Natalie Christie, 918376646 Github: nataliekchristie
Joseph Kois, 920921089, Github: josephkois
Jade Simien, 920258687, Github: JadeS01
Eugene San Juan, 918354065, Github: esanjuan915

File System Project
Team Arrays of Dread

Github: https://github.com/CSC415-Fall2021/csc415-filesystem-esanjuan915

1. Description of File System:

fsInit

Contains everything required to initialize our file system.

Functions:

initFilesystem:
Takes the parameters of numberOfBlocks and blockSize. This will designate the amount of memory we are using. This function initializes our volume control block which is a global variable inside our file system. It assigns all necessary values including: blockCount, blockSize, freeSpaceMap (location of our free space map), rootDir (location of root directory), rootDirBlocks (amount of blocks our root directory uses), and the signature used to match our VCB. It then writes the vcb and freemap to memory.

fsRoot

Contains a function that initializes the root directory. This is utilized in our fsInit function.

Functions:

initRootDir:
Takes a parameter blockSize from the fsInit file. It uses this to calculate the root directory size using our predefined number of entries we have for our directories. It finds a free space in memory using the free space map for where it will be written into memory. It then assignes values to the first few entries of the rootdir entry struct array. The first entry has the name “.” and is a pointer to itself. The second is “..” which is typically a pointer to the parent but being root it is the same as the first. It also assigns blockLocation (where it is in memory), blockCount (amount of logical blocks it uses), size of the entry, the type (directory/entry/undefined), and amount of children. It assignes undefined value to the rest of the values, writes it to memory, and sets the cwd using a function from mfs.

fsFree

Natalie Christie, 918376646 Github: nataliekchristie
Joseph Kois, 920921089, Github: josephkois
Jade Simien, 920258687, Github: JadeS01
Eugene San Juan, 918354065, Github: esanjuan915

Contains everything needed to initialize our freespacemap. The free space map is a global variable inside fsInit.h that is then written to memory. These functions are utilized in fsInit.

Functions:

findFreeBlocks:
Takes argument blocksNeeded which is as it sounds, the amount of logical blocks we need within our memory for whatever we are writing. What this function does is loop through the freemap in order to find the first free space that has the amount of blocks we need. It keeps track of how many blocks we have in a row and the location that starts at, and if there arent enough, starts over. This function returns the starting block location of the memory that is available.

initFreeSpace:
Takes a parameter of numberOfBlocks and blockSize. Uses this to malloc the appropriate space. It also marks down the blocks used by the vcb and freemap in memory and writes them. It returns where the free space map starts in memory so it can be saved to the volume control block.

B_io

Includes all the functions needed to read and write files.

Functions:

B_open:
This function will open up a buffered file. We first initialize our system, and get our own file descriptor. Then wrote an error check for the return variable we made, to make sure we actually have the fcb. Then, malloc the buffer to be the size of our vcb. Then, error check again to make sure there is enough space. Finally we return the file descriptor on success.

B_read:
This function reads through a file given a descriptor, buffer, and number of bytes the user wants read. The buffer reads through a portion of the file one block at a time and returns what is read to the user. If an amount in the buffer is enough to satisfy the count the user needs, it simply returns that amount and saves the impartial amount. It also must account for sizes larger than 512 bytes, so there are several if/else statements to compensate for that. It will return the amount read to the user when done and will quit at EOF.

B_write:
The purpose of this function is to write data to the free space of our vcb. It takes in 3 arguments, a file descriptor, buffer, and count. We created some variables for the bytes that are being

Natalie Christie, 918376646 Github: nataliekchristie
Joseph Kois, 920921089, Github: josephkois
Jade Simien, 920258687, Github: JadeS01
Eugene San Juan, 918354065, Github: esanjuan915

copied, we created two in case of overflow on the buffer. The return of this function will be the total bytes that are being written, which would be those two variables added together which we created to copy the bytes. The function will need to calculate the freespace first and foremost, because we need to know how many bytes can fit on the buffer. And we used an if else statement to determine if there is room on the block to write.

B_seek:
The seek function has the purpose of repositioning the file offset of the open file description. It will use if else if statements for the cases whence = SEEK_SET,CUR,END. Each case will move and reposition the file pointer according to what SET, CUR, and END actually do. So for the set we set that at the location + offset, then set the CUR to be pointer += offset, and finally set the END to be fileSize + offset to get the end of the file. Finally we have an error printout at the end in the case that whence is not valid.

B_close:
incomplete

Mfs

Contains the majority of functions and variables that are used in order to navigate our filesystem. Global variables include parsedPath (used to save a parsed path that a function is accessing), parentPath (used in functions that need to access the parent to an entry), currentDirectoryPath (used to store the cwd) , readDirIndex (keeps track of the index of the entry we are at), readDirRead (keeps track of how many children entries we have read of a directory).

Functions

parsePath:
This function takes a parameter of type char *. This would be the path entered by the user. When called it will start off by nullifying the path. It will then use strtok_r in order to parse the path by “/” and store it within a char array. It saves it in the parsedPath variable.

getArrLength:
Takes a parameter of type char * []. This will keep track of the size of an array and return the int size. This function is used in conjunction with parsePath for the getEntryFromPath function. It essentially is used to keep track of how many entries are within a path.

getParentPath:
This function takes no parameter and instead accesses the parsedPath global variable. It essentially loops through the path and saves everything but the last c string (which would be the name of the child path). It saves this within the parentPath global variable for further use.

Natalie Christie, 918376646 Github: nataliekchristie
Joseph Kois, 920921089, Github: josephkois
Jade Simien, 920258687, Github: JadeS01
Eugene San Juan, 918354065, Github: esanjuan915

getEntryFromPath:
Takes a parameter of char *arr[] and int for the length. Uses the information regarding the root directory saved into our volume control block to load the root directory into memory in the currentEntry global variable. It then checks if the arr has a string length of 0 or is blank and if so it simply returns the root directory (empty path). It uses the arr length in order to loop through and find each entry. It checks each child entry name of the current entry and returns an error if not found or reads the next one into memory if found. If not found it returns currentEntry as null.

Fs_opendir:
Opens a directory. Uses the char parameter as the name and creates a filename variable that is fed into b_open in order to get a file descriptor. If the file descriptor is less than 0 it returns an error. If not, an fdDir variable is made in order to save all the proper information gathered from the directory entry. To get the directory entry getEntryFromPath is called and all the information is saved into the fdDir struct that was declared. The entry is freed and fdDir is returned.

fs_readdir:
Returns a struct diriteminfo with all the directory entry information of an entry. Creates and mallocs a diriteminfo object, returns if the amount of entries read from the directory is equal to the amount of children. This function finds the specified parent entry and moves through each child entry of the parent entry and reads the type and name into the diriteminfo object which can then be accessed to do things such as print the child entries. Each time at the end it increments the index (by whatever number is the next valid entry, in case one is deleted) and amount of entries read (which is used to track when we finish reading). The diriteminfo struct is returned.

Fs_closedir:
Closes a directory. Essentially checks if a directory is open, and if it isn't, returns an error, and if it is, it frees the object.

fs_isFile:
Checks if the specified entry given by a path is a file. This function uses parsePath to get a parsed path and the getarrlength function to be fed into getEntryFromPath. This gives us the currentEntry which contains the entry we are looking for. We check the first entry in the arr for its type, indicated by integer values. 0 = defined, 1 = directory, 2 = file. We check the first entry because it is a pointer to itself and contains its own information. If the type matches 2, it returns 1 (for true).

fs_isDir:

Natalie Christie, 918376646 Github: nataliekchristie
Joseph Kois, 920921089, Github: josephkois
Jade Simien, 920258687, Github: JadeS01
Eugene San Juan, 918354065, Github: esanjuan915

Checks if the specified entry given by a path is a directory. Has the same logic as isFile. GetEntryFromPath is called and given to use in the currentEntry variable. We check the first entry in the entry struct array because it is the pointer to itself that contains all necessary information. If the type matches 1 it returns true.

Fs_stat:
Returns the information from a directory when given the path. Checks if the path is a file or directory and returns error if neither. Entry is found and then the information is printed.

Fs_delete:
Removes a file. Checks for the current working directory because we are only given the file name. Attaches the current working directory with the file name at the end to get the full path.The full path is then parsed and used to find the entry. An error returned if it isn't found. We also use a parentEntry variable to find the parent entry to the path (because its information must be updated as well) and read it into memory using the blockLocation and blockCount saved in the second entry in our entry we are looking to delete (the parent). The parent entry is then searched for the file name and if it matches, all values are put to zero and the value is changed to undefined so it can be used in the future. The free map is updated as well.

Fs_mkdir:
Make a directory. The specified pathname is parsed and an entry struct is declared. Using the size we call the findFreeBlocks method described before which will give us a free area to put our directory in memory. The parent path is retrieved and so is the parent entry. The parent entry's children amount is updated for other functions. All the appropriate info (type, name, etc) is given to the first and second entry (parent) in the new directory structure. The parent entry finds a free spot to store the information for the new entry and assigns all the same values that are assigned to the first entry of the new struct (all of its location, type, etc). This is written to memory and the parent entry is written to memory as well. The free map is updated and all malloc'd items are free'd.

Fs_rmdir:
Removes a directory. The path is parsed and the directory is found (and if not found, an error is returned). It checks if the directory has children still and returns an error if it does. The parent is loaded into memory using the information saved by the entry we are trying to delete. A loop cycles through each entry of the parent entry in order to nullify all the child entry information and declare it as undefined so that space can be used. This is written to memory and the freemap is updated.

Fs_getcwd:
Gets the current working directory. String copies the global variable currentDirectoryPath that holds the current working directory into a variable and returns it.

Natalie Christie, 918376646 Github: nataliekchristie
Joseph Kois, 920921089, Github: josephkois
Jade Simien, 920258687, Github: JadeS01
Eugene San Juan, 918354065, Github: esanjuan915

fs_setcwd:
Sets current working directory. Checks if the entry is a directory using isDir. If the entry exists and is a directory, the path is copied into our currentDirectoryPath variable.

2. Issues we had:

General issues with meeting and working: Because of the business of all of our teammates we found it fairly difficult to find times to meet where all of us (or a majority of us) could attend. Combined with this, we unfortunately had several teammates that had some personal issues that caused a lot of conflicts.

General issues with understanding overall concepts: We realized after our submission for milestone 1 and working on milestone 2 that we had some misconceptions about going about keeping track of our directories and reading and writing into memory. For example, when initiating our root directory we were assigning values improperly and weren't really understanding how to pull the directories from memory and write them. After a lot of constructive thinking and multiple attempts we came into the understanding that we would need some sort of function that used the vcb locations for root, used LBAread to read into memory using an entry struct with allocated memory, make the edits to the entry (or declare a new one and used the found entry parent to save the information) and then finally re-write the entry back into memory. Our solution essentially parsed the path into pieces and read root into memory, then looped through root to try to find the name of the entry and then read that into memory, and etc until the entire path pieces were processed.

Issues with including our header files: We were originally not including header files properly and ended up with compilation errors that basically saw that we had "duplicates" of certain structs and functions. This was fixed after some debugging and reworking.

Not allocating free space correctly

Unable to access some global variables: We had trouble accessing our vcb structure which we had initialized in fsinit.c. there were errors being thrown during compilation. We believe this is because of the scope of our files, we have tons of different files and headers that need to be included, so what we did as a temporary fix for now is move some of the other initializations like root and free space to the fsinit.c file to temporarily bypass these errors.

Natalie Christie, 918376646 Github: nataliekchristie
Joseph Kois, 920921089, Github: josephkois
Jade Simien, 920258687, Github: JadeS01
Eugene San Juan, 918354065, Github: esanjuan915

isFIle and isDir functions: at first, our idea was to create a variable called entries length which is the size of the list of entries / size of entry struct and then we would look for the path in the entries with a simple for loop and compare the path, to the path in the entry. And then after that we would do a string comparison of the type, either “F” or “D” for file or directory and if it is a good comparison it would return 1, if not 0. This did not work because we needed to parse the path first, and get the length of the parsed path. Followed by calling a function getEntryFromPath.

Difficulty allocating for pointers when finding parent entries: fixed by creating a global variable we are accessing which allows the malloc space to be freed properly. Had issues when trying to return functions as structures, not entirely sure why.

Issues with memory corruption: While working on getting entries from memory I noticed that our type system wasn’t working properly. We tried saving a char that had “d” for directory, “e” for entry and “u” for undefined in order to track each entry. Well, when entries were loaded into memory, much would be accurate (for example it would load the memory address and name properly) but the chars for type were getting corrupted. As a quick way to work around the issue I decided to change the variable to an int that has 0 = undefined, 1 = directory and 2 = file, which seemed to work properly.

B_io.c: Unfortunately b_io.c gave us a lot of trouble and we simply struggled with the overall concepts and implementing them correctly.

Double free/other malloc problems: Due to the sheer size of the program and the amount of variables we are working with we had some issues with getting errors about double freeing, corruption, etc. We tried to compensate for this by NULLifying structs when they are freed so we can check them in the future, which fixed some problems, but we still ended up with issues.

General overall issues: Unfortunately we had a lot of issues with implementing b_io and with malloc issues which caused our program to break randomly. Sometimes there are free/malloc errors that will cause the program to crash after a couple of commands. This likely has to do with poor malloc/free practices but we were struggling with finding all the errors and keeping everything together.

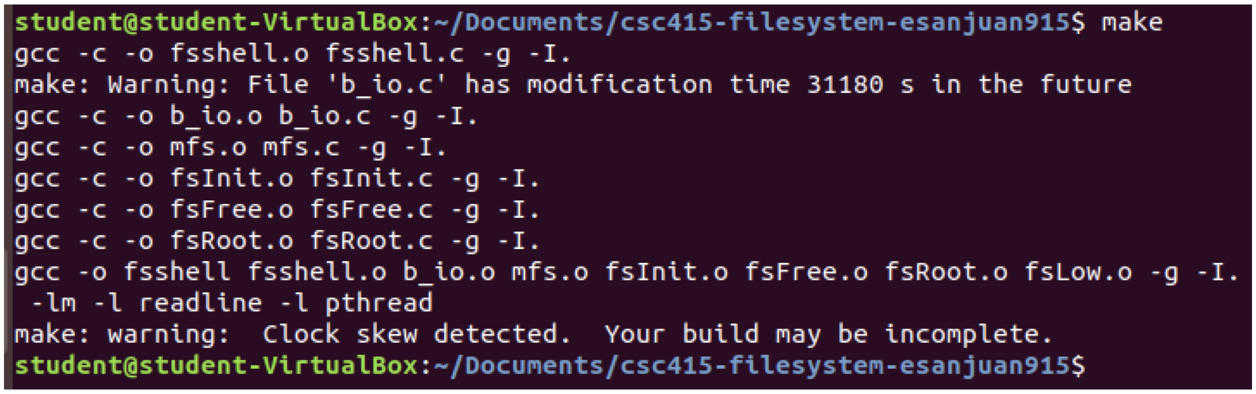
3. Detail of how driver program works:

The driver program is actually not heavily edited except where it needs to be. For the most part it stays close to the original commit. We were able to include all our functions just by using the mfs.h header file. Outside of that the only function that was really personalized was the function for moving files.

Natalie Christie, 918376646 Github: nataliekchristie
Joseph Kois, 920921089, Github: josephkois
Jade Simien, 920258687, Github: JadeS01
Eugene San Juan, 918354065, Github: esanjuan915

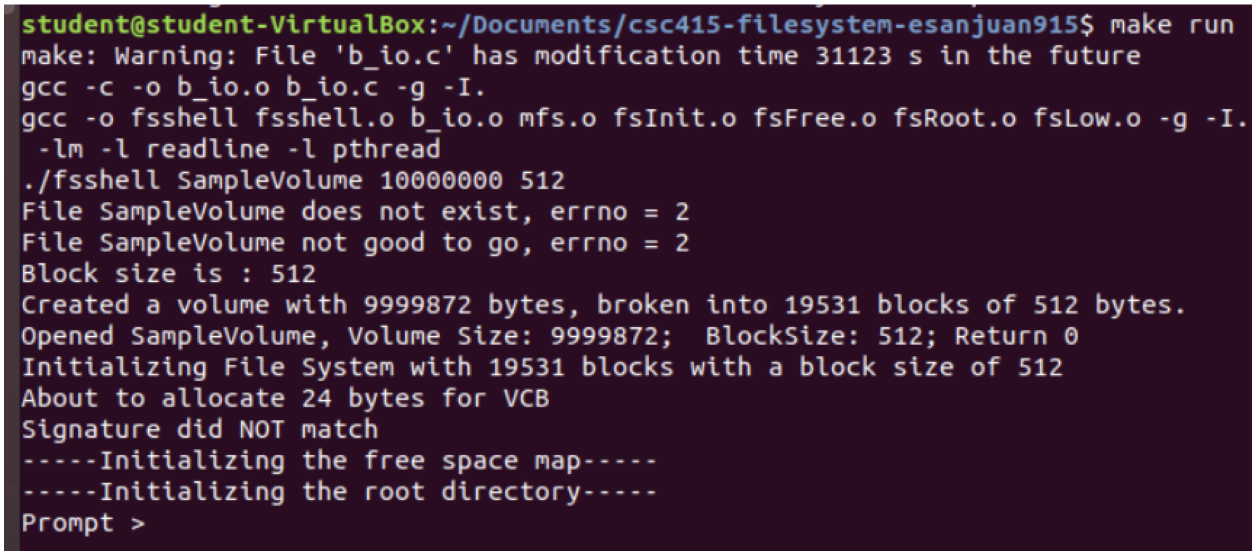
4. Screenshots:

Compilation:



```
student@student-VirtualBox:~/Documents/csc415-filesystem-esanjuan915$ make
gcc -c -o fsshell.o fsshell.c -g -I.
make: Warning: File 'b_io.c' has modification time 31180 s in the future
gcc -c -o b_io.o b_io.c -g -I.
gcc -c -o mfs.o mfs.c -g -I.
gcc -c -o fsInit.o fsInit.c -g -I.
gcc -c -o fsFree.o fsFree.c -g -I.
gcc -c -o fsRoot.o fsRoot.c -g -I.
gcc -o fsshell fsshell.o b_io.o mfs.o fsInit.o fsFree.o fsRoot.o fsLow.o -g -I.
 -lm -l readline -l pthread
make: warning:  Clock skew detected.  Your build may be incomplete.
student@student-VirtualBox:~/Documents/csc415-filesystem-esanjuan915$
```

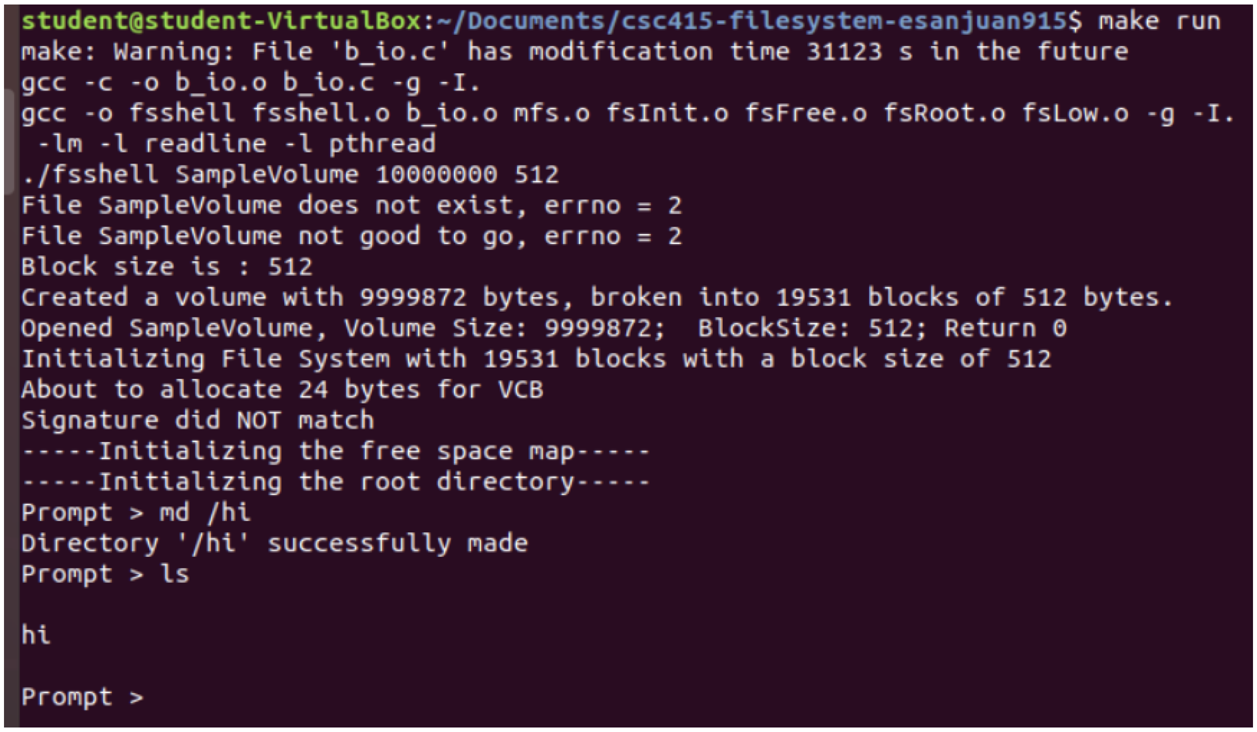
Run:



```
student@student-VirtualBox:~/Documents/csc415-filesystem-esanjuan915$ make run
make: Warning: File 'b_io.c' has modification time 31123 s in the future
gcc -c -o b_io.o b_io.c -g -I.
gcc -o fsshell fsshell.o b_io.o mfs.o fsInit.o fsFree.o fsRoot.o fsLow.o -g -I.
 -lm -l readline -l pthread
./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
About to allocate 24 bytes for VCB
Signature did NOT match
-----Initializing the free space map-----
-----Initializing the root directory-----
Prompt >
```

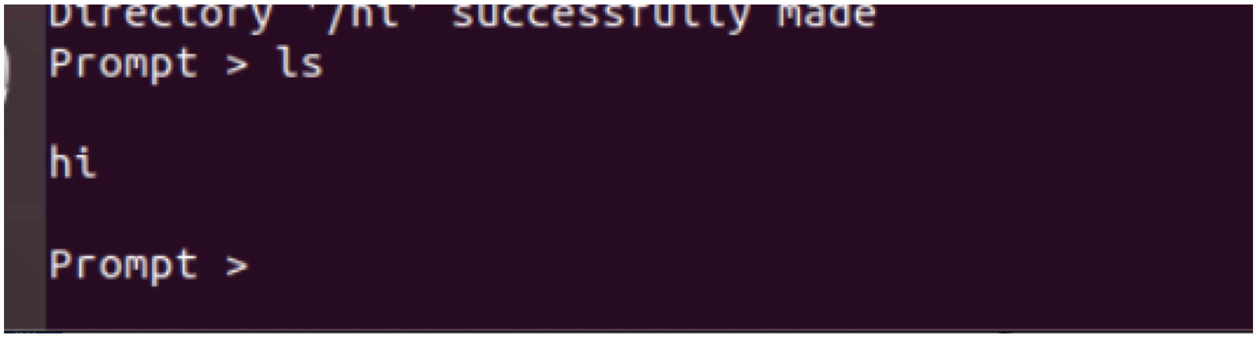
Natalie Christie, 918376646 Github: nataliekchristie
Joseph Kois, 920921089, Github: josephkois
Jade Simien, 920258687, Github: JadeS01
Eugene San Juan, 918354065, Github: esanjuan915

Md:

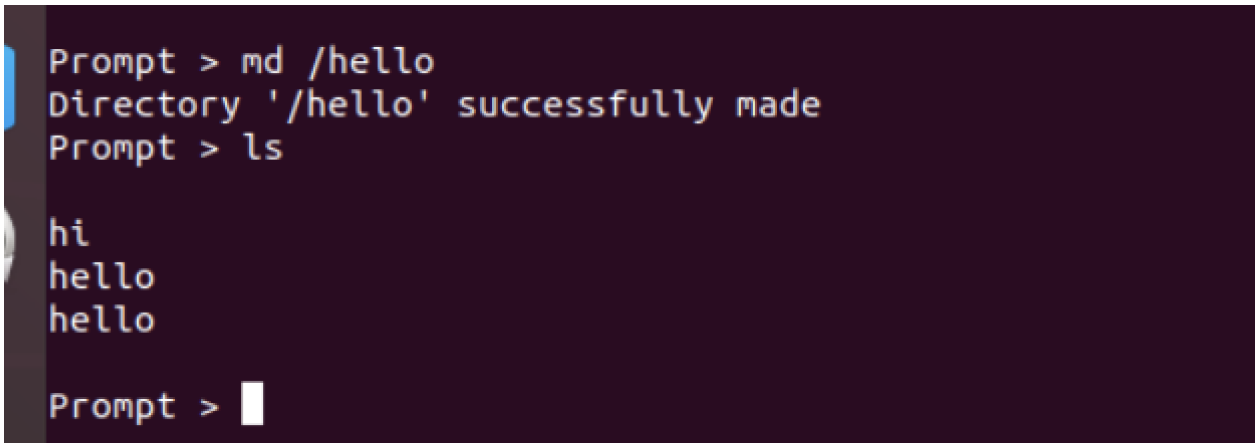


```
student@student-VirtualBox:~/Documents/csc415-filesystem-esanjuan915$ make run
make: Warning: File 'b_io.c' has modification time 31123 s in the future
gcc -c -o b_io.o b_io.c -g -I.
gcc -o fsshell fsshell.o b_io.o mfs.o fsInit.o fsFree.o fsRoot.o fsLow.o -g -I.
 -lm -l readline -l pthread
./fsshell 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
About to allocate 24 bytes for VCB
Signature did NOT match
-----Initializing the free space map-----
-----Initializing the root directory-----
Prompt > md /hi
Directory '/hi' successfully made
Prompt > ls

hi

Prompt >
```

Ls:



```
Directory '/hi' successfully made
Prompt > ls

hi

Prompt >
```

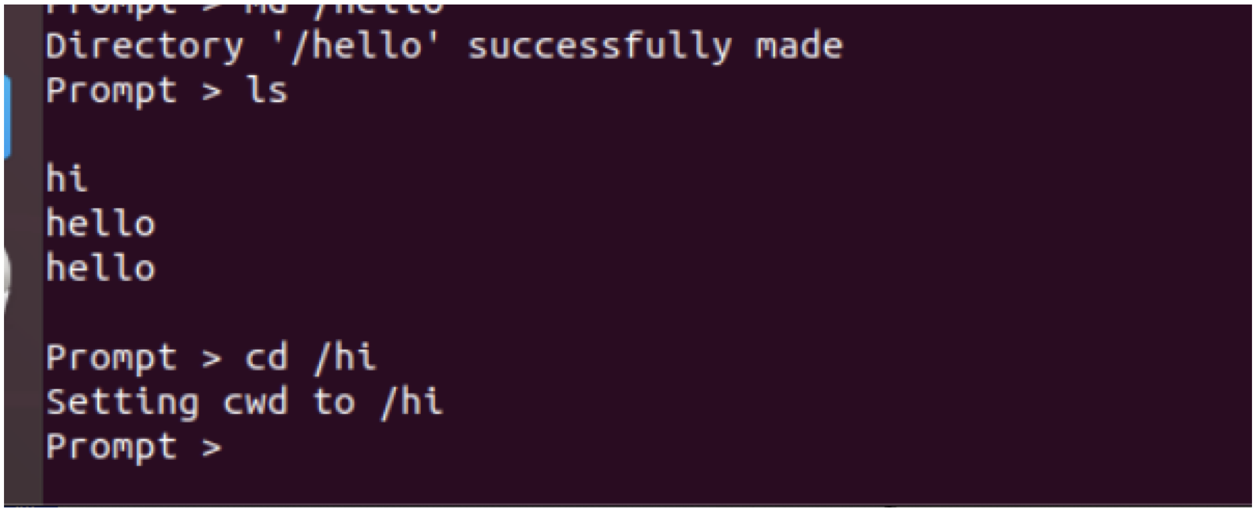


```
Prompt > md /hello
Directory '/hello' successfully made
Prompt > ls

hi
hello
hello

Prompt >
```

Natalie Christie, 918376646 Github: nataliekchristie
Joseph Kois, 920921089, Github: josephkois
Jade Simien, 920258687, Github: JadeS01
Eugene San Juan, 918354065, Github: esanjuan915

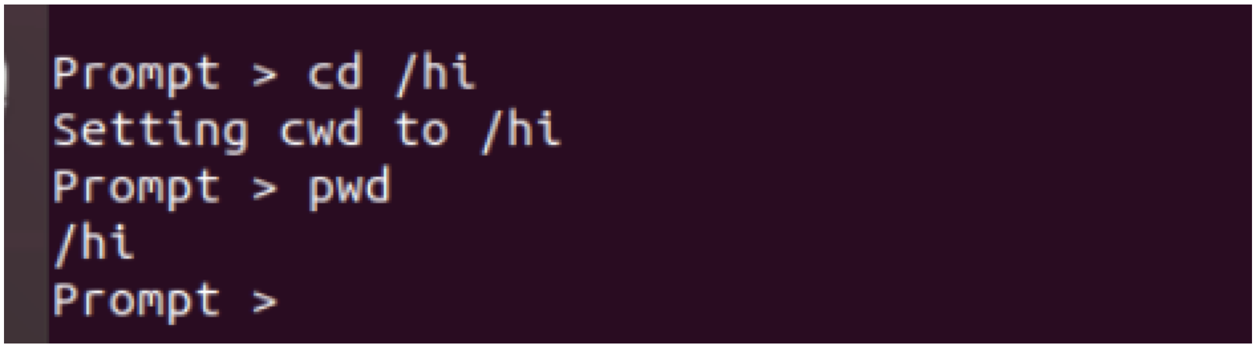
Had some issues with double printing for ls

Cd:



```
Prompt > md /hello
Directory '/hello' successfully made
Prompt > ls

hi
hello
hello

Prompt > cd /hi
Setting cwd to /hi
Prompt >
```

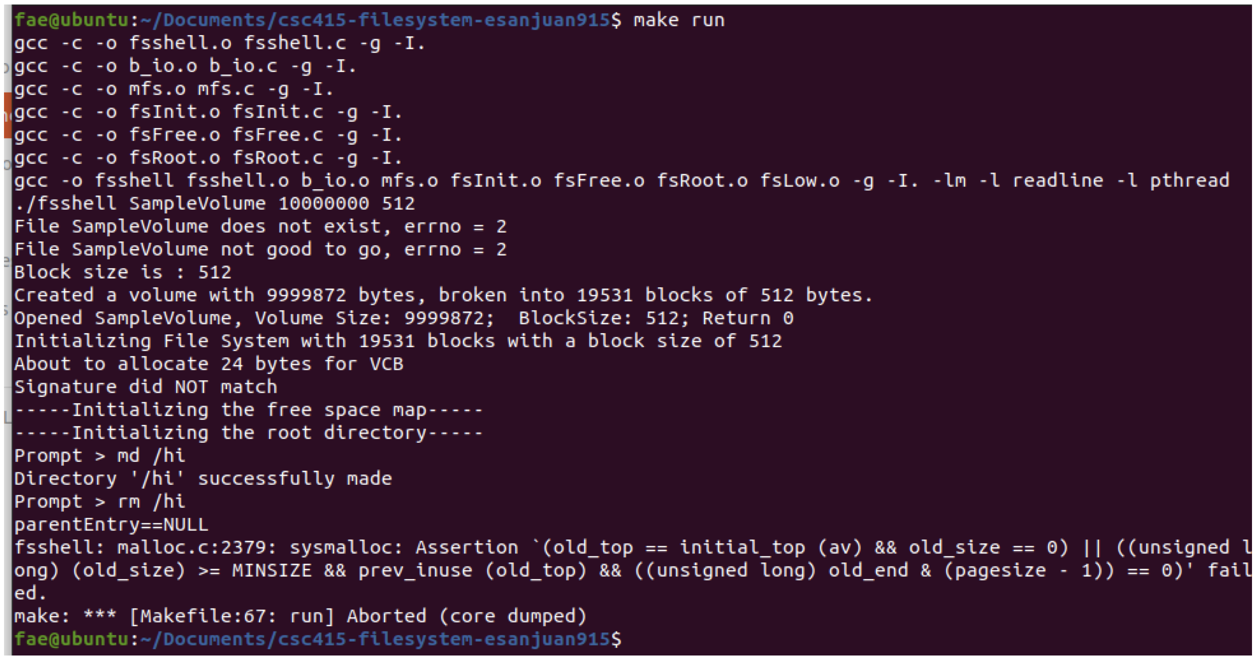
Pwd:



```
Prompt > cd /hi
Setting cwd to /hi
Prompt > pwd
/hi
Prompt >
```

Natalie Christie, 918376646 Github: nataliekchristie
Joseph Kois, 920921089, Github: josephkois
Jade Simien, 920258687, Github: JadeS01
Eugene San Juan, 918354065, Github: esanjuan915

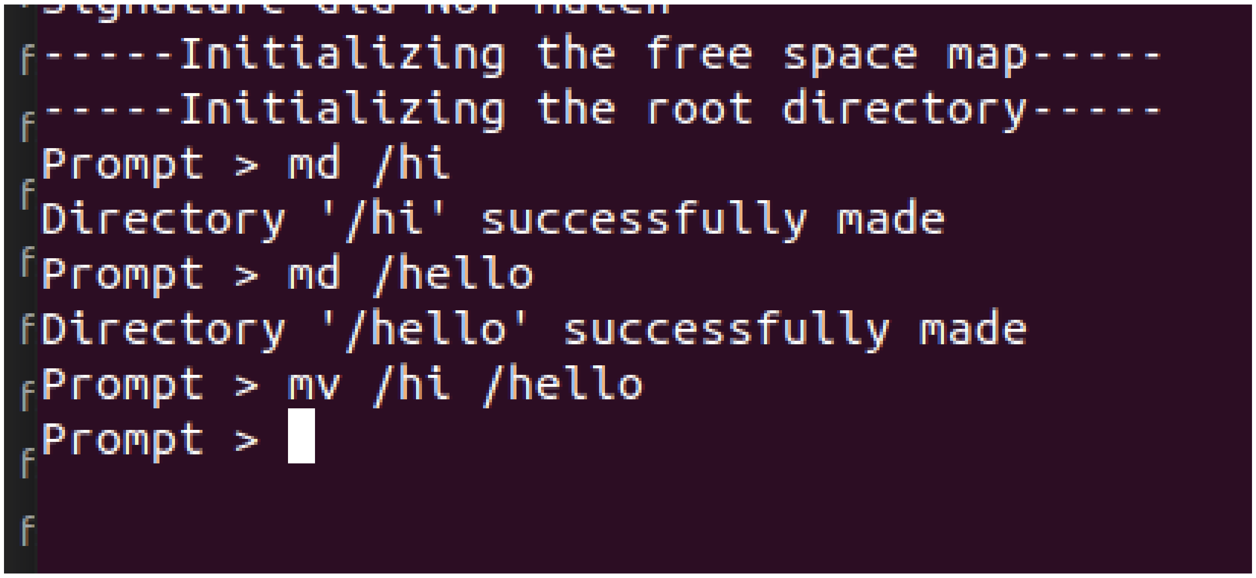
Rm:



```
fae@ubuntu:~/Documents/csc415-filesystem-esanjuan915$ make run
gcc -c -o fsshell.o fsshell.c -g -I.
gcc -c -o b_io.o b_io.c -g -I.
gcc -c -o mfs.o mfs.c -g -I.
gcc -c -o fsInit.o fsInit.c -g -I.
gcc -c -o fsFree.o fsFree.c -g -I.
gcc -c -o fsRoot.o fsRoot.c -g -I.
gcc -o fsshell fsshell.o b_io.o mfs.o fsInit.o fsFree.o fsRoot.o fsLow.o -g -I. -lm -l readline -l pthread
./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
About to allocate 24 bytes for VCB
Signature did NOT match
-----Initializing the free space map-----
-----Initializing the root directory-----
Prompt > md /hi
Directory '/hi' successfully made
Prompt > rm /hi
parentEntry==NULL
fsshell: malloc.c:2379: sysmalloc: Assertion `(old_top == initial_top (av) && old_size == 0) || ((unsigned l
ong) (old_size) >= MINSIZE && prev_inuse (old_top) && ((unsigned long) old_end & (pagesize - 1)) == 0)' fail
ed.
make: *** [Makefile:67: run] Aborted (core dumped)
fae@ubuntu:~/Documents/csc415-filesystem-esanjuan915$
```

Having issues with remove where the file is removed but it crashes

Mv:

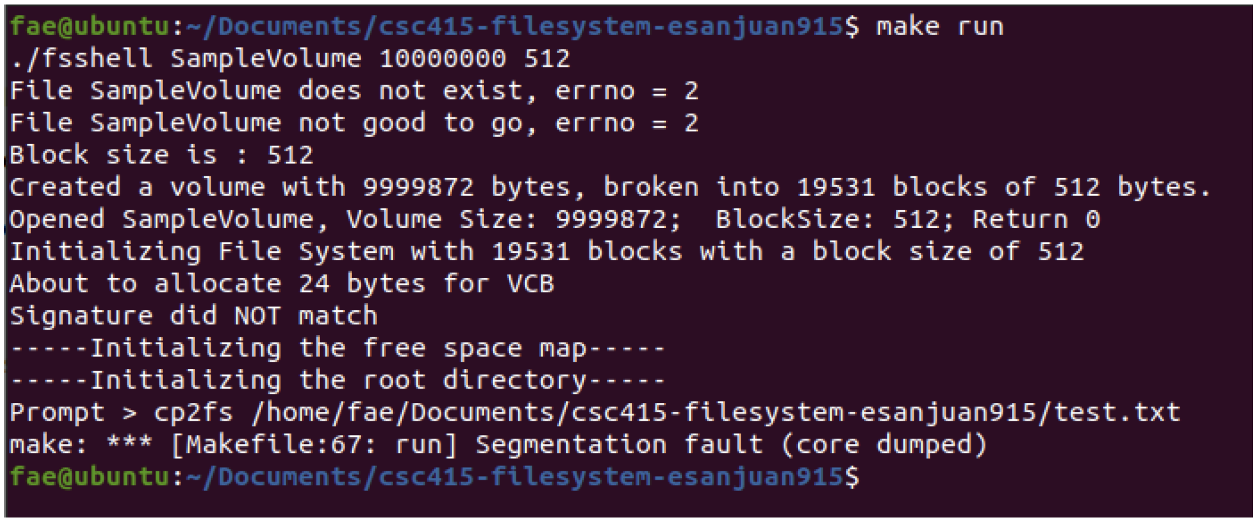


```
-----Initializing the free space map-----
-----Initializing the root directory-----
Prompt > md /hi
Directory '/hi' successfully made
Prompt > md /hello
Directory '/hello' successfully made
Prompt > mv /hi /hello
Prompt > 
```

Mv wasn’t fully implemented

Natalie Christie, 918376646 Github: nataliekchristie
Joseph Kois, 920921089, Github: josephkois
Jade Simien, 920258687, Github: JadeS01
Eugene San Juan, 918354065, Github: esanjuan915

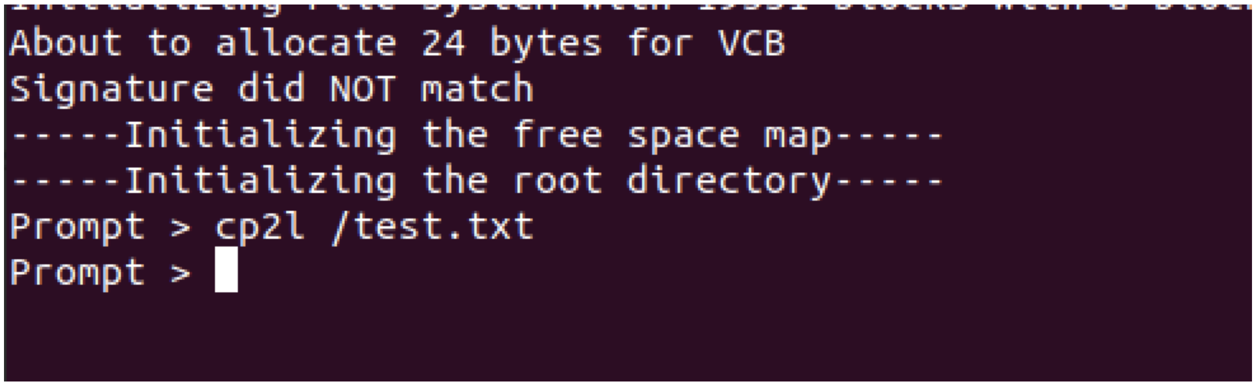
Cp2fs:



```
fae@ubuntu:~/Documents/csc415-filesystem-esanjuan915$ 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
About to allocate 24 bytes for VCB
Signature did NOT match
-----Initializing the free space map-----
-----Initializing the root directory-----
Prompt > cp2fs /home/fae/Documents/csc415-filesystem-esanjuan915/test.txt
make: *** [Makefile:67: run] Segmentation fault (core dumped)
fae@ubuntu:~/Documents/csc415-filesystem-esanjuan915$
```

Cp2fs gets segmentation fault

Cp2l:

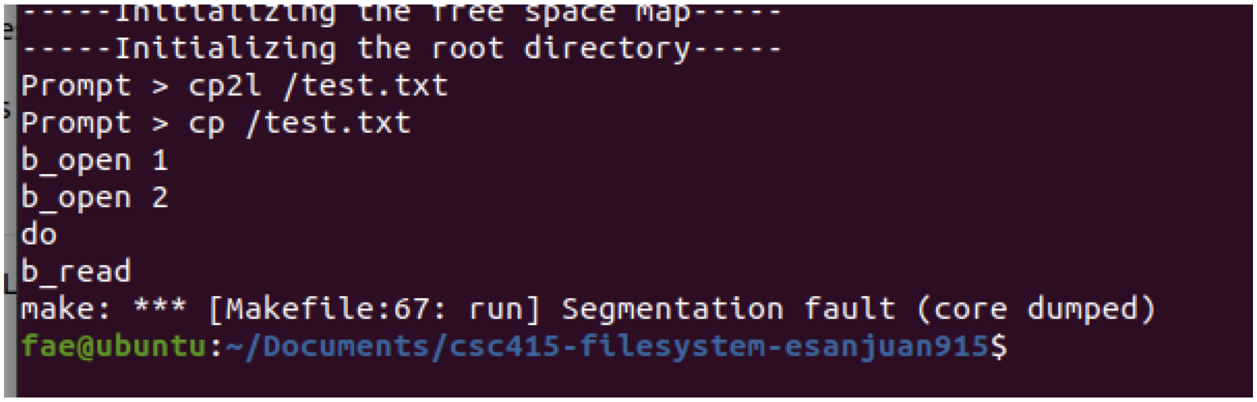


```
About to allocate 24 bytes for VCB
Signature did NOT match
-----Initializing the free space map-----
-----Initializing the root directory-----
Prompt > cp2l /test.txt
Prompt >
```

Cp2l not fully implemented

Natalie Christie, 918376646 Github: nataliekchristie
Joseph Kois, 920921089, Github: josephkois
Jade Simien, 920258687, Github: JadeS01
Eugene San Juan, 918354065, Github: esanjuan915

Cp:



```
-----Initializing the free space map-----
-----Initializing the root directory-----
Prompt > cp2l /test.txt
Prompt > cp /test.txt
b_open 1
b_open 2
do
b_read
make: *** [Makefile:67: run] Segmentation fault (core dumped)
fae@ubuntu:~/Documents/csc415-filesystem-esanjuan915$
```

Cp gets segmentation fault at read