CSC415 - Operating Systems

Team Name: Satisfaction

Team Info:

Member Name	Student ID#	Github Name
Sulav Jung Hamal	923075813	Sulavjung
Miguel Maurer	922097199	miguelCmaurer
Yuvraj Gupta	922933190	YuvrajGupta1808
Fasika Abera	923038932	Fasikaabera

Github Name with our group work: miguelCmaurer

File System Assignment

1) The github link for your group submission.

https://github.com/CSC415-2024-Summer/csc415-filesystem-miguelCmaurer

2) The plan for each phase and changes made

Phase file system design:

The original plan for this assignment stayed generally the same throughout the project, starting from the file system design. For the file system design portion of the assignment, we decided that our directory entry struct would have the following form:

```
#define MAX_LEN_FOR_NAME 255

typedef struct{
        int location;
        int size;
        int permissions;
        time_t creationTime;
        time_t modificationTime;
        time_t accessTime;
        char name[MAX_LEN_FOR_NAME];
} DirEntry;
```

We planned it this way because it was important to keep extra fields to a minimum so we would have time to do everything. The structure is mostly identical, except we added another field, "int usedSize," representing the number of bytes currently being used. This addition allowed us to keep track of files, which would eventually be used as the limiting size for the read and write functions.

Volume Control Block was planned to have the structure:

```
typedef struct{
        int totalBlocks;
        int blockSize;
        int locRootDir;
        char signature[30];
} VCB;
```

The final iteration of the VCB remains largely the same. However, we made two modifications:

1. Following the advice, we replaced the 'char signature[30]' with an integer 'magic number.'
2. We added another integer to indicate the location of free space, which simplified the process of reinitializing free space.

Our plan to track free space using a bitmap remained consistent throughout the project. The current version of our filesystem code continues to use this bitmap to manage free space.

Phase Milestone 1:

For milestone 1, our plan was to implement the code for the plan we had made for the filesystem design. We first had to do to get the Volume Control Block (VCB) working, as it contains the information for reinitializing the filesystem even after the program terminates.
We knew the VCB would be in the first (0) block read by LBAread, so our approach was to:

1. Allocate memory
2. Fill this memory with the contents of the first block
3. Check for the presence of a "magic number"

This plan worked well, allowing us to create a program that could be shut down and rerun while persisting data written to the disk in previous runs.
After successfully writing the VCB, we worked on implementing our freespace bitmap.

We made two changes since the original plan:

1. We added a field in the VCB to indicate the starting location of the freespace blocks to simplify the reinitialization and loading of free space into memory.

2. We created a separate function to write changed bits from memory to disk. We planned to iterate and add them initially, but this would have led to unnecessary disk writes. Our new approach involves switching all desired bits in memory and then writing them once bitmap manipulation is complete.

For the root directory initialization, we utilized the initDir code demonstrated in class. This successful utilization of existing resources has allowed us to focus on other aspects of the project. We later added methods such as the setRoot function, which loads the root directory into a global variable.

Phase Milestone 2:
Milestone 2 required us to implement several functions, all of which depended on a parse path function. Although we were provided with most of the code for a parse path function, we still needed to integrate it properly into each function. Here's how we planned to use it for

fs_setcwd:
We planned to use the parse path to set a global reference to the current working directory. We had to verify that the returned parent exists and is a directory, then set the current directory to that reference. This approach worked well initially, but we encountered issues when switching the current working directory and freeing the previous directory. To fix these problems, we added a load directory function that allocates memory for a directory only if it's not the root or current working directory. If it is, it just returns the global reference. This way, there is only one pointer to the allocated memory. This modification helped us manage memory more efficiently.

fs_getcwd:
The implementation of fs_getcwd was relatively straightforward because the logic was provided to us in class. Our plan involves a stack that updates whenever the directory changes. When
fs_getcwd is called, it converts these stack values to a string, providing the current working directory path. While the core logic of this function remained unchanged throughout the implementation process, we faced some challenges with memory management.

fs_isFile:

The plan for fs_isFile was to check the entry that was returned by the parsePath and see if its permissions had the directory flag. This plan and implementation stayed the same.

fs_isDir
The plan was the same as fs_isFile but to check if the permission matched that of a directory.

fs_mkdir
The implementation plan for fs_mkdir relied heavily on the initDir function provided to us in class. This function allows us to create a new directory by passing in a parent directory and size, returning the location of the newly created directory. After receiving this new directory, we utilized the DirToMem function, which was developed during milestone 1, to load the directory parent into memory. Set the '.' directory entry to the location in the parent with the correct name. From this point, all that remained was to write the directory to disk. However, we encountered a synchronization issue between the memory and disk states of directories. To address this, we modified the DirToMem function to only return the global or current directory. This change allowed the directory state to remain consistent between memory and disk, preventing issues with memory inconsistencies.

fs_opendir
For this function, our implementation plan was to return a file descriptor to an array that contains a struct. Throughout the implementation process, this worked enough and didn't require significant modifications. The use of file descriptors worked to manage the fs_diriteminfo structures and allow for easy cleanup.

fs_readdir
The plan for fs_readdir wasn't too difficult. It only required us to update the current directory number and return a struct what contained all the information. The only part of this plan that was not predicted was when to correctly send the struct and when to keep iterating to the next de.

fs_closedir
The plan for fs_closedir relied on the fs_opendir and cleaned up the memory. Nothing really changed from the original plan when we coded it.

fs_stat
The original plan for fs_stat was to fill the struct with more values on open and read them into the buffer, but this didn't work out because the value being passed in was only

a path, which meant we had to run parsePath on it, and then get the values from the PPI.

fs_rmdir:
The plan for this was to use parsePath to verify that it was a directory and to get the directory that was to be deleted; once we had that, all that we needed to do was to clear the parent directory values and set it to memory.
We also overlooked updating the free space and preventing the deletion of the root and CWD. This was fixed by implementing a clearBit function and checking if the directory was a deletable function.

Phase Milestone 3:
Milestone 3 required us to implement all the file functions specified in the b_io.h file and essentially finish the assignment. Our initial plan was to create a file that would allow us to do all the other functions after that. That meant we had to implement the b_open function. To accomplish this, we wrote a helper function for file creation. This create_file function is responsible for locating the next available space in the directory and setting the values for that open Directory Entry, first updating the DE in memory, then writing it to disk, and finally updating the free space information.
Once we had a way to create a file, we could implement b_open. This function's implementation involved populating an array of File Control Block struct with the DE and other helper values. The overall approach for b_open was similar to the open function in a previous assignment, allowing us to follow a similar method. However, we also had to incorporate the usage of flags.
After implementing b_open, our next step was the b_write method. We prioritized this function because it would allow us to verify the file operations more easily. Our implementation approach for b_write was similar to the write function from our previous assignment but instead of reading from disk to our buffer, we were to transfer the user's buffer to our own internal buffer. We also used the parts methodology to write the correct number of bytes.
Once we knew that the files were being created correctly, we had to create the b_read function, whose logic was explained to us in class. We also used the parts methodology to read the correct number of bytes.
The last function that had to be created in the b_io.h file was the close function. This function frees up used memory and sets the buffer to null so the fd can be used again.
After all the functions were done, we planned to test the program and fix anything that did not work or try to.

b_open

Opens a file with the specified `filename` and `flags`. It parses the provided path, checks if the file exists, and if not, creates the file if the `O_CREAT` flag is set. It then initializes an FCB for the file and returns its index. If any error occurs during these operations, it returns `-1`. This function ensures that files are correctly opened and managed within the file system.

b_seek

Adjusts the file offset for the file descriptor `fd`. The function currently does nothing and always returns `0`. This placeholder function can be expanded to handle seeking within a file.

b_write

Writes `count` bytes from the buffer to the file associated with the file descriptor `fd`. It manages the FCB buffer, handles partial writes, and updates the file size and access time. It returns the number of bytes written or `-1` in case of an error. This function ensures data is correctly written to files, managing buffers and disk writes efficiently.

b_read

Reads `count` bytes into the buffer from the file associated with the file descriptor `fd`. It manages the FCB buffer, handles partial reads, and updates the file offset. It returns the number of bytes read or `-1` in case of an error. This function allows data to be read from files, handling buffers and disk reads effectively.

b_close

Closes the file associated with the file descriptor `fd`. It writes any remaining data in the FCB buffer to the disk, updates the directory entry, and frees the allocated memory for the FCB. It then marks the FCB as free by setting its buffer pointer to `NULL`. This function ensures that files are properly closed and resources are freed.

createfile

Creates a new file with the specified `name`, `size`, and `flags` in the directory represented by `dirEntry`. It finds an unused directory entry, sets the file's metadata, allocates space for the file, and updates the free space bitmap. It returns the index of the new directory entry or `-1` in case of an error. This function handles the creation of new files, ensuring proper allocation and metadata management.

3) A description of your file system:

The file system we created for this project is a simple version of a file system with basic functionality. Our file system allows the user to create a file, traverse the folders, and display the contents of the file; it also has two non regular functions: 1 to bring a file from the real file system to the one we made and also to bring a file from our file system to the main filesystem. This filesystem includes volume formatting and free space management. The formatting is done at the start of the application; first, it checks to see if the volume has been formatted. If the volume needs formatting, then it formats it, creates a root directory, brings it into memory, and creates a free space map. If it's already formatted, it brings the root directory to memory and the previous free space map.

4) Issues you had

Throughout the development of our filesystem, we encountered several issues, many of which were related to memory management. One problem was in the implementation of the loadDir command. The issue occurred when we called ls on a directory to get all the names within it; however, when we used it we were using an outdated value of the directory. This problem was shared between multiple functions that utilized loadDir, including the mkdir function.

The issue was that after creating a directory and then running an ls command, the new directory would not appear unless we changed to another directory and then returned. We resolved these issues by being more cautious when allocating memory for directories, especially for root or current directories. We paid closer attention to finding out when a new directory allocation was necessary and when we should use an existing allocation. This fix made it so data is shared across all its uses.

Another issue that was related to memory was when we would change the current directory into a folder for example, "cd ./1," then tried to switch to a nod

existent directory like "cd ../....fsd././", then if we did a "cd /1" the filesystem would just break.

To fix this, we had to create a function that would set the current directory to the global directory and ensure that it is not the root directory. This was fixed just by adding an if check.

One issue that we had that was with the read function. The function would work when we had to read less than the buffer size of 512 bytes, but when it was over, and we had to fill the remainder of a buffer and part of a new one, the output was not correct.

We fixed this by adding the number of bytes that have already been added to the memcpy in the part 3 portion of the write.

5) Details of how each of your functions work

`initFileSystem`: This function allocates memory for a block and then gets read into from LBAread with block 0. Then, it checks if the VCB has a matching magic number; if it does not match, the volume has not been initiated. So, to initialize it, the function first sets all the values in the VCB and initiates a bitmap for free space. It then creates a directory and sets it to root. We then write the new VCB To memory. If the volume has already been formatted, we set the global block size, bring the freespace bitmap back to memory, and set the root to memory.

`setBlockSize`: this is a helper function that takes the size of a block and sets a global variable to that value.

`initBitMap`: This function takes in the number of bytes and blocks, allocates memory for the free space, and sets a structure to hold all the values that will be helpful for other free space functions.

`setBit`: This function sets a bit which is used to represent a block number to be used, this only sets it in memory.

`writeBits``:` writes the bitmap in memory to the bitmap on the volume.

DirToMem: takes in the start location of a directory, and then it allocates enough memory for that director '.' entry and based on that, it will load the entire directory and return the pointer.

setRoot: Is only once and at the initialization of the directory so it sets the root to a DE and alto the CWD to the same location.

setBlockSize: helper function that takes in an integer and sets the global reference of block size to the passed in value.

reInitBitMap: This function is used when the volume has already been formatted and it loads the values from the bitmap on disk to the values in memory.

freeNeededDirs: Is used at the close of the program and provides the only way to free both the current and root directories. Checks if they are the same if they are not it will free them both.

freeBitMap: The function is run at closed and frees the allocated memory for the freespace.

freeSTRCWD: frees the string that is used for the current directory.

exitFileSystem: calls freeNededDirs, freeBitMap, freeSTRCWD to clean up memory.

b_open: This function is used to open a file given the parameters of a file name and flags. It parses the path that was passed in it then checks if the file exists or not. If the file does not exist and the flags allow for the creation of a file a file will be created in the parent directory. If the truncate flag is active it sets the size to 0; it then gets a field descriptor and fills the values of the FCB at the file descriptor, and returns the file descriptor.

b_write: It takes in a user buffer and writes it to the file. This function gets the current FCB in the FCB array. Once we have that, we need to check to make sure it will be within the range for the max size of the file. It then checks if the buffer has been allocated, allocates it, and sets starting values. If not, the

initialization also checks for the append flag. If it is there, it will set the size to 0. Then, we calculate the amount that we need to write to the existing buffer. Then, it checks how many complete buffers can be written directly to the disk, and part three reads the remaining content to a new buffer. This function also updates the file size in both memory and on Disk. This function also keeps count of how many bytes have been added and returns that to the user.

b_read: Fills the user's buffer, similar to b_write. This function checks the validity of the path and verifies the flags that allow for the file to be read. This function also checks to see if the buffer has been allocated. If not, it will allocate the memory for it and all other values used to fill the user's buffer correctly. Once we have all the values we need to start filling the user's buffer, we first read from the buffer that already is in memory; in part two, we check to see if we can directly add to the user buffer from disk, in part three we add the values that need a new buffer and read that into our buffer then the remaining amount to the users buffer.

b_close: takes in the file descriptor and frees the memory associated with it, such as the buffer, which is also then set to NULL. It also frees the DE if it is needed.

createFile: The create file function takes in the name, the directory it is to be placed in, and flags. It then checks flags and whether there is an available DE in the directory. It then gets the next free available size, sets the name to the passed-in parameter, and sets all default values. Then, it sets the bits and writes them to memory. It also writes the directory to memory.

nextFree: Next free iterates the free space to find the following available start location of the free space map. During the iteration, if the bit is not used, it will increment the count of length. If the length is also the size of how much is needed, we return the start index. If not, we reset the start index and the count of consecutive blocks.

getBit: returns if bit at a given location is used or not

clearBit: using bitwise operations it sets a bit to 0.

writeBits: using bitwise operations it sets a bit to 0.

initDir: this function takes in a minimum number of directory entries, a parent Directory entry, and a block size. It then sets the global block size variable if it needs to be updated, then it calculator how many bytes are required, then it calculates how many blocks are needed, then we allocate memory for the directory entries; once we have that we set the blocks used in our bit map and set the default values for the . and .. entries we then write the values to memory and return the unneeded memory and return the location where the new directory is located.

findNameInDir: This function iterates the passed in parent directory and then returns the location of the directory, it it is not found it returns -1;

findUnusedDE: iterates the passed-in directory, and it looks for the first directory entry whose location is 0, this is a signal of an unused DE because the VCB is at location 0.

entryIsDir: check the permissions bit that represents if it is a dir or not.

freeIfNotNeeded: takes in a parent and sees if it is the root or current directory if it is not it will free the memory for the de.

loadDir: takes in the parent and position in parent and checks if the location is the same as the root directory or the current directory. If it is, it returns either the root or current directory, but if not, it calls DirToMem and returns the directory entry.

getRoot: returns the global pointer to the root directory.

getCWD: returns the global pointer to the current directory.

getBlockSize: helper function to return the size of the global block.

setBlockSize: helper function to set the size of the global block.

setCWD: helper function to set the current directory.

File: `fsPath.h` and `fsPath.c`

`pathCleaner`: Path cleaner takes in a path, and if the first character is /, then it removes all other values in the stack because it is the root directory. Otherwise, it will loop through the path, tokenizing it at the '/' character if the value is '.' nothing is added to the stack if it is '..', then it pops a value if it can.

`parsePath`: This function, parsePath, is given a file path and ppinfo struct, which will be filled with information about the path. It starts by checking the input path and seeing whether to start from the root or the current working directory. The function then tokenizes the path, iterating through each element. For each token, it updates the position in the parent directory, checks if it's the last element, and verifies if the path is valid and if each component is a directory. If it is the last element, it sets the parent and last element information in the ppinfo structure. The function also handles empty paths or invalid paths. It also frees memory while it is running to prevent lost memory.

`getCWDStr`: This function is a wrapper around the toString function for the stack. It gets the total size that will be required from the stack and loops through the stack its contents, the values of the stack, into the string.

`freePPI`: This method frees the `ppinfo` structure. It checks if the parent is the current working directory or it is the root. If not, frees the parent and then frees the structure.

`freeSTRCWD`: This method frees the current working directory string and stack. It pops all values from the stack and frees the current path string if it is not `null`.

File: `fsDirIteration.c` and `mfs.c`

`fs_mkdir`: This function takes in a path name and also modes. First, the function parses the path that is passed in and checks if there is an error or if the name already exists in the current directory. Then, it finds the location of the next available DE. It then initiates a new directory and brings that newly created directory to memory. And sets the parent directory to the location that was free in the parent directory. Then, it writes everything in memory and frees unneeded values.

`fs_rmdir`: This function starts by allocating memory for path parsing and creates a copy of the provided path. After parsing the path, it checks if the directory exists and is not the root or current working directory. If the directory exists but is not empty, the method aborts the removal process. Otherwise, it clears the directory entries in the bitmap, updates the modification time, and writes the changes to the disk. Finally, it frees allocated memory and returns 0 on successful removal.

`fs_opendir`: This function takes in a path name and runs parsePath on that path to get the parent directory, which will be used by fs_readdir. It allocates memory for the struct for directory item info and returns a pointer to the directory info array.

`Fs_readdir`: The function takes in a directory info pointer and continues to fill the values of it until it eventually returns null. Each time it is called the directory entry that is return is incremented until the values are exhausted and returns null.

`fs_closedir`: The `fs_closedir` function closes the given directory stream. It frees the memory allocated for the directory entry information and the directory structure if not needed and returns the directory entry position.

`fs_getcwd`: This function retrieves the current working directory and stores it in the provided buffer up to a maximum of size characters.

`fs_setcwd`: It starts by parsing the `filename`. If the path parsing is successful, it ensures that the last element in the path is a valid directory. If it is, the function loads the directory into memory, sets it as the current working directory, update the string representation of the current working directory using `pathCleaner`, and returns 0 to indicate success.

`fs_isFile`: This method parses the `filename`. If the path parsing is successful, it then verifies weather the parsed path is not a directory using the `entryIsDir` function. The function negates the result (to confirm it's a file, not a directory), frees the allocated memory, and returns the result, indicating if the path is a file(non-zero) or not (zero).

`fs_isDir`: Takes `filename` and parses it using parsing method. It checks for errors or if the file does not exist, in which case it frees the allocated memory and returns 0. If the path parsing is successful, it checks weather the parse path is a

directory using `entryIsDir` function. The function then frees the allocated memory and returns the result in zero(not directory) or non-zero(directory).

`fs_delete`: It allocates memory for path parsing and creates a copy of the filename. After parsing the path, it verifies if the file exists in the parent directory. If the file does not exist or if there are errors during parsing, the method aborts and prints an error message. If the file exists, it updates the modification time, clears the file's entries in the bitmap, resets the file's size, and location to zero, and clears the filename. Then writes that to the disk using `LBAwrite`. Then, allocated memory is freed, and the method returns 0.

`fs_stat`: This function retrieves the status of the file or directory specified by the given path and fills the provided `fs_stat` structure with the relevant information from the `ppinfo` structure returned from the parsing.

Screen shots showing each of the commands listed in the readme

ScreenShot of Compilation:

```
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ make
gcc -c -o fsshell.o fsshell.c -g -I.
gcc -c -o fsInit.o fsInit.c -g -I.
gcc -c -o fsBitmap.o fsBitmap.c -g -I.
gcc -c -o fsDir.o fsDir.c -g -I.
gcc -c -o mfs.o mfs.c -g -I.
gcc -c -o fsPath.o fsPath.c -g -I.
gcc -c -o fsDirIteration.o fsDirIteration.c -g -I.
gcc -c -o b_io.o b_io.c -g -I.
gcc -o fsshell fsshell.o fsInit.o fsBitmap.o fsDir.o mfs.o fsPath.o fsDirIteration.o b_io.o fsL
owM1.o -g -I. -lm -l readline -l pthread
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$
```

Initial Run:

```
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ 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                   |    ON    |
| cp2fs                |    ON    |
| cp2l                 |    ON    |
|--------------------------------|
Prompt >
```

Command: ls

```
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ 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                   |    ON    |
| cp2fs                |    ON    |
| cp2l                 |    ON    |
|--------------------------------|
Prompt > ls

test2
sulav
miguel
yuvraj
fasika
sulav.txt
Prompt >
```

Command: cd

```
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ 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                   |    ON    |
| cp2fs                |    ON    |
| cp2l                 |    ON    |
|--------------------------------|
Prompt > ls

test2
sulav
miguel
yuvraj
fasika
sulav.txt
Prompt > cd sulav
Prompt > ls

one.pdf
two.pdf
three.pdf
test
Prompt > cd ..
Prompt > ls

test2
sulav
miguel
yuvraj
fasika
sulav.txt
Prompt >
```

Command: md

```
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ 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                   |    ON    |
| cp2fs                |    ON    |
| cp2l                 |    ON    |
|--------------------------------|
Prompt > ls

sulav
miguel
yuvraj
fasika
sulav.txt
Prompt > md test2
Prompt > ls

test2
sulav
miguel
yuvraj
fasika
sulav.txt
Prompt >
```

Command pwd

```
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ 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                   |    ON    |
| cp2fs                |    ON    |
| cp2l                 |    ON    |
|--------------------------------|
Prompt > pwd
/
Prompt > cd sulav
Prompt > cd test
Prompt > cd test2
Prompt > pwd
/sulav/test/test2/
Prompt >
```

Command: touch

```
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ 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                   |    ON    |
| cp2fs                |    ON    |
| cp2l                 |    ON    |
|--------------------------------|
Prompt > cd /sulav/test/test2/
Prompt > ls

Prompt > touch trying.txt
Prompt > ls

trying.txt
Prompt >
```

Command: cat

```
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ 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                   |    ON    |
| cp2fs                |    ON    |
| cp2l                 |    ON    |
|--------------------------------|
Prompt > ls

test2
sample.txt
sulav
miguel
yuvraj
fasika
sulav.txt
Prompt > cat sample.txt
Team Info:
Member Name
Sulav Jung Hamal 923075813 Sulavjung
Miguel Maurer 922097199 miguelCmaurer
Yuvraj Gupta 922933190 YuvrajGupta1808
Fasika Abera 923038932 Fasikaabera
Prompt >
```

Command: rm

```
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ 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                   |    ON    |
| cp2fs                |    ON    |
| cp2l                 |    ON    |
|--------------------------------|
Prompt > ls

test2
sample.txt
sulav
miguel
yuvraj
fasika
sulav.txt
Prompt > rm sample.txt
Prompt > ls

test2
sulav
miguel
yuvraj
fasika
sulav.txt
Prompt >
```

Command: cp

```
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ 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                   |    ON    |
| cp2fs                |    ON    |
| cp2l                 |    ON    |
|--------------------------------|
Prompt > ls

test2
sample.txt
sulav
miguel
yuvraj
fasika
sulav.txt
Prompt > cd sulav
Prompt > ls

one.pdf
two.pdf
three.pdf
test
Prompt > cd ..
Prompt > cp sample.txt sulav/teamInfo.txt
Prompt > cat sulav/teamInfo.txt
Team Info:
Member Name
Sulav Jung Hamal 923075813 Sulavjung
Miguel Maurer 922097199 miguelCmaurer
Yuvraj Gupta 922933190 YuvrajGupta1808
Fasika Abera 923038932 Fasikaabera
Prompt >
```

Command: mv

```
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ 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                   |    ON    |
| cp2fs                |    ON    |
| cp2l                 |    ON    |
|--------------------------------|
Prompt > ls

test2
sample.txt
sulav
miguel
yuvraj
fasika
sulav.txt
Prompt > cd sulav
Prompt > ls

one.pdf
two.pdf
three.pdf
test
teamInfo.txt
Prompt > cd ..
Prompt > mv sample.txt sulav/sample.txt
Prompt > ls
```

```
test2
sulav
miguel
yuvraj
fasika
sulav.txt
Prompt > cd sulav
Prompt > ls

one.pdf
two.pdf
three.pdf
test
teamInfo.txt
sample.txt
Prompt > cat sample.txt
Team Info:
Member Name
Sulav Jung Hamal 923075813 Sulavjung
Miguel Maurer 922097199 miguelCmaurer
Yuvraj Gupta 922933190 YuvrajGupta1808
Fasika Abera 923038932 Fasikaabera
Prompt >
```

Command: cp2fs

```
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ 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                   |    ON    |
| cp2fs                |    ON    |
| cp2l                 |    ON    |
|--------------------------------|
Prompt > cd sulav/test
Prompt > cp2fs sample.txt
Prompt > ls

sample.txt
Prompt > cat sample.txt
Team Info:
Member Name
Sulav Jung Hamal 923075813 Sulavjung
Miguel Maurer 922097199 miguelCmaurer
Yuvraj Gupta 922933190 YuvrajGupta1808
Fasika Abera 923038932 Fasikaabera
Prompt >
```

Command: cp2l

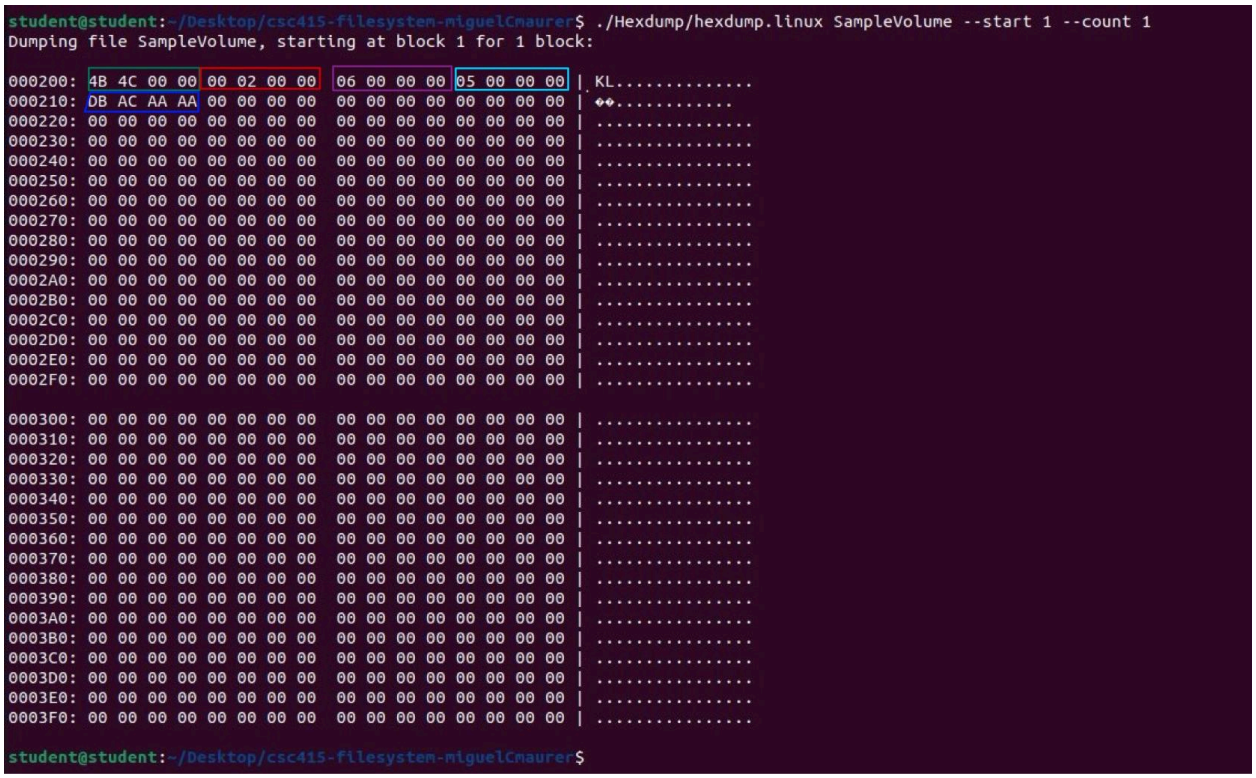
```
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ ls
b_io.c  fsBitmap.c  fsDir.c          fsDirIteration.o  fsInit.o   fsLow.o   fsPath.o   fsshell.o  mfs.c  README.md     Satisfaction_writeup_M1.pdf
b_io.h  fsBitmap.h  fsDir.h          fsDir.o           fsLow.h    fsPath.c  fsshell    Hexdump    mfs.h  sample.txt
b_io.o  fsBitmap.o  fsDirIteration.c  fsInit.c          fsLowM1.o  fsPath.h  fsshell.c  Makefile   mfs.o  SampleVolume
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ 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                   |    ON    |
| cp2fs                |    ON    |
| cp2l                 |    ON    |
|--------------------------------|
Prompt > cd sulav/test
Prompt > ls

sample.txt
Prompt > cp2l sample.txt samplecopy.txt
Prompt > ls

sample.txt
Prompt > exit
System exiting
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ ls
b_io.c  fsBitmap.c  fsDir.c          fsDirIteration.o  fsInit.o   fsLow.o   fsPath.o   fsshell.o  mfs.c  README.md       SampleVolume
b_io.h  fsBitmap.h  fsDir.h          fsDir.o           fsLow.h    fsPath.c  fsshell    Hexdump    mfs.h  samplecopy.txt  Satisfaction_writeup_M1.pdf
b_io.o  fsBitmap.o  fsDirIteration.c  fsInit.c          fsLowM1.o  fsPath.h  fsshell.c  Makefile   mfs.o  sample.txt
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$
```

Analysis:

VCB analysis:



1. `totalBlocks` : Because the first value in the VCB struct is the total blocks it appears first in the hexdump. Because it is an int it takes up 32 bits, or 8 hex digits. Because the representation of all values are in little-endian format the value is 0x4C4B = 19531.This is correct because when the program runs it prints out how many blocks there are:

```
Initializing File System with 19531 blocks with a block size of 512
```

2. `blockSize` : Block size is the size of each of the blocks. This is represented by an int. So, it will also take up 32 bits, or 8 hex digits. Also, because we already know all values in the hexdump will be in a little-endian format the value is 0x200 = 512, this is the expected value as shown above.

3. `locRootDir` : The location of root directory is stored next, similar to all the other values it is also in so 8 hex digits, which is expected to be at 6, this is because the first block is used for the VCB and the following five is used for the bitmap representation of free block.

4. `freeSpace` : The freespace field represents the number of blocks that are being used to represent the freespace and we know that this value is five and will take up 8 hex digits. We know this is correct because the location of the root directory is after this, so 1 for the VCB + 5 for freespace.

5. `signature:` Because the current signature being used is 0b10101010101010101010110011011011 this has a hex value of 0xAA AA AC DB, which when converted to little endian format is seen in the screenshot.

FreeSpace analysis:

```
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ make run
./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
|---------------------------------|
|------- Command ------|- Status -|
| ls                   |    ON    |
| cd                   |    ON    |
| md                   |    ON    |
| pwd                  |    ON    |
| touch                |    ON    |
| cat                  |    ON    |
| rm                   |    ON    |
| cp                   |    ON    |
| mv                   |    ON    |
| cp2fs                |    ON    |
| cp2l                 |    ON    |
|---------------------------------|
Prompt > ls

Prompt > md ./1
Prompt > md ./2
Prompt > ls

1
2
Prompt > cp2fs ./Makefile ./1/Makefile
Prompt > ls ./1/

Makefile
Prompt > exit
System exiting
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$
```

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

000400: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
000410: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
000420: FF FF FF FF 1F 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 | ................

000600: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000610: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000620: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000630: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000640: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000650: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000660: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000670: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000680: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000690: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

```
0006B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000700: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000710: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000720: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000730: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000740: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000750: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000760: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000770: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000780: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000790: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

student@student:~/Desktop/csc415-filesystem-miguelCmaurer$
```

Because the free space bitmap is not a struct there is little need to highlight the important bits, and because the used blocks can all be represented in the first block, the first block shows all the important information about the free space. We can verify that this is correct by knowing what is being saved in the current program. We know that the first 6 bits should be 1, because the VCB and the freespace itself. The remaining bits being used come from the initialization of the root directory. When we initialize it we ask for 50 entries, and with a size of 296, which means that we need 14800 bits -> 29 blocks + 58 for the directories 1 and 2, then we create a file of size 200 blocks. So in total the freespace is expected to use 87 + 200 + 6 = 293 bytes.
This is what is seen in the hex dump: the first 2 complete rows of F count for 256 blocks then the 4 couples of F count for 32 then +5 for the last couple, this gives a total of 293 bits being = 1 which is what is expected.

```
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ 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                   |    ON    |
| cp2fs                |    ON    |
| cp2l                 |    ON    |
|---------------------------------|
Prompt > rm ./1/Makefile
Prompt > cd ./1
Prompt > ls

Prompt > exit
size: 296
System exiting
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$
```

Here we removed the file:

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

000400: FF FF FF FF FF FF FF FF  FF FF FF 1F 00 00 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:~/Desktop/csc415-filesystem-miguelCmaurer$
```

Now that the file is gone we expect to see a size of 93, The value shown in the hexdump after the file has been removed is 93. So we know the file has been removed correctly.

Now we remove a directory.

```
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ make run
gcc -c -o fsInit.o fsInit.c -g -I.
gcc -o fsshell fsshell.o fsInit.o fsBitmap.o fsDir.o mfs.o fsPath.o fsDirIteration.o b_io.o fsLow.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                   |    ON    |
| cp2fs                |    ON    |
| cp2l                 |    ON    |
|---------------------------------|
Prompt > ls

1
2
Prompt > rm ./1
Prompt > ls

2
Prompt > exit
System exiting
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$
```

Here we expect to see a gap of 29 where the freespace used to be taken up by the directory.

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

000400: FF FF FF FF 07 00 00 00  FF FF FF 1F 00 00 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:~/Desktop/csc415-filesystem-miguelCmaurer$
```

And as we can see the gaps (4+1+8+8+8) = 29 bytes that are in the gap

```
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ 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                   |    ON    |
| cp2fs                |    ON    |
| cp2l                 |    ON    |
|---------------------------------|
Prompt > ls

2
Prompt > rm ./2
Prompt > ls

Prompt > exit
System exiting
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$
```

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

000400: FF FF FF FF 07 00 00 00  00 00 00 00 00 00 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:~/Desktop/csc415-filesystem-miguelCmaurer$
```

And here after we deleted the last directory we are left with the same amount of free space we started with seen below.

```
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$ make run
./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
|---------------------------------|
|------- Command ------|- Status -|
| ls                   |    ON    |
| cd                   |    ON    |
| md                   |    ON    |
| pwd                  |    ON    |
| touch                |    ON    |
| cat                  |    ON    |
| rm                   |    ON    |
| cp                   |    ON    |
| mv                   |    ON    |
| cp2fs                |    ON    |
| cp2l                 |    ON    |
|---------------------------------|
Prompt > exit
System exiting
student@student:~/Desktop/csc415-filesystem-miguelCmaurer$
```

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

000400: FF FF FF FF 07 00 00 00  00 00 00 00 00 00 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:~/Desktop/csc415-filesystem-miguelCmaurer$
```