Debug Thugs

Members:

Michael Thompson - 922707016
Utku Tarhan - 918371654
Eric Ahsue - 922711514
Randy Chen - 922525848

GitHub: **Jasuv**

Michael Thompson	**Utku Tarhan**	**Eric Ahsue**	**Randy Chen**
922707016	918371654	922711514	922525848

Assignment - File System Milestone One

Description:

For this milestone of the project, the main goal is to get the volume “formatted”. Formatting the volume requires that the Volume Control Block (block 0) is written, the Free Space Management is written, and the Root Directory is written and initialized with the ‘.’ and ‘..’ entries.

Approach:

Our approach was to separate and distribute the assignment among our team members. Each person was responsible for a specific part of the file system initialization. Michael focused on the VCB initialization. He began by allocating memory for the Volume Control Block using malloc and reading block 0 using LBAread to load the volume's data into the VCB structure. Then he checked the signature (magic number) in the structure. If the signature matched the expected value, it meant the volume was already initialized. If it did not, further initialization was needed.

Next, Utku focused on initializing and allocating the free space and after discussing with the group, we decided that it would be better to use a bitmap. We assume that the bitmap will manage free space separately from the VCB, which will only hold the starting block number of the free space map. First, we plan to allocate the necessary memory using malloc to get enough space for the bitmap. Then, we will initialize the bitmap by marking the blocks reserved for system data as used and the remaining blocks as free. Next, we will write the bitmap to disk using LBAwrite to write the designated blocks. Finally, we will update the VCB with the starting block number of the free space so that the system knows where free space begins.

After Utku implements the free space manager, Eric will utilize his block allocation function for allocating the first few blocks (besides the first VCB Block) for the root directory, and populate it with “free” entries. He will then initialize the first 2 entries of root, itself ‘.’ and its parent (itself) ‘..’

For the last part, Randy plans on using 512 bytes for each block, 128 bytes per directory entry, and 6 total blocks. To find our entry size, Randy multiplies the size of our directory entry by the number of entries, leaving us with the size of our entry.

A description of the VCB structure:

The VCB stores necessary metadata about the disk volume. In our code, we defined it in mfs.h as vcb_struct. The VCB contains:

1. volume_name[64]: A character array that stores the name of the volume.
2. block_size: The size of each block in bytes. This is passed in when we initialize the file system.
3. block_count: The total number of blocks in the volume.
4. freespace_list_start: The starting block number for the free space list.
5. root_dir_start: The starting block number for the root directory.
6. signature: An identifier to detect if the volume has already been formatted. (We chose 0x4275675468756773, which is “BugThugs”).

The VCB is stored in the first block (block 0) of the disk volume. When the system starts, it reads this block to determine if the volume is already formatted and to get the necessary information to access other important parts of the file system.

A description of the Free Space structure

The initFreeSpace() function is responsible for implementing the initialization logic of the free space management system using a bitmap-based approach. This bitmap tracks which blocks on the volume are free or in use. Our freespace.c / and freespace.h holds everything relevant to the free space handling that we decided as a team, and allows us to dynamically allocate storage blocks for files and directories.

During initialization, memory is allocated for the bitmap based on a few things. **BLOCK_COUNT** defined in mfs.h are the block counts that we decided to block for holding file system relevant information. So Initialization would account for block 0 + 6 other blocks and mark these as used.

Next, once initialized, the bitmap is written to the disk using *LBAwrite()* starting at block 1. Which would make the free space state persistent. This setup allows us to dynamically allocate free blocks using *allocateBlocks()* function which we also implemented following this logic.

The allocateBlocks() function handles the dynamic allocation of free blocks by scanning the bitmap for available space that is available besides the FS used blocks. The key part of the logic was to be able to allocate multiple blocks at once dynamically, and handle any potential edge cases. For instance if initFs() requested 10 blocks and if there was no available space for that request. We would have to account for rolling back the allocated space. If the request was able to be handled properly, the function then returns a pointer to int array that holds the block numbers that were allocated. If there were no allocated block numbers, or updating bitmap fails, we simply return a NULL. The decision I made behind this was intended to allow us to get precise block numbers in the bitmap to write files/directories, and do allocations en masse, instead of calling functions one by one. The allocateBlocks() function returns a pointer to an array of ints, where each int is a block number that was allocated. This works well because we are storing files discontiguously because blocks might not necessarily be together.

Michael Thompson	**Utku Tarhan**	**Eric Ahsue**	**Randy Chen**
922707016	918371654	922711514	922525848

A description of the Directory system

The dirOperations file will be responsible for all directory related functions (creating new directories, getting parent directories, etc.). This file utilizes functions from the free space manager to allocate/get blocks for newly created directories.

The newDir function is responsible for creating new directories and writing them to disk. It takes in the starting block of its parent directory as the only parameter. That's because if the parent starting block is -1 (no parent), then we are creating the root directory. First it will get an array of blocks that are allocated by the free space manager, then initializes all its entries as "free". Then the '.' and '..' entries are created; the '..' changing properties if it's a root directory (no parent block). Finally the newly created directory is written to disk and the starting block is returned.

A table of who worked on which components:

Michael	**Utku**	**Eric**	**Randy**
● vcb init ● description of the vcb structure	● free space init & alloc	● directory initialization ● hexdump analysis	● root dir init in collaboration with eric ● worked on documentation

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

When we first began working on milestone one, we decided that we should set up a good system for meeting up and checking in on each other. We decided that we should meet either every other day or every two days just to check in and see how things are going, these meetings could either be in person or online on Zoom or Discord. The instructions laid out for the first milestone in three steps, so we decided we would split the work between the steps, with each person assigned to one step with the other person being there to help work on each part and work on the PDF.

Issues and Resolutions:

- Since we wanted to avoid branching and merging as much as possible we planned to segment the workload and work on our own respective files to avoid any collision. This made pulling and pushing to the repo much safer without the use of any branches.

Michael Thompson	**Utku Tarhan**	**Eric Ahsue**	**Randy Chen**
922707016	918371654	922711514	922525848

- During the meetings for how we wanted to approach the assignment, Randy wanted to use a 512 byte block size. Other members of the group disagreed with him saying that it was too much for our file system because we did not need to store that many directories. Ultimately, we decided to just leave it at the default 512 and change it in the future if needed.

- We also had to decide how many entries should be in a directory. Initially we wanted 128 entries, but then each directory would take up 24 blocks. So we decided on only 32 entries for now as 6 blocks is much more reasonable.

Analysis:

(analysis explaining our hexdump, showing the VCB, free space, and root directory)

```
student@student:~/Documents/csc415-filesystem-Jasuv/Hexdump$ ./hexdump --file
SampleVolume --start 1 --count 1
Dumping file SampleVolume, starting at block 10 for 1 block:

000200: 44 65 62 75 67 54 68 75  67 73 20 56 6F 6C 75 6D | DebugThugs Volum
000210: 65 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | e...............
000220: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000230: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000240: 00 02 00 00 00 00 00 00  4B 4C 00 00 07 00 00 00 | ........KL......
000250: 09 00 00 00 00 00 00 00  73 67 75 68 54 67 75 42 | ........sguhTguB
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 | ................

volume name: “DebugThugs Volume”
block_size: 512
block_count: 19531
free space start block: 7
root dir start block: 9
vcb_signature: BugThugs
```

Michael Thompson	**Utku Tarhan**	**Eric Ahsue**	**Randy Chen**
922707016	918371654	922711514	922525848

```
student@student:~/Documents/csc415-filesystem-Jasuv/Hexdump$ ./hexdump --file
SampleVolume --start 2 --count 1
Dumping file SampleVolume, starting at block 10 for 1 block:

000400: 01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 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 | ................

block 1-6: VCB
block 7-8: free space map
block 9-15: root directory
```

```
student@student:~/Documents/csc415-filesystem-Jasuv/Hexdump$ ./hexdump --file
SampleVolume --start 10 --count 1
Dumping file SampleVolume, starting at block 10 for 1 block:

001400: 2E 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001410: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001420: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001430: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001440: D0 00 00 00 00 00 00 00  09 00 00 00 00 00 00 00 | ................
001450: 3E 81 F4 67 00 00 00 00  3E 81 F4 67 00 00 00 00 | >g....>g....
001460: 01 00 00 00 00 00 00 00  2E 2E 00 00 00 00 00 00 | ................
001470: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001480: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001490: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0014A0: 00 00 00 00 00 00 00 00  D0 00 00 00 00 00 00 00 | ................
0014B0: 09 00 00 00 00 00 00 00  3E 81 F4 67 00 00 00 00 | ........>g....
0014C0: 3E 81 F4 67 00 00 00 00  01 00 00 00 00 00 00 00 | >g............
0014D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

Michael Thompson	Utku Tarhan	Eric Ahsue	Randy Chen
922707016	918371654	922711514	922525848

```
0014E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0014F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

1st directory name: .
2nd directory name: ..
directory size: 208
starting block: 09
timestamp: 1744077118
is directory: true
```

Screen shot of compilation:

```
student@student:~/Documents/csc415-filesystem-Jasuv$ make
gcc -c -o fsshell.o fsshell.c -g -I.
gcc -c -o fsInit.o fsInit.c -g -I.
gcc -c -o freeSpace.o freeSpace.c -g -I.
gcc -c -o dirOperations.o dirOperations.c -g -I.
gcc -o fsshell fsshell.o fsInit.o freeSpace.o dirOperations.o fsLowM1.o -g -I. -lm -l readline -l pthread
student@student:~/Documents/csc415-filesystem-Jasuv$
```

Screen shot(s) of the execution of the program:

```
student@student:~/Documents/csc415-filesystem-Jasuv$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does not exist, errno = 2
File SampleVolume not good to go, errno = 2
Block size is : 512
Created a volume with 9999872 bytes, broken into 19531 blocks of 512 bytes.
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Initializing File System with 19531 blocks with a block size of 512
Allocated blocks: 7 8
Volume formatted!
|---------------------------------|
|------- Command ------|- Status -|
| ls                   |    OFF   |
| cd                   |    OFF   |
| md                   |    OFF   |
| pwd                  |    OFF   |
| touch                |    OFF   |
| cat                  |    OFF   |
| rm                   |    OFF   |
| cp                   |    OFF   |
| mv                   |    OFF   |
| cp2fs                |    OFF   |
| cp2l                 |    OFF   |
|---------------------------------|
Prompt > exit
System exiting
student@student:~/Documents/csc415-filesystem-Jasuv$
```