**CSC 415 - Operating Systems**
**Team Buffer - Sajan Gurung, Micheal Hua, , Deep Dhorajiya, Ron Elijah Rivera**

# File System Project

Team: Team Buffer

Micheal Hua: 918729206

Sajan Gurung: 921149577

Deep Dhorajiya: 920842114

Ron Elijah Rivera: 921656746

GitHub Repository Name: michealhuaa

**Github link**

https://github.com/CSC415-2022-Fall/csc415-filesystem-michealhuaa

**CSC 415 - Operating Systems**
**Team Buffer - Sajan Gurung, Micheal Hua, , Deep Dhorajiya, Ron Elijah Rivera**

## Description

A lot of different components are required to design a file system. There are a number of components that operate in this manner, including those responsible for allocating memory, keeping track of open space, and finding addresses. Our group will concentrate on a volume control block, how we keep track of free space, our design for directory entries, and the kinds of metadata we intend to include in our file system as part of this project. What each of those things will look like will be discussed in detail on the following pages.

## Issues faced

One of the issues that we faced was understanding how to get our bitmap working. At first, we set each character in a char* array to either 1 or 0. We quickly realized that the five blocks of space allotted to us could only accommodate 2560 0s or 1s. That was inconsistent with Professor Bierman's claim that we should be able to accommodate 20,000 0s and 1s in the same space. We discovered that a bitmap could change all eight bits in a single one-byte character to either 1 or 0. Upon research, we figured out how to do this.Each character has 8 bits. We want to store eight times as much data in this array as we normally could by using one character for each data point.As a result, each character must contain eight data values. To accomplish this, we divide the position by 8 to determine the storage index. Since the file is a whole number, it will continuously give us an entire number as opposed to a twofold. All of these numbers, from 0 to 7, will be stored in index 0, since they will all be equal to 0 when divided by 8. These eight distinct values must now be stored within the same character. To get the position we utilize the modulus administrator, which gives the rest of when separated. This will grant us a unique position among the char* arrays indexed from 0 to 7. Last but not least, we shift the position of a "1" using the  operator and store it in the bitmap, regardless of how many places the modulus operator returns.

Additionally, we had difficulty comprehending how to initialize each root directory with known free space. During one of our meetings, we sat down to discuss the issue and try to figure out how to get started. We tried looking into various approaches to initializing our root directory, but we realized we were overthinking the situation. We only needed to change one of the variables to a value that we knew would only be used by a directory entry that had not been initialized. We were able to concentrate on it being in a free space rather than worrying about root directories being overwritten, thereby resolving our problem.

Another issue was finding a way to obtain a complete path from a relative path was one challenge. We realized that the relative path needed to be added to the current path. We needed to read in the directory entry location and proceed backwards through ".." from there in order to

obtain a complete path from just a file location. We added the new fileName to a character string separated by a "/" after each pass. When we were only given a file location, this gave us a full path.
Moving directories was another problem we encountered. Because you have to work with the parent directory, the destination directory, and the source directory, it gets complicated. Even if nothing is actually being moved, all three of these will change during the move. We made a diagram that showed what we would work with and what would change in order to solve this issue. This made it easier for us to imagine how the move works. It was clear from that point on that all we had to do was modify the directory entry in the parent, move the moved directory's parent directory into the destination directory, and add the moved directory as a directory entry in the destination.

We also had trouble understanding how freeing memory worked. We discovered after rewatching lectures that we are simply marking the blocks in our bitmap array as unused rather than deleting anything from memory. The system is able to write over these data blocks because of this. When we can simply mark data as inactive and allow the system to overwrite it, it would be pointless to delete it.

The parsePath function was probably the hardest part of the assignment.We had to figure out how to deal with all the special cases, like "..," relative paths, absolute paths, and "..". At first, we were able to make a function that worked, was messy, and could take absolute paths. In the end, we were able to resolve the issue by establishing a global variable that kept track of the current working directory. We always update that variable. We were able to simply append absolute paths to a given relative path using that variable.

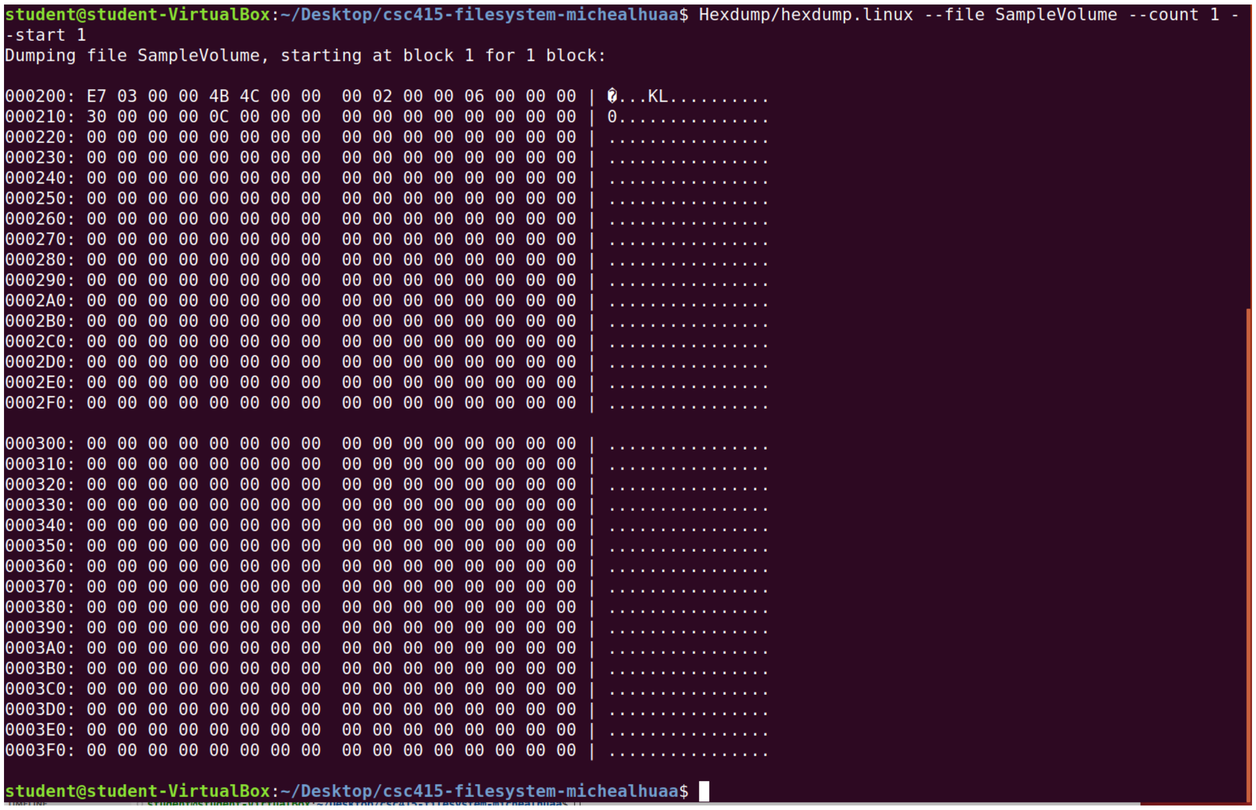
The ls command was involved in the subsequent issues. Since there are a lot of moving parts, a segmentation fault could result from an incorrect malloc. Through a couple of long stretches of troubleshooting, we had the option to sort out a couple of causes. It appears that the ls function is working well.

Many of our problems fell into one of these two categories:1) technical issues with our repository; or 2) comprehending the project's various components and how to get them to work properly. In comparison to the real world, I believe that these are issues that affect anyone from novice software developers to senior software engineers .In the end, the common denominators that assisted us in overcoming our challenges were a combination of research and collaboration.

## Detail of driver program

**CSC 415 - Operating Systems**
**Team Buffer - Sajan Gurung, Micheal Hua, , Deep Dhorajiya, Ron Elijah Rivera**

## Volume Control Block:

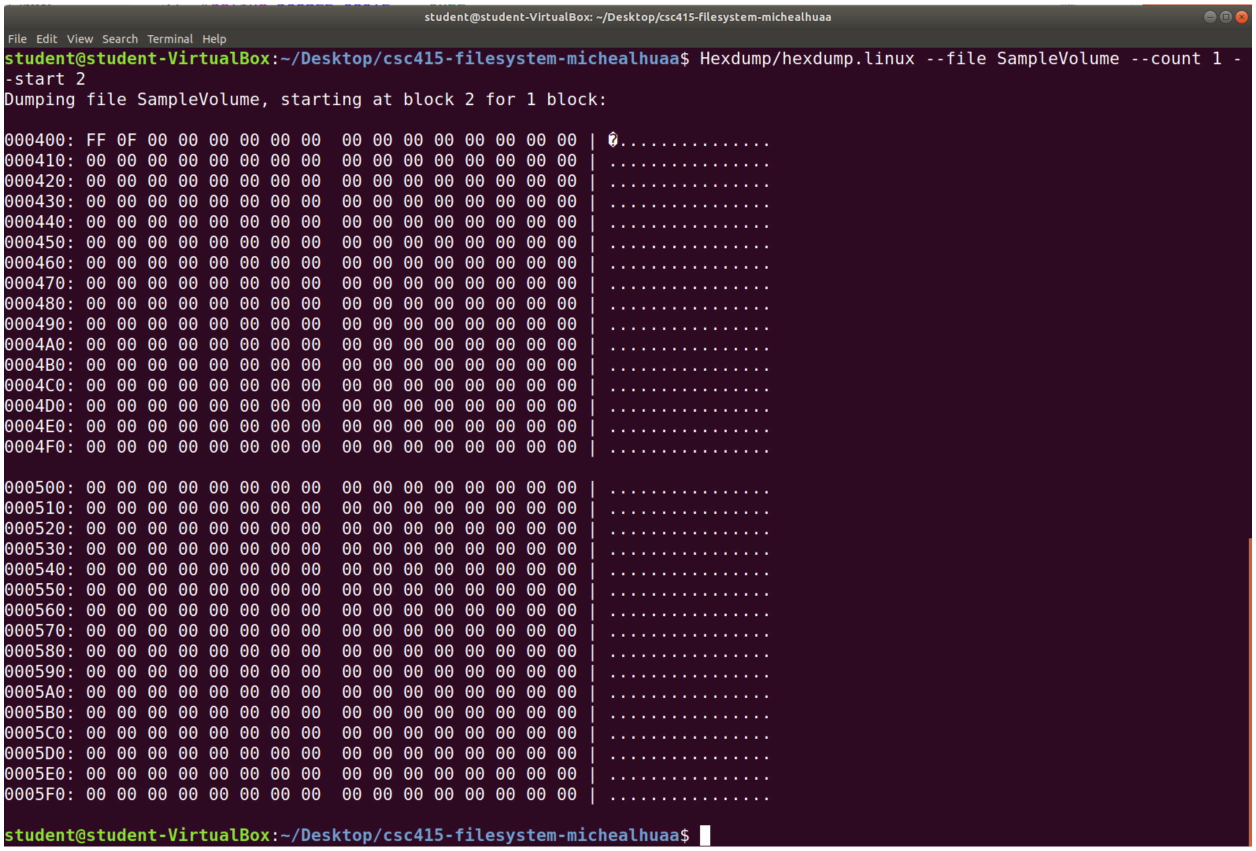


```
student@student-VirtualBox:~/Desktop/csc415-filesystem-michealhuaa$ Hexdump/hexdump.linux --file SampleVolume --count 1 -
-start 1
Dumping file SampleVolume, starting at block 1 for 1 block:

000200: E7 03 00 00 4B 4C 00 00  00 02 00 00 06 00 00 00 | �...KL..........
000210: 30 00 00 00 0C 00 00 00  00 00 00 00 00 00 00 00 | 0...............
000220: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000230: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000240: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000250: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000260: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000270: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000280: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000290: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

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

student@student-VirtualBox:~/Desktop/csc415-filesystem-michealhuaa$
```

Here we see the volume control block. Freespace is the first variable inside of the structure. E7 translates to 231, which is the index inside of the bitmap where freespace currently starts. The next data point is the number of blocks, which is 19531. After that, we see block size, root directory starting location, and the magic number. A total of 20 bytes, or 5 integers.

## Bitmap Dump/Free Space:

**CSC 415 - Operating Systems**
**Team Buffer - Sajan Gurung, Micheal Hua, , Deep Dhorajiya, Ron Elijah Rivera**


student@student-VirtualBox: ~/Desktop/csc415-filesystem-michealhuaa
File Edit View Search Terminal Help

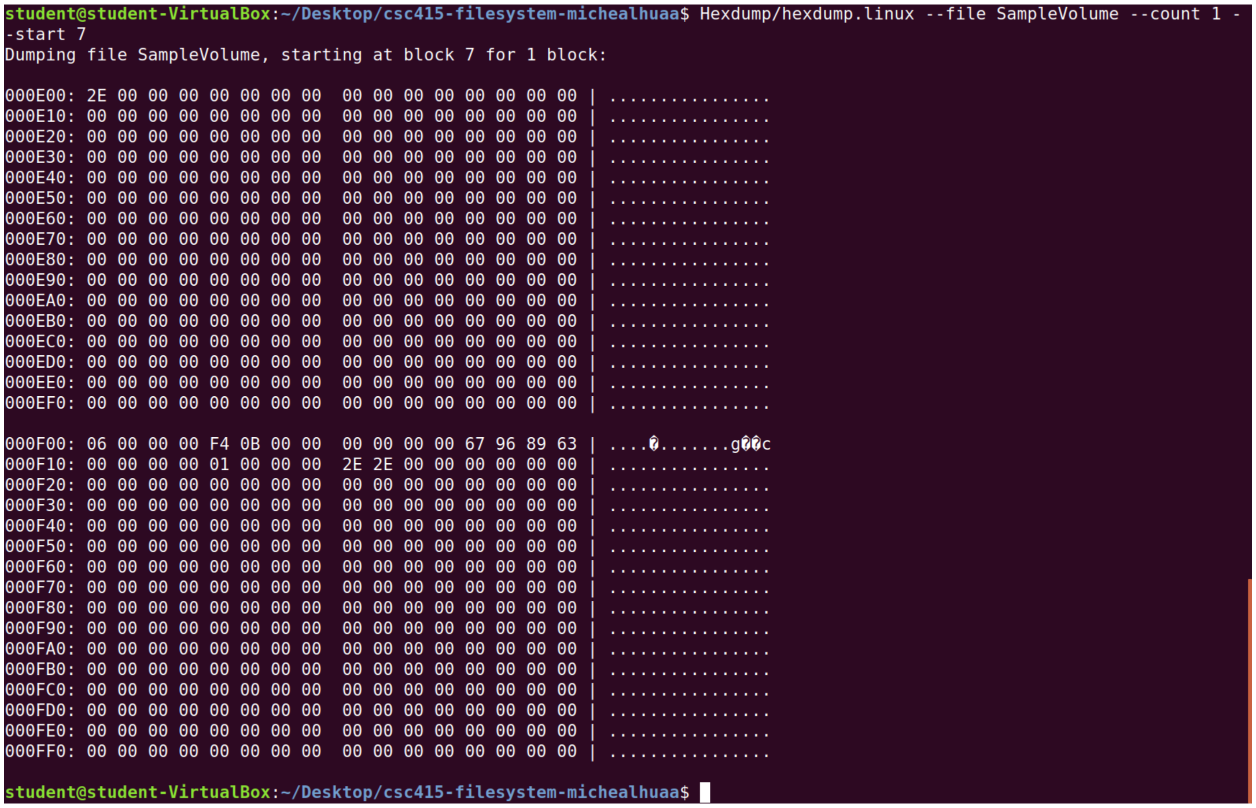
```
student@student-VirtualBox:~/Desktop/csc415-filesystem-michealhuaa$ Hexdump/hexdump.linux --file SampleVolume --count 1 -
-start 2
Dumping file SampleVolume, starting at block 2 for 1 block:

000400: FF 0F 00 00 00 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-VirtualBox:~/Desktop/csc415-filesystem-michealhuaa$
```

Two bytes are being utilized here.The freespace bitmap currently has twelve filled blocks .One byte contains eight bits. The first byte is filled, and the value is "FF," which in binary is 11111111. Four bits, or 1111, have been added to the second byte. Our bitmap has a function called resetBits that releases the bits back into memory before updating the bitmap. It does this by iterating through a predetermined position and a predetermined number of blocks before retrieving the index in the array and the specific position within the index. The function then has a value of clearBit, which is equal to one and is an unsigned integer. After that, we shift it by the number of bits that correspond to the position in the index, change the value of clearBit to zero, write the current index value of the bitmap to bitmap[i], and clearBit returns the value to zero. The bitmap is updated after being edited.

**CSC 415 - Operating Systems**

**Team Buffer - Sajan Gurung, Micheal Hua, , Deep Dhorajiya, Ron Elijah Rivera**

**Root Directory Dump:**



```
student@student-VirtualBox:~/Desktop/csc415-filesystem-michealhuaa$ Hexdump/hexdump.linux --file SampleVolume --count 1 -
-start 7
Dumping file SampleVolume, starting at block 7 for 1 block:

000E00: 2E 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000EA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000EB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000EC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000ED0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000EE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000EF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000F00: 06 00 00 00 F4 0B 00 00  00 00 00 00 67 96 89 63 | ....�.......g��c
000F10: 00 00 00 00 01 00 00 00  2E 2E 00 00 00 00 00 00 | ................
000F20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000FA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000FB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000FC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000FD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000FE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000FF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

student@student-VirtualBox:~/Desktop/csc415-filesystem-michealhuaa$
```

**Volume Control Block:**

The start of the free space is initialized by the VCB (Volume Control Block). We are able to store the necessary information and minimize the amount of wasted space within the blocks by setting the number of blocks and their sizes. We can determine whether this is our Volume Control Block by using the magic number. We must start a new VCB if the magic number is absent. In the program, we check where the root directory is located in memory, how big the memory blocks are, and where free space currently begins.

**Free Space Structure:**

Since a char is 1 byte, that means you can store 8 bits inside of 1 char. It would be inefficient to store only one 0 or 1 per char. You could be 8 times more efficient by storing a 0 or 1 on each bit. So for our free space, we used a bitmap array of chars. Free space is initialized by using a bitmap using the contents within the VCB. We use the bitmap to check whether the block has space using the values 0 and 1, each block corresponds to a different bit position in the array.
Since a bitmap can store 8 times more 0s or 1s than a regular char value, we get the index by

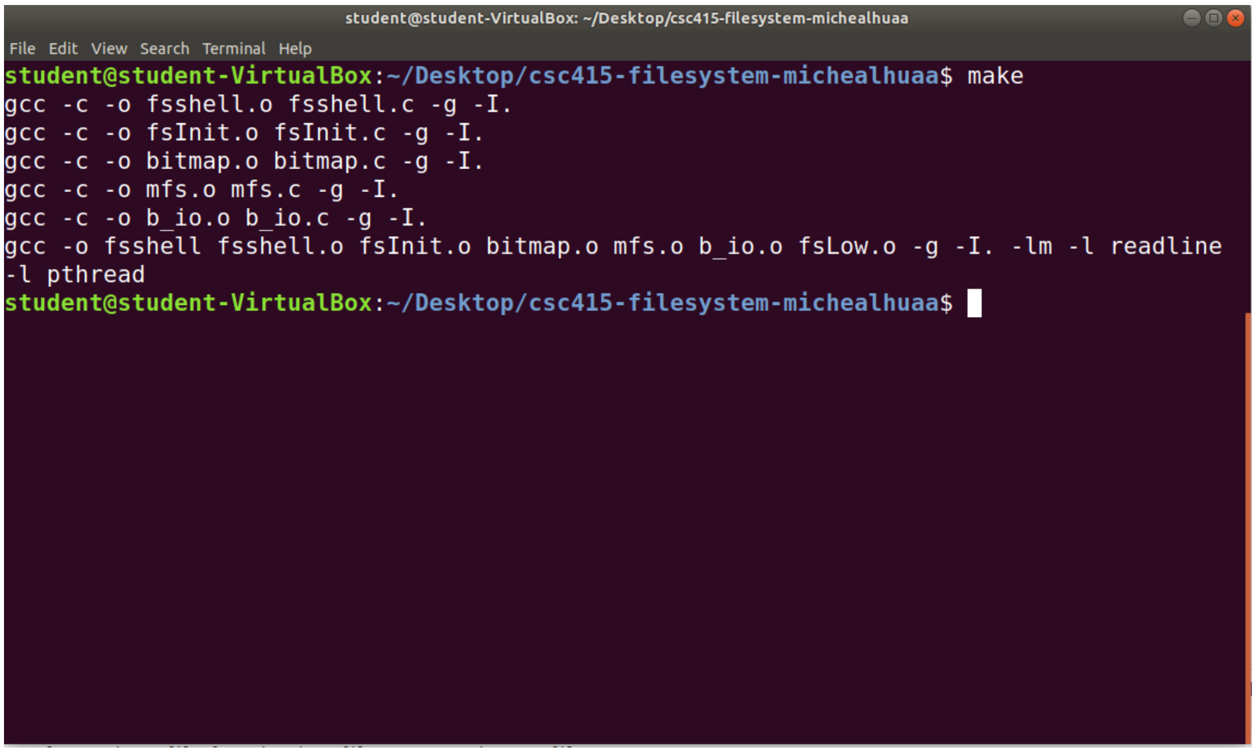
dividing the number by 8. Then we use the modulus operator(number%8) to get the position inside of the index. This allows for any number to have a unique position inside of an integer. We then use the << operator to shift the "1" bit through the char to the correct position. A similar process is used to check the value of a given position.

## Directory System:

All of the metadata as well as the core data (file logical address start point and file size) are contained in our directory entry. Root is the only array of directory entries in our current directory system.Since a directory entry cannot have a logical block address of 0, all entries in Root have been pre-initialized to have a fileLocation of 0, indicating that they are not used. Two entries in the root directory have been fully initialized. The characters "." which refers back to itself. With the exception of the name, index [1] is identical to index [0] in every way. This is due to the fact that the root is at the top of the tree, and ".." indicates a parent. Root is the child of itself.

## Screenshots

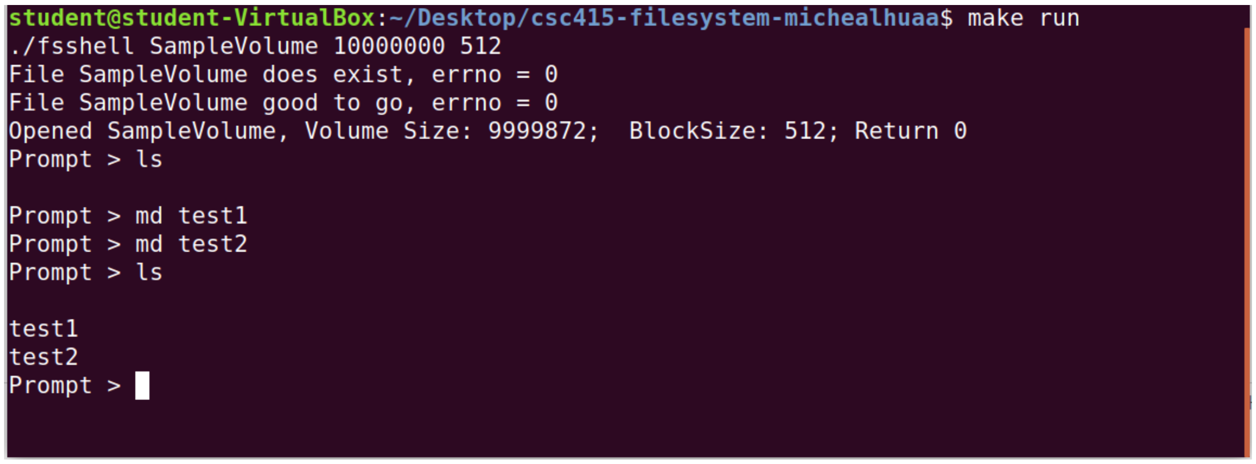
**Compilation Screenshot:**


student@student-VirtualBox: ~/Desktop/csc415-filesystem-michealhuaa
File Edit View Search Terminal Help

```
student@student-VirtualBox:~/Desktop/csc415-filesystem-michealhuaa$ make
gcc -c -o fsshell.o fsshell.c -g -I.
gcc -c -o fsInit.o fsInit.c -g -I.
gcc -c -o bitmap.o bitmap.c -g -I.
gcc -c -o mfs.o mfs.c -g -I.
gcc -c -o b_io.o b_io.c -g -I.
gcc -o fsshell fsshell.o fsInit.o bitmap.o mfs.o b_io.o fsLow.o -g -I. -lm -l readline
-l pthread
student@student-VirtualBox:~/Desktop/csc415-filesystem-michealhuaa$ 
```

**CSC 415 - Operating Systems**
**Team Buffer - Sajan Gurung, Micheal Hua, , Deep Dhorajiya, Ron Elijah Rivera**

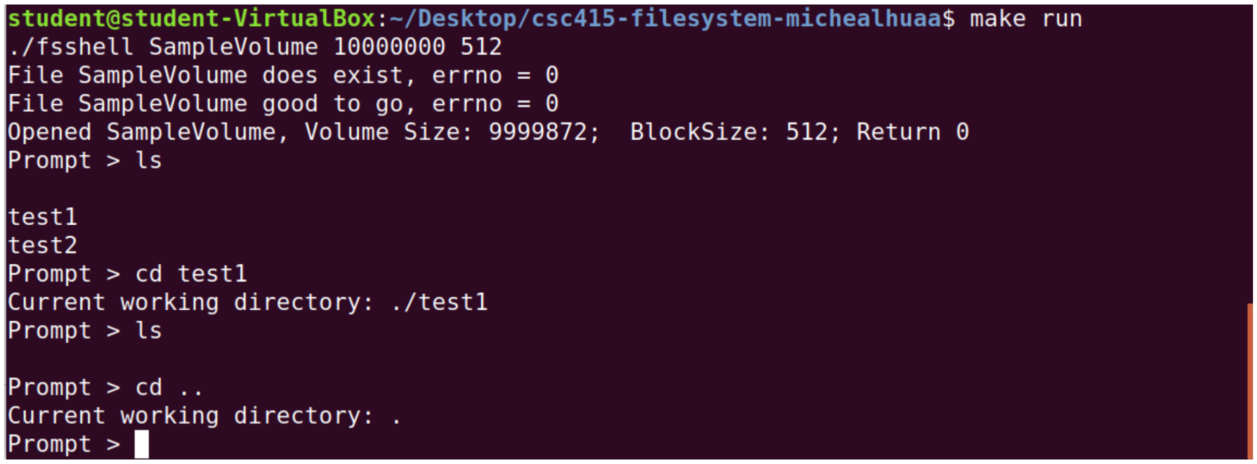
**“md” and “ls” command**



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

Prompt > md test1
Prompt > md test2
Prompt > ls

test1
test2
Prompt > 
```

In the above picture we create two directories using md(make directory) command and use ls to check if the directories have been created.

**“cd” command**

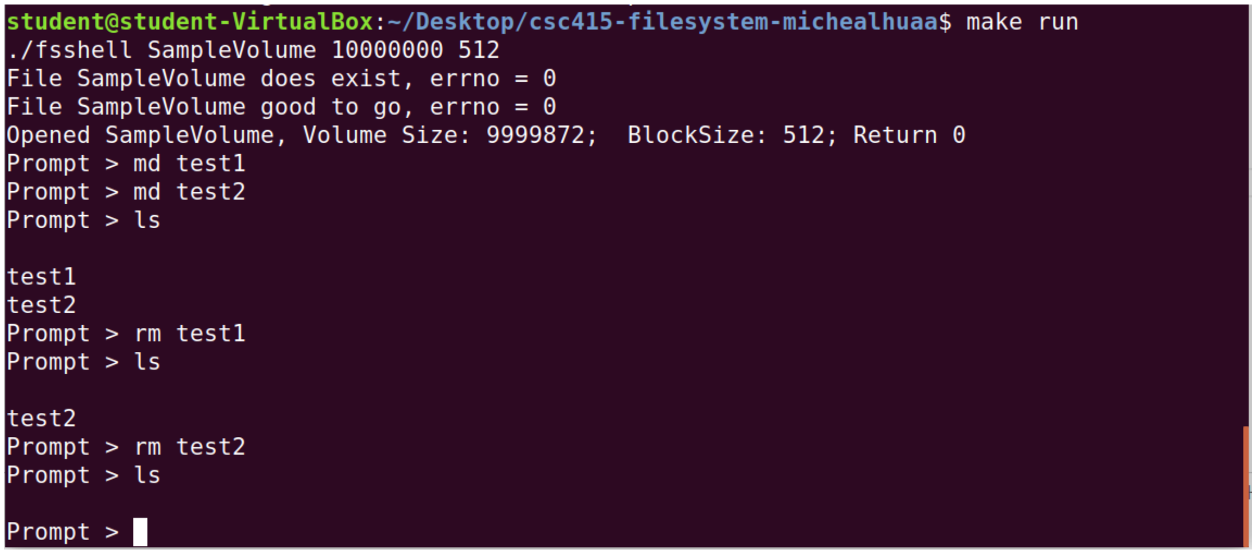


```
student@student-VirtualBox:~/Desktop/csc415-filesystem-michealhuaa$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Prompt > ls

test1
test2
Prompt > cd test1
Current working directory: ./test1
Prompt > ls

Prompt > cd ..
Current working directory: .
Prompt > 
```

Here, in the picture we tested the cd command. First we created to directories and then we used cd command to enter into the directory as shown in the picture. We also tried to go back from the current directory to the parent directory using cd.. command.

**CSC 415 - Operating Systems**
**Team Buffer - Sajan Gurung, Micheal Hua, , Deep Dhorajiya, Ron Elijah Rivera**

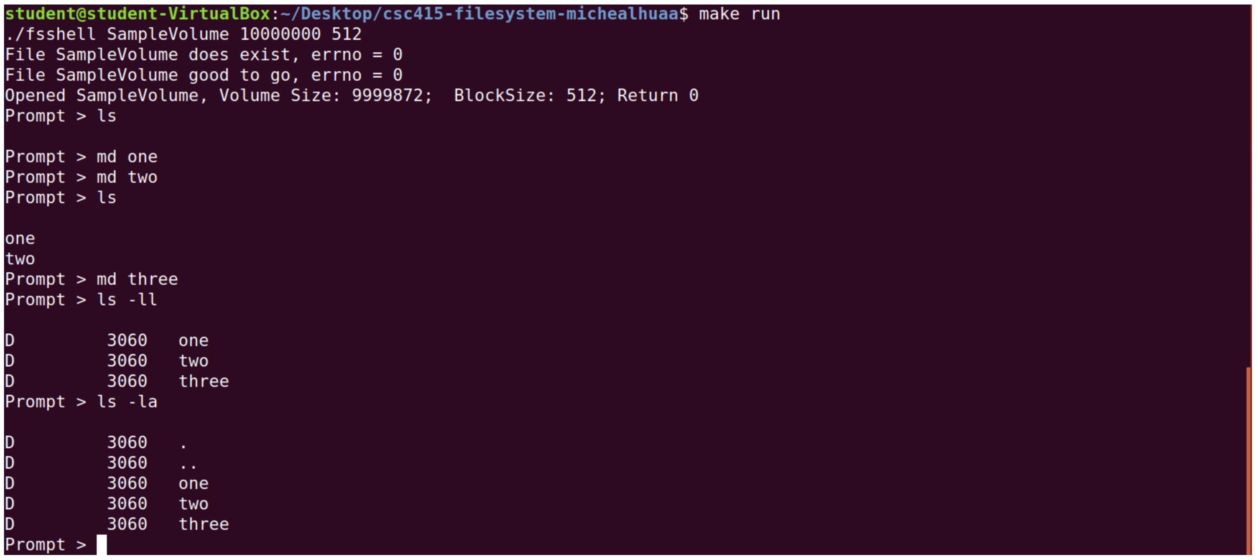
## “rm ” command



```
student@student-VirtualBox:~/Desktop/csc415-filesystem-michealhuaa$ 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
Prompt > md test1
Prompt > md test2
Prompt > ls

test1
test2
Prompt > rm test1
Prompt > ls

test2
Prompt > rm test2
Prompt > ls

Prompt >
```

In the above picture, we created two test directories using md command and later we deleted those directories using rm command as shown in the picture. After using the rm command we used ls command to check if the directory is deleted or not.

## Demonstration ls, ls -ll, la -ll

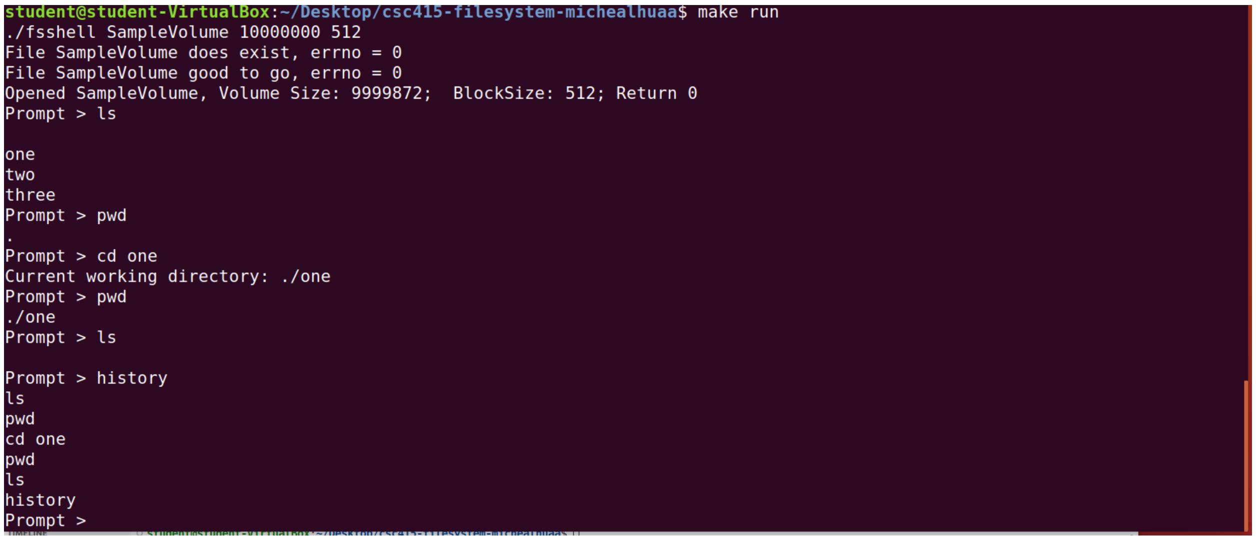


```
student@student-VirtualBox:~/Desktop/csc415-filesystem-michealhuaa$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Prompt > ls

Prompt > md one
Prompt > md two
Prompt > ls

one
two
Prompt > md three
Prompt > ls -ll

D         3060   one
D         3060   two
D         3060   three
Prompt > ls -la

D         3060   .
D         3060   ..
D         3060   one
D         3060   two
D         3060   three
Prompt >
```

**CSC 415 - Operating Systems**

**Team Buffer - Sajan Gurung, Micheal Hua, , Deep Dhorajiya, Ron Elijah Rivera**

“pwd” and “history” command



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

one
two
three
Prompt > pwd
.
Prompt > cd one
Current working directory: ./one
Prompt > pwd
./one
Prompt > ls

Prompt > history
ls
pwd
cd one
pwd
ls
history
Prompt >
```