Milestone One

Group: Segmentation Fault
Members:

Member	Student ID
Christian Francisco	920603057
David Ye Luo	917051959
Marc Castro	921720147
Rafael Sunico	920261261

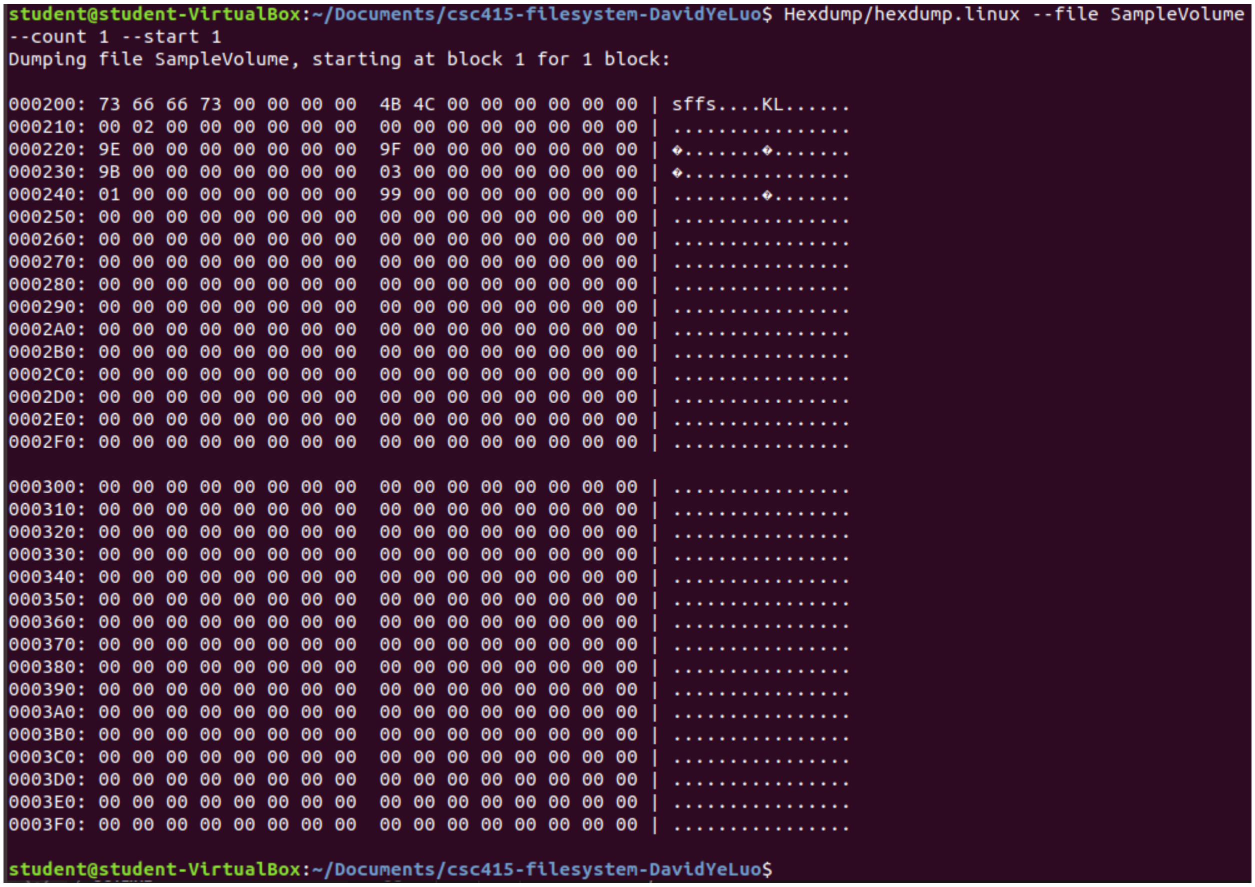
HexDump of Volume Control Block, FreeSpace, and Root Directory

HexDump Note

For strings, they are stored left to right while numbers are stored from right to left.
For example:

Value	Hexadecimal	Hexdump
19531	00 00 00 00 00 00 4C 4B	4B 4C 00 00 00 00 00 00

Volume Control Block

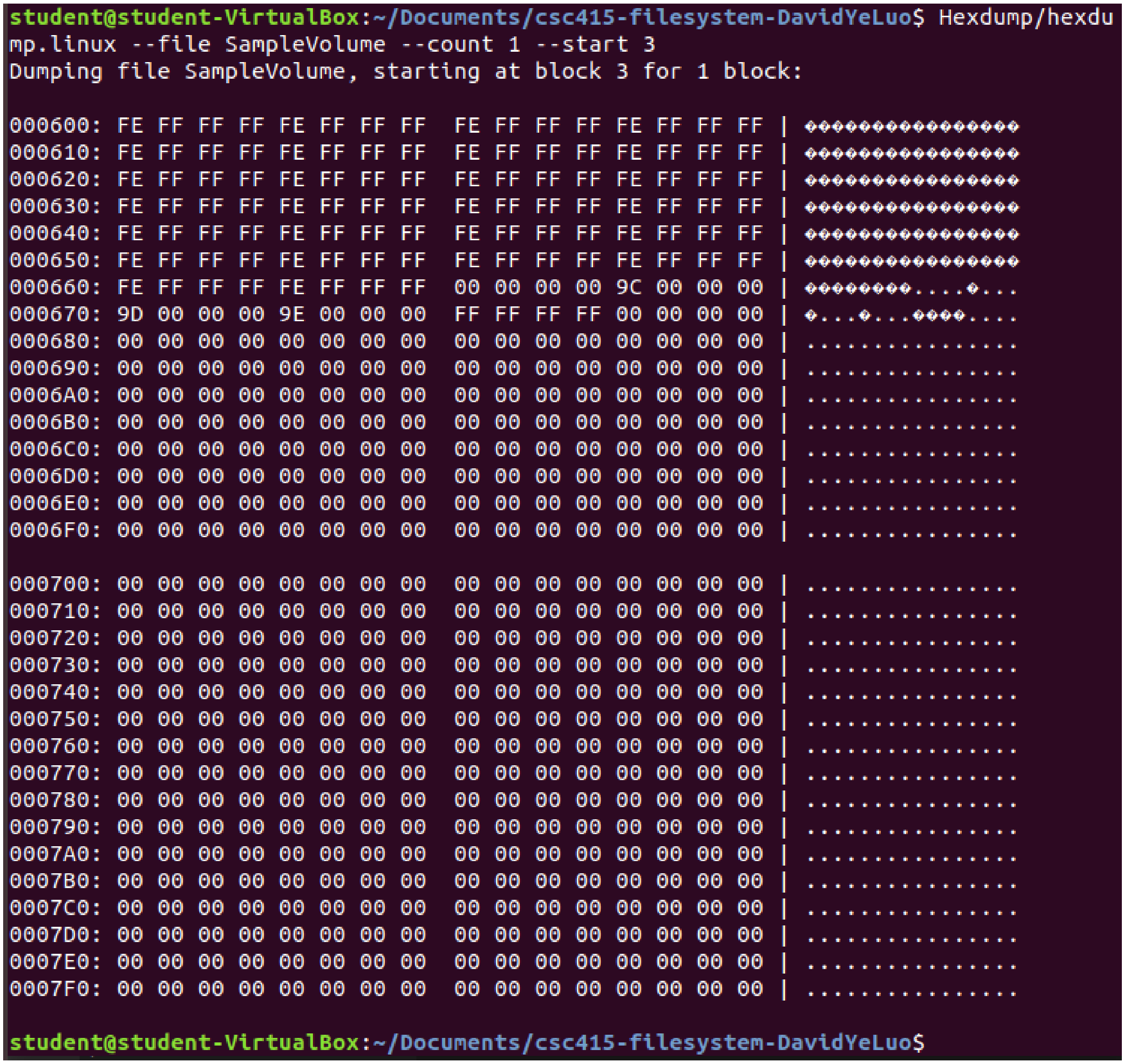


```
student@student-VirtualBox:~/Documents/csc415-filesystem-DavidYeLuo$ Hexdump/hexdump.linux --file SampleVolume
--count 1 --start 1
Dumping file SampleVolume, starting at block 1 for 1 block:

000200: 73 66 66 73 00 00 00 00  4B 4C 00 00 00 00 00 00 | sffs....KL......
000210: 00 02 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000220: 9E 00 00 00 00 00 00 00  9F 00 00 00 00 00 00 00 | �.......�.......
000230: 9B 00 00 00 00 00 00 00  03 00 00 00 00 00 00 00 | �...............
000240: 01 00 00 00 00 00 00 00  99 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:~/Documents/csc415-filesystem-DavidYeLuo$
```

This is the screenshot of the hexdump for the volume control block. This table represents the vcb and stored hexadecimal.

Size (Bytes)	Represents	Value	Hexadecimal
4	signature	“sffs”	73 66 66 73
8	Block Count	19,531	00 00 00 00 00 00 4C 4B
8	Block Size	512	00 00 00 00 00 00 02 00
8	VCB Location	0	00 00 00 00 00 00 00 00
8	Start Location of Free Space	158	00 00 00 00 00 00 00 9E
8	Next Location of Free Space	159	00 00 00 00 00 00 00 9F
8	Root Location	155	00 00 00 00 00 00 00 9B
8	Root Size	3	00 00 00 00 00 00 00 03
8	Fat Location	1	00 00 00 00 00 00 00 01
8	Fat Size	153	00 00 00 00 00 00 00 99

FreeSpace

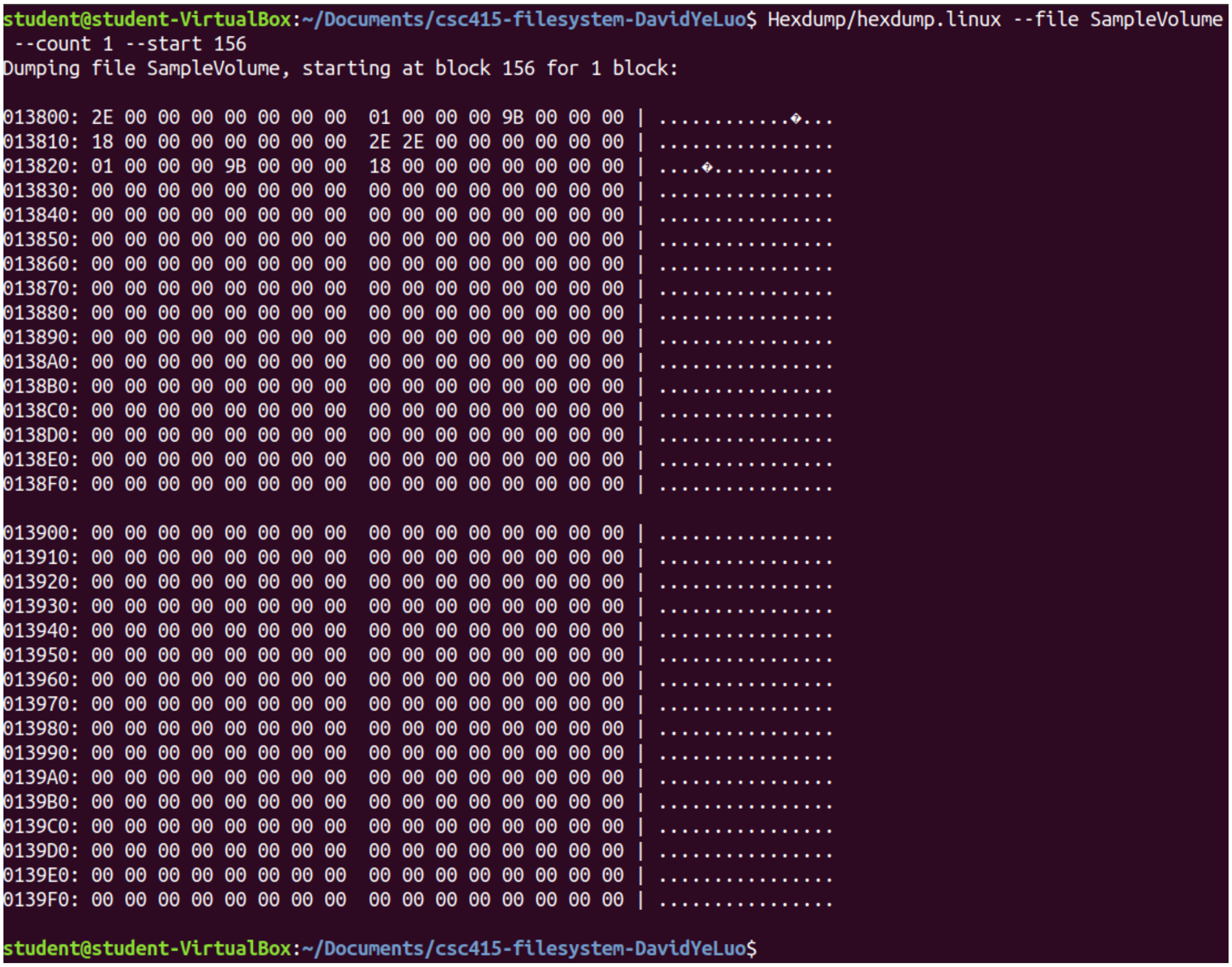


```
student@student-VirtualBox:~/Documents/csc415-filesystem-DavidYeLuo$ Hexdump/hexdu
mp.linux --file SampleVolume --count 1 --start 3
Dumping file SampleVolume, starting at block 3 for 1 block:

000600: FE FF FF FF FE FF FF FF  FE FF FF FF FE FF FF FF | ����������������
000610: FE FF FF FF FE FF FF FF  FE FF FF FF FE FF FF FF | ����������������
000620: FE FF FF FF FE FF FF FF  FE FF FF FF FE FF FF FF | ����������������
000630: FE FF FF FF FE FF FF FF  FE FF FF FF FE FF FF FF | ����������������
000640: FE FF FF FF FE FF FF FF  FE FF FF FF FE FF FF FF | ����������������
000650: FE FF FF FF FE FF FF FF  FE FF FF FF FE FF FF FF | ����������������
000660: FE FF FF FF FE FF FF FF  00 00 00 00 9C 00 00 00 | ��������....�...
000670: 9D 00 00 00 9E 00 00 00  FF FF FF FF 00 00 00 00 | �...�...����....
000680: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000690: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000700: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000710: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000720: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000730: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000740: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000750: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000760: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000770: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000780: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000790: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

student@student-VirtualBox:~/Documents/csc415-filesystem-DavidYeLuo$
```

Size (Bytes)	Line in hexdump	Represents	Value	Hexadecimal
8	000660, 9th byte to 16th	Reserved block	0xFFFFFFFE	FF FF FF FE
8	000660, 25th byte to 32nd	Allocated block, pointing to block 9C (156)	0x0000009C	00 00 00 9C
8	000670,	Allocated block,	0xFFFFFFFF	FF FF FF FF

	17th byte to 24th	end of file		
8	000670, 25th byte to 32nd	Free space	0x00000000	00 00 00 00

Root Directory



```
student@student-VirtualBox:~/Documents/csc415-filesystem-DavidYeLuo$ Hexdump/hexdump.linux --file SampleVolume
 --count 1 --start 156
Dumping file SampleVolume, starting at block 156 for 1 block:

013800: 2E 00 00 00 00 00 00 00  01 00 00 00 9B 00 00 00 | ............�...
013810: 18 00 00 00 00 00 00 00  2E 2E 00 00 00 00 00 00 | ................
013820: 01 00 00 00 9B 00 00 00  18 00 00 00 00 00 00 00 | ....�...........
013830: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013840: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013850: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013860: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013870: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013880: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013890: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0138A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0138B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0138C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0138D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0138E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0138F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

013900: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013910: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013920: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013930: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013940: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013950: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013960: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013970: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013980: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
013990: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0139A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0139B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0139C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0139D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0139E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0139F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

student@student-VirtualBox:~/Documents/csc415-filesystem-DavidYeLuo$
```

Size (Bytes)	**Represents**	**Value**	**Hexadecimal**
8	name	“.”	2E 00 00 00 00 00 00 00
1	isDir	1	01
4	location	155	00 00 00 9B

8	size	24	00 00 00 00 00 00 00 18

8	name	“..”	2E 2E 00 00 00 00 00 00
1	isDir	1	01
4	location	155	00 00 00 9B
8	size	24	00 00 00 00 00 00 00 18

Description of Volume Control Block Structure

VCB Fields

char disk_sig[4]	Signature of Segmentation Fault File System (sffs). Used to check if the volume needs to be formatted.
uint64_t block_count	Number of blocks in volume.
uint64_t block_size	Number of bytes per block.
uint64_t vcb_location	Where VCB is in volume. Should be 0.
uint64_t fs_start_location	Where free space starts on the volume. Should be the next block after VCB and FAT.
uint64_t fs_next_location	Where the next free block is volume.
uint64_t root_location	Where the root directory starts in volume.
uint64_t root_size	Size of root directory in volume in blocks.
uint64_t fat_location	Where FAT is in volume.
uint64_t fat_size	Size of FAT in volume in blocks.

Description of Free Space Management

Design

File Allocation Table

Stored data is represented in this table which is a singly linked list. Each element in this table represents a block indicating if they are free space or not. In addition, it contains the connection of the next data block(doesn't have to be continuous).

Linked List

Nodes

The data of the node contains the location of the next location of

Size

By default, the table should need 153 blocks starting from sector 1. Where each block is represented as a table entry of 4 bytes (32 bits).

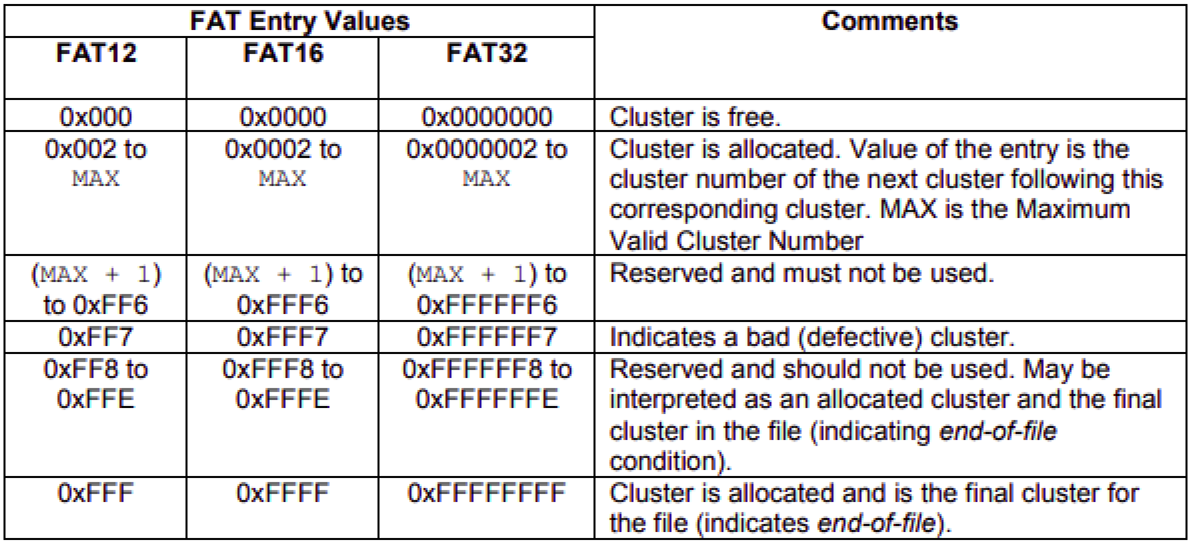
Math (This is an example, we didn't hard code this)

Block size: 512 Bytes
Total Blocks (Default size): 19,531 Blocks
Entry size: 4 Bytes (32 bits)

SizeOfTable = Total Blocks * Entry size = 78,124 Bytes
SizeOfTable = 78,124 Bytes / 512 Bytes = 153 Blocks

Entry Values

Source 1
Source 2



FAT Entry Values			Comments
FAT12	FAT16	FAT32	
0x000	0x0000	0x0000000	Cluster is free.
0x002 to MAX	0x0002 to MAX	0x0000002 to MAX	Cluster is allocated. Value of the entry is the cluster number of the next cluster following this corresponding cluster. MAX is the Maximum Valid Cluster Number
(MAX + 1) to 0xFF6	(MAX + 1) to 0xFFF6	(MAX + 1) to 0xFFFFFF6	Reserved and must not be used.
0xFF7	0xFFF7	0xFFFFFF7	Indicates a bad (defective) cluster.
0xFF8 to 0xFFE	0xFFF8 to 0xFFFE	0xFFFFFF8 to 0xFFFFFFE	Reserved and should not be used. May be interpreted as an allocated cluster and the final cluster in the file (indicating *end-of-file* condition).
0xFFF	0xFFFF	0xFFFFFFFF	Cluster is allocated and is the final cluster for the file (indicates *end-of-file*).

Implementation

File Allocation Table is implemented within fsInit.c and is defined as uint32_t *fat_table. Our group allocates space in fsInit.c via the function initFileSystem. We run an LBA read function to see if the signature matches, if it does then the vcb is already formatted and we have to reload the fat_array from memory. Within the said reloadFreeSpace function, we run another LBA read to check if it is the correct size and if it exists, if it does then we return 0 and move back to the initFileSystem function. If the VCB is not already formatted and the signatures do not match, then we format the VCB and run initFreeSpace to initialize the FAT by allocating memory, setting the blocks 1 to 1 after the FAT's size to reserved, and setting blocks after those to free, and setting the vcb's next location to the first free block. we then run an LBAwrite to commit the FAT array to the volume and return the location from the initFreeSpace. After initializing the root, we run allocateFreeSpace to iterate over the FAT array, starting at the file system's next location, and set that pointer's next location to the following block. Once we iterate to the end of that FAT array, we set the final block in the file to 0xFFFFFFFF and run LBAwrite to write the FAT array to the volume.

Table of who worked on which component

Member	**Component**
Christian Francisco	File Allocation Table planning, implementing directory entry, and creating directory function.
David Ye Luo	File Allocation Table planning, implementing directory entry, and creating directory function.
Marc Castro	Reloading/initializing FAT linked list from/to volume, allocating and initializing free space, implementation write up of FAT table.
Rafael Sunico	Defining and Initialising VCB, Defining functions for free space and FAT, organising and coordinating individual tasks for the group

Team Management

Our team worked together by researching and planning out how we will implement the VCB, Free Space and FAT, and root directory. We tackled each problem as a group, one at a time, to make sure everyone knows how each component works before actually programming it.

Then we split up the individual programming tasks into small tasks among the four of us based on who wanted to work on what, and we each implemented our components, making sure to commit and push whenever we have a function implemented so that our teammates can use them for their components.

We met five times between April 1st and April 5th. We met mostly on discord calls, and had in person meetings on April 5th in study rooms. We also met in a call the night of the deadline to write this document and to make sure everything was working correctly before submitting.

Issues and Resolutions

Location was a barrier for meetups and group meetings when we were not on campus. Solution was to meet via Discord call.

At the start of the project, we were all confused on how to implement the FAT. At first we were just going to use a bitmap after reading the “Steps for milestone 1” document, but after going over our class notes again we figured out that we needed to implement a linked list or something similar to it. After meeting in person and looking at the microsoft FAT32 specs again, we decided on how to implement the FAT by having an array of uint32_t, which holds either; 0 which means free; a block number which means allocated and points to the next block that holds the file that's also in this block; or 0xFFFFFFFF which means reserved.