Zari Haidarian ID: 917423031
Akim Tarasov ID: 922761746
Diego Antunez ID: 922061514

Group Name: Coney Island
CSC415 Operating Systems

Assignment- File System Milestone 1

Group Name: Coney Island
Github: AkimT13

Zari Haidarian ID: 917423031
Akim Tarasov ID: 922761746
Diego Antunez ID: 922061514

Group Name: Coney Island
CSC415 Operating Systems

Description:

This assignment was to start setting up our groups file system project. We formatted the volume and implemented many structs and functions in order to help us initialize the free space and the root directory for our file system.

Approach:

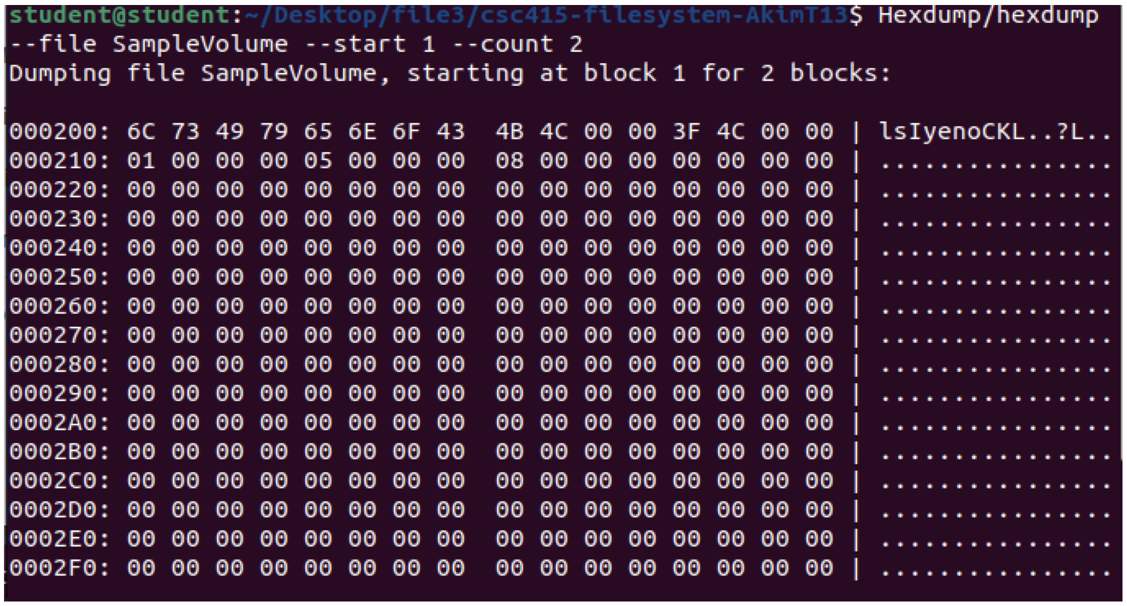
Our approach for this assignment was to split the work into three parts and each take on a section of the requirements. Diego worked on formatting the volume, Akim worked on initializing the free space, and Zari worked on initializing the root directory. We made sure to coordinate with each other while working on our tasks. We each made a plan for what we needed to do based off of the instructions given to us and worked through our sections individually. Once we finished we planned to put our code together and make sure everything was working as planned.

Issues and Resolutions:

See analysis Part 7.

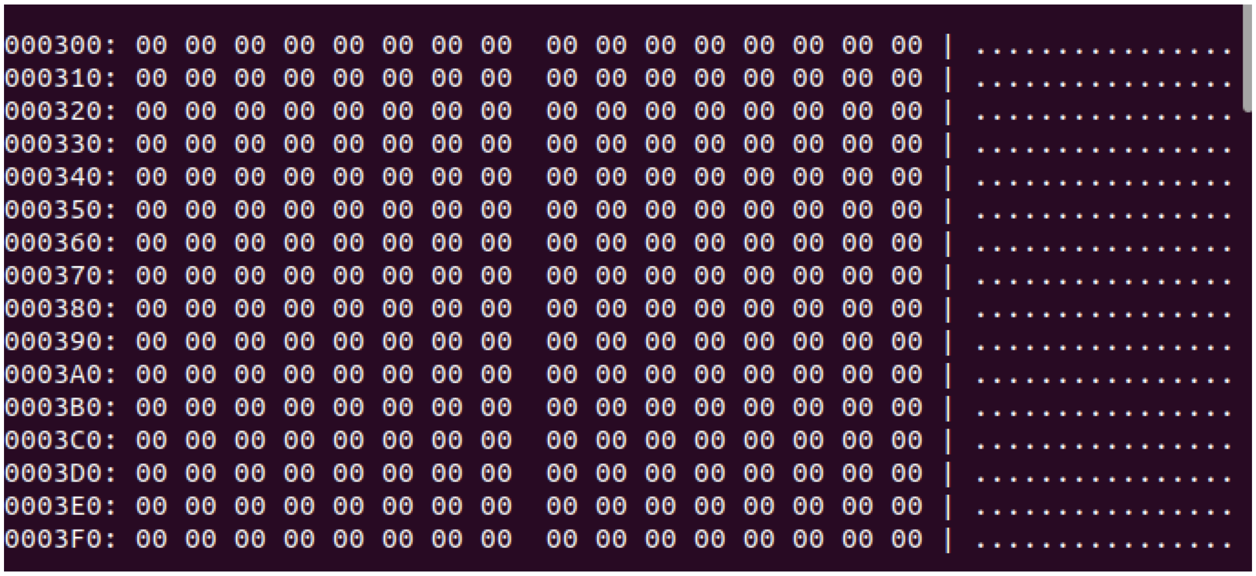
Analysis:

1. A dump (use the provided HexDump utility) of the volume file that shows the VCB, FreeSpace, and complete root directory.

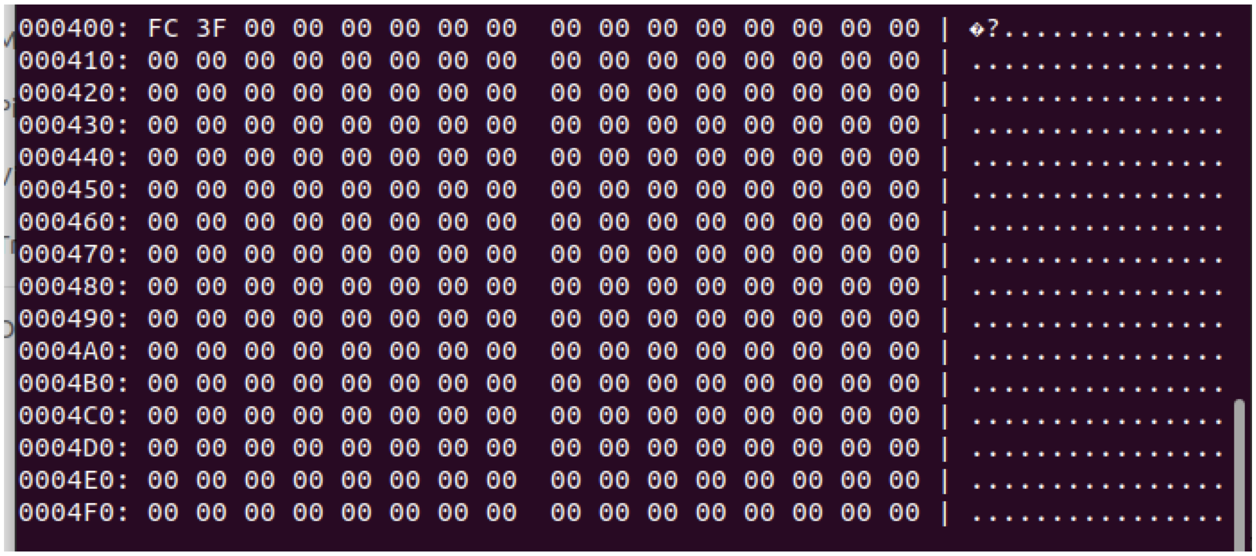


```
student@student:~/Desktop/file3/csc415-filesystem-AkimT13$ Hexdump/hexdump
--file SampleVolume --start 1 --count 2
Dumping file SampleVolume, starting at block 1 for 2 blocks:

000200: 6C 73 49 79 65 6E 6F 43  4B 4C 00 00 3F 4C 00 00 | lsIyenoCKL..?L..
000210: 01 00 00 00 05 00 00 00  08 00 00 00 00 00 00 00 | ................
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 | ................
```

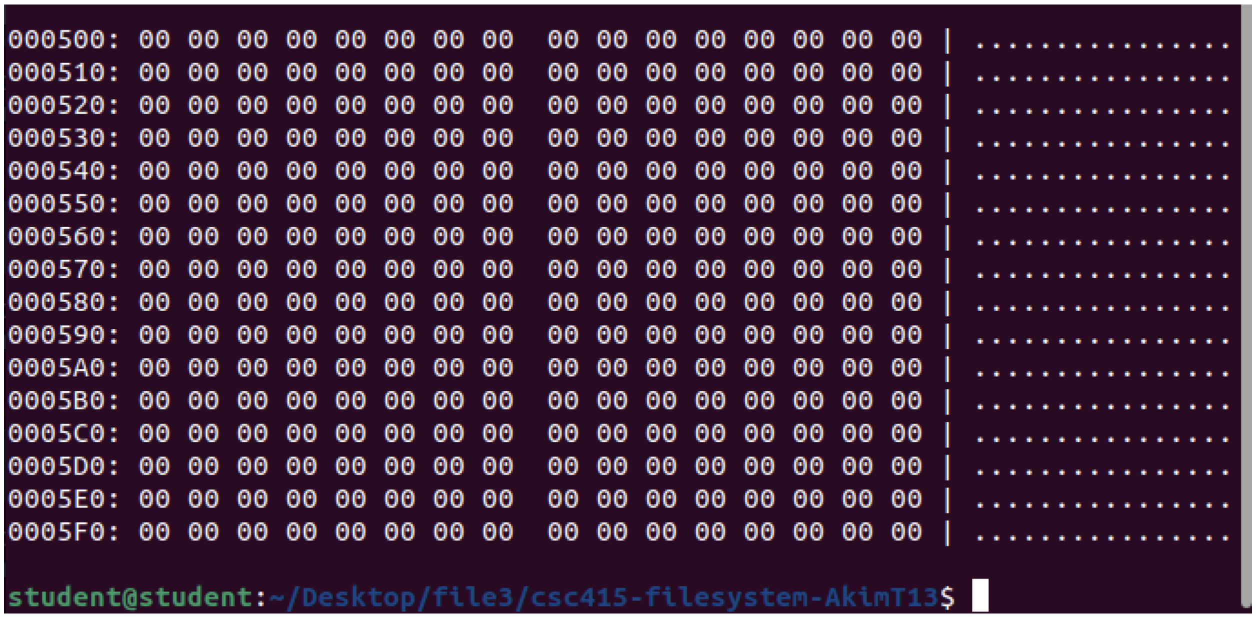
Zari Haidarian ID: 917423031
Akim Tarasov ID: 922761746
Diego Antunez ID: 922061514



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

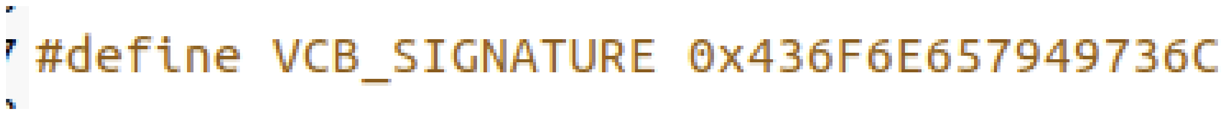


```
000400: FC 3F 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | �?..............
000410: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000420: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000430: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
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 | ................
```



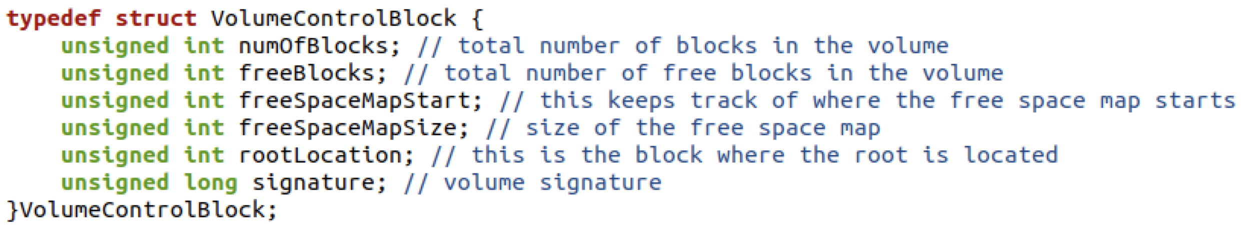
```
000500: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000510: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000520: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000530: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
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:~/Desktop/file3/csc415-filesystem-AkimT13$
```

First row: 6C 73 49 79 65 6E 6F 43
This is our signature for the volume control block



```
#define VCB_SIGNATURE 0x436F6E657949736C
```

2. A description of the VCB structure



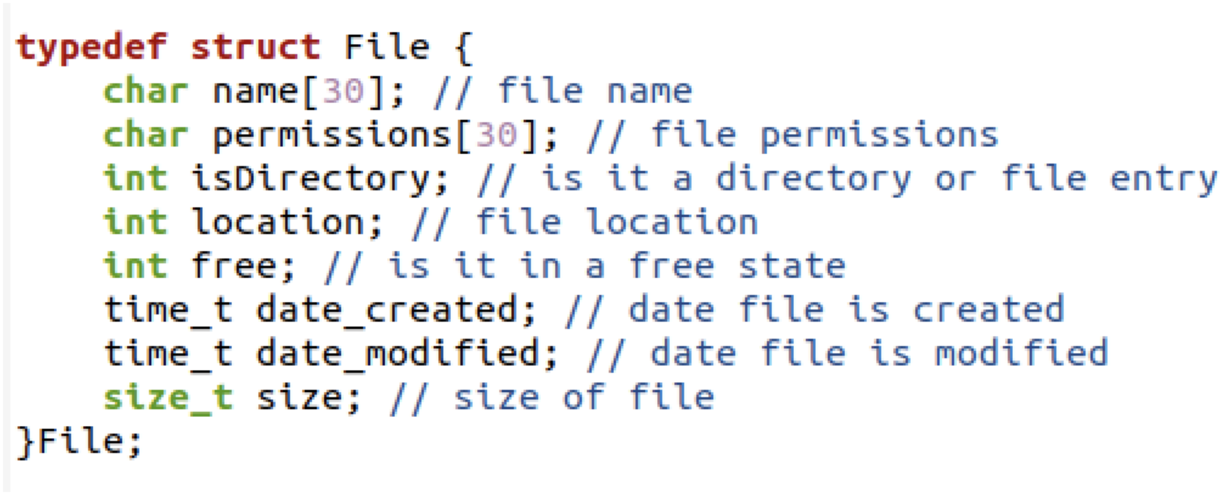
```
typedef struct VolumeControlBlock {
    unsigned int numOfBlocks; // total number of blocks in the volume
    unsigned int freeBlocks; // total number of free blocks in the volume
    unsigned int freeSpaceMapStart; // this keeps track of where the free space map starts
    unsigned int freeSpaceMapSize; // size of the free space map
    unsigned int rootLocation; // this is the block where the root is located
    unsigned long signature; // volume signature
}VolumeControlBlock;
```

The Volume Control Block or the VCB consists of metadata about volume. It has many pieces of data including a signature to confirm the volume, the total number of blocks available, as well as the total number of free blocks available. It also includes information about the free space map such as where it starts from, its size, and the root location. All the data is stored in a specific order to prevent unnecessary padding. This struct helps us track and allocate space for directories and files in our file system.

3. A description of the Free Space structure

We decided to use a simple bitmap for our free space system. Each bit represents a block. If the bit is 0 that block is free, if it is 1 then it is used. The initFreeSpace function is called when the vcb's function signature doesn't match and will return the starting block of the free space. I used calloc to allocate memory for the freespace because calloc initializes every bit to 0. If malloc was used, every char would be uninitialized so setting it to 0 when allocating prepares for eventual bit flipping. Simililary to the lectures and instructions, we chose 5 blocks to be the size of the bitmap which will manage 40 blocks. We will increase this in the future. In order to flip the bits, we used a bit mask where the first 6 bits are 1. This is then OR'd to the first byte of the freespace in order to mark the first 6 blocks as used. This is because the VCB is the first block and the freespace is the next 5 blocks. LBAWrite() is then called in order to have the free space map be persistent. allocBlocks will take in a count that represents how many blocks the caller wants to use. It will then try to find that number of consecutive blocks within the freespace. currBlock tracks which bit we are checking and currently tracks how many consectutive free blocks we have found. To check if a block is free, we determine the corresponding byte and bit position using byteIndex = currBlock / 8 and bitIndex = currBlock % 8, since each byte contains 8 bits. We create a bit mask by shifting 1 by bitIndex positions. This results in a number where all the bits are 0 except the bit we are interested in checking. This can then be AND'd with the byte in the bitmap to see if it's free. If the result is 0, then the block is available. Once curr hits the desired count, we need to mark all of those bits as used and return the where that consecutive sequence starts. LBAwrite is then called to save the new bitmap.

4. A description of the Directory system



```
typedef struct File {
    char name[30]; // file name
    char permissions[30]; // file permissions
    int isDirectory; // is it a directory or file entry
    int location; // file location
    int free; // is it in a free state
    time_t date_created; // date file is created
    time_t date_modified; // date file is modified
    size_t size; // size of file
}File;
```

The Directory struct (which we named File) consists of metadata about the free space. We used a bitmap to allocate memory based on the number of blocks needed to cover the disk's total blocks. Every bit in freeSpaceMap represents a block's availability. It has many pieces of data including the name, permissions, location, size, and date modified/created. We also included data about whether it is a directory or not, and whether it is free or not. All the data is stored in a specific order to prevent unnecessary padding.

5. A table of who worked on which components

Name	**What was worked on**
Zari Haidarian	Initializing Root Directory
Akim Tarasov	Initializing Free Space Map
Diego Antunez	Formatting Volume Control Block

6. How did your team work together, how often you met, how did you meet, how did you divide up the tasks.

Our team met up several times throughout the course of working on milestone one. We met up through discord to plan out what we needed to do and divide the work up between all members of the team. Since we have only three members in our group we divided the work into three separate tasks, with plans to put everything together and finalize all the testing and requirements when we were done with our individual portions.

7. A discussion of what issues you faced and how your team resolved them.

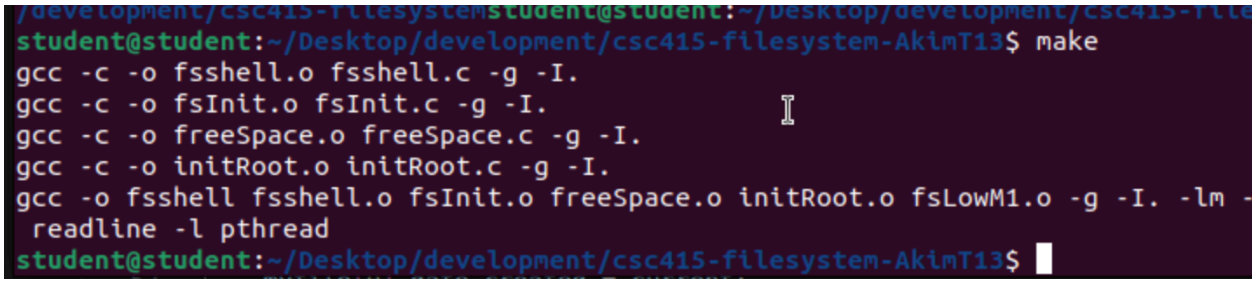
In our file system design, we got a point taken off for unnecessary padding. I googled how a struct might introduce padding and learned that you can avoid unnecessary padding by reordering the long variables to be at the end of the struct. http://www.catb.org/esr/structure-packing/ -Akim

Another issue is the fact that we made the freeSpaceStart be an unsigned integer. It should be signed because my initFreeSpace function should return a -1 if there is an error with the initialization with the freespace bitmap. Therefore we should change the freeSpaceMapStart variable's type to be just int so we can do proper error handling and return the proper start position when it's successful. -Akim

Zari Haidarian ID: 917423031
Akim Tarasov ID: 922761746
Diego Antunez ID: 922061514

One incorrect assumption I made when allocating space for the bitmap was assuming that the default value of each index would be a char represented by 8 0's. This is not the case, malloc allocates space but leaves the char's uninitialized. I searched up a list of more memory functions that might help me and found one called calloc, which also allocates memory similarly to malloc and initializes everything to 0. -Akim

Screenshot of compilation:

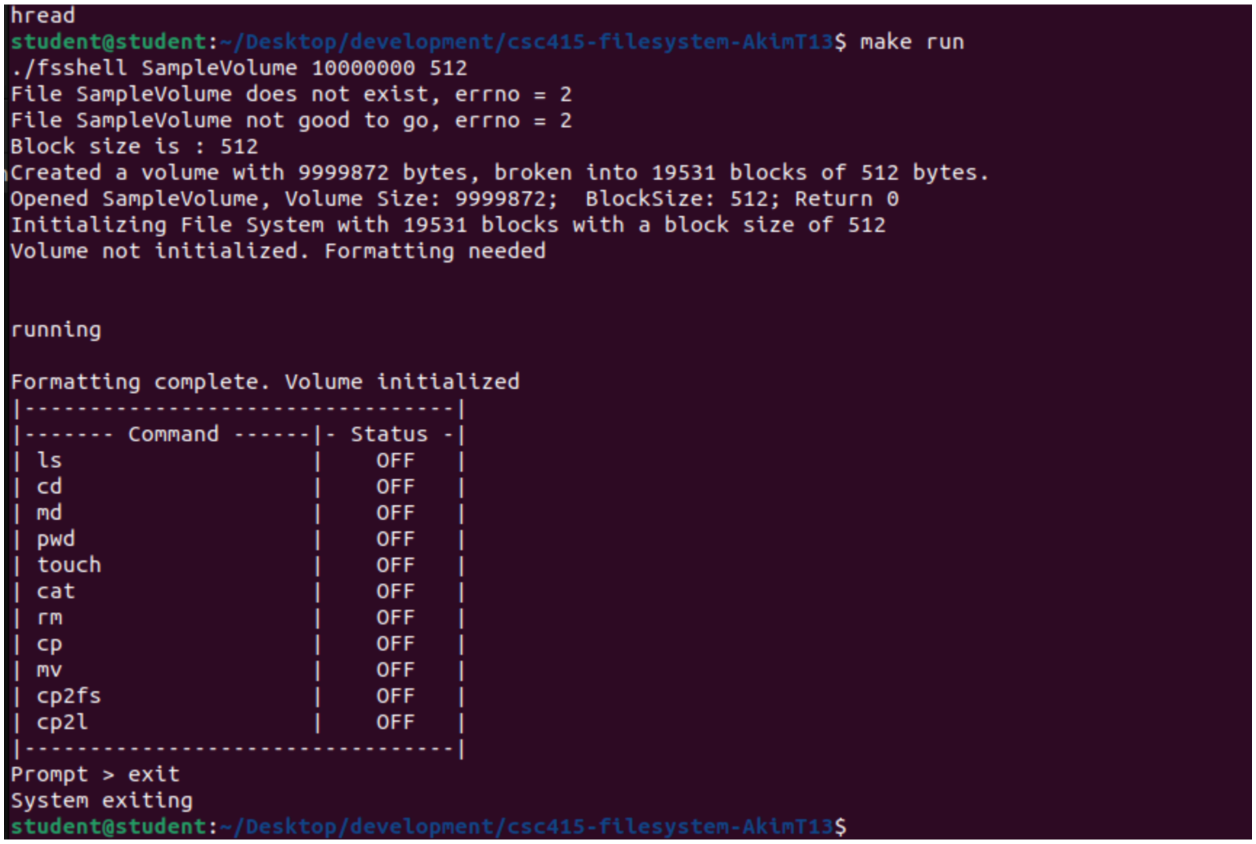


```
/development/csc415-filesystemstudent@student:~/Desktop/development/csc415-file
student@student:~/Desktop/development/csc415-filesystem-AkimT13$ make
gcc -c -o fsshell.o fsshell.c -g -I.
gcc -c -o fsInit.o fsInit.c -g -I.
gcc -c -o freeSpace.o freeSpace.c -g -I.
gcc -c -o initRoot.o initRoot.c -g -I.
gcc -o fsshell fsshell.o fsInit.o freeSpace.o initRoot.o fsLowM1.o -g -I. -lm -
 readline -l pthread
student@student:~/Desktop/development/csc415-filesystem-AkimT13$
```

Screen shot(s) of the execution of the program:

Zari Haidarian ID: 917423031
Akim Tarasov ID: 922761746
Diego Antunez ID: 922061514

Group Name: Coney Island
CSC415 Operating Systems



```
hread
student@student:~/Desktop/development/csc415-filesystem-AkimT13$ 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
Volume not initialized. Formatting needed


running

Formatting complete. Volume initialized
|---------------------------------|
|------- Command ------|- Status -|
| ls                   |    OFF   |
| cd                   |    OFF   |
| md                   |    OFF   |
| pwd                  |    OFF   |
| touch                |    OFF   |
| cat                  |    OFF   |
| rm                   |    OFF   |
| cp                   |    OFF   |
| mv                   |    OFF   |
| cp2fs                |    OFF   |
| cp2l                 |    OFF   |
|---------------------------------|
Prompt > exit
System exiting
student@student:~/Desktop/development/csc415-filesystem-AkimT13$
```