File System Project Milestone 1

Team name: Robert Bierman

csc415-2024-summer-csc415-filesystem-CSC415-FileSystem created by GitHub Classroom

Team members:

Bryan Yao	922709642	https://github.com/xiangco
Bryan Lee	922649673	https://github.com/BryanL43
Matt Stoffel	923127111	https://github.com/MattRStoffel
Kevin Lam	922350179	https://github.com/Pot4ssium

File System - Milestone 1

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

Description
The assignment is to format a volume for the rest of the file system project, including writing the Volume Control Block (VCB), starting on our free space manager, and initializing the root directory.

VCB:
The VCB is a repository for information about the volume. It will be the first thing loaded into memory when the file system starts. Once loaded into memory, the program checks the signature to see if the volume is already formatted. The program will initialize the volume and free space if the volume is not formatted. If the volume is formatted, the system will load free space and the root directory into memory.

Free Space structure:
We decided to store our free space in a file allocation table (FAT). This is an external linked list structure, where the FAT points to the next node (array index for array implementation of linked list), and each node represents a block in volume. Every entry in the FAT is a value pointing to the next block; entries with a sentinel value of 0xFFFFFFFD indicate the end of the linked list.

The VCB block in the volume is marked with the sentinel value to indicate a used block. The following 153 blocks store the FAT, ending with the sentinel value. Since we initialized and wrote the root directory into the disk, this process is shown in the FAT. After the free space list is stored, we can see another sentinel value after four blocks, indicating the end of the root directory blocks.

Directory:
We create a new file that serves as a directory entry marked as a directory. This marker indicates that the file is part of our file system. Our directory is an array of directory entries, and it is optimized regarding the entry padding, making it easier to analyze in the hex dump. We store the root directory right after the free space FAT, and the root takes up four blocks.

To set up a new directory, we calculate the number of blocks needed to store the directory entries. We ensure the entries fit within the allocated blocks, utilizing any extra space to minimize wastage. All directory entries are initially marked as unused, except for (.) and (..).

The first entry (.) is initialized with relevant information and (..). If it's the root directory, (..) is the same as (.), otherwise, it points to the parent directory. After assigning the dot and “dot dot,” we write the new directory structure to disk and return a pointer to the new directory for use in other functions.

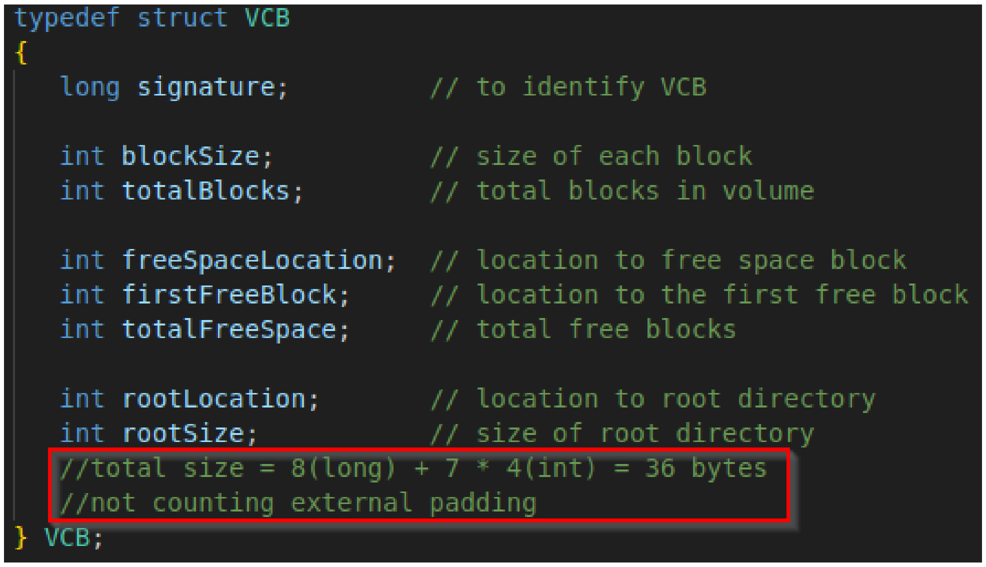
<u>Hexdumps:</u>
Note: The entire volume is displayed in the following hex dumps is in little-endian format.

All values in the hex dump are shifted by 0X200 or 512 bytes or a block. This is because the low-level system's partition table is stored at block 0, which shifts the start of our volume by 1 block.

Volume Control Block

```
000200: 52 42 69 65 72 6D 61 6E  00 02 00 00 4B 4C 00 00 | RBierman....KL..
000210: 01 00 00 00 9E 00 00 00  AD 4B 00 00 9A 00 00 00 | ....�...�K..�...
000220: 04 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 | ................
```

Why do we know this is the Volume Control Block? One giveaway is the structure stored in this block. By examining the structure alone for the VCB in our code, we see that the structure of the VCB should occupy 36 bytes.



```
typedef struct VCB
{
    long signature;          // to identify VCB

    int blockSize;           // size of each block
    int totalBlocks;         // total blocks in volume

    int freeSpaceLocation;   // location to free space block
    int firstFreeBlock;      // location to the first free block
    int totalFreeSpace;      // total free blocks

    int rootLocation;        // location to root directory
    int rootSize;            // size of root directory
    //total size = 8(long) + 7 * 4(int) = 36 bytes
    //not counting external padding
} VCB;
```

Counting up all the highlighted portions from 0x000200 to 0x000223, you get exactly 36 bytes, which matches the predicted structure of our VCB. Another giveaway is that we LBAwrite the VCB in the sample volume at block location 0, so if we hex dump starting at that location, it should give us the VCB. We further verify the information by breaking down each highlighted section to match our obtained values. We achieve the following breakdown by checking the hex dump and remembering to convert from little-endian to big-endian.

- 0x000200-0x000207
    - The signature we picked is `52 42 69 65 72 6D 61 6E`, this value when displayed in the hexdump's ASCII portion will show up as RBierman, which is our group name.
    - To check if this matches our decimal signature:
        - small endian -> big endian
            - `52 42 69 65 72 6D 61 6E -> 6E 61 6D 72 65 69 42 52`
            - 6E 61 6D 72 65 69 42 52 `in decimal 7953758755008102994` (our chosen signature)
- 0x000208-0x00020B
    - The size of each block is 512 bytes.
    - small endian -> big endian
    - `00 02 00 00 -> 00 00 02 00`
    - `00 00 02 00 in decimal is 512`
- 0x00020C-0x00020F
    - The total number of blocks in the volume is 19531.
    - small endian -> big endian
    - `4B 4C 00 00 -> 00 00 4C 4B`
    - `00 00 4C 4B in decimal 19531`
- 0x000210-0x000213
    - The location of the free space block is at block 1.
    - small endian -> big endian
    - `01 00 00 00 -> 00 00 00 01`
    - `00 00 00 01 in decimal 1`
- 0x000214-0x000217
    - The first free block is at block 158, because the fat table takes up 153 blocks to keep track of free space, and the root directory takes up another 4 blocks.
    - small endian -> big endian
    - `9E 00 00 00 -> 00 00 00 9E`
    - `00 00 00 01 in decimal 158`
- 0x000218-0x00021B
    - After initializing our file system, the total number of free blocks is 0x4BAD = 19373.
    - small endian -> big endian
    - `AD 4B 00 00 -> 00 00 4B AD`
    - `00 00 4B AD in decimal 19373`
- 0x00021C-0x00021F
    - The root directory is located at block 154.
    - Refer to the free space and root directory section to show that it is there
    - small endian -> big endian
    - `9A 00 00 00 -> 00 00 00 9A`
    - `00 00 00 9A in decimal 154`

- 0x000220-0x000223
    - The size of the root directory is 4 blocks.
    - Refer to the free space and root directory section to show that it is there
    - small endian -> big endian
    - `04 00 00 00 -> 00 00 00 04`
    - `00 00 00 04 in decimal 4`

The remainder of the block is zeros because no other data is stored in the Volume control block.

Free Space

<u>This is a portion of the second block on the volume.</u>

```
000400: FD FF FF FF 02 00 00 00  03 00 00 00 04 00 00 00 | ����............
000410: 05 00 00 00 06 00 00 00  07 00 00 00 08 00 00 00 | ................
000420: 09 00 00 00 0A 00 00 00  0B 00 00 00 0C 00 00 00 | ................
000430: 0D 00 00 00 0E 00 00 00  0F 00 00 00 10 00 00 00 | ................
000440: 11 00 00 00 12 00 00 00  13 00 00 00 14 00 00 00 | ................
000450: 15 00 00 00 16 00 00 00  17 00 00 00 18 00 00 00 | ................
000460: 19 00 00 00 1A 00 00 00  1B 00 00 00 1C 00 00 00 | ................
000470: 1D 00 00 00 1E 00 00 00  1F 00 00 00 20 00 00 00 | ............ ...
000480: 21 00 00 00 22 00 00 00  23 00 00 00 24 00 00 00 | !..."...#...$...
000490: 25 00 00 00 26 00 00 00  27 00 00 00 28 00 00 00 | %...&...'...(...
0004A0: 29 00 00 00 2A 00 00 00  2B 00 00 00 2C 00 00 00 | )...*...+...,...
0004B0: 2D 00 00 00 2E 00 00 00  2F 00 00 00 30 00 00 00 | -......./...0...
0004C0: 31 00 00 00 32 00 00 00  33 00 00 00 34 00 00 00 | 1...2...3...4...
0004D0: 35 00 00 00 36 00 00 00  37 00 00 00 38 00 00 00 | 5...6...7...8...
0004E0: 39 00 00 00 3A 00 00 00  3B 00 00 00 3C 00 00 00 | 9...:...;...<...
0004F0: 3D 00 00 00 3E 00 00 00  3F 00 00 00 40 00 00 00 | =...>...?...@...
```

- 0x000400-0x000403
    - We have stored the VCB starting from block 1; thus, we should expect to see the sentinel value on the first block, indicating that this block is used in the FAT.

- 0x000404-0x000663
    - From the second integer until the end of the FAT are values pointing to the next index, like a linked list. These are stored as integers, explaining why they are indicated with 4 hex values in the hex dump, 4 bytes for each link. For example, we can see that the next 4 bytes after the VCB (highlighted yellow), 0x00000002 in hex and 2 in decimal, points to the second index, and the next 4 points to the third index. This link will continue until we reach the end of the FAT indicated by another sentinel value.

This is a separate portion of a different block, skipping to block 3 in the hex dump.

```
000600: 81 00 00 00 82 00 00 00  83 00 00 00 84 00 00 00 | �...�...�...�...
000610: 85 00 00 00 86 00 00 00  87 00 00 00 88 00 00 00 | �...�...�...�...
000620: 89 00 00 00 8A 00 00 00  8B 00 00 00 8C 00 00 00 | �...�...�...�...
000630: 8D 00 00 00 8E 00 00 00  8F 00 00 00 90 00 00 00 | �...�...�...�...
000640: 91 00 00 00 92 00 00 00  93 00 00 00 94 00 00 00 | �...�...�...�...
000650: 95 00 00 00 96 00 00 00  97 00 00 00 98 00 00 00 | �...�...�...�...
000660: 99 00 00 00 FD FF FF FF  9B 00 00 00 9C 00 00 00 | �...�����...�...
000670: 9D 00 00 00 FD FF FF FF  9F 00 00 00 A0 00 00 00 | �...�����...�...
000680: A1 00 00 00 A2 00 00 00  A3 00 00 00 A4 00 00 00 | �...�...�...�...
000690: A5 00 00 00 A6 00 00 00  A7 00 00 00 A8 00 00 00 | �...�...�...�...
0006A0: A9 00 00 00 AA 00 00 00  AB 00 00 00 AC 00 00 00 | �...�...�...�...
0006B0: AD 00 00 00 AE 00 00 00  AF 00 00 00 B0 00 00 00 | �...�...�...�...
0006C0: B1 00 00 00 B2 00 00 00  B3 00 00 00 B4 00 00 00 | �...�...�...�...
0006D0: B5 00 00 00 B6 00 00 00  B7 00 00 00 B8 00 00 00 | �...�...�...�...
0006E0: B9 00 00 00 BA 00 00 00  BB 00 00 00 BC 00 00 00 | �...�...�...�...
0006F0: BD 00 00 00 BE 00 00 00  BF 00 00 00 C0 00 00 00 | �...�...�...�…
```

- 0x000664-0x000667
  - Knowing that our volume size is 19531 blocks with block sizes 512, we calculated that our FAT will take up 153 blocks. Index 153 in the FAT would be a sentinel value indicating the end of the linked list. Having the value 0xFFFFFFFD at this location matches our expectations, since the index before the sentinel value is 0x00000099, which is 153 in decimal, pointing to the sentinel value, matching the size of the FAT.
- 0x000668-0x000673
  - These bytes represent the linked list of the root directory, indicating that the root directory occupies these blocks.
- 0x000674-0x000677
  - Because we initialized our root directory with 20 directory entries by default, and the actual entries are calculated to fill in to accommodate the size of blocks, we should expect it to take 4 blocks when we first initialize our filesystem. This sentinel value 0xFFFFFFFD is evidence of that since after 4 blocks from the FAT sentinel value, we can see another one indicating the end of the root directory.
- 0x000678-0x01352B
  - This is our entire free space in the FAT; it shows how much free space we have through a linked list system right after the root directory ends. The VCB shows that this will take up 152 blocks as we represent the volume's free space, which is 19373 blocks.

This is another separate portion of a different block, skipping to block 153 in the hex dump.

```
013500: 41 4C 00 00 42 4C 00 00  43 4C 00 00 44 4C 00 00 | AL..BL..CL..DL..
013510: 45 4C 00 00 46 4C 00 00  47 4C 00 00 48 4C 00 00 | EL..FL..GL..HL..
013520: 49 4C 00 00 4A 4C 00 00  4B 4C 00 00 FD FF FF FF | IL..JL..KL..����
013530: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013540: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013550: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013560: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013570: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013580: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013590: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0135A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0135B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0135C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0135D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0135E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0135F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

- 0x01352C-0x01352F
  - The sentinel value for total free space indicates the end of the free space linked list.

Root Directory

```
013600: 13 9D 8E 66 00 00 00 00  13 9D 8E 66 00 00 00 00 | .��f.....��f....
013610: 2E 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013620: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013630: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013640: 00 00 00 64 D0 07 00 00  9A 00 00 00 00 00 00 00 | ...d.(..�.......
013650: 13 9D 8E 66 00 00 00 00  13 9D 8E 66 00 00 00 00 | .��f.....��f....
013660: 2E 2E 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013670: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013680: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013690: 00 00 00 64 D0 07 00 00  9A 00 00 00 00 00 00 00 | ...d.(..�.......
0136A0: 13 9D 8E 66 00 00 00 00  13 9D 8E 66 00 00 00 00 | .��f.....��f....
0136B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0136C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0136D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0136E0: 00 00 00 20 00 00 00 00  FF FF FF FF 00 00 00 00 | ... ....����....
0136F0: 13 9D 8E 66 00 00 00 00  13 9D 8E 66 00 00 00 00 | .��f.....��f....
```

The root directory starts at offset 0x13600 when our volume size is 9999872 and the block size is 512. Above is half of block 154; it shows the “dot dot”, dot, and the first empty directory entry in the root directory.

- 0x13600 - 0x1360F
  - Starting at the beginning of the block, because of the structure of our directory entry struct the first 2 values are the date created and last modified. 0x668E9D13 = 1720622355 which is the number of seconds since January 1st 1970 or July 10, 2024 at 14:39:15 UTC
- 0x13610-0x013642
  - The filename dot or “2E” is "." in ASCII and the rest is just 00.
- 0x013643
  - Is a directory marker 0x64 in ASCII is ‘d’ which indicates that it indeed is a directory
- 0x013644-0x013647
  - 0x000007D0 is the size of the directory which is 2000 bytes. Our directory entry takes up 80 bytes. We are initializing a minimum of 20 entries. That requires 4 blocks so we fill up the remaining space with as many spare directory entries as possible bringing that up to 25, or 2000 bytes of data, 0x000007D0.
- 0x013648-0x01365B
  - The block where the root directory is stored will vary with the volume size because the free space manager will also vary. In this case 0x9A = 154. In other words, the root directory is located on block 154. It is its own parent.
- 0x01364C-0x01365F
  - Permissions are not very important right now, but as part of our structure, they are all set to 0. Everything after this is information for the next directory entry.
- 0x013698-0x01369B
  - This is referencing itself, the root directory.
- 0x013660
  - This is the name field for the dot-dot or “2E 2E” directory entry the zeros are the remainder of the space for the DEs name
- 0x0136E8-0x0136EB
  - Empty directories are initialized with a location of -1

<u>Workload distribution table</u>:

Names	Components worked on
Bryan Yao	VCB
Bryan Lee and Kevin Lam	Free Space
Matt Stoffel	Directory

<u>Team Description</u>

How the team worked together:

We would have daily meetings, or open voice calls on Discord. Since not everyone is accessible daily, we would join the calls whenever we are free and work in groups of two or more. Since we divided the tasks, each team member focused on their parts and reconciled with the team to link dependent parts.

How did we meet:
Through our discord channel's voice chat.

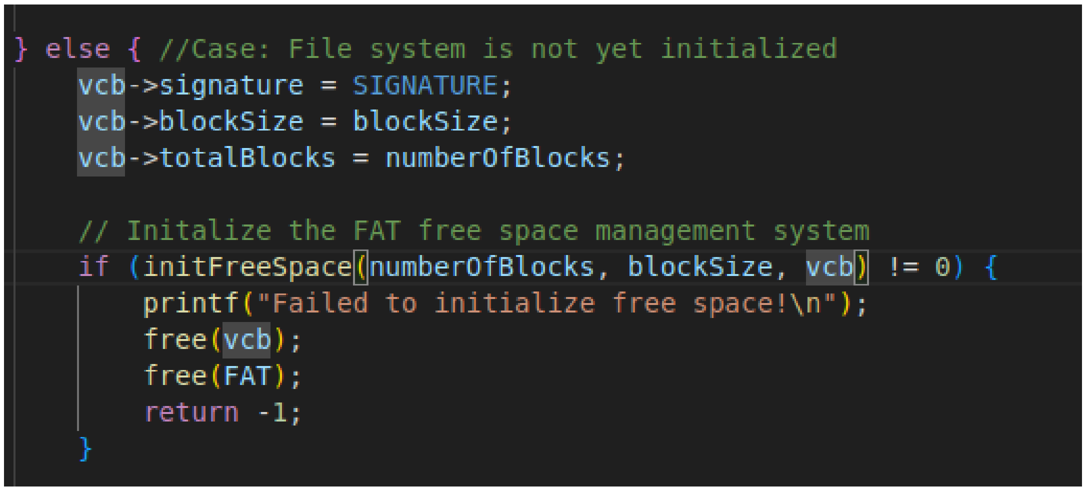
How often the team met:
The team met up every day.

Dividing up the task:
We divided up the tasks according to the workload needed to accomplish them. Since the lecture thoroughly discussed the code structure for directory entries, and the VCB mainly concerns populating the structure, we assigned these tasks to one person each. On the other hand, implementing free space requires more research and effort. Since the VCB and initializing the root directory depend on the free space, more time and effort were necessary to develop the FAT system accurately, requiring two people to work on this task.

Issues and Resolutions

During development, we encountered an issue regarding access to the volume control block, file allocation table, and root directory. The first challenge arose during the initialization of free space and the root directory. Here, we assign initial values like total free space to the volume control block. The second challenge emerged when attempting to read data directly into the FAT blocks during the filesystem initialization process. The read (LBAread) occurs in the fsInit file; however, since another file, freeSpace.c, handles free space initialization, we initially attempted to use pass-by-reference to share the pointer across various methods in different files. Unfortunately, this approach proved problematic for effectively maintaining and accessing the VCB, FAT, and root directory.

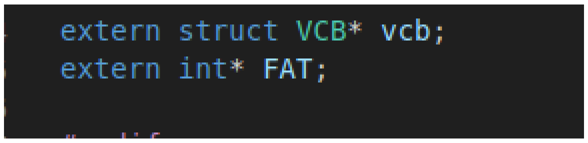
An image of what we initially did:



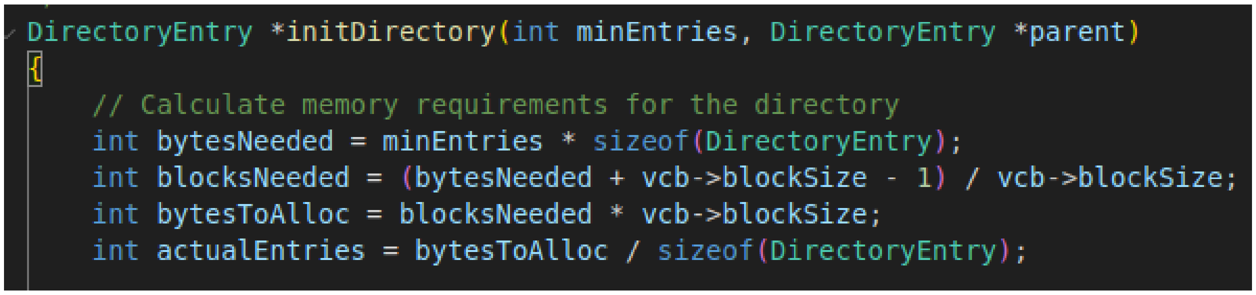
```
} else { //Case: File system is not yet initialized
    vcb->signature = SIGNATURE;
    vcb->blockSize = blockSize;
    vcb->totalBlocks = numberOfBlocks;

    // Initalize the FAT free space management system
    if (initFreeSpace(numberOfBlocks, blockSize, vcb) != 0) {
        printf("Failed to initialize free space!\n");
        free(vcb);
        free(FAT);
        return -1;
    }
```

We resolved this by turning the VCB, FAT, and root directories into global variables accessible by other files using the extern keyword. This allows us to allocate memory for these structures and variables in the “fsInit” file while accessing and initializing the data in a separate file.

Image of the solution:



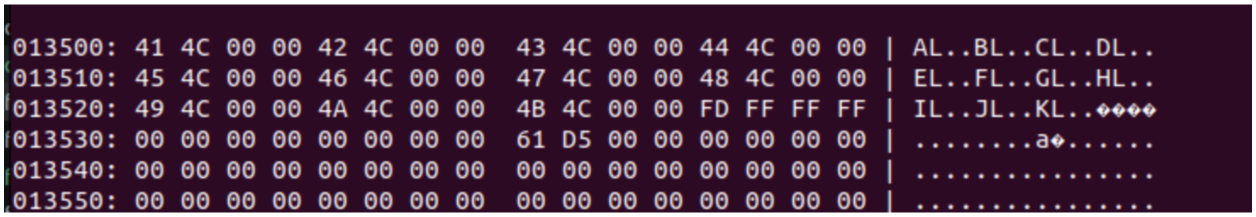
```
extern struct VCB* vcb;
extern int* FAT;
```



```
DirectoryEntry *initDirectory(int minEntries, DirectoryEntry *parent)
{
    // Calculate memory requirements for the directory
    int bytesNeeded = minEntries * sizeof(DirectoryEntry);
    int blocksNeeded = (bytesNeeded + vcb->blockSize - 1) / vcb->blockSize;
    int bytesToAlloc = blocksNeeded * vcb->blockSize;
    int actualEntries = bytesToAlloc / sizeof(DirectoryEntry);
```

Another issue was memory leakage at the end of the FAT hex dump (hex dumping block 154). Despite marking the end of the FAT with a sentinel value, two extra hexes were beyond the scope of the FAT data.

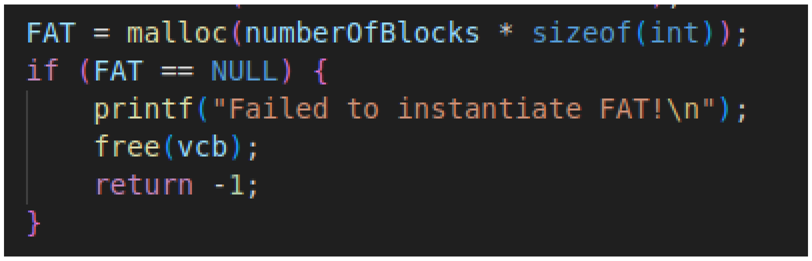
Image of the issue:



```
013500: 41 4C 00 00 42 4C 00 00  43 4C 00 00 44 4C 00 00 | AL..BL..CL..DL..
013510: 45 4C 00 00 46 4C 00 00  47 4C 00 00 48 4C 00 00 | EL..FL..GL..HL..
013520: 49 4C 00 00 4A 4C 00 00  4B 4C 00 00 FD FF FF FF | IL..JL..KL..����
013530: 00 00 00 00 00 00 00 00  61 D5 00 00 00 00 00 00 | ........a�......
013540: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013550: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

We initially thought the FAT was adding extra links beyond the end of free space, but that wasn't the issue. After some debugging, we found the solution and problem in the most unlikely place. We thought the space necessary for the FAT system was only the number of blocks * the size of the integer, as we needed to store the "numberOfBlock" amount of links.

Image of the error:

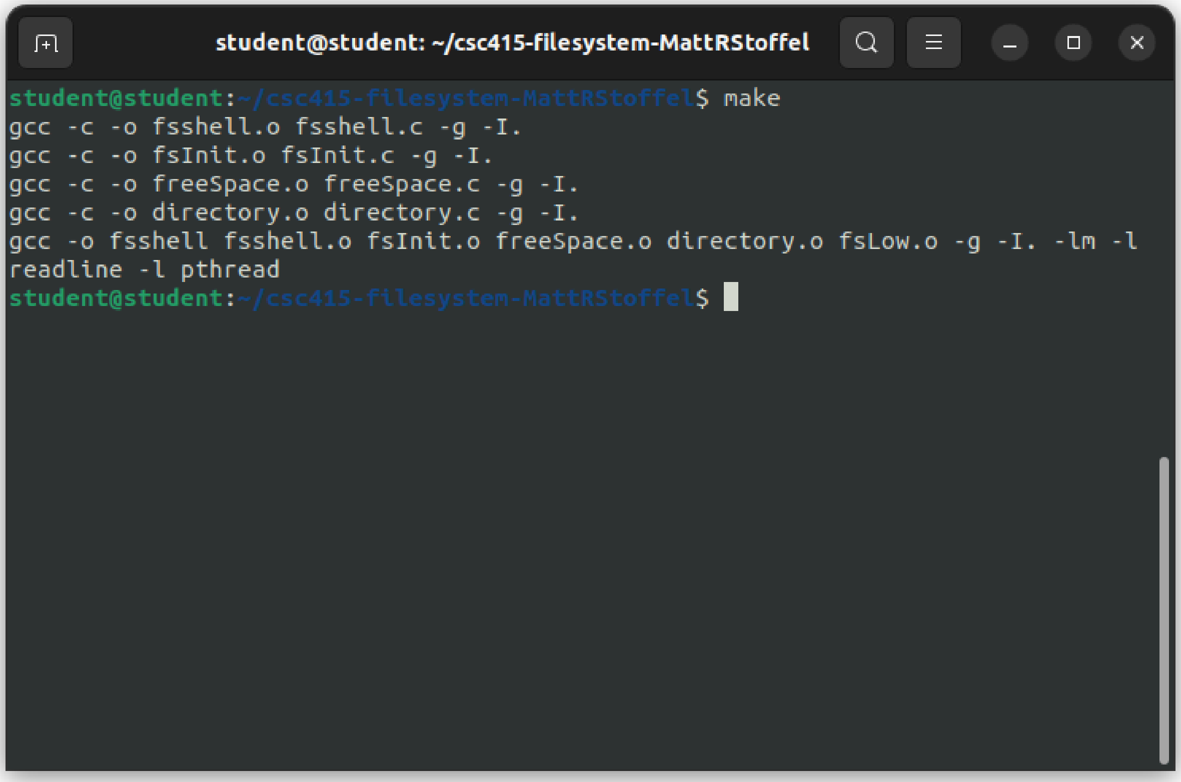


```
FAT = malloc(numberOfBlocks * sizeof(int));
if (FAT == NULL) {
    printf("Failed to instantiate FAT!\n");
    free(vcb);
    return -1;
}
```

However, this was fixed simply by correctly calculating the correct block space required for the FAT (shown below):

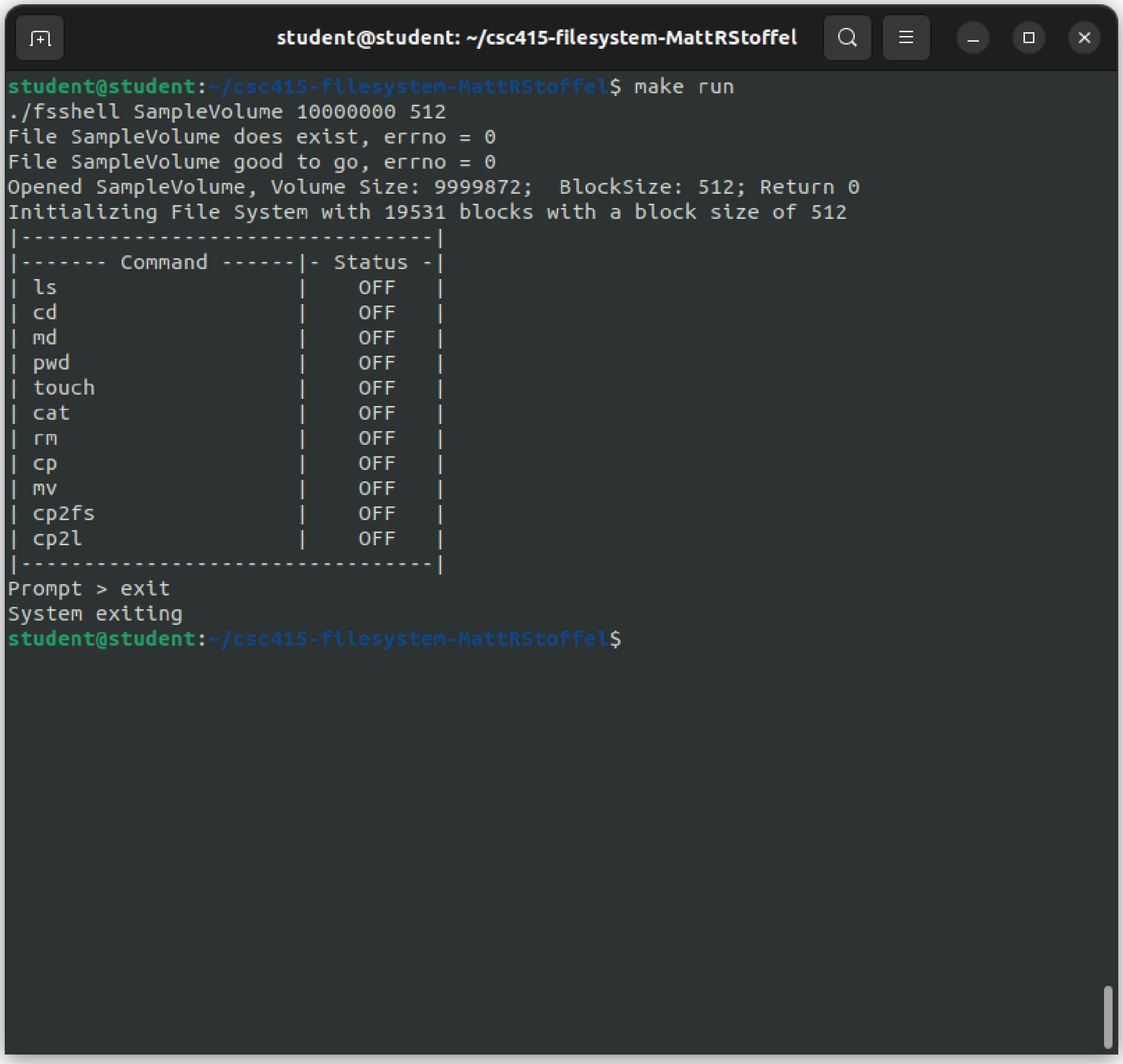
```
int blocksNeeded = ((numberOfBlocks * sizeof(int)) + blockSize - 1) / blockSize;
FAT = malloc(blocksNeeded * blockSize);
if (FAT == NULL) {
```

<u>Screenshot of compilation:</u>



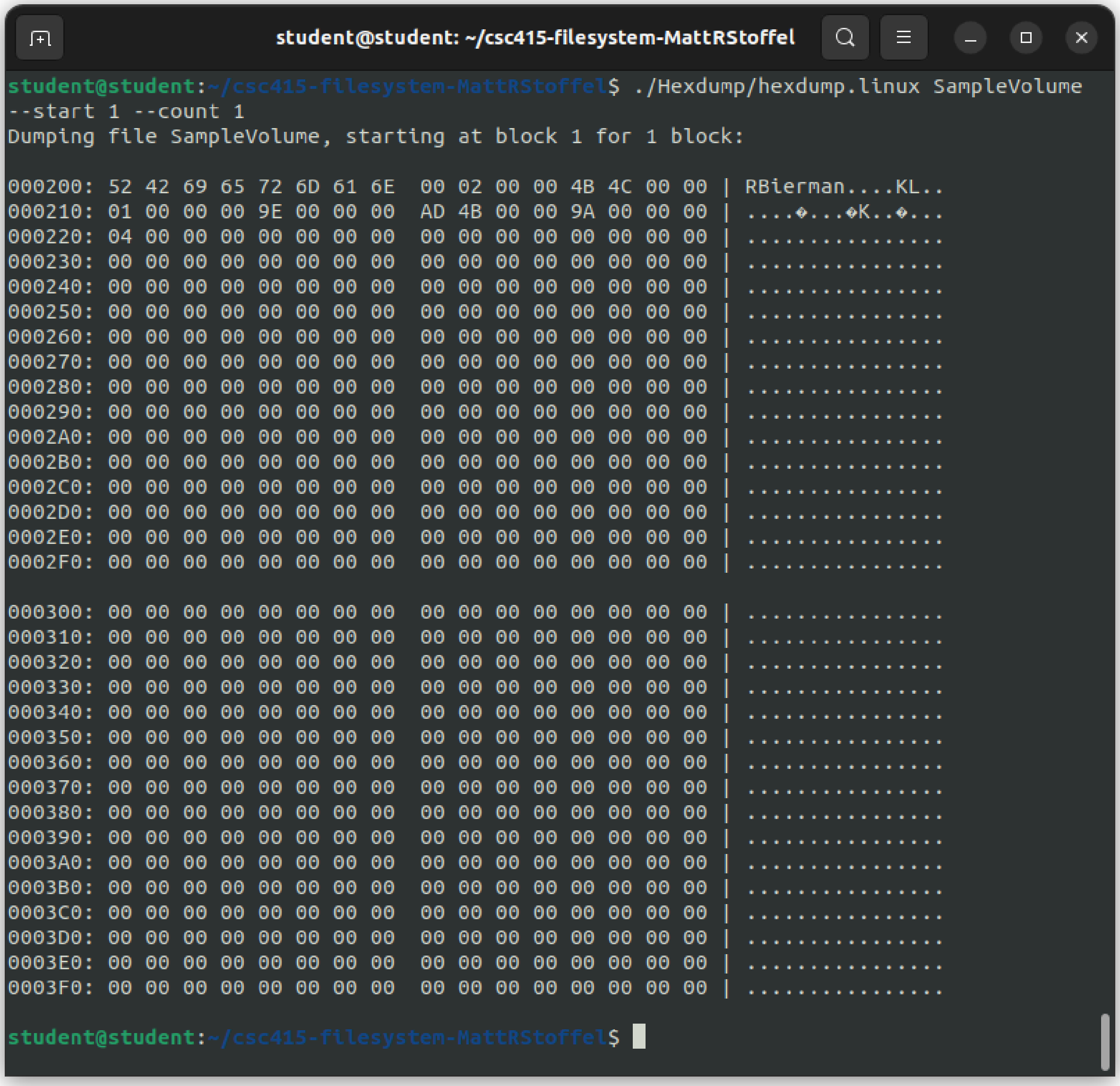
```
student@student:~/csc415-filesystem-MattRStoffel$ make
gcc -c -o fsshell.o fsshell.c -g -I.
gcc -c -o fsInit.o fsInit.c -g -I.
gcc -c -o freeSpace.o freeSpace.c -g -I.
gcc -c -o directory.o directory.c -g -I.
gcc -o fsshell fsshell.o fsInit.o freeSpace.o directory.o fsLow.o -g -I. -lm -l
readline -l pthread
student@student:~/csc415-filesystem-MattRStoffel$
```

Screenshot(s) of the execution of the program:



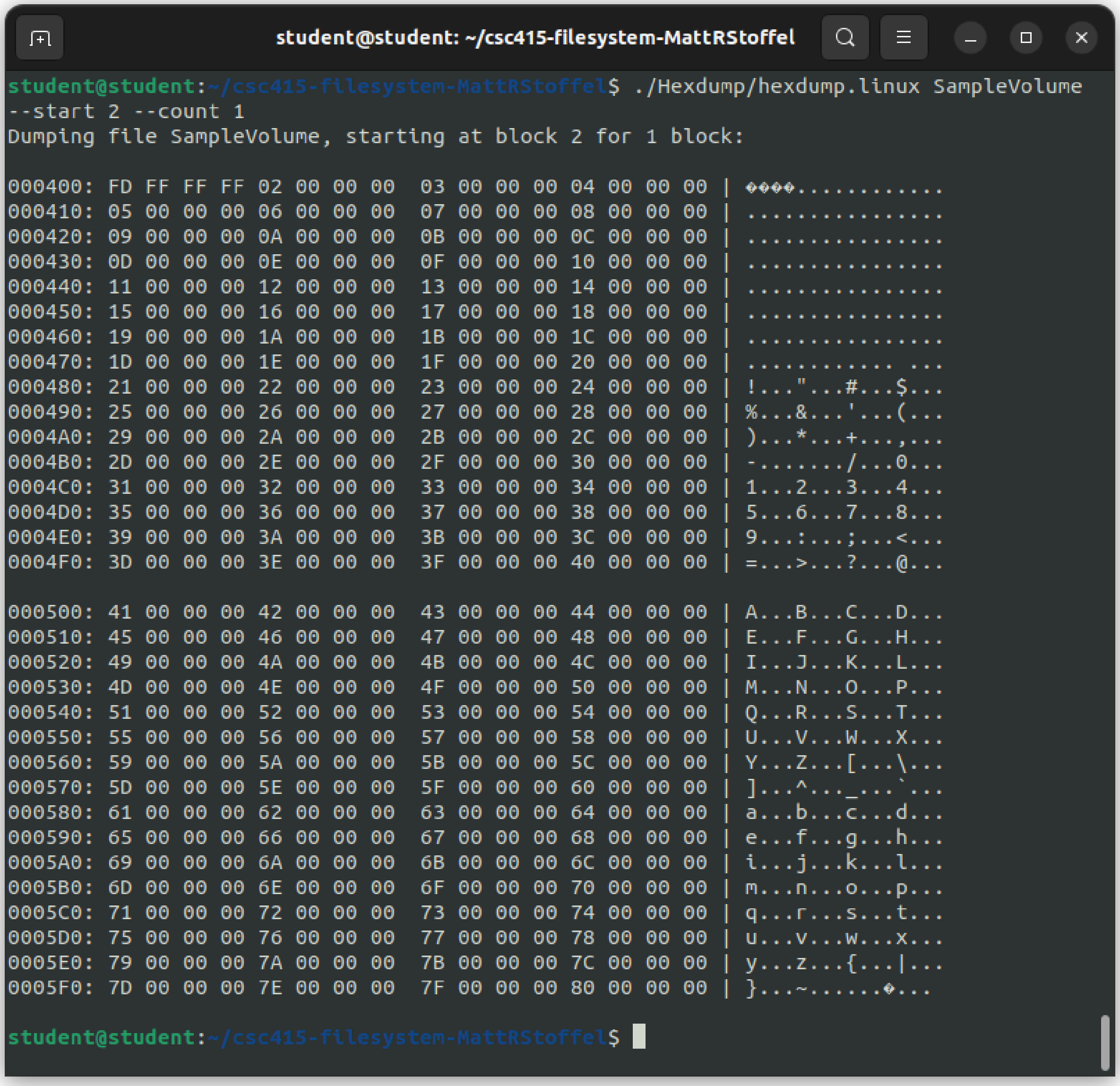
student@student: ~/csc415-filesystem-MattRStoffel

```
student@student:~/csc415-filesystem-MattRStoffel$ 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                   |    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:~/csc415-filesystem-MattRStoffel$
```

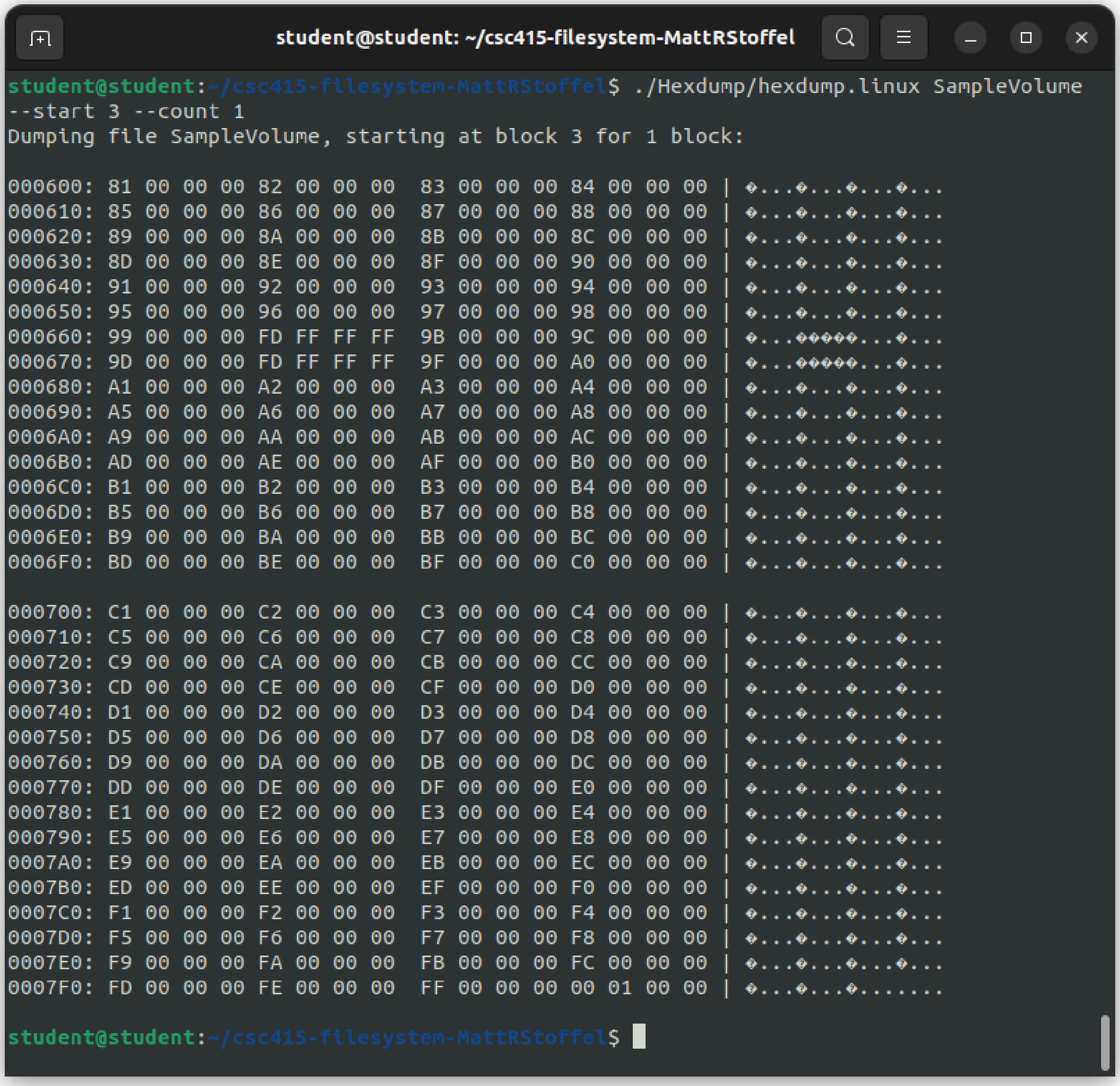


student@student: ~/csc415-filesystem-MattRStoffel

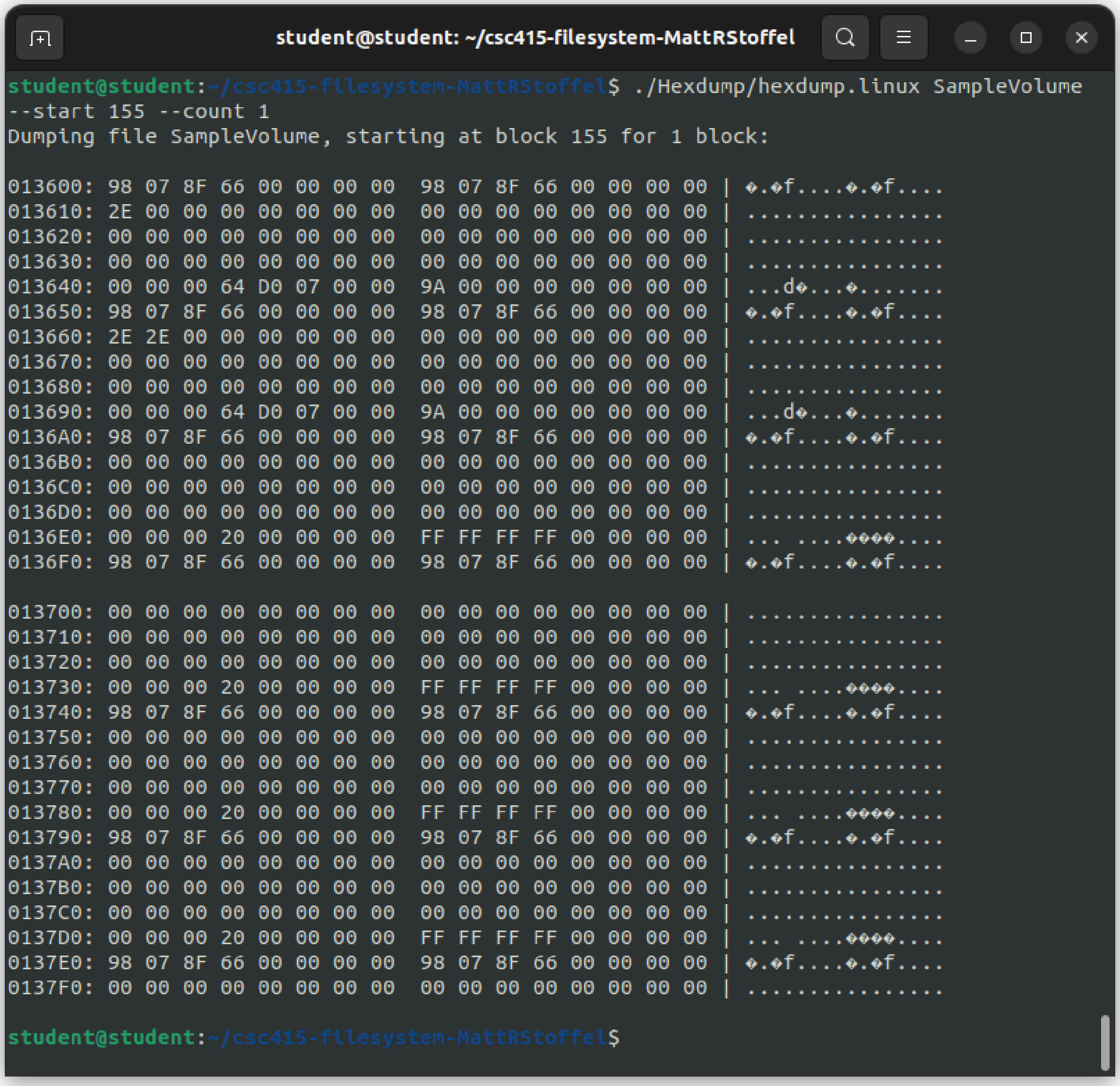
```
student@student:~/csc415-filesystem-MattRStoffel$ ./Hexdump/hexdump.linux SampleVolume
--start 1 --count 1
Dumping file SampleVolume, starting at block 1 for 1 block:

000200: 52 42 69 65 72 6D 61 6E  00 02 00 00 4B 4C 00 00 | RBierman....KL..
000210: 01 00 00 00 9E 00 00 00  AD 4B 00 00 9A 00 00 00 | ....�...�K..�...
000220: 04 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:~/csc415-filesystem-MattRStoffel$
```



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

000400: FD FF FF FF 02 00 00 00  03 00 00 00 04 00 00 00 | ����............
000410: 05 00 00 00 06 00 00 00  07 00 00 00 08 00 00 00 | ................
000420: 09 00 00 00 0A 00 00 00  0B 00 00 00 0C 00 00 00 | ................
000430: 0D 00 00 00 0E 00 00 00  0F 00 00 00 10 00 00 00 | ................
000440: 11 00 00 00 12 00 00 00  13 00 00 00 14 00 00 00 | ................
000450: 15 00 00 00 16 00 00 00  17 00 00 00 18 00 00 00 | ................
000460: 19 00 00 00 1A 00 00 00  1B 00 00 00 1C 00 00 00 | ................
000470: 1D 00 00 00 1E 00 00 00  1F 00 00 00 20 00 00 00 | ............ ...
000480: 21 00 00 00 22 00 00 00  23 00 00 00 24 00 00 00 | !..."...#...$...
000490: 25 00 00 00 26 00 00 00  27 00 00 00 28 00 00 00 | %...&...'...(...
0004A0: 29 00 00 00 2A 00 00 00  2B 00 00 00 2C 00 00 00 | )...*...+...,...
0004B0: 2D 00 00 00 2E 00 00 00  2F 00 00 00 30 00 00 00 | -......./...0...
0004C0: 31 00 00 00 32 00 00 00  33 00 00 00 34 00 00 00 | 1...2...3...4...
0004D0: 35 00 00 00 36 00 00 00  37 00 00 00 38 00 00 00 | 5...6...7...8...
0004E0: 39 00 00 00 3A 00 00 00  3B 00 00 00 3C 00 00 00 | 9...:...;...<...
0004F0: 3D 00 00 00 3E 00 00 00  3F 00 00 00 40 00 00 00 | =...>...?...@...

000500: 41 00 00 00 42 00 00 00  43 00 00 00 44 00 00 00 | A...B...C...D...
000510: 45 00 00 00 46 00 00 00  47 00 00 00 48 00 00 00 | E...F...G...H...
000520: 49 00 00 00 4A 00 00 00  4B 00 00 00 4C 00 00 00 | I...J...K...L...
000530: 4D 00 00 00 4E 00 00 00  4F 00 00 00 50 00 00 00 | M...N...O...P...
000540: 51 00 00 00 52 00 00 00  53 00 00 00 54 00 00 00 | Q...R...S...T...
000550: 55 00 00 00 56 00 00 00  57 00 00 00 58 00 00 00 | U...V...W...X...
000560: 59 00 00 00 5A 00 00 00  5B 00 00 00 5C 00 00 00 | Y...Z...[...\...
000570: 5D 00 00 00 5E 00 00 00  5F 00 00 00 60 00 00 00 | ]...^..._...`...
000580: 61 00 00 00 62 00 00 00  63 00 00 00 64 00 00 00 | a...b...c...d...
000590: 65 00 00 00 66 00 00 00  67 00 00 00 68 00 00 00 | e...f...g...h...
0005A0: 69 00 00 00 6A 00 00 00  6B 00 00 00 6C 00 00 00 | i...j...k...l...
0005B0: 6D 00 00 00 6E 00 00 00  6F 00 00 00 70 00 00 00 | m...n...o...p...
0005C0: 71 00 00 00 72 00 00 00  73 00 00 00 74 00 00 00 | q...r...s...t...
0005D0: 75 00 00 00 76 00 00 00  77 00 00 00 78 00 00 00 | u...v...w...x...
0005E0: 79 00 00 00 7A 00 00 00  7B 00 00 00 7C 00 00 00 | y...z...{...|...
0005F0: 7D 00 00 00 7E 00 00 00  7F 00 00 00 80 00 00 00 | }...~.......�...

student@student:~/csc415-filesystem-MattRStoffel$
```



```
student@student:~/csc415-filesystem-MattRStoffel$ ./Hexdump/hexdump.linux SampleVolume
--start 3 --count 1
Dumping file SampleVolume, starting at block 3 for 1 block:

000600: 81 00 00 00 82 00 00 00  83 00 00 00 84 00 00 00 | �...�...�...�...
000610: 85 00 00 00 86 00 00 00  87 00 00 00 88 00 00 00 | �...�...�...�...
000620: 89 00 00 00 8A 00 00 00  8B 00 00 00 8C 00 00 00 | �...�...�...�...
000630: 8D 00 00 00 8E 00 00 00  8F 00 00 00 90 00 00 00 | �...�...�...�...
000640: 91 00 00 00 92 00 00 00  93 00 00 00 94 00 00 00 | �...�...�...�...
000650: 95 00 00 00 96 00 00 00  97 00 00 00 98 00 00 00 | �...�...�...�...
000660: 99 00 00 00 FD FF FF FF  9B 00 00 00 9C 00 00 00 | �...�����...�...
000670: 9D 00 00 00 FD FF FF FF  9F 00 00 00 A0 00 00 00 | �...�����...�...
000680: A1 00 00 00 A2 00 00 00  A3 00 00 00 A4 00 00 00 | �...�...�...�...
000690: A5 00 00 00 A6 00 00 00  A7 00 00 00 A8 00 00 00 | �...�...�...�...
0006A0: A9 00 00 00 AA 00 00 00  AB 00 00 00 AC 00 00 00 | �...�...�...�...
0006B0: AD 00 00 00 AE 00 00 00  AF 00 00 00 B0 00 00 00 | �...�...�...�...
0006C0: B1 00 00 00 B2 00 00 00  B3 00 00 00 B4 00 00 00 | �...�...�...�...
0006D0: B5 00 00 00 B6 00 00 00  B7 00 00 00 B8 00 00 00 | �...�...�...�...
0006E0: B9 00 00 00 BA 00 00 00  BB 00 00 00 BC 00 00 00 | �...�...�...�...
0006F0: BD 00 00 00 BE 00 00 00  BF 00 00 00 C0 00 00 00 | �...�...�...�...

000700: C1 00 00 00 C2 00 00 00  C3 00 00 00 C4 00 00 00 | �...�...�...�...
000710: C5 00 00 00 C6 00 00 00  C7 00 00 00 C8 00 00 00 | �...�...�...�...
000720: C9 00 00 00 CA 00 00 00  CB 00 00 00 CC 00 00 00 | �...�...�...�...
000730: CD 00 00 00 CE 00 00 00  CF 00 00 00 D0 00 00 00 | �...�...�...�...
000740: D1 00 00 00 D2 00 00 00  D3 00 00 00 D4 00 00 00 | �...�...�...�...
000750: D5 00 00 00 D6 00 00 00  D7 00 00 00 D8 00 00 00 | �...�...�...�...
000760: D9 00 00 00 DA 00 00 00  DB 00 00 00 DC 00 00 00 | �...�...�...�...
000770: DD 00 00 00 DE 00 00 00  DF 00 00 00 E0 00 00 00 | �...�...�...�...
000780: E1 00 00 00 E2 00 00 00  E3 00 00 00 E4 00 00 00 | �...�...�...�...
000790: E5 00 00 00 E6 00 00 00  E7 00 00 00 E8 00 00 00 | �...�...�...�...
0007A0: E9 00 00 00 EA 00 00 00  EB 00 00 00 EC 00 00 00 | �...�...�...�...
0007B0: ED 00 00 00 EE 00 00 00  EF 00 00 00 F0 00 00 00 | �...�...�...�...
0007C0: F1 00 00 00 F2 00 00 00  F3 00 00 00 F4 00 00 00 | �...�...�...�...
0007D0: F5 00 00 00 F6 00 00 00  F7 00 00 00 F8 00 00 00 | �...�...�...�...
0007E0: F9 00 00 00 FA 00 00 00  FB 00 00 00 FC 00 00 00 | �...�...�...�...
0007F0: FD 00 00 00 FE 00 00 00  FF 00 00 00 00 01 00 00 | �...�...�.......

student@student:~/csc415-filesystem-MattRStoffel$
```



```
student@student:~/csc415-filesystem-MattRStoffel$ ./Hexdump/hexdump.linux SampleVolume
--start 155 --count 1
Dumping file SampleVolume, starting at block 155 for 1 block:

013600: 98 07 8F 66 00 00 00 00  98 07 8F 66 00 00 00 00 | �.�f....�.�f....
013610: 2E 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013620: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013630: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013640: 00 00 00 64 D0 07 00 00  9A 00 00 00 00 00 00 00 | ...d�...�.......
013650: 98 07 8F 66 00 00 00 00  98 07 8F 66 00 00 00 00 | �.�f....�.�f....
013660: 2E 2E 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013670: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013680: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013690: 00 00 00 64 D0 07 00 00  9A 00 00 00 00 00 00 00 | ...d�...�.......
0136A0: 98 07 8F 66 00 00 00 00  98 07 8F 66 00 00 00 00 | �.�f....�.�f....
0136B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0136C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0136D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0136E0: 00 00 00 20 00 00 00 00  FF FF FF FF 00 00 00 00 | ... ....����....
0136F0: 98 07 8F 66 00 00 00 00  98 07 8F 66 00 00 00 00 | �.�f....�.�f....

013700: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013710: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013720: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013730: 00 00 00 20 00 00 00 00  FF FF FF FF 00 00 00 00 | ... ....����....
013740: 98 07 8F 66 00 00 00 00  98 07 8F 66 00 00 00 00 | �.�f....�.�f....
013750: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013760: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013770: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013780: 00 00 00 20 00 00 00 00  FF FF FF FF 00 00 00 00 | ... ....����....
013790: 98 07 8F 66 00 00 00 00  98 07 8F 66 00 00 00 00 | �.�f....�.�f....
0137A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0137B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0137C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0137D0: 00 00 00 20 00 00 00 00  FF FF FF FF 00 00 00 00 | ... ....����....
0137E0: 98 07 8F 66 00 00 00 00  98 07 8F 66 00 00 00 00 | �.�f....�.�f....
0137F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

student@student:~/csc415-filesystem-MattRStoffel$
```