CSC415 - Operating Systems

Team Name: Satisfaction

Team Info:

Member Name	Student ID#	Github Name
Sulav Jung Hamal	923075813	Sulavjung
Miguel Maurer	922097199	miguelCmaurer
Yuvraj Gupta	922933190	YuvrajGupta1808
Fasika Abera	923038932	Fasikaabera

Github Name with our group work: miguelCmaurer

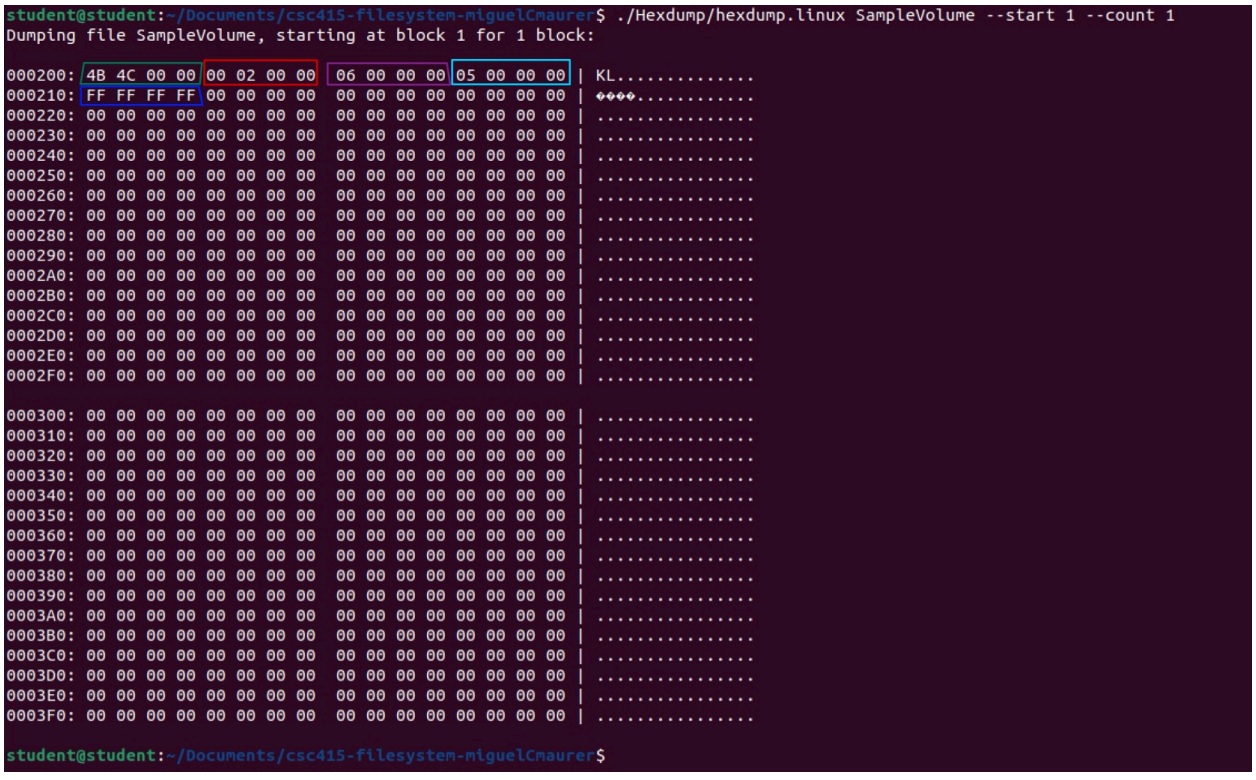
File System Assignment: Milestone 1

Description:

In this milestone, we initialized and wrote the Volume Control Block (block 0), ensuring our volume is ready for use. We also implemented the free space management system, which included initializing free space and creating an allocation procedure to allocate free space. Additionally, we wrote and initialized the root directory, ensuring it contains the essentials `.` and `..` entries.

Hexdump analysis

VCB analysis:



1. `totalBlocks` : Because the first value in the VCB struct is the total blocks it appears first in the hexdump. Because it is an int it takes up 32 bits, or 8 hex digits. Because the representation of all values are in little-endian format the value is 0x4C4B = 19531.This is correct because when the program runs it prints out how many blocks there are:

```
Initializing File System with 19531 blocks with a block size of 512
```

2. `blockSize` : Block size is the size of each of the blocks. This is represented by an int. So, it will also take up 32 bits, or 8 hex digits. Also, because we already know all values in the hexdump will be in a little-endian format the value is 0x200 = 512, this is the expected value as shown above.

3. `locRootDir` : The location of root directory is stored next, similar to all the other values it is also in so 8 hex digits, which is expected to be at 6, this is because the first block is used for teh VCB and the following five is used for the bitmap representation of free block.

4. `freeSpace` : The freespace field represents the number of blocks that are being used to represent the freespace and we know that this value is five and will take up 8 hex digits. We know this is correct because the location of the root directory is after this, so 1 for the VCB + 5 for freespace.

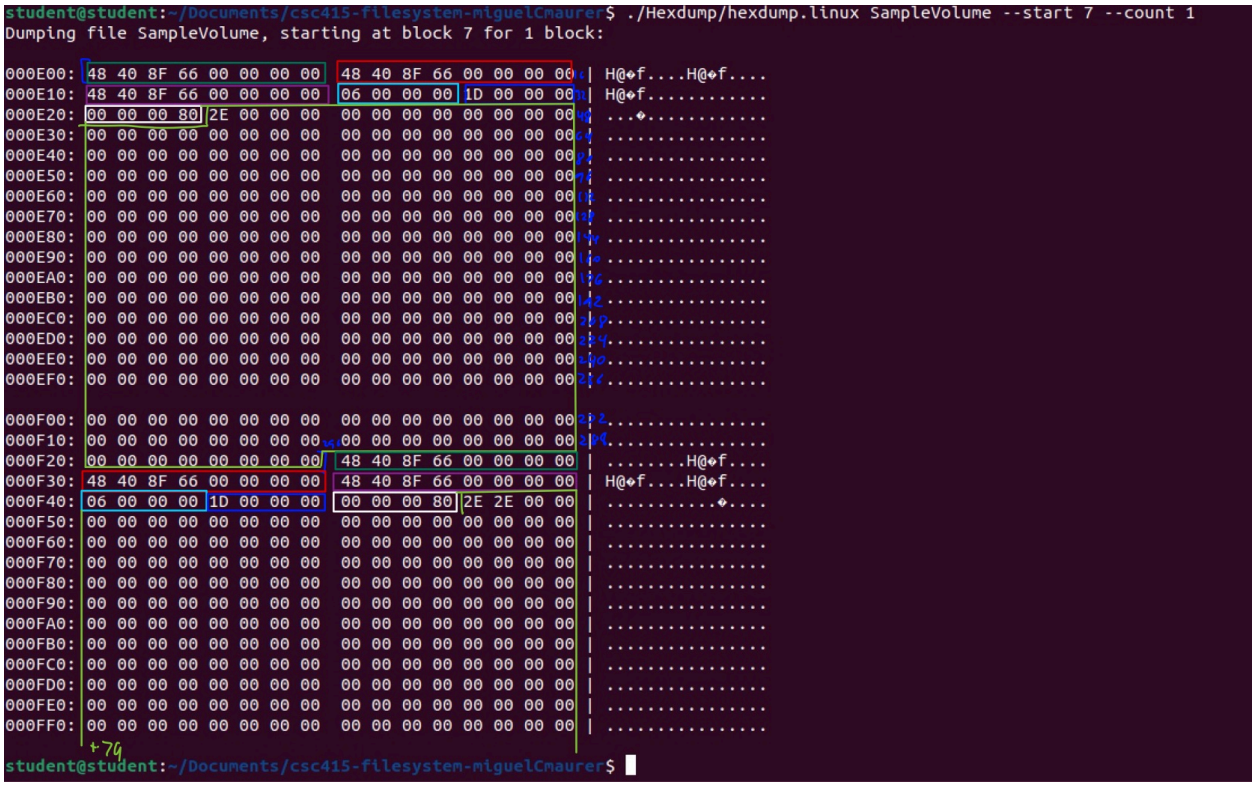
5. `signature:` Because the current signature being used is -1 mostly because it was easy to distinguish in the hexdump, with a value of 0x FF FF FF FF, which is the value that is seen in the screenshot.

FreeSpace analysis:

```
student@student:~/Documents/csc415-filesystem-miguelCmaurer$ ./Hexdump/hexdump.linux SampleVolume --start 2 --count 1
Dumping file SampleVolume, starting at block 2 for 1 block:

000400: FF FF FF FF 07 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:~/Documents/csc415-filesystem-miguelCmaurer$
```

Because the free space bitmap is not a struct there is little need to highlight the important bits, and because the used blocks can all be represented in the first block, the first block shows all the important information about the free space. We can verify that this is correct by knowing what is being saved in the current program. We know that the first 6 bits should be 1, because the VCB and the freespace itself. The remaining bits being used come from the initialization of the root directory. When we initialize it we ask for 50 entries, and with a size of 296, which means that we need 14800 bits -> 29 blocks, so we expect the free space to have 35 bits to the used state. The value we get from the free space is 0x7FFFFFFFF = 111 1111 1111 1111 1111 1111 1111 1111 1111(35 bits).

Root directory analysis:



1. `creationTime`: The creation time is a time_t type which means that it takes up 8 bytes -> 16 hex digits, the creation time portion of the struct is in dark green, this is the time since 1970 in seconds, the value represented here is about 7:15 on july 11. This is the expected value because its when the volume and rood dir were initialized.

2. `modificationTime`: The modification time is set to the same time as the creation time on initialization of dir. Thus, we can see that the hex dump is the same.

3. `accessTime`: Same logic goes for the access time as the `modificationTime`. Thus, we can see the same hex value as the `creationTime` and `modificationTime`.

4. `location`: The location is the following value which is shown to be 6, this is correct because the location of this directory (root) is at the 6th block, after the VCB which is block 0, then the 5 blocks used for the free space bitmap.

5. `size`: This represents the size of the entry stored. In this case both the . and .. represent the same directory so they are the same expert for the name. Here the value is 0x1d =

29, we know this is correct because in the freespace we proved that 29 blocks are required for the root directory.

6. `permissions`: The only permission that has been included this far is a flag to represent if a directory entry is a directory or not. The value 0x08000000 => 0b00001000000000000000000000000000, this is correct because one bit(directory flag) has been marked to be a directory.

7. `name`: For the name we can see that the first entry has a name of 0x2E => '.' , this name is followed by 254 "00". This is because the name has a max length of 255 but we are only using 1 character. The next struct has identical values except the name has a the value of 0x 2E 2E, which means it is the '..' directory with the exact same values.

VCB Structure:

The VCB (Volume Control Block) structure is designed to store the metadata about the disk. Here is what our volume control block structure looks like.

```
typedef struct VCB
{
      int totalBlocks;
      int blockSize;
      int locRootDir;
      int freeSpace;
      int signature;
} VCB;
```

1. `totalBlocks` : This field represents the total number of blocks available on the volume.
2. `blockSize` : Indicates the size of each block in bytes. This field is used to determine the size of memory allocation and to calculate the number of blocks needed for various structures. This will be set when the volume is first initialized.
3. `locRootDir` : This field stores the location of the root directory. This field is used to quickly access the root directory, which is the starting point for navigating the file system.
4. `freeSpace` : This field stores the number of blocks that are being used for the free space bitmap. This is calculated and set during the volume initialization.
5. `signature:` This acts as a unique identifier to verify that the volume has been initialized.

Directory Structure:

The `DirEntry` structure represents a directory entry in a file system. This structure is useful to track different attributes associated with a file or directory. Here is what our directory structure looks like.

```
typedef struct DirEntry{
 time_t creationTime;
 time_t modificationTime;
 time_t accessTime;
 int location;
 int size;
 int permissions;
 char name[MAX_LEN_FOR_NAME];
} DirEntry;
```

Here is the breakdown of each attribute and how they are being used and will be used within our filesystem.

1. `creationTime`: The time when the directory entry was created. It is set to the current time when the directory entry is initialized.
2. `modificationTime`: The time when the directory entry was last modified. It will be changed each time we write something on the file.
3. `accessTime`: The time when the directory entry was last accessed. It will be used or updated when we read from the directory entry.
4. `location`: The location will give us where the file blob is written. It will give us the location of the block number.
5. `size`: The size of the directory entry in the block. Initially set to 0 for the directory that is not being used.
6. `permissions`: The permission to read, write, or execute associated with the directory entry will be stored with the help of these attributes.
7. `name`: The name of the directory entry set by the user will be saved with this attribute.

Group Name: Satisfaction
Github: miguelCmaurer CSC 415 Operating System

Table of who worked on which components

Team Member Name	Worked On
Sulav Jung Hamal	Writeup, code requirement checking and testing.
Miguel Maurer	Majority of coding, testing, and writeup.
Yuvraj Gupta	Code formatting, header requirement and writeup.
Fasika Abera	Writeup

How did your team work together?

Our team worked together using Slack for daily communication and Zoom for brainstorming sessions. Also, we used Github to know each other's progress on coding and we maintained a shared writeup document to track our ongoing work effectively.

How often did the team meet?

Since we were using Slack, we communicated using it every day to keep track of where we were on what we were doing. Additionally, we held one Zoom meeting of about 1 hour and 30 minutes to divide tasks and discuss complex problems. Yet, we are working on increasing that to regular meetings for Milestone 2.

How did we meet?

1. Slack
2. Zoom Meeting

How did we divide up the tasks?

Miguel had already completed much of the coding before our first meeting, where he explained the logic behind each method and file. We then divided the remaining tasks as: one handled the write-up, another checked the logic, one focused on coding, and the fourth ensured all milestone requirements were met. We are working to improve our task division for the next milestone.

Issues and Resolutions

1. One of the key issues we faced during Milestone 1 was determining the starting block for the directory and correctly initializing the `.` and `..` entries in the root directory. We initially didn't quite understand what or more importantly how to start working on it.
   Resolution:
   Our solution to that was to determine the starting block by offsetting from block 0, accounting for the blocks used by the Volume Control Block(VCB) and the bitmap. Then, we set up the directory entries as follows:

   1. We allocated memory for the directory entries based on the number of blocks required.
   2. The `.` entry was set up with its `location` attribute pointing to the current directory block, and its `name` set to `".`.
   3. The `..` entry was setup with its `location` attribute pointing to the parent directory block (or itself if it is the root directory), and its `name` set to `".."`.
   4. We marked the relevant blocks as used in the free space bitmap to ensure they were not overwritten.