File System - M1

Group Name: BeerMan

Members in Group:
Andy Almeida - 922170012
Masen Beacham - 918724721
Christian Mcglothen - 918406078
Kendrick Rivas - 922898506

Github:

https://github.com/CSC415-2023-Summer/csc415-filesystem-chris3953.git

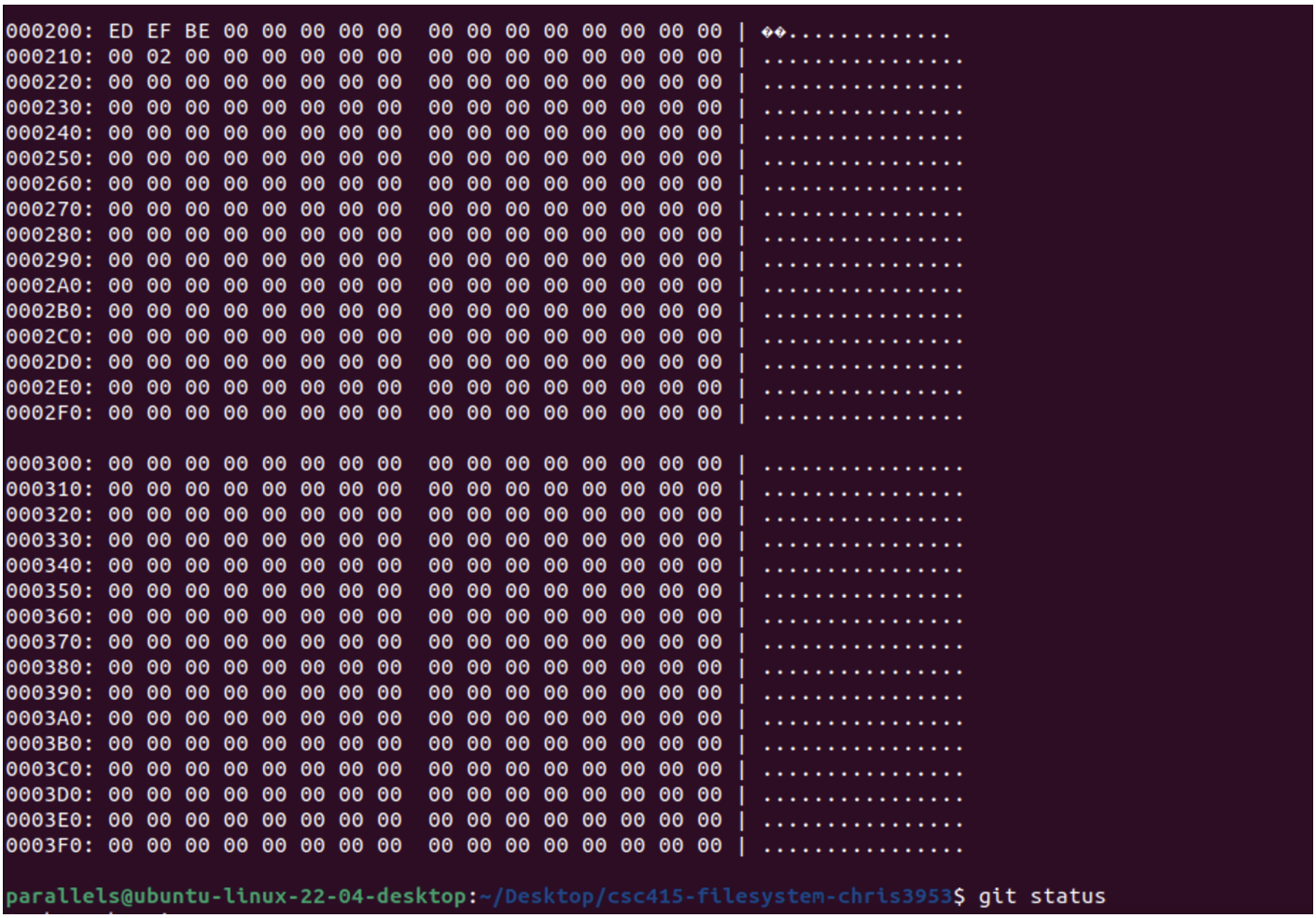
Game Plan/Understanding of Project:

- Determine the file system layout: Decide on the overall organization of the file system, including the block size, the number of blocks in the file system, and the starting block for free space.
- Create the VCB: Allocate memory for the VCB structure and initialize its fields. Set the root block number, totalBlocks, block_size, free_space_start_block, and the signature as appropriate.
- Design the directory structure: Determine how directory entries will be organized and stored. In this case, the DirectoryEntry structure includes information such as size, location, name, type, and timestamps. You can define additional structures or data structures as needed to support directory operations efficiently.
- Implement file and directory management functions: Create functions to handle file and directory operations such as creating a new file or directory, deleting files or directories, moving or renaming files, and listing directory contents. These functions should interact with the VCB and directory structures to manage file system metadata.
- Implement file allocation and extents management: Implement a mechanism for allocating file blocks and managing file extents (contiguous blocks of a file). The Extents structure, present within the DirectoryEntry, can be used to track the location and size of each extent in a file.
- Implement free space management: Develop a method to track and manage free space within the file system. The free_space_start_block field in the VCB can be used to mark the starting block of free space. You may use a free space bitmap, a linked list, or any other suitable data structure to manage free blocks efficiently.
- Implement file system initialization: When creating a new file system, initialize the VCB, set appropriate values for the fields, allocate and initialize the root directory block, and set up any necessary data structures to ensure a functional file system.
- Implement persistence: If you want the file system to persist across sessions, you need to implement mechanisms to store and load the file system data from a storage medium (e.g., a disk). This typically involves reading and writing data structures to/from disk blocks.
- Develop supporting functions: Implement additional functions as needed, such as functions for file read/write operations, file attribute modification, error handling, and integrity checks.

Explanation of the HexDump:

Starting from address: 000200: this is where we initialize the VCB in fsInit.c

The value **ED EF BE** represents the signature and the value at address 000210 represents what we call the block_size in our VCB struct with a value of **00 02**. The rest of the block should be zero as we have no other entries or values to be placed in this block, as the project progresses we imagine that the rest of the addresses in block 2 will be filled with information in regards to other directory entries.



A description of the VCB structure:

The volume control block serves as a main point of information regarding the state of the volume that we are allocating for space. To complete this, the VCB contains fields that help the system figure out its most important data. In our VCB, we have the signature which is a field that will help us ensure that our volume control block is correctly allocated as expected, followed by [root] which holds the number indicating where the directory starts, [totalBlocks] which indicated the total number of blocks in the system, [block_size] which indicates the size of a block in bytes, and [free_space_start_block] which indicates the block at which the free space starts. All

fields except for the signature holds its value as an int, whereas the signature is in the form of an unsigned long.

Description of the Free Space structure:

Our freespace system has a functionality of allocating and releasing free space. It uses a simple bitmap representation to track the availability of blocks and provides functions to allocate and release contiguous blocks as needed. The FreeSpace[] is an array of unsigned char in bytes, where each bit of each byte represents a block of storage. If the bit is 0, the block is free; if the bit is 1, the block is occupied. The size of the array is defined by FreeSpaceTotal, which is 2560 bytes. Given that each byte has 8 bits, this means the system can manage 8 * 2560 blocks of storage. The next four are the functions within FreeSpace.c file and what they do:

InitFreeSpace(): This function initializes the FreeSpace[] array, setting all bits to 0 (all blocks free). It then requests a certain amount of free space, adds 6 (as given in the instructions for allocating free space) to the requested space, and writes the FreeSpace[] array to disk.

GetFreeSpace(int NumberOfBlocks): This function is used to find and allocate a certain number of blocks of free space. It uses SearchForSpace() to find the first block of sufficient free space, then marks the necessary number of blocks as occupied by setting the corresponding bits in FreeSpace[] to 1. If there's not enough space, it returns -1.

SearchForSpace(): This function loops through the FreeSpace[] array to find the first bit that's set to 0 (indicating a free block). It returns the location of this bit.

ReleaseSpace(int Location, int NumberOfBlocks): This function takes a starting location and a number of blocks, and frees up that space by setting the corresponding bits in FreeSpace[] to 0.

Description of the Directory system:

Our directory system starts at the beginning of the storage space with our Volume Control Block. Our VCB contains the location for the root block. With this location, we can begin to allocate our Directory Entries. The directory entries contain the data on the information that relates to the blocks containing the files/directories saved in our file system. The directory entries contain data such as the file name, the file type, file size, metadata (creation, last modified, last opened), and very importantly, its location block in the file system. With the use of Directory Entries, we can keep track of files and their information/location. Directory Entries on directories would simply point us to a block that contains another list of Directory Entries. This is the basis for our File System.

A table of who worked on which components:

Component	Contributors
Directory Entry	Andy, Chris
VCB	Andy, Chris
Init.h	Chris, Andy, Masen, Kendrick
Init.c - initFileSystem	Chris, Andy, Masen
Init.c - initRootDirectory	Chris, Masen
FreeSpace.h	Chris
FreeSpace.c	Masen
PDF Milestone 1 Write-Up	Andy, Chris, Masen, Kendrick

* Contributors are ranked from left to right (left is most contributed)

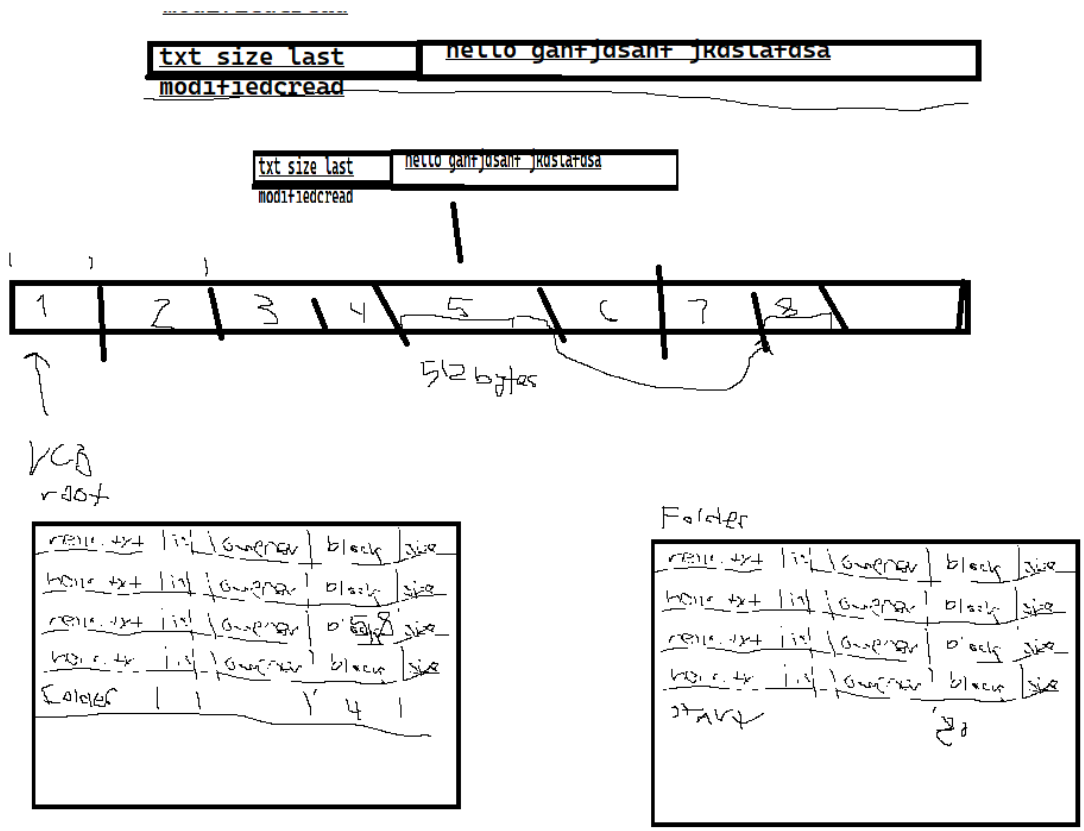
How did your team work together, how often you met, how did you meet, how did you divide up the tasks:

Our team is very active on this project. We have a solid team of dedicated workers in our group. We have had everyone meet one time in person, not counting the times before or in class. We have also all met in discord two times. Through these meetings we tend to work together on one machine, discussing and collaborating ideas, having one person write those thoughts down in code. The remaining work for the milestone ended up getting filled up by the members who took it upon themselves to complete it. Our group is focused, and the work that is required is being completed by whoever notices it needs to be done. This is how this group has successfully divided up the tasks.

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

The first issue we have faced as a group was building the necessary structures such as the Directory Entry and the Volume Control Block. The directory entry was pretty straightforward and relatively easy since we were able to get feedback on what we had and modify it accordingly. The Volume Control Block was a bit harder since we had very little knowledge of how a VCB comes into play in the entire File System. Two of our team members went into Office Hours in hopes of clearing the air up. Along with notes taken from class, we were able to eventually come up with a structure that we believe makes sense for our project.

The second issue we ran into started after the structures were built. As a group, we had very little understanding of what a File System even looked like. We had some conflicting ideas of how to approach the whole project. We had some saying it would be best to learn as we go and try to start coding. Others proposed that it would be a good idea to spend some time reading and learning about what a File System is. Ultimately we agreed that we should allocate some time to read and study up on what a File System is. This helped our group get some sense of direction and understanding of the task at hand. This helped us move forward with the implementation of our file system. Attached below is a small drawing one of our group members made to help the other members of the team understand block and memory allocation. The rectangle above was meant to signify the bytes of data, with the header at the start. The rectangle with numbered blocks was meant to signify the memory space that contains the blocks of data. The two squares below it were meant to signify the volume control blocks that would contain directory entries.



The third issue we ran into came after we started building the initFileStructure function in the fsInit.c file. We were able to gain some ground within the function, but we hit a wall very quickly when we realized that we had to allocate and initialize the Volume Control Block. Most of the fields were fairly straightforward, but when we had to start allocating the root directory along with the “.” and “..” entries, we had some troubles. Eventually our questions were answered by the next upcoming class we went to. Our professor ended up clearing the air on how the “.” and

“..” entries played a role. This led us to being able to build the initRootDirectory function for our fsInit.c file, making it successfully compile and run how it should.

The fourth issue we came across was involving extents. As a group we didn't have any prior knowledge on what it was or how to possibly implement it. During class on Monday, the professor went over examples to help get a better understanding of how to use it in this project and it helped out a lot. Instead of storing information about every single block, like we had before, we only needed to store the start block and the number of blocks that made up that extent, simplifying our space management.