Cole Brausen ID: 918418987
Gursimran Singh ID: 922759471
Jose Rios ID:921160471
CSC415 Operating Systems

Milestone 1 Writeup

Team 3

Github Link: https://github.com/CSC415-2024-Spring/csc415-filesystem-Colorbomb1

Github Name: Colorbomb1

Cole Brausen ID: 918418987
Gursimran Singh ID: 922759471
Jose Rios ID:921160471
CSC415 Operating Systems

Milestone 1 Description:

Our plan for milestone 1 was to implement a custom file system with elements that are based on the ext4 implementation in the Linux file system. We implement a volume control block in (VCB) to act as the metadata center for the volume, this aids in detailing critical information needed to interpret the rest of the file system. We used a bitmap api in order to track the amount of free space we have and then read and write to disk using lbaRead and lbaWrite from fsLow.h based on the bitmap's logic. For the root directory entry, we incorporate that in DirectoryEntry.h and DirectoryEntry.c based off of the notes in lecture in which it looks for free allocated blocks of freespace and then writes the directory's information and metadata to the SampleVolume.

Approach:

This custom file system's intent is to utilize a hierarchical directory structure with the root directory being the beginning point of the rest of the file system. The Volume Control Block (VCB) utilizes several key fields to manage and validate the file system volume. Digital signatures, represented by **dig_sign1** and **dig_sign2**, ensure the volume's integrity and authenticity. With comparing these signatures to predefined constants (**DIGITAL_SIGNATURE1 and DIGITAL_SIGNATURE2**), the system can verify the volume's legitimacy, offering a basic yet effective method of verification. The **block_number** field is imperative in the description of the volume's structure; it represents the total count of blocks within the volume. This count is essential for interpreting the volume's capacity and for identifying specific blocks. The **block_size** field specifies the size of each individual block, a fundamental aspect that influences the overall capacity of the volume. Lastly, the **position_freespace** field indicates where free space starts within the volume which is an important detail for the allocation of new files or directories. This positioning is vital for efficient space management and for maintaining the volume's organization and accessibility In order to allocate space in the SampleVolume, we implement a free space bitmap to keep track of which blocks are free denoted as 0xFF which is the hex representation of 11111111 11111111 in each line of the block that is free, so 1 is a free block and 0 means that block is marked as used. For the Directory entry of root, we implemented functions that are crucial for managing the root directory of our file system. This includes initializing the root directory with special entries (. and ..), allocating and freeing blocks, and reading the root directory from the disk. To initialize root, we included the current directory

(.) and parent directory (..) entries alongside the root itself. This involves allocating a contiguous block of memory large enough to hold three DirectoryEntry structures. If the root directory is already initialized, we free the existing space before reallocating, preventing memory leaks and ensuring our root directory starts consistently.

Issues/Resolutions:

Most of the issues in this project came from implementing the Free Space Manager in the sense that we weren't implementing free and used blocks in a way that we understood. To fix this we denoted the freeblocks with 1's and used blocks as 0's, 1's being signified as 0xFF in the free blocks that we could write to.

The next issue we ran into with that was getting it to properly write to disk, it wasn't showing up in the hexdump because of how we were keeping track of the starting block for each allocation of freespace. However this ended up being an improper use of testing if it was working properly. Setting the blocks to free initially cleared this issue right up and showed the 5 free space blocks that we allocated for testing to be filled up with F's or 1's in this case.

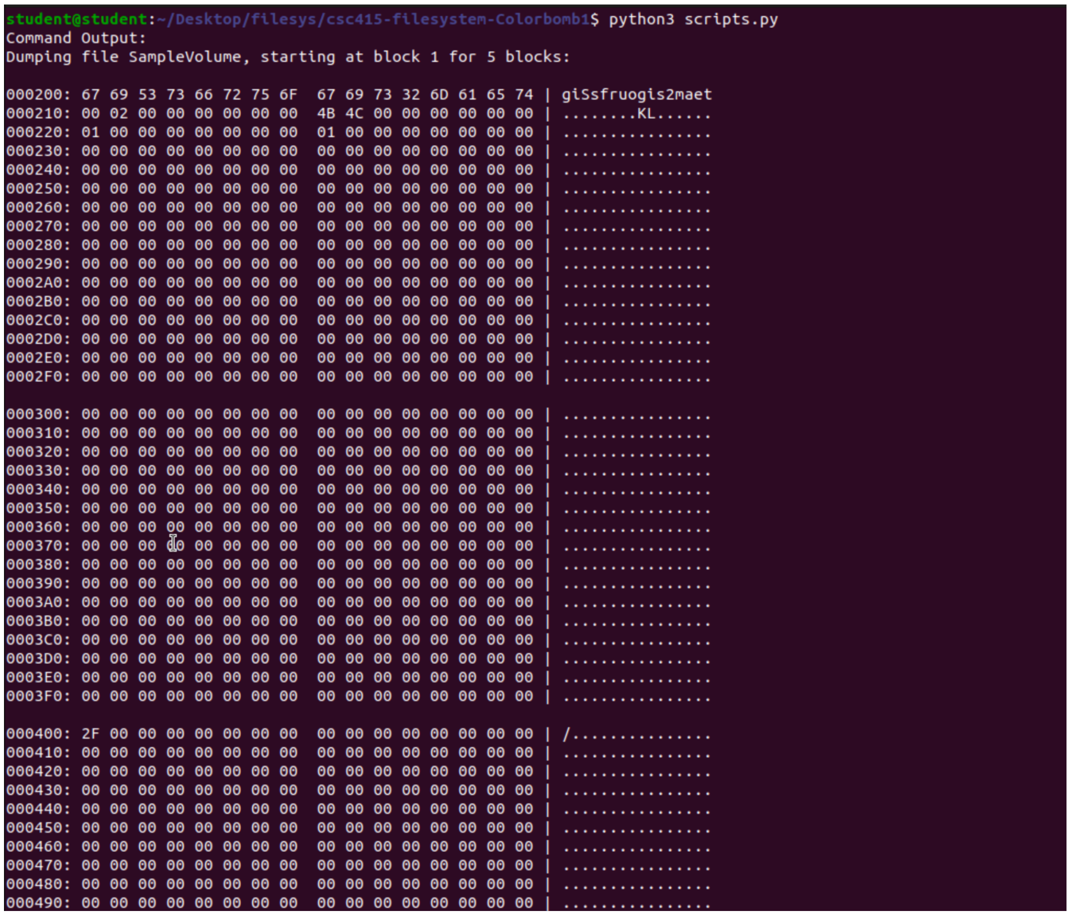
Our other issue was getting the directory entry to not only write to disk properly but to utilize the closest free block in our bitmap that we allocated. The resolution to this issue came in the form of rewriting of our function **check_block_free.** Because the logic was improperly implemented, we weren't actively getting the right free space block that had been marked as free to use. Before this fix it was ignoring the free blocks entirely and writing to disk after the bitmap had been initialized.

We do have an unresolved issue in the memory management part of this file system in the sense that we used a fixed value for blockSize in our DirectoryEntry, this needs to be handled dynamically as the fsInit could give any value for blockSize. I think for this fix that we will be implementing a function that handles the size that's given by calculating the size of the directory necessary for that given block size.

Analysis of Hexdump:

Cole Brausen ID: 918418987
Gursimran Singh ID: 922759471
Jose Rios ID:921160471
CSC415 Operating Systems

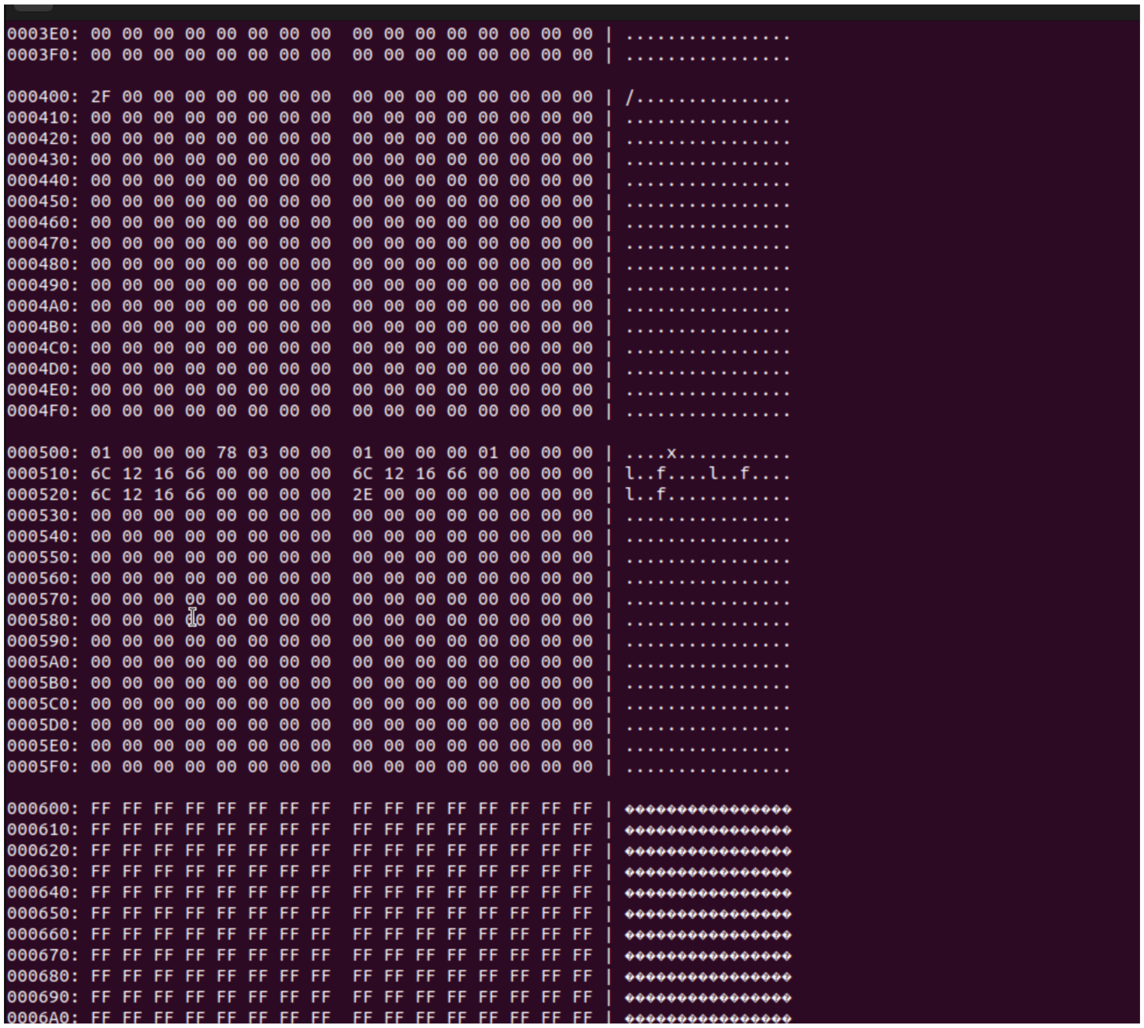
Below is a screenshot of our Hexdump that showcases the VCB and DirectoryEntry writes to disk:



```
student@student:~/Desktop/filesys/csc415-filesystem-Colorbomb1$ python3 scripts.py
Command Output:
Dumping file SampleVolume, starting at block 1 for 5 blocks:

000200: 67 69 53 73 66 72 75 6F  67 69 73 32 6D 61 65 74 | giSsfruogis2maet
000210: 00 02 00 00 00 00 00 00  4B 4C 00 00 00 00 00 00 | ........KL......
000220: 01 00 00 00 00 00 00 00  01 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 | ................

000400: 2F 00 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 | ................
```

As you can see in address 000200, there is a string of characters that signify our digital signature for our Volume Control Block to show that it was properly allocated to disk, the signatures are team2sig and ourfsSig joined together and put in reverse order because of the little endian structure we are given. At address 000210, we see 00 02 00 which indicates the block size address for our VCB which is 512. Further down is 4B 4C which indicates the block number address which is 19531 in hex.

Cole Brausen ID: 918418987
Gursimran Singh ID: 922759471
Jose Rios ID:921160471
CSC415 Operating Systems



```
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: 2F 00 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: 01 00 00 00 78 03 00 00  01 00 00 00 01 00 00 00 | ....x...........
000510: 6C 12 16 66 00 00 00 00  6C 12 16 66 00 00 00 00 | l..f....l..f....
000520: 6C 12 16 66 00 00 00 00  2E 00 00 00 00 00 00 00 | l..f............
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: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000610: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000620: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000630: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000640: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000650: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000660: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000670: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000680: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000690: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
0006A0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
```

At address 000400, we can see a hex value of 2F which indicated the directory name which is '/' in this case, from here it goes to another block to write the directory metadata, probably need to fix that so the metadata is written with the name itself to consolidate free space. At 000500, we have 01 00 00 00 78 03 00 in the first 8, the 01 signifies the 1 in our FS_type enumerations to signify the entry is a directory, like a file descriptor sort of. Since our directory entry size is 888, we can see that that's the little endian representation in hex of 78 03 00 00. In 000510, the 6C 12 16 66 represents the address of the date-time stamp for file creation of our Directory entry. This address contains the value of 1813124710 which is a formatted time of 2027-06-16 05:45 which isn't correct so we need to fix this as well before advancing to milestone 2.
]

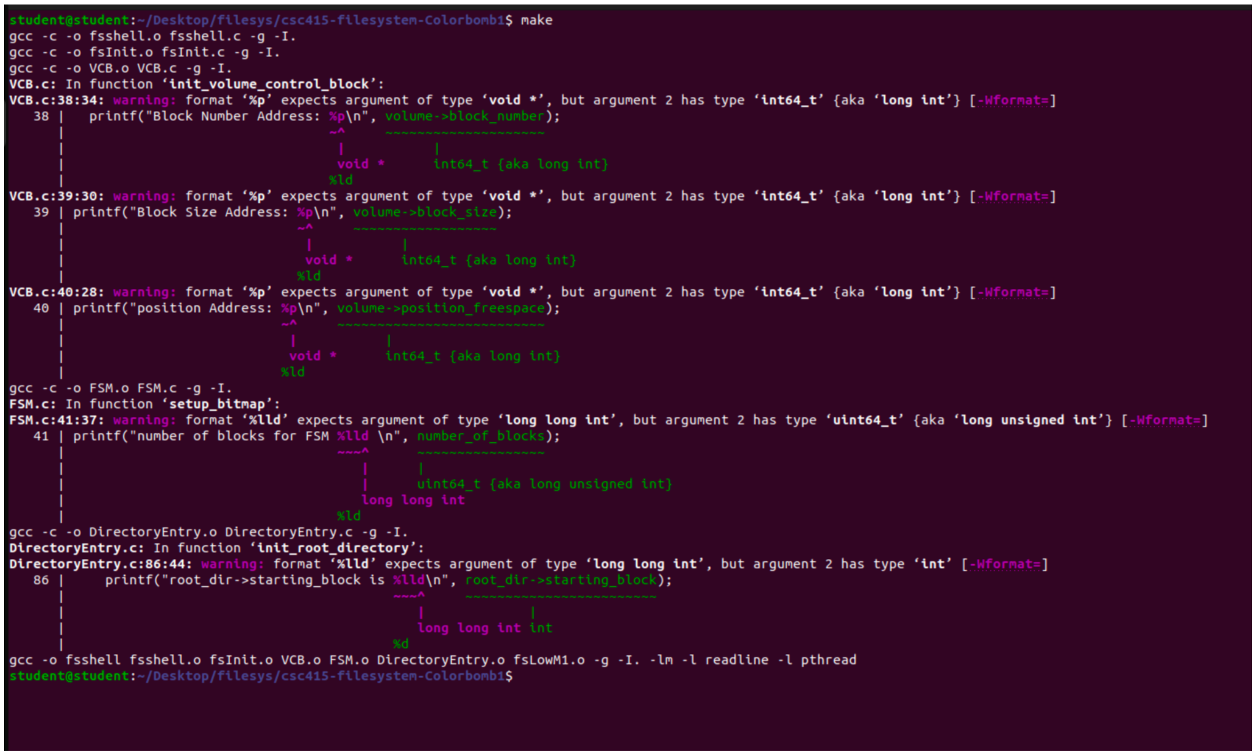
Cole Brausen ID: 918418987
Gursimran Singh ID: 922759471
Jose Rios ID:921160471
CSC415 Operating Systems

Who worked on what:

Member	**Contribution**
Jose Rios	Directory Entry
Gursimran Singh	Volume Control Block (VCB)
Cole Brausen	Free Space

Screenshots:

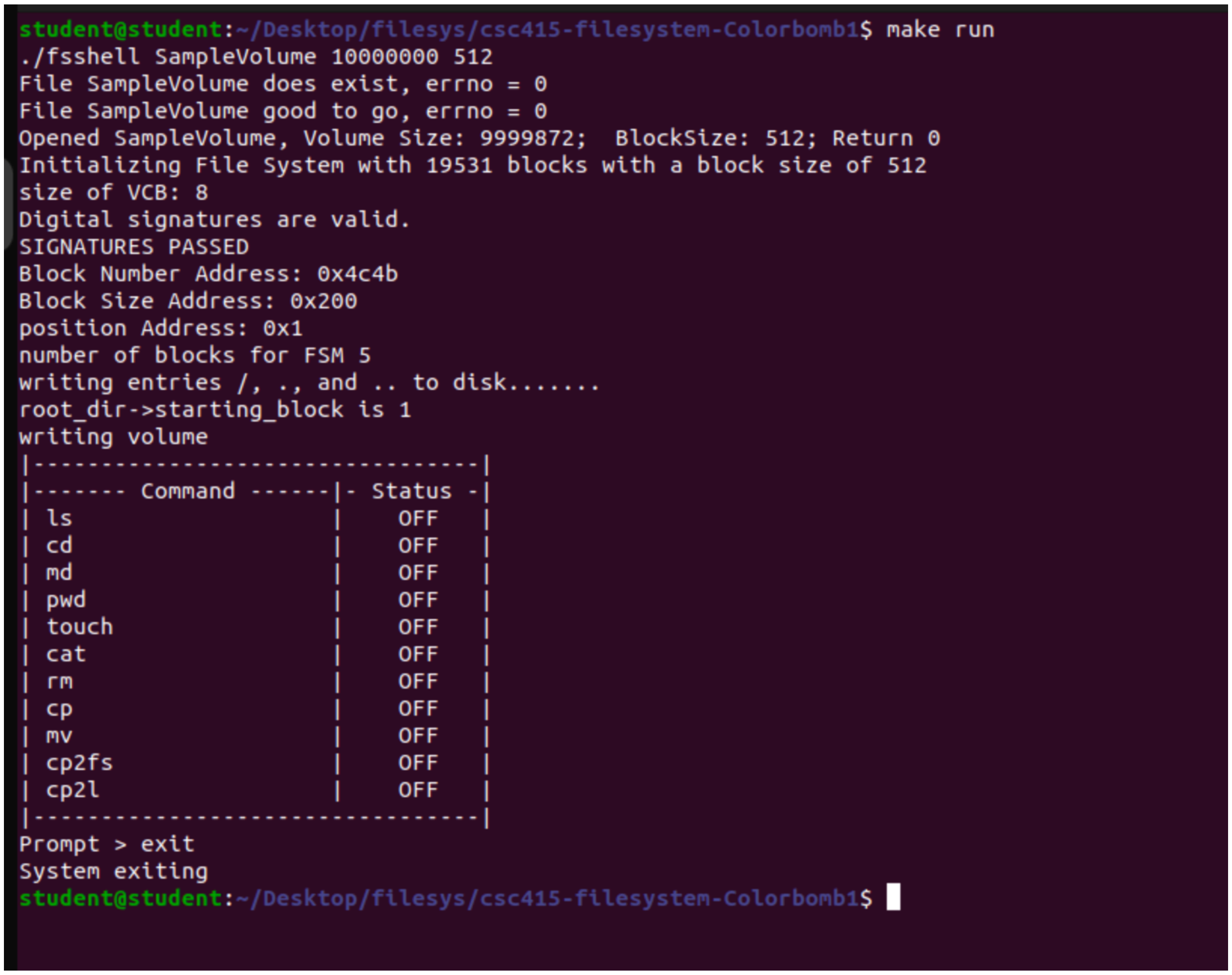
Proof of compilation:



```
student@student:~/Desktop/filesys/csc415-filesystem-Colorbomb1$ make
gcc -c -o fsshell.o fsshell.c -g -I.
gcc -c -o fsInit.o fsInit.c -g -I.
gcc -c -o VCB.o VCB.c -g -I.
VCB.c: In function ‘init_volume_control_block’:
VCB.c:38:34: warning: format ‘%p’ expects argument of type ‘void *’, but argument 2 has type ‘int64_t’ {aka ‘long int’} [-Wformat=]
   38 |   printf("Block Number Address: %p\n", volume->block_number);
      |                                 ~^     ~~~~~~~~~~~~~~~~~~~~
      |                                  |           |
      |                                  void *      int64_t {aka long int}
      |                                 %ld
VCB.c:39:30: warning: format ‘%p’ expects argument of type ‘void *’, but argument 2 has type ‘int64_t’ {aka ‘long int’} [-Wformat=]
   39 | printf("Block Size Address: %p\n", volume->block_size);
      |                             ~^     ~~~~~~~~~~~~~~~~~~
      |                              |           |
      |                              void *      int64_t {aka long int}
      |                             %ld
VCB.c:40:28: warning: format ‘%p’ expects argument of type ‘void *’, but argument 2 has type ‘int64_t’ {aka ‘long int’} [-Wformat=]
   40 | printf("position Address: %p\n", volume->position_freespace);
      |                           ~^     ~~~~~~~~~~~~~~~~~~~~~~~~~~
      |                            |           |
      |                            void *      int64_t {aka long int}
      |                           %ld
gcc -c -o FSM.o FSM.c -g -I.
FSM.c: In function ‘setup_bitmap’:
FSM.c:41:37: warning: format ‘%lld’ expects argument of type ‘long long int’, but argument 2 has type ‘uint64_t’ {aka ‘long unsigned int’} [-Wformat=]
   41 | printf("number of blocks for FSM %lld \n", number_of_blocks);
      |                                  ~~~^      ~~~~~~~~~~~~~~~~
      |                                     |      |
      |                                     |      uint64_t {aka long unsigned int}
      |                                     long long int
      |                                  %ld
gcc -c -o DirectoryEntry.o DirectoryEntry.c -g -I.
DirectoryEntry.c: In function ‘init_root_directory’:
DirectoryEntry.c:86:44: warning: format ‘%lld’ expects argument of type ‘long long int’, but argument 2 has type ‘int’ [-Wformat=]
   86 |     printf("root_dir->starting_block is %lld\n", root_dir->starting_block);
      |                                         ~~~^     ~~~~~~~~~~~~~~~~~~~~~~~~
      |                                            |             |
      |                                            long long int int
      |                                         %d
gcc -o fsshell fsshell.o fsInit.o VCB.o FSM.o DirectoryEntry.o fsLowM1.o -g -I. -lm -l readline -l pthread
student@student:~/Desktop/filesys/csc415-filesystem-Colorbomb1$
```

Cole Brausen ID: 918418987
Gursimran Singh ID: 922759471
Jose Rios ID:921160471
CSC415 Operating Systems

Output:



```
student@student:~/Desktop/filesys/csc415-filesystem-Colorbomb1$ 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
size of VCB: 8
Digital signatures are valid.
SIGNATURES PASSED
Block Number Address: 0x4c4b
Block Size Address: 0x200
position Address: 0x1
number of blocks for FSM 5
writing entries /, ., and .. to disk.......
root_dir->starting_block is 1
writing volume
|---------------------------------|
|------- 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/filesys/csc415-filesystem-Colorbomb1$
```