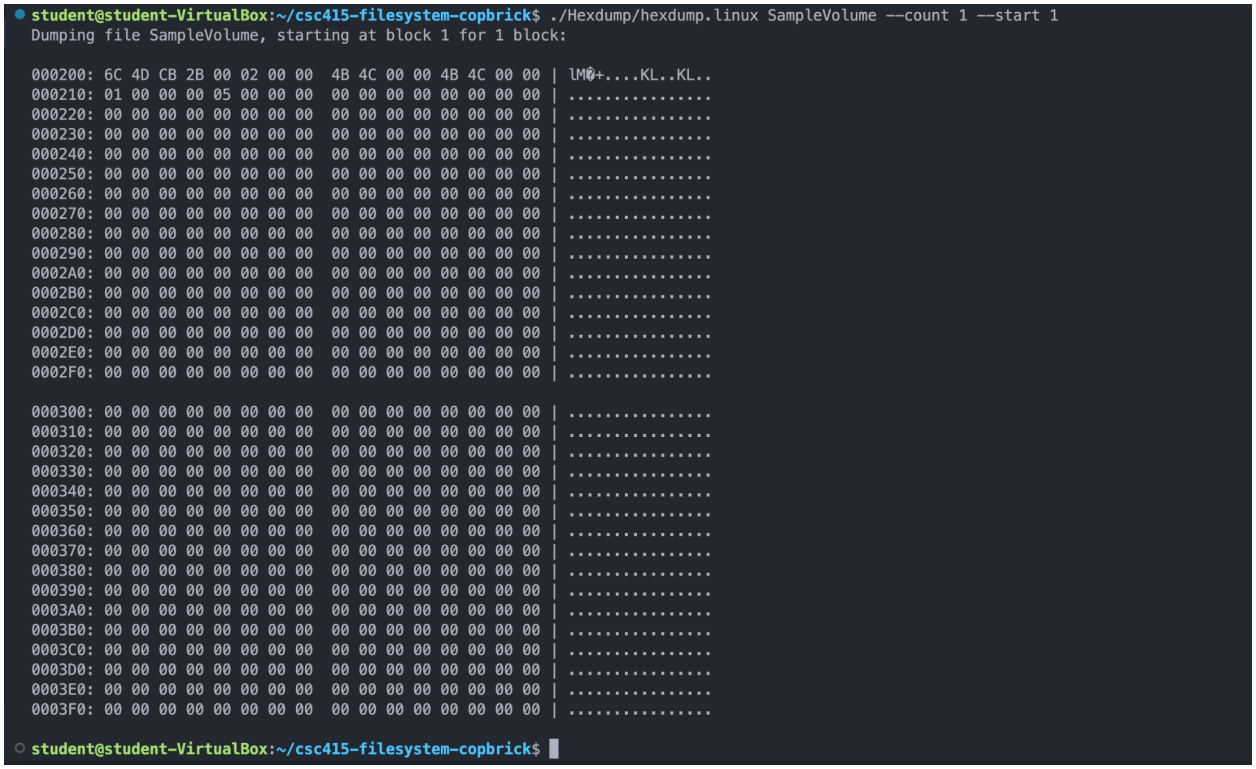
Team DALJ
Anthony Benjamin | 921119898
Nyan Ye Lin | 921572181
David Chen | 922894099
Joshua Hayes | 922379312
Github #: copbrick

1. **Hexdump**



2. **Description of VCB Structure:**

The VCB Structure contains all of important information about the volume, including our magic number, block size, etc. All of the members in the struct are integer values. We have our signature (magic number) that checks if the filesystem has been initalized. If the magic number of the file system doesn't equal what's stored in the VCB, then the fileSystem will initalize the VCB values. We also keep track of the blockSize, which in our case is 512 bytes. Also, we have a totalBlocks variable to keep track of the amount of blocks that the file system has, based on the size of the volume. There is a freeBlocks variable which keeps track of the amount of freeBlocks that the file system can allocate. We're also keeping track of what the starting block of the free space manager is. This is stored in a variable called "freeSpaceManagerBlock". Finally, we store the starting block of our root directory in our VCB.

```c
typedef struct VCB
{
    int signature;             // signature to know if filesystem
    int blockSize;             // size of each block
    int totalBlocks;           // total blocks available
    int freeBlocks;            // number of free blocks
    int freeSpaceManagerBlock; // starting block for free space m
    int rootDirBlock;          // starting block for root directo
} VCB;
```

3. **Description of the Free Space structure:**

The Free Space Structure contains information for tracking free disk space. To accomplish this, we create a “Free Space Manager”. This is a bitmap array, filled with 1’s (occupied) and 0’s (free), these are tied to which blocks are free/occupied in our disk. Given our default size, we need 2,442 bytes which fits in 5 blocks. We reserve 5 blocks of memory for the freeSpaceManager, then we fill the first 5 blocks with 1’s to indicate that they’re occupied, then we fill the remaining blocks with 0. After this, we write the freeSpaceManager into our disk, and store the starting block (1) into our VCB.

```
int initFreeSpaceManager(int totalBlocks, int blockSize)
{
    int freeSpaceManagerBlocks = totalBlocks / (8 * blockSize) +
    int *freeSpaceManager = malloc((freeSpaceManagerBlocks * blo
    for (int i = 0; i < freeSpaceManagerBlocks; i++)
    {
        freeSpaceManager[i] = 1;
    }
    for (int j = freeSpaceManagerBlocks; j <= freeSpaceManagerBl
    {
        freeSpaceManager[j] = 0;
    }
    LBAwrite(freeSpaceManager, freeSpaceManagerBlocks, 1);
    return freeSpaceManagerBlocks + 1;
}
```

4. **Description of the Directory system:**

The Directory system is used as an address to locate files in a file system. For our implemented directory structure, we have an integer id to hold the inode number of each directory entry, an integer to tell whether an entry is either a directory or a file, a integer starting location to mark the block where the block starts at (each file occupies a set of contiguous blocks), a char[] file name so that we can know what data each file will have, a char [] for the file author's name, and metadata such as epoch timestamps to know when the file was created, edited, and last accessed. The directory structure is 136 bytes and we initialize the root directory to start off with 52 entries which uses 14 blocks and wastes 96 bytes. During the init, we initialize the root directory by creating an array to store 52 directory entries and assign the first two entries to create the root directory.

```c
typedef struct directoryEntry
{
    // size of struct is 136 bytes
    int id;                // unique ID that is associated wit
    int isFile;            // boolean value that's assocaited
    int location;          // stores block location of file
    int fileSize;          // the file size in bytes
    time_t createDate;     // timestamp associated with when t
    time_t lastAccessDate; // timestamp associated when the fi
    time_t lastModifyDate; // timestamp associated when the las
    char fileName[64];     // string with max 64 characters tha
    char author[32];       // string with max 32 characters tha
} directoryEntry;
```

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

Anthony Benjamin	• Worked on the implementation of initFreeSpaceManager, and provided general approach behind free space allocation algorithm. • Worked on this document (writeup),and provided information of how our VCB, and freeSpaceManager work at a high-level. • Worked on the implementation of the findFreeBlocks function. We previously had a bug where the values in VCB, weren't being referenced correctly. It was an issue due to not reading from disk, I had fixed that. • Added comments throughout our codebase, explaining how our functional components work.
Nyan Ye Lin	• Worked on implementing findFreeBlocks function • Bug fixes
Joshua Hayes	• Contributed towards creating the initFileSystem and initRootDirectory implementation and documentation for the directory system.

David Chen	● Contributed towards making the initialization logic of the free space manager.

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

Our team worked together by meeting up multiple times per week, and having a shared google doc that we could use to plan out our solutions. We copied the Milestone 1 pdf, and color coded it based on what we had finished and what still needed to be worked on. This helped us a lot, as it served as our action item list. Whenever there was work that needed to be done, we could refer back to this document we created in order to see what still needed to be worked on. We also created a Discord server that was our main form of communication. We met through voice calls, and a lot of text communication going back and forth about our general approach. We focused on building out this milestone together, as it was foundational, we wanted to all understand the process of building this out together.

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

Our team faced multiple issues, mainly around the conceptual understanding of how the initialization of our file system would work. One of our problems was not understanding how the magic number works. We were wondering what the actual number should be. After speaking to some classmates, I found out that the the magic number is arbitrary. So long that it stays the same between compilation, we will maintain data persistence.

Another issue that we had run into was not understanding how the free space manager works. We had gone back to the lecture video and the professors video helped us out a ton. We realized that we just needed to allocate a array with 1’s and 0’s into memory, and these would indicate whether a specific block is free or occupied. We initialized the space that the free space manager took up, with 1’s in our bitmap array, then we free’d up the rest of the bitmap array.