CSC415 - 02 WriteUp

Team Name: Something Simple

Team Members:

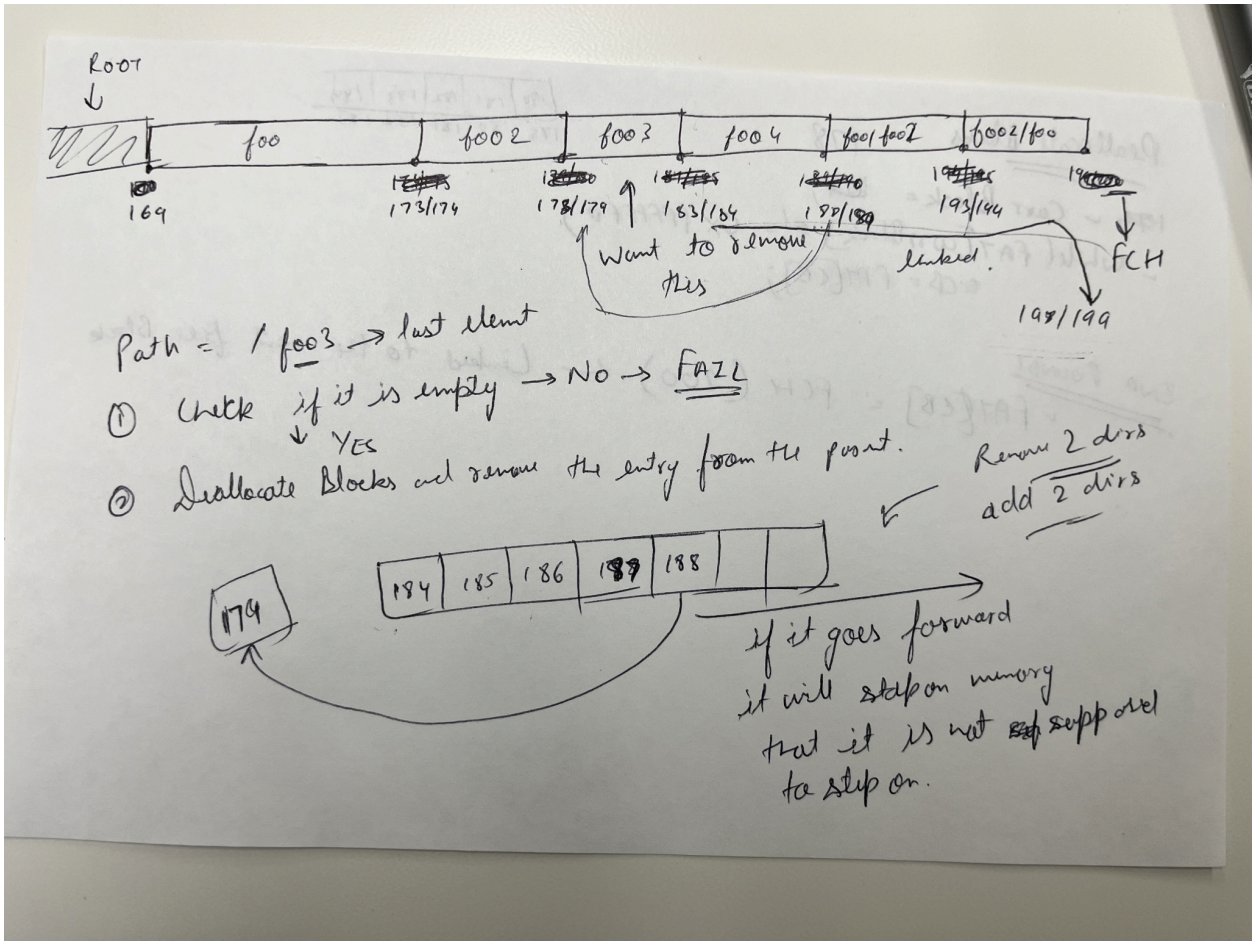
Name	Student ID	GitHub Username
Tushin Kulshreshtha	922180763	Dextron04
Hayden Coke	921741974	crowcode17
Eric Khuong	923406338	ekhuong
Thanh Duong	922438176	DanielDoubleDx

[Repository Link: github crowcode17]

Plans

Milestone1

The initial approach for the start of our file system project was to divide the task among the 4 group members which included the 2 main parts of milestone 1 which were initializing the FAT and creating the Root Directory, and we divided these 2 tasks such that 2 group members can work on each of the assigned task. The setting up of FAT was done by Tushin and Hayden, and the creation of the root directory was done by Eric and Thanh.



Milestone2

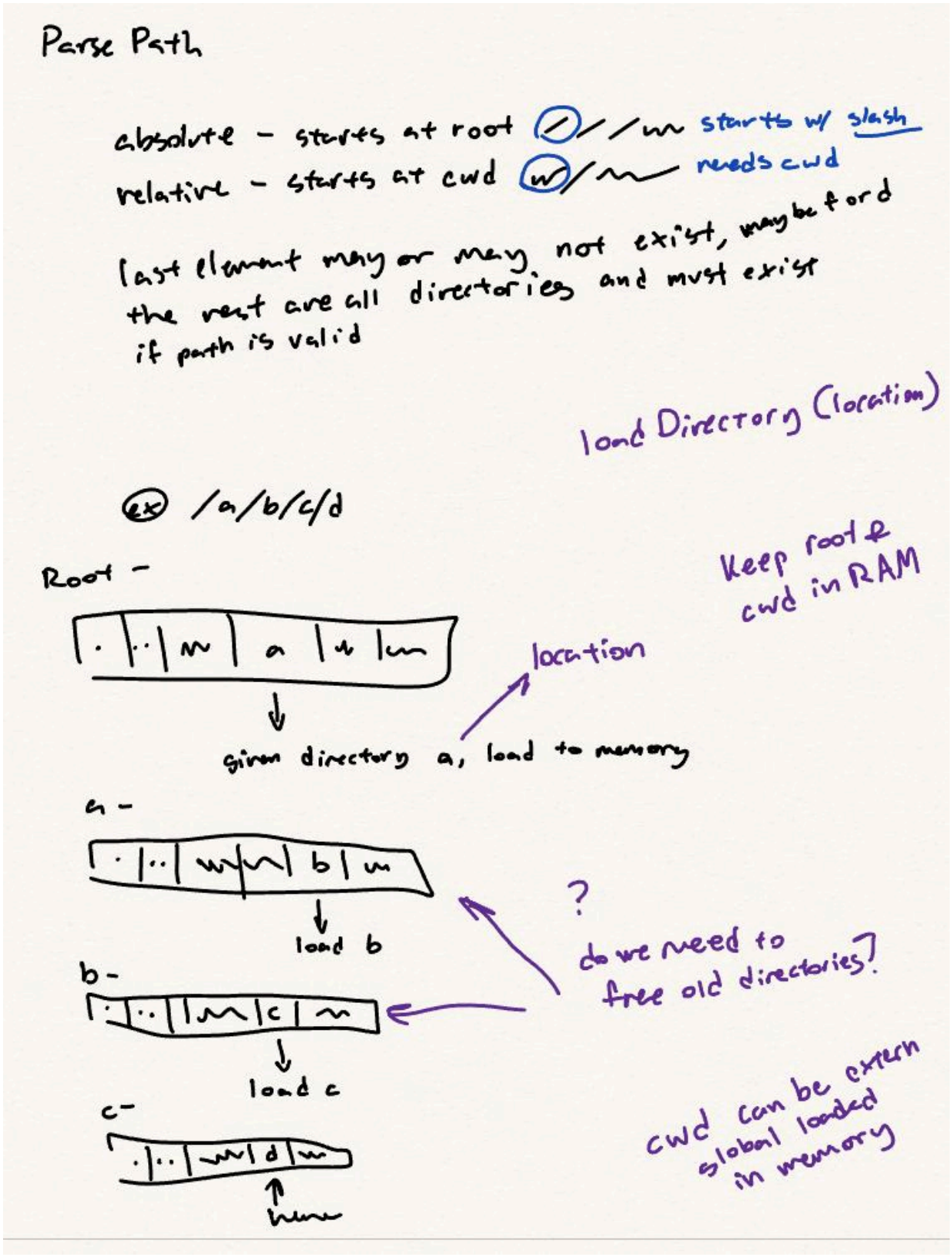
The first thing we wanted to get done was parsePath, because almost all directory functions would utilize it. Because of its complexity, we all worked on parsePath together.

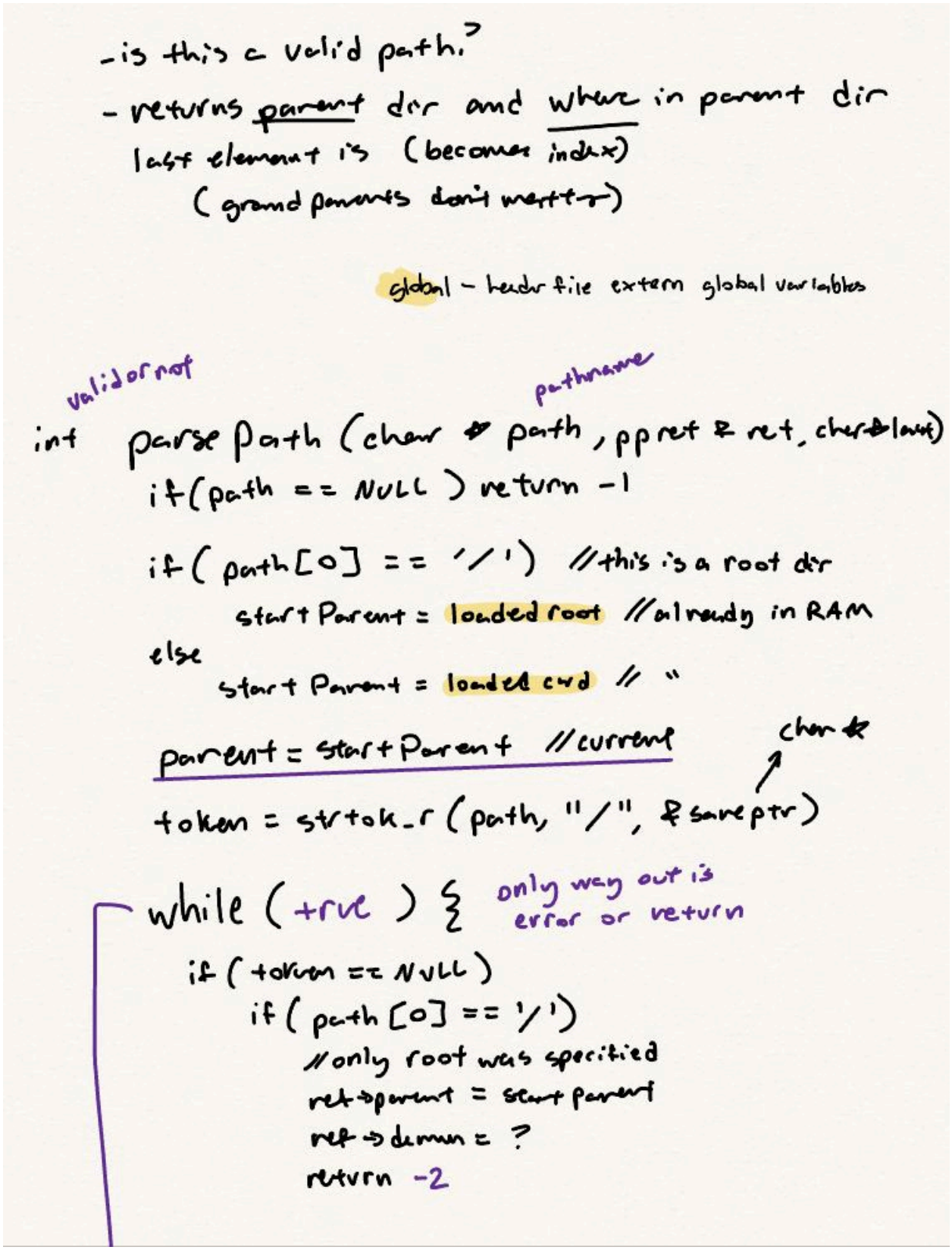
Once parsePath was done, we divided up the directory functions:

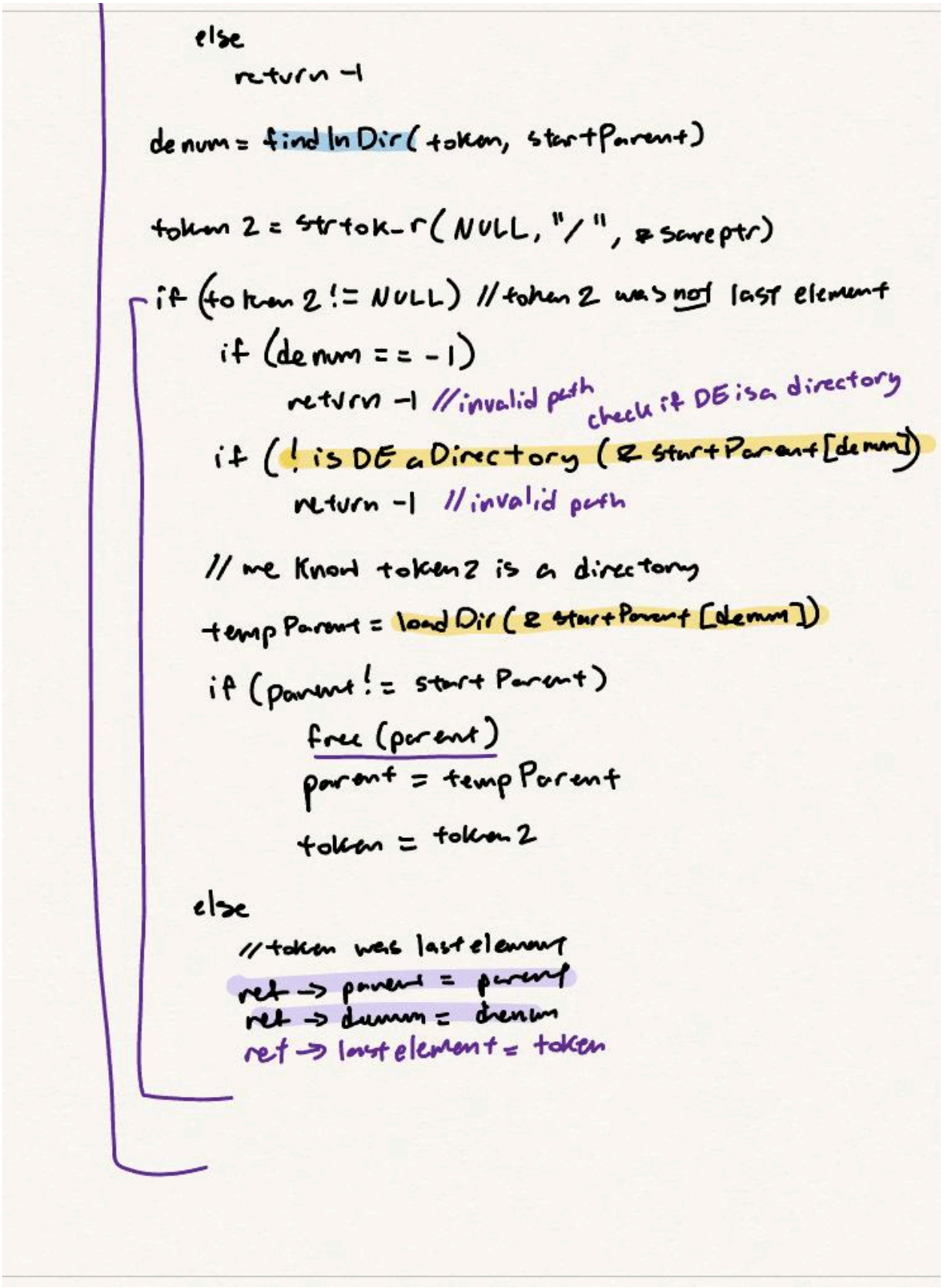
- Eric and Hayden worked on getcwd and setcwd.
- Tushin worked on fs_opendir, fs_mkdir, fs_readdir, fs_stat, fs_isFile and fs_isDirectory fs_closedir.
- Thanh and Tushin worked on rmdir and fs_delete.

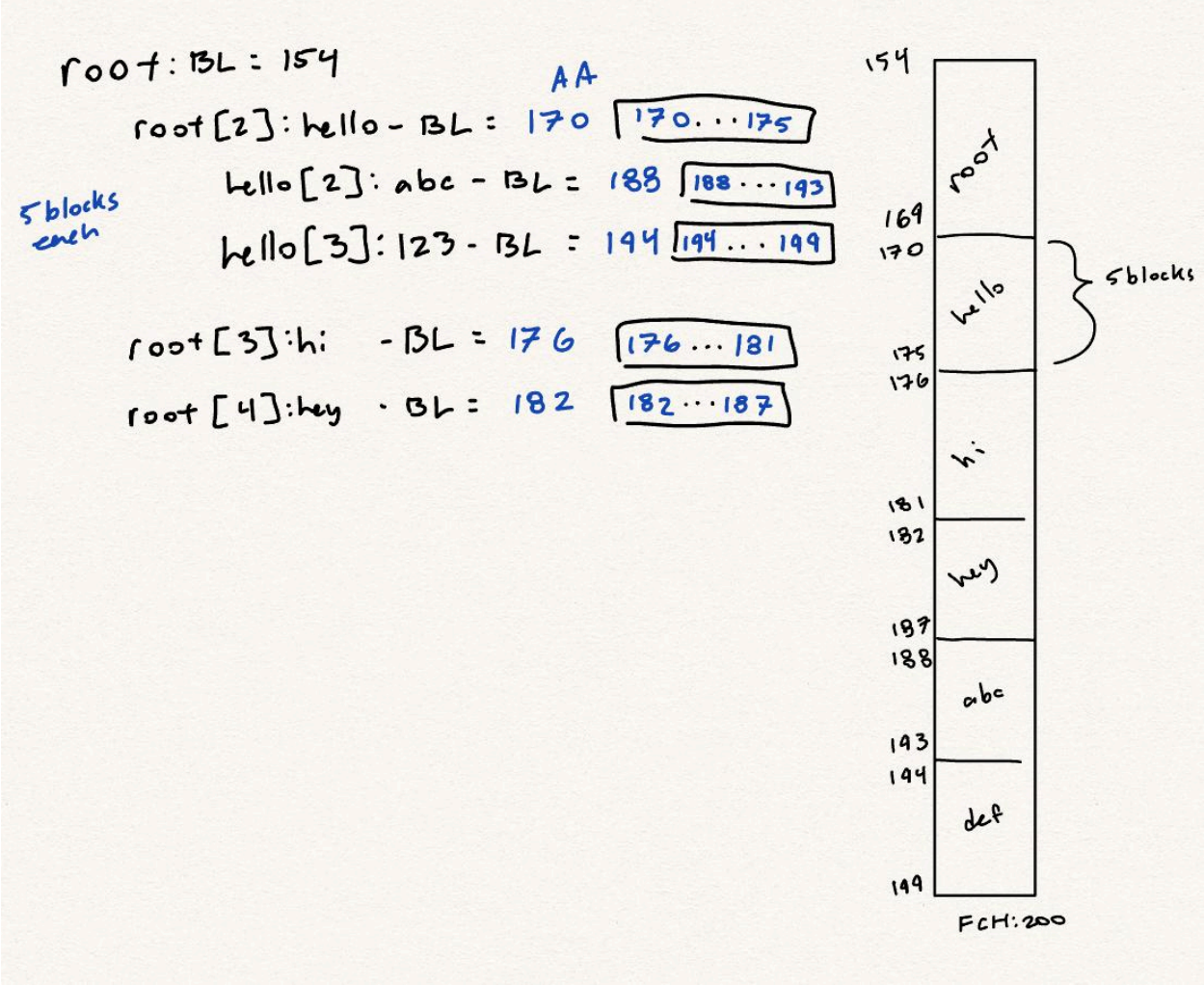
We all worked together to test and debug.

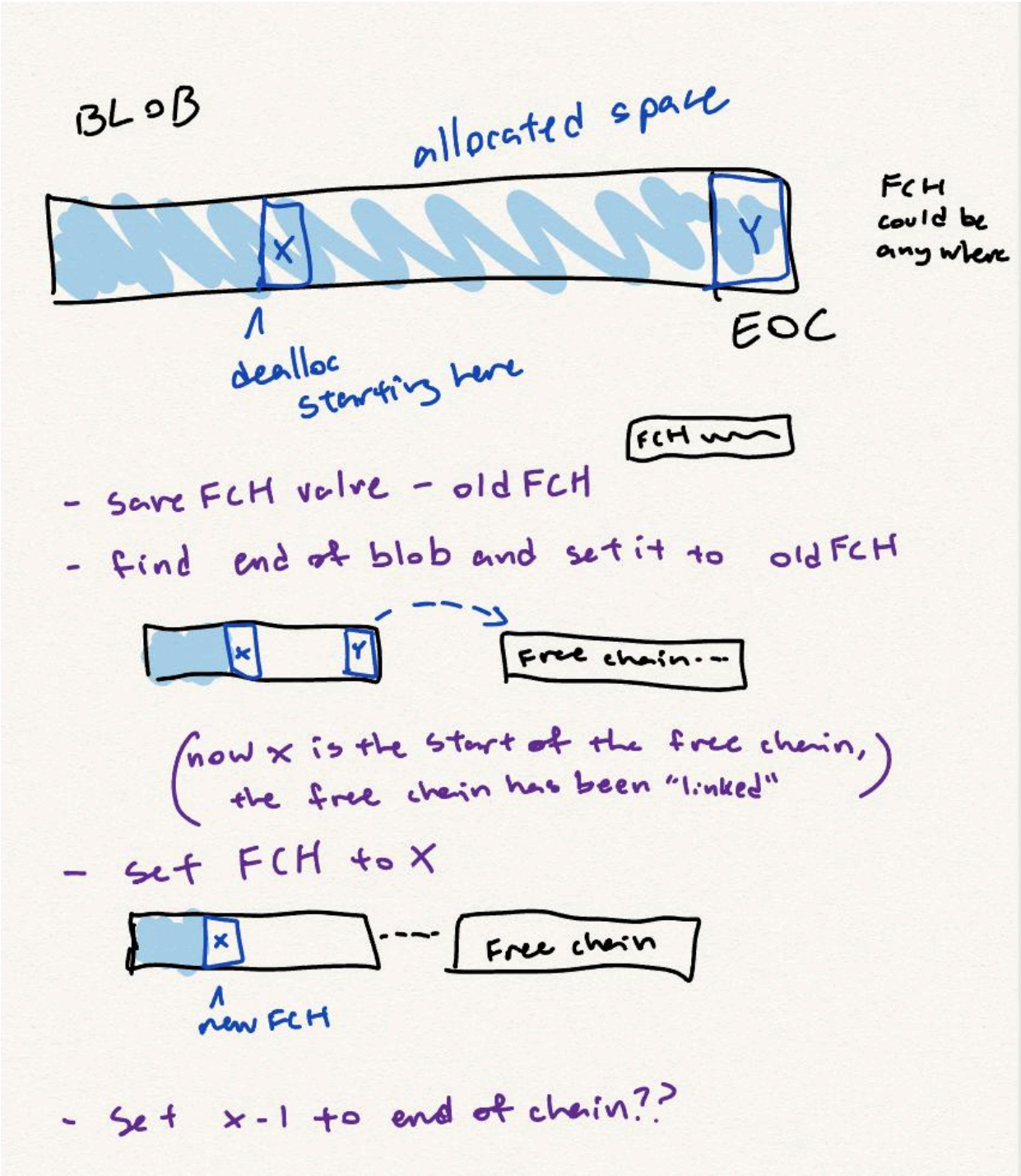
Hayden's planning notes for milestone 2 attached:

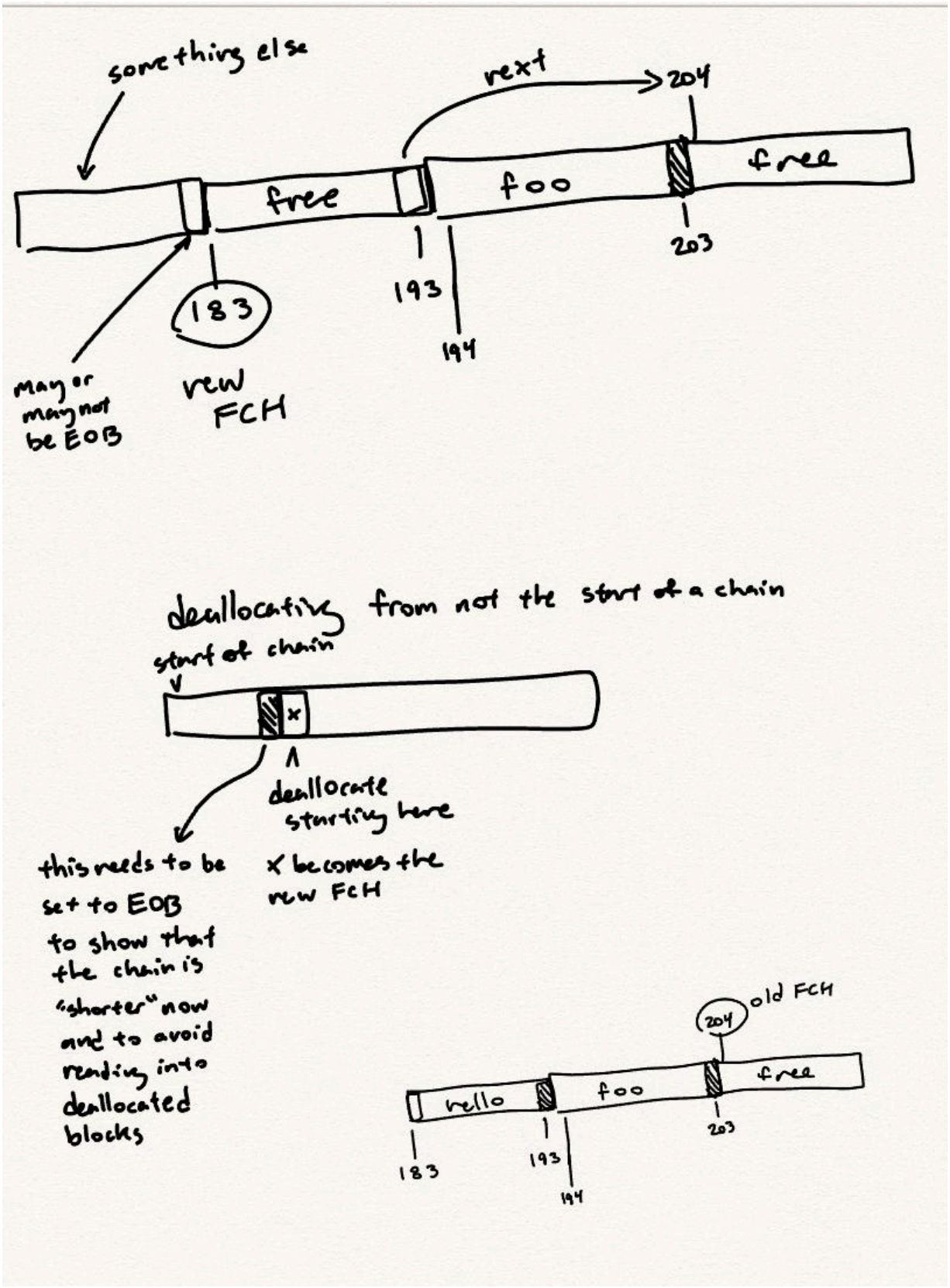



- is this a valid path?
- returns parent dir and where in parent dir
last element is (becomes index)
(grandparents don't matter)
global - header file extern global variables
valid or not
pathname
int parsePath (char * path, ppret & ret, char* last)
if(path == NULL) return -1
if(path[0] == '/') //this is a root dir
startParent = loaded root //already in RAM
else
startParent = loaded cwd //
parent = startParent //current
char *
token = strtok_r(path, "/", &saveptr)
while (true) { only way out is error or return
if (token == NULL)
if (path[0] == '/')
//only root was specified
ret->parent = startParent
ret->dirname = ?
return -2


else
return -1
denum = findInDir(token, startParent)
token2 = strtok_r(NULL, "/", &scureptr)
if (token2 != NULL) //token2 was not last element
if (denum == -1)
return -1 //invalid path
check if DE is a directory
if (!isDEaDirectory(&startParent[denum])
return -1 //invalid path
// we know token2 is a directory
tempParent = loadDir(&startParent[denum])
if (parent != startParent)
free(parent)
parent = tempParent
token = token2
else
//token was last element
ret -> parent = parent
ret -> dnum = denum
ret -> lastelement = token


root: BL = 154
AA
root[2]: hello - BL = 170
170...175
hello[2]: abc - BL = 188
188...193
hello[3]: 123 - BL = 144
144...149
5 blocks each
root[3]: hi - BL = 176
176...181
root[4]: hey - BL = 182
182...187
154
root
169
170
hello
5 blocks
175
176
hi
181
182
hey
187
188
abc
193
144
def
149
FCH: 200

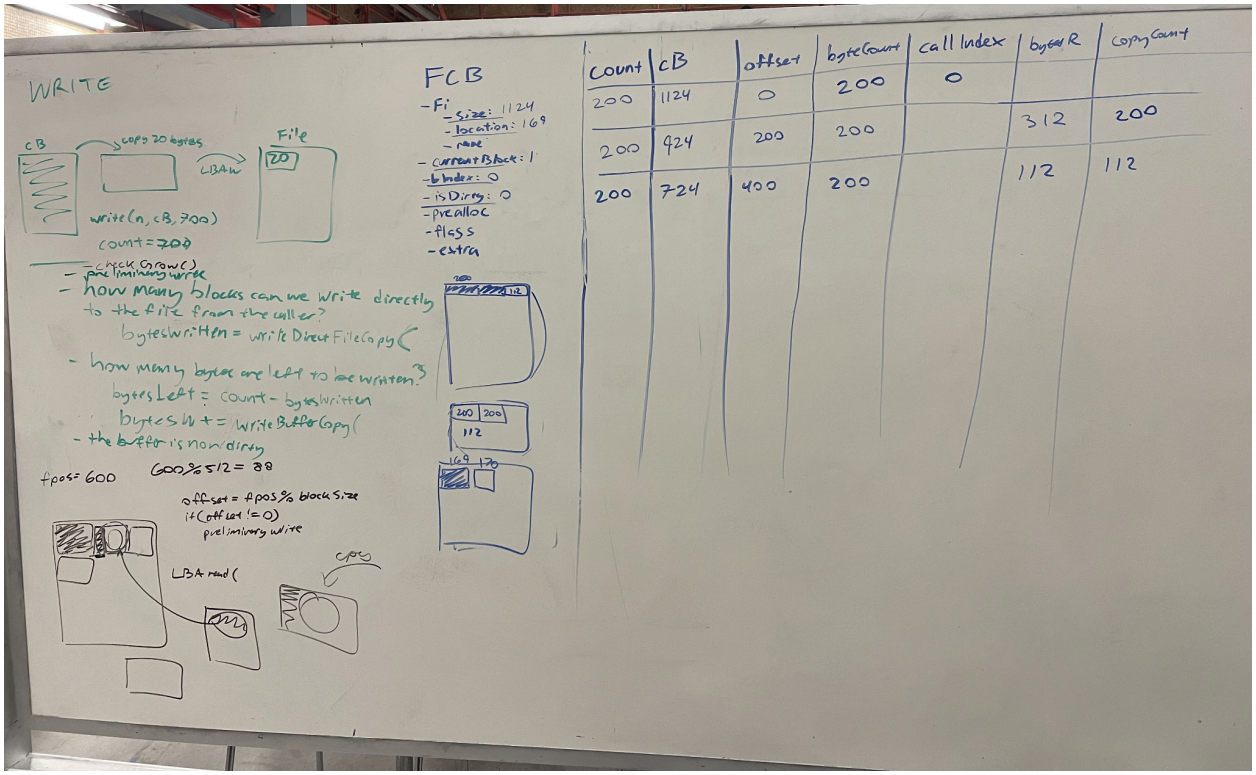

BLOB
allocated space
X
Y
FCH could be any where
EOC
dealloc starting here
FCH
- save FCH valve - old FCH
- find end of blob and set it to old FCH
X
Y
Free chain...
(now x is the start of the free chain, the free chain has been "linked")
- set FCH to X
X
Free chain
new FCH
- Set x-1 to end of chain??


something else
next
204
free
foo
free
203
183
193
194
may or may not be EOB
new FCH
deallocating from not the start of a chain
start of chain
x
deallocate starting here
this needs to be set to EOB to show that the chain is "shorter" now and to avoid reading into deallocated blocks
x becomes the new FCH
old FCH
204
hello
foo
free
183
193
194
203

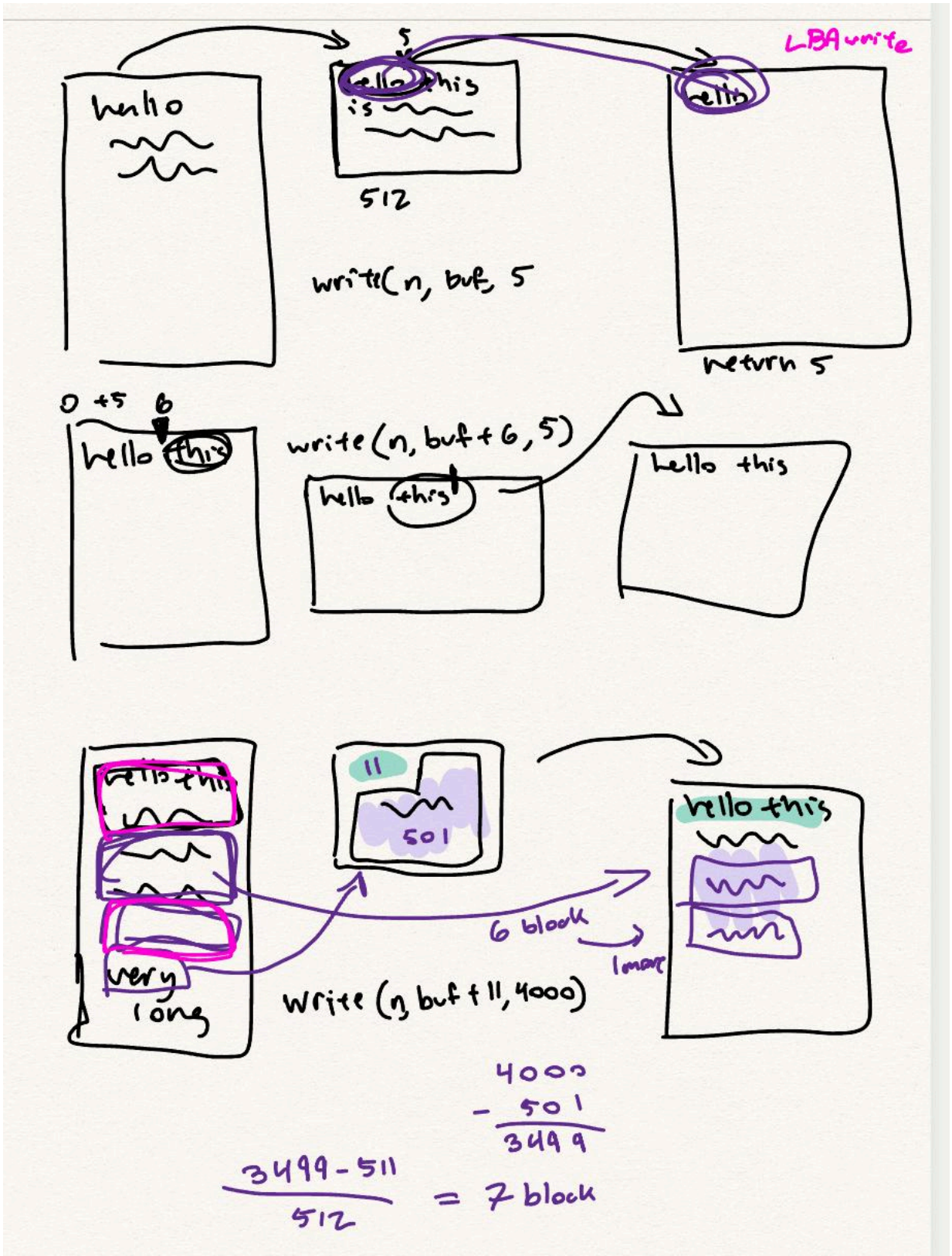
Milestone3

Our first goal for milestone 3 was to transfer over the read function from Assignment 5. Hayden planned to copy over his code and adjust it to the rest of the project. From that point, we planned to do the write function using read as a baseline for how the logic should work. Instead of dividing things up, we all worked on write, open, and close together. Once those were done, Tushin and Hayden planned out and worked on move and seek.

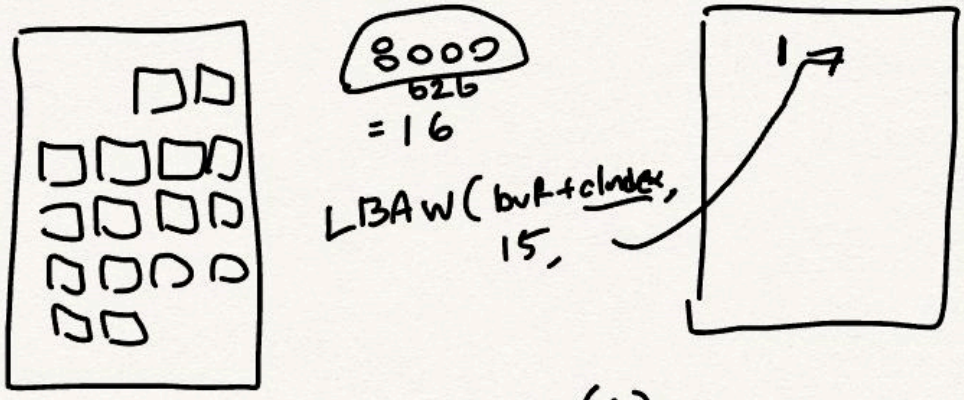
Write planning from group work session in the CS lab:



Hayden's planning notes for milestone 3:

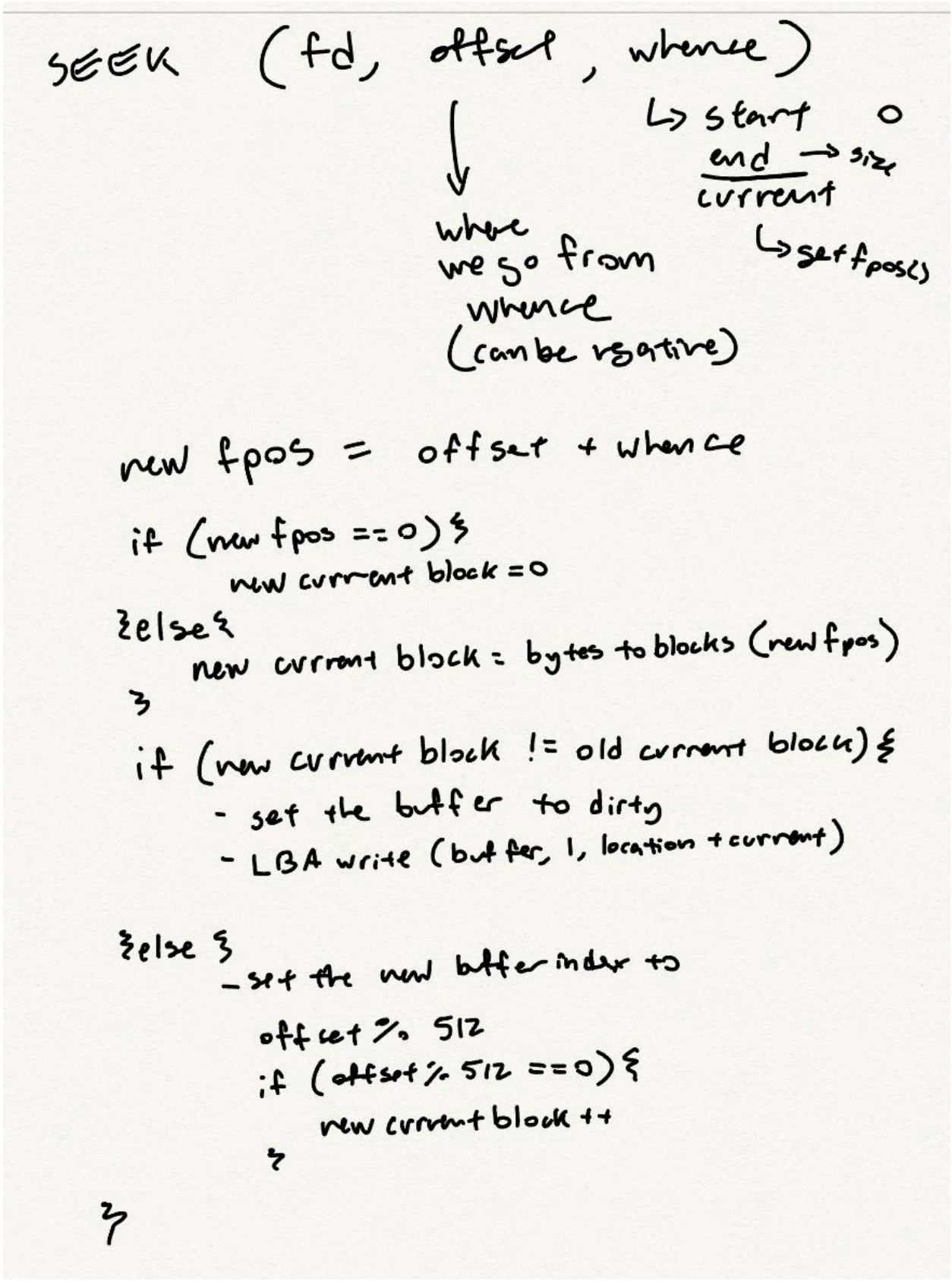


direct copy from caller buffer to file

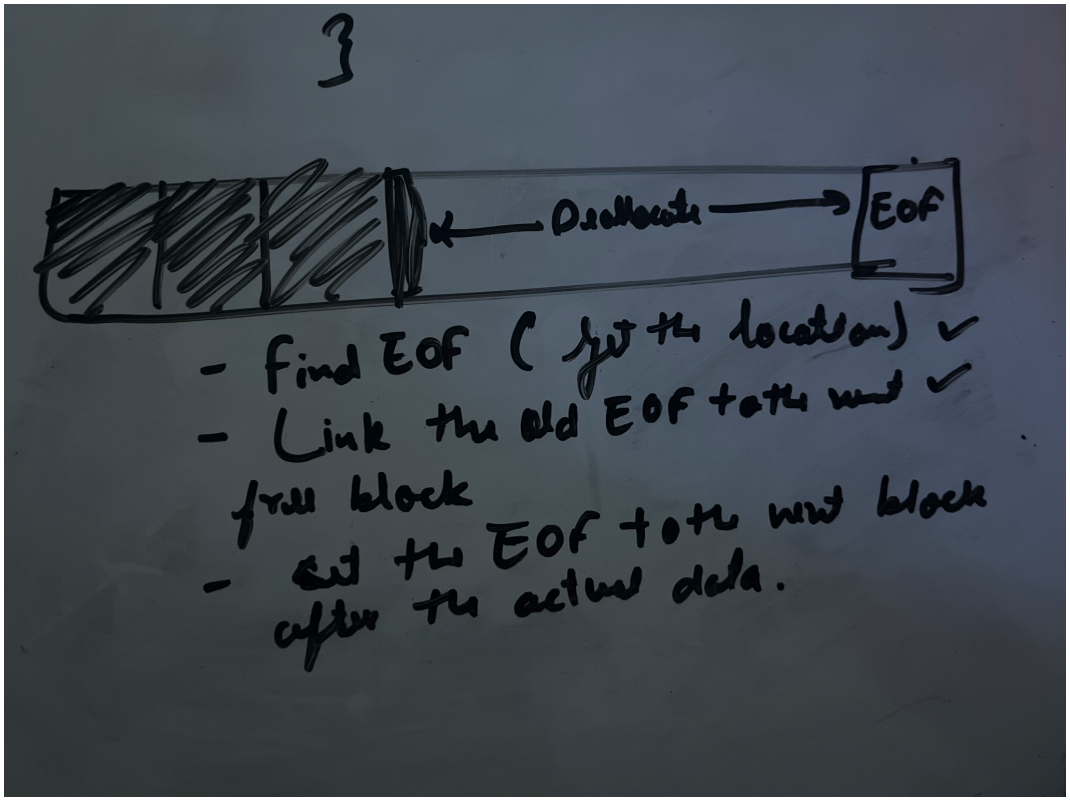


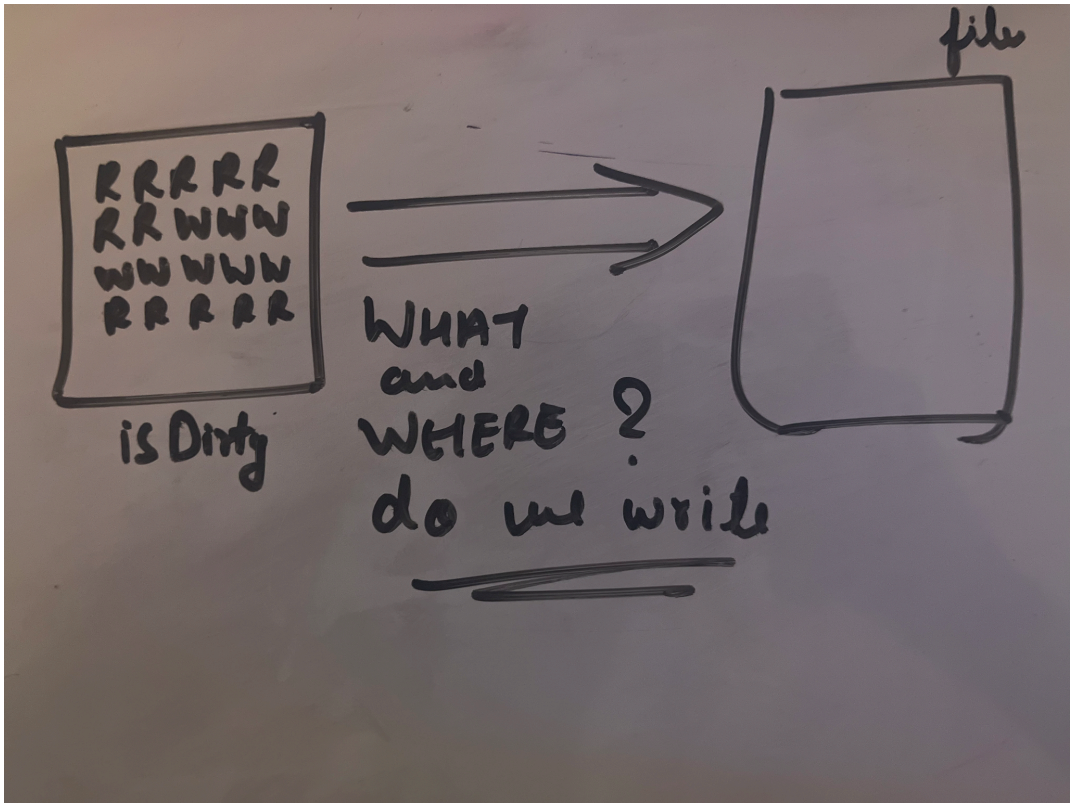
- set block amount (b)
- LBA write (buf + cinder, b-1, loc)
- fcb -> current block += b - 1
- bytes W = 512 * (b-1)

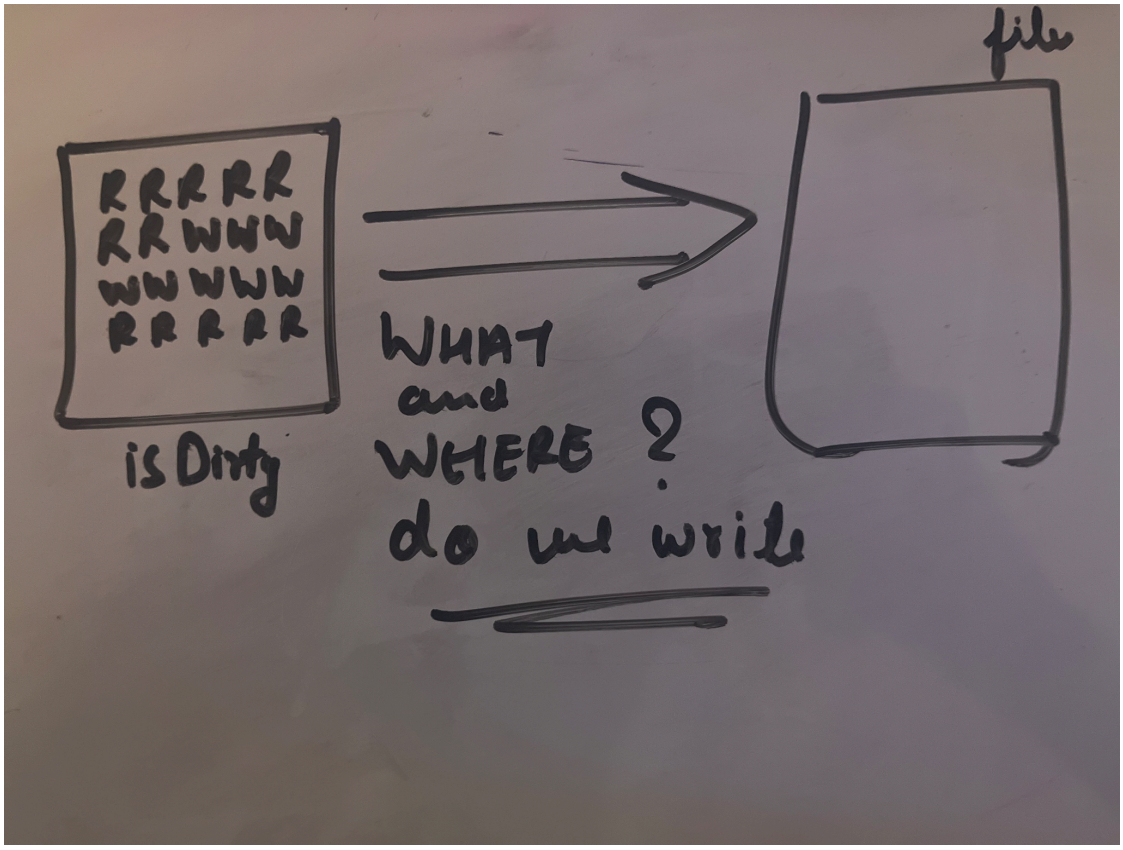
- if (byteCount - bytesWritten == 512)
  - LBA write
  - fcb -> current +1
  - bytes W += 512
- return bytes W

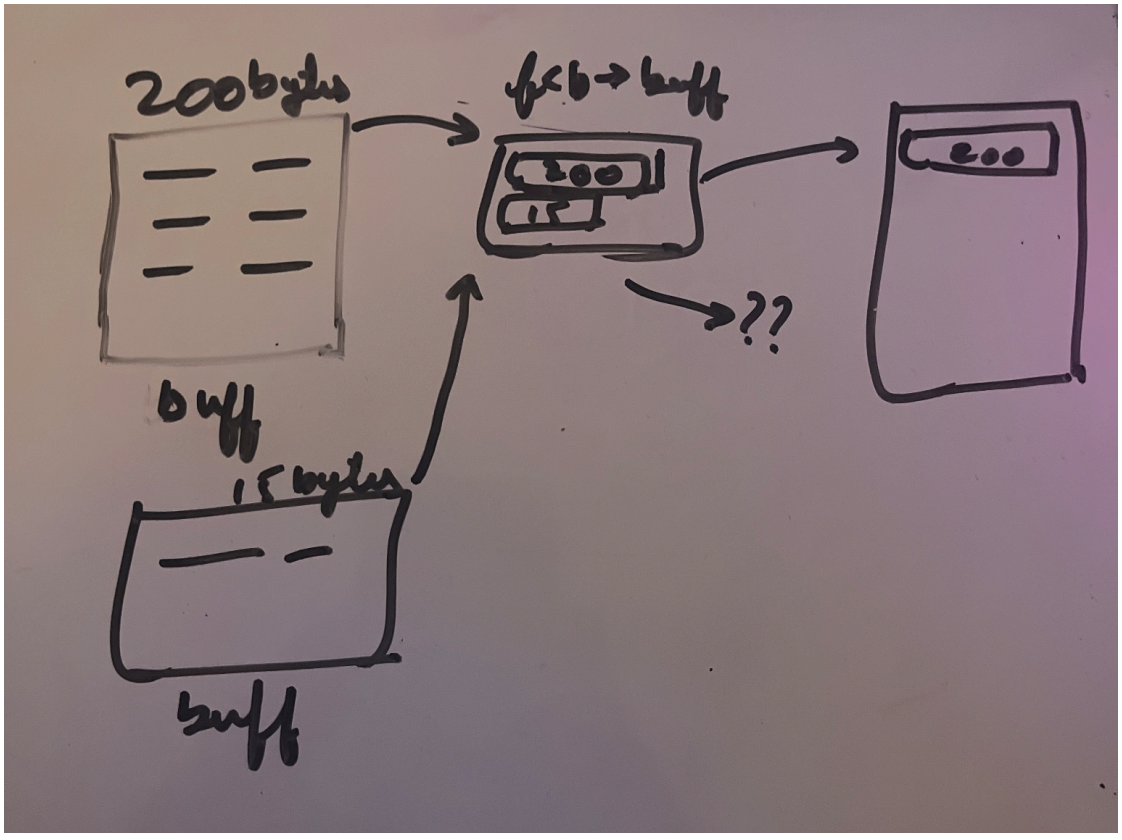

SEEK (fd, offset, whence)
start 0
end → size
current
where we go from
whence
(can be negative)
↳ setfpos()
new fpos = offset + whence
if (new fpos == 0) {
new current block = 0
}else{
new current block = bytes to blocks (new fpos)
}
if (new current block != old current block) {
- set the buffer to dirty
- LBA write (buffer, 1, location + current)
}else {
- set the new buffer index to
offset % 512
if (offset % 512 == 0) {
new current block ++
}
}

Tushin's planning notes from milestone 3:






file
R R R R
R R W W W
W W W W W
R R R R R
WHAT
and
WHERE ?
is Dirty


200bytes
fcb→buff
200
15
200
??
buff
15 bytes
buff

File System Description

Our basic file system provides the user with the basic functionality needed to manage files and directories. Using a terminal prompt, the user can create, navigate, remove, and move directories. Files can be created using touch and written to by copying a source file from linux to the file system, displayed with cat, removed from the file system, moved or copied within the file system, and copied back out to linux. If the user wants to modify files within the directory, they must write code to do so using our b_open, b_read, b_write, b_seek, and b_close functions. All data stored on our file system is persistent, meaning that as long as the volume file (in our case, “SampleVolume”) exists, we can exit and re-run the file system program and read data from previous runs.

To internally manage the file system, our Volume Control Block keeps track of the volume signature, the total amount of blocks in the volume, the location and size of the root directory, the location of the FAT, and the block location of the free chain head. For our free space management we implemented a File Allocation Table that keeps track of files using chains of blocks, allowing us to discontinuously allocate files. Free space is its own chain of blocks, the “free chain.”

Issues

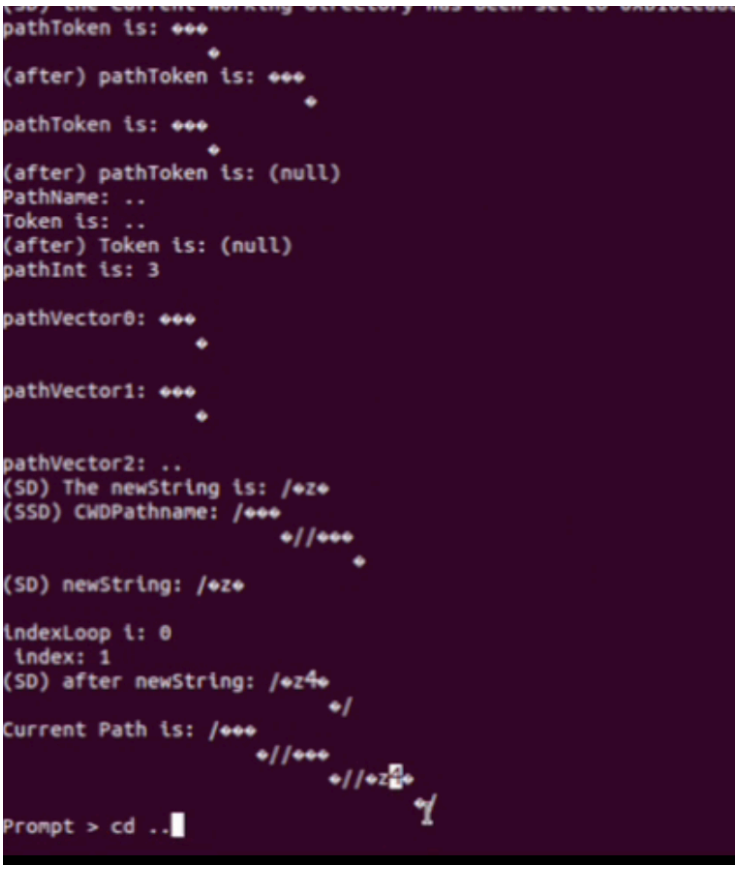
One of the more significant issues we encountered when constructing our File System was when we found out that we were overwriting other directories with our LBAwrite. This issue initially formed when we created our createDirectory function for milestone 1 as we accidentally used direcBytes instead of directoryBlocks in our LBAwrite and we didn’t catch onto this for the longest time. We didn’t catch on as when we were testing to ensure that it worked we never had to create an entry in front of another entry so it didn’t matter if it wrote to the next 5000 blocks as they would never be in use. We only managed to discover that there was something wrong after we created rmdir for milestone 2 as when we removed a directory freeing up space in front of another and then created another directory in that no longer used space the directory afterward would be wiped from the disc. Since we created createDirectory for milestone 1 and rmdir for milestone 2 we thought that there was an issue in the code we made for milestone 2 that caused this and believed that our milestone 1 code was perfectly fine with no errors whatsoever. This meant that we ended up checking everything except for the code that the issue was actually in. It was only when we had an outside eye take a look at our code (aka the professor) that it was found out as he didn’t have the same assumptions our group had about the milestone 1 code. After this it was just a simple changing of direcBytes to directoryBlocks in the LBAwrite to fix the issue and not overwrite other entries.

Issue: getting burnt out in the CS lab after working for 6 hours straight.

Resolution: going outside and sitting in the grass for half an hour.

Issue: We were having problems when changing the directory as it was showing up as gibberish and was not able to change the directory successfully.

Resolution: We fixed the issue by doing a memset to that path to a 0 which cleared up any gibberish or garbage that was inside the disk.



Issue: During write, all of our files seemed to be cut off on an even block boundary, so if there were any bytes at the end of a file that didn't take up a whole block they wouldn't get written.

Resolution: After hours of debugging and tracing through our write, we realized we just weren't writing our dirty buffer to the file in close.

Issue: Garbage at the beginning of the file in read appearing as either gibberish or red squares.

Resolution: Skipped the preliminary buffer read if the buffer was invalid, such as if it was dirty or empty.

createDirectory

The createDirectory function accepts an int describing the number of entries wanted and a Directory Entry pointer to a parent that can be null. The desired number of entries is then used to determine the actual amount of entries that can fit inside of the block amount needed for the desired amount. Using this new actual amount of entries we get the total amount of bytes and put that in direcBytes, put the block amount in directoryBlocks, call allocateBlocks and put that result in directoryLocation, create a Directory Entry and malloc it byte amount of the blocks, and initialize each entry in our created directory. After this, we set the 0th entry of the directory to the self directory with a . name. If the parent is null we use the self directory to also fill out the 1st entry of the directory as it represents the parent of the self directory and having a parent of null means that it'll be the root directory which parent is itself. If the parent isn't null the 1st entry is filled with the information of the parent and the modification date of the parent will be modified to reflect this newly created entry. In either case, the entry will be marked as a Directory and assigned a name of .. before being saved with saveDir returning with the directory afterwards.

initializeFAT

The initialize FAT function runs in the initialization part when the signature doesn't match with our signature (i.e. SSimple) and it runs the **initializeVCB** function which in turn runs the initializeFAT function. The function starts off by taking in the input of the total number of blocks inside the volume and since each integer represents the location of a block in the disk space we convert the totalBlocks times the size of int (**bytesToBlocks(sizeof(int) * totalBlocks)**) which will give us the required number of blocks that will be used up by the FAT in the disk. Then we malloc the FAT array with the number of bytes that will be used up by the FAT and the malloc is done in full block amounts (i.e. 512 bytes in our case). The first entry in the FAT is set to END_OF_BLOB (i.e. 0xFFFFFFFD) to mark that the first block has been used by the VCB. Furthermore, a for loop initializes the FAT and links each block to another through FAT[i] = i + 1; which makes the array to an external linked list. Then the last entry of the FAT and the last point in the disk space is marked as END_OF_BLOB to mark the end. Finally, the freeChainHead is set to the next block after the FAT and saved in the VCB. (So, FAT uses 153 blocks, freeChainHead is at 154) and then the **saveFAT** function is executed to save the FAT array to the disk and to finish the initialization the function returns the value of the freeChainHead which signifies successful initialization of the FAT.

fs_setcwd

Our fs_setcwd function accepts one char pointer string to describe the path we will use to set the current working directory. Checks will be performed on this path to determine if it's null or just the root. If it's null it will return immediately with a -1 failing to change the current working directory and if it's just the root it'll check if the current working directory is already the root and if not free the cwd, assign the cwd as the root, change the path name of the cwd to just the root, and return with 0. If neither of

these checks passes the function strdup's the path, creates a ppReturn to hold parsePath results, and then sends this duplicate to parsePath. After returning from parsePath it'll check if the last element of ppReturn is the root or not. If it is the root then it'll set the cwd to root, set the cwd path to /, free the cwd if it isn't already the root, and return with 0 but if it isn't then it will check to see if the path is actually there and if it's a valid directory to set as returning -1 if not. If it is a valid directory it will then load that directory as the cwd, free the cwd if it isn't currently the root, and then return by calling buildString which will provide the pathstring of the new cwd.

fs_getcwd

The fs_getcwd function works by accepting a buffer and a size. Using this size it'll check if it'll fit in our buffer as if the size is greater than our maximum allowed path size or if the string length of the current working directory path is greater than our given size it won't fit. If it doesn't fit in our buffer then the function will fail and return with null but if it will the path to the current working directory (CWDPathname) will be memcpy'd into the buffer and returned.

fs_isFile

The fs_isFile function accepts a char pointer string path to a specified file/directory. It will duplicate this string using strdup, create a ppReturn structure to store the results from parsePath, and then send this duplicated string to parsePath to determine if it's a valid path. We then check if the returned value from parsePath is any number other than 0 and if it is so the function will return with -1 as it means that parsePath has failed. If parsePath succeeds then we will check our ppReturn structure to see if the given path is valid returning with -1 if not. If this also succeeds the function will finally check whether or not the boolean that determines if a specified Directory Entry is a directory or not is marked as true or false returning the opposite of said result.

fs_isDir

The function fs_isDir performs a check to determine whether a given pathname corresponds to a directory. First, it creates a duplicate of the input pathname to ensure that the original input remains unchanged. Then, it invokes parsePath function from fileSystem.h, passing the duplicated pathname and a pointer to a ppReturn structure to store parsing the results. If the return value of parsePath is not zero, indicating an error in parsing the path, the function returns signifying failure. Next, it checks if the deNum attribute of the parsing results structure is -1, which would mean that the path does not exist in the file system. Otherwise, it accesses the isDirectory attribute of the directory entry specified by deNum within the parent array in the ret structure and returns its value, indicating whether the path corresponds to a directory or not.

fs_mkDir

The fs_mkdir function serves the purpose of creating a new directory within the file system. It starts by duplication the input path name to prevent modifications to the original string. Then, it invokes the parsePath function to parse the pathname and retrieve information about the parent directory where the new directory will be created. If parsing fails, the function returns -1 to indicate an error. Subsequently, it checks if the directory already exists by examining the deNum attribute in the parsing results. If a directory with the same name already exists, the function returns -1 to prevent duplication. Assuming parsing and existence checks succeed, the function proceeds to create the new directory using the createDirectory function, which allocates space for the directory and initializes its attributes such as location, size, date created, and date modified. After successfully creating the directory, it finds an empty space in the parent directory using the findSpaceInDir function, which locates an empty slot within a directory. It links the newly created directory to the parent by updating the directory entry with relevant information. Finally, the changes are saved to FAT, and the parent directory is updated and saved to disk using saveFAT and saveDir functions.

fs_openDir

The fs_openDir function is responsible for opening directories in the file system. It begins by initializing a fdDir structure to hold directory-related information. The function takes the pathname of the directory to be opened as input. It then parses the provided path to extract relevant information about the directory's location and parent directories. If the parse operation indicates an error, the function returns NULL and indicates failure. Assuming successful parsing, the function proceeds to load the directory specified by the pathname. If the parsed path indicates the root directory, it loads the root directory directly from the file system. Otherwise, it loads the directory specified by the parsed path from its parent directory. This ensures that the function accurately retrieves the directory to be opened. Once the directory is loaded, the function populates the fdDir structure with relevant information, such as the directory's entry count, the directory entry's position within the directory, and a pointer to the loaded directory itself.

fs_readDir

The fs_readDir function is designed to read directory entries sequentially from a directory stream represented by the dirp parameter, which is a pointer to a structure containing directory-related information. The function first allocates memory for an array of fs_diriteminfo structures, each representing a directory entry. The size of this array is determined by the entryCount field of the dirp structure, ensuring that enough memory is allocated to hold all directory entries. Next, the function iterates through the directory entries starting from the dirEntryPosition within the directory. It checks each entry to determine if it is valid. If a valid entry is found, its information is copied to the corresponding fs_diriteminfo structure in the allocated array. Additionally, the dirEntryPosition is

updated to point to the next directory entry in the sequence. The function returns a pointer to the populated fs_diriteminfo structure, allowing the caller to access information about the directory entry. It will continue reading directory entries sequentially from where the previous call left off. If there are no more directory entries to read, the function returns NULL.

fs_closeDir

The fs_closedir function is used to close a directory stream represented by the dirp parameter, which is a pointer to a structure containing information about the directory. When calling, the function first performs checks to ensure that the directory being closed is not the root directory or the current working directory, as closing these directories is not permitted and could lead to unintended consequences. If either of these conditions is met, the function returns an error code -2 to indicate the failure to close the directory. Assuming the directory being closed is not the root or current working directory, the function proceeds to free the memory allocated for the directory structure itself. Additionally, if the directory structure contains dynamically allocated memory for directory items, it is also freed to prevent memory leaks. Finally, the memory allocated for the dirp structure itself is freed, and the pointer is set to NULL to prevent accidental access to deallocated memory. The function then performs a simple validation check to ensure that the directory pointer is indeed set to NULL after deallocation. If dirp is NUll, it indicates successful closure of the directory stream, and the function returns 0. However, if dirp is not NULL, it suggests that the closure process encountered an error or that the directory pointer was not properly deallocated, resulting in a return value of -1 to indicate the failure to close the directory.

fs_stat

The fs_stat function will accept a const char pointer string path to a specified file/directory along with a fs_stat structure to store the information in. These two variables will then be checked to determine if either of them is null returning with -1 if so. Continuing on from this, a ppReturn structure will be created to store parsePath results and the path will be strdup'd and sent into parsePath. If parsePath encounters an error and returns with -1 or if the path isn't valid the function will return with -1 failing to fill the fs_stat structure with info but if parsePath doesn't fail and the path is valid then the passed-in fs_stat structure will store the Directory Entry information of byte size, block amount, block size, creation time, and last modified time returning with 0 afterward to indicate success.

fs_delete

The fs_delete function accepts a char pointer string path to a file to delete. It works by first creating a ppReturn structure to store parsePath results and using strdup on the path sending them both into parsePath. The returned value from parsePath and the deNum will then be checked to determine if parsePath experienced an error and if the path is valid or not returning with -1 if either statement is true.

A final check will then be performed to determine if the entry at the given location is a directory or a file returning -1 if directory and continuing with the delete if not. To delete the file deallocateBlocks is then called with the location of the entries blocks and all the values of the entry are set back to their original values to indicate that it is no longer in use. These changes then get saved in the parent directory and the VCB with saveDir and writeVCB completing the delete and returning with 0 to indicate successful delete.

fs_rmDir

The fs_rmDir function accepts a const char pointer to a path that should lead to a directory the user wants to remove. This path is strdup'd and a ppReturn structure is created calling parsePath. The returned value from parsePath along with the deNum and the Directory Entry at said location are then checked to ensure that parsePath didn't encounter an error, the path is valid, and the entry at that location is actually a directory and not a file returning with -1 if any of these are true. If all are false it continues on creating a DirectoryEntry and loading in the directory the path leads to with loadDirectory. If loadDirectory fails and it loads in null the function returns with -1 but if it succeeds variables are created to contain the entryCount, dirLocation, and dirSize. Using these variables it loops through the entire DirectoryEntry to determine if it's completely empty outside of the . and .. that are automatically created. If it isn't empty then fs_rmDir will fail returning with -1 and if it is it'll call deallocateBlocks with the dirLocation, set the entry it takes up in the parent back to unused, free the directory we loaded, save the parent with saveDir, write to the VCB with writeVCB, and return with 0.

b _open

The b_open function accepts a char pointer string and an int to determine the name of the file along with the flags that are to be passed in. Initially upon first being called the function checks to ensure that our system has already been initialized doing so if it hasn't and continuing on with attempting to get a valid file descriptor by calling b_getFCB afterward. If b_getFCB returns with an invalid file descriptor the function returns with -1 as all the FCB's are in use but if it returns with a valid one the function assigns to returnFd then goes on to create a mask that can be either O_RDONLY, O_WRONLY, or O_RDWR indicating the file to be opened is read only, write only, or read and write. This mask is then compared with the passed in flags to obtain the bits of interest and these bits of interest are subsequently anded with the mask to obtain only the read and write flag locations for the file storing them in permissionMask. With permissionMask we then and it with the actual passed in flags to then determine as to whether or not the file is O_RDONLY, O_WRONLY, or O_RDWR returning with -1 if none are true. Next, the flags are anded with O_CREAT, O_APPEND, and O_TRUNC to figure out which one of the three it is. After determining all of this the function creates a ppReturn structure, a fileInfo pointer malloced with the size of fileInfo, and an int fileIndex along with strduping the filename to pass into parsePath. If parsePath fails and returns with a -1 b_open also returns with -1 as the path given is invalid but if it is a valid path it will check to see if the deNum in the ppReturn structure is set to -1 and if the handler is marked as O_CREAT.

If it's set to -1 and marked as O_CREAT it will signify that the file is not currently present in the parent and should be created calling createNewFile. The return value from this is assigned to fileIndex and is then saved to the parent with saveDir if possible returning with -1 if not. If this check fails it'll mean that the file is in our directory already and fileIndex is just set to it instead of having to create it. Using this fileIndex the information in the parent is stored in the fileInfo pointer and a buffer is created and malloced. If this malloc fails the functions returns with -1 but if everything works out fine it creates a fcb, checks if the handler is O_CREAT setting the preallocated block amount in the fcb to the initial preallocated amount if so, assigns the created fcb to the fcbArray at returnFd, and then returns with the returnFd.

b_read

The b_read function serves as the interface for reading data from a file. It starts by checking if the file system has been initialized. If not, it initializes the system using b_init() to ensure it is ready for operation. Then, it validates the file descriptor fd to ensure it falls within the valid range, which is between 0 and MAXFCBS - 1. If the file descriptor is invalid, it returns -1 to indicate an error. Next, it checks the permissions of the file associated with the file descriptor. If the file does not have read permissions O_RDONLY or O_RDWR, it prints an error message and returns -1. To prevent reading past the end of the file, the function calculates the number of unread bytes in the file fileUnreadBytes. It limits the number of bytes to read byteCount to the minimum of the requested count or the number of unread bytes. The function proceeds to read data from the file using three different methods. It attempts to read data from the file buffer readPreliminaryBufferCopy. This buffer contains data that has been previously read from the file and may satisfy some or all of the requested bytes. If there are still bytes remaining to be read after the preliminary buffer copy, it checks if any blocks can be read directly from the file readDirectFileCopy. This method reads data directly from the file into the caller's buffer in block-sized chunks. If there are still bytes remaining to be read after the first two methods, it reads data into the file buffer and then copies the remaining bytes to the caller's buffer readSecondaryBufferCopy. Finally the function tracks the total number of bytes read bytesRead and returns this value as its output.

b_readHelpers

The readPreliminaryBufferCopy function is a step to read data from a file, particularly when the file buffer already contains some data. The function begins by checking if the file buffer is empty or marked as dirty. If either condition is met, indicating that the buffer is not valid for reading, the function returns 0, indicating that no bytes were read. If the buffer is valid for reading, the function calculates the number of bytes remaining in the buffer bytesRemaining by subtracting the current buffer index from the buffer size. It then determines the number of bytes to copy from the buffer to the caller's buffer. This is determined as the minimum of the requested byte count byteCount and the number of bytes remaining in the buffer. This ensures that the function does not attempt to copy more bytes than are available in the buffer. Using memcpy, the function copies the determined number of bytes from the file buffer

starting from the current buffer index to the caller's buffer. This transfers the data from the file buffer to the caller's buffer. After copying the data, the function updates the buffer index by adding the number of bytes copied copyCount. If the buffer index reaches the end of the buffer, it resets the index to 0 and increments the current block index. This ensures that the function operates correctly when reading spans multiple blocks. Finally, the function returns the number of bytes copied copyCount to indicate the amount of data successfully read from the file buffer.

The readDirectFileCopy efficiently reads data directly from the file into the caller's buffer without the need for an intermediate buffer. The function begins by checking if the byteCount parameter is non-zero, ensuring that there is data to be read. If byteCount is zero, indicating that no data needs to be read, the function returns 0 immediately. Next, the function calculates the number of blocks required to read the specified number of bytes byteCount. This calculation is performed using the bytesToBlocks helper function. If the byte count is not a multiple of the block size BlockSize, the number of blocks is adjusted to ensure that only complete blocks are read. Using the LBAread function, the function reads data directly from the disk into the caller's buffer. The data is read starting from the specified callIndex within the caller's buffer. The number of blocks to read and the location from which to read are passed as parameters to LBAread. After reading the data, the function updates the current block index by adding the number of blocks read. This ensures that subsequent reads start from the correct block. Finally, the function calculates the total number of bytes read by multiplying the number of blocks read by the block size. This value is returned to the caller, indicating the amount of data successfully read from the file.

The readSecondaryBufferCopy function serves the purpose of reading data from the file into an intermediate buffer and then copying the required bytes from this buffer to the caller's buffer. The function starts by checking if the byteCount parameter is non-zero, ensuring that there is data to be read. If byteCount is zero, indicating that no data needs to be read, the function returns 0 immediately. Using the LBAread function, the function reads a single block of data from the disk into the file buffer. The block is read from the location specified by getBlockLocation. After reading the block into the file buffer, the function copies the required number of bytes byteCount from the file buffer to the caller's buffer starting from the specified callIndex. This copying operation is performed using the memcpy function. The function updates the buffer index by adding the number of bytes read byteCount. This ensures that subsequent reads start from the correct position within the file buffer. Finally, the function returns the byteCount, indicating the number of bytes successfully read and copied from the file buffer to the caller's buffer.

b_write

The b_write function is used to write data into files within the file system while respecting file permissions and preventing reads beyond the end of the file. It begins by performing initialization if the system startup flag is set to 0, ensuring that the file system is properly initialized before proceeding with

the write operation. Next, it validates the file descriptor fd to ensure it falls within the valid range of file control blocks fcbArray. If the file descriptor is invalid, the function returns -1, indicating an error. Following this, the function checks whether the file associated with the file descriptor has the necessary write permissions. If the file does not have write permissions, an error message is printed, and the function returns -1. It evaluates whether the file was opened with either O_WRONLY or O_RDWR flags. If the file was opened with any other flags or if it was opened with O_RDONLY permissions, the condition triggers an error message indicating that the file does not have the required write permissions, and it returns -1 to signify an error. The function then checks whether the file needs to be grown to accommodate the write operation by calling the checkGrow function. If additional space is required, the growSpace function is called to dynamically allocate the necessary blocks for the file. After ensuring sufficient space, the function proceeds with the actual write operation. It calculates the file position and offset within the current block to determine where to begin writing data. The writePreliminaryBufferCopy function is then called to copy data from the caller's buffer to the file's buffer, ensuring proper alignment and preparation for subsequent write operations. Next, the function calls writeDirectFileCopy to write data directly from the caller's buffer to the file system, bypassing the file's buffer. In the secondary buffer copy stage, any remaining data is written to the file buffer and subsequently flushed to the disk. This stage ensures that all data is safely persisted, even if it cannot be directly written to the file system due to size constraints. The writeSecondaryBufferCopy function manages this operation, copying the remaining data to the file buffer and ensuring its eventual persistence to disk. Throughout the write process, the function tracks the number of bytes written and updates the file's metadata accordingly. Additionally, it ensures that the file's maximum position is updated to reflect the newly written data. Finally, upon completion, the function returns the total number of bytes successfully written.

b_writeHelpers

The writePreliminaryBufferCopy function prepares data for writing by copying it from the caller's buffer to the fcb's buffer. It handles buffer management, data copying, and block writing, ensuring that data is correctly staged for writing to the file system. The function starts by logging a message indicating that it's performing a preliminary buffer copy with the given offset. Using LBAread, the function reads the current block from the file system into the fcb's buffer. This buffer is where the data will be temporarily stored before being written back to the file system. Then, the function calculates the number of bytes remaining in the block bytesRemaining after the provided offset. It then determines the number of bytes to copy copyCount based on whether byteCount exceeds bytesRemaining. If byteCount is larger, it sets copyCount to bytesRemaining, otherwise, it sets it to byteCount. Before copying data, the function logs the number of bytes being copied from the caller's buffer to the fcb's buffer. Using memcpy, it copies copyCount bytes from the caller's buffer to the fcb's buffer, starting from the specified offset. After copying the data, the function writes the contents of the fcb's buffer back to the file system using LBAwrite. It writes the buffer to the block corresponding to the current block index. The function updates the buffer index to reflect the new position in the buffer after the copy operation. It adds the

offset and copyCount to determine the new index. If the buffer index reaches the size of a chunk, it indicates that the buffer is full and needs to be written to the file system. In this case, the buffer index is reset to 0, and the current block index is incremented to move to the next block. Finally, the function returns the copyCount, indicating the number of bytes copied from the caller's buffer to the fcb's buffer.

The writeDirectFileCopy function directly writes data from the caller's buffer to the filesystem. The function begins by checking if the byteCount parameter is non-zero. If byteCount is zero, it immediately returns, indicating that no data needs to be written. Next, the function calculates the number of blocks required to accommodate the specified number of bytes. It does so by converting byteCount into blocks using the bytesToBlocks function. If the byte count is not a multiple of the block size, it adjusts the block count to write a complete number of blocks. Then, it writes blocks number of blocks starting from the callIndex position within the buffer parameter to the file location. After writing the data, the function updates the current block position to reflect the blocks written. Finally, the function calculates the total number of bytes written by multiplying the number of blocks written by the block size.

The writeSecondaryBufferCopy function copies data from the caller’s buffer to the file buffer and then writes it to the filesystem. The function starts by checking if the byteCount parameter is zero. If it is, indicating that no data needs to be copied, the function returns 0, effectively terminating further execution. Next, the function copies data from the caller's buffer starting from the callIndex position into the file control block's buffer. It uses the memcpy function to perform this operation. The byteCount parameter determines the number of bytes to copy. Once the data is copied to the file control block's buffer, the function writes the entire buffer to the file system. It writes the data as a single block to the file location. After writing the data to the file system, the function updates the buffer index to reflect the position within the buffer. If the buffer index reaches the maximum buffer size, indicating that the buffer is full, it resets the buffer index to 0 and increments the current block position. Finally, the function returns the byteCount parameter, indicating the number of bytes copied from the caller's buffer to the file system buffer.

b_close

Our b_close function accepts a b_io_fd which will be used to determine which fcb in our fcbArray will be closed. This fd will be checked to ensure that it’s a valid file descriptor number returning -1 if not and continuing on to check if it’s already open if so. If the file at the specified file descriptor is not open the function will return with -1 and if it is then it will assign it as a b_fcb and check if the buffer is dirty or not. If the buffer is dirty then it’ll clear the memory past the part of the buffer we want to write and then LBAwrite our buffer. After this, it’ll then call shrinkSpace which will shrink the file size removing unneeded preallocated blocks and set the file size to its actual allocated used amount. Finally, the buffer will be freed and the file descriptor set as unused returning with 0 afterward to indicate a successful close.

b_seek

Seek changes the position in the given file based on a given offset and whence, where whence is a starting position in the file (start, end or current) and offset is the amount of bytes away from that position. After determining the actual file position specified by the whence, we calculate a new file position by adding the whence position and the offset. If the user passes something other than SEEK_SET, SEEK_END, or SEEK_CUR (defined by linux), the seek fails. We calculate which block the file will be positioned to based on the new file position. If the user has seeked to a new block, the file buffer becomes dirtied, and is written to the disk if it wasn't dirty already. Otherwise, if the user seeked within the current block, the buffer index is updated to match the new file position.

The user can successfully seek within the file and past the end of the file, but seeking before the beginning of the file is not fully implemented and may give unexpected behavior, as it hasn't been fully tested. If the user seeks past the end of the file, the max file position is not updated until the user calls write, to avoid inaccurately representing the size of the file.

parsePath

The parsePath function is responsible for parsing file paths and locating directory entries, facilitating directory navigation and file access. It begins by checking the length of the provided path to ensure it does not exceed the maximum allowable size, preventing potential buffer overflows. The function then duplicates the path string to ensure safe manipulation without modifying the original input. Next, it determines whether the path is absolute or relative based on the presence of leading "/". If it is an absolute path, parsing starts from the root directory, otherwise, it begins from the current working directory. Using the strtok function, the path is tokenized, splitting it into individual directory names. The function iterates through each token, attempting to locate the corresponding directory entry within the current parent directory using findInDirectory. If a directory entry is found, it checks if the token represents a directory, loading its contents into memory if necessary. During traversal, the function dynamically manages memory by freeing previously loaded directories to prevent memory leaks. This process continues until the function reaches the last element in the path, where it returns information about the parent directory and the directory entry number corresponding to the final token.

allocateBlocks

Our allocateBlocks function works with a passed-in int representing the number of bytes we want to allocate blocks for. To determine where to allocate the blocks it then creates two ints assigning freeChainHead of the vcb to the first of returnLocation and the block amount needed to contain the passed-in bytes in the second of blocks. Using this block amount it will then call findEndOfChain and assign this marked end location to another int called chainEnd and the block this chainEnd points to as another int named nextBlock which will be assigned as the new vcb freeChainHead. Finally, it will assign

chainHead as the EOB, save the FAT with saveFAT, and return with returnLocation to provide the start location of the allocated blocks.

deallocateBlocks

The deallocateBlocks function accepts an int that is used as the location of the initial starting block to deallocate. Using this given starting location it will utilize a while loop to find the end of the chain and then link the deallocated blocks to the start of the freeChain with freeChainHead. After this linking the freeChainHead will then be set to the location of the initial starting block to deallocate and the location will be checked to ensure that it isn't part of another file. If it isn't then the block before the location will be marked as EOB. Finally, these changes will be saved to the fat with the saveFAT call and return with 0 to indicate success.

buildString

The buildString function accepts two char pointer strings with the first being the string of the new path and the second being the string of the current path. If the passed-in path is a relative path it'll token through the current path and the new path concatenating them together to form the actual path and if it's an absolute path then it'll token from the new path only to form the actual path. After the tokening, it'll strdup the actual path to the cwd path and return.

manageFileSize

The growSpace function is responsible for increasing the space allocated to a file if needed. The function begins by determining the current file position using the getFilePosition function. This position signifies the offset within the file where data is to be written. It then calculates the minimum number of blocks required to accommodate the additional data. This calculation considers the current file position and the number of bytes needed, converting them into blocks using the bytesToBlocks function. Then, the function compares the calculated minimum blocks with the extra blocks already allocated. If the minimum blocks exceed the extra blocks, it implies that additional blocks need to be allocated to fulfill the data growth requirement. If additional blocks are needed, the function increments the count to include the new blocks. It then calculates the total number of new blocks required by adding the minimum blocks to the extra blocks. The function locates the sentinel block at the start of the file using the findSentinel function. Next, it allocates the required number of blocks using the allocateBlocks function. This allocates contiguous blocks in the file system to accommodate the new data. The newly allocated blocks are linked to the existing file structure using the linkChain function. The file size is updated to reflect the additional space allocated for the file. The function saves the directory entry to reflect the changes in the file's size and updates the preAllocatedBlocks count. Finally, the function returns the total number of new blocks allocated.

The checkGrow function ensures that sufficient space is available to accommodate the incoming data. If the current file size is insufficient, it triggers the allocation of additional space through the growSpace function. The function starts by retrieving the current file position using the getFilePosition function. This position indicates the current offset within the file. It then checks whether writing count bytes at the current position would exceed the current size of the file. If writing count bytes surpasses the current file size, it indicates that the file needs to be grown to accommodate the additional data. In such cases, the function calls the growSpace function, passing the b_fcb pointer fcb and the required additional bytes count as arguments. The growSpace function is responsible for allocating additional space in the file system to accommodate the new data. After growing the file, checkGrow returns a value of 1 to indicate that file growth was necessary. If writing count bytes does not exceed the current file size, the function returns 0, indicating that file growth is not necessary.

The function shrinkSpace works by accepting a b_fcb and will use this fcb to determine the amount of actual blocks needed to hold the stored file removing the extra allocated blocks given to the file. The first thing shrinkSpace will do is change the fileSize of the current file in the parents information to the current files maxFilePos. After this, it will call bytesToBlocks on maxFilePos to get the actual used block amount assigning it to an int. This int will then be used in getEndBlock which will return with the location of the last used block and then call deallocateBlocks with the block after the last used block deallocating all of the unused blocks, call saveDir with the files parent to save the updated fileSize in the parent, and then return with 0 to indicate success.

Screenshots showing each of the commands listed

- cp - Copies a file

- mv - Moves a file

```
Prompt > mv hello foo
Prompt > ls -la

D         7680   .
D         7680   ..
-      3196730   works
D         2560   foo
Prompt > ls foo

hello
```

- md - Make a new directory

```
Prompt > md foo
Prompt > ls -la

D         7680   .
D         7680   ..
-      3196730   works
-      3196730   hello
D         2560   foo
Prompt >
```

- rm - Removes a file or directory

```
Prompt > cd foo
Current Path is: /foo/
Prompt > rm hello
Prompt > ls -la

D         2560   .
D         7680   ..
Prompt >
```

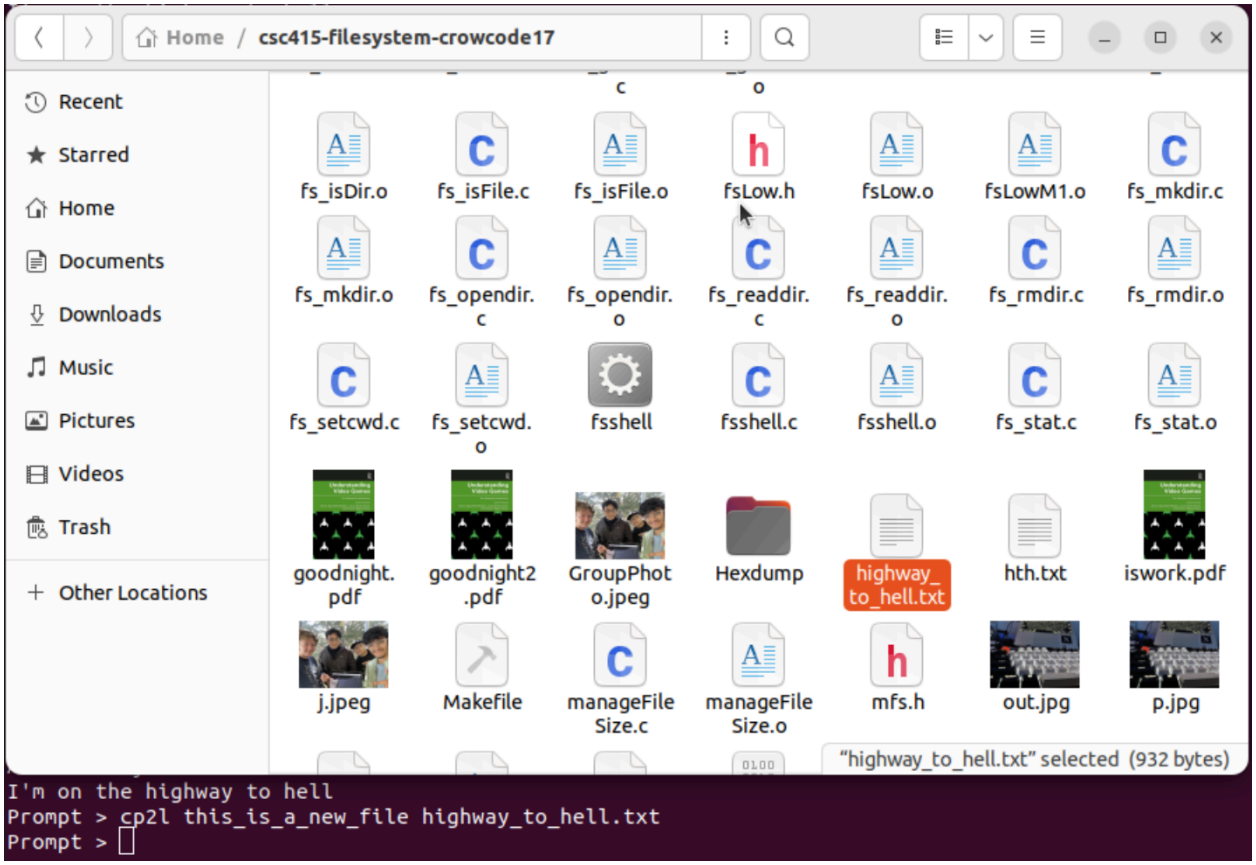
- touch - creates a file

```
Prompt > touch this_is_a_new_file
Prompt > ls -la

D         2560   .
D         7680   ..
-            0   this_is_a_new_file
Prompt >
```

- cat - (limited functionality) displays the contents of a file

```
Prompt > touch this_is_a_new_file
Prompt > cp2fs hth.txt this_is_a_new_file
Prompt > cat this_is_a_new_file
Livin' easy
Lovin' free
Season ticket on a one way ride
Askin' nothin'
Leave me be
Takin' everythin' in my stride
Don't need reason
Don't need rhyme
Ain't nothin' that I'd rather do
Goin' down
Party time
My friends are gonna be there too
I'm on the highway to hell
On the highway to hell
Highway to hell
I'm on the highway to hell
No stop signs
Speed limit
Nobody's gonna slow me down
Like a wheel
Gonna spin it
Nobody's gonna mess me around
Hey satan
Payin' my dues
Playin' in a rockin' band
Hey mumma
Look at me
I'm on the way to the promised land
I'm on the highway to hell
Highway to hell
I'm on the highway to hell
Highway to hell
Don't stop me
I'm on the highway to hell
On the highway to hell
Highway to hell
I'm on the highway to hell
(highway to hell) I'm on the highway to hell
(highway to hell) highway to hell
(highway to hell) highway to hell
(highway to hell)
And I'm goin' down
All the way
I'm on the highway to hell
Prompt >
```

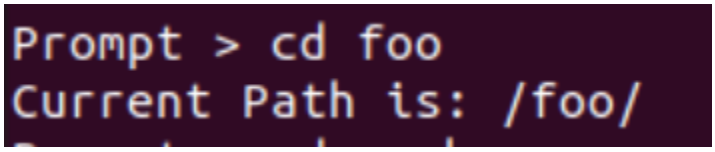
- cp2l - Copies a file from the test file system to the Linux file system



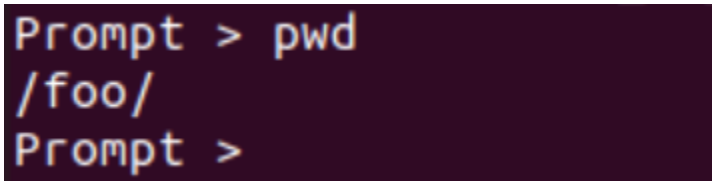
- cp2fs - Copies a file from the Linux file system to the test file system

```
student@student:~/csc415-filesystem-crowcode17$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does not exist, errno = 2
File SampleVolume not good to go, errno = 2
Block size is : 512
Created a volume with 9999872 bytes, broken into 19531 blocks of 512 bytes.
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Initializing File System with 19531 blocks with a block size of 512
Signature did not match, initializing VCB
(INIT) The freeChainHead is at: 154
|--------------------------------|
|------- 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 > cp2fs GroupPhoto.jpeg works
Prompt > ls -la

D         7680   .
D         7680   ..
-      3196730   works
Prompt >
```

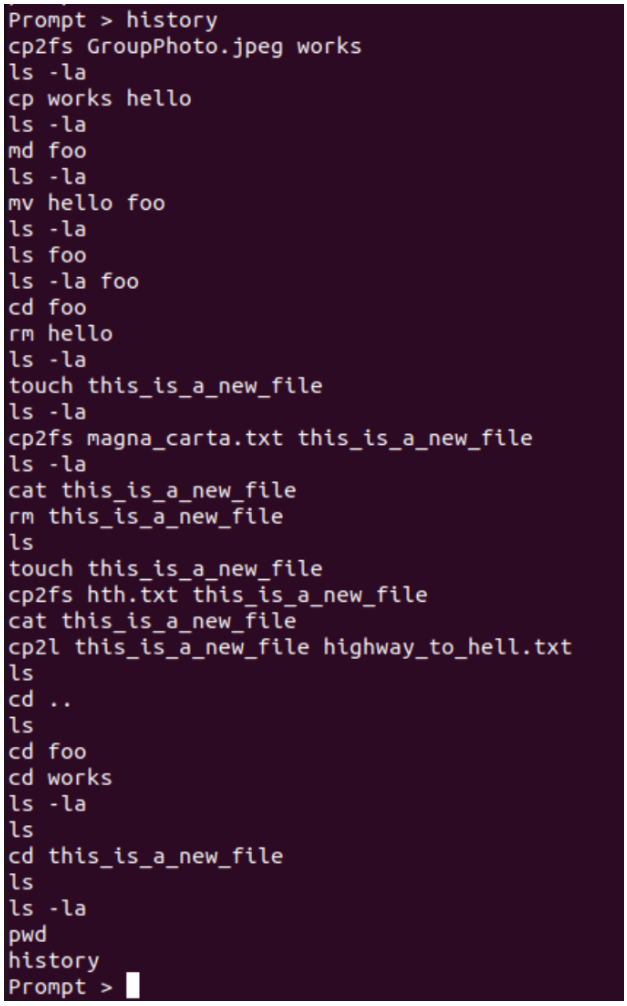
- cd - Changes directory



- pwd - Prints the working directory



- history - Prints out the history



- help - Prints out help

```
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
Prompt >
```

VCB

Yellow: Signature

Cyan: block count

Green: root location

Orange: root size

Red: free space location

Pink: free chain head

```
000200: 65 6C 70 6D 69 53 53 00  4B 4C 00 00 9A 00 00 00 | elpmiSS.KL..•...
000210: 0F 00 00 00 01 00 00 00  16 19 00 00 00 00 00 00 | ................
000220: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000230: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000240: 00 00 00 00 00 00 00 00  00 00 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  00 00 00 00 00 00 00 00 | ................
000270: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000280: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000290: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

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 | ................
```

FAT (File Allocation Table)

Purple: VCB

Yellow: Root directory

Cyan: foo directory

Green: out.txt

```
000400: FD FF FF FF 02 00 00 00  03 00 00 00 04 00 00 00 | ????............
000410: 05 00 00 00 06 00 00 00  07 00 00 00 08 00 00 00 | ................
000420: 09 00 00 00 0A 00 00 00  0B 00 00 00 0C 00 00 00 | ................
000430: 0D 00 00 00 0E 00 00 00  0F 00 00 00 10 00 00 00 | ................
000440: 11 00 00 00 12 00 00 00  13 00 00 00 14 00 00 00 | ................
000450: 15 00 00 00 16 00 00 00  17 00 00 00 18 00 00 00 | ................
000460: 19 00 00 00 1A 00 00 00  1B 00 00 00 1C 00 00 00 | ................
000470: 1D 00 00 00 1E 00 00 00  1F 00 00 00 20 00 00 00 | ............ ...
000480: 21 00 00 00 22 00 00 00  23 00 00 00 24 00 00 00 | !..."...#...$...
000490: 25 00 00 00 26 00 00 00  27 00 00 00 28 00 00 00 | %...&...'...(...
0004A0: 29 00 00 00 2A 00 00 00  2B 00 00 00 2C 00 00 00 | )...*...+...,...
0004B0: 2D 00 00 00 2E 00 00 00  2F 00 00 00 30 00 00 00 | -......./...0...
0004C0: 31 00 00 00 32 00 00 00  33 00 00 00 34 00 00 00 | 1...2...3...4...
0004D0: 35 00 00 00 36 00 00 00  37 00 00 00 38 00 00 00 | 5...6...7...8...
0004E0: 39 00 00 00 3A 00 00 00  3B 00 00 00 3C 00 00 00 | 9...:...;...<...
0004F0: 3D 00 00 00 3E 00 00 00  3F 00 00 00 40 00 00 00 | =...>...?...@...

000500: 41 00 00 00 42 00 00 00  43 00 00 00 44 00 00 00 | A...B...C...D...
```

```
000510: 45 00 00 00 46 00 00 00  47 00 00 00 48 00 00 00 | E...F...G...H...
000520: 49 00 00 00 4A 00 00 00  4B 00 00 00 4C 00 00 00 | I...J...K...L...
000530: 4D 00 00 00 4E 00 00 00  4F 00 00 00 50 00 00 00 | M...N...O...P...
000540: 51 00 00 00 52 00 00 00  53 00 00 00 54 00 00 00 | Q...R...S...T...
000550: 55 00 00 00 56 00 00 00  57 00 00 00 58 00 00 00 | U...V...W...X...
000560: 59 00 00 00 5A 00 00 00  5B 00 00 00 5C 00 00 00 | Y...Z...[...\...
000570: 5D 00 00 00 5E 00 00 00  5F 00 00 00 60 00 00 00 | ]...^..._...`...
000580: 61 00 00 00 62 00 00 00  63 00 00 00 64 00 00 00 | a...b...c...d...
000590: 65 00 00 00 66 00 00 00  67 00 00 00 68 00 00 00 | e...f...g...h...
0005A0: 69 00 00 00 6A 00 00 00  6B 00 00 00 6C 00 00 00 | i...j...k...l...
0005B0: 6D 00 00 00 6E 00 00 00  6F 00 00 00 70 00 00 00 | m...n...o...p...
0005C0: 71 00 00 00 72 00 00 00  73 00 00 00 74 00 00 00 | q...r...s...t...
0005D0: 75 00 00 00 76 00 00 00  77 00 00 00 78 00 00 00 | u...v...w...x...
0005E0: 79 00 00 00 7A 00 00 00  7B 00 00 00 7C 00 00 00 | y...z...{...|...
0005F0: 7D 00 00 00 7E 00 00 00  7F 00 00 00 80 00 00 00 | }...~.......€...

000600: 81 00 00 00 82 00 00 00  83 00 00 00 84 00 00 00 | ?...?...?...?...
000610: 85 00 00 00 86 00 00 00  87 00 00 00 88 00 00 00 | ?...?...?...?...
000620: 89 00 00 00 8A 00 00 00  8B 00 00 00 8C 00 00 00 | ?...?...?...?...
000630: 8D 00 00 00 8E 00 00 00  8F 00 00 00 90 00 00 00 | ?...?...?...?...
000640: 91 00 00 00 92 00 00 00  93 00 00 00 94 00 00 00 | ?...?...?...?...
000650: 95 00 00 00 96 00 00 00  97 00 00 00 98 00 00 00 | ?...?...?...?...
000660: FD FF FF FF 9A 00 00 00  9B 00 00 00 9C 00 00 00 | ?????...?...?...
000670: 9D 00 00 00 9E 00 00 00  9F 00 00 00 A0 00 00 00 | ?...?...?...?...
000680: A1 00 00 00 A2 00 00 00  A3 00 00 00 A4 00 00 00 | ?...?...?...?...
000690: A5 00 00 00 A6 00 00 00  A7 00 00 00 A8 00 00 00 | ?...?...?...?...
0006A0: FD FF FF FF AA 00 00 00  AB 00 00 00 AC 00 00 00 | ?????...?...?...
0006B0: AD 00 00 00 FD FF FF FF  AF 00 00 00 B0 00 00 00 | ?...?????...?...
0006C0: FD FF FF FF B2 00 00 00  B3 00 00 00 B4 00 00 00 | ?????...?...?...
0006D0: B5 00 00 00 B6 00 00 00  B7 00 00 00 B8 00 00 00 | ?...?...?...?...
0006E0: B9 00 00 00 BA 00 00 00  BB 00 00 00 BC 00 00 00 | ?...?...?...?...
0006F0: BD 00 00 00 BE 00 00 00  BF 00 00 00 C0 00 00 00 | ?...?...?...?…
```

foo (directory that we made)

Yellow: isDirectory

Cyan: name

Green: size

Orange: location

Pink: date created

Purple: date modified

```
015400: 01 2E 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015410: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015420: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015430: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015440: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015450: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015460: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015470: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015480: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015490: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0154A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0154B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0154C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0154D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0154E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0154F0: 00 0A 00 00 A9 00 00 00  63 C7 39 66 63 C7 39 66 | ....©...cÇ9fcÇ9f

015500: 01 2E 2E 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015510: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015520: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015530: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015540: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015550: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015560: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015570: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015580: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

```
015590: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0155A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0155B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0155C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0155D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0155E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0155F0: 00 1E 00 00 9A 00 00 00  57 C7 39 66 63 C7 39 66 | ....►...W►9fc►9f

015600: 00 6F 75 74 2E 74 78 74  00 00 00 00 00 00 00 00 | .out.txt........
015610: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015620: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015630: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015640: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015650: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015660: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015670: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015680: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
015690: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0156A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0156B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0156C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0156D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0156E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0156F0: A4 03 00 00 AE 00 00 00  77 C7 39 66 7C C7 39 66 | ►...►...w►9f|►9f
```

Out.txt

The file that we imported from the linux to the file system using cp2fs

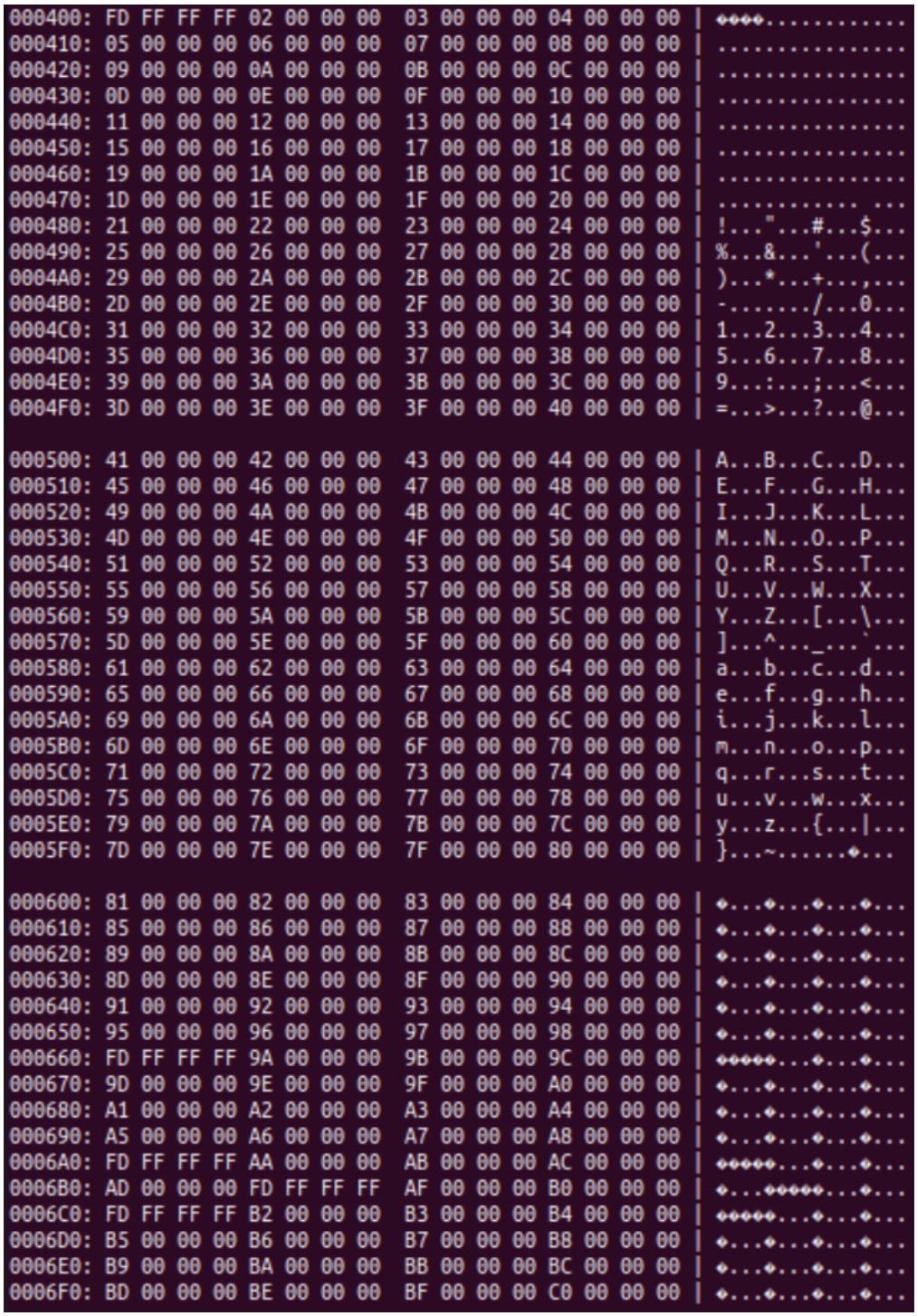
```
015E00: 4C 69 76 69 6E 27 20 65  61 73 79 0A 4C 6F 76 69 | Livin' easy.Lovi
015E10: 6E 27 20 66 72 65 65 0A  53 65 61 73 6F 6E 20 74 | n' free.Season t
015E20: 69 63 6B 65 74 20 6F 6E  20 61 20 6F 6E 65 20 77 | icket on a one w
015E30: 61 79 20 72 69 64 65 0A  41 73 6B 69 6E 27 20 6E | ay ride.Askin' n
```

```
015E40: 6F 74 68 69 6E 27 0A 4C  65 61 76 65 20 6D 65 20 | othin'.Leave me
015E50: 62 65 0A 54 61 6B 69 6E  27 20 65 76 65 72 79 74 | be.Takin' everyt
015E60: 68 69 6E 27 20 69 6E 20  6D 79 20 73 74 72 69 64 | hin' in my strid
015E70: 65 0A 44 6F 6E 27 74 20  6E 65 65 64 20 72 65 61 | e.Don't need rea
015E80: 73 6F 6E 0A 44 6F 6E 27  74 20 6E 65 65 64 20 72 | son.Don't need r
015E90: 68 79 6D 65 0A 41 69 6E  27 74 20 6E 6F 74 68 69 | hyme.Ain't nothi
015EA0: 6E 27 20 74 68 61 74 20  49 27 64 20 72 61 74 68 | n' that I'd rath
015EB0: 65 72 20 64 6F 0A 47 6F  69 6E 27 20 64 6F 77 6E | er do.Goin' down
015EC0: 0A 50 61 72 74 79 20 74  69 6D 65 0A 4D 79 20 66 | .Party time.My f
015ED0: 72 69 65 6E 64 73 20 61  72 65 20 67 6F 6E 6E 61 | riends are gonna
015EE0: 20 62 65 20 74 68 65 72  65 20 74 6F 6F 0A 49 27 |  be there too.I'
015EF0: 6D 20 6F 6E 20 74 68 65  20 68 69 67 68 77 61 79 | m on the highway

015F00: 20 74 6F 20 68 65 6C 6C  0A 4F 6E 20 74 68 65 20 |  to hell.On the
015F10: 68 69 67 68 77 61 79 20  74 6F 20 68 65 6C 6C 0A | highway to hell.
015F20: 48 69 67 68 77 61 79 20  74 6F 20 68 65 6C 6C 0A | Highway to hell.
015F30: 49 27 6D 20 6F 6E 20 74  68 65 20 68 69 67 68 77 | I'm on the highw
015F40: 61 79 20 74 6F 20 68 65  6C 6C 0A 4E 6F 20 73 74 | ay to hell.No st
015F50: 6F 70 20 73 69 67 6E 73  0A 53 70 65 65 64 20 6C | op signs.Speed l
015F60: 69 6D 69 74 0A 4E 6F 62  6F 64 79 27 73 20 67 6F | imit.Nobody's go
015F70: 6E 6E 61 20 73 6C 6F 77  20 6D 65 20 64 6F 77 6E | nna slow me down
015F80: 0A 4C 69 6B 65 20 61 20  77 68 65 65 6C 0A 47 6F | .Like a wheel.Go
015F90: 6E 6E 61 20 73 70 69 6E  20 69 74 0A 4E 6F 62 6F | nna spin it.Nobo
015FA0: 64 79 27 73 20 67 6F 6E  6E 61 20 6D 65 73 73 20 | dy's gonna mess
015FB0: 6D 65 20 61 72 6F 75 6E  64 0A 48 65 79 20 73 61 | me around.Hey sa
015FC0: 74 61 6E 0A 50 61 79 69  6E 27 20 6D 79 20 64 75 | tan.Payin' my du
015FD0: 65 73 0A 50 6C 61 79 69  6E 27 20 69 6E 20 61 20 | es.Playin' in a
015FE0: 72 6F 63 6B 69 6E 27 20  62 61 6E 64 0A 48 65 79 | rockin' band.Hey
015FF0: 20 6D 75 6D 6D 61 0A 4C  6F 6F 6B 20 61 74 20 6D |  mumma.Look at m

016000: 65 0A 49 27 6D 20 6F 6E  20 74 68 65 20 77 61 79 | e.I'm on the way
016010: 20 74 6F 20 74 68 65 20  70 72 6F 6D 69 73 65 64 |  to the promised
016020: 20 6C 61 6E 64 0A 49 27  6D 20 6F 6E 20 74 68 65 |  land.I'm on the
016030: 20 68 69 67 68 77 61 79  20 74 6F 20 68 65 6C 6C |  highway to hell
016040: 0A 48 69 67 68 77 61 79  20 74 6F 20 68 65 6C 6C | .Highway to hell
```

```
016050: 0A 49 27 6D 20 6F 6E 20  74 68 65 20 68 69 67 68 | .I'm on the high
016060: 77 61 79 20 74 6F 20 68  65 6C 6C 0A 48 69 67 68 | way to hell.High
016070: 77 61 79 20 74 6F 20 68  65 6C 6C 0A 44 6F 6E 27 | way to hell.Don'
016080: 74 20 73 74 6F 70 20 6D  65 0A 49 27 6D 20 6F 6E | t stop me.I'm on
016090: 20 74 68 65 20 68 69 67  68 77 61 79 20 74 6F 20 |  the highway to
0160A0: 68 65 6C 6C 0A 4F 6E 20  74 68 65 20 68 69 67 68 | hell.On the high
0160B0: 77 61 79 20 74 6F 20 68  65 6C 6C 0A 48 69 67 68 | way to hell.High
0160C0: 77 61 79 20 74 6F 20 68  65 6C 6C 0A 49 27 6D 20 | way to hell.I'm
0160D0: 6F 6E 20 74 68 65 20 68  69 67 68 77 61 79 20 74 | on the highway t
0160E0: 6F 20 68 65 6C 6C 0A 28  68 69 67 68 77 61 79 20 | o hell.(highway
0160F0: 74 6F 20 68 65 6C 6C 29  20 49 27 6D 20 6F 6E 20 | to hell) I'm on

016100: 74 68 65 20 68 69 67 68  77 61 79 20 74 6F 20 68 | the highway to h
016110: 65 6C 6C 0A 28 68 69 67  68 77 61 79 20 74 6F 20 | ell.(highway to
016120: 68 65 6C 6C 29 20 68 69  67 68 77 61 79 20 74 6F | hell) highway to
016130: 20 68 65 6C 6C 0A 28 68  69 67 68 77 61 79 20 74 |  hell.(highway t
016140: 6F 20 68 65 6C 6C 29 20  68 69 67 68 77 61 79 20 | o hell) highway
016150: 74 6F 20 68 65 6C 6C 0A  28 68 69 67 68 77 61 79 | to hell.(highway
016160: 20 74 6F 20 68 65 6C 6C  29 0A 41 6E 64 20 49 27 |  to hell).And I'
016170: 6D 20 67 6F 69 6E 27 20  64 6F 77 6E 0A 41 6C 6C | m goin' down.All
016180: 20 74 68 65 20 77 61 79  0A 49 27 6D 20 6F 6E 20 |  the way.I'm on
016190: 74 68 65 20 68 69 67 68  77 61 79 20 74 6F 20 68 | the highway to h
0161A0: 65 6C 6C 0A 20 67 6F 6E  6E 61 20 6D 65 73 73 20 | ell. gonna mess
0161B0: 6D 65 20 61 72 6F 75 6E  64 0A 48 65 79 20 73 61 | me around.Hey sa
0161C0: 74 61 6E 0A 50 61 79 69  6E 27 20 6D 79 20 64 75 | tan.Payin' my du
0161D0: 65 73 0A 50 6C 61 79 69  6E 27 20 69 6E 20 61 20 | es.Playin' in a
0161E0: 72 6F 63 6B 69 6E 27 20  62 61 6E 64 0A 48 65 79 | rockin' band.Hey
0161F0: 20 6D 75 6D 6D 61 0A 4C  6F 6F 6B 20 61 74 20 6D |  mumma.Look at m
```

Screenshots from the terminal:

The FAT:


000400: FD FF FF FF 02 00 00 00 03 00 00 00 04 00 00 00
000410: 05 00 00 00 06 00 00 00 07 00 00 00 08 00 00 00
000420: 09 00 00 00 0A 00 00 00 0B 00 00 00 0C 00 00 00
000430: 0D 00 00 00 0E 00 00 00 0F 00 00 00 10 00 00 00
000440: 11 00 00 00 12 00 00 00 13 00 00 00 14 00 00 00
000450: 15 00 00 00 16 00 00 00 17 00 00 00 18 00 00 00
000460: 19 00 00 00 1A 00 00 00 1B 00 00 00 1C 00 00 00
000470: 1D 00 00 00 1E 00 00 00 1F 00 00 00 20 00 00 00
000480: 21 00 00 00 22 00 00 00 23 00 00 00 24 00 00 00
000490: 25 00 00 00 26 00 00 00 27 00 00 00 28 00 00 00
0004A0: 29 00 00 00 2A 00 00 00 2B 00 00 00 2C 00 00 00
0004B0: 2D 00 00 00 2E 00 00 00 2F 00 00 00 30 00 00 00
0004C0: 31 00 00 00 32 00 00 00 33 00 00 00 34 00 00 00
0004D0: 35 00 00 00 36 00 00 00 37 00 00 00 38 00 00 00
0004E0: 39 00 00 00 3A 00 00 00 3B 00 00 00 3C 00 00 00
0004F0: 3D 00 00 00 3E 00 00 00 3F 00 00 00 40 00 00 00
000500: 41 00 00 00 42 00 00 00 43 00 00 00 44 00 00 00
000510: 45 00 00 00 46 00 00 00 47 00 00 00 48 00 00 00
000520: 49 00 00 00 4A 00 00 00 4B 00 00 00 4C 00 00 00
000530: 4D 00 00 00 4E 00 00 00 4F 00 00 00 50 00 00 00
000540: 51 00 00 00 52 00 00 00 53 00 00 00 54 00 00 00
000550: 55 00 00 00 56 00 00 00 57 00 00 00 58 00 00 00
000560: 59 00 00 00 5A 00 00 00 5B 00 00 00 5C 00 00 00
000570: 5D 00 00 00 5E 00 00 00 5F 00 00 00 60 00 00 00
000580: 61 00 00 00 62 00 00 00 63 00 00 00 64 00 00 00
000590: 65 00 00 00 66 00 00 00 67 00 00 00 68 00 00 00
0005A0: 69 00 00 00 6A 00 00 00 6B 00 00 00 6C 00 00 00
0005B0: 6D 00 00 00 6E 00 00 00 6F 00 00 00 70 00 00 00
0005C0: 71 00 00 00 72 00 00 00 73 00 00 00 74 00 00 00
0005D0: 75 00 00 00 76 00 00 00 77 00 00 00 78 00 00 00
0005E0: 79 00 00 00 7A 00 00 00 7B 00 00 00 7C 00 00 00
0005F0: 7D 00 00 00 7E 00 00 00 7F 00 00 00 80 00 00 00
000600: 81 00 00 00 82 00 00 00 83 00 00 00 84 00 00 00
000610: 85 00 00 00 86 00 00 00 87 00 00 00 88 00 00 00
000620: 89 00 00 00 8A 00 00 00 8B 00 00 00 8C 00 00 00
000630: 8D 00 00 00 8E 00 00 00 8F 00 00 00 90 00 00 00
000640: 91 00 00 00 92 00 00 00 93 00 00 00 94 00 00 00
000650: 95 00 00 00 96 00 00 00 97 00 00 00 98 00 00 00
000660: FD FF FF FF 9A 00 00 00 9B 00 00 00 9C 00 00 00
000670: 9D 00 00 00 9E 00 00 00 9F 00 00 00 A0 00 00 00
000680: A1 00 00 00 A2 00 00 00 A3 00 00 00 A4 00 00 00
000690: A5 00 00 00 A6 00 00 00 A7 00 00 00 A8 00 00 00
0006A0: FD FF FF FF AA 00 00 00 AB 00 00 00 AC 00 00 00
0006B0: AD 00 00 00 FD FF FF FF AF 00 00 00 B0 00 00 00
0006C0: FD FF FF FF B2 00 00 00 B3 00 00 00 B4 00 00 00
0006D0: B5 00 00 00 B6 00 00 00 B7 00 00 00 B8 00 00 00
0006E0: B9 00 00 00 BA 00 00 00 BB 00 00 00 BC 00 00 00
0006F0: BD 00 00 00 BE 00 00 00 BF 00 00 00 C0 00 00 00

The VCB:

```
000200: 65 6C 70 6D 69 53 53 00  4B 4C 00 00 9A 00 00 00 | elpmiSS.KL..◆...
000210: 0F 00 00 00 01 00 00 00  16 19 00 00 00 00 00 00 | ................
000220: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000230: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000240: 00 00 00 00 00 00 00 00  00 00 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  00 00 00 00 00 00 00 00 | ................
000270: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000280: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000290: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

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 | ................
```

The file copied using cp2fs:

```
015E00: 4C 69 76 69 6E 27 20 65  61 73 79 0A 4C 6F 76 69 | Livin' easy.Lovi
015E10: 6E 27 20 66 72 65 65 0A  53 65 61 73 6F 6E 20 74 | n' free.Season t
015E20: 69 63 6B 65 74 20 6F 6E  20 61 20 6F 6E 65 20 77 | icket on a one w
015E30: 61 79 20 72 69 64 65 0A  41 73 6B 69 6E 27 20 6E | ay ride.Askin' n
015E40: 6F 74 68 69 6E 27 0A 4C  65 61 76 65 20 6D 65 20 | othin'.Leave me
015E50: 62 65 0A 54 61 6B 69 6E  27 20 65 76 65 72 79 74 | be.Takin' everyt
015E60: 68 69 6E 27 20 69 6E 20  6D 79 20 73 74 72 69 64 | hin' in my strid
015E70: 65 0A 44 6F 6E 27 74 20  6E 65 65 64 20 72 65 61 | e.Don't need rea
015E80: 73 6F 6E 0A 44 6F 6E 27  74 20 6E 65 65 64 20 72 | son.Don't need r
015E90: 68 79 6D 65 0A 41 69 6E  27 74 20 6E 6F 74 68 69 | hyme.Ain't nothi
015EA0: 6E 27 20 74 68 61 74 20  49 27 64 20 72 61 74 68 | n' that I'd rath
015EB0: 65 72 20 64 6F 0A 47 6F  69 6E 27 20 64 6F 77 6E | er do.Goin' down
015EC0: 0A 50 61 72 74 79 20 74  69 6D 65 0A 4D 79 20 66 | .Party time.My f
015ED0: 72 69 65 6E 64 73 20 61  72 65 20 67 6F 6E 6E 61 | riends are gonna
015EE0: 20 62 65 20 74 68 65 72  65 20 74 6F 6F 0A 49 27 |  be there too.I'
015EF0: 6D 20 6F 6E 20 74 68 65  20 68 69 67 68 77 61 79 | m on the highway

015F00: 20 74 6F 20 68 65 6C 6C  0A 4F 6E 20 74 68 65 20 |  to hell.On the
015F10: 68 69 67 68 77 61 79 20  74 6F 20 68 65 6C 6C 0A | highway to hell.
015F20: 48 69 67 68 77 61 79 20  74 6F 20 68 65 6C 6C 0A | Highway to hell.
015F30: 49 27 6D 20 6F 6E 20 74  68 65 20 68 69 67 68 77 | I'm on the highw
015F40: 61 79 20 74 6F 20 68 65  6C 6C 0A 4E 6F 20 73 74 | ay to hell.No st
015F50: 6F 70 20 73 69 67 6E 73  0A 53 70 65 65 64 20 6C | op signs.Speed l
015F60: 69 6D 69 74 0A 4E 6F 62  6F 64 79 27 73 20 67 6F | imit.Nobody's go
015F70: 6E 6E 61 20 73 6C 6F 77  20 6D 65 20 64 6F 77 6E | nna slow me down
015F80: 0A 4C 69 6B 65 20 61 20  77 68 65 65 6C 0A 47 6F | .Like a wheel.Go
015F90: 6E 6E 61 20 73 70 69 6E  20 69 74 0A 4E 6F 62 6F | nna spin it.Nobo
015FA0: 64 79 27 73 20 67 6F 6E  6E 61 20 6D 65 73 73 20 | dy's gonna mess
015FB0: 6D 65 20 61 72 6F 75 6E  64 0A 48 65 79 20 73 61 | me around.Hey sa
015FC0: 74 61 6E 0A 50 61 79 69  6E 27 20 6D 79 20 64 75 | tan.Payin' my du
015FD0: 65 73 0A 50 6C 61 79 69  6E 27 20 69 6E 20 61 20 | es.Playin' in a
015FE0: 72 6F 63 6B 69 6E 27 20  62 61 6E 64 0A 48 65 79 | rockin' band.Hey
015FF0: 20 6D 75 6D 6D 61 0A 4C  6F 6F 6B 20 61 74 20 6D |  mumma.Look at m

016000: 65 0A 49 27 6D 20 6F 6E  20 74 68 65 20 77 61 79 | e.I'm on the way
016010: 20 74 6F 20 74 68 65 20  70 72 6F 6D 69 73 65 64 |  to the promised
016020: 20 6C 61 6E 64 0A 49 27  6D 20 6F 6E 20 74 68 65 |  land.I'm on the
016030: 20 68 69 67 68 77 61 79  20 74 6F 20 68 65 6C 6C |  highway to hell
016040: 0A 48 69 67 68 77 61 79  20 74 6F 20 68 65 6C 6C | .Highway to hell
016050: 0A 49 27 6D 20 6F 6E 20  74 68 65 20 68 69 67 68 | .I'm on the high
016060: 77 61 79 20 74 6F 20 68  65 6C 6C 0A 48 69 67 68 | way to hell.High
016070: 77 61 79 20 74 6F 20 68  65 6C 6C 0A 44 6F 6E 27 | way to hell.Don'
016080: 74 20 73 74 6F 70 20 6D  65 0A 49 27 6D 20 6F 6E | t stop me.I'm on
016090: 20 74 68 65 20 68 69 67  68 77 61 79 20 74 6F 20 |  the highway to
0160A0: 68 65 6C 6C 0A 4F 6E 20  74 68 65 20 68 69 67 68 | hell.On the high
0160B0: 77 61 79 20 74 6F 20 68  65 6C 6C 0A 48 69 67 68 | way to hell.High
0160C0: 77 61 79 20 74 6F 20 68  65 6C 6C 0A 49 27 6D 20 | way to hell.I'm
0160D0: 6F 6E 20 74 68 65 20 68  69 67 68 77 61 79 20 74 | on the highway t
0160E0: 6F 20 68 65 6C 6C 0A 28  68 69 67 68 77 61 79 20 | o hell.(highway
0160F0: 74 6F 20 68 65 6C 6C 29  20 49 27 6D 20 6F 6E 20 | to hell) I'm on

016100: 74 68 65 20 68 69 67 68  77 61 79 20 74 6F 20 68 | the highway to h
016110: 65 6C 6C 0A 28 68 69 67  68 77 61 79 20 74 6F 20 | ell.(highway to
016120: 68 65 6C 6C 29 20 68 69  67 68 77 61 79 20 74 6F | hell) highway to
016130: 20 68 65 6C 6C 0A 28 68  69 67 68 77 61 79 20 74 |  hell.(highway t
016140: 6F 20 68 65 6C 6C 29 20  68 69 67 68 77 61 79 20 | o hell) highway
016150: 74 6F 20 68 65 6C 6C 0A  28 68 69 67 68 77 61 79 | to hell.(highway
016160: 20 74 6F 20 68 65 6C 6C  29 0A 41 6E 64 20 49 27 |  to hell).And I'
016170: 6D 20 67 6F 69 6E 27 20  64 6F 77 6E 0A 41 6C 6C | m goin' down.All
016180: 20 74 68 65 20 77 61 79  0A 49 27 6D 20 6F 6E 20 |  the way.I'm on
016190: 74 68 65 20 68 69 67 68  77 61 79 20 74 6F 20 68 | the highway to h
0161A0: 65 6C 6C 0A 20 67 6F 6E  6E 61 20 6D 65 73 73 20 | ell. gonna mess
0161B0: 6D 65 20 61 72 6F 75 6E  64 0A 48 65 79 20 73 61 | me around.Hey sa
0161C0: 74 61 6E 0A 50 61 79 69  6E 27 20 6D 79 20 64 75 | tan.Payin' my du
0161D0: 65 73 0A 50 6C 61 79 69  6E 27 20 69 6E 20 61 20 | es.Playin' in a
0161E0: 72 6F 63 6B 69 6E 27 20  62 61 6E 64 0A 48 65 79 | rockin' band.Hey
0161F0: 20 6D 75 6D 6D 61 0A 4C  6F 6F 6B 20 61 74 20 6D |  mumma.Look at m
```

The foo directory that we made using md: