based operations. The fsLow.h file contains two low-level LBA based read and write functions, *LBAwrite()* and *LBAread()*. The LBAwrite() function writes any data created from our file system from memory to the "hard disk". The LBAread() function reads data stored on the volume from the "hard disk" to memory. In the helperFunctions.c file, we have functions that pertain to our free space Bitmap, such as allocating free space blocks, setting and checking the bits, and converting the bits from the bitmap to bytes, as we are performing byte operations in the file system.

Issues We Had:

1. For our first issue, we were not able to get our File System's hexdump to be outputted in order to know if our SampleVolume has been formated and initialized correctly.
    a. We resolved this issue by re-doing the implementation of initilazing our free space Bitmap and initialzing our Root Directory. This was in the form of

adding additional helper functions, such checking and setting the bits inside our bitmap, and allocating free space for our root directory at the start of the free space block.

2. The second issue we had was that we had actual code implementation inside our .h header files, which caused several complie errors to appear whenever we did a “make” command.
    a. We resolved this issue by only putting function prototypes and declarations inside these header files, while the actual written implementation will be contained in a corresponding .c file under the same file name.

3. The third issue we had was that there was a variable called filePath defined in our Directory Entry structure. Since every Directory Entry in a directory has the same exact path, this would cause a lot of problems for us if we continued to use it.
    a. We resolved this issue by removing this from our Directory Entry structure and instead used a parent directory pointer that contains the file’s path. This also makes use of our parse path function parsePath that uses the last slash of the file’s directory path to parse the full path of the current directory.

4. For our fourth issue, we had to fit our File System’s VCB into one block.
    a. We resolved this by re-visiting our free space bitmap implementation and made some changes in allocating space for the VCB at initialization.

5. For our fifth issue, during the project’s Milestone One stage, we were not sure how to implement the code for initializing the VCB, root directory, and allocateFreeSpace_Bitmap() when we started the project. Since this was our first time implementing a file system, we found it difficult in deciding what to include in these functions and what things to consider when designing the VCB, allocateFreeSpace_Bitmap(), and Root Directory structure.

   a. We resolved this issue, we went back and re-watched the lecture videos and reviewed our notes that went over the basics of initialzing the VCB and setting up the bitmap. By doing so, along with outside research, we were able to understand each components of the file system, which then led us to successfully deciding what to include in all three of these structures.

6. For our sixth issue, we were not able to allocate free space correctly in our bitmap due to how we were storing and allocating them in blocks.
   a. We resolved this issue by re-doing our allocateFreeSpace_Bitmap() function implementation, taking into account that we had to start from block 2, rather than block 0 as the first two blocks in the Bitmap are allocated for the VCB and root directory.

7. For our seventh issue, we couldn't get any of the file-based commands to work such as touch, cat, cp2l, cp2fs, and mv, as they required a file to be created in our volume. Although we were able to paritally get the command mv to work, we couldn't fully move a file as we kept getting an "entry item is not of type file" error whenever we try to move an item in the directory.
   a. We tried to resolve this issue by making a create directory function, to handle creating directory entry files in our volume, however, we still couldn't get any of the file-based commands to work yet. Although, after looking more into our code, we were at least able to now use the touch and mv command to resolve half of this issue. For the touch command, we write about check the file name is include "." or not, if the file name is include "." such as test1.txt, it will set it type is file, if not will be set it be a directory. Also, we create a new function call fs_mkFile function to set the file type, is almost same with the fs_mkDir function.

8. For our eighth issue, we had an error for our b_open() function
   a. We resolved this issue by fixing the bugs for b_open() to make sure the cmd_touch() can work for creating a file. From our previous code

implementations, b_open() doesn't check the user input if it is of type dir or file name. That's the reason why we decided to add one more for-loop code to check if the pathname doesn't contain a dot, if it doesn't contain a dot, then the pathname will be considered a directory pathname. The file system will do nothing if the user input is of a directory name, so it will return 0 after it checks if it contains a dot.

9. For our ninth issue, we had some problems in the cmd_mv command. Since the mv command works by moving a file to a different directory location while maintaining the original file name, it should work the same way when moving to the same directory but renaming the file.
    a. We resolved this issue by adding one function inside of the fs_isDir() to check if the third argvec[2] is a directory name or a file name. It checks if the argvec[2]'s name contains a dot, which means it is a file type, otherwise, it should be considered a directory type. Also, the location we put this if-statement to check the "dot" is important, we should check it at the beginning of the fs_isDir() function to make sure if it is a file type, then we don't have to run anything inside of the fs_isDir().
    b. Then, we can use check if fs_isDir(argvec[2]) → the third name/parameter of the user input, is true or false inside our cmd_mv function. The renaming process in the same dir is the easier one, since we already did this part first in the previous step. After we check that the third name/parameter is not a dir name, we basically do the same thing as the cmd_cp function.
    c. After we check the third name is a dir name, we have some additional steps to work on. Firstly, we check if the dir name exists or not in the volume(this part we've already done in fs_isDir()). Secondly, we keep the original directory path, since we have to go back to the original path to delete the file after we finish copying the file. Third, we change to the new dir (by using fs_setcwd()). Fourth, we copy the file to the new dir by using the cmd_cp() function. Fifth, we go back to the previous directory and delete the original file (we kept the previous cwd before, to go back to the original path by using fs_setcwd() agian).

10. For our tenth issue, we tested the code and find out the program sometimes is unstable, most time the program will work fine, but rare case when we try using the command mv will crash the program.
    a. we try to find out the problem and locate it when we have 2 the same name file in different directory, when we use mv to move a same name file to other directory and inside also have the same name file it will be crash the program. We fix it by create a checkContainFile() to check those two file have the same name or not, if there no same file name, the file will move to the directory we want. If there are two same name file, it will print out a error and return to the main interface.
    b. But the crash is not fully fix, After we make clean and re-run the program, there will still crash when we move the same name file to other directory and they have same name file will crash. But if we use the command touch to create a new test1.txt file in the target directory, we back to the previous directory and use mv to move the file name test1.txt to that directory, it will not crash the program then print out the error message and return to the main interface.

11. For our eleventh issue, we are still finding out the problem why using mv still will crash the program.
    a. we fix it by add some code in the fsshell.c file, we add the code into the cmd_mv function and check the argv[2] is null or not, if argv[2] is null it will print our a remind message and give a example how to type the command line and return back to the main interface.

How Our Driver Program Works:

The implementation of our driver program can be found in the fsShell.c file, which contains important functions needed to format and initialize our file system. Once the file system is loaded and ready to be used, the shell program will display a command line terminal that can accept user input as valid command line arguments. These arguments will

then be processed by the file system which is done by completing file operations requested by the user before displaying the result back into the console. The main driver function starts partitioning the volume by calling startPartitionSystem(), which takes in a file name, volume size, and block size as a parameter by extracting commands from the command line arguments. Once that is finished, the driver program calls the initFileSystem() function to re-iniltialize the bitmap that is used for our free-space management, which indicates that our volume has been formatted and initilized already. However, if our volume has not been formatted, then the main driver program will initialize the volume along with initializing our free-space Bitmap and root directory before writing this volume back into our disk.

After formatting and initialzing our volume, the driver program will then switch to running an interactive shell terminal that will prompt the user to enter commands supported by the file system. The file system will then process the file or directory entered by the user by performing I/O operations before outputting the information back into the console for the use to see. The supported commands are what follows:

- ls - Lists the current file(s) in the directory
- cp - Copies a file from a source location
- mv - Moves a file to a destination location
- md - Makes a new directory
- rm - Removes a file or directory
- touch - Creates a file
- cat - 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 to a specified directory
- pwd - Prints the current working directory
- history - Prints out recent command history
- help - Prints out help options

When the user wants to terminate the shell to exit the program, the main driver function will call the exitFileSystem() function to exit the shell along with calling the closePartitionSystem() to close our file system. To summarize, our main driver program is a simple file system that can perform basic tasks and handle files in a volume similar to that of the Linux operating system.

Screenshots of Commands:

```
student@student-VirtualBox:~/csc415-filesystem-AkiZhao614$ 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
*** VCB STATUS ***
number of blocks: 19531
block size: 512
vcb block count: 1
freespace block count: 5
first free block index: 98

Prompt > exit
System exiting
student@student-VirtualBox:~/csc415-filesystem-AkiZhao614$
```

Execution Screenshot:

Compilation Screenshot:

```
student@student-VirtualBox:~/csc415-filesystem-AkiZhao614$ 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.
mfs.c: In function ‘fs_readdir’:
mfs.c:878:36: warning: return from incompatible pointer type [-Wincompatible-pointer-types]
             return (dirp->dirEntry + check_de_index);
                    ~~~~~~~~~~~~~~~~~~~~^~~~~~~~~~~~~~
gcc -c -o b_io.o b_io.c -g -I.
gcc -o fsshell fsshell.o fsInit.o mfs.o b_io.o fsLow.o -g -I. -lm -l readline -l pthread
student@student-VirtualBox:~/csc415-filesystem-AkiZhao614$
```

ls - Lists the current file(s) in the directory

```
student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$ 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
*** VCB STATUS ***number of blocks: 19531
block size: 512
vcb block count: 1
freespace block count: 5
first free block index: 150
Prompt > ls

h
foobar
testDemo
barfoo
fsTest
Prompt > exit
System exiting
student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$
```

cp - Copies a file from a source location

```
student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$ 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
*** VCB STATUS ***number of blocks: 19531
block size: 512
vcb block count: 1
freespace block count: 5
first free block index: 150
Prompt > ls

h
testDemo
fsTest
Prompt > cd testDemo
find: testDemo
Prompt > ls

testDemo.txt
foobar.txt
Prompt > cp testDemo.txt testDemo123.txt
Creating a file, filename: testDemo.txt
[mfs.c -- mkFile] name already exsist in directory, you may want to copy file
Creating a file, filename: testDemo123.txt
freespaceLocation: 150
Prompt > ls

testDemo.txt
foobar.txt
testDemo123.txt
Prompt > cp foobar.txt foobar1999.txt
Creating a file, filename: foobar.txt
[mfs.c -- mkFile] name already exsist in directory, you may want to copy file
Creating a file, filename: foobar1999.txt
freespaceLocation: 163
Prompt > ls
```

mv - Moves a file to a destination location

```
student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$ 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
*** VCB STATUS ***number of blocks: 19531
block size: 512
vcb block count: 1
freespace block count: 5
first free block index: 176
Prompt > ls

h
foobar
barfoo
fsTest
testDemo
Prompt > cd foobar
find: foobar
Prompt > ls

testfile12345.txt
Prompt > mv testfile12345.txt testfile1999.txt
check is not Dir
Creating a file, filename: testfile12345.txt
[mfs.c -- mkFile] name already exsist in directory, you may want to copy file
Creating a file, filename: testfile1999.txt
freespaceLocation: 176
testfile12345.txt : testfile12345.txt was removed
Prompt > ls

testfile1999.txt
Prompt > pwd
./foobar
Prompt > exit
System exiting
student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$
```

md - Makes a new directory

```
student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$ 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
*** VCB STATUS ***number of blocks: 19531
block size: 512
vcb block count: 1
freespace block count: 5
first free block index: 72
Prompt > md foobar
dir_location in mkdir: 72
Creating directory foobar success
Prompt > md barfoo
dir_location in mkdir: 85
Creating directory barfoo success
Prompt > md testDemo
[mfs.c -- mkdir] name already exsist in directory
Prompt > md fsTest
dir_location in mkdir: 137
Creating directory fsTest success
Prompt > ls

h
foobar
testDemo
barfoo
fsTest
Prompt > exit
System exiting
student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$
```

rm - Removes a file or directory

```
student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$ 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
*** VCB STATUS ***number of blocks: 19531
block size: 512
vcb block count: 1
freespace block count: 5
first free block index: 150
Prompt > ls

h
foobar
testDemo
barfoo
fsTest
Prompt > rm foobar
find: foobar
find: foobar
foobar : foobar was removed
Prompt > ls

h
testDemo
barfoo
fsTest
Prompt > rm barfoo
find: barfoo
find: barfoo
barfoo : barfoo was removed
Prompt > ls

h
testDemo
fsTest
Prompt > exit
System exiting
student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$
```

touch - Creates a file

```
student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$ 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
*** VCB STATUS ***number of blocks: 19531
block size: 512
vcb block count: 1
freespace block count: 5
first free block index: 72
Prompt > ls

h
testDemo
fsTest
Prompt > cd testDemo
find: testDemo
Prompt > touch testDemo.txt
Creating a file, filename: testDemo.txt
freespaceLocation: 72
Prompt > ls

testDemo.txt
Prompt > touch foobar.txt
Creating a file, filename: foobar.txt
freespaceLocation: 85
Prompt > ls

testDemo.txt
foobar.txt
Prompt > exit
System exiting
student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$
```

cat - Displays the contents of a file

-We cannot locate our stored created file in our file system, so we do not have a screenshot for this command.

cp2L - Copies a file from the test file system to the Linux file system

-We cannot locate our stored created file in our file system, so we do not have a screenshot for this command.

cp2fs - Copies a file from the Linux file system to the test file system

-We cannot locate our stored created file in our file system, so we do not have a screenshot for this command.

cd - Changes to a specified directory

```
student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$ 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
*** VCB STATUS ***number of blocks: 19531
block size: 512
vcb block count: 1
freespace block count: 5
first free block index: 150
Prompt > ls

h
foobar
testDemo
barfoo
fsTest
Prompt > cd testDemo
find: testDemo
Prompt > pwd
./testDemo
Prompt > cd ..
find: ..
Prompt > ls

h
foobar
testDemo
barfoo
fsTest
Prompt > cd foobar
find: foobar
Prompt > pwd
./foobar
Prompt > exit
System exiting
student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$
```

pwd - Prints the current working directory

```
student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$ 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
*** VCB STATUS ***number of blocks: 19531
block size: 512
vcb block count: 1
freespace block count: 5
first free block index: 150
Prompt > ls

h
foobar
testDemo
barfoo
fsTest
Prompt > cd barfoo
find: barfoo
Prompt > pwd
./barfoo
Prompt > cd ..
find: ..
Prompt > ls

h
foobar
testDemo
barfoo
fsTest
Prompt > cd fsTest
find: fsTest
Prompt > pwd
./fsTest
Prompt > exit
System exiting
student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$
```

history - Prints out recent command history

```
Prompt > touch testDemo1234.txt
Creating a file, filename: testDemo1234.txt
freespaceLocation: 163
Prompt > history
ls
md foobar
md barfoo
ls
cd testDemo
ls
pwd
rm foobar1999.txt
ls
cd ..
ls
cd barfoo
ls
pwd
cd ..
ls
history
ls
cd barfoo
ls
touch testDemo1234.txt
history
Prompt > exit
System exiting
student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$
```

help - Prints out help options

```
student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$ 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
*** VCB STATUS ***number of blocks: 19531
block size: 512
vcb block count: 1
freespace block count: 5
first free block index: 150
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 > exit
System exiting
student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$
```

Hexdump Screenshots:

-At File System initialization (no directories or files)

```
student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$ Hexdump/hexdump.linux --start 1 --count 2 SampleVolume
Dumping file SampleVolume, starting at block 1 for 2 blocks:

000200: 4A 43 4A 43 00 00 00 00  00 02 00 00 00 00 00 00 | JCJC............
000210: 4B 4C 00 00 00 00 00 00  01 00 00 00 05 00 00 00 | KL..............
000220: 55 00 00 00 00 00 00 00  06 00 00 00 00 00 00 00 | U...............
000230: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000240: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000250: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000260: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000270: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000280: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000290: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000300: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000310: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000320: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000330: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000340: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000350: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000360: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000370: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000380: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000390: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000400: FF FF FF FF FF FF FF FF  FF FF 1F 00 FC 7F 00 00 | ◆◆◆◆◆◆◆◆◆◆..◆..
000410: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000420: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000430: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000440: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000450: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000460: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000470: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

```
000400: FF FF FF FF FF FF FF FF  FF FF 1F 00 FC 7F 00 00 | ◆◆◆◆◆◆◆◆◆◆..◆..
000410: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000420: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000430: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000440: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000450: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000460: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000470: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000480: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000490: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000500: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000510: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000520: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000530: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000540: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000550: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000560: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000570: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000580: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000590: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$
```

-The file system when there are directories and files stored in volume

```
ent@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$ Hexdump/hexdump.linux --start 1 --count 2 SampleVolume
ing file SampleVolume, starting at block 1 for 2 blocks:

00: 4A 43 4A 43 00 00 00 00  00 02 00 00 00 00 00 00 | JCJC............
10: 4B 4C 00 00 00 00 00 00  01 00 00 00 05 00 00 00 | KL..............
20: CA 00 00 00 00 00 00 00  06 00 00 00 00 00 00 00 | ◆...............
30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

00: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

00: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
10: FF FF FF FF FF FF FF FF  FF 03 00 00 00 00 00 00 | ◆◆◆◆◆◆◆◆◆.......
20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

```
000400: FF FF FF FF FF FF FF FF  FF FF 1F 00 FC 7F 00 00 | ◆◆◆◆◆◆◆◆◆◆..◆..
000410: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000420: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000430: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000440: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000450: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000460: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000470: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000480: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000490: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000500: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000510: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000520: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000530: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000540: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000550: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000560: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000570: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000580: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000590: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

student@student-VirtualBox:~/test/csc415-filesystem-AkiZhao614$
```

Core File System Functions:

Mfs.c (Directory Management Operations):

fs_mkdir() - This function initializes and creates a new directory with the given name from the last token of a passed-in path and updates the parent directory's information on the disk

fs_rmdir() - This function removes the directory based on the given path. If the directory is empty or does not exist, then an error message will be printed out indicating the function has failed.

fs_opendir() - This function opens a directory from a given directory name and stores the data. It returns a pointer to the structure that stores the information of the directory if the directory exists. If the directory does not exist, then print out an error message.

fs_readdir() - This function returns a pointer to the structure that stores information for each directory entry of a directory. This function then returns NULL once it reaches the end of the directory.

fs_closedir() - This function closes the directory by deallocating the stored data of the opened directory in memory. And it also resets the opened directory pointer to NULL.

fs_getcwd() - This function creates a temporary working directory and is malloced. It then makes a copy of the current working directory and loops backward by getting to the parent directory until the root directory is reached, which will then return and print the full path of the current working directory.

fs_setcwd() - This function finds the directory based on the given path passed-in and deallocates the original cwd, which will then set this directory, if found, as the new current working directory (CWD).

fs_isFile() - This function will look for the parent directory and checks the directory entry list for all stored entries. If there is an entry that has the same name, then it returns true, meaning the entry is of type File.

fs_isDir() - This function will look for the specified directory, if it exists, based on the path name specified, and returns true, meaning the item is of type Directory.

fs_delete() - This function goes to the parent directory from the path, and checks if the file exists. If it does, then it marks this space as free in the parent directory's corresponding index and releases it as freespace. This function deletes a file or directory.

B_io.c: (Individual File I/O Operations):

b_open() - This function returns a free file descriptor to an individual file, in which the file descriptor will be used by other file I/O operations to refer to that specific file.

b_read() - This function reads data from the volume from the start location of the entry. It reads up to the buffer size of bytes or the remaining bytes in the data from the file into the buffer found in the fcb structure.

b_write() - This function writes up the whole passed in buffer into our buffer in fcb, and we concatenate the data so that they are actually connected in the memory.

b_seek() - This function returns the offset location in bytes from the beginning of the file, by taking the file descriptor of the file when the b_write and b_read function is used.

b_close() - This function closes the file descriptor so that it no longer refers to any file stored in memory. Then it will free the data that was malloc() in the fcb that was associated with that file and sets the buffer pointer to NULL.

Core File System Structures:

```
//Struct for our VCB
typedef struct VCB{
   uint64_t magicNumber;              // integer variable for magic number, useful
                                      // when opening files based on their
                                      // file Signature, and also for hex dumps
   uint64_t blockSize;                // variable bytes for the size of each
                                      // blocks in VCB (512 bytes)
   uint64_t numberOfBlocks;           // integer variable for the number of blocks in VCB
   unsigned int VCB_blockCount;       //integer variable for starting position of free
                                      // block space VCB (2560 bytes)
   unsigned int freeSpace_BlockCount;      // used for check when it is not the first run
   uint64_t current_FreeBlockIndex;        // used for check when it is not the first run
```

```
    uint64_t location_RootDirectory;    // can be calculated by adding the other two counts


} volume_ControlBlock;



//Struc for our Directory Entry
struct Directory_Entry
{
    unsigned short d_reclen;                //length of this record
    unsigned char fileType;                 // 0->dir 1->file
    unsigned char dirUsed;                  //to check if any Dir is in use, 0 is free, 1 is using
    uint64_t dir_Location;                  //integer variable to store file location
    size_t fileSize;                        //variable for file size
    char file_name[MAX_NAME_LENGTH];    //character variable to store file name

    time_t create_time;                     //variable for file create time
    time_t last_access_time;                //variable for when you access the file's time
    time_t modified_time;                   //variable for when you modify the file's time
    // char comment [300];                  // comment for the file

}Directory_Entry;
```