File System

Team CN2S

Caimin Rybolt-O'Keefe 922914119, Nathan Rennacker 921348958, Spencer Holsinger 918140540, Suzanna Li 917694159

Github Repository: [teamCN2S]

Index

Description of File System

Volume Control Block

For our volume control block, we decided on the struct:

```
typedef struct volumeControlBlock
{
        int totalBlock;         //total number of blocks
        int freeBlock;          //number of free blocks
        int blockSize;          //size= 512
        int rootLocation;       //location of root
        int mapLocation;        //location of free space map
        int initNumer;          //the number to check if VCB initialized
} VCB;
```

There is a volume control block pointer, vcbPointer, that is outside the struct that we will use when we want to access the data within the struct. The following table describes each different data.

totalBlock	4 bytes	how many blocks there are
freeBlock	4 bytes	how much free space is available
blockSize	4 bytes	how big each block can hold
rootLocation	4 bytes	This is where the root and its location
mapLocation	4 bytes	bitmap location in the free space
initNumber	4 bytes	We check to see if the VCB has already been made or not. If initNumber != the magic number then the VCB will be initialized. If the two numbers do match, the VCB will not be initialized.

VOLUMECONTROL.C

This file is about the volume control and the information that it holds.
The following table talks about what is what.

initVolumeControl	Initializes the volume control block if the magic number does not match the initNumber
getTotalBlock	Gets the total block size
getBlockSize	Get the size of each block
getFreeBlocks	Gets the amount of free blocks left
getMapLocation	Gets the free map location, which is different than the root location

Free Space Structure

We decided on a bitmap for managing our free space. The map's dimensions are based on the project's maximum size, which is 19,531 blocks of 512 bytes. This is equivalent to 19,531 bits needed for tracking that space (or 2,442 bytes). For this, we opted to use an unsigned integer array to represent the mapping. Given that each unsigned integer occupies 4 bytes, an array of 611 integers is required to accommodate all the bits.

There are a few surplus bits, which could create issues if they were accessible. To resolve this, a maximum block limit of 19,531 was implemented in bitmap.c to guarantee that the total block count is not surpassed, and we do not attempt to write space not allocated for the bitmap (in our case it would be the directory space).

To locate available space, we employed a hybrid iterative and recursive method. Initially, the bitMap is scanned until an open space is identified, then a recursive function is utilized to verify if the free space is large enough (either recursion or iteration would work here but most importantly the position is incremented by the last filled bit found if there isn't enough space, to make the search more efficient). If it is big enough, the bit position is returned.

There is also an extent system implemented within the bitmap that helps for non-contiguous allocation (if a file is too big or a file needs to be edited and there isn't enough space for contiguous allocation). This system has some logic issues currently when deleting files, but most of the functionality is there.

BITMAP.C

This file is for the functionality of the free space and related functionality.

writeBlocks	Writes to a block with given position and the given length
printMap	Prints the free space map
clearBlocks	Clears the bit and its information on the file and extent
findEmptyBlocks	Find n length empty blocks
recursiveCount	Goes through set number of bits backwards and if it encounters a filled bit, it will return position of filled bit
setBit	Takes value from map and uses OR operation to set a specific bit
clearBit	Clears the bit with AND operation of a specific bit
findBit	Checks specific blocks in the bit is set
initMap	Initializes the free space map
freeMap	Frees the entire free space map
allocateFirstBlocks	Allocates the first blocks needed for any memory that needs to be written to the LBA
allocateAdditionalBlocks	Same as allocate first blocks but for anything after the initial allocation

Directory System

Directory entry

```
typedef struct directoryEntry {
        time_t date;
        uint32_t fileSize;
        extent extentLocations[MAX_EXTENTS];
        short location;
        bool isDirectory;
        char name[37];
} directoryEntry;
```

The organization was carefully considered so that a directoryEntry would take up all possible space, leaving no bytes unused due to byte boundaries.

date	8 bytes	To store our files date we are utilizing the C library function time_t
fileSize	4 bytes	This is the overall size of the file stored as unsigned 32 bit int
extentLocations	12 bytes	Pointer to Extent struct
location	2 bytes	We're using extents to track file locations with primary, and possible secondary and tertiary, extents for storing and retrieval.
isDirectory	1 byte	IsDirectory will mark a file as a directory
name	37 bytes	We're designating 27 Bytes towards storing the file name in a char array

Extent

```
typedef struct extent{
   short blockNumber;  // block start location
   short count;  // length of the entry
} extent;
```

blockNumber	2 bytes	Block position in the volume
count	2 bytes	Number of blocks it takes up in the volume

DIRECTORYENTRY.C

The following table is about the functions in the directoryEntry.c file. The directoryentry.c file takes care of the creation of entries, directories and other related functions.

createEntry	Short function to quickly create a new directory entry and everything that has to do with a single directory entry.
createDirectory	Checks for any existing directory of the newDirecName within the parent directory. If dup is found returns error because it exists already. If there are no errors with adding the new directory, then a directory will be added into the parent directory. The parent directory will be updated to the new size and any relevant information that has changed.
initRootDirectory	Initializes the root directory
readDirectory	Help us debug anything that may of gone wrong or is not showing up

B_IO.C

The following table is the b_io.c functions and what they are supposed to do.

```
typedef struct b_fcb {
   char* buff;         // holds the open file buffer
   int index;          // holds the current position in the buffer
   int buflen;         // holds how many valid bytes are in the buffer

   int lbaPos;     // current block position. allows working with main +
                      extents
   int flagRDWR;   // flag showing
   int bytesRead;
   int blocksAtMainLoc;

   struct fs_stat* fileInfo;
} b_fcb;
```

b_init	Initializes the file control blocks (FCB) array
b_getFCB	Returns a free FCB index, if possible
b_open	Opens a file, with the appropriate flags that the user has given, if the file exists and there is a free FCB. Can also create a file if the flag O_CREAT is provided. Will not open a file if given incorrect flags
b_seek	Changes the active position in a file to the source and its offset. Clamps to beginning / end of file if needed.
b_write	Writes to a file
b_read	Reads from a file
b_close	Closes a file

MFS.C

The following table is about the functions and variables in mfs.c.

curWorkingDir	This is a macro that gets the current working directory and its information for us
parsePath	Parses given pathname and return the corresponding directory entry
fs_readdir	Reads the directory into our buffer
fs_mkdir	Makes a new directory within the current directory
fs_rmdir	Removes an empty directory, if the directory is not empty then it sends an error, and the directory will not be deleted
fs_opendir	Opens the directory, meaning we go into the directory and now it is the current working directory. The previous directory is not its parent
fs_closedir	Closes the current working directory and returns to its parent directory
fs_getcwd	Gets the current working directory
fs_setcwd	Set a given directory, if it exists, to the current working directory
fs_isFile	Checks if the element in the directory is a file
fs_isDir	Checks if the element in the directory is a directory
fs_delete	Deletes a file in the current working directory if the file exists. If the file

	does not then an error is thrown.
fs_stat	Gets the status of the user given path and copies the information to a buffer that acts as the middle man, so that the user does not directory access the information

PARSEPATH.C

parsePathInfo

```
Typedef struct {
        Int location;
        Int size;
} parsePathInfo;
```

isAbsolutePath	Sets pathname[0] to “/” and returns “/”
isSingleSlash	Compares the pathname and if it is equal to “/”
isSingleDot	Compare the pathname and if it is equal to “.”
isDoubleDot	Compare the pathname and if it is equal to “..”
setInitialDirectoryInfo	Sets the initial directory information
getFirstToken	Gets the first token from the given pathname
handleSpecialCases	Handles any special cases for path names such as single slash, single dot, and double dot, if they are the first and only part of the path
handleDotDotToken	This handles the “..” token and updates the parePathInfo as well as the last found entry
findTokenInEntryArray	Searches for the given token within the array of directory entries and updates the parsePathInfo structure. It will also update the last found entry as well.
parsePath	Takes in pathname and return a

	pointer to a directoryEntry, if the pathname does not exist then return NULL

Table of Who worked on What

Name	**Milestone 1**
Caimin	Root Directory
Nathan	Free Space Management
Spencer	Root Directory
Suzanna	Volume Control Block

Name	**Milestone 2 + Stuff**
Caimin	Lower-level directory entry functions: createDirectory(), createEntry()
Nathan	parsepath(), fs_readdir(), fs_mkdir(), fs_opendir(), fs_closedir(), fs_getcwd(), fs_setcwd()
Spencer	fs_isFile(), fs_isDir(), fs_stat()
Suzanna	fs_delete(), fs_rmdir()

Name	**Milestone 3**
Caimin	b_open(), b_close(), b_seek(), b_read(), b_write()

Nathan	Debugging, command testing, cmd_mv
Spencer	Command testing, writeup
Suzanna	Command testing, writeup

Different Approaches

Below is a table of different approaches with each milestone.

Milestone 1

Name	How we approached our own section
Suzanna	So my section was on the volume control block. I looked back at the class recordings and read the book on what is a volume control block. After I got a better understanding of what a control block is, I wrote and coded what we decided will be in the volume control block with previous assignments. After that, I asked for help with putting all the different parts together.
Caimin	With the creation of the root directory, I began researching how it was done and how it was described in the milestone guide. Everytime I came on a part I was uncertain about, I would ask other group members their opinion on some section of the implementation or how they interpreted instructions or research information. Using the knowledge and opinions of my team allowed me to quickly write and finish this section.
Nathan	In order to begin planning for the bitmap I had to understand the way bit math worked. I did a lot of research as prep for making some of the functions for the bitmap. Using AND and OR operations with logical shifts was hard to get my head around but once I understood that the rest was

	fairly easy. While coding the bitmap I wanted to be efficient when checking for enough space, and I thought about it for a while and came up with a recursive solution that returns the length so that that number of bits can be skipped.
Spencer	Working with Caiman regarding the root directory we discussed past lectures as well as online resources regarding the necessary components to instantiating the root directory

Milestone 2

Name	How we approached our own section
Suzanna	I worked on fs_delete and fs_rmdir. I asked for a lot of help from my team members because I was confused on what was what. Nathan and Caimin are very knowledgeable and helped me with the issues I encountered.
Caimin	Creating directories came down to copying what I did for creating the root, and turning it into a more generalized function. I would then later go back and add in more functionality based on the development of the functions in mfs.c and pathparse.c, and to fix cases I had not identified myself.
Nathan	In order to understand the different facets of parsepath, I needed to create all the different cases in which it would be used. Thankfully we had a pretty good endpoint for the function to return with our solid definition of

	directoryentry which allowed for using that struct in a massive amount of ways. The actual use cases required a lot of understanding how paths could be passed to functions and how they interacted with the current working directory. Once I had that basis down, I finally started coding the parsepath functionality. And it didn't go as smoothly as expected, but once it was done I was glad I spent so long on it, since it was handy in basically every function I worked on afterwards.
Spencer	Having Nathan explain his parse path functionality I was able to utilize the directory entry entity that it created in order to implement the isDir and isFile checks as well as fs_stat. We revisited stat several times as we often found there was more to initialize as well as to fix memory allocation issues

Milestone 3

Name	How we approached our own section
Suzanna	I approached this write-up with what we had already done. I asked members of our team on specific functions and their purpose for the tables. I also created tables for everyone so that it is easier for everyone to add their inputs when they get the chance.
Caimin	I basically copied and modified what I had for assignment 5 for the b_read() and b_close() functions. While they were very similar, they were extremely different from the original assignment

	in many aspects. I knew that in b_write() we would have to allocate more space for a file if the position was the end of the file, otherwise it would overwrite until it reached the end where it would continue to write. I would also have to update the file size of the file's directory entry every time the file was written to past its original file size. I also understood that b_read() and b_write() could have to read from extents and not just the main location, which added quite a bit of logic and checks.
Spencer	Once the logic for each function was written we began testing commands slowly. First were directory related commands as our directory related code came before our b_io code. Going through each command I made sure to focus on both the basics of each as well as edge cases. For this portion Nathan, Caiman and myself were in a discord call and would bring up found bugs and then work through both the why of the bug and the how of fixing it. Once the bugs were worked through I compiled the list of screenshots for the commands

Issues With the File System

Milestone 1 Issues

Name Individual / Team	Issues and resolutions
Suzanna: Individual Issues	The number one issue I had was how do I take in the bitmap and the Directory structure into the Volume Control Block. I asked the team how to do that and they answered my question. Another issue I had was how to use the functions for the bitmap and the directory because when I compiled the files it gave me an unidentified element error. The team figured it out and it was because I did not add the bitmap object into the make file.
Nathan: Individual Issues	Extents: While the idea for extents seemed, at face value, to mesh well with the existing structure planned for free space mapping. Actually programming the extent logic proved to be quite the challenge. After a lot of testing we finally managed to get them working but it took quite a lot of deleting blocks of code and restarting. (Deleting parts of files still needs work as far as extents go) Object files: When we finally started to do group testing, we couldn't figure out why some function calls weren't working. Turns out we had to place the object files we needed in the Makefile

	to ensure that they were being built so that the init function calls could take place.
Caimin: Individual Issues	The largest issue I faced was trying to determine how much space should be used by a directoryEntry, what data type the data members would be represented by, and how to efficiently move around the data members so there was no bloating of an entry's size due to byte boundaries. After determining proper data types, changes were made to organize the bytes they represented to ensure no bloating through trial and error of checking the size.

Milestone 2 Issues

Name Individual / Team	Issues and Resolutions
Suzanna: Individual Issues	Some of the issues I had with the two functions I was doing were how to free the location on the bitmap. For example when a file is deleted I can just set the name to NULL but I also need to free the location it was taking up. This issue was revolved when I asked about how to free the bitmap.
Nathan:	Making parsepath function correctly was the single most important part of this milestone as most of the other

	functionality required for this milestone needed parsepath as a base. Starting from a blank script was really challenging and it took a lot of time just to come up with the logic for how the function would be laid out.
Caimin: Individual Issues	Making directories proved to be a lot more complicated than it initially seemed due to the need to acknowledge all of the issues that came with using extents and resizing of the parent directory if needed. It created logic that was a little difficult to work with in multiple areas, but ultimately was solved through dedication.
Spencer: individual issues	Despite using the commands regularly I found myself unable to grasp the actual logic with which the commands operated as when using them I was more focused on the result. The link that was provided in the read me as well as the man pages were instrumental to solving this issue

Milestone 3 Issues

Name Individual / Team	Issues and Resolutions
Suzanna:	For our project we used many different branches. I was a bit confused about what will happen when we merge all

	the different branches back into the main. This issue was resolved when we discussed how to merge it together.
Nathan:	cmd_mv: There was a lot of edge case checking in mv that required test case problem creation and checking. I definitely couldn't have done all the QA for this function alone.
Caimin: Individual Issues	While reading and writing files are very similar, they proved to have a lot more added logic that was very different. The largest issue was adapting the functions to work under the various cases and various inputs, as a user could attempt to: truncate a file they can not write to, open a directory, create a file that already exists, and many more cases. Assimilating all this with the various other components, such as allocating and deallocating via bitmap, and creating entries under various parent directories, proved to be very difficult. Solutions were crafted after much trial and error, and checking each possible edge case considered for every function involved in file creation and editing.
Spencer: Testing issues	Testing of the shell commands was difficult as oftentimes the return was either to write a new prompt or just a blank line that would require the make

	run to be force closed. I injected print statements throughout the code to find where it was failing however those were often never reached. We found that resetting the volume block when compiling would fix this issue.

Other Issues

Name Individual / Team	Issues and Resolutions
Everyone	VSCode just crashing or being very slow
Suzanna - Individual	I had some trouble when I was merging different branches and there were a lot of conflicts. The resolution was for me to go through everything and see what each different conflict was. It was resolved in the end.
Caimin - Individual	An issue I often encountered was having to go back and update old definitions and implementations as I would later need to accommodate more edge cases or new code produced by myself or other members of the group. Another issue was identifying specific issues as the code became longer and more difficult to navigate. This led to sudden printfs everywhere in an attempt to read certain variables at various parts of any particular process.

How Driver Program works

Our driver provides a command-line interface to interact with our custom file system. It supports various file and directory operations through a set of supported commands. The program employs the GNU Readline library to enable command history and facilitate user input.

Sections of the File System Driver Program

The File System Driver program consists of the following key sections:

1. **main function**: The main function initializes the file system and partition system using the provided volume filename, volume size, and block size. It then enters an infinite loop that reads user input, processes commands, and updates the command history. The loop terminates upon receiving the "exit" command, which triggers the closure of the file system and partition system.

2. **Dispatch table**: The dispatch table is an array of structs (struct dispatch_t) that maps each command name to its corresponding function and description. This table allows the program to call the appropriate function based on user input.

3. **processcommand function**: This function accepts a user command as input, tokenizes it into an array of strings, and searches the dispatch table to invoke the corresponding command function.

4. **Command functions**: The program implements several command functions responsible for executing respective commands in the file system. These functions include:

    - **cmd_create**: Create a new file
    - **cmd_delete**: Delete an existing file
    - **cmd_open**: Open an existing file
    - **cmd_close**: Close an open file
    - **cmd_read**: Read data from an open file

- **cmd_write**: Write data to an open file
- **cmd_seek**: Move the read/write pointer in an open file
- **cmd_cp2l**: Copy a file from the test file system to Linux
- **cmd_cp2fs**: Copy a file from Linux to the test file system
- **cmd_cd**: Change the current working directory
- **cmd_pwd**: Print the current working directory
- **cmd_history**: Display command history
- **cmd_help**: Display help for available commands

Screenshots of Commands listed in README

ls - Lists the file in a directory

```
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$ make run fsshell
./fsshell SampleVolume 10000000 512
File SampleVolume does not exist, errno = 2
File SampleVolume not good to go, errno = 2
Block size is : 512
Created a volume with 9999872 bytes, broken into 19531 blocks of 512 bytes.
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Initializing File System with 19531 blocks with a block size of 512
Prompt > ls

Prompt > md dir
Prompt > touch newfile
Prompt > ls -l -a

D         3584   .
D         3584   ..
D         3584   dir
-            0   newfile
Prompt > exit
System exiting
make: 'fsshell' is up to date.
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$
```

cp - Copies a file - source [dest]

```
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$ make run fsshell
./fsshell SampleVolume 10000000 512
File SampleVolume does not exist, errno = 2
File SampleVolume not good to go, errno = 2
Block size is : 512
Created a volume with 9999872 bytes, broken into 19531 blocks of 512 bytes.
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Initializing File System with 19531 blocks with a block size of 512
Prompt > touch newfile
Prompt > md cpdir
Prompt > ls -l -a

D         3584   .
D         3584   ..
-            0   newfile
D         3584   cpdir
Prompt > cp newfile cpdir/newlocation
Prompt > cd cpdir
Prompt > ls

newlocation
Prompt > exit
System exiting
make: 'fsshell' is up to date.
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$
```

mv - Moves a file - source dest

```
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$ make run fsshell
./fsshell SampleVolume 10000000 512
File SampleVolume does not exist, errno = 2
File SampleVolume not good to go, errno = 2
Block size is : 512
Created a volume with 9999872 bytes, broken into 19531 blocks of 512 bytes.
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Initializing File System with 19531 blocks with a block size of 512
Prompt > md dir1
Prompt > md dir2
Prompt > cd dir1
Prompt > touch fileToBeMoved
Prompt > cd ..
Prompt > mv dir1/fileToBeMoved dir2
Prompt > cd dir2
Prompt > ls -l -a

D         3584   .
D         3584   ..
-            0   fileToBeMoved
Prompt > cd ..
Prompt > cd dir1
Prompt > ls -l -a

D         3584   .
D         3584   ..
Prompt > exit
System exiting
make: 'fsshell' is up to date.
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$
```

md - Make a new directory

```
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$ 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 > md dirToBeRemoved
Prompt > ls -l -a

D          3584    .
D          3584    ..
D          3584    dirToBeRemoved
Prompt > rm dirToBeRemoved
Prompt > ls -l -a

D          3584    .
D          3584    ..
Prompt > md newDirectory
Prompt > ls -l -a

D          3584    .
D          3584    ..
D          3584    newDirectory
Prompt > exit
System exiting
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$
```

```
Prompt > md testdir
Prompt > ls

testdir
Prompt > ls -l -a

D          3584    .
D          3584    ..
D          3584    testdir
Prompt >
```

```
Prompt > md dir1
Prompt > cd dir1
Prompt > md dir2
Prompt > cd dir2
Prompt > md dir3
Prompt > cd dir3
Prompt > ls

Prompt > pwd
/dir1/dir2/dir3
```

rm - Removes a file or directory

```
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$ make run fsshell
gcc -c -o mfs.o mfs.c -g -I.
gcc -o fsshell fsshell.o fsInit.o bitmap.o directoryEntry.o mfs.o pathparse.o b_io.o fsLow.o -g -I. -lm -l readline -l pthread
./fsshell SampleVolume 10000000 512
File SampleVolume does not exist, errno = 2
File SampleVolume not good to go, errno = 2
Block size is : 512
Created a volume with 9999872 bytes, broken into 19531 blocks of 512 bytes.
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Initializing File System with 19531 blocks with a block size of 512
Prompt > touch filetoberemoved
Prompt > ls -l -a

D         3584   .
D         3584   ..
-            0   filetoberemoved
Prompt > rm filetoberemoved
Prompt > ls -l -a

D         3584   .
D         3584   ..
Prompt > exit
System exiting
make: 'fsshell' is up to date.
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$
```

```
Initializing File System with 19531 blocks with a block size of 512
Prompt > md dirToBeRemoved
Prompt > ls -l -a

D          3584   .
D          3584   ..
D          3584   dirToBeRemoved
Prompt > rm dirToBeRemoved
Prompt > ls -l -a

D          3584   .
D          3584   ..
Prompt >
```

touch - creates a file

```
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$ 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 > ls -l -a

D         3584   .
D         3584   ..
D         3584   newDirectory
Prompt > touch newFile.txt
Prompt > ls -l -a

D         3584   .
D         3584   ..
D         3584   newDirectory
-            0   newFile.txt
Prompt > exit
System exiting
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$
```

cat - (limited functionality) displays the contents of a file

```
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$ make run fsshell
./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 > cp2fs README.md
Prompt > ls

newDirectory
newFile.txt
README.md
Prompt > cat README.md
# CSC415 Group Term Assignment - File System

This is a GROUP assignment written in C.  Only one person on the team needs to submit the project.

Your team have been designing components of a file system.  You have defined the goals and designed the di
r file system.

While each of you can have your own github, only one is what you use for the project to be turned in.  Mak

The project is in three phases.  The first phase is the "formatting" the volume.  This is further describe

The second phase is the implementation of the directory based functions.  See Phase two assignment.

The final phase is the implementation of the file operations.
```

cp2l - Copies a file from the test file system to the linux file system

```
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$ 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 > touch test.txt
Prompt > ls -l -a

D        3584   .
D        3584   ..
D        3584   dir1
-           0   test.txt
Prompt > cp2l test.txt
Prompt > exit
System exiting
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$ ls
 b_io.c   b_io.o    bitmap.h   directoryEntry.c   directoryEntry.o   fsInit.o   fsLowM1.o   fsshell    fsshell.o   Makefile   mfs.h  'Milestone One.pdf'   pathparse.h   README.md      test.txt
 b_io.h   bitmap.c  bitmap.o   directoryEntry.h   fsInit.c           fsLow.h    fsLow.o     fsshell.c  Hexdump     mfs.c      mfs.o   pathparse.c           pathparse.o   SampleVolume
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$
```

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

```
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$ 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 > ls

newDirectory
newFile.txt
README.md
Prompt > rm README.md
Prompt > cp2fs README.md
Prompt > ls -l -a

D          3584   .
D          3584   ..
D          3584   newDirectory
-             0   newFile.txt
-          6986   README.md
Prompt > exit
System exiting
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$
```

cd - Changes directory

```
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$ make run fsshell
./fsshell SampleVolume 10000000 512
File SampleVolume does not exist, errno = 2
File SampleVolume not good to go, errno = 2
Block size is : 512
Created a volume with 9999872 bytes, broken into 19531 blocks of 512 bytes.
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Initializing File System with 19531 blocks with a block size of 512
Prompt > ls

Prompt > md dir1
Prompt > pwd
/
Prompt > ls

dir1
Prompt > cd dir1
Prompt > pwd
/dir1
Prompt > exit
System exiting
make: 'fsshell' is up to date.
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$
```

```
Prompt > md dir1
Prompt > cd dir1
Prompt > md dir2
Prompt > cd dir2
Prompt > md dir3
Prompt > cd dir3
Prompt > ls

Prompt > pwd
/dir1/dir2/dir3
```

```
Prompt > md newDirectory
Prompt > ls

newDirectory
Prompt > cd newDirectory
Prompt > pwd
/newDirectory
Prompt >
```

pwd - Prints the working directory

```
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$ 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 > pwd
/
Prompt > cd dir1
Prompt > pwd
/dir1
Prompt > exit
System exiting
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$
```

```
Prompt > md newDirectory
Prompt > ls

newDirectory
Prompt > cd newDirectory
Prompt > pwd
/newDirectory
Prompt >
```

```
Prompt > md dir1
Prompt > cd dir1
Prompt > md dir2
Prompt > cd dir2
Prompt > md dir3
Prompt > cd dir3
Prompt > ls

Prompt > pwd
/dir1/dir2/dir3
```

history - Prints out the history

```
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$ make run fsshell
./fsshell SampleVolume 10000000 512
File SampleVolume does not exist, errno = 2
File SampleVolume not good to go, errno = 2
Block size is : 512
Created a volume with 9999872 bytes, broken into 19531 blocks of 512 bytes.
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Initializing File System with 19531 blocks with a block size of 512
Prompt > ls -l -a

D         3584   .
D         3584   ..
Prompt > md newDir
Prompt > cd newDir
Prompt > touch file.txt
Prompt > touch file2.txt
Prompt > cd ..
Prompt > cat newDir/file.txt

Prompt > history
ls -l -a
md newDir
cd newDir
touch file.txt
touch file2.txt
cd ..
cat newDir/file.txt
history
Prompt > exit
System exiting
make: 'fsshell' is up to date.
student@student-VirtualBox:~/Desktop/csc415-filesystem-teamCN2S$
```

help - Prints out help

```
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 >
```