Group Project Writeup (github: jeawanjang)

Group name: GeriatricCats

Name: Jeawan Jang, Paige Hodgkinson, Randale Reyes, Carlos Campos Lozano

ID: 923070860, 922282852, 921008696, 920768261

Github: jeawanjang, phodgkinson1, RandaleReyes, ccamposlozano

Whole working progress github link: https://github.com/phodgkinson1/GeriatricCats2
Final submission github link: https://github.com/CSC415-2023-Fall/csc415-filesystem-jeawanjang.git

Milestone 1:

- Volume Control Block (VCB structure)

Screenshot (regarding volume control block structure):

```
// struct VCB
typedef struct VCB
{
        long signature;         // Signature determines if formatted
        int totalNumBlocks;           //total blocks in entire volume including VCB
        int sizeOfBlock;        // bytes per block (512)
        int startFreeSpaceManagement;
        int startDirectory;

} VCB;
```

First of all, the Volume Control Block (VCB) is a fundamental structure in a file system that allows managing and maintaining crucial information about the entire file system. The VCB is located at the initial location of the volume so that serves as a reference indicator for various file system operations.

Variables Usage:

- Signature: is a long integer value which determines whether the file system has been initialized or not. In other words, it acts as an unique identifier.

- TotalNumBlocks: indicates the total number of blocks in the entire volume, including VCB itself. It defines the maximum capacity of the family system in terms of blocks, which can be derived through equation, volume / blockSize

- SizeOfBlock: specifies the size of each block in bytes. Particularly in this code, it is set to 512.
- StartFreeSpaceManagement: holds the block number where the free space management data structure commences.

- StartDirectory: manifests the block number where the root directory of the file system begins.

- Free space

Screenshot (regarding free space structure → Our choice: extent):

```
// struct extent
// Includes
// initFreeSpace() loadFreeSpace() allocateBlocks() releaseBlocks()
typedef struct extent
{
        int start;
        int count;
} extent, *pextent;
```

The extent structure manages the availability of contiguous free space in the system by using two attributes, start and count. This structure streamlines the allocation of free blocks, reducing fragmentation via allowing contiguous space for data storage.

Functions Usage:

```c
// InitFreeSpace is called when you initialize the volume
// returns the block number where the freespace map starts
int initFreeSpace (int blockCount, int bytesPerBlock)
        {

        // Calculate the size of the bitmap in bytes
        int bitmapSize = blockCount / 8;

        if (bitmapSize % BLOCK_SIZE > 0)
                {
                bitmapSize = BLOCK_SIZE * (bitmapSize / BLOCK_SIZE + 1);
                }
        bytesInBitmap = bitmapSize;
        int freeSpaceBlockCounts = bitmapSize / BLOCK_SIZE;

        // Allocate a memory of bitmap based on bitmapSize
        unsigned char * bitmap = malloc(bitmapSize);
        bitmapGlobal = bitmap;

        //initialize every bit on the bitmap to 0
        memset(bitmap, 0, bitmapSize);

        //marked the first 6 bits to used, block 0 for VCB + blocks 1-5 for the bitmap
        for (int i = 0; i < freeSpaceBlockCounts + 1; i++)
                {
                int byteIndex = i / 8; //Find the index of the byte (not user it should be *)
                int bitIndex =  i % 8; //Find index of bit inside the byte
                uint64_t mask= 1;
                bitmap[byteIndex] |= (mask << bitIndex); //set the bit to one
                }

        //write to the disk
        LBAwrite(bitmap, freeSpaceBlockCounts, 1);

        //return the block where the free space starts
        return 1;
}
```

- The initFreeSpace function's primary role is to set up and initialize the Free Space Map (FSM), a data structure responsible for tracking the availability of blocks in the file system. By calculating the size of the bitmap in bytes, with each bit in the bitmap representing the status of a block. To ensure the block size, bitmapSize is rounded up to the nearest multiple of the block size. It allocates memory for the bitmap and initializes all bits to free blocks. Besides, as noting certain blocks 0 for VCB and blocks 1 to 5 for FSM, it confirms that the initial reserved blocks are correctly identified.

- Directory

Screenshot (DE structure):

```c
// struct directory
typedef struct DE
{
    char fileName[32];
    int offset;                 //= i in array for both directory entries array and extents array
    int fileSize;
    time_t createdTime;
    time_t modifiedTime;
    time_t lastAccessedTime;
    char isDirectory;       //0 for file, 1 for directory
} DE;
```

The DE structure, standing for Directory Entry, represents a single entry in a file system directory. It is defined using the C programming language syntax. Here's a detailed description of each of its fields:

Variables Usage:

- fileName: A character array of size 32, which holds the name of the file or directory.
- offset: An integer that stores the position (index) of the directory entry in some array of directory entries. This offset also corresponds to the index in another array that stores the extents (blocks of storage) of the file or directory.
- fileSize: An integer that represents the size of the file in bytes. For directories, this size may have a different meaning, or could be set to zero or a fixed value.
- createdTime: time_t value that records the creation time of the file or directory.
- modifiedTime: time_t value that records the last modification time of the file or directory.
- lastAccessedTime: time_t value that records the last time the file or directory was accessed.
- isDirectory: A character used as a boolean flag to indicate whether the directory entry is a file or a directory. The value '0' represents a file, and '1' represents a directory.

This structure encapsulates the metadata of a file or directory in a file system, allowing for the management, retrieval, and manipulation of files and directories. The offset field suggests that this structure might be part of a larger file system implementation that involves arrays of entries and extents, facilitating file system navigation and file management operations.

Description Milestone 2:

Approach / What we did:

Milestone 1:

We wrote the foundational routine to initialize a file system, encompassing the initialization of the Volume Control Block (VCB), free space management, and the root directory. It started by announcing its operation, indicating the number of blocks and the block size of the file system. Two buffers are then allocated: one for the VCB and another for a directory entry (DE). Our group commenced by attempting to read the VCB from the first block of the file system using the LBAread function. A signature check follows, determining if the volume has already been initialized. If the signature matches, it indicates that the file system is already set up, and the function proceeds to load the existing free space management structure. If the signature does not match, indicating an uninitialized file system, the function transitions into the initialization phase.

It starts by initializing the VCB, setting up necessary metadata and parameters for the file system's operation. Subsequently, we initialized the free space management and the root directory. It printed out status messages and return values from these operations, providing insight into their potential issues. In the final part of the function, an additional directory initialization was attempted using the previously allocated DE buffer. Overall, we meticulously initialized and validated each component of the file system even though we did not separate each file.

Milestone 2:

First we had to finish the initExtent() for using extents as planned in our program, which we did not utilize in milestone 1 but would need now. We planned to separate all of our files properly into their own .c files and headers, as well as finishing releaseBlocks() from milestone 1. We set the parsePath and its helper functions. To write the basic algorithm as a model for how other functions would work, one of us coded and tested full mk_dir while testing the parsePath and the helper functions called by parsePath, and then finishing mk_dir in a nested work plan. After the parsePath and all of our mfsHelper functions were complete, the remaining directory operations open_dir()/read_dir()/close_dir() were done, with the stat and most remaining functions from mfs.c. The auxiliary functions from mfs.c utilized primarily in the shell commands were left until the end.

Issues / Resolutions

Milestone 1:

There were no issues in the programming of allocateBlocks() as programming was done according to deep planning and it was coded with systematic testing.
Issues in initDir() below

Issue 1:

We realized there was no if/else structure applied to the hard coded initialization of the first two entries, so we wrote a different routine for those *inside* the if/else condition for null/root creation or else for subdirectory. There was also no print checking of the parent passed in so that we could test for the else condition creating a subdirectory- this is why nothing printed when trying to call again for a subdirectory when passing in the new root. And last, we never checked for a condition where there was no space for the directory requested- instead of wrapping everything we had in an if statement, which would look messy, we just had an exit statement with proper closure of function resources. This implied moving our closure of the tempArray of extent structs into conditional checks, as we wanted to close it as soon as possible but only when necessary.

```
GNU nano 2.9.3                                                    fsInit.c

    dir[0].lastAccessedTime = t;
    dir[0].isDirectory = 1;

    struct DE *p;  // Declare p here to make it accessible in the if-else blocks

    if (parent == NULL)
    {
        // If there's no parent (root directory), set p to the root directory entry
        p = &dir[0];  // Removed the struct DE * declaration
    }
    else
    {
        // If there's a parent directory, set p to the provided parent
        p = parent;  // Removed the struct DE * declaration
    }

    // Initialize ".." entry for root
    strcpy(dir[1].fileName, "..");
    dir[1].fileSize = p->fileSize;
    dir[1].location = p->location;
    dir[1].createdTime = p->createdTime;
    dir[1].modifiedTime = p->modifiedTime;
    dir[1].lastAccessedTime = p->lastAccessedTime;
    dir[1].isDirectory = p->isDirectory;

    // Now write it to disk
    LBAwrite(dir, blocksNeeded, 0);

        free(tempArray);

        //save location of block number After filecontrolblock (directories entries) to set as extent location in each dir Entry
        //in each directory entry set offset to i

        //mark freespacemap extent returned by allocateBlocks to 1's
        //return start of directory location
        return 0;
/*
        // 0 -> used 1 -> unused
        for (int i = 2; i < actualDirEntries; i++)
        {
```

Resolution 1:

```
        dir[i].fileName[i]= '\0'; // Null-terminated name
        dir[i].fileSize = bytesNeeded;
        dir[i].offset= i;
        time_t t = time(NULL);
        dir[i].createdTime = t;
        dir[i].modifiedTime = t;
        dir[i].lastAccessedTime = t;
        dir[i].isDirectory = 0;

    // printf("filename: %c\n", dir[i].fileName[i]);
    }

    if (p == NULL)
    {
        // Initialize  . and .. manually

        // If there's no parent (root directory), set p to the root directory entry
        p = &dir[0];  // Removed the struct DE * declaration
        // Initialize "." entry
        strcpy(p->fileName, ".");
        p->fileSize = 0;
        p->offset = 0;
        time_t t = time(NULL);
        p->createdTime = t;
        p->modifiedTime = t;
        p->lastAccessedTime = t;
        p->isDirectory = 1;
```

Issue 2:

The second significant challenge encountered by our group pertains to the debugging of the file system implementation. Debugging in such a complex environment requires a meticulous approach to trace back issues, understand the flow of execution, and verify the data at various

points in the program. The intricacy of the system demands a detailed and granular insight into the program, which was tough without handling the proper debugging.

Resolution 2:

To effectively address this debugging challenge, our group has implemented a strategy of extensive printf statement insertion throughout the codebase. This approach enables inspection of variable values and data structures. We strategically place these printf statements to ensure that they encapsulate key areas of the code, such as function entry and exit points, critical conditional branches, and after significant operations. To further enhance this strategy, we maintain clarity and relevance in our debug messages, ensuring that they provide sufficient context and information to aid in troubleshooting. Through this structured and methodical approach to debugging, our group aims to swiftly identify, understand, and resolve issues within the file system somewhat.

```
    // Just for variable check
    printf("bitmapSize: %d\n", bitmapSize); // 2560
    printf("blockCount : %d\n", blockCount); // 19531
    printf("bytesPerBlock: %d\n", bytesPerBlock); // 512
    printf("FreeSpace block counts: %d\n", freeSpaceBlockCounts); // 5
```

```
            // end tempArray loop
/*
                    printf("bitmap printed, bits order low to high: ");
                    //printbitmap allocateblock marks FSM
                    for (int i = 0; i <= exitByte; i++)
                            {
                            printf("[");
                            for(int j=0; j < 8 ; j++)
                                    {
                                    printf("%d", !!((bitmapGlobal[i] >> j) & 0x01));
                                    }
                            printf("]");
                            }
                    printf("|end bitmap \n");
*/
```

```
// (2) Initialize free space management
int startFreeSpaceManagement = initFreeSpace(numberOfBlocks, blockSize);
printf("start FSM: %d\n",startFreeSpaceManagement);

// (3) Initialize root directory
int startDirectory = initDir(DEFAULT_ENTRIES, NULL);
printf("startDirectory: %d\n", startDirectory);

int returnCheck= LBAread(de, 1, 6);
printf("returnCheck: %d, de[0].fileName: %s\n", returnCheck, de[0].fileName);

int subDirReturn = initDir(DEFAULT_ENTRIES, *&de);
printf("subDirReturn: %d\n", subDirReturn);
```

Issue 3:

The current state of our file system implementation presents a significant organizational challenge, as the entirety of the code resides within a single file, fsInit.c. We aim to distribute the code across multiple files, specifically vcb.c, freespace.c, and directory.c, as well as those header files, to encapsulate distinct functionalities related to the Volume Control Block, free space management, and directory handling, respectively. However, this transition has been fraught with challenges, particularly in maintaining parameter consistency across the various functions and modules. Additionally, there may be hidden dependencies and implicit assumptions in the code that were manageable in a single file structure but become problematic when the code is separated across multiple files. These issues have led to functionality problems and unexpected behavior in the separated modules.

Resolution 3:

The files needed to be added to the list of .o dependencies in the makefile.

Screenshots (compilation) Milestone 1:

```
student@student-VirtualBox:~/Documents/csc415-filesystem-jeawanjang$ make
gcc -c -o fsshell.o fsshell.c -g -I.
gcc -c -o fsInit.o fsInit.c -g -I.
gcc -o fsshell fsshell.o fsInit.o fsLow.o -g -I. -lm -l readline -l pthread
student@student-VirtualBox:~/Documents/csc415-filesystem-jeawanjang$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Initializing File System with 19531 blocks with a block size of 512
signature: 123456789
SIGNATURE: 1234567890


 Entered condition to initialize volume
***** After Initializing VCB *****
signature: 1234567890
total num blocks: 19531
block size: 512
bitmapSize: 2560
blockCount : 19531
bytesPerBlock: 512
FreeSpace block counts: 5
start FSM: 1
p filename: (null)

INITDIR CALLED
byteNeeded: 4096, blocksNeeded: 8, actualDirEntries: 56
allocateBlocks() called
required: 8 blocks, minperextent: 8 blocks
extent made ->start: 6, ->count: 8
block: 6, tempByte: 0, tempBitIndex: 6, exitByte: 1, exitBit: 6
dir[0].fileName: .
dir[1].fileName: ..
startDirectory: 6
returnCheck: 1, de[0].fileName: .
p filename: .

INITDIR CALLED
byteNeeded: 4096, blocksNeeded: 8, actualDirEntries: 56
allocateBlocks() called
required: 8 blocks, minperextent: 8 blocks
extent made ->start: 14, ->count: 8
block: 14, tempByte: 1, tempBitIndex: 6, exitByte: 2, exitBit: 6
locaiton rootGlobal: block: 6
else root copy returnCheck: 1, rootDir[0].fileName: .
check subdir[0].fileName: .
dir[1].fileName: .. at block 14 subdirectory
subDirReturn: 14
|---------------------------------|
|------- 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 >
```

Milestone 1 *case 2 (when volume is initialized already):*

```
student@student-VirtualBox:~/Documents/csc415-filesystem-jeawanjang$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Initializing File System with 19531 blocks with a block size of 512
signature: 1234567890
SIGNATURE: 1234567890


Volume is already initialized
bitmapSize: 2560
blockCount : 19531
BLOCK_SIZE= bytesPerBlock: 512
FreeSpace block counts: 5
bitmap printed, bits order low to high: start FSM: 1
|---------------------------------|
|------- 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 >
```

Compilation at Milestone 2:

Execution at Milestone 2:

Analysis Hexdump:

- A dump (use the provided HexDump utility) of the volume file that shows the VCB, FreeSpace, and complete root directory.

  ☐ **VCB**

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

000200: D2 02 96 49 00 00 00 00  4B 4C 00 00 00 02 00 00 | ◆.◆I....KL......
000210: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000220: 00 00 00 00 00 00 00 00  AE 3A 3C 65 00 00 00 00 | ........◆:<e....
000230: AE 3A 3C 65 00 00 00 00  AE 3A 3C 65 00 00 00 00 | ◆:<e....◆:<e....
000240: 01 00 00 00 00 00 00 00  2E 2E 00 00 00 00 00 00 | ................
000250: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000260: 00 00 00 00 00 00 00 00  01 00 00 00 00 00 00 00 | ................
000270: AE 3A 3C 65 00 00 00 00  AE 3A 3C 65 00 00 00 00 | ◆:<e....◆:<e....
000280: AE 3A 3C 65 00 00 00 00  01 00 00 00 00 00 00 00 | ◆:<e............
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:~/Documents/GeriatricCats$
```

D2 02 96 49 00 00 00 00 (= 0x499602D2): These 8 bytes correspond to the long signature in the VCB structure. Based on the little endian format, D2 02 96 49 is 1234567890, which matches the signature that we've defined.

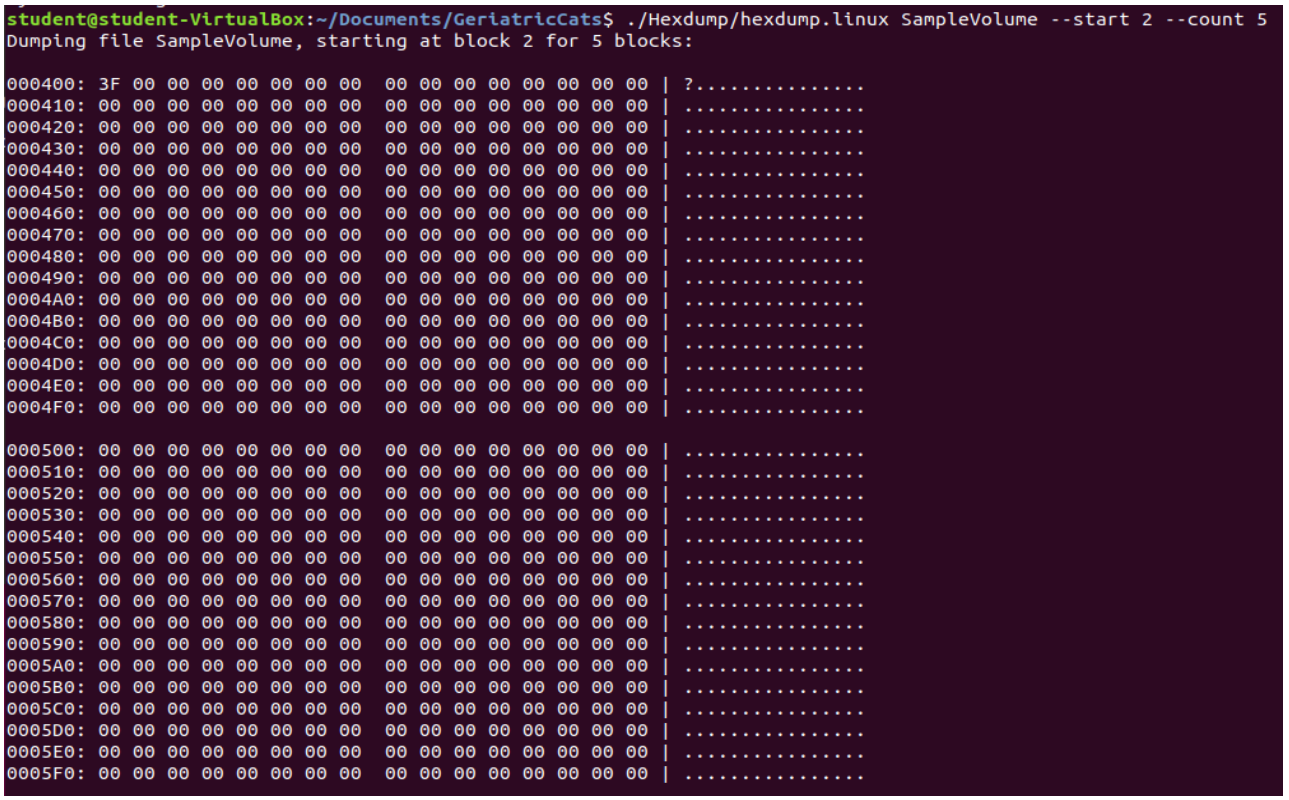
4B 4C 00 00 (= 0x00004C4B): It represents the totalNumBlocks, which is 19531. Since it is integer, it allocates 4 bytes.

00 02 00 00 (= 0x00000200): Just as totalNumBlocks, sizeOfBlock is 4 bytes and manifests the value of 512.

*****

StartFreeSpaceManagement, startDirectory: We got 0, which is not set, but I assume that since we initialize in initFileSystem at the end and it works well, we should get 1 for FSM and 6 for Dir.

- ☐ **FreeSpace**


student@student-VirtualBox:~/Documents/GeriatricCats$ ./Hexdump/hexdump.linux SampleVolume --start 2 --count 5
Dumping file SampleVolume, starting at block 2 for 5 blocks:
000400: 3F 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 | ....
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 | ....

```
000600: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000610: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000620: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000630: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000640: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000650: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000660: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000670: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000680: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000690: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000700: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000710: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000720: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000730: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000740: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000750: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000760: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000770: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000780: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000790: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

```
000800: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000810: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000820: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000830: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000840: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000850: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000860: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000870: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000880: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000890: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0008A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0008B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0008C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0008D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0008E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0008F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

```
000900: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000910: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000920: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000930: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000940: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000950: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000960: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000970: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000980: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000990: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0009A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0009B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0009C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0009D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0009E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0009F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000A00: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000A10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000A20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000A30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000A40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000A50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000A60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000A70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000A80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000A90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000AA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000AB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000AC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000AD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000AE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000AF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

```
000B00: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000B10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000B20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000B30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000B40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000B50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000B60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000B70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000B80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000B90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000BA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000BB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000BC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000BD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000BE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000BF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

```
000C00: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000C10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000C20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000C30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000C40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000C50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000C60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000C70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000C80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000C90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000CA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000CB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000CC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000CD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000CE0: 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  00 00 00 00 00 00 00 00 | ................
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:~/Documents/GeriatricCats$
```

Hexadecimal value *3F* at the beginning is equivalent to 00111111 and indicates that within a series of 8 blocks, 6 are used and the last two are free. Line numbers [410 to DFO] in decimal is [3568 to 1040] = 2544 bytes of 0 + 16 bytes with 3F marked, represents the expected 2560 bytes of the bitmap (5 blocks x 512 bytes), where the each 0 value represents 8 contiguous blocks on the disk that are free and available for allocation.

- ☐ **Root Directory**

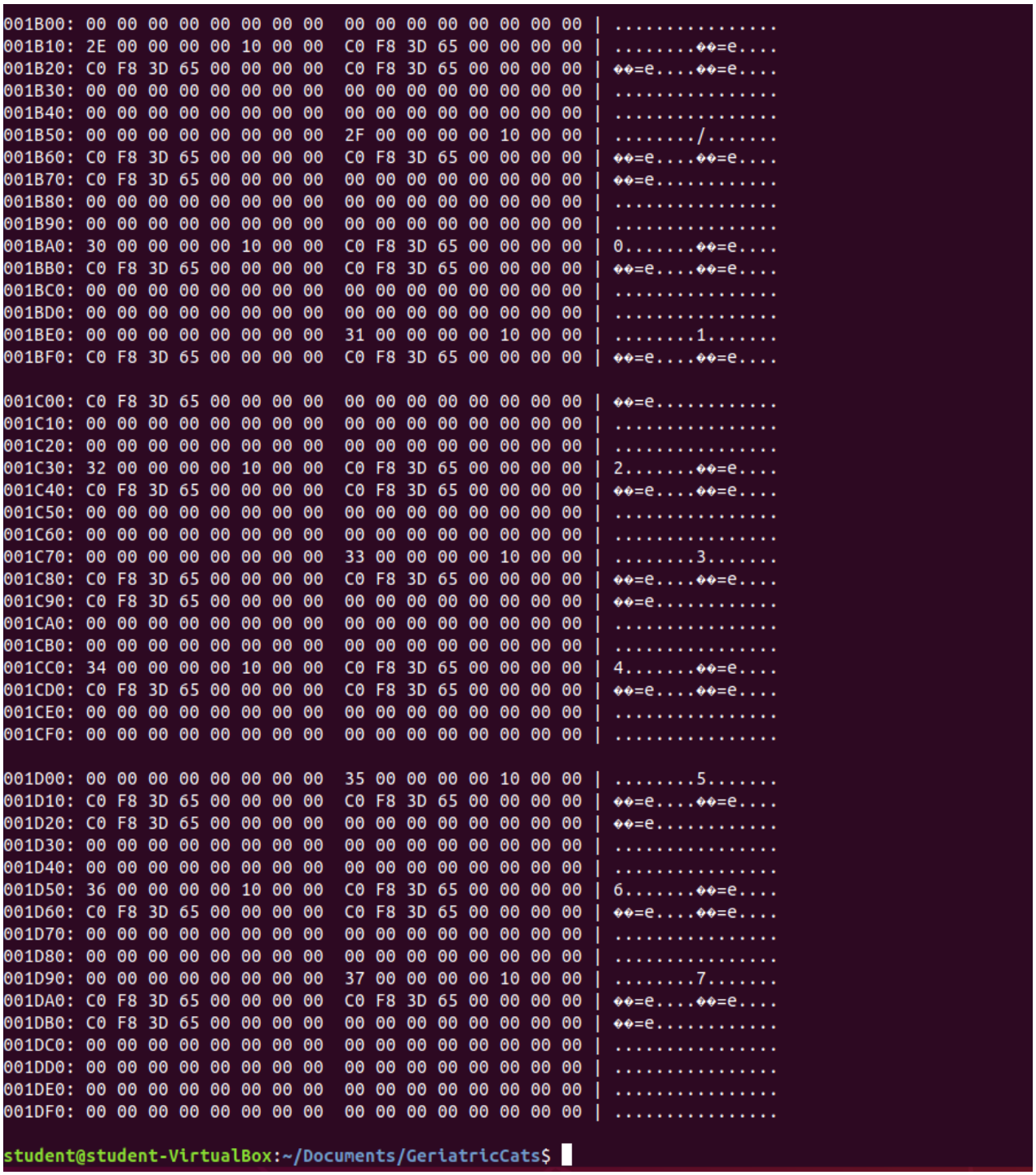
```
student@student-VirtualBox:~/Documents/GeriatricCats$ ./Hexdump/hexdump.linux SampleVolume --start 7 --count 8
Dumping file SampleVolume, starting at block 7 for 8 blocks:

000E00: 2E 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E20: 00 00 00 00 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ........◆◆=e....
000E30: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
000E40: 01 00 00 00 00 00 00 00  2E 2E 00 00 00 00 00 00 | ................
000E50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E60: 00 00 00 00 00 00 00 00  01 00 00 00 00 00 00 00 | ................
000E70: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
000E80: C0 F8 3D 65 00 00 00 00  01 00 00 00 00 00 00 00 | ◆◆=e............
000E90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000EA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000EB0: 02 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | ........◆◆=e....
000EC0: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
000ED0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000EE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000EF0: 00 00 00 00 00 00 00 00  03 00 00 00 00 10 00 00 | ................
```

```
000F00: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ▯▯=e....▯▯=e....
000F10: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ▯▯=e............
000F20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F40: 04 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | ........▯▯=e....
000F50: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ▯▯=e....▯▯=e....
000F60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F80: 00 00 00 00 00 00 00 00  05 00 00 00 00 10 00 00 | ................
000F90: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ▯▯=e....▯▯=e....
000FA0: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ▯▯=e............
000FB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000FC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000FD0: 06 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | ........▯▯=e....
000FE0: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ▯▯=e....▯▯=e....
000FF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

001000: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001010: 00 00 00 00 00 00 00 00  07 00 00 00 00 10 00 00 | ................
001020: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ▯▯=e....▯▯=e....
001030: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ▯▯=e............
001040: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001050: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001060: 08 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | ........▯▯=e....
001070: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ▯▯=e....▯▯=e....
001080: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001090: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0010A0: 00 00 00 00 00 00 00 00  09 00 00 00 00 10 00 00 | ................
0010B0: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ▯▯=e....▯▯=e....
0010C0: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ▯▯=e............
0010D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0010E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0010F0: 0A 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | ........▯▯=e....

001100: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ▯▯=e....▯▯=e....
001110: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001120: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001130: 00 00 00 00 00 00 00 00  0B 00 00 00 00 10 00 00 | ................
001140: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ▯▯=e....▯▯=e....
001150: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ▯▯=e............
001160: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001170: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001180: 0C 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | ........▯▯=e....
001190: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ▯▯=e....▯▯=e....
0011A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0011B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0011C0: 00 00 00 00 00 00 00 00  0D 00 00 00 00 10 00 00 | ................
0011D0: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ▯▯=e....▯▯=e....
0011E0: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ▯▯=e............
0011F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

```
001200: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001210: 0E 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | ........ø=e....
001220: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ø=e....ø=e....
001230: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001240: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001250: 00 00 00 00 00 00 00 00  0F 00 00 00 00 10 00 00 | ................
001260: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ø=e....ø=e....
001270: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ø=e............
001280: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001290: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0012A0: 10 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | ........ø=e....
0012B0: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ø=e....ø=e....
0012C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0012D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0012E0: 00 00 00 00 00 00 00 00  11 00 00 00 00 10 00 00 | ................
0012F0: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ø=e....ø=e....

001300: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ø=e............
001310: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001320: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001330: 12 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | ........ø=e....
001340: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ø=e....ø=e....
001350: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001360: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001370: 00 00 00 00 00 00 00 00  13 00 00 00 00 10 00 00 | ................
001380: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ø=e....ø=e....
001390: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ø=e............
0013A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0013B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0013C0: 14 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | ........ø=e....
0013D0: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ø=e....ø=e....
0013E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0013F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

001400: 00 00 00 00 00 00 00 00  15 00 00 00 00 10 00 00 | ................
001410: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ø=e....ø=e....
001420: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ø=e............
001430: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001440: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001450: 16 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | ........ø=e....
001460: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ø=e....ø=e....
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  17 00 00 00 00 10 00 00 | ................
0014A0: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ø=e....ø=e....
0014B0: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ø=e............
0014C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0014D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0014E0: 18 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | ........ø=e....
0014F0: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ø=e....ø=e....
```

```
001500: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001510: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001520: 00 00 00 00 00 00 00 00  19 00 00 00 00 10 00 00 | ................
001530: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
001540: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ◆◆=e............
001550: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001560: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001570: 1A 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | ........◆◆=e....
001580: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
001590: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0015A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0015B0: 00 00 00 00 00 00 00 00  1B 00 00 00 00 10 00 00 | ................
0015C0: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
0015D0: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ◆◆=e............
0015E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0015F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

001600: 1C 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | ........◆◆=e....
001610: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
001620: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001630: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001640: 00 00 00 00 00 00 00 00  1D 00 00 00 00 10 00 00 | ................
001650: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
001660: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ◆◆=e............
001670: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001680: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001690: 1E 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | ........◆◆=e....
0016A0: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
0016B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0016C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0016D0: 00 00 00 00 00 00 00 00  1F 00 00 00 00 10 00 00 | ................
0016E0: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
0016F0: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ◆◆=e............

001700: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001710: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001720: 20 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 |  .......◆◆=e....
001730: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
001740: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001750: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001760: 00 00 00 00 00 00 00 00  21 00 00 00 00 10 00 00 | ........!.......
001770: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
001780: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ◆◆=e............
001790: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0017A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0017B0: 22 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | ".......◆◆=e....
0017C0: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
0017D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0017E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0017F0: 00 00 00 00 00 00 00 00  23 00 00 00 00 10 00 00 | ........#.......
```

```
001800: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
001810: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ◆◆=e............
001820: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001830: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001840: 24 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | $.......◆◆=e....
001850: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
001860: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001870: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001880: 00 00 00 00 00 00 00 00  25 00 00 00 00 10 00 00 | ........%.......
001890: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
0018A0: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ◆◆=e............
0018B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0018C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0018D0: 26 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | &.......◆◆=e....
0018E0: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
0018F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

001900: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001910: 00 00 00 00 00 00 00 00  27 00 00 00 00 10 00 00 | ........'.......
001920: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
001930: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ◆◆=e............
001940: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001950: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001960: 28 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | (.......◆◆=e....
001970: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
001980: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001990: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0019A0: 00 00 00 00 00 00 00 00  29 00 00 00 00 10 00 00 | ........).......
0019B0: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
0019C0: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ◆◆=e............
0019D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0019E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0019F0: 2A 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | *.......◆◆=e....

001A00: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
001A10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001A20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001A30: 00 00 00 00 00 00 00 00  2B 00 00 00 00 10 00 00 | ........+.......
001A40: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
001A50: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ◆◆=e............
001A60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001A70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001A80: 2C 00 00 00 00 10 00 00  C0 F8 3D 65 00 00 00 00 | ,.......◆◆=e....
001A90: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
001AA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001AB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001AC0: 00 00 00 00 00 00 00 00  2D 00 00 00 00 10 00 00 | ........-.......
001AD0: C0 F8 3D 65 00 00 00 00  C0 F8 3D 65 00 00 00 00 | ◆◆=e....◆◆=e....
001AE0: C0 F8 3D 65 00 00 00 00  00 00 00 00 00 00 00 00 | ◆◆=e............
001AF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```



Based on where the free space map ends at 000DFO, the next line E00 represents the start of the file control block of directory entries. As we expect 8 blocks for the default entries amount at 512 bytes/block, the total bytes for the FCB is 4096 bytes. 4096 bytes/16 bytes per line is 256 lines. From line E00 hex to decimal 3584 bytes + 4096 bytes expected (or 256 lines) = 7664 bytes, or line 479 decimal/ 0001DF0 in hex.

directoryEntry structure {

```
    char fileName[32];
    int offset;          //= i in array for both directory entries array and extents array
    int fileSize;
    time_t createdTime;
    time_t modifiedTime;
    time_t lastAccessedTime;
    char isDirectory;    //0 for file, 1 for directory
}
```

Each directory entry has 65 bytes of members, but we allocate 72 bytes in our algorithm which determines the maximum fit of entries per block. We see that the name "." is 2E in hex, and the string name is 32 bytes or 2 lines at 000E00-000E10. The 4 bytes of int for the offset member is 0 for the root entry, so the first four bytes of 000E20 are 0. The 4 bytes of int for the fileSize member is 0 for the root entry, so the next 4 bytes of 000E20 are 0. Following the fileSize int, the last 8 bytes of the line of 000E20 is 65 3D F8 C0 00 00 00 00 for the time_t createdTime. This same value is repeated in the first 8 and last 8 bytes of the next line at 000E30, because we used the same value to store in modifiedTime and lastAccessedTime. At line 000E40, we have a 01 for the 1 byte char isDirectory, which is marked 1 in our program. The following 6 bytes are waste bytes from our algorithm. The exact same pattern for the structure is repeated at byte 8 in line 000E40, but with "2E 2E" of the 32 byte string because our second entry is "..". The offset at the appropriate position after a 32 byte string is 01, because the second directory entry will have an offset of 1 in our extent table array block. We can easily observe the offset of each entry increasing by 1 every 72 bytes or after 9 sections of 8 bytes, showing the directoryEntries as they index through the blocks remaining.

Work Table:

Team Work- Based on Github history, Google Doc History, and Meeting Hrs in SFSU portal

Milestone 1

We met once/week for the first three weeks on Mondays, then 2x/week for the following week on Monday/Wednesday and then 3x the last week on Mon/Wed/Fri. Meetings were scheduled in-person in the library reserve study rooms, with a laptop hooked up to the tv for in person, and using the Discord screen share and audio/video lounge for whoever could not attend in person. We did not assign sections of work itself ahead of the entire plan, but maintained a plan and timeline of work progress to meet the deadline for Sunday, which is the exact time it took from our start to finish. Each meeting, someone would take the assignment of a section or portion of a section and complete it by the next meeting, and if they could not complete it, someone else would take the remaining work the morning before the meeting or the work would be done during the meeting to stay on track.

Milestone 1
Member
Carlos- Work planned ratio 0.25%, Actual Work ratio 0.18%, time spent 12 hrs, complete 72%
Jaewan- Work planned ratio 0.25%, Actual Work ratio 0.35%, time spent 18 hrs, complete 100%
Paige- Work planned ratio 0.25%, Actual Work ratio 0.42%, time spent 18 hrs, complete 100%
Randale- Work planned ratio 0.25%, Actual Work ratio 0.05%, time spent 3 hrs, complete 0.20%
Overall work hours: 51

// MileStone 1 Done

Milestone 2

Change:

- We separate each .c and .h files properly to easily handle with, particularly when debugging (i.e vcb.h, vcb.c freespace.c …)

- We add more useful helper functions to track the path, update extent/dir and remove the directory

- Write a code to work these functions properly as we intended in knowledge

    - fs_setcwd
    - fs_getcwd
    - fs_isFile
    - fs_isDir
    - fs_mkdir
    - fs_opendir
    - fs_readdir
    - fs_closedir
    - fs_stat
    - fs_delete
    - fs_rmdir

Helper functions:

```c
#ifndef MFSHELPER_H
#define MFSHELPER_H
#include "freespace.h"
#include "directoryEntry.h"

// Global variables
 extern DE * rootDir;
 extern DE * cwd;
 extern char *cwdAbsolutePath;

typedef struct parsePathInfo
{
    DE *parent;
    char *lastElement;
    int indexOfLastElement;

} parsePathInfo;

int writeExtent(DE * dir, EXTTABLE * ext);
int writeDir(DE * dir, int location);
int parsePath(char * path, parsePathInfo * ppi);
char * pathUpdate(const char * pathname);
int FindEntryInDir(DE * dir, char * fileName);
int isDirectory(DE * entry);
int isDirEmpty(DE *dir);
void markDirUnused(DE *dir);


EXTTABLE * loadExtent(DE * dir);
DE * loadDir(DE * dir, int index);


#endif // MFSHELPER_H
```

1. **parsePath(char *path, parsePathInfo *ppi)**: analyzes a given file path and updates the parsePathInfo structure with information about the path. It checks if the path is absolute or relative, determines the parent directory, and identifies the last element in the path. It handles special cases, such as the root directory.
2. **pathUpdate(const char *pathname)**: updates a given path to include the current working directory if the path is not absolute. It then returns the updated path.
3. **FindEntryInDir(DE *dir, char *fileName)**: searches for a file or directory named fileName within the specified directory dir. It returns the index of the directory entry if found, or -1 if not found.
4. **isDirectory(DE *dir)**: checks if the given directory entry dir is a directory or not. It returns 1 if it's a directory, 0 otherwise.
5. **isDirEmpty(DE *dir)**: determines if the specified directory dir is empty. It returns 0 if the directory is empty, 1 if it contains any files or subdirectories.
6. **markDirUnused(DE *dir)**: marks the given directory entry dir as unused. This involves setting various fields in the directory entry to represent an unused state.
7. **loadExtent(DE *dir)**: loads the extent table for a given directory dir. This table is used to retrieve the locations of directory entry items. The function returns a pointer to the loaded extent table.
8. **writeExtent(DE *dir, EXTTABLE *ext)**: writes the extent table ext for the specified directory dir. This is used to update the filesystem with changes to the extent table.
9. **writeDir(DE *dir, int location)**: writes the directory dir to a specified location on the disk. It's used for updating or saving changes made to a directory.

10. **loadDir(DE *dir, int index)**: loads a directory from the disk into memory based on a given index, which is essential for reading directories from the storage into memory for manipulation or navigation.

Elaborate Function by Function (mfs.c):

1. fs_mkdir
   - *Code*

```
int fs_mkdir(const char *pathname, mode_t mode)
{
    // update pathname
    printf("called fs_mkdr with pathname : |%s|\n", pathname);
    if (pathname == NULL)
        return -1;
    // ******* add condition here to check if cdw==rootDir if no absolute
    // update pathname with new element
    char *newPath = pathUpdate(pathname);

    parsePathInfo *ppiTest = malloc(sizeof(parsePathInfo));
    // ppiTest->lastElement[0]= 'E';test for structure passing
    // parsePath returns 0 if valid path for a directory, -2 if invalid
    int pathValidity = parsePath(newPath, ppiTest);
    //  printf("parsePath returned : %d\n", pathValidity);
    if (newPath != NULL)
        free(newPath);

    if (pathValidity != 0)
    {
        printf("Invalid path!\n");
        return -1;
    }
    if (ppiTest->indexOfLastElement != -1)
    {
        printf("already a directory with same name.\n");
        return -1;
    }
    // printf("ppi members- parent->filesize: %d, lastElement[0]: %s, indexOfLastElement: %d\n",
    // ppiTest->parent->fileSize, ppiTest->lastElement, ppiTest->indexOfLastElement);
    char *empty = malloc(1);
    empty[0] = '\0';
    int nextAvailable = FindEntryInDir(ppiTest->parent, empty);
    if (nextAvailable == -1)
    {
        printf("Directory at capacity.\n");
        return -1;
    }

    // load parent
    int startBlockNewDir = initDir(DEFAULT_ENTRIES, ppiTest->parent, nextAvailable);
    printf("in mkdir startBlockNewDir: %d \n", startBlockNewDir);

    // load parent extent block
    EXTTABLE *ext = loadExtent(ppiTest->parent);

    // set parent's extent with start of new dir
    ext[nextAvailable].tableArray[0].start = startBlockNewDir;
    printf("ROOT PARENT ext[%d].tableArray[0].start: %d\n", nextAvailable, ext[nextAvailable].tableArray[0].start);
```

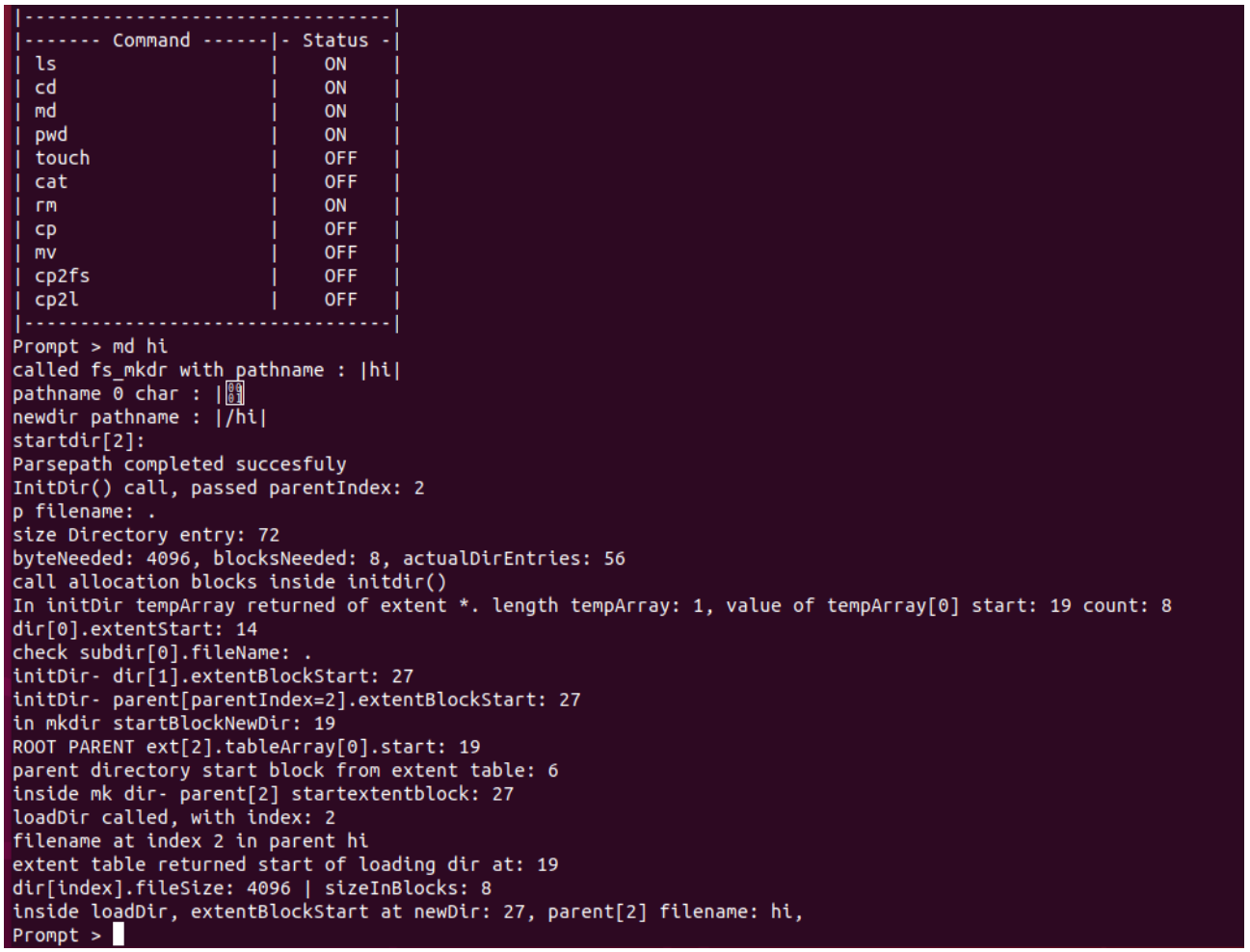
```
        return -1;
    }
    if (ppiTest->indexOfLastElement != -1)
    {
        printf("already a directory with same name.\n");
        return -1;
    }
    // printf("ppi members- parent->filesize: %d, lastElement[0]: %s, indexOfLastElement: %d\n",
    // ppiTest->parent->fileSize, ppiTest->lastElement, ppiTest->indexOfLastElement);
    char *empty = malloc(1);
    empty[0] = '\0';
    int nextAvailable = FindEntryInDir(ppiTest->parent, empty);
    if (nextAvailable == -1)
    {
        printf("Directory at capacity.\n");
        return -1;
    }

    // load parent
    int startBlockNewDir = initDir(DEFAULT_ENTRIES, ppiTest->parent, nextAvailable);
    printf("in mkdir startBlockNewDir: %d \n", startBlockNewDir);

    // load parent extent block
    EXTTABLE *ext = loadExtent(ppiTest->parent);

    // set parent's extent with start of new dir
    ext[nextAvailable].tableArray[0].start = startBlockNewDir;
    printf("ROOT PARENT ext[%d].tableArray[0].start: %d\n", nextAvailable, ext[nextAvailable].tableArray[0].start);
    int parentDirStart = ext[1].tableArray[0].start;
    printf("parent directory start block from extent table: %d\n", ext[1].tableArray[0].start);
    // write parent's extent
    writeExtent(ppiTest->parent, ext);
    if (ext != NULL)
        free(ext);

    // update directory entry name for new directory
    char *copy = ppiTest->lastElement;
    int i = 0;
    while (copy[i])
    {
        ppiTest->parent[nextAvailable].fileName[i] = copy[i];
        i++;
    }

    copy = NULL;

    //    printf("new filename at  ppiTest->parent[nextAvailable].fileName: |%s|\n",
    //           ppiTest->parent[nextAvailable].fileName);

    printf("inside mk dir- parent[2] startextentblock: %d\n", ppiTest->parent[nextAvailable].extentBlockStart);

    writeDir(ppiTest->parent, parentDirStart);

    writeDir(loadDir(ppiTest->parent, nextAvailable), startBlockNewDir);
```

```
    if (ppiTest->parent[1].extentBlockStart == cwd[1].extentBlockStart)
    {
        cwd = ppiTest->parent;
    }

    // cleanup
    if (ppiTest != NULL)
    {
        if(ppiTest->parent != rootDir && ppiTest->parent != cwd)
                {
                free(ppiTest->parent);
                }
        free(ppiTest);
        ppiTest = NULL;
    }

    return 1;
}
```

- *Description*

fs_mkdir: is designed for creating new directories within a file system, handling several while validating the provided path, updating it with the current working directory if necessary, and checking for the existence of a directory with the same name. It leverages a series of internal functions to find an available entry in the parent directory, initializes the new directory, and updates the parent directory's extent table to include the new directory. Key steps include checking the path's validity, locating a suitable location for the new directory, setting up the directory's structure, and updating relevant file system metadata.

- *Test*



2. fs_rmdir && fs_delete
   - *Code*

```
// Remove empty directory ***** (DONE)
int fs_rmdir(const char *pathname)
{
    // ---------------------------------------------------------------------------
    printf("fs_rmdir starts: \n");
    printf("called fs_mkdr with pathname : |%s|\n", pathname);

    if (pathname == NULL)
        return -1;

    // update pathname with new element
      char *newPath = pathUpdate(pathname);

    // ParsePath
    parsePathInfo *ppi = malloc(sizeof(parsePathInfo));

    int parsePathResult = parsePath(newPath, ppi);
    printf("parsePathResult : %d \n", parsePathResult);
    if (newPath != NULL)
        free(newPath);

    if (parsePathResult != 0)
        return -1; // ParsePath check done(o)

    // find index
    int index = FindEntryInDir(ppi->parent, ppi->lastElement);
    printf("index : %d \n", index);

    if (index == -1)
        return -1; // find index check done(o)

    // check fs_isDir is 1 --> dir (must be return dir)
    int checkDir = fs_isDir((char *)pathname);
    printf("fs_isDir result: %d \n", checkDir);

    if (checkDir != 1)
        return -1; // check fs_isDir check done(o)

    // load the directory that targets to be removed
    DE *dirRemove = &ppi->parent[index];
```

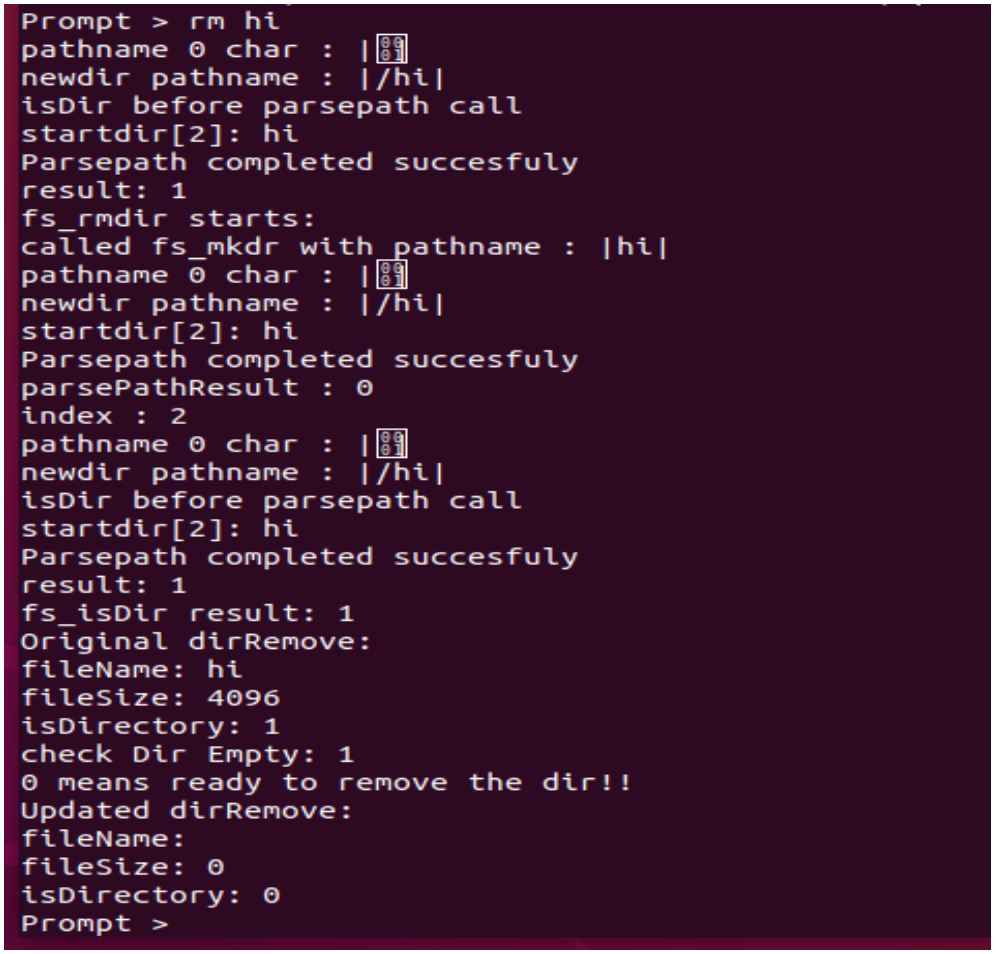
```
    printf("Original dirRemove: \n");
    printf("fileName: %s \n", dirRemove->fileName);
    printf("fileSize: %d \n", dirRemove->fileSize);
    printf("isDirectory: %d \n", dirRemove->isDirectory);

    // by iterating through the entries, check its empty condition
    // checkDirEmpty == 0 -----> target dir is empty ----> ready to remove
    int checkDirEmpty = isDirEmpty(dirRemove);

    printf("check Dir Empty: %d \n", checkDirEmpty);
    printf("0 means ready to remove the dir!!\n");

    // Release the blokcs associated with dirRemove
    EXTTABLE *extTable = loadExtent(dirRemove);

    for (int i = 0; i < 5; i++)
    {
        if (extTable[index].tableArray[i].start > 0)
        {
            releaseBlocks(extTable[index].tableArray[i].start, extTable[index].tableArray[i].count);
        }
    }

    markDirUnused(dirRemove);

    writeDir(ppi->parent, index);

    printf("Updated dirRemove: \n");
    printf("fileName: %s \n", dirRemove->fileName);
    printf("fileSize: %d \n", dirRemove->fileSize);
    printf("isDirectory: %d \n", dirRemove->isDirectory);

        if(ppi->parent != rootDir && ppi->parent != cwd)
                {
                free(ppi->parent);
                }

    if (ppi != NULL)
        {
        free(ppi);
        ppi=NULL;
        }

  return 0;
}
```

- *Description*

fs_rmdir: crafted for removing directories from a filesystem. Initially, it validates the given pathname and employs parsePath to analyze the path structure. Validations include confirming the existence of the directory at the specified path and ensuring that it is indeed a directory, not a file. Subsequently, check if the target directory is empty, a prerequisite for removal. Once these checks are passed, it proceeds to release any blocks associated with the directory and marks the directory entry as unused using markDirUnused. Finally, it updates the parent directory's information to reflect this removal and handles memory cleanup thoroughly.

fs_delete: is literally the same concept as fs_rmdir. The only difference between fs_rmdir and fs_delete is checking its parameter's type whether it is file or directory.

- *Test*



3. fs_closerdir

-Code

```c
int fs_closedir(fdDir *dirPath)
{
    // Close the directory specified by dirPath
    releaseBlocks(dirPath->directory->extentBlockStart, dirPath->directory->extentIndex);

    // Free resources allocated for the dirPath structure
    free(dirPath);
    return 0;
}
```

- *Description*

fs_closedir: closes a directory and frees the associated resources in a file system. It starts by releasing the blocks associated with the directory, specifically targeting the starting block and extent index of the directory, as indicated by dirPath→directory→extentBlockStart and dirPath→directory→extentIndex. After releasing these blocks, the function proceeds to free the memory allocated for the dirPath structure, which holds information about the directory being closed.

4. fs_getcwd && fs_setcwd
    - *Code*

```c
char *fs_getcwd(char *pathname, size_t size)
{
    printf("fs_getcwd function called\n");
    if (size <= 0 || pathname == NULL)
    {
        return NULL;
    }

    // to check
    // long cwd = 4096;

    if (strlen(cwdAbsolutePath) > size - 1)
    {
        printf("buffer size is not fit into cwdAbsolutePath");
        return NULL;
    }

    printf("cwdAbsolutePath's length: %ld \n", strlen(cwdAbsolutePath));
    printf("Limitation of buffer size: %ld \n", size - 1);

    strncpy(pathname, cwdAbsolutePath, size);

    pathname[size - 1] = '\0';

    printf("fs_getcwd: %s \n", pathname);

    return pathname;
}
```

```
int fs_setcwd(char *pathname)
{
        if(rootDir==NULL) loadDir(rootDir, rootGlobal);
    printf("fs_setcwd starts with cwdAbsolutePath: %s\n", cwdAbsolutePath);

    printf("setcwd start root[0]:|%s| filesize: %d _____ root[1]: |%s| filesize: %d  root[2]: |%s| filesize: %d\n", rootDir[0].file$
           rootDir[0].fileSize, rootDir[1].fileName, rootDir[1].fileSize, rootDir[2].fileName, rootDir[2].fileSize);

    printf("setcwd start cwd[0]:|%s| filesize: %d _____ cwd[1]: |%s| filesize: %d  cwd[2]: |%s| filesize: %d\n", cwd[0].fileName,
           cwd[0].fileSize, cwd[1].fileName, cwd[1].fileSize, cwd[2].fileName, cwd[2].fileSize);

    char *pathComponents[32];
    int componentCount = 0;
    char *newPath = malloc(256);

    // ParsePath
    parsePathInfo *ppi = malloc(sizeof(parsePathInfo));

    if (pathname[0] == '/')
        strcat(newPath, "/");

    // first tokenize path components to array and parse '.' and '..'
    char *saveptr = NULL;
    char *token = strtok_r(pathname, " /", &saveptr);

    if (token != NULL)
    {
        pathComponents[componentCount] = strdup(token);
        printf("token %d: |%s| pathComponents[%d]: |%s|\n", componentCount, token, componentCount, pathComponents[componentCount]);
        componentCount++;
    }
        if (strcmp(token, "..") == 0)
                {
                        //get second to last item from parsePath
                        int parsePathResult0 = parsePath(cwdAbsolutePath, ppi);
                        if(parsePathResult0 == 0)
                                {
                                        printf("parsePathResult set to parent");
                                        cwd=ppi->parent;
                                         printf("setcwd after cd . root[0]:|%s| filesize: %d _____ root[1]: |%s| filesize: %d  root$
                                        rootDir[0].fileSize, rootDir[1].fileName, rootDir[1].fileSize, rootDir[2].fileName, rootDir$

                                        printf("setcwd after cd . cwd[0]:|%s| filesize: %d _____ cwd[1]: |%s| filesize: %d  cwd[2]:$
                                        cwd[0].fileSize, cwd[1].fileName, cwd[1].fileSize, cwd[2].fileName, cwd[2].fileSize);

                                        printf("cwdAbsolutePath before cd .. : |%s|\n", cwdAbsolutePath);
                                        printf("strlen cwdABspath: |%ld|\n", strlen(cwdAbsolutePath));
                                        int i= strlen(cwdAbsolutePath)-1;
                                        while(cwdAbsolutePath[i]!='/'){
                                        i--;
```

```
    printf("index : %d \n", index);

    if (index == -1)
    {
        printf("find entry returned not found\n");
        return -1; // find index check done(o)
    }

    // Check if the target is a directory
    if (isDirectory(&ppi->parent[ppi->indexOfLastElement]) == 0)
    {
        printf("%s is not a directory \n", ppi->lastElement);
        return -1; // Target is not a directory
    }

    if (cwdAbsolutePath != NULL)
        free(cwdAbsolutePath);
    cwdAbsolutePath = newPath;
    printf("stored in cwdAbsolutePath: |%s|\n", cwdAbsolutePath);

    // set new cwd, cannot free old
        if(cwd != rootDir) free(cwd);
    cwd = loadDir(ppi->parent, index);

    printf("2 setcwd root[0]:|%s| filesize: %d _____ root[1]: |%s| filesize: %d  root[2]: |%s| filesize: %d\n", rootDir[0].fileName,
           rootDir[0].fileSize, rootDir[1].fileName, rootDir[1].fileSize, rootDir[2].fileName, rootDir[2].fileSize);

    printf("2 setcwd cwd[0]:|%s| filesize: %d _____ cwd[1]: |%s| filesize: %d  cwd[2]: |%s| filesize: %d\n", cwd[0].fileName,
           cwd[0].fileSize, cwd[1].fileName, cwd[1].fileSize, cwd[2].fileName, cwd[2].fileSize);

    // cleanup
        if(ppi->parent != rootDir && ppi->parent != cwd)
                {
                free(ppi->parent);
                }

    if (ppi != NULL)
        {
        free(ppi);
        ppi=NULL;
        }

    return 0; // Success
}
```

- *Description*

fs_getcwd: simply returns the pathname, in other words, cwdAbosolutePath, if it fits into buffer size.

fs_setcwd: is responsible for changing the current working directory in a file system to a specified path. It starts by loading the root directory if it's not already loaded and then tokenizes the provided pathname, handling special cases such as "." and ".." to navigate the file system. It updates cwdAbsolutePath with the new path and loads the new current working directory into memory.

- *Test*

```
Prompt > md hi
called fs_mkdr with pathname : |hi|
pathname 0 char : |
newdir pathname : |/hi|
startdir[2]:
Parsepath completed succesfuly
InitDir() call, passed parentIndex: 2
p filename: .
size Directory entry: 72
byteNeeded: 4096, blocksNeeded: 8, actualDirEntries: 56
call allocation blocks inside initdir()
In initDir tempArray returned of extent *. length tempArray: 1, value of tempArray[0] start: 19 count: 8
dir[0].extentStart: 14
check subdir[0].fileName: .
initDir- dir[1].extentBlockStart: 27
initDir- parent[parentIndex=2].extentBlockStart: 27
in mkdir startBlockNewDir: 19
ROOT PARENT ext[2].tableArray[0].start: 19
parent directory start block from extent table: 6
inside mk dir- parent[2] startexteentblock: 27
loadDir called, with index: 2
filename at index 2 in parent hi
extent table returned start of loading dir at: 19
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 27, parent[2] filename: hi,
Prompt > cd hi
fs_setcwd starts with cwdAbsolutePath: /
setcwd start root[0]:|.| filesize: 4096 _____ root[1]: |..| filesize: 4096  root[2]: |hi| filesize: 4096
setcwd start cwd[0]:|.| filesize: 4096 _____ cwd[1]: |..| filesize: 4096  cwd[2]: |hi| filesize: 4096
token 0: |hi| pathComponents[0]: |hi|
value of cwdAbsolutepath: |/|
startdir[2]: hi
Parsepath completed succesfuly
index : 2
stored in cwdAbsolutePath: |/hi|
loadDir called, with index: 2
filename at index 2 in parent hi
extent table returned start of loading dir at: 19
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 27, parent[2] filename: hi,
2 setcwd root[0]:|.| filesize: 4096 _____ root[1]: |..| filesize: 4096  root[2]: |hi| filesize: 4096
2 setcwd cwd[0]:|.| filesize: 4096 _____ cwd[1]: |..| filesize: 4096  cwd[2]: || filesize: 0
Prompt > pwd hi
fs_getcwd function called
cwdAbsolutePath's length: 3
Limitation of buffer size: 4095
fs_getcwd: /hi
/hi
Prompt > exit
System exiting
```

5. fs_isDir && fs_isFile

- *Code*

```
int fs_isFile(char *filename)
{
    if (filename == NULL)
    {
        printf("invalid file name\n");
        return -1;
    }

    // update pathname with new element
    char *newPath = pathUpdate(filename);

    parsePathInfo *ppi;
    ppi = malloc(sizeof(parsePathInfo)); // Allocate and initialize ppi

    if (parsePath(filename, ppi) != 0)
        return 0;

    if (newPath != NULL)
        free(newPath);

    int index = FindEntryInDir(ppi->parent, ppi->lastElement);
    if (index == -1)
    {
        // Entry not found, assuming not a file
        return 0;
    }

    DE *dirEntry = &(ppi->parent[index]);
    int result = dirEntry->isDirectory == 0; // Returns 1 if file (isDirectory == 0), 0 otherwise

        if(ppi->parent != rootDir && ppi->parent != cwd)
                {
                free(ppi->parent);
                }

    if (ppi != NULL)
        {
        free(ppi);
        ppi=NULL;
        }

    return result;
}
```

```
int fs_isDir(char *pathname)
{
    if (pathname == NULL)
    {
        printf("invalid pathname\n");
        return -1;
    }

    // update pathname with new element
    char *newPath = pathUpdate(pathname);

    parsePathInfo *ppi;
    ppi = malloc(sizeof(parsePathInfo)); // Allocate and initialize ppi

    printf("isDir before parsepath call\n");

    if (parsePath(pathname, ppi) != 0)
    {
        // Parsing failed, assuming not a directory
        return 0;
    }
    if (newPath != NULL)
        free(newPath);

    int index = FindEntryInDir(ppi->parent, ppi->lastElement);
    if (index == -1)
    {
        // Entry not found, assuming not a directory
        return 0;
    }

    DE *dirEntry = &(ppi->parent[index]);
    int result = dirEntry->isDirectory == 1; // Returns 1 if directory (isDirectory == 1), 0 otherwise

    printf("result: %d \n", result);
    //    printf("Dir removed\n");

if(ppi->parent != rootDir && ppi->parent != cwd)
                {
                free(ppi->parent);
                }

    if (ppi != NULL)
        {
        free(ppi);
        ppi=NULL;
        }
```

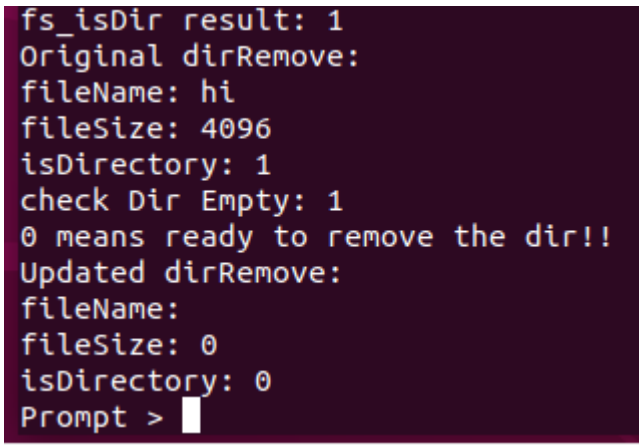
- *Description*

fs_isFile: determines whether a given path refers to a file. It initially employs parsePath through the directory structure and locates the specified file, utilizing FindEntryInDir to find the specific directory entry. (isDirectory == 0) represents that given parameter is a file.

fs_isDir: ,similar to fs_isFile, commences by checking the validity of the provided pathname and then updates it accordingly. The FindEntryInDir call searches for the pathname in the parent directory. It assists to determine whether it is a directory using the isDirectory attribute, where (isDirectory == 1) signifies a directory.

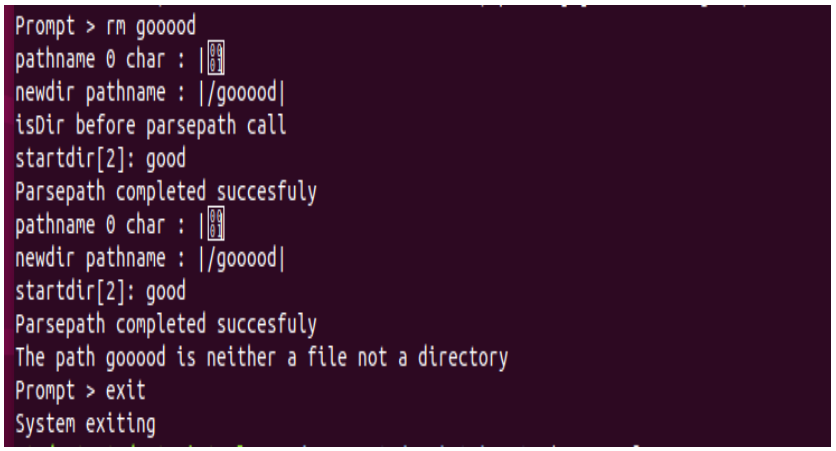
→ Both functions are automatically called when the fs_rmdir || fs_delete executes since both are associated with the removal of either file or directory.

- *Test*



So far, we can only create a directory, it is feasible to check fs_rmdir and fs_isDir.

- *Additional Test (when provided pathname is not created)*



6. fs_stat

-Code

```
int fs_stat(const char *path, struct fs_stat *buf)
{
    printf("fs_stat function called with path: %s\n", path);
    if (path == NULL || buf == NULL)
    {
        return -1; // Invalid input
    }

    // Parse the path to find the file or directory
    parsePathInfo *ppi = malloc(sizeof(parsePathInfo));
    int parsePathResult = parsePath((char *)path, ppi);

    if (parsePathResult != 0)
    {
        printf("PARSE PATH FAILED");
        return -1; // Parsing failed
    }

    // Check if the file or directory specified by 'path' exists
    int index = FindEntryInDir(ppi->parent, ppi->lastElement);
    if (index < 0)
    {
        printf("INDEX RETRIEVE FAILED");
        return -1; // File or directory does not exist
    }

    // fill in dir->information into buf->information

    DE *dir = &(ppi->parent[index]);

    buf->st_size = dir->fileSize;
    buf->st_blksize = BLOCK_SIZE;
    buf->st_blocks = (dir->fileSize + BLOCK_SIZE - 1) / BLOCK_SIZE;
    buf->st_accesstime = dir->lastAccessedTime;
    buf->st_modtime = dir->modifiedTime;
    buf->st_createtime = dir->createdTime;
    buf->st_isdir = dir->isDirectory;

        if(ppi->parent != rootDir && ppi->parent != cwd)
                {
                free(ppi->parent);
                }

    if (ppi != NULL)
        {
        free(ppi);
        ppi=NULL;
        }

    return 0; // Success
}
```

- *Description*

fs_stat: utilizes for retrieving metadata about a file or directory within a filesystem, mirroring the Unix-like system. If the entry exists, the function proceeds to populate the fs_stat structure pointed to by buf with various attributes of the file or directory. These attributes include size (st_size), block size (st_blksize), number of blocks (st_blocks), access time (st_accesstime), modification time (st_modtime), creation time (st_createtime), and a flag indicating if it's a directory (st_isdir). It deeply relates to fs_opendir && fs_readdir, specifically the ls command line.

7. **fs_opendir**

Code:

```
fdDir *fs_opendir(const char *pathname)
{
    printf("\nstart of fs_opendir with pathname : |%s|\n", pathname);

    if (pathname == NULL)
    {
        printf("Directory Not Found");
        return NULL;
    }

    // update pathname with new element
    char *newPath = pathUpdate(pathname);

    // initilize a directory and a parsePathInfo struct
    DE *myDir;
    parsePathInfo *ppi = malloc(sizeof(parsePathInfo));

    // call parsePath() to traverse and update ppi
    int parsePathCheck = parsePath(newPath, ppi);
    printf("Inside fs_opendir return value of parsePath(): %d\n", parsePathCheck);
    printf("Inside fs_opendir ppi indexOfLast Element: %d\n", ppi->indexOfLastElement);
    if (newPath != NULL)
        free(newPath);

    // check if directory with pathname exists
    if (parsePathCheck != 0)
    {
        printf("Invalid path!\n");
        return NULL;
    }

    // check if pathname is a directory

    if (ppi->indexOfLastElement < 0)
    {
        printf("%s is not a directory\n", ppi->lastElement);
        return NULL;
    }
```

```
    printf("Inside of fs_opendir isDirectory() value returned: %d\n", isDirectory(&ppi->parent[ppi->indexOfLastElement]));
    printf("Inside of fs_opendir ppi parent isDirectory value: %d\n", ppi->parent[ppi->indexOfLastElement].isDirectory);

    if (isDirectory(&ppi->parent[ppi->indexOfLastElement]) <= 0)
    {
        printf("%s is not a directory\n", ppi->lastElement);
        return NULL;
    }

    // load directory to initialize it in fdDir struct that will be the return value
    myDir = loadDir(ppi->parent, ppi->indexOfLastElement);
    printf("Inside of fs_opendir ppi->indexOfLastElement: %d\n", ppi->indexOfLastElement);

    fdDir *fdd = malloc(sizeof(fdDir));

    fdd->directory = myDir;
    printf("openDir fdd->firectory.isDirectory: %d\n", fdd->directory->isDirectory);
    fdd->dirEntryPosition = 0;
    fdd->d_reclen = sizeof(fdDir);

    printf("End of fs_opendir\n");
    // cleanup


    if(ppi->parent != rootDir && ppi->parent != cwd)
                {
                free(ppi->parent);
                }

    if (ppi!= NULL) free(ppi);

    return (fdd);
}
// end of fs_opendir()
```

Description:

The fs_opendir function is designed to open a directory within a file system. The function takes a pathname as input, which specifies the directory to be opened. It processes this pathname to understand the directory structure using parsePath to navigate through the filesystem. It checks whether the given path actually leads to a directory. This involves confirming that the path exists and refers to a directory rather than a file, using functions like isDirectory. Upon validating the directory, the function proceeds to load the directory's contents into memory. This involves reading the directory's metadata and its contents from the disk into a structure in memory. The function loadDir is responsible for this task. The function initializes or updates a file descriptor structure (fdd) that will be used to track the state of the open directory. On successful opening of the directory, the function returns a file descriptor that can be used in subsequent operations like reading from the directory (fs_readdir) or closing it (fs_closedir).

8. **fs_readdir**

Code:

```
struct fs_diriteminfo *fs_readdir(fdDir *dirPath)
{
    // Read the next directory entry from the directory specified by dirPath
    // Update the dirEntryPosition in dirPath to keep track of the position
    // Return NULL if there are no more entries

    printf("\nInside fs_readdir\n");
    int directoryEntries = dirPath->directory->fileSize / sizeof(DE);
    // int directoryEntries = 0;
    printf("fs_readdir directoryEntries value: %d\n", directoryEntries);
    printf("dirPath->dirEntryPosition: %d\n", dirPath->dirEntryPosition);

    if (dirPath == NULL || dirPath->directory == NULL)
    {
        printf("dirPath is NULL\n");
        return NULL;
    }

    printf("After checking if dirPath = NULL\n");

    // check if the directory entry is being used, iterate until you find a use entry
    while (dirPath->directory[dirPath->dirEntryPosition].fileName[0] == '\0')
    {
        printf("Inside while loop\n");
        dirPath->dirEntryPosition++;
        if (dirPath->dirEntryPosition >= directoryEntries)
        {
            return NULL;
        }
    }

    struct fs_diriteminfo * dii= malloc(sizeof(struct fs_diriteminfo));
    dirPath->di= dii;

    // copy the name of currenct directory to fs_diriteminfo
    strcpy(dirPath->di->d_name, dirPath->directory[dirPath->dirEntryPosition].fileName);
    printf("di->d_name: %s\n", dirPath->di->d_name);
```

```
    // check the type of the directory
    if (isDirectory(&dirPath->directory[dirPath->dirEntryPosition]) == 1)
    {
        dirPath->di->fileType = 'd';
    }
    else
    {
        dirPath->di->fileType = 'f';
    }

    // update positon for next iteration
    dirPath->dirEntryPosition++;

    return dirPath->di;
}
```

Description:

The fs_readdir function is used to read the contents of an opened directory in a file system. The function takes a directory descriptor as its input. This is obtained from a previous call to fs_opendir, which opens a directory for reading. fs_readdir` reads the next directory entry in the sequence from the opened directory. A directory entry could be a file or a subdirectory. The function iterates through the directory entries sequentially. The function returns a structure that represents the current directory entry. This information includes the name of the file or directory, its type, and size of record. The function maintains the state of the directory reading process. This includes keeping track of the current position within the directory. Each call to fs_readdir advances this position to the next entry, making it ready for subsequent reads. When all entries in the directory have been read, fs_readdir indicates that the end of the directory has been reached. This is done by returning a null value.

Challenges:

1. Unable to execute properly → fs_readdir

- *Short Cut Code*

```c
struct fs_diriteminfo *fs_readdir(fdDir *dirPath)
{
    // Read the next directory entry from the directory specified by dirPath
    // Update the dirEntryPosition in dirPath to keep track of the position
    // Return NULL if there are no more entries

    printf("\nInside fs_readdir\n");
    int directoryEntries = dirPath->directory->fileSize / sizeof(DE);
    // int directoryEntries = 0;
    printf("fs_readdir directoryEntries value: %d\n", directoryEntries);
    printf("dirPath->dirEntryPosition: %d\n", dirPath->dirEntryPosition);

    if (dirPath == NULL || dirPath->directory == NULL)
    {
        printf("dirPath is NULL\n");
        return NULL;
    }

    printf("After checking if dirPath = NULL\n");

    // check if the directory entry is being used, iterate until you find a use entry
    while (dirPath->directory[dirPath->dirEntryPosition].fileName[0] == '\0')
    {
        printf("Inside while loop\n");
        dirPath->dirEntryPosition++;
        if (dirPath->dirEntryPosition >= directoryEntries)
        {
            return NULL;
        }
    }
```

- *Issue (no Resolution so far)*

The issue is an infinite loop when executing an ls command. In fs_readdir, directory entries are read and the dirEntryPosition is incremented to move to the next entry. However, since the code doesn't correctly handle the end of the directory entries or if it fails to properly identify unused entries (entries where fileName[0] == '\0'), it might trigger multiple loops. This behavior suggests that the condition to exit the while loop in fs_readdir (dirPath→dirEntryPosition >= directoryEntries) might not be met, possibly due to an incorrect calculation of directoryEntries or an error in updating dirPath→dirEntryPosition. We believe that another feasible issue could be in the fs_opendir function, particularly in how it sets up the fdDir structure or initializes dirEntryPosition, which could affect the behavior of fs_readdir. We tried to resolve this issue, but so far, it is unavailable.

- *Test Failure*

```
Inside fs_readdir
fs_readdir directoryEntries value: 56
dirPath->dirEntryPosition: 2
After checking if dirPath = NULL
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
```

2. Nested md and cd

   - *Issue*

     case 1:

     md hi → cd hi → md hello → cd hello → pwd

     case 2:

     md hi → md hi/hello → cd hi/hello → pwd

     The expected output is /hi/hello because fs_getcwd always returns the absolute path of the current working directory. I am not sure why, but it does not update the pathname properly so that still remains /hi

- *case1: Test Failure*

```
Prompt > md hi
called fs_mkdr with pathname : |hi|
pathname 0 char : |
newdir pathname : |/hi|
startdir[2]:
Parsepath completed succesfuly
InitDir() call, passed parentIndex: 2
p filename: .
size Directory entry: 72
byteNeeded: 4096, blocksNeeded: 8, actualDirEntries: 56
call allocation blocks inside initdir()
In initDir tempArray returned of extent *. length tempArray: 1, value of tempArray[0] start: 19 count: 8
dir[0].extentStart: 14
check subdir[0].fileName: .
initDir- dir[1].extentBlockStart: 27
initDir- parent[parentIndex=2].extentBlockStart: 27
in mkdir startBlockNewDir: 19
ROOT PARENT ext[2].tableArray[0].start: 19
parent directory start block from extent table: 6
inside mk dir- parent[2] startextentblock: 27
loadDir called, with index: 2
filename at index 2 in parent hi
extent table returned start of loading dir at: 19
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 27, parent[2] filename: hi,
Prompt > cd hi
fs_setcwd starts with cwdAbsolutePath: /
setcwd start root[0]:|.| filesize: 4096 _____ root[1]: |..| filesize: 4096  root[2]: |hi| filesize: 4096
setcwd start cwd[0]:|.| filesize: 4096 _____ cwd[1]: |..| filesize: 4096  cwd[2]: |hi| filesize: 4096
token 0: |hi| pathComponents[0]: |hi|
value of cwdAbsolutepath: |/|
startdir[2]: hi
Parsepath completed succesfuly
index : 2
stored in cwdAbsolutePath: |/hi|
loadDir called, with index: 2
filename at index 2 in parent hi
extent table returned start of loading dir at: 19
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 27, parent[2] filename: hi,
2 setcwd root[0]:|.| filesize: 4096 _____ root[1]: |..| filesize: 4096  root[2]: |hi| filesize: 4096
2 setcwd cwd[0]:|.| filesize: 4096 _____ cwd[1]: |..| filesize: 4096  cwd[2]: || filesize: 0
Prompt > pwd
fs_getcwd function called
cwdAbsolutePath's length: 3
Limitation of buffer size: 4095
fs_getcwd: /hi
/hi
```

```
Prompt > md hello
called fs_mkdr with pathname : |hello|
pathname 0 char : |
newdir pathname : |/hi/hello|
startdir[2]: hi
loadDir called, with index: 2
filename at index 2 in parent hi
extent table returned start of loading dir at: 19
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 27, parent[2] filename: hi,
Parsepath completed succesfuly
InitDir() call, passed parentIndex: 2
p filename: .
size Directory entry: 72
byteNeeded: 4096, blocksNeeded: 8, actualDirEntries: 56
call allocation blocks inside initdir()
In initDir tempArray returned of extent *. length tempArray: 1, value of tempArray[0] start: 32 count: 8
dir[0].extentStart: 27
check subdir[0].fileName: .
initDir- dir[1].extentBlockStart: 40
initDir- parent[parentIndex=2].extentBlockStart: 40
in mkdir startBlockNewDir: 32
ROOT PARENT ext[2].tableArray[0].start: 32
parent directory start block from extent table: 19
inside mk dir- parent[2] startextentblock: 40
loadDir called, with index: 2
filename at index 2 in parent hello
extent table returned start of loading dir at: 32
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 40, parent[2] filename: hello,
Prompt > cd hello
fs_setcwd starts with cwdAbsolutePath: /hi
setcwd start root[0]:|.| filesize: 4096 _____ root[1]: |..| filesize: 4096  root[2]: |hi| filesize: 4096
setcwd start cwd[0]:|.| filesize: 4096 _____ cwd[1]: |..| filesize: 4096  cwd[2]: |hello| filesize: 4096
token 0: |hello| pathComponents[0]: |hello|
value of cwdAbsolutepath: |/hi|
startdir[2]: hi
loadDir called, with index: 2
filename at index 2 in parent hi
extent table returned start of loading dir at: 19
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 27, parent[2] filename: hi,
Parsepath completed succesfuly
index : 2
stored in cwdAbsolutePath: |/hi|
loadDir called, with index: 2
filename at index 2 in parent hello
extent table returned start of loading dir at: 32
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 40, parent[2] filename: hello,
2 setcwd root[0]:|.| filesize: 4096 _____ root[1]: |..| filesize: 4096  root[2]: |hi| filesize: 4096
2 setcwd cwd[0]:|.| filesize: 4096 _____ cwd[1]: |..| filesize: 4096  cwd[2]: || filesize: 0
Prompt > pwd
fs_getcwd function called
cwdAbsolutePath's length: 3
Limitation of buffer size: 4095
fs_getcwd: /hi
/hi
Prompt >
```

→ As previously mentioned, our intended output would be /hi/hello, but we got /hi

- *case 2: Test failure*

```
Prompt > md hi
called fs_mkdr with pathname : |hi|
pathname 0 char : |
newdir pathname : |/hi|
startdir[2]:
Parsepath completed succesfuly
InitDir() call, passed parentIndex: 2
p filename: .
size Directory entry: 72
byteNeeded: 4096, blocksNeeded: 8, actualDirEntries: 56
call allocation blocks inside initdir()
In initDir tempArray returned of extent *. length tempArray: 1, value of tempArray[0] start: 19 count: 8
dir[0].extentStart: 14
check subdir[0].fileName: .
initDir- dir[1].extentBlockStart: 27
initDir- parent[parentIndex=2].extentBlockStart: 27
in mkdir startBlockNewDir: 19
ROOT PARENT ext[2].tableArray[0].start: 19
parent directory start block from extent table: 6
inside mk dir- parent[2] startextentblock: 27
loadDir called, with index: 2
filename at index 2 in parent hi
extent table returned start of loading dir at: 19
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 27, parent[2] filename: hi,
Prompt > md hi/hello
called fs_mkdr with pathname : |hi/hello|
pathname 0 char : |
newdir pathname : |/hi/hello|
startdir[2]: hi
loadDir called, with index: 2
filename at index 2 in parent hi
extent table returned start of loading dir at: 19
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 27, parent[2] filename: hi,
Parsepath completed succesfuly
InitDir() call, passed parentIndex: 2
p filename: .
size Directory entry: 72
byteNeeded: 4096, blocksNeeded: 8, actualDirEntries: 56
call allocation blocks inside initdir()
In initDir tempArray returned of extent *. length tempArray: 1, value of tempArray[0] start: 32 count: 8
dir[0].extentStart: 27
check subdir[0].fileName: .
initDir- dir[1].extentBlockStart: 40
initDir- parent[parentIndex=2].extentBlockStart: 40
in mkdir startBlockNewDir: 32
ROOT PARENT ext[2].tableArray[0].start: 32
parent directory start block from extent table: 19
inside mk dir- parent[2] startextentblock: 40
loadDir called, with index: 2
filename at index 2 in parent hello
extent table returned start of loading dir at: 32
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 40, parent[2] filename: hello,
```

```
Prompt > cd hi/hello
fs_setcwd starts with cwdAbsolutePath: /
setcwd start root[0]:|.| filesize: 4096 _____ root[1]: |..| filesize: 4096  root[2]: |hi| filesize: 4096
setcwd start cwd[0]:|.| filesize: 4096 _____ cwd[1]: |..| filesize: 4096  cwd[2]: |hi| filesize: 4096
token 0: |hi| pathComponents[0]: |hi|
value of cwdAbsolutepath: |/|
startdir[2]: hi
loadDir called, with index: 2
filename at index 2 in parent hi
extent table returned start of loading dir at: 19
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 27, parent[2] filename: hi,
Parsepath completed succesfuly
index : 2
stored in cwdAbsolutePath: |/hi|
loadDir called, with index: 2
filename at index 2 in parent hello
extent table returned start of loading dir at: 32
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 40, parent[2] filename: hello,
2 setcwd root[0]:|.| filesize: 4096 _____ root[1]: |..| filesize: 4096  root[2]: |hi| filesize: 4096
2 setcwd cwd[0]:|.| filesize: 4096 _____ cwd[1]: |..| filesize: 4096  cwd[2]: || filesize: 0
Prompt > pwd
fs_getcwd function called
cwdAbsolutePath's length: 3
Limitation of buffer size: 4095
fs_getcwd: /hi
/hi
Prompt >
```

→ As previously mentioned, our intended output would be /hi/hello, but we got /hi

- *case 1: Temporary Resolution*

Instead of parsePath(newPath, ppi), we tried to use parsePath(pathname, ppi)

```
Prompt > md hi
called fs_mkdr with pathname : |hi|
pathname 0 char : |
newdir pathname : |/hi|
startdir[2]:
Parsepath completed succesfuly
InitDir() call, passed parentIndex: 2
p filename: .
size Directory entry: 72
byteNeeded: 4096, blocksNeeded: 8, actualDirEntries: 56
call allocation blocks inside initdir()
In initDir tempArray returned of extent *. length tempArray: 1, value of tempArray[0] start: 19 count: 8
dir[0].extentStart: 14
check subdir[0].fileName: .
initDir- dir[1].extentBlockStart: 27
initDir- parent[parentIndex=2].extentBlockStart: 27
in mkdir startBlockNewDir: 19
ROOT PARENT ext[2].tableArray[0].start: 19
parent directory start block from extent table: 6
inside mk dir- parent[2] startextentblock: 27
loadDir called, with index: 2
filename at index 2 in parent hi
extent table returned start of loading dir at: 19
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 27, parent[2] filename: hi,
Prompt > cd hi
fs_setcwd starts with cwdAbsolutePath: /
setcwd start root[0]:|.| filesize: 4096 _____ root[1]: |..| filesize: 4096  root[2]: |hi| filesize: 4096
setcwd start cwd[0]:|.| filesize: 4096 _____ cwd[1]: |..| filesize: 4096  cwd[2]: |hi| filesize: 4096
token 0: |hi| pathComponents[0]: |hi|
value of cwdAbsolutepath: |/|
startdir[2]: hi
Parsepath completed succesfuly
index : 2
stored in cwdAbsolutePath: |/hi|
loadDir called, with index: 2
filename at index 2 in parent hi
extent table returned start of loading dir at: 19
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 27, parent[2] filename: hi,
2 setcwd root[0]:|.| filesize: 4096 _____ root[1]: |..| filesize: 4096  root[2]: |hi| filesize: 4096
2 setcwd cwd[0]:|.| filesize: 4096 _____ cwd[1]: |..| filesize: 4096  cwd[2]: || filesize: 0
```

```
Prompt > md hello
called fs_mkdr with pathname : |hello|
pathname 0 char : |
newdir pathname : |/hi/hello|
startdir[2]: hi
loadDir called, with index: 2
filename at index 2 in parent hi
extent table returned start of loading dir at: 19
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 27, parent[2] filename: hi,
Parsepath completed succesfuly
InitDir() call, passed parentIndex: 2
p filename: .
size Directory entry: 72
byteNeeded: 4096, blocksNeeded: 8, actualDirEntries: 56
call allocation blocks inside initdir()
In initDir tempArray returned of extent *. length tempArray: 1, value of tempArray[0] start: 32 count: 8
dir[0].extentStart: 27
check subdir[0].fileName: .
initDir- dir[1].extentBlockStart: 40
initDir- parent[parentIndex=2].extentBlockStart: 40
in mkdir startBlockNewDir: 32
ROOT PARENT ext[2].tableArray[0].start: 32
parent directory start block from extent table: 19
inside mk dir- parent[2] startextentblock: 40
loadDir called, with index: 2
filename at index 2 in parent hello
extent table returned start of loading dir at: 32
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 40, parent[2] filename: hello,
Prompt > cd hello
fs_setcwd starts with cwdAbsolutePath: /hi
setcwd start root[0]:|.| filesize: 4096 _____ root[1]: |..| filesize: 4096  root[2]: |hi| filesize: 4096
setcwd start cwd[0]:|.| filesize: 4096 _____ cwd[1]: |..| filesize: 4096  cwd[2]: |hello| filesize: 4096
token 0: |hello| pathComponents[0]: |hello|
value of cwdAbsolutepath: |/hi|
startdir[2]: hello
Parsepath completed succesfuly
index : 2
stored in cwdAbsolutePath: |/hi/hello|
loadDir called, with index: 2
filename at index 2 in parent
extent table returned start of loading dir at: 32
dir[index].fileSize: 0 | sizeInBlocks: 0
inside loadDir, extentBlockStart at newDir: 0, parent[2] filename: ,
2 setcwd root[0]:|.| filesize: 4096 _____ root[1]: |..| filesize: 4096  root[2]: |hi| filesize: 4096
2 setcwd cwd[0]:|◆◆ filesize: 32529 _____ cwd[1]: || filesize: 0  cwd[2]: || filesize: 0
Prompt > pwd
fs_getcwd function called
cwdAbsolutePath's length: 9
Limitation of buffer size: 4095
fs_getcwd: /hi/hello
/hi/hello
Prompt >
```

→ we got appropriate getcwd, /hi/hello. However, there is some weird symbol at the line 2 setcwd

case 2: Temporary Resolution

```
Prompt > md hi
called fs_mkdr with pathname : |hi|
pathname 0 char : |
newdir pathname : |/hi|
startdir[2]:
Parsepath completed succesfuly
InitDir() call, passed parentIndex: 2
p filename: .
size Directory entry: 72
byteNeeded: 4096, blocksNeeded: 8, actualDirEntries: 56
call allocation blocks inside initdir()
In initDir tempArray returned of extent *. length tempArray: 1, value of tempArray[0] start: 19 count: 8
dir[0].extentStart: 14
check subdir[0].fileName: .
initDir- dir[1].extentBlockStart: 27
initDir- parent[parentIndex=2].extentBlockStart: 27
in mkdir startBlockNewDir: 19
ROOT PARENT ext[2].tableArray[0].start: 19
parent directory start block from extent table: 6
inside mk dir- parent[2] startextentblock: 27
loadDir called, with index: 2
filename at index 2 in parent hi
extent table returned start of loading dir at: 19
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 27, parent[2] filename: hi,
Prompt > md hi/hello
called fs_mkdr with pathname : |hi/hello|
pathname 0 char : |
newdir pathname : |/hi/hello|
startdir[2]: hi
loadDir called, with index: 2
filename at index 2 in parent hi
extent table returned start of loading dir at: 19
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 27, parent[2] filename: hi,
Parsepath completed succesfuly
InitDir() call, passed parentIndex: 2
p filename: .
size Directory entry: 72
byteNeeded: 4096, blocksNeeded: 8, actualDirEntries: 56
call allocation blocks inside initdir()
In initDir tempArray returned of extent *. length tempArray: 1, value of tempArray[0] start: 32 count: 8
dir[0].extentStart: 27
check subdir[0].fileName: .
initDir- dir[1].extentBlockStart: 40
initDir- parent[parentIndex=2].extentBlockStart: 40
in mkdir startBlockNewDir: 32
ROOT PARENT ext[2].tableArray[0].start: 32
parent directory start block from extent table: 19
inside mk dir- parent[2] startextentblock: 40
loadDir called, with index: 2
filename at index 2 in parent hello
extent table returned start of loading dir at: 32
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 40, parent[2] filename: hello,
```

```
Prompt > cd hi/hello
fs_setcwd starts with cwdAbsolutePath: /
setcwd start root[0]:|.| filesize: 4096 _____ root[1]: |..| filesize: 4096  root[2]: |hi| filesize: 4096
setcwd start cwd[0]:|.| filesize: 4096 _____ cwd[1]: |..| filesize: 4096  cwd[2]: |hi| filesize: 4096
token 0: |hi| pathComponents[0]: |hi|
value of cwdAbsolutepath: |/|
startdir[2]: hi
Parsepath completed succesfuly
index : 2
stored in cwdAbsolutePath: |/hi/hello|
loadDir called, with index: 2
filename at index 2 in parent hi
extent table returned start of loading dir at: 19
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 27, parent[2] filename: hi,
2 setcwd root[0]:|.| filesize: 4096 _____ root[1]: |..| filesize: 4096  root[2]: |hi| filesize: 4096
2 setcwd cwd[0]:|.| filesize: 4096 _____ cwd[1]: |..| filesize: 4096  cwd[2]: |hello| filesize: 4096
Prompt > pwd
fs_getcwd function called
cwdAbsolutePath's length: 9
Limitation of buffer size: 4095
fs_getcwd: /hi/hello
/hi/hello
Prompt >
```

Issue 4

Parsepath was returning failure with creating absolute directories in root.

Resolution

Print statements were uncommented back in, and we found that the parse path routine was missing a conditional check and adding an extra “/” to the start of absolute directories. Shown in code below.

```
|---------------------------------|
Prompt > md fire
called fs_mkdr with pathname : |fire|
newdir pathname : |/fire|
Success- directory made
Prompt > md /fire
called fs_mkdr with pathname : |/fire|
newdir pathname : |//fire|
Success- directory made
Prompt >
```

```
int fs_mkdir(const char *pathname, mode_t mode)
{

    // update pathname
    printf("called fs_mkdr with pathname : |%s|\n", pathname);
    // update pathname with new element
    char *newDir = malloc(256 * sizeof(char));
    strcpy(newDir, currentDir);
    printf("pathname 0 char : |%c|\n", pathname[0] != '/');
    if (currentDir[strlen(currentDir) - 1] != '/' && pathname[0] != '/')
    {
        strcat(newDir, "/");
    }
    strcat(newDir, pathname);
    printf("newdir pathname : |%s|\n", newDir);
```

Issue 5

setCwd

```
    return pathname;
}


int fs_setcwd(char *pathname) {

    printf("fs_setcwd starts: \n");

    // ParsePath
    parsePathInfo *ppi = malloc(sizeof(parsePathInfo));

    int parsePathResult = parsePath(pathname, ppi);
    printf("parsePathResult : %d \n", parsePathResult);

    if (parsePathResult != 0) return -1; // ParsePath check done(o)

    // find index
    int index = FindEntryInDir(ppi->parent, ppi->lastElement);
    printf("index : %d \n", index);

    if (index == -1) return -1;  // find index check done(o)

    // Check if the target is a directory
    if (!isDirectory(&ppi->parent[ppi->indexOfLastElement])) {
        printf("filename is not a directory \n");
        return -1; // Target is not a directory
    }

    // Free the previous cwd if it is not the root directory
    if (cwd != rootDir) free(cwd);

    cwd = loadDir(ppi->parent, index);
        printf("cwd filename: |%s|\n", ppi->parent[index].fileName);
    char *pathComponents[32];
    int componentCount = 0;
  //  cwdGlobal = index;
//    printf("cwdGlobal before iterating through: %d \n", cwdGlobal);
// ----------------------------------------------------------------

/*
```

```
int fs_setcwd(char *pathname)
{
    if(rootDir==NULL) loadDir(rootDir, rootGlobal);
    printf("fs_setcwd starts with cwdAbsolutePath: %s\n", cwdAbsolutePath);

    printf("setcwd start root[0]:|%s| filesize: %d _____ root[1]: |%s| filesize: %d  root[2]: |%s| filesize: %d\n", rootDir[0].fileName,
        rootDir[0].fileSize, rootDir[1].fileName, rootDir[1].fileSize, rootDir[2].fileName, rootDir[2].fileSize);

    printf("setcwd start cwd[0]:|%s| filesize: %d _____ cwd[1]: |%s| filesize: %d  cwd[2]: |%s| filesize: %d\n", cwd[0].fileName,
        cwd[0].fileSize, cwd[1].fileName, cwd[1].fileSize, cwd[2].fileName, cwd[2].fileSize);

    char *pathComponents[32];
    int componentCount = 0;
    char *newPath = malloc(256);

    // ParsePath
    parsePathInfo *ppi = malloc(sizeof(parsePathInfo));

    if (pathname[0] == '/')
        strcat(newPath, "/");

    // first tokenize path components to array and parse '.' and '..'
    char *saveptr = NULL;
    char *token = strtok_r(pathname, " /", &saveptr);

    if (token != NULL)
    {
        pathComponents[componentCount] = strdup(token);
        printf("token %d: |%s| pathComponents[%d]: |%s|\n", componentCount, token, componentCount, pathComponents[componentCount]);
        componentCount++;
    }
```

At first, there was an issue with parsing the array before calling parsePath. To fix this, it was just moved into the parsePath function. Also, the code used strtok instead of strtok_r, which caused issues with the threading.

```
if (strcmp(token, "..") == 0)
    {
                //get second to last item from parsePath
                int parsePathResult0 = parsePath(cwdAbsolutePath, ppi);
                if(parsePathResult0 == 0)
                        {
                            printf("parsePathResult set to parent");
                            cwd=ppi->parent;
                             printf("setcwd after cd . root[0]:|%s| filesize: %d _____ root[1]: |%s| filesize: %d  root[2]: |%s| filesize: %d\n", r
                            rootDir[0].fileSize, rootDir[1].fileName, rootDir[1].fileSize, rootDir[2].fileName, rootDir[2].fileSize);

                            printf("setcwd after cd . cwd[0]:|%s| filesize: %d _____ cwd[1]: |%s| filesize: %d  cwd[2]: |%s| filesize: %d\n", cwd[0
                            cwd[0].fileSize, cwd[1].fileName, cwd[1].fileSize, cwd[2].fileName, cwd[2].fileSize);

                            printf("cwdAbsolutePath before cd .. : |%s|\n", cwdAbsolutePath);
                            printf("strlen cwdABspath: |%ld|\n", strlen(cwdAbsolutePath));
                          int i= strlen(cwdAbsolutePath)-1;
            while(cwdAbsolutePath[i]!='/'){
            i--;
            }
            i++;
            cwdAbsolutePath[i]= '\0';
            printf("new cwdAbsolutePath after cd . : |%s|\n", cwdAbsolutePath);
            componentCount--;
            pathComponents[componentCount]= "";
        }
                }
if (strcmp(token, ".") == 0)
    {
    componentCount--;
            pathComponents[componentCount]= "";
    }
```

```
while (token != NULL && componentCount < 32)
{
    token = strtok_r(NULL, " /", &saveptr);
    if (token == NULL) break;

if(strcmp(token, ".") == 0)
        {
    if (componentCount > 0)
        {
        componentCount--;
        pathComponents[componentCount]= "";
        }
    }
    else if (strcmp(token, "..") != 0 )
    {
        // Normal directory component
        pathComponents[componentCount] = strdup(token);
        componentCount++;
    }
}

if(componentCount==0){
    return 0;

    }
printf("value of cwdAbsolutepath: |%s|\n", cwdAbsolutePath);
```

At first, the code looped through the entire array after tokenizing. To change this, logic was integrated to handle '.' and '..' cases during tokenization, reducing the need for a separate loop.

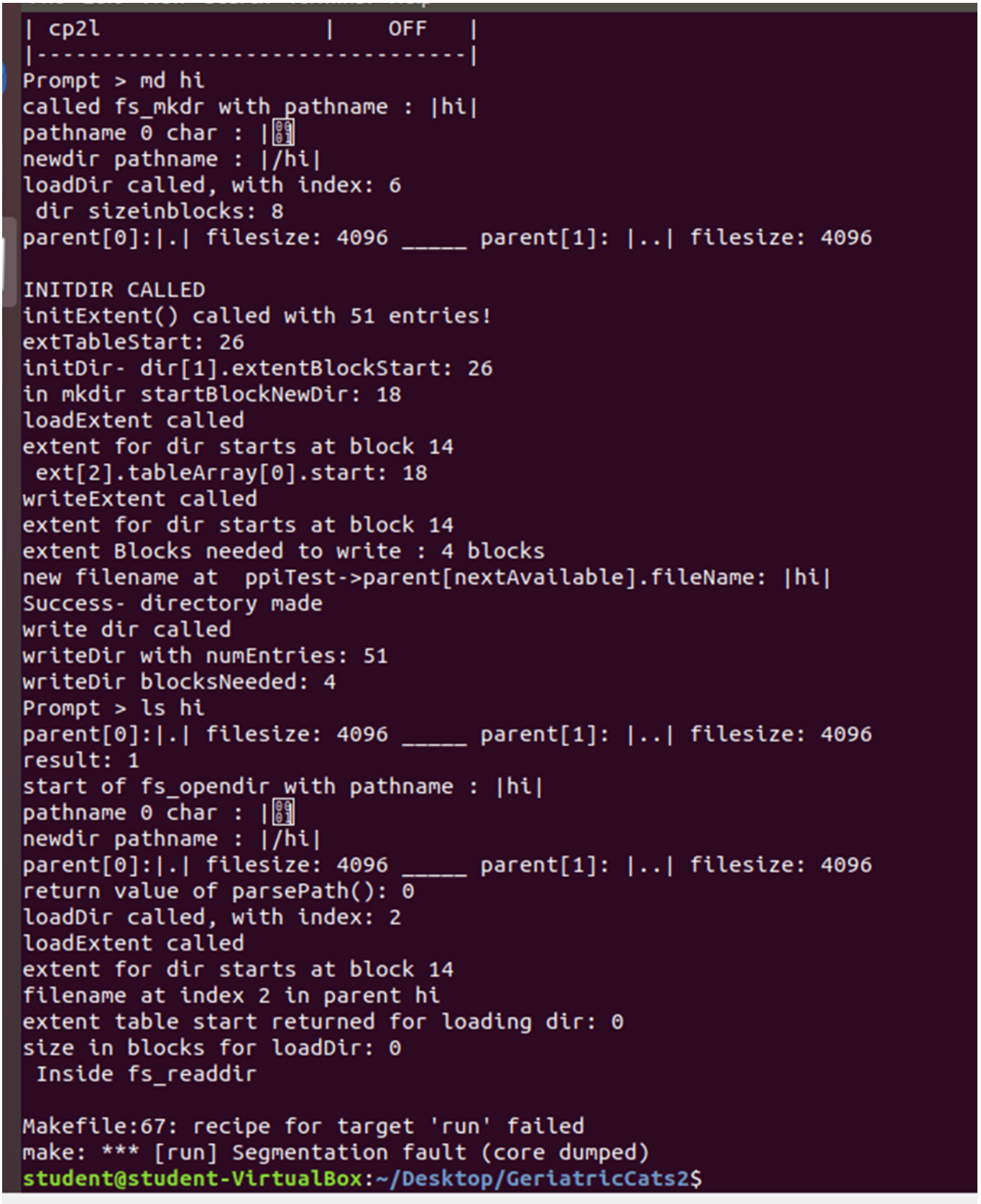
```
// add cwd to path (Relative)
if (pathname[0] != '/')
{
    strcpy(newPath, cwdAbsolutePath);
    // Ensure there's a trailing slash after cwdAbsolutePath, if not already present
    if (cwdAbsolutePath[strlen(cwdAbsolutePath) - 1] != '/')
    {
        strcat(newPath, "/");
    }
}

for (int i = 0; i < componentCount; i++)
{
    strcat(newPath, pathComponents[i]);
    if (i != componentCount - 1)
        strcat(newPath, "/");
}

// ParsePath
int parsePathResult = parsePath(pathname, ppi);

if (parsePathResult != 0)
    return -1; // ParsePath check done(o)

// find index
int index = FindEntryInDir(ppi->parent, ppi->lastElement);
printf("index : %d \n", index);

if (index == -1)
{
    printf("find entry returned not found\n");
    return -1; // find index check done(o)
}
```

```
// Check if the target is a directory
if (isDirectory(&ppi->parent[ppi->indexOfLastElement]) == 0)
{
    printf("%s is not a directory \n", ppi->lastElement);
    return -1; // Target is not a directory
}

if (cwdAbsolutePath != NULL)
    free(cwdAbsolutePath);
cwdAbsolutePath = newPath;
printf("stored in cwdAbsolutePath: |%s|\n", cwdAbsolutePath);

// set new cwd, cannot free old
if(cwd != rootDir) free(cwd);
cwd = loadDir(ppi->parent, index);

printf("2 setcwd root[0]:|%s| filesize: %d _____ root[1]: |%s| filesize: %d  root[2]: |%s| filesize: %d\n", rootDir[0].fileName,
      rootDir[0].fileSize, rootDir[1].fileName, rootDir[1].fileSize, rootDir[2].fileName, rootDir[2].fileSize);

printf("2 setcwd cwd[0]:|%s| filesize: %d _____ cwd[1]: |%s| filesize: %d  cwd[2]: |%s| filesize: %d\n", cwd[0].fileName,
      cwd[0].fileSize, cwd[1].fileName, cwd[1].fileSize, cwd[2].fileName, cwd[2].fileSize);

// cleanup
if(ppi->parent != rootDir && ppi->parent != cwd)
          {
          free(ppi->parent);
          }

if (ppi != NULL)
    {
    free(ppi);
    ppi=NULL;
    }

return 0; // Success
```

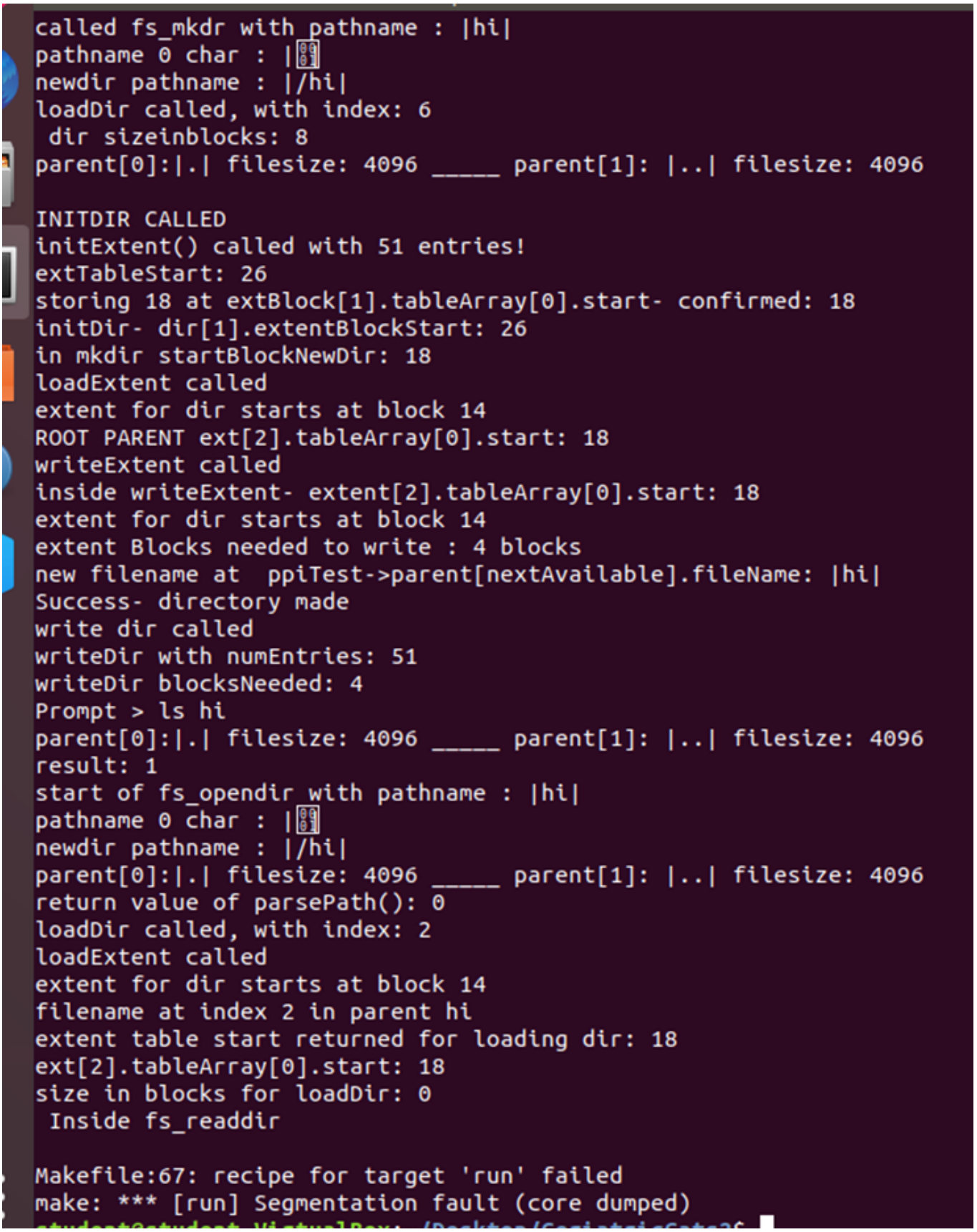
There was an issue where the current working directory was not loaded onto the path when it wasn't absolute. To address this, logic was added to connect the current working directory to the path if it was not absolute.

Issue 6

After loading dir, there was not the expected data after loading the extent table for the parent of our new dir. We decided that perhaps because we forgot to write the parent directory back out in md. So we made helpers for writeDir() and its accompanying writeExtent(). After writing the parent directory back out, this still did not fix the issue.



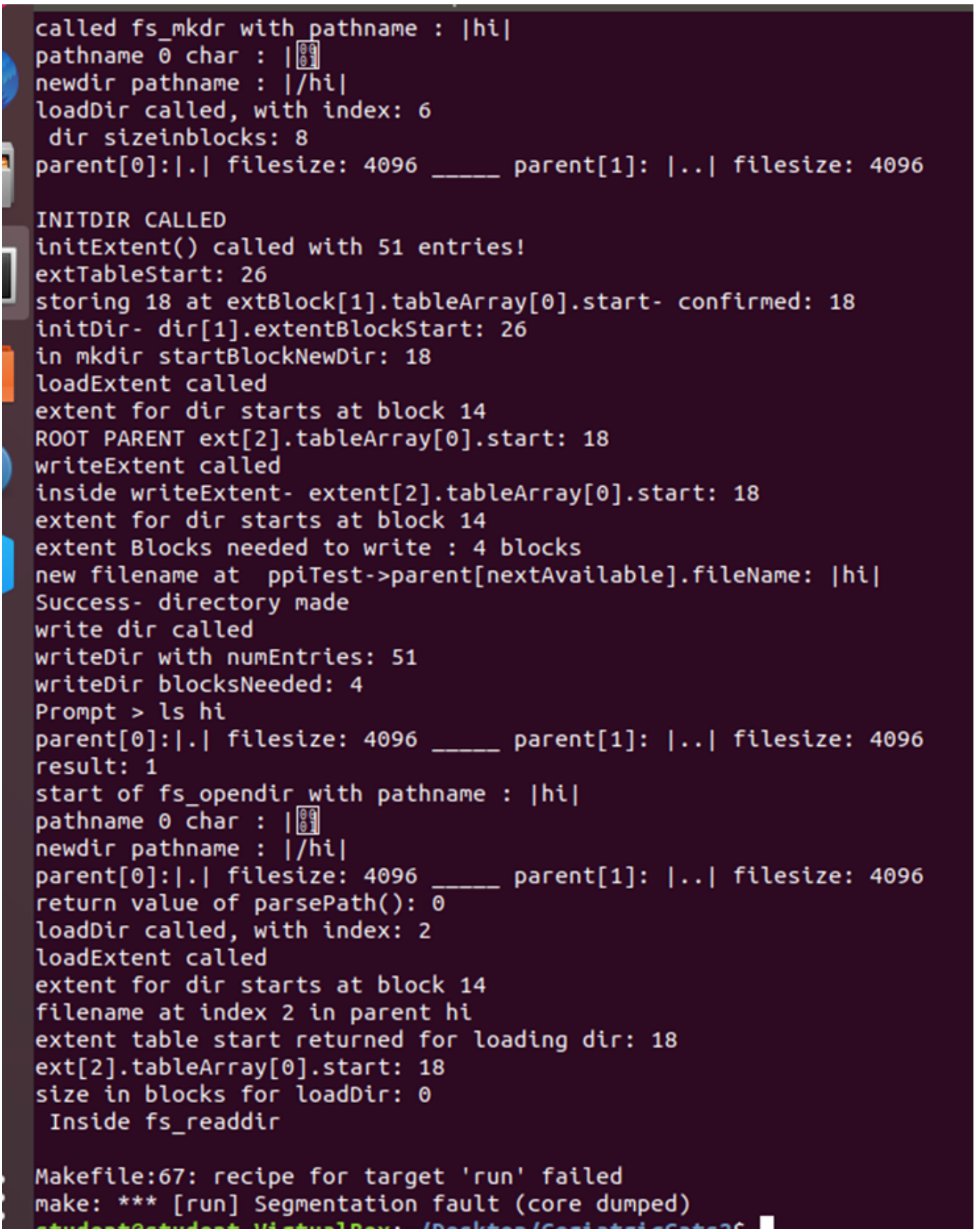
After is below.



We realized that loadDir and writeDir reported different number of blocks, and we rectified this with a small code change. This did not fix the error.

```
extTableStart: 14
storing 6 at extBlock[1].tableArray[0].start- confirmed: 6
|--------------------------------|
|------- Command ------|- Status -|
| ls                   |    ON    |
| cd                   |    OFF   |
| md                   |    ON    |
| pwd                  |    OFF   |
| touch                |    OFF   |
| cat                  |    OFF   |
| rm                   |    ON    |
| cp                   |    OFF   |
| mv                   |    OFF   |
| cp2fs                |    OFF   |
| cp2l                 |    OFF   |
|--------------------------------|
Prompt > md hi
called fs_mkdr with pathname : |hi|
pathname 0 char : |
newdir pathname : |/hi|
loadDir called, with index: 6
 dir sizeinblocks: 8
parent[0]:|.| filesize: 4096 _____ parent[1]: |..| filesize: 4096
inside initDir, passed parentIndex: 2

INITDIR CALLED
initExtent() called with 51 entries!
extTableStart: 26
storing 18 at extBlock[1].tableArray[0].start- confirmed: 18
initDir- dir[1].extentBlockStart: 26
initDir- parent[parentIndex=2].extentBlockStart: 26
in mkdir startBlockNewDir: 18
loadExtent called
extent for dir starts at block 14
ROOT PARENT ext[2].tableArray[0].start: 18
writeExtent called
inside writeExtent- extent[2].tableArray[0].start: 18
extent for dir starts at block 14
extent Blocks needed to write : 4 blocks
new filename at  ppiTest->parent[nextAvailable].fileName: |hi|
Success- directory made
write dir called
writeDir with numEntries: 51
writeDir blocksNeeded: 4
Prompt >
```

After is shown below


called fs_mkdr with pathname : |hi|
pathname 0 char : |
newdir pathname : |/hi|
loadDir called, with index: 6
dir sizeinblocks: 8
parent[0]:|.| filesize: 4096 _____ parent[1]: |..| filesize: 4096
INITDIR CALLED
initExtent() called with 51 entries!
extTableStart: 26
storing 18 at extBlock[1].tableArray[0].start- confirmed: 18
initDir- dir[1].extentBlockStart: 26
in mkdir startBlockNewDir: 18
loadExtent called
extent for dir starts at block 14
ROOT PARENT ext[2].tableArray[0].start: 18
writeExtent called
inside writeExtent- extent[2].tableArray[0].start: 18
extent for dir starts at block 14
extent Blocks needed to write : 4 blocks
new filename at ppiTest->parent[nextAvailable].fileName: |hi|
Success- directory made
write dir called
writeDir with numEntries: 51
writeDir blocksNeeded: 4
Prompt > ls hi
parent[0]:|.| filesize: 4096 _____ parent[1]: |..| filesize: 4096
result: 1
start of fs_opendir with pathname : |hi|
pathname 0 char : |
newdir pathname : |/hi|
parent[0]:|.| filesize: 4096 _____ parent[1]: |..| filesize: 4096
return value of parsePath(): 0
loadDir called, with index: 2
loadExtent called
extent for dir starts at block 14
filename at index 2 in parent hi
extent table start returned for loading dir: 18
ext[2].tableArray[0].start: 18
size in blocks for loadDir: 0
Inside fs_readdir
Makefile:67: recipe for target 'run' failed
make: *** [run] Segmentation fault (core dumped)

At this point ls was still not returning the correct start of directory value from its extent. Then we realized that we were not updating all of the information in the parent directory, only the name. We added a routine to the initDir for non-root directories to update the parent passed into them.

```
printf("check subdir[0].fileName: %s\n", dir[0].fileName);
// Initialize ".." entry for root AS PARENT PASSED
strcpy(dir[1].fileName, "..");
dir[1].fileSize = bytesNeeded;
dir[1].extentBlockStart= dirExtentBlock;
dir[1].extentIndex=1;
time_t t = time(NULL);
dir[1].createdTime = t;
dir[1].modifiedTime = t;
dir[1].lastAccessedTime = t;
dir[1].isDirectory = 1;
printf("initDir- dir[1].extentBlockStart: %d\n", dir[1].extentBlockStart);

//update parent's entry at index with same information
p[parentIndex].fileSize = bytesNeeded;
p[parentIndex].extentBlockStart= dirExtentBlock;
p[parentIndex].extentIndex =parentIndex;
p[parentIndex].createdTime = t;
p[parentIndex].modifiedTime = t;
p[parentIndex].lastAccessedTime = t;
p[parentIndex].isDirectory = 1;
printf("initDir- parent[parentIndex=2].extentBlockStart: %d\n", p[parentIndex].extentBlockStart);

}
```

And finally, while we checked that we update the parent entire directory entry with size, and wrote out the parent again, stored the new directory start block in the parent's extent entry and wrote that out, and then updated and wrote our new directory, and that its extent was properly initiated in initDir.
STILL the expected value was not in ls. At this point, we noticed that the writeDir for the parent was being written to the location of our new directory!
Then the issue was resolved.

```
//load parent
int startBlockNewDir = initDir(DEFAULT_ENTRIES, ppiTest->parent, nextAvailable)
printf("in mkdir startBlockNewDir: %d \n", startBlockNewDir);

// load parent extent block
EXTTABLE *ext = loadExtent(ppiTest->parent);

//set parent's extent with start of new dir
ext[nextAvailable].tableArray[0].start = startBlockNewDir;
printf("ROOT PARENT ext[%d].tableArray[0].start: %d\n", nextAvailable, ext[next
//write parent's extent
writeExtent(ppiTest->parent, ext);

// update directory entry name for new directory
char *copy = ppiTest->lastElement;
int i = 0;
while (copy[i])
{
    ppiTest->parent[nextAvailable].fileName[i] = copy[i];
    i++;
}
printf("new filename at  ppiTest->parent[nextAvailable].fileName: |%s|\n",
ppiTest->parent[nextAvailable].fileName);

printf("inside mk dir- parent[2] startextentblock: %d\n", ppiTest->parent[nextA
writeDir(ppiTest->parent, startBlockNewDir);
if(ext != NULL) free(ext);
```

As there were many functions involved in producing the bug- first tier at mk_dir(), calling second tier parsePath(), loadExtent(), writeExtent(), writeDir(), which called third tier initDir(), initExtent(), and loadDir(); but also many structures and pieces of data, there is a wash of various print statements for the data presented in orders that at least are based after the code that should have changed them. For this issue, we found that once I had isolated a problem, it would reveal another. The debugging was rather fluid and in a more frantic fashion, but a mass of print statements saved the day as they grew over two hours. The most important thing though, was diagramming what should be expected throughout the chained functions. This made the remaining debugging hour of three the most productive, as the data could be analyzed more efficiently, and smarter testing could be implemented.

Issue 7

When going to review and debug readDir, we needed contents in the directory to test it. That is when we realized that subdirectories did not work for md, such as calling mk_dir hi, and then mk_dir hi/there- if calling ls hi after creating a subdirectory, its data was corrupted. It functioned when root was the parent,

but not given another parent.

```
new filename at  ppiTest->parent[nextAvailable].fileName: |hi|
inside mk dir- parent[2] startextentblock: 26
write dir called
writeDir with numEntries: 51
dir[index].fileSize: 4096 | blocksNeeded: 8
Prompt > md hi/there
called fs_mkdr with pathname : |hi/there|
pathname 0 char : |
newdir pathname : |/hi/there|
parent[0]:|.| filesize: 4096 _____ parent[1]: |..| filesize: 4096
loadDir called, with index: 2
loadExtent called
extent for dir starts at block 14
filename at index 2 in parent hi
extent table returned start of loading dir at: 18
ext[2].tableArray[0].start: 18
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 26
inside initDir, passed parentIndex: 2

INITDIR CALLED
initExtent() called with 51 entries!
extTableStart: 39
storing 31 at extBlock[1].tableArray[0].start- confirmed: 31
initDir- dir[1].extentBlockStart: 39
initDir- parent[parentIndex=2].extentBlockStart: 39
in mkdir startBlockNewDir: 31
loadExtent called
extent for dir starts at block 26
ROOT PARENT ext[2].tableArray[0].start: 31
parent directory start block from extent table: 18
writeExtent called
inside writeExtent- extent[2].tableArray[0].start: 31
extent for dir starts at block 26
extent Blocks needed to write : 4 blocks
new filename at  ppiTest->parent[nextAvailable].fileName: |there|
inside mk dir- parent[2] startextentblock: 39
write dir called
writeDir with numEntries: 51
dir[index].fileSize: 4096 | blocksNeeded: 8
Prompt > ls hi
parent[0]:|there| filesize: -397714272 _____ parent[1]: || filesize: 33
Makefile:67: recipe for target 'run' failed
make: *** [run] Segmentation fault (core dumped)
student@student-VirtualBox:~/Desktop/GeriatricCats2$
```

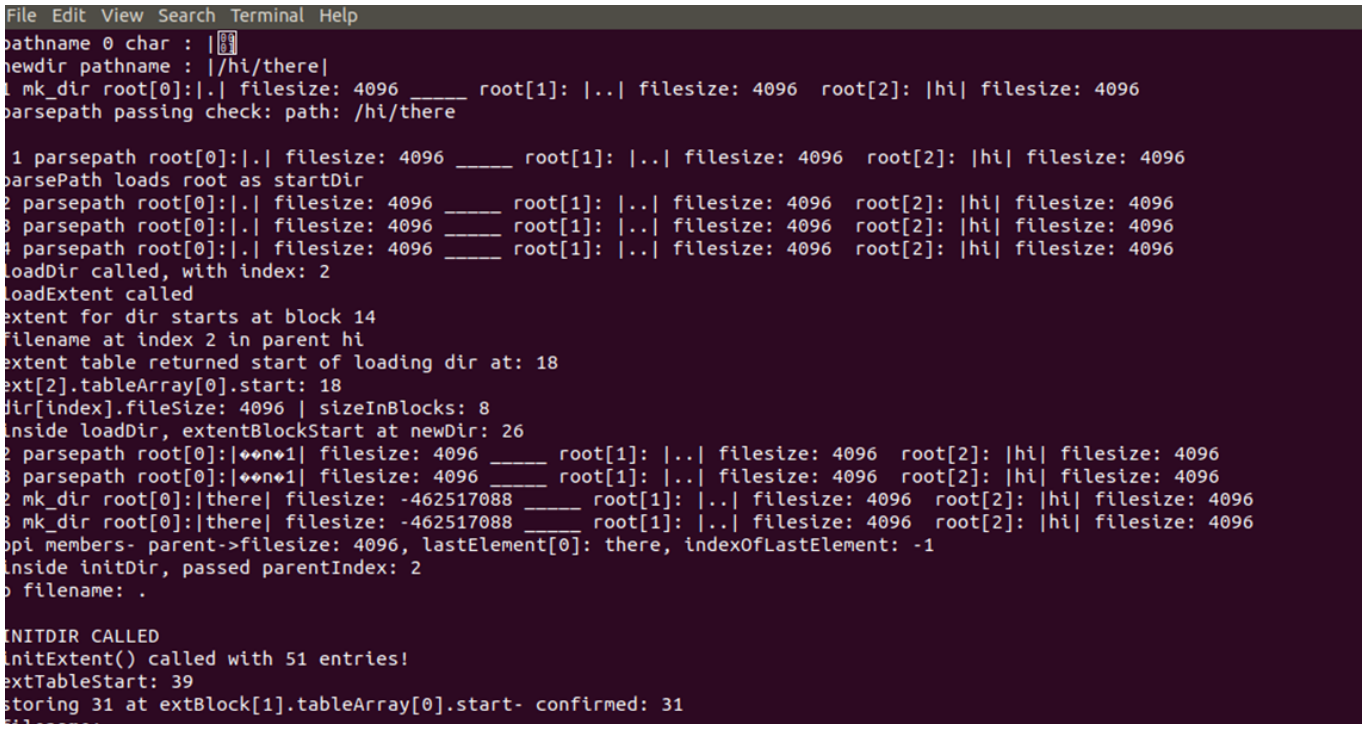
We chose to do 3-4 matching and more focused print statements per function distributed across the functions in a way that isolated the helper calls or other crux or “essential code pieces”, because the act of going through a mass of print statements (all of which were necessary to see at once but maybe not

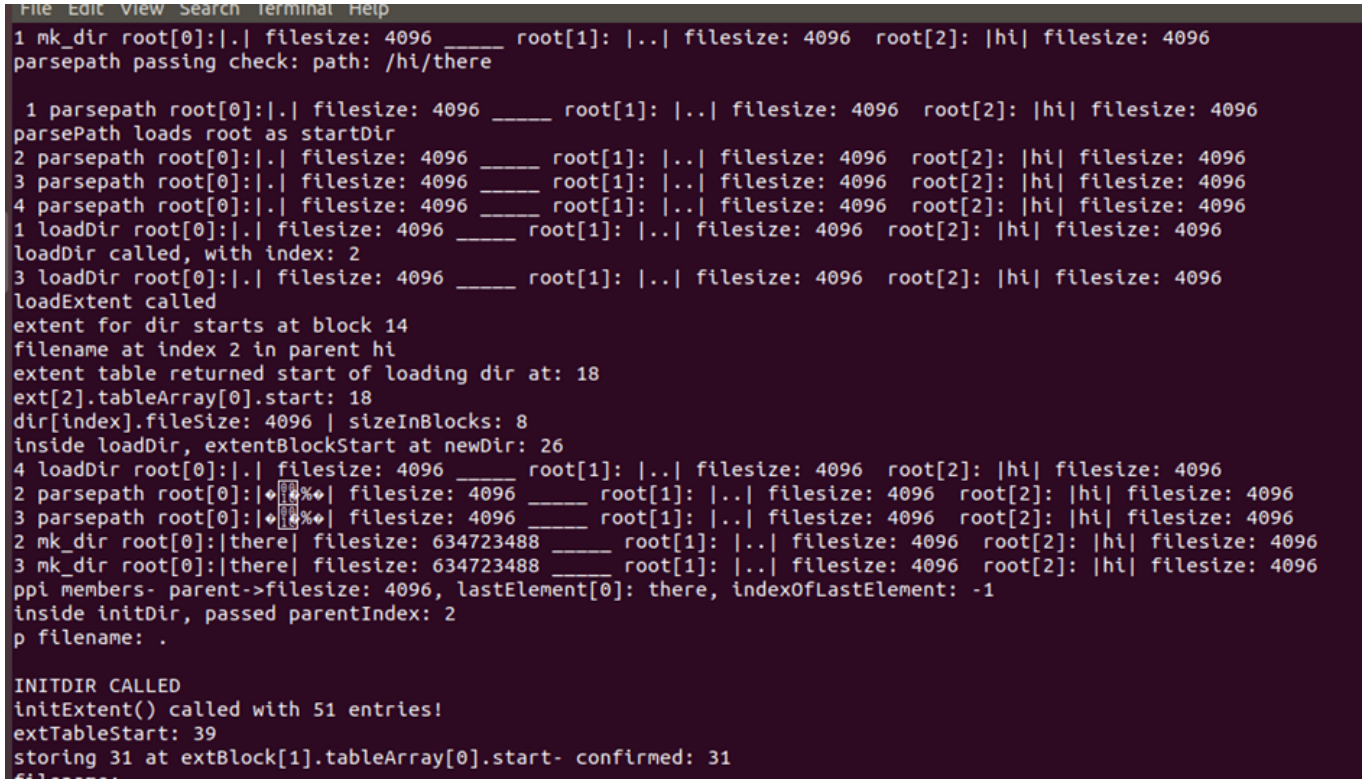
printed in an efficient way to use), too longer to read and analyze than to copy print statements as I would most likely be relocating them after each run while isolating the piece of code.

```
new filename at  ppiTest->parent[nextAvailable].fileName: |hi|
inside mk dir- parent[2] startextentblock: 26
write dir called
writeDir with numEntries: 51
dir[index].fileSize: 4096 | blocksNeeded: 8
Prompt > md hi/there
called fs_mkdr with pathname : |hi/there|
pathname 0 char : |
newdir pathname : |/hi/there|
parent[0]:|.| filesize: 4096 _____ parent[1]: |..| filesize: 4096
loadDir called, with index: 2
loadExtent called
extent for dir starts at block 14
filename at index 2 in parent hi
extent table returned start of loading dir at: 18
ext[2].tableArray[0].start: 18
dir[index].fileSize: 4096 | sizeInBlocks: 8
inside loadDir, extentBlockStart at newDir: 26
inside initDir, passed parentIndex: 2

INITDIR CALLED
initExtent() called with 51 entries!
extTableStart: 39
storing 31 at extBlock[1].tableArray[0].start- confirmed: 31
initDir- dir[1].extentBlockStart: 39
initDir- parent[parentIndex=2].extentBlockStart: 39
in mkdir startBlockNewDir: 31
loadExtent called
extent for dir starts at block 26
ROOT PARENT ext[2].tableArray[0].start: 31
parent directory start block from extent table: 18
writeExtent called
inside writeExtent- extent[2].tableArray[0].start: 31
extent for dir starts at block 26
extent Blocks needed to write : 4 blocks
new filename at  ppiTest->parent[nextAvailable].fileName: |there|
inside mk dir- parent[2] startextentblock: 39
write dir called
writeDir with numEntries: 51
dir[index].fileSize: 4096 | blocksNeeded: 8
Prompt > ls hi
parent[0]:|there| filesize: -397714272 _____ parent[1]: || filesize: 33
Makefile:67: recipe for target 'run' failed
make: *** [run] Segmentation fault (core dumped)
student@student-VirtualBox:~/Desktop/GeriatricCats2$
```

We traced the first issue with the corruption of the '.' In the root directory, assuming that when mk_dir later search for next available, some reference was not updating to the new directory and then storing the new directory in th first available at '.'. This traced to an in inappropriate free at the end of loadDir. It

turned out that the reference to rootDir had to be set at initDir time during volume initialization and not just at the first mk_dir call.





After this point, the issue now was that when calling ls after a md hi, md hi/there command pair,
The hi directory disappeared from the root, yielding in ls
root[0]: |.| filesize 4096 root[1]: |..| filesize 2096 root [2]: | | filesize: 0

We continued my search method as described in resolution, until a single bad code line was found and fixed in the loadDir.

```
// set parent's extent with start of new dir
ext[nextAvailable].tableArray[0].start = startBlockNewDir;
printf("ROOT PARENT ext[%d].tableArray[0].start: %d\n", nextAvailable, ext[nextAvailable].tableArray[0].start);
int parentDirStart = ext[1].tableArray[0].start;
printf("parent directory start block from extent table: %d\n", ext[1].tableArray[0].start);
// write parent's extent
writeExtent(ppiTest->parent, ext);
if (ext != NULL)
    free(ext);

// update directory entry name for new directory
char *copy = ppiTest->lastElement;
int i = 0;
while (copy[i])
{
    ppiTest->parent[nextAvailable].fileName[i] = copy[i];
    i++;
}
copy = NULL;
printf("new filename at  ppiTest->parent[nextAvailable].fileName: |%s|\n",
       ppiTest->parent[nextAvailable].fileName);

printf("inside mk dir- parent[2] startextentblock: %d\n", ppiTest->parent[nextAvailable].extentBlockStart);
writeDir(ppiTest->parent, parentDirStart);

    printf("mk_dir parent[0]:|%s| filesize: %d _____ parent[1]: |%s| filesize: %d  parent[2]: |%s| filesize: %d\n", ppiTest->parent[0].fileName,
    ppiTest->parent[0].fileSize, ppiTest->parent[1].fileName, ppiTest->parent[1].fileSize, ppiTest->parent[2].fileName, ppiTest->parent[2].fileSize);
    printf("mk_dir root[0]:|%s| filesize: %d _____ root[1]: |%s| filesize: %d  root[2]: |%s| filesize: %d\n", rootDir[0].fileName,
    rootDir[0].fileSize, rootDir[1].fileName, rootDir[1].fileSize, rootDir[2].fileName, rootDir[2].fileSize);


    // cleanup
    if(ppiTest!=NULL)
            {
            free(ppiTest);
            ppiTest=NULL;
            }

return 1;
```

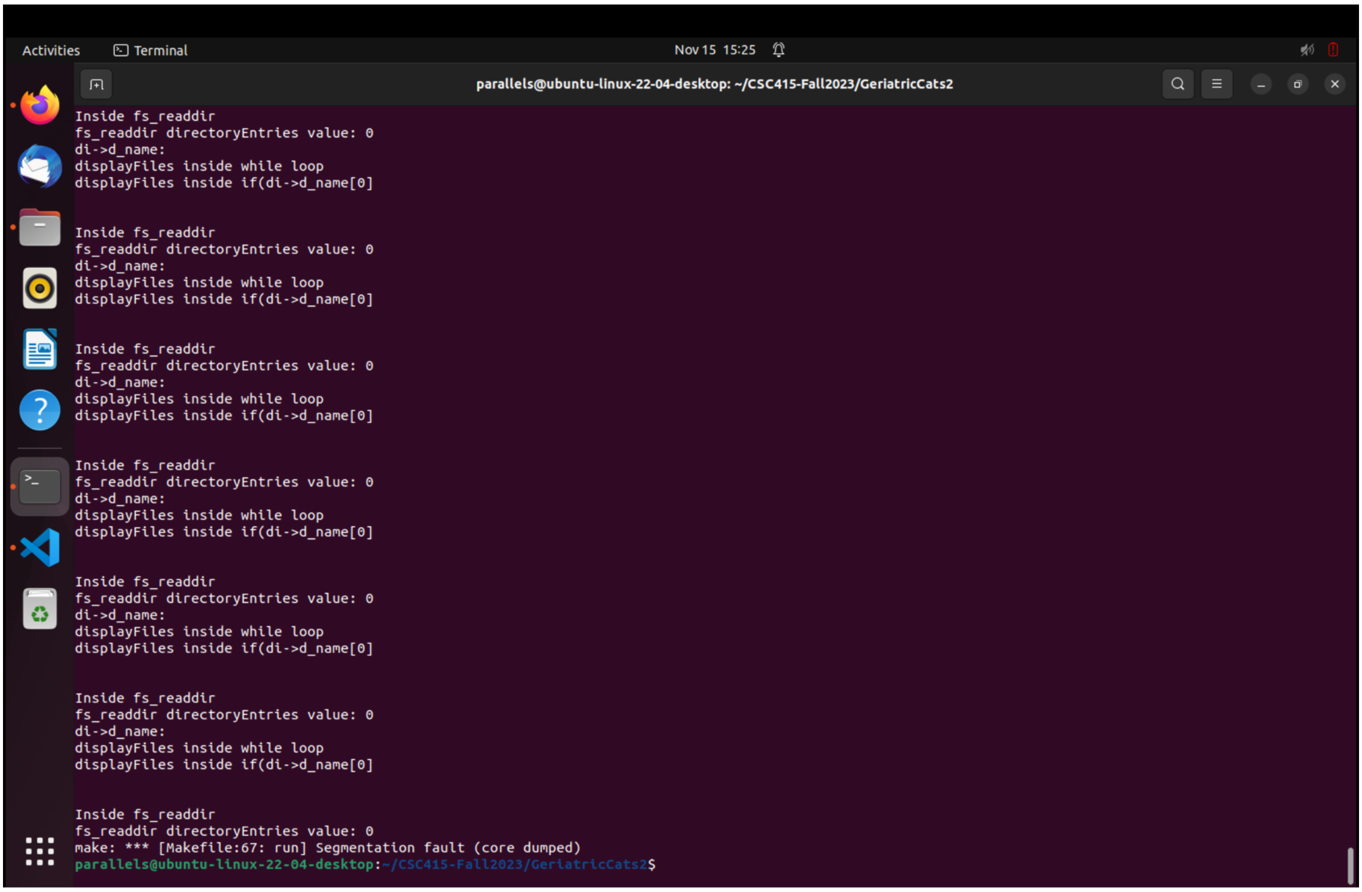
Resolution

Then, we worked my way through from the outer function of md_dir and isolated any code lines or helper functions, and then worked the same way in the helper functions through their code lines or functions causing the discrepancy until it was isolated.

This got us realizing that we can definitely plan prints in a more useful manner to debugging. In fact, when we get down to it, the act of debugging is tightly tied to the original plan of writing and checking while writing, as a style of maximal efficiency. Essentially, the larger picture of the program should determine the output as you go, making debugging faster as more essential data and their sections are.

Issue 8

When executing the ls command, which internally calls the fs_readdir function, an error was encountered. The fs_readdir function failed to correctly detect when a directory was not in use. As a consequence, it incorrectly processed directory data. This misbehavior resulted in the function populating the file descriptor with incorrect data, leading to inaccurate directory listings.



Resolution:

```
// check if the directory entry is being used, iterate until you find a use entry
while (&dirPath->directory[dirPath->dirEntryPosition] == NULL)
{
    printf("Inside while loop\n");
    dirPath->dirEntryPosition++;
    if (dirPath->dirEntryPosition >= directoryEntries)
    {
        return NULL;
    }
}
```

On debugging the ls command, we found that the loop condition in fs_readdir used to determine the validity of a directory entry was incorrect. The condition checked for a NULL pointer, which was not the correct method to verify an unused directory entry. The code was modified to check the first character of the directory entry's name against the null character '\0'. An entry with the first character as '\0' indicates an unused or empty directory entry. After implementation of the fix, the ls command was tested, and it was confirmed that only valid directory entries were being returned by fs_readdir.

```
// check if the directory entry is being used, iterate until you find a use entry
while (dirPath->directory[dirPath->dirEntryPosition].fileName[0] == '\0')
{
    //printf("Inside while loop\n");
    dirPath->dirEntryPosition++;
    if (dirPath->dirEntryPosition >= directoryEntries)
    {
        return NULL;
    }
}
```

Issue 9

The program encountered linkage errors and was not running as expected due to global variables being declared directly in header files without the use of the extern keyword. This practice led to multiple definitions of the same variable, which violates the One Definition Rule of the programming language C, causing conflicts at the linking stage. This issue made the program fail to compile and run.

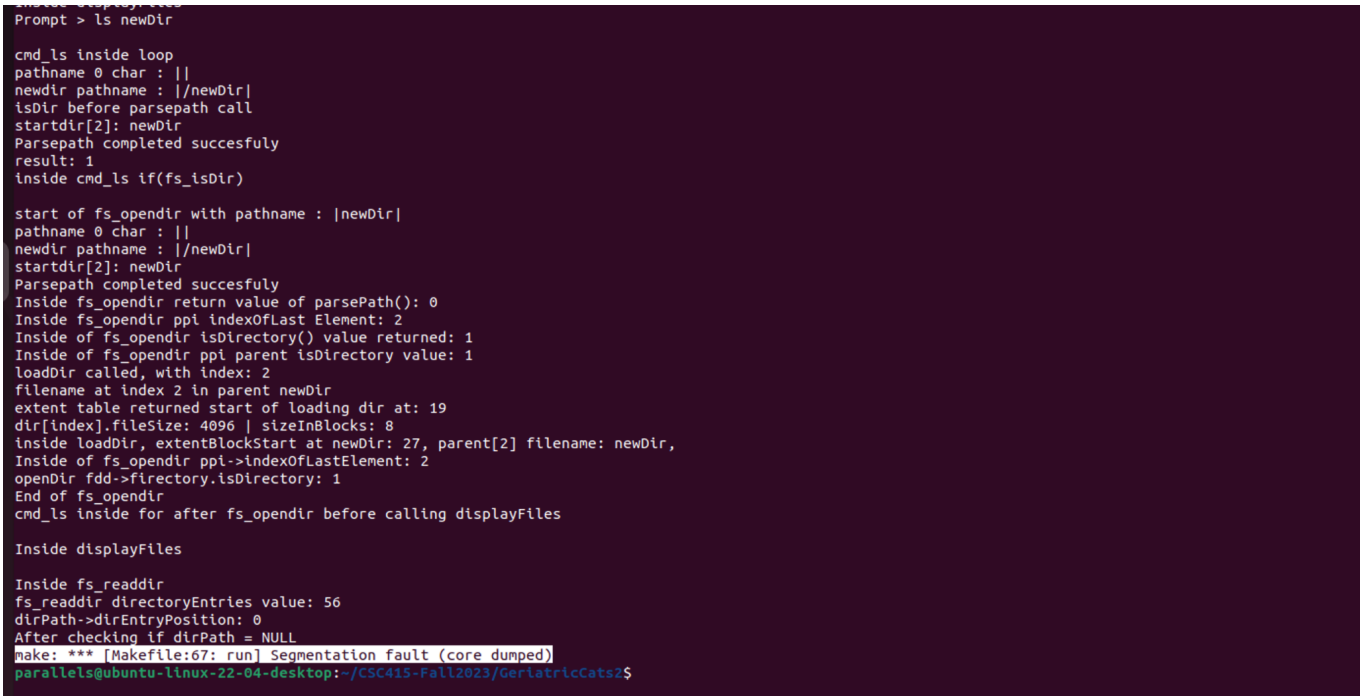
```
parallels@ubuntu-linux-22-04-desktop:~/CSC415-Fall2023/GeriatricCats2$ make run
gcc -c -o fsshell.o fsshell.c -g -I.
gcc -c -o freespace.o freespace.c -g -I.
gcc -c -o directoryEntry.o directoryEntry.c -g -I.
gcc -c -o mfs.o mfs.c -g -I.
gcc -c -o b_io.o b_io.c -g -I.
gcc -o fsshell fsshell.o fsInit.o vcb.o freespace.o directoryEntry.o mfs.o mfsHelper.o b_io.o fsLowM1.o -g -I. -lm -l readline -l pthread
/usr/bin/ld: freespace.o:/home/parallels/CSC415-Fall2023/GeriatricCats2/freespace.h:14: multiple definition of `bytesInBitmap'; fsshell.o:/home/parallel
s/CSC415-Fall2023/GeriatricCats2/./freespace.h:14: first defined here
/usr/bin/ld: freespace.o:/home/parallels/CSC415-Fall2023/GeriatricCats2/freespace.h:15: multiple definition of `bitmapGlobal'; fsshell.o:/home/parallels
/CSC415-Fall2023/GeriatricCats2/./freespace.h:15: first defined here
/usr/bin/ld: directoryEntry.o:/home/parallels/CSC415-Fall2023/GeriatricCats2/directoryEntry.h:13: multiple definition of `rootGlobal'; fsshell.o:/home/p
arallels/CSC415-Fall2023/GeriatricCats2/./directoryEntry.h:13: first defined here
/usr/bin/ld: directoryEntry.o:/home/parallels/CSC415-Fall2023/GeriatricCats2/./freespace.h:14: multiple definition of `bytesInBitmap'; fsshell.o:/home/p
arallels/CSC415-Fall2023/GeriatricCats2/./freespace.h:14: first defined here
/usr/bin/ld: directoryEntry.o:/home/parallels/CSC415-Fall2023/GeriatricCats2/./freespace.h:15: multiple definition of `bitmapGlobal'; fsshell.o:/home/pa
rallels/CSC415-Fall2023/GeriatricCats2/./freespace.h:15: first defined here
/usr/bin/ld: mfs.o:/home/parallels/CSC415-Fall2023/GeriatricCats2/./directoryEntry.h:13: multiple definition of `rootGlobal'; fsshell.o:/home/parallels/
CSC415-Fall2023/GeriatricCats2/./directoryEntry.h:13: first defined here
/usr/bin/ld: mfs.o:/home/parallels/CSC415-Fall2023/GeriatricCats2/./freespace.h:14: multiple definition of `bytesInBitmap'; fsshell.o:/home/parallels/CS
C415-Fall2023/GeriatricCats2/./freespace.h:14: first defined here
/usr/bin/ld: mfs.o:/home/parallels/CSC415-Fall2023/GeriatricCats2/./freespace.h:15: multiple definition of `bitmapGlobal'; fsshell.o:/home/parallels/CSC
415-Fall2023/GeriatricCats2/./freespace.h:15: first defined here
/usr/bin/ld: b_io.o:/home/parallels/CSC415-Fall2023/GeriatricCats2/./directoryEntry.h:13: multiple definition of `rootGlobal'; fsshell.o:/home/parallels
/CSC415-Fall2023/GeriatricCats2/./directoryEntry.h:13: first defined here
/usr/bin/ld: b_io.o:/home/parallels/CSC415-Fall2023/GeriatricCats2/./freespace.h:14: multiple definition of `bytesInBitmap'; fsshell.o:/home/parallels/C
SC415-Fall2023/GeriatricCats2/./freespace.h:14: first defined here
/usr/bin/ld: b_io.o:/home/parallels/CSC415-Fall2023/GeriatricCats2/./freespace.h:15: multiple definition of `bitmapGlobal'; fsshell.o:/home/parallels/CS
C415-Fall2023/GeriatricCats2/./freespace.h:15: first defined here
collect2: error: ld returned 1 exit status
make: *** [Makefile:61: fsshell] Error 1
parallels@ubuntu-linux-22-04-desktop:~/CSC415-Fall2023/GeriatricCats2$
```

Resolution:

The build process revealed errors indicating multiple definitions of global variables. This was traced back to the header files where globals were being declared. The global variables were refactored by applying the extern keyword to their declarations in the header files. This signals to the compiler that the variable is defined elsewhere, preventing it from creating a new instance. The actual definitions of the global variables were moved to a single .c file, ensuring that only one instance of each variable exists. After adjusting the global variable declarations and definitions, the program compiled successfully without linkage errors.

Issue 10

During the execution of the fs_readdir function, the program was encountering a segmentation fault. The fault was manifesting when trying to assign values to the file descriptor structure. The pointer fdDir was used without proper initialization, leading to attempts to access unauthorized memory. The lack of memory allocation for the struct fs_diriteminfo meant that the fdDir pointer was not pointing to a valid memory block. Consequently, when the program tried to access or modify the structure's attributes, it led to a segmentation fault, causing the program to crash.



Resolution

Investigation into the crash revealed that the struct fs_diriteminfo was never allocated memory. The code assumed that the memory was already available, which was incorrect. The root cause was the absence of a malloc call to dynamically allocate memory for the structure before its usage.The code was updated to include a malloc call to allocate memory for struct fs_diriteminfo prior to its use. This ensured that fdDir pointed to a valid memory location, preventing the segmentation fault.

```c
struct fs_diriteminfo *fs_readdir(fdDir *dirPath)
{
    // Read the next directory entry from the directory specified by dirPath
    // Update the dirEntryPosition in dirPath to keep track of the position
    // Return NULL if there are no more entries

    printf("\nInside fs_readdir\n");
    int directoryEntries = dirPath->directory->fileSize / sizeof(DE);
    // int directoryEntries = 0;
    printf("fs_readdir directoryEntries value: %d\n", directoryEntries);
    printf("dirPath->dirEntryPosition: %d\n", dirPath->dirEntryPosition);

    if (dirPath == NULL || dirPath->directory == NULL)
    {
        printf("dirPath is NULL\n");
        return NULL;
    }

    printf("After checking if dirPath = NULL\n");

    // check if the directory entry is being used, iterate until you find a use entry
    while (dirPath->directory[dirPath->dirEntryPosition].fileName[0] == '\0')
    {
        //printf("Inside while loop\n");
        dirPath->dirEntryPosition++;
        if (dirPath->dirEntryPosition >= directoryEntries)
        {
            return NULL;
        }
    }

    struct fs_diriteminfo * dii= malloc(sizeof(struct fs_diriteminfo));
    dirPath->di= dii;

    // copy the name of currenct directory to fs_diriteminfo
    strcpy(dirPath->di->d_name, dirPath->directory[dirPath->dirEntryPosition].fileName);
    printf("di->d_name: %s\n", dirPath->di->d_name);

    // check the type of the directory
```

Milestone 2:

We met twice/week for two weeks on Mondays, Wednesdays, or Friday. Meetings were scheduled in-person in the library reserve study rooms, with a laptop hooked up to the tv for in person, and using the Discord screen share and audio/video lounge for whoever could not attend in person. Our first meeting was spent going over all project code/stubs provided so that we could diagram out everything that needed to happen.

First, we had to finish the remaining work from milestone 1. We realized that we had to finish releaseBlocks() and initExtent() (to set an extent table for each directory) from milestone 1, which we did not use then but would need for milestone 2. In our second meeting, we separated all of our files properly into their own .c files and headers.

We all realized that the proper parsePath implementation and helper functions would be the initial core of the second milestone, so we as a group stubbed functions with various comments on what each should do. To write the basic algorithm as a model for how other functions would work, one of us wrote

full mk_dir while testing the parsePath and the helper functions called by parsePath. After the parsePath and all of our mfsHelper functions were complete, the remaining directory operations open_dir()/read_dir()/close_dir() were given to one member to write, and stat and most remaining functions in from mfs.c were passed on to another member.

The last portion required turning on and testing every command line function from fsshell.c to see if there was any remaining work in milestone 2.

Milestone 2 Work Table

Carlos- Work planned ratio 0.25%, Actual Work ratio %, time spent hrs, complete tasks %
tasks: fs_opendir(), fs_readdir(), fs_closedir()

Jaewan- Work planned ratio 0.25%, Actual Work ratio %, time spent hrs, complete tasks %
tasks: fs_stat(), fs_rmdir(), markDirUnused(), fs_delete(), findentryindir, isdir, isfile, isDirEmpty, fs_move

Paige- Work planned ratio 0.25%, Actual Work ratio %, time spent hrs 27, complete tasks %
tasks: releaseblocks() 1 hrs, initExtent() 3 hrs, parsePath() 1 hrs, loadExtent() 1 hrs, loaddir() 1 hrs, mk_dir() 11 hrs, writeExtent() 1 hrs, writeDir 1hrs, pathUpdate() 1 hrs, addtl fs_setcwd() 5 hours.

Randale- Work planned ratio 0.25%, Actual Work ratio 12%, time spent hrs, complete tasks 60%
tasks: fs_setcwd(), fs_getcwd()

Overall work hours:

// MileStone 2 Done

Milestone 3:

1) cp

```
Prompt > cp file dir
passed O_RDONLY!
b_open received filename: file
pathname 0 char : |
newdir pathname : |/file|
newPath: /file
startdir[2]: dir
Parsepath completed succesfuly
pathValidity: 0, ppi->indexOfLastElement: 4
condition to load a file!
passed O_WRONLY!
passed O_CREAT!
passed O_TRUNC!
b_open received filename: dir
pathname 0 char : |
newdir pathname : |/dir|
newPath: /dir
startdir[2]: dir
Parsepath completed succesfuly
pathValidity: 0, ppi->indexOfLastElement: 2
pathname is a directory.
passed fd= -1, count: 0
```

2) touch

```
Prompt > touch hello
passed O_WRONLY!
passed O_CREAT!
b_open received filename: hello
pathname 0 char : |
newdir pathname : |/hello|
newPath: /hello
startdir[2]:
Parsepath completed succesfuly
pathValidity: 0, ppi->indexOfLastElement: -1
condition to create a file!
nextAvailable in directory: 2
filename: |hello| added for dir[2]
Error writing directory information
```

******* Error writing directory information (comes from b_close LBAwrite handling) *******

3) cat

```
Prompt > touch file
passed O_WRONLY!
passed O_CREAT!
b_open received filename: file
pathname 0 char : |
newdir pathname : |/file|
newPath: /file
startdir[2]:
Parsepath completed succesfuly
pathValidity: 0, ppi->indexOfLastElement: -1
condition to create a file!
nextAvailable in directory: 2
filename: |file| added for dir[2]
Error writing directory information
Prompt > cat file
passed O_RDONLY!
b_open received filename: file
pathname 0 char : |
newdir pathname : |/file|
newPath: /file
startdir[2]: file
Parsepath completed succesfuly
pathValidity: 0, ppi->indexOfLastElement: 2
condition to load a file!
Prompt >
```

4) cp2l

```
Prompt > cp2l
Usage: cp2l srcfile [Linuxdestfile]
Prompt > cp2l file /Documents/
passed O_RDONLY!
b_open received filename: file
pathname 0 char : |
newdir pathname : |/file|
newPath: /file
startdir[2]: hi
Parsepath completed succesfuly
pathValidity: 0, ppi->indexOfLastElement: -1
file not found, no create permissions
Prompt > touch file
passed O_WRONLY!
passed O_CREAT!
b_open received filename: file
pathname 0 char : |
newdir pathname : |/file|
newPath: /file
startdir[2]: hi
Parsepath completed succesfuly
pathValidity: 0, ppi->indexOfLastElement: -1
condition to create a file!
nextAvailable in directory: 3
filename: |file| added for dir[3]
Error writing directory information
Prompt > cp2l file Documents/
passed O_RDONLY!
b_open received filename: file
pathname 0 char : |
newdir pathname : |/file|
newPath: /file
startdir[2]: hi
Parsepath completed succesfuly
pathValidity: 0, ppi->indexOfLastElement: 3
condition to load a file!
Prompt >
```

5) cp2fs

```
Prompt > cp2fs
Usage: cp2fs Linuxsrcfile [destfile]
Prompt > cp2fs Documents/ file
passed O_WRONLY!
passed O_CREAT!
passed O_TRUNC!
b_open received filename: file
pathname 0 char : |
newdir pathname : |/file|
newPath: /file
startdir[2]: hi
Parsepath completed succesfuly
pathValidity: 0, ppi->indexOfLastElement: -1
condition to create a file!
Directory at capacity.
passed fd= -1, count: -1
Prompt >
```

6) mv

```
Prompt > mv hello hi
startdir[2]: hello
Parsepath completed succesfuly
pathname 0 char : |
newdir pathname : |/hello|
startdir[2]: hello
Parsepath completed succesfuly
startdir[2]: hello
Parsepath completed succesfuly
pathname 0 char : |
newdir pathname : |/hi|
isDir before parsepath call
startdir[2]: hello
Parsepath completed succesfuly
result: 1
pathname 0 char : |
newdir pathname : |/hello|
pathname 0 char : |
newdir pathname : |/hi|
File moved successfully.
```

7) history

```
|---------------------------------|
|------- Command ------|- Status -|
| ls                   |    ON    |
| cd                   |    ON    |
| md                   |    ON    |
| pwd                  |    ON    |
| touch                |    ON    |
| cat                  |    ON    |
| rm                   |    ON    |
| cp                   |    ON    |
| mv                   |    ON    |
| cp2fs                |    ON    |
| cp2l                 |    ON    |
|---------------------------------|
Prompt > help
ls      Lists the file in a directory
cp      Copies a file - source [dest]
mv      Moves a file - source dest
md      Make a new directory
rm      Removes a file or directory
touch   Touches/Creates a file
cat     Limited version of cat that displace the file to the console
cp2l    Copies a file from the test file system to the linux file system
cp2fs   Copies a file from the Linux file system to the test file system
cd      Changes directory
pwd     Prints the working directory
history Prints out the history
help    Prints out help
```

8) help

```
Prompt > history
help
history
```

Issue 1:

failed to handle the proper blockCount and block location when utilizing LBAwrite, a function called inside the b_close(fd), which triggers errors in multiple command lines above.

```
    // If the file was opened with write permissions, update the file system structures
    if ((fcbArray[fd].permissions & O_WRONLY) == O_WRONLY || (fcbArray[fd].permissions & O_RDWR) == O_RDWR)
    {
/*
        int dir = (sizeof(DE) * DEFAULT_ENTRIES + BLOCK_SIZE -1)/ BLOCK_SIZE;

        if (LBAwrite(fcbArray[fd].parent, dir, fcbArray[fd].parent[fcbArray[fd].dirIndex].extentBlockStart) != dir) {
            printf("Error writing directory information\n");
            return -1;
        }

        int ext = (sizeof(EXTTABLE) * DEFAULT_ENTRIES + BLOCK_SIZE -1) / BLOCK_SIZE;

        // int ext = (sizeof(EXTTABLE) * DEFAULT_ENTRIES + BLOCK_SIZE -1) / BLOCK_SIZE;
        if (LBAwrite(fcbArray[fd].parentExtent, ext, fcbArray[fd].parentExtent[fcbArray[fd].dirIndex].tableArray[0].start) != ext) {
            printf("Error writing extent table information\n");
            return -1;
        }
*/

        // Write directory information back to the file system
        if (LBAwrite(fcbArray[fd].parent, sizeof(DE), fcbArray[fd].parent[fcbArray[fd].dirIndex].extentIndex) != 1) {
            // Handle error in writing directory information
            printf("Error writing directory information\n");
            return -1;
        }

        // Write extent table information back to the file system
        if (LBAwrite(fcbArray[fd].parentExtent, sizeof(EXTTABLE), fcbArray[fd].parentExtent[fcbArray[fd].dirIndex].tableArray[0].start) != 1) {
            // Handle error in writing extent table information
            printf("Error writing extent table information\n");
            return -1;
        }
    }
```

→ This is the most critical problem that our shell does not work properly.

Description:

The driver program handles operations like opening, reading, writing, seeking, and closing files in a buffered process.

1. **Initialization**: The program starts with the b_init function, which initializes an array of file control blocks. Each FCB represents an open file and contains information like file descriptor, permissions, and buffer for read/write operations.
2. **Opening Files**: The b_open function is used to open files. It supports various flags (e.g., read-only, write-only, read-write, create, truncate) similar to the Linux open system call. It handles file path parsing, checks for validity, and determines whether to create a new file or load an existing one. It allocates an FCB for the opened file and prepares it for subsequent operations.
3. **Reading and Writing**: The b_read and b_write functions handle the reading and writing of data to and from files, respectively. They operate with a buffered approach, meaning they read/write data in chunks (blocks) for efficiency. The read function includes logic to handle different scenarios of reading from the current buffer, directly in block size multiples, or from a refilled buffer. The write function similarly handles appending data, allocating new blocks if needed, and writing in block-sized chunks.
4. **Seeking**: The b_seek function allows repositioning the file pointer within a file, supporting different modes like SEEK_SET, SEEK_CUR, and SEEK_END.
5. **Closing Files**: The b_close function is responsible for closing open files. It ensures any modified data is written back to the underlying file system structures, releases allocated resources, and resets the FCB.