# File System Milestone 1

Team: Team Buffer

Micheal Hua: 918729206

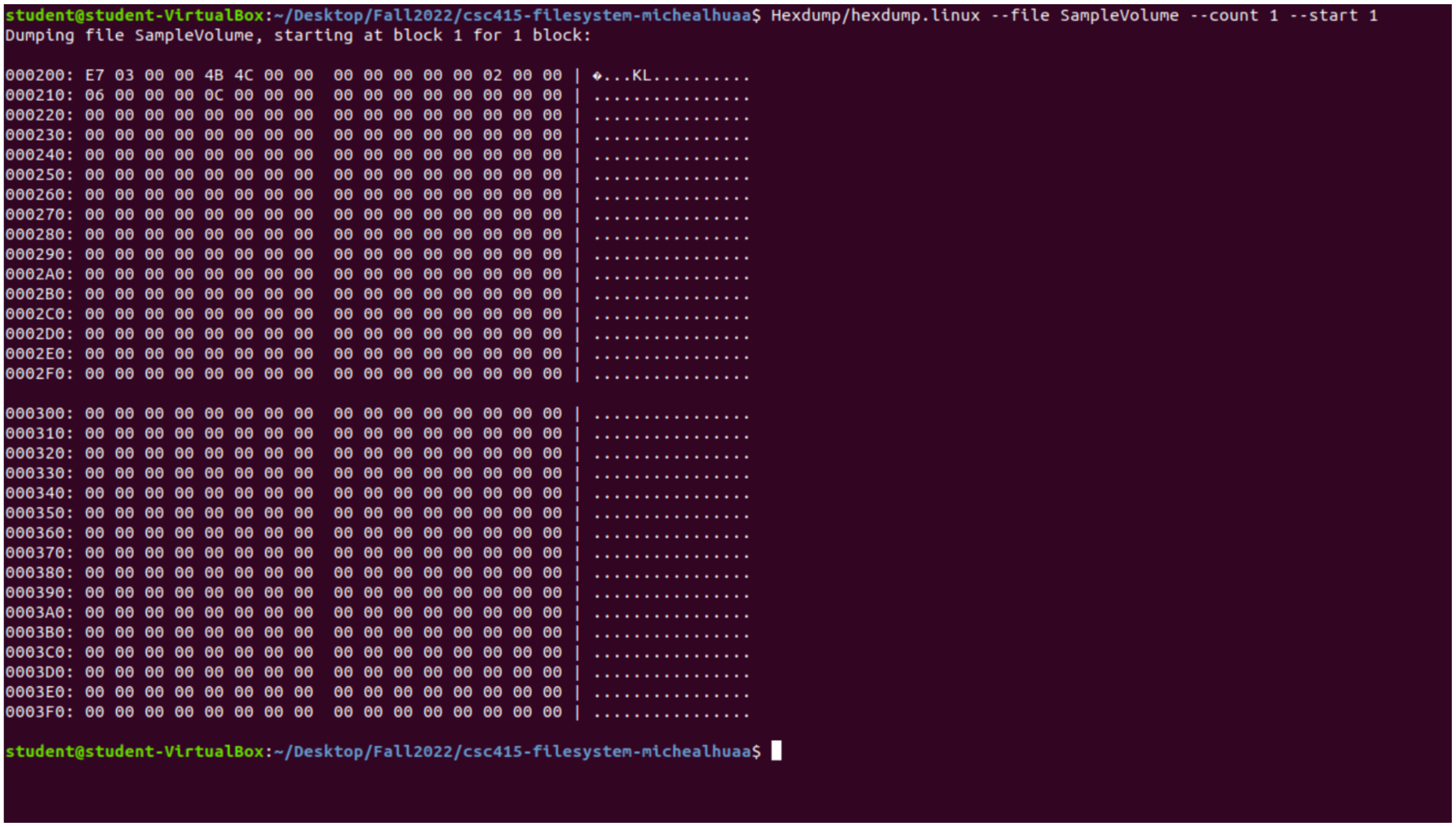
Sajan Gurung: 921149577

Deep Dhorajiya: 920842114

Ron Elijah Rivera:

GitHub Repository Name: michealhuaa

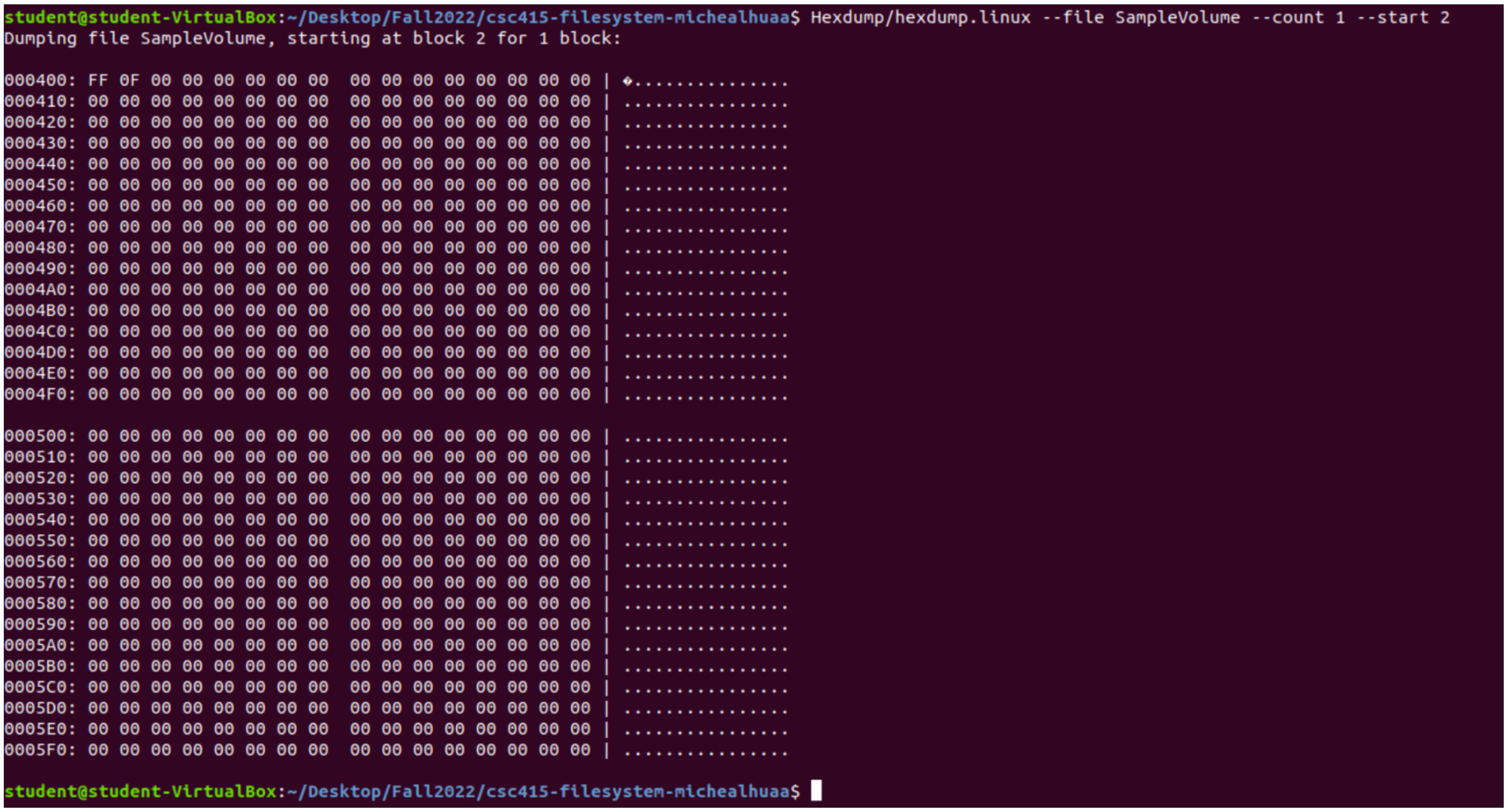
## Volume Control Block Dump:



```
student@student-VirtualBox:~/Desktop/Fall2022/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 00 00 00 00 02 00 00 | �...KL..........
000210: 06 00 00 00 0C 00 00 00  00 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 | ................

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/Fall2022/csc415-filesystem-michealhuaa$ 
```

The volume control block is visible here. The structure's first variable is free space. The index inside the bitmap where free space currently begins is 12, which translates to 0C. The number of blocks is the next data point, which is 19531. We can also see the block size, the starting point of the root directory, and the magic number with a total of five integers, or 20 bytes.

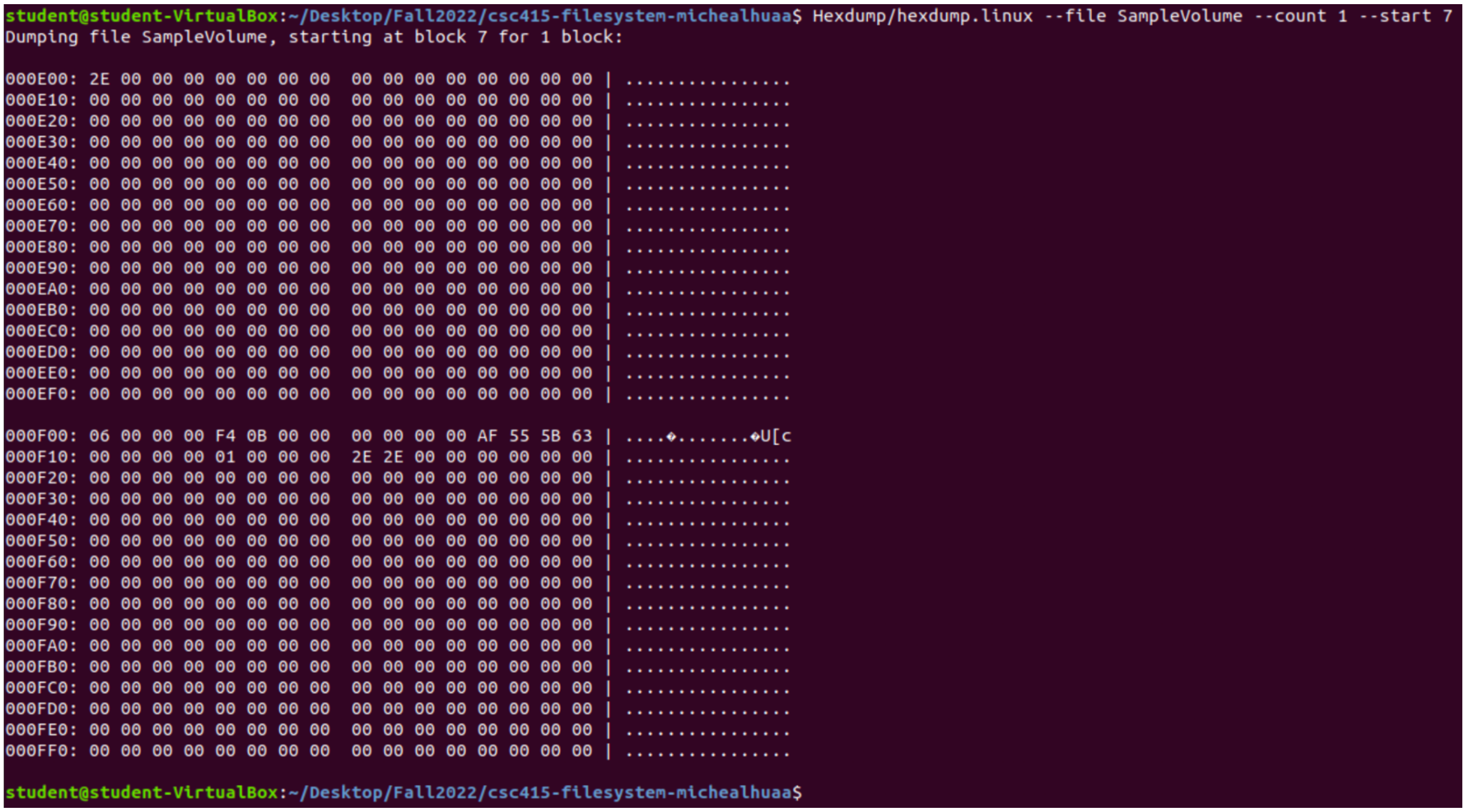
## BitMap Dump



```
student@student-VirtualBox:~/Desktop/Fall2022/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/Fall2022/csc415-filesystem-michealhuaa$ 
```

Two bytes are being utilized here. There are 12 blocks filled as of now within the free space bitmap. One byte contains eight bits. The first byte is loaded, and the value is "FF," which in binary is 11111111. Four bits, or 1111, have been added to the second byte.

## Root Directory Dump



```
student@student-VirtualBox:~/Desktop/Fall2022/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 AF 55 5B 63 | ....�.......�U[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/Fall2022/csc415-filesystem-michealhuaa$
```

Two nearly identical directory entries are set up here. The name is the only difference. 2E(".") or “current directory” is the name of Index[0], and the name of Index[1] is 2E2E(“..”) or “parent directory”.

## Volume Control Block Structure

Ans. The start of the free space is initialized by the VCB (Volume Control Block). We can store the necessary information and reduce the amount of wasted space within the blocks by setting the number of blocks and their sizes. Using the magic number, we can determine whether this is our Volume Control Block. 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.

**Structure of our VCB**

```
typedef struct VolumeControlBlock{

        int magicNum; // a magic number to identify the disk

        // disk layout

        int numOfBlocks; // total blocks in our disk

        int numOfFreeBlocks; // free blocks

        int sizeOfBlock;         // 512 bytes

        int firstBlock;  // initial block in the disk

        int rootDirectory;       // reference to the root

        int freeSpacePointer; //free space pointer using bitmap
```

} VCB;

## Free Space Structure

Eight bits can be stored within a single character because a character is one byte. By storing a 0 or 1 on each bit, you could boost your efficiency by eight times. Therefore, we utilized a bitmap array of chars for our free space. The contents of the VCB are used in a bitmap to initialize free space. Using the values 0 and 1, we use the bitmap to determine whether a block has space. Each block corresponds to a distinct bit position in the array. By dividing the number by 8, we can determine the index because a bitmap can store eight times more 0s and s than a standard char value. The position within the index is then obtained by employing the modulus operator (number%8). Because of this, every number can be in its place within an integer. The "1" bit is then moved to the right place through the char using the operator. The value of a particular position is checked using a similar method.

## Directory System Structure,

All of the metadata and the core data (file logical address start point and file size) are contained in our directory entry. The 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 file location of 0, indicating that they are not used. Two entries in the root directory have been fully initialized: index [0] and index[1], also known as the current and parent directory. Due to the fact that the root is at the top of the tree, and ".." indicates a parent. The root is the parent itself.

**Structure of our Directory Entry**

```
typedef struct ValidCStruct{
char directoryName[255]; // name of the directory
int location; // location of the directory
int size; // size of the directory
time_t lastModifiedDate; // time last modified
time_t createdDate; // when the directory was created
} directory_struct;
```

## Teammates' Contribution Table

Red = No Activity,  Yellow = Some Activity,  Green = Very Active

| | VCB Structure | Free Space | Root Directory | Write-up |
|---|---|---|---|---|
| Micheal | 🟩 | 🟩 | 🟨 | 🟩 |
| Sajan | 🟨 | 🟩 | 🟩 | 🟩 |
| Deep | 🟩 | 🟨 | 🟩 | 🟩 |
| Ron | 🟥 | 🟩 | 🟩 | 🟨 |

## Team Comradery

We all worked together and divided the task based on the situation of the file system project. We started recapping the volume control block, discussed the structure, and how to implement free space and directory entry.  We decided to meet twice a week or sometimes three times in two weeks through a discord lounge meeting. We started by explaining the situation to group members and assigned the task based on the situation to achieve the result. We decided on the work as follows: Micheal was set to initialize the volume control block. Sajan was assigned to take care of the root directory we built. Ron Elijah was assigned to maintain the free space structure while Deep handled the VCB structure and directory system. Team Buffer worked through coordinating with each other at every step and took help from each other wherever we got stuck.

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

After carefully demonstrating the file system project, we realized that each section is linked with each other, and we need to understand the overflow of the project. At the initial stage, the team was suffering with the VCB structure, and then we followed the class materials and the resources to understand what the VCB structure looks like. We were able to create the structure and carry on further in the project. We made the linked paths of the file system and started coding the project. We got several errors while debugging the code, but after meeting in the discord and trying several syntaxes, we solved the issues and got the result. Coordination among the team members helped us to achieve the assignment.

## Problems We Faced

One of the issues we faced was understanding the bitmap structure. 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. We were able to accomplish this through research. Each character has 8 bits. We want to store eight times as much data in this array as we usually 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. We will always receive a whole number rather than a double from the index because it is an integer. All of these numbers will be stored in index 0, as 0 through 7 will all be equal to 0 when divided by 8. These eight distinct values must now be stored within the same character. We use the modulus operator, which divides to get the remainder to get the position. 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 also needed help understanding how to initialize each root directory to a known amount of 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 realized we were overthinking the problem. 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 issue.

Lastly, one of our members encountered a file permissions error indicating that they could not access our hex dump. The problem was when we called to talk about our progress, we quickly fixed it by using the chmod command in our terminal. The issue was resolved when used, and we were able to move on quickly.