File System- Milestone 1

Group Name: File Explorers

Team Github Repository: CSC415-2022-Fall/csc415-filesystem-HadiRam

Team Members List:

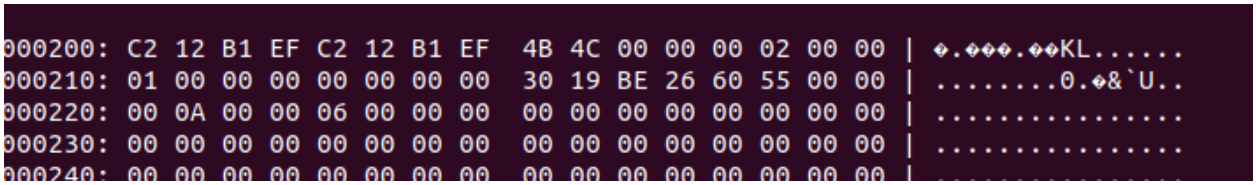
Name	Student ID
Manish Sharma	922220894
Hadi Ramlaoui	922155413
Kimheng Peng	921847378
Rajan Shrestha	922223741

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

Hexdump for Magic Number:

```
#define MAGIC_NUMBER 0xEFB112C2EFB112C2
```

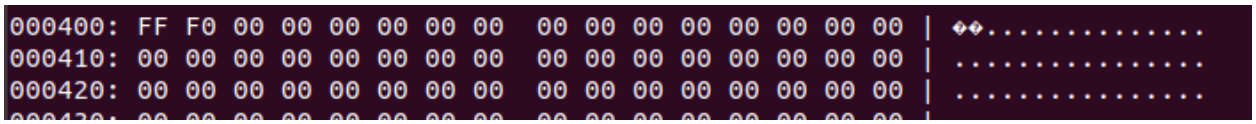
That's our magic number defined for VCB initialization. And we can see that it is shown in little Endian in Hexdump at Address @0002000.



Hexdump for FreeSpace BitMap:

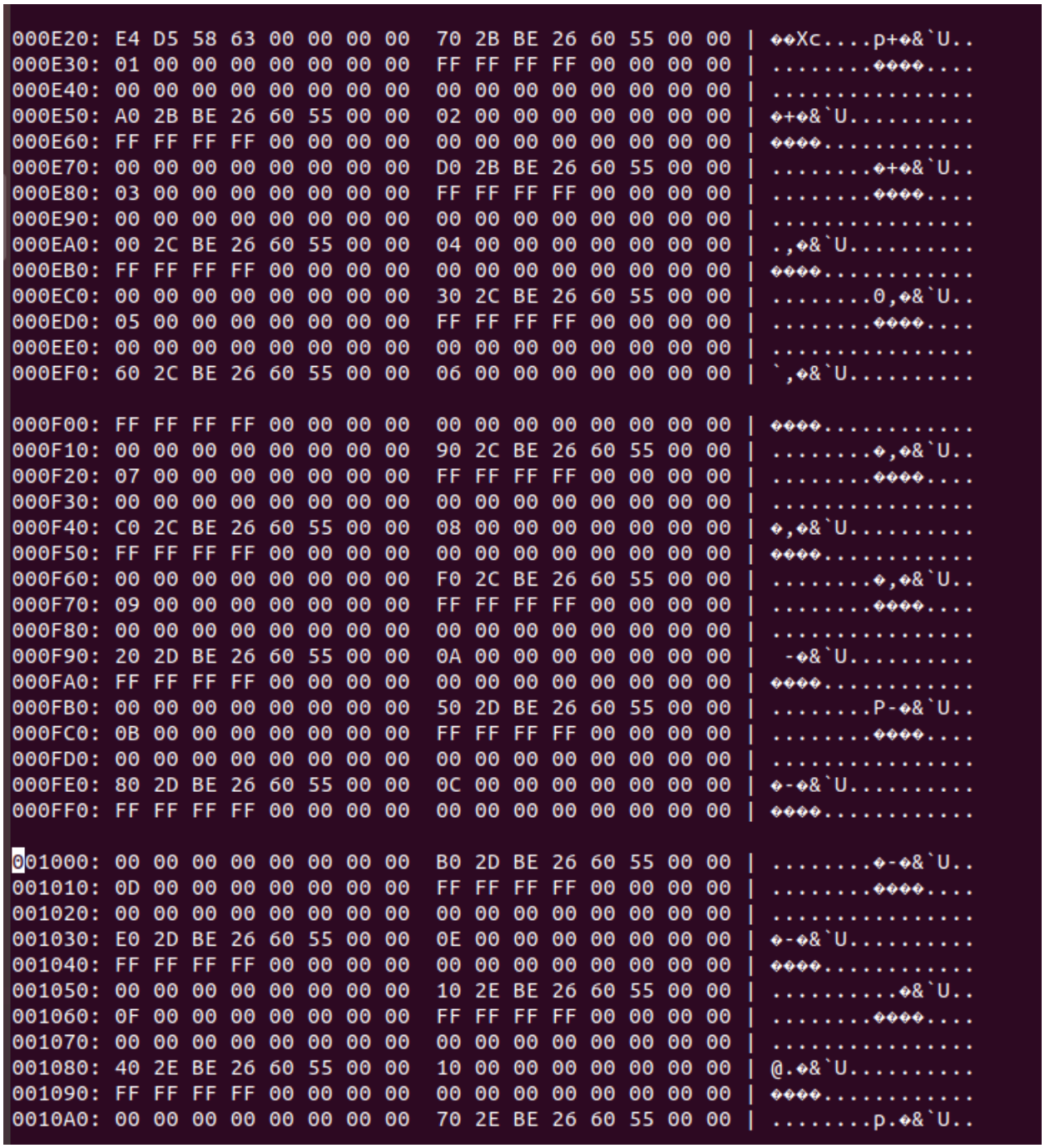
1111 1111 1111 0000

In hexdump we can see at address 000400: FF F0 which is in little Endian format for our bitMap representing the blocks of our disk. So we have said that we have already allocated 12 blocks for our VCB, freespace and root directory. First block or at index 0(highlighted in green) is VCB, from index 1 to index 5 (highlighted in yellow) we have allocated for free space map or our Bitmap and from index 6 to index 11(highlighted in sky-blue) we have allocated it for our root directory.



Hexdump for Root Directory:

Our root directory is starting from address 000E20 in the hexdump.



```
000E20: E4 D5 58 63 00 00 00 00  70 2B BE 26 60 55 00 00 | ◆◆Xc....p+◆&`U..
000E30: 01 00 00 00 00 00 00 00  FF FF FF FF 00 00 00 00 | ........◆◆◆◆....
000E40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E50: A0 2B BE 26 60 55 00 00  02 00 00 00 00 00 00 00 | ◆+◆&`U..........
000E60: FF FF FF FF 00 00 00 00  00 00 00 00 00 00 00 00 | ◆◆◆◆............
000E70: 00 00 00 00 00 00 00 00  D0 2B BE 26 60 55 00 00 | ........◆+◆&`U..
000E80: 03 00 00 00 00 00 00 00  FF FF FF FF 00 00 00 00 | ........◆◆◆◆....
000E90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000EA0: 00 2C BE 26 60 55 00 00  04 00 00 00 00 00 00 00 | .,◆&`U..........
000EB0: FF FF FF FF 00 00 00 00  00 00 00 00 00 00 00 00 | ◆◆◆◆............
000EC0: 00 00 00 00 00 00 00 00  30 2C BE 26 60 55 00 00 | ........0,◆&`U..
000ED0: 05 00 00 00 00 00 00 00  FF FF FF FF 00 00 00 00 | ........◆◆◆◆....
000EE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000EF0: 60 2C BE 26 60 55 00 00  06 00 00 00 00 00 00 00 | `,◆&`U..........

000F00: FF FF FF FF 00 00 00 00  00 00 00 00 00 00 00 00 | ◆◆◆◆............
000F10: 00 00 00 00 00 00 00 00  90 2C BE 26 60 55 00 00 | ........◆,◆&`U..
000F20: 07 00 00 00 00 00 00 00  FF FF FF FF 00 00 00 00 | ........◆◆◆◆....
000F30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F40: C0 2C BE 26 60 55 00 00  08 00 00 00 00 00 00 00 | ◆,◆&`U..........
000F50: FF FF FF FF 00 00 00 00  00 00 00 00 00 00 00 00 | ◆◆◆◆............
000F60: 00 00 00 00 00 00 00 00  F0 2C BE 26 60 55 00 00 | ........◆,◆&`U..
000F70: 09 00 00 00 00 00 00 00  FF FF FF FF 00 00 00 00 | ........◆◆◆◆....
000F80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F90: 20 2D BE 26 60 55 00 00  0A 00 00 00 00 00 00 00 |  -◆&`U..........
000FA0: FF FF FF FF 00 00 00 00  00 00 00 00 00 00 00 00 | ◆◆◆◆............
000FB0: 00 00 00 00 00 00 00 00  50 2D BE 26 60 55 00 00 | ........P-◆&`U..
000FC0: 0B 00 00 00 00 00 00 00  FF FF FF FF 00 00 00 00 | ........◆◆◆◆....
000FD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000FE0: 80 2D BE 26 60 55 00 00  0C 00 00 00 00 00 00 00 | ◆-◆&`U..........
000FF0: FF FF FF FF 00 00 00 00  00 00 00 00 00 00 00 00 | ◆◆◆◆............

001000: 00 00 00 00 00 00 00 00  B0 2D BE 26 60 55 00 00 | ........◆-◆&`U..
001010: 0D 00 00 00 00 00 00 00  FF FF FF FF 00 00 00 00 | ........◆◆◆◆....
001020: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001030: E0 2D BE 26 60 55 00 00  0E 00 00 00 00 00 00 00 | ◆-◆&`U..........
001040: FF FF FF FF 00 00 00 00  00 00 00 00 00 00 00 00 | ◆◆◆◆............
001050: 00 00 00 00 00 00 00 00  10 2E BE 26 60 55 00 00 | ..........◆&`U..
001060: 0F 00 00 00 00 00 00 00  FF FF FF FF 00 00 00 00 | ........◆◆◆◆....
001070: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001080: 40 2E BE 26 60 55 00 00  10 00 00 00 00 00 00 00 | @.◆&`U..........
001090: FF FF FF FF 00 00 00 00  00 00 00 00 00 00 00 00 | ◆◆◆◆............
0010A0: 00 00 00 00 00 00 00 00  70 2E BE 26 60 55 00 00 | ........p.◆&`U..
```

2. A description of the VCB structure

The size of our VCB struct is 24 bytes, the struct contains several variables, first a long signature, which is our unique “magic number” which is used to check if a VCB has already been initialized or not. Next we have several integers to represent the amount of blocks, the size of a block and the amount of free space. The VCB also has a pointer to our freeSpaceBitMap, this pointer is valid only during runtime but has no meaning on the disk. There is also an int to store the bitmap byte size and lastly an int to store the location of the root directory.

```
//Size of VCB is 24 bytes
                    //Volume Control Block struct

                    typedef struct VCB{
                        //unique magic number
                        long signature;

                        int numBlocks;
                        int blockSize;
                        int freeSpace;
                        //Memory Pointer allocated at runtime
                        unsigned char* freeSpaceBitMap;
                        int bitMapByteSize;
                        int RootDir;
                    } VCB;
```

3. **A description of the Free Space structure**

We have used bitmap to represent the free space structure in our file system. Our free space map starts at index 1 of the logical block array of our disk and ends at index 5. Since our disk is of size 10 MB we have 19531 bits to represent each block of size 512 bytes in the disk. We decided if the bit is 1 it is in used/allocated state and if the bit is 0 then it is free/available.

To initialize the free space map we created a helper function initBitMap.

```
void initBitMap(char* bitMapPointer, u_int64_t blockSize)
```

This function takes a pointer to the bitmap and size of a block. It then sets the first byte or 8 blocks to 0xFC which means that the first 6 blocks are already in use for VCB and bitmap. It then loops through the second byte to the size of 5 blocks and sets it to 0x00 to represent that they are in free state. And after that we write back to the disk using LBAWrite.

We have created other helper functions to manage our free space structure. They are listed and explained below:

```
int bitCounter(unsigned char myByte);
```

bitCoutner takes in a byte and counts the amount of bits within that byte that are set to 1. This will return an integer from 0 to 8.

```
unsigned char mask(int offset);
```

Mask takes in an “offset” integer from 0 to 7 and gives the byte where all bits within that byte are set to 0 except for the bit at the position of the “offset”.

```
int freeSpaceCounter(unsigned char myByte);
```

This function will count the bits that are 0 and return the value. In another word, it gives you the remaining number of free bits in a byte.

```
int checkABit(unsigned char myByte, int offset);
```

This function takes in a byte as well as an integer offset and checks whether or not the bit at the offset position of within the given byte is set to 1, if that bit is 0 the function returns 0, if that bit is 1 the function returns 1.

```
int checkForConsecFreeSpace(unsigned char myByte, int count);
```

This function takes a byte and desired number of bits that are free consecutively as count as a parameter to check if there is consecutive free space in the bitmap. If the byte has consecutive free space it returns 1 which means true.

```
void setABit(unsigned char* bitMap, int offset);
```

This function takes in a bitMap and an integer offset, and changes the specific bit within the bitMap at the offset from 0 to 1.

```
void clearABit(unsigned char* bitMap, int offset);
```

This function takes in a bitMap and an integer offset, and changes the specific bit within the bitMap at the offset from 1 to 0.

```
int getConsecFreeSpace(unsigned char* bitMap, int
bitMapSize, int numOfBlocks);
```

This function takes a pointer to bitMap, size of the bitmap, and desired number of blocks as contiguous blocks. Inside this function, first we calculate the minimum amount of free bytes we would need, based on the desired number of blocks, then we have a loop that finds the first free byte within the bitmap, and once that byte is found, it adds to the count of free consecutive bytes, it then continues to the next byte within the bitMap to check if that is also free, and continues to add to that count if that is true. Once the count of free consecutive bytes is equal to the minimum amount of free bytes needed, we have another loop to check for and utilize any free bits at the previous byte, before the first free byte. If there are free bits within that previous byte, then we will then set the last bit that is 0 to the first free block, otherwise the first free block will simply be the first bit of the first free byte. This function then sets the bits to 1 for the bits that are allocated. At the end, this function returns the index to the first free block.

4. A description of the Directory system

Our directory entry struct is of size 60 bytes. It contains a variable pointer to char to store the name of the directory entry. An int variable to store the location of the entry. An int variable to determine the size of the entry. An int variable dirType to know which state is the entry at, if it's free it will be set at -1, if it is a file then it is set to 0 and if it is a directory itself then it's set to 1. We have used two time_t variables for time created and time last modified.

```
//Directory Entry struct with the
size of 60 bytes
                                typedef struct dirEntry{
                                        char* name;
                                        int location;
                                        int size;
                                        //Free state is -1 || is a file directory entry 0
                                || is a directory 1
                                        int dirType;
                                        time_t created;
                                        time_t lastModified;

                                } dirEntry;
```

5. **A table of who worked on which components**

Checkmark indicates that the group member adequately contributed to the component listed within a column of the table.

	VCB Structure	Directory Structure	BitMap and bit Manipulation functions	Initializing VCB, Bitmap/freespace map and Root Directory
Kimheng Peng	✅	✅	✅	✅
Hadi Ramlaoui	✅	✅	✅	✅
Manish Sharma	✅	✅	✅	✅
Rajan Shrestha	✅	✅	✅	✅

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

Our team decided to meet every other day to discuss and work together. We have utilized discord and zoom for online meetings and workspace. For this milestone, we decided not to divide up the tasks but instead fully work on each task together during our group meetings. This was possible since we met regularly from the time this milestone was assigned. The way we worked on tasks was having one group member share their screen with the code open, and all other group members would contribute ideas and suggestions, as well as sending code through discord chat. We found that this approach was extremely effective as it ensured teamwork amongst all group members. It also ensured all group members had the same understanding of the current task we were working on within the milestone.

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

Issue: One of the initial issues that we faced was where to store our Directory Entry and VCB struct.

Solution: At first we wanted to store those struct in mfs.h. However, after further discussion, we thought it would be a better idea to create separate header files for them. We kept the VCB object as an external global object in order to initialize it in fsinit.c. We did the same thing with Bitmap however, we created both the header file and the c file in order to implement the bit manipulation functions.

Issue: We faced an error that said undefined reference to our bitmap functions even though we included the header files.

Solution: We had to modify the makefile in order for it to include the BitMap object. We added the BitMap.o as an object.