File System Milestone One

CSC 415.01

Group A

Youssef Hammoud | 921558141 |

Alexander del Rio | 920764010 |

Joshua Alfaro | 918551821 |

Aneida Blanco Palacio | 918466281 |

Github: ajdelrio

1. Hexdump

1.1 This is the hexdump for the VCB of our filesystem:

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

000200: 00 02 00 00 00 00 00 00  4B 4C 00 00 00 00 00 00 | ........KL......
000210: 01 00 00 00 00 00 00 00  8A 09 00 00 00 00 00 00 | ........◆.......
000220: 05 00 00 00 00 00 00 00  06 00 00 00 00 00 00 00 | ................
000230: 01 00 00 00 00 00 00 00  6E 61 6F 6A 6C 61 6F 79 | ........naojlaoy
000240: 30 39 A9 54 1A 56 00 00  D0 42 A9 54 1A 56 00 00 | 09◆T.V..◆B◆T.V..
000250: D0 42 A9 54 1A 56 00 00  00 00 00 00 00 00 00 00 | ◆B◆T.V..........
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:~/csc415-filesystem-ajdelrio$
```

A closer look on a different run, some values are pointers and will be different on each run:

```
000200: 00 02 00 00 00 00 00 00  4B 4C 00 00 00 00 00 00 | ........KL......
000210: 01 00 00 00 00 00 00 00  8A 09 00 00 00 00 00 00 | ........◆.......
000220: 05 00 00 00 00 00 00 00  06 00 00 00 00 00 00 00 | ................
000230: 01 00 00 00 00 00 00 00  6E 61 6F 6A 6C 61 6F 79 | ........naojlaoy
000240: 30 79 11 4F 51 56 00 00  D0 82 11 4F 51 56 00 00 | 0y.OQV..Ђ.OQV..
000250: D0 82 11 4F 51 56 00 00  00 00 00 00 00 00 00 00 | Ђ.OQV..........
000260: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

Comparing the non-zero values to our VCB definition we can see the values are as expected.

```
000200: 00 02 00 00 00 00 00 00  4B 4C 00 00 00 00 00 00 | ........KL......
000210: 01 00 00 00 00 00 00 00  8A 09 00 00 00 00 00 00 | ................
000220: 05 00 00 00 00 00 00 00  06 00 00 00 00 00 00 00 | ................
000230: 01 00 00 00 00 00 00 00  6E 61 6F 6A 6C 61 6F 79 | ........naojlaoy
000240: 30 39 A9 54 1A 56 00 00  D0 42 A9 54 1A 56 00 00 | 09.T.V..B.T.V..
000250: D0 42 A9 54 1A 56 00 00  00 00 00 00 00 00 00 00 | .B.T.V..........
000260: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

Bytes 0x200-0x207 are our uint64_t blockSize, with value 0x200 which is equal to 512 in decimal, the expected value.
Bytes 0x208-0x20F are our uint64_t numBlocks, with value 0x4C4B which is equal to 19531 in decimal, the expected value for a disk size of 10,000,000 bytes separated into 512 byte sectors.
Bytes 0x210-0x217 are our uint64_t freeMapAddr, with value 0x01, meaning our freespace map is stored in block 1 on disk which is correct.
Bytes 0x218-0x21F are our uint64_t freeLen, the length in bytes of our freespace map, with value 0x098A, which is equal also equal to 19,531 which is correct, but maybe we can just use numBlocks in lieu of this additional variable.
Bytes 0x220-0x223 are our uint32_t mapBlocks, with value 0x05, which is equal to 5 in decimal, the correct number of blocks needed by our freespace map.
Bytes 0x224-0x227 are padding added by the compiler, we could do away with this by more efficiently ordering our variables in the VCB.
Bytes 0x230-0x233 are our uint32_t rootSize, with value 0x01, equal to 1 in decimal, which is the correct number of blocks our root directory occupies.
Bytes 0x234-0x237 are padding added by the compiler, we could do away with this by more efficiently ordering our variables in the VCB.
Bytes 0x238-0x23F are our signature, with value 0x796F616C6A6F616E, which translates to “yoaljoan” when converted to ASCII, a combination of the first two letters of each of our names, which is the expected value.
Bytes 0x240-0x247 are for the uint8_t* freeMap variable, a pointer to the freeMap. This value on disk is not a valid address and this variable must be reinitialized each run.
Bytes 0x248-0x24F are for the dirEnt_t* root variable, a pointer to the root directory. This value on disk is not a valid address and this variable must be reinitialized each run.
Bytes 0x250-0x25F are for the dirEnt_t* cwd variable, a pointer to the current working directory, This value on disk is not a valid address and this variable must be reinitialized each run. We can see that it has the same value as our root pointer, this is because the cwd is set to root on initialization.

The remaining bytes in the VCB are not used, so the values of 0x00 are expected.

1.2 The second dump shows us the boundaries in which the free space begins and ends:

```
student@student-VirtualBox:~/csc415-filesystem-ajdelrio$ ./Hexdump/hexdump.linux SampleVolume --count 5 --start 2
Dumping file SampleVolume, starting at block 2 for 5 blocks:

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 | ................
```

```
000CF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000D00: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000D10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000D20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000D30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000D40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000D50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000D60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000D70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000D80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000D90: 00 00 00 00 00 00 00 00  41 FD 01 00 00 00 00 00 | ........A.......
000DA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000DB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000DC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000DD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000DE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000DF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

student@student-VirtualBox:~/csc415-filesystem-ajdelrio$
```

The only bytes that are set are the very first byte, 0x400, and bytes 0xD98-0xD9A.

Byte 0x400 has a value of 0x7F, which is equivalent to 0b01111111. Since the first 7 bits are set to 1, this indicates that the first 7 blocks of our disk are in use, which is expected as our VCB and root directory each use 1 block, and the freeMap uses 5 blocks, for a total of 7 blocks.

The values at 0xD98-0xD9A appear to be garbage values, as they come after the end of the freeMap and are not allocated or initialized. They do not appear meaningful, but further investigation is required to confirm.

2.VCB structure

```
typedef struct{
      uint64_t blockSize;     // Number of bytes per block;
      uint64_t numBlocks;     // Number of blocks on this volume
      unit64_t freeMapAddr;   // LBA of free-space bitmap
      uint64_t freeLen;       // Number of bytes in the freeMap
      unit32_t mapBlocks;     // Number of blocks freeMap occupies
      unit64_t rootAddr;// LBA of the root directory
      unit32_t rootSize;// Number of blocks the root directory occupies
      uni64_t ourSig;         // Signature for our filesystem

      /* These are pointers to be used during operation and must be
      initialized on each run. The values stored on disk will not be valid*/

      uint8_t* freeMap;       // Pointer to freeMap
      dirEnt_t* root;         // Pointer to root directory
      dirEnt_t* cwd;          // Pointer to the current working directory
} vcb_t;
```

The VCB (volume control block) is a structure that contains important information such as the number of blocks on the volume, the freemap logical block address, and the root directory logical block address. We will also use the VCB to store pointers to the freeMap, root directory, and current working directory for easy access while our filesystem is running.

We have plenty of room for other fields we may wish to add later, but we were able to complete the steps for milestone 1 with only these so far.

3. Free Space

```
uint8_t* bitV;
void setBit(uint8_t* byte, uint8_t bit);
void unsetBit(uint8_t* byte, uint8_t bit);
void isBitFree(uint8_t byte, uint8_t bit);
uint64_t allocBlocks(int count);
```

Our free space management uses a bitvector, bitV, to store an array of bytes (uint8_t), whose bits represent blocks on the disk. If a bit is 0, it means the block of the same index is free to use, if it is 1, then that block is in use and not available.

We found it helpful to write a few helper functions to take care of actions we'll commonly need to do:

setBit() takes a byte address and a bit and sets that bit on that byte to 1.

```
void setBit(uint8_t* byte, uint8_t bit) {
    bit = 1 << bit;      // set the bitmask
    *byte |= bit;        // OR byte with the bitmask
}
```

`unsetBit()` takes a byte address and a bit and sets that bit on that byte to 0.

```
void unsetBit(uint8_t* byte, uint8_t bit) {
    bit = 1 << bit;      // set the bitmask
    bit ^= 0xFF;         // XOR with 0xFF to invert the bitmask
    *byte &= bit;        // AND byte with the bitmask
}
```

`isBitFree()` takes a byte and a bit and returns 1 if that bit is set to free (0) or 0 if that bit is in use (1).

```
int isBitFree(uint8_t byte, uint8_t bit) {
    bit = 1 << bit;             // set the bitmask
    if((byte & bit) == 0) {     // if byte & bit == 0, the bit is free
        return 1;
    }
    else {
        return 0;
    }
}
```

`allocBlocks()` takes a number, tries to find that many free contiguous blocks on the freeMap and if successful, marks those bits as in use on the freeMap and returns the starting LBA of those blocks, if it fails it returns 0, as we know we will not be overwriting the VCB on block 0.
(Too large to screenshot)

4. Directory system

```
Typedef struct {
      time_t dateTimeCr;// date and time the file was created
      time_t dateTimeMd;// date and time the file was last modified
      uint64_t location;// LBA of this file's first block on disk
      uint32_t size;          // size of the file
      char name[35];          // Name and extension
      uint8_t attr;           // Attributes
} dirEnt_t;
```

The directory system creates the directories. It creates the root directory and initializes its attributes (root name, size, location, etc.). The date and time the file is created will be stored in dateTimeCr. Each time the file is modified, the date and time of such action will be stored in dateTimeMd. The LBA of the file's first block is stored in the location field. The size field will store the size of the file. The name field will store the name and extension of the file, delimited by "." if applicable. Finally, the attr field will store the attributes of the file.
So far we have 2 attributes we wish to use:

`0b00001111` will mean that directory entry is for the root directory
`0b00000001` will mean that directory entry is for a directory

As we progress we will try to think of other useful information to store in the attr field.

5. Division of Work

Task	Name
VCB	Alex
Free Space	Josh + Youssef + Alex
Directory System	Youssef + Alex
Write-up	Aneida + Josh + Youssef + Alex

6. How our team worked together

Our team collaborated using the discord application. In the application we are able to communicate with each other and have meetings. The meetings are important because it is where the team can voice their opinion and comments regarding certain tasks from the file system. We are also able to share our screen with one another in case we need help debugging or simply need help understanding a topic. Our team was able to talk in the

group chat and discuss our concerns and ideas regarding the assignment. We made sure to create branches so that we would not push to the master. It is not ideal to push into the master, due to many issues we could face in the future. Our team divided work from one another and each worked in what they felt they were most skilled at.

7. Issues we faced

Issues we faced as a team was finding time to work together as a team. Our team has a very busy schedule so it is important that we coordinate appropriately. Since it is Summer, we are on a smaller time frame. So it is important to keep improving our workload in order to meet the deadlines.