Team NorthFace

Team Members

Abel Seyoum, Student ID: 921747304

Brian Nguyen, Student ID: 915503815

Jonathan Sum, Student ID: 917856087

Ysidro Alfaro De Leon, Student ID: 916365403

Github:

https://github.com/CSC415-2023-Spring/csc415-filesystem-Eskinaz

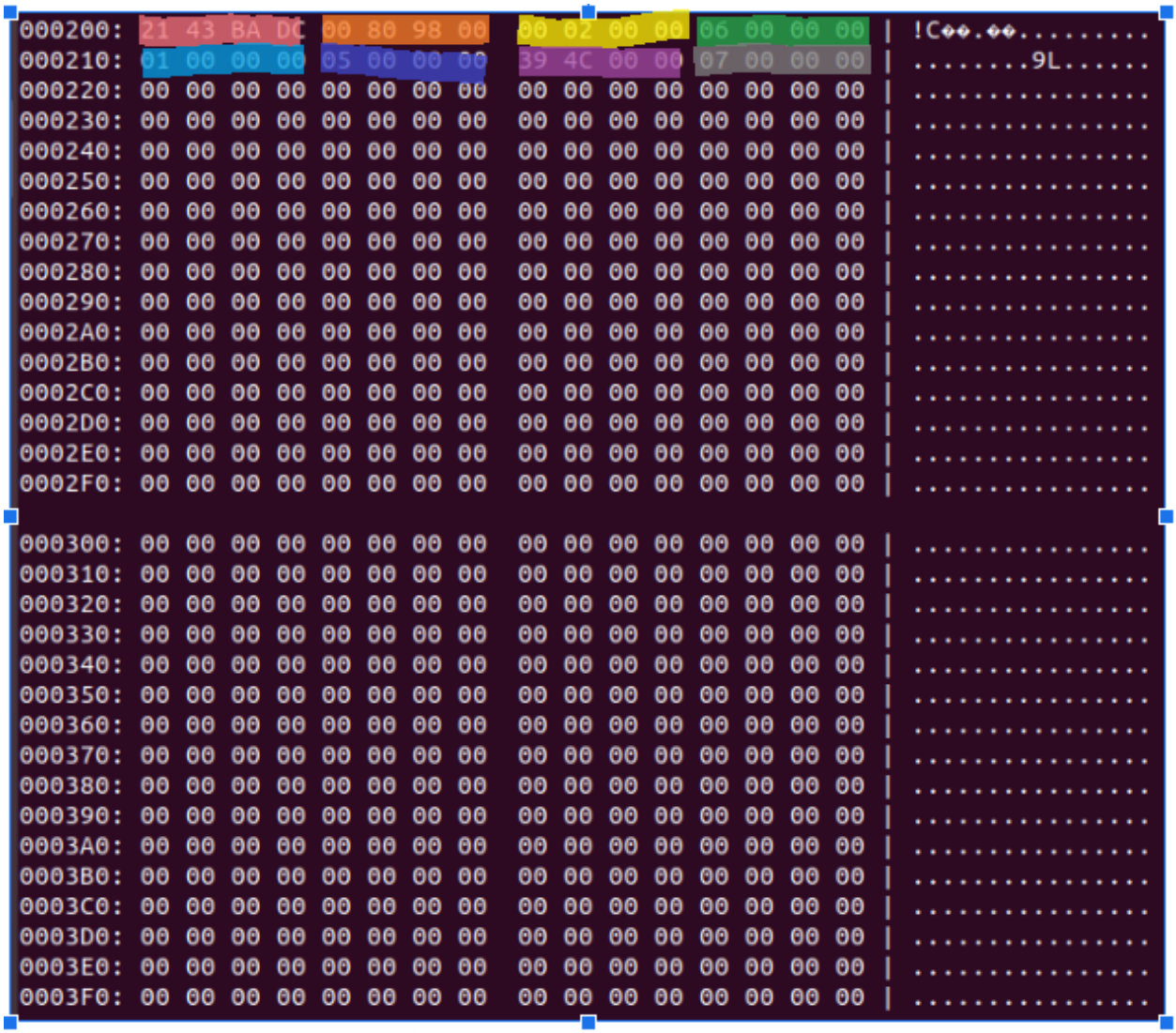
1. **A dump (use the provided HexDump utility) of the volume file that shows the VCB, FreeSpace, and complete root directory.**

Partition Header

```
000000: 43 53 43 2D 34 31 35 20  2D 20 4F 70 65 72 61 74 | CSC-415 - Operat
000010: 69 6E 67 20 53 79 73 74  65 6D 73 20 46 69 6C 65 | ing Systems File
000020: 20 53 79 73 74 65 6D 20  50 61 72 74 69 74 69 6F |  System Partitio
000030: 6E 20 48 65 61 64 65 72  0A 0A 00 00 00 00 00 00 | n Header........
000040: 42 20 74 72 65 62 6F 52  00 80 98 00 00 00 00 00 | B treboR.◆◆.....
000050: 00 02 00 00 00 00 00 00  40 4C 00 00 00 00 00 00 | ........@L......
000060: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000070: 52 6F 62 65 72 74 20 42  55 6E 74 69 74 6C 65 64 | Robert BUntitled
000080: 0A 0A 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000090: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0000A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0000B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0000C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0000D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0000E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0000F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000100: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000110: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000120: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000130: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000140: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000150: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000160: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000170: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000180: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000190: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0001A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0001B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0001C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0001D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0001E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0001F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

Block #0



Contains our VCB.

READING IN LITTLE-ENDIAN

Bytes 000200 - 000203

These four bytes are the value 0xDCBA4321 stored in little-endian, which is our magic number.

Bytes 000204 - 000207

The next four bytes, “00 80 98 00” = 0x988000 = 9994240 are our volume size.

Bytes 000208 - 00020B

The next four bytes “00 02 00 00” is hexadecimal for our blockSize of 512

Bytes 00020C - 00020F

The next four bytes “06 00 00 00” is hexadecimal for our rootLBA position of 6

Bytes 000210 - 000213

The next four bytes “01 00 00 00” is hexadecimal for our mapLBA position of 1

Bytes 000214 - 000217

The next four bytes “05 00 00 00” is hexadecimal for our mapCount with a value of 5

Bytes 000218 - 00021B

The next four bytes “39 4C 00 00” is hexadecimal for our freeBlocks with a value of 19513

This can be confirmed by dividing our volumeSize by the blockSize 9994240/512 - 19513

Bytes 00021C - 00021F

The next four bytes “07 00 00 00” is hexadecimal for our usedBlocks with a value of 7

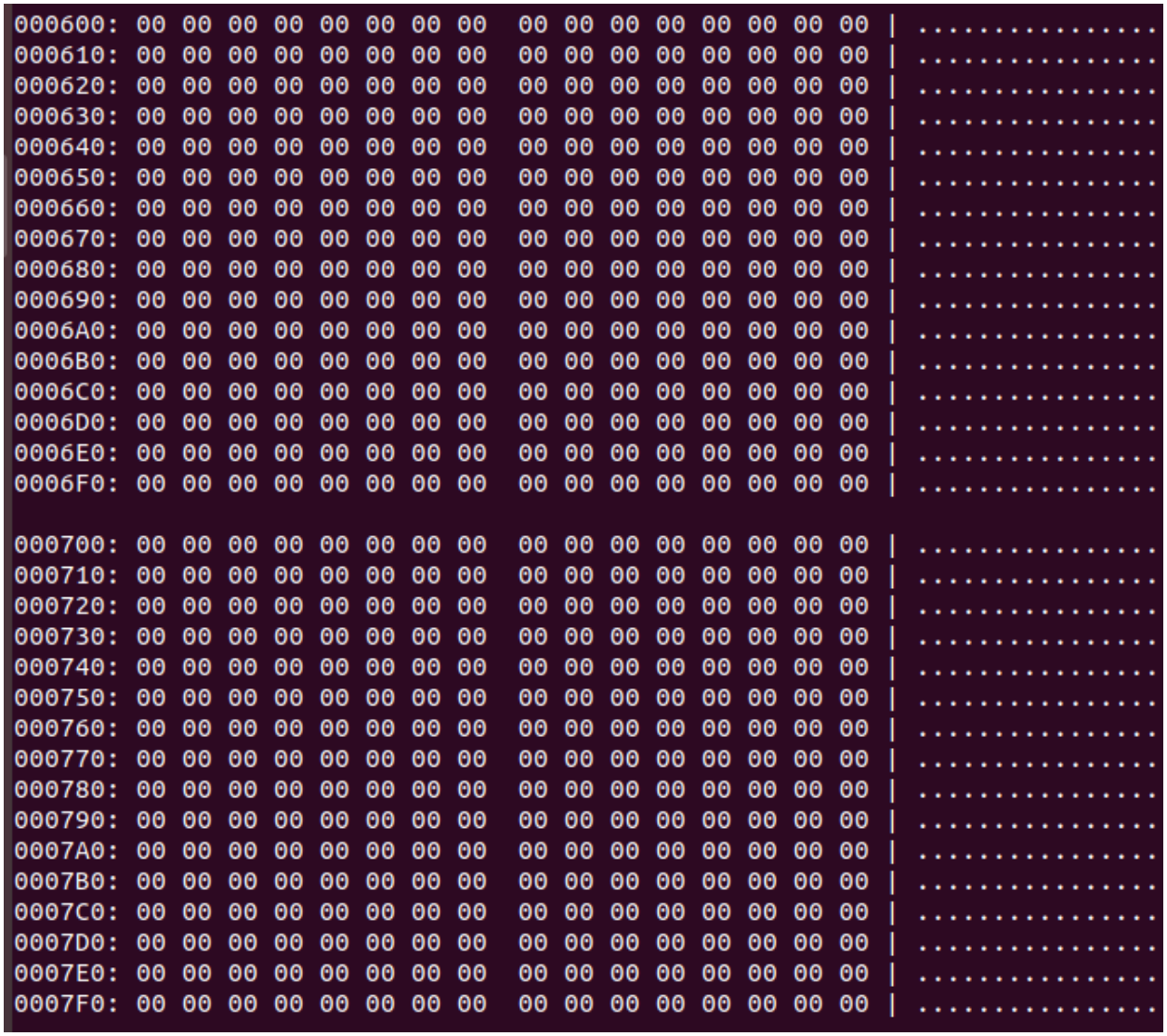
Block #1

```
000400: 7F 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: 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 | ................
```

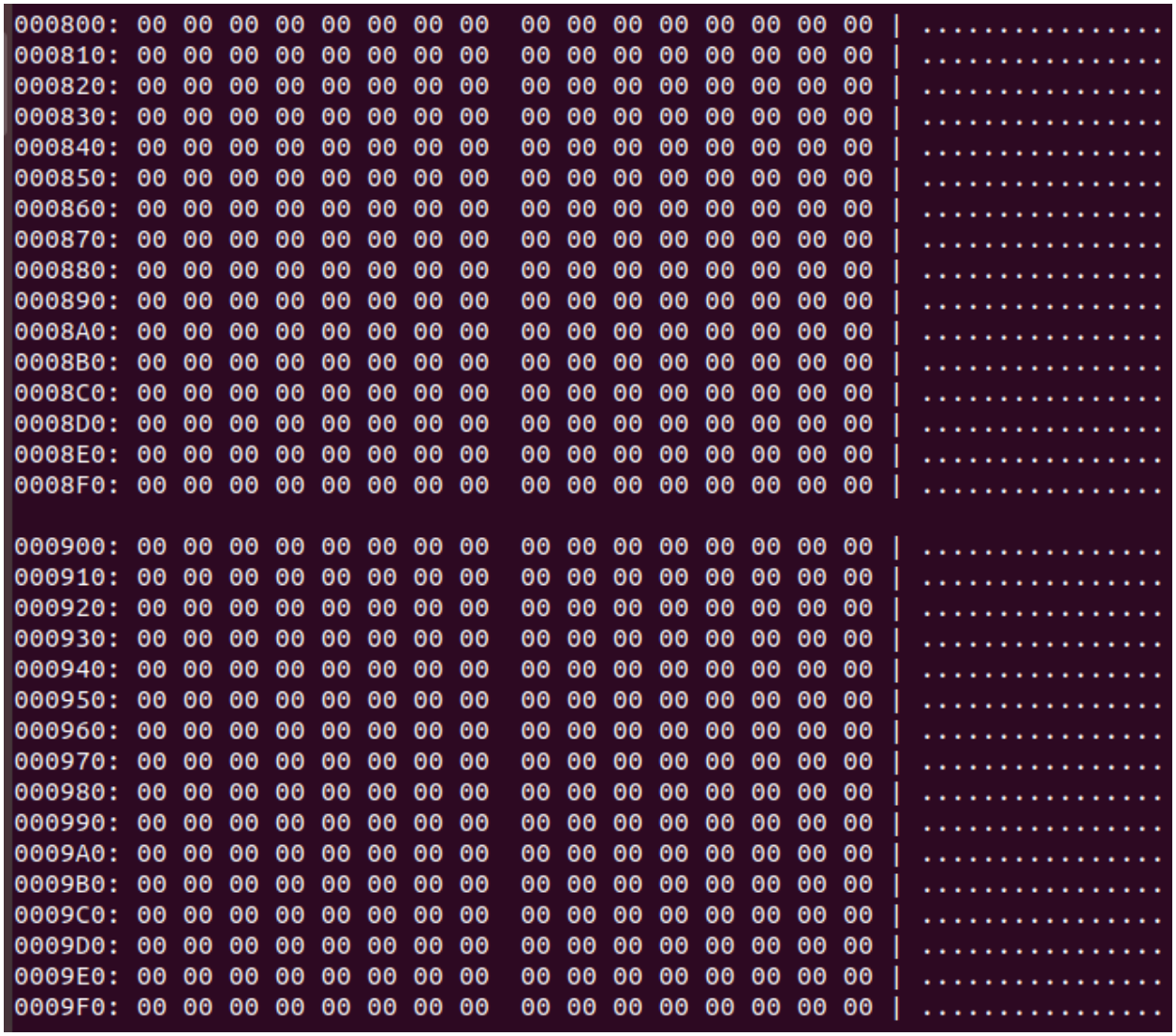
The start of our free map. The first byte is “7F” because that is the value that corresponds to the first integer’s first 7 bits and is set to one in order to mark those blocks as used.

Block #2



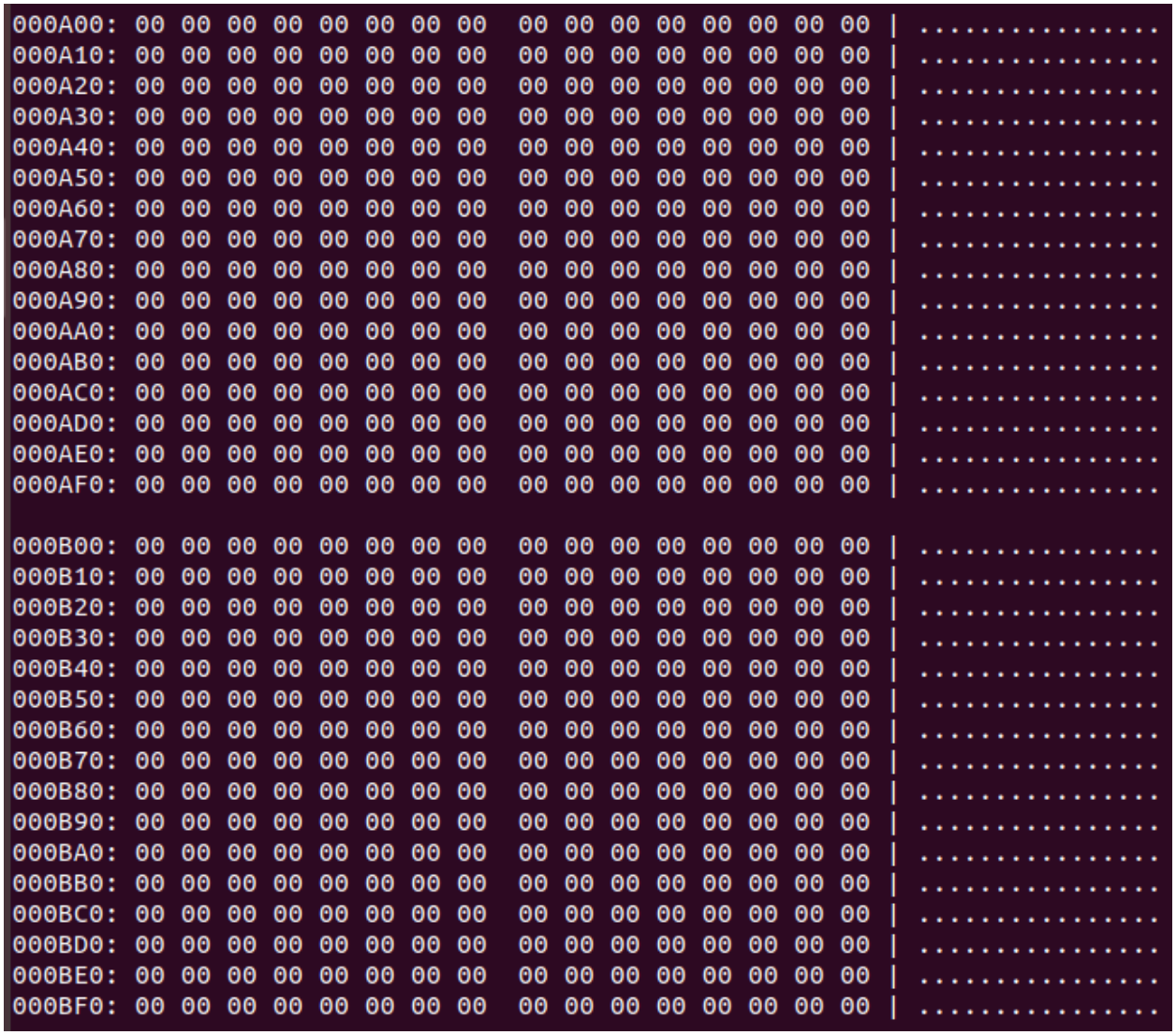
The second block of our free space map storage.

Block #3



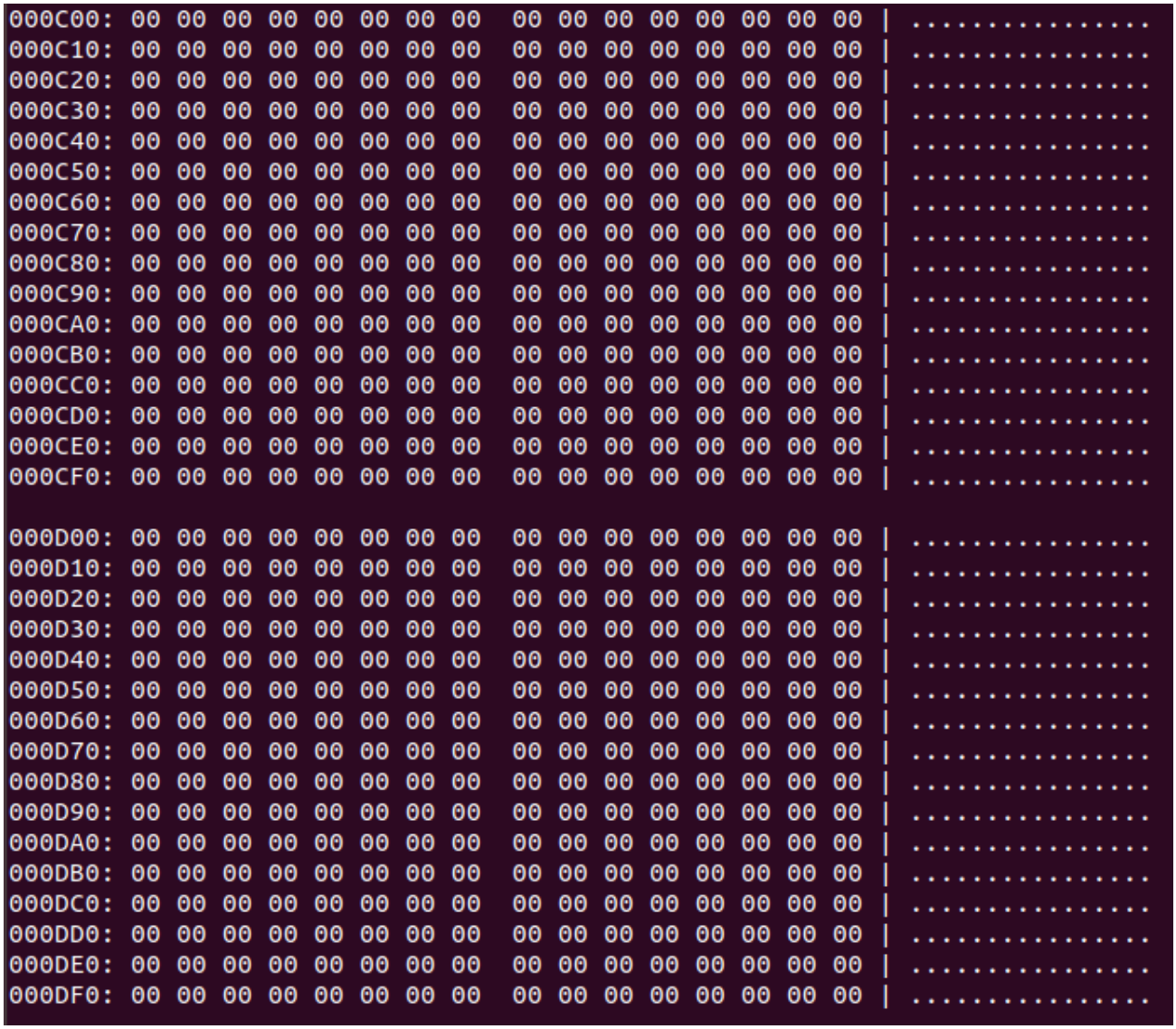
The third block of our free space map storage.

Block #4



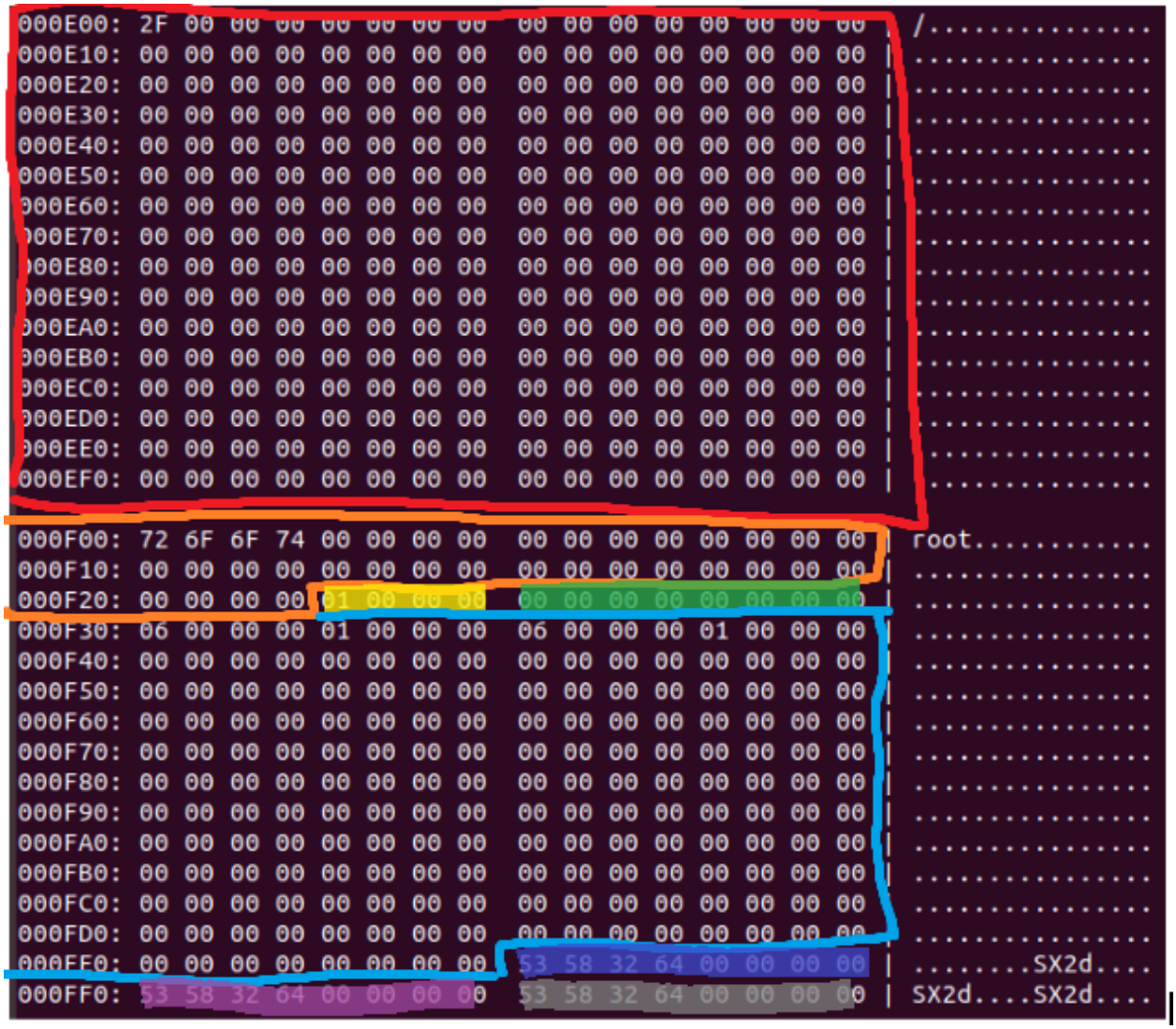
The fourth block of our free space map storage.

Block #5



The fifth block of our free space map storage.

Block #6



Our root directory. Still reading in little-endian

Bytes 000E00- 000EFF

The name of the root directory entry is “/” denoted by 2F

Bytes 000F00 - 000F23

The author of the root directory entry which is “root” denoted by 72 6F 6F 74

Bytes 000F24 - 000F27

The type of the directory entry is “0x1” to signify that root is a directory.

Bytes 000F28 - 000F2F

The size of the root directory entry is zero because it is empty.

Bytes 000F30- 000F37

The first extent of the root's extent table is used to point to itself for the "." directory

Bytes 000F38- 000F3F

The second extent of the root's extent table is used to point to itself for the ".." directory

Bytes 000F40- 000FE7

The 21 remaining extents of the entry's extent table

Bytes 000FE8- 000FEF

The UNIX epoch time when the root was created

Bytes 000FF0- 000FE7

The UNIX epoch time when the root was modified

Bytes 000FF8- 000FFF

The UNIX epoch time when the root was accessed

All 3 sets of bytes above from Bytes 000FE8 to 000FFF are created, modified, and accessed in the form of time_t which has a size of 8 bytes.

Converting from Hex to Decimal, the hex of 00 00 00 00 64 32 58 53 translates to 1681021011.

Further converting this to UNIX Epoch time, below shows the time since which we created the root directory, which is also the last date we modified and accessed it as well.

2. A description of the VCB structure

The newVolume() function creates a Volume Control Block (VCB) structure that contains the properties of the file system.

- **magicNumber:** a unique value that identifies the files system as valid
- **totalBytes:** the total size of the volume in bytes
- **blockSize:** the size of each block in bytes
- **rootLBA:** a logical block address to the root directory
- **mapLBA:** a logical block address to the free map
- **mapCount:** the number of blocks needed to store the free map
- **usedBlocks:** the total number of used blocks in the volume
- **freeBlocks:** the total number of free blocks in the volume

3. A description of the Free Space structure

We keep the counters related to the free map within the volume control block. The actual structure itself is just an integer array that we iterate through bit by bit. After every change, we commit the VCB and free map to the volume.

4. A description of the Directory system

For our file system, we have a single Entry structure for our directories and files.

- name: a 255-length string for the entry's name
- author: a 35-length string for the entry's author
- type: a bit field to indicate whether the entry is free/used and if it's a directory or file
- size: the file's size in bytes
- data: the primary extent table, an array holding 23 extents
- created: the epoch time for when created
- modified: the epoch time for when last modified
- accessed: the epoch time for when last opened

The entry structure represents the directory system, and all directories, including the root directory, "." directory, and ".." directory are all constructed using malloc entry structures. Each directory has a name, type, and one or more data blocks that point to the directory's data. The root directory contains special "." and ".." entries that point to itself and the parent directory, respectively. All entries are written to the volume using the LBAwrite function.

The reason why our structure is laid out this way is due to the assignment requirements, which specify a maximum file name length of 255 characters. Therefore, we decided to use up the entire space of a block for an entry since half of a block would already be used. After adding all the necessary metadata, we had enough room within the block to hold 23 extents for the entry's primary table. However, if we need more than that, we decided to use the "combined scheme" and reserve the last extent as a pointer to other tables in a "double indirect blocks" system.

To differentiate between directories and files, we added an integer “type” in the entry structure. This not only tells us whether the data in the extents are just raw data or can be read as an entry but also handles allocation slightly differently. Instead of creating two entries (thus allocating two blocks), we reserve the first two extents for the “.” and “..” directories and use them as pointers to those entries on the volume. The directory itself has its space for entries allocated on the third extent of its table and by default, can hold 50 entries.

5. A table of who worked on which components

main(), newVolume(), blockAlloc()	group
memFree_extentArray(), volumeRead(), volumeWrite()	Abel Seyoum
freeBlock(), markBlock()	Brian Nguyen
openVolume(), closeVolume()	Jonathan Sum
commitMap(), commitVCB()	Ysidro Alfaro De Leon

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

We have been meeting every day since Sunday (4/2- 4/8)via Discord, where we join a call and share our screens to discuss the goals/tasks for the day. We would meet at whatever time fits our availability. During the meetings, we try to tackle the goals we have set for the day. If we are unable to complete them, we carry them over to the next day’s goals.

These daily meetings have helped us stay on track and ensure that everyone is making progress. By sharing our screens, we can discuss any issues or roadblocks that we encounter and collaborate on finding solutions. We hope to work on better communication and teamwork due to this being the first milestone and there is still room for improvement. Overall, we believe that meeting often is essential for our team’s success.

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

Initially, we assigned one person to do the coding while the rest of us watched their screen and discussed what needed to be done. However, we soon realized that this approach was inefficient, so we split the remaining tasks and started working on them individually. Additionally, we recognized that working on the same file could cause merging conflicts. Therefore, to avoid these conflicts in the future, we are considering creating multiple files so in the future everyone can work independently.

The second major issue that our team faced was not allocating memory using malloc/calloc at the correct times. This would lead to memory leaks when it was time to free the memory. To solve this issue, we used printf statements to print out the memory addresses and understand that the volume control block (VCB) was not getting enough space allocated. This led us to go back to where we initialized the data fields for the VCB and found out that our malloc size would end up being zero since the data fields were not initialized at the malloc.

To avoid such issues in the future, we decided to use dynamic memory allocation more carefully and keep track of all allocated memory to ensure it is released appropriately. Additionally, we will have meetings where we can code review each other when we are having problems with our code and need assistance. This will help us identify any potential issues and find solutions before they become more significant and harder to debug because this is a big file. By taking these steps, we hope to improve the efficiency of our development process.