CSC 415 File System Project

Spring 2021

Tania Nemeth, 920680290
Battulga Tsogtgerel, 920203983
Mitchel Baker, 917679066
Hanwen Geng, 920623935

Github:

https://github.com/CSC415-Spring2021/filesystemproject-best-case

Description of File System:

Our file system consists of many working pieces, all of which serve their own purposes. First and foremost, the file system's volume must be formatted correctly before we do anything else. In order to format our volume, we implemented the functions fs_init() which is called inside of our driver function driver.c. The purpose behind fs_init() is to not only format the Volume Control Block, or VCB, but to also format where the freespace and the root directory begins. Within fs_init(), we call the functions initVCB(), loadVCB(), map_initialize(), and initRootDir(). initVCB() is called first, since it is necessary to initialize the VCB with its attributes such as: numberOfBlocks, sizeOfBlock, and magicNumber. initVCB() is called first because we cannot initialize the freespace or root directory without it. After initVCB() is finished, loadVCB() is called directly after, which is where we write the initialized VCB structure into the Logical Block Array, or LBA.

Now that the VCB has been initialized and written to the LBA, we can now proceed with initializing the freespace bitmap in addition to the root directory. When calling map_initialize(), we first calculate the total number of freespace blocks required for the specified volume size, and then allocate enough memory to represent the total number of freespace blocks. We then iterate over the total number of blocks, where we then set each freespace block to 1 signifying that the block is allocated. The loop will then set the remaining number of freespace blocks to 0 signifying that these blocks are free for future allocation, whether the purpose is for the root directory, new directories, or new files created. Within map_initialize(), we also set the LBA index of freespace to 1, since we already know the VCB is written at block 0. Lastly, we then write the total number of freespace blocks into the LBA, which starts at block 1.

The next task for fs_init() is to initialize the root directory, which is accomplished with initRootDir(). The first task for initRootDir() is to create and then initialize the root directory with its basic attributes. These consist of the: root name, parent, and any entries which are stored in the root directory. After creating the root directory, it is then necessary to request and then allocate a sufficient amount of freespace blocks for the root directory. After successful allocation, initRootDir() then sets up the '.' and '..' directory entries, which are later used in the shell prompt to specify the current working directory or the parent directory of either a new directory or a new file. After '.' and '..' directory entries are set, we then write the root directory onto the LBA. After writing the root directory, we can now update the LBA index of the root directory attribute in the VCB to its block position, and then rewrite the VCB changes onto the LBA as well.

At this point in our file system, our volume has been formatted with the VCB written at block 0, the freespace bitmap written to block 1, and the root directory position at wherever the next free block is in the LBA. Now it is necessary to go back to driver.c as reference. Since fs_init() has finished, we then call temp_init() and start_shell(). temp_init() is simply a function used in mfs.c which aids in the implementation of shell commands. Then, start_shell() must be called in order to start up the shell prompt for the user. From the shell prompt, the user can input the following commands: ls, cp, mv, md, rm, cp2l, cp2fs, cd, pwd, history, and help. For the sake of reference, most of our file system command implementations are written in mfs.c, while the cp2l, cp2fs, and cp commands are implemented in b_io.c. Our ls command is written in the function fs_getcwd(), which serves the purpose of displaying to the user which directories or files are in the current working directory. On the other hand, our mv command is implemented in fs_mv(), fs_createNewPath(), and fs_removeOldEntry(). The mv command can be used to move a directory or file from its source to a new destination in the file system. Next, the rm command can be located in fs_rmdir(), which is important if a user plans on removing a directory from the file system. For the cd command, our implementation is located in fs_setcwd() which allows a user to change the current working directory to the path they specify. For the pwd command, fs_getcwd() is also used. Lastly, the history and help commands can be referenced within fsshell.c.

Issues:

-milestone one: On milestone one we received a D- for the following reasons: we used the system version of hexdump utility, we dumped the partition block and we dumped the VCB from 0x0210 to 0x027F instead of from 0x0200 through 0x03FF. We also wrote our directory entries into LBA 0 and submitted an invalid directory structure. Our milestone one feedback also stated, “you have a structure for directories that contain pointers which can not be written to disk, so this is not a valid directory structure and overall did not present a valid dump of the free space or the volume control block.” We had not correctly initialized the root directory or created a proper structure, as well as our handling of the free space and volume control block.

-milestone two:

-for ls: mainly had to work through a ton of seg faults and problems with c memory management. Properly implementing the other functions i.e., get entry from path

-for get_entry_from_path: had a lot of trouble figuring out how to handle relative vs. absolute paths, initially was using strtok but that created changes so i had to instead copy the path.

- for rmdir: had slight trouble figuring out how to get an array of entries within the root directory; had to remember to deallocate the blocks allocated for the removed entry. Rmdir was working when first pushing it, but there ended up being a memory corruption issue. We fixed it by removing old code which was no longer in use.

- buffered io: We changed linux open() read() write() functions and implemented our function. We changed file control block struct to accommodate new functions. We struggled a lot with correctly implementing the flags and luckily thanks to slack were able to add the `if(flags & O_CREAT) {`
In b_open. We also struggled a lot with memory management and forgetting to free and realized we weren't correctly allocating space for entry.

- bitmap implementation: Initially, our bitmap implementation involved changing bit values from 1 to 0. Implementing this was quite a large feat since it required having extensive knowledge of bit math and bit operations. We created an initial MVP with getBit(), setBit(), functions, but we realized after implementing these that it would be too time consuming to continue with. The first issue I faced was initializing every single bit in order to represent the freespace. I had trouble using malloc() with allocating memory for the freespace, since malloc() allocates memory in bytes. My first attempt involved converting each byte to 8 bits, but this was quite a tedious process. By the time I finished this version of the bitmap implementation, we were running short on time which is why we ended up redeveloping our bitmap implementation to be much more streamlined and simple to use.

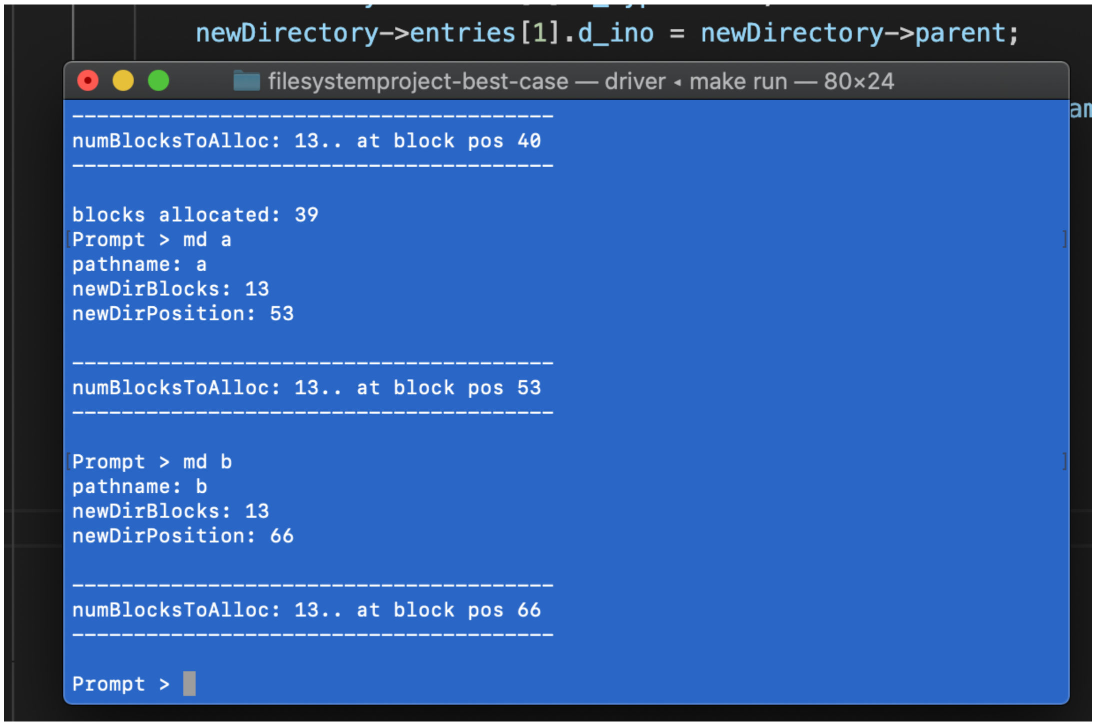
- initializing root directory: I face a lot of issues implementing the root directory, mainly because it took a lot of time to conceptualize what was needed. The first implementation I tried to follow what Professor Bierman outlined in lecture, where I attempted to follow the st4eps of allocating space for an array of directory entries, where a directory entry was a struct that represented a file. To calculate the number of bytes I used the number of directory entries * sizeof(DE) and for number of blocks the calculation was number of directory entries * sizeof(DE) + (block size -1) all over block size. Everything was initialized as free and then the . and .. entries were made, where the name had to be set, the parent, the type, the inode (root position) and then those had to be written to the file system with an LBA write. I was initially not correctly setting the parent or inode for the entries or correctly updating the index of the root directory to be the root position. In the beginning I really didn't even understand the importance of the struct elements and how they would be used so that required a LOT of rewatching the lecture videos and weekly videos and the Thursday help session and adjusting the implementation.
- Throughout the development process we had issues where the program crashes. We had to find the root cause of error and figure out what's the right way to implement that faulty piece of code.

Driver Program:

We added the header files for our file system so that the fsshell.c would work with the interfaces we implemented instead of the linux/library file routines. We turned the commands to 'on' (1) and a main function which takes three arguments: file name, volume size, and block size. Then it partitions the system and initializes the file system by calling fs_init with the volume size and block size. Temporary storage is also initialized for the volume control block and users are able to input commands in the terminal. When exit is prompted the file system is closed.

Screen Shots:

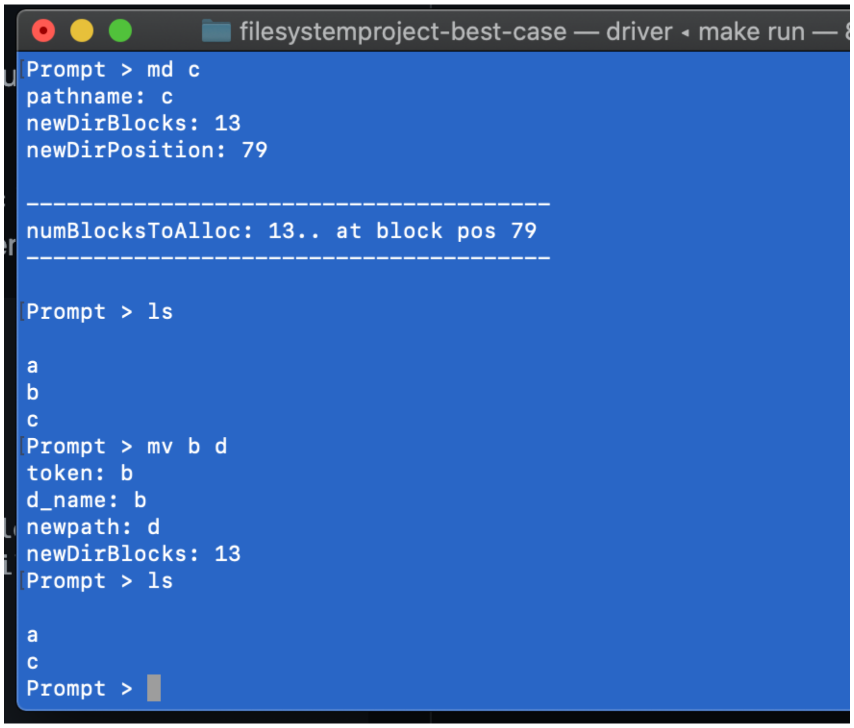
Md command:



```
newDirectory->entries[1].d_ino = newDirectory->parent;
```

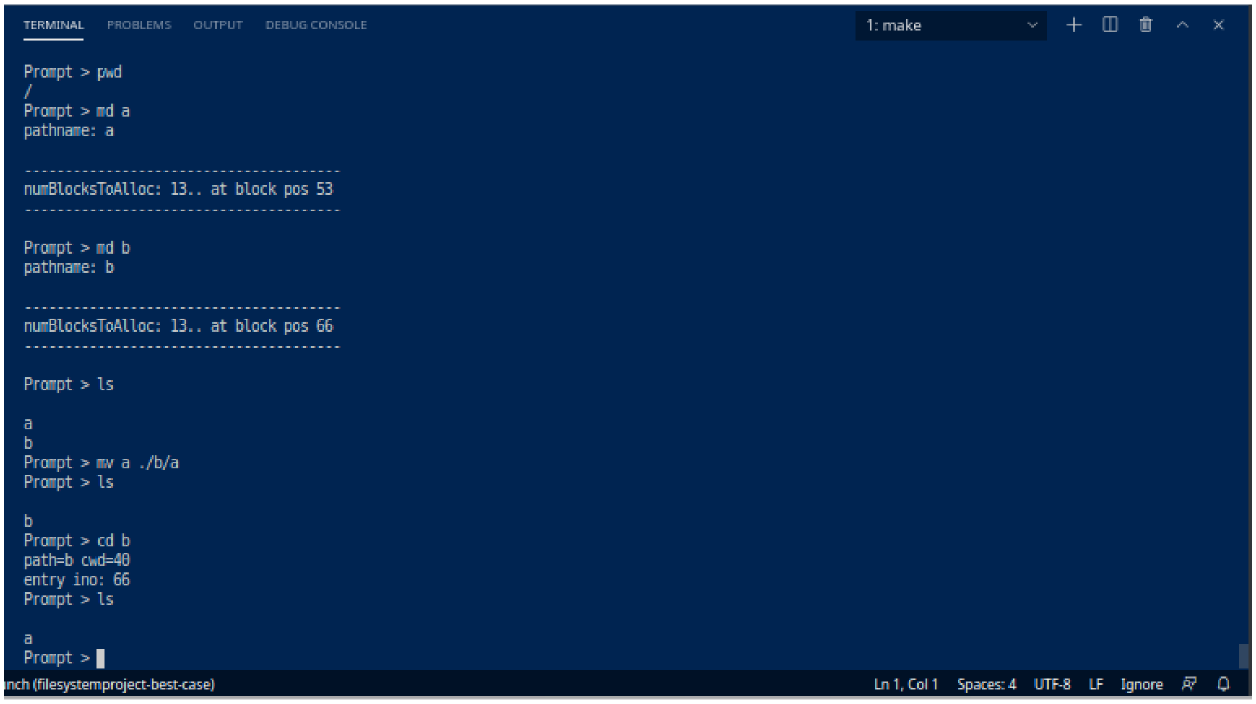
```
---------------------------------------
numBlocksToAlloc: 13.. at block pos 40
---------------------------------------

blocks allocated: 39
Prompt > md a
pathname: a
newDirBlocks: 13
newDirPosition: 53

---------------------------------------
numBlocksToAlloc: 13.. at block pos 53
---------------------------------------

Prompt > md b
pathname: b
newDirBlocks: 13
newDirPosition: 66

---------------------------------------
numBlocksToAlloc: 13.. at block pos 66
---------------------------------------

Prompt >
```

Mv command:



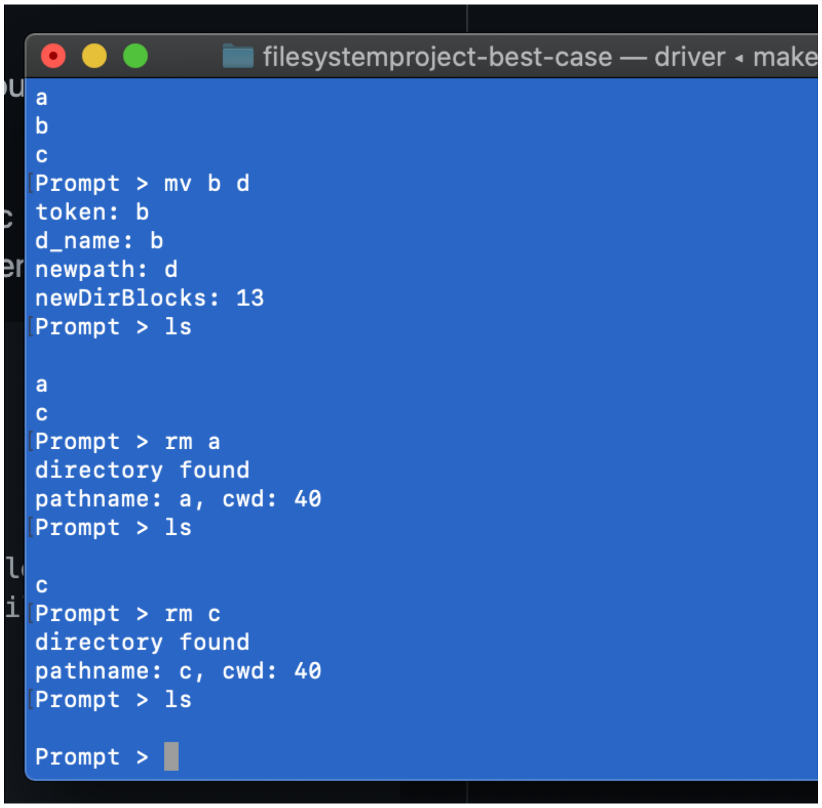
filesystemproject-best-case — driver ◂ make run —

```
Prompt > md c
pathname: c
newDirBlocks: 13
newDirPosition: 79

---------------------------------------
numBlocksToAlloc: 13.. at block pos 79
---------------------------------------

Prompt > ls

a
b
c
Prompt > mv b d
token: b
d_name: b
newpath: d
newDirBlocks: 13
Prompt > ls

a
c
Prompt >
```



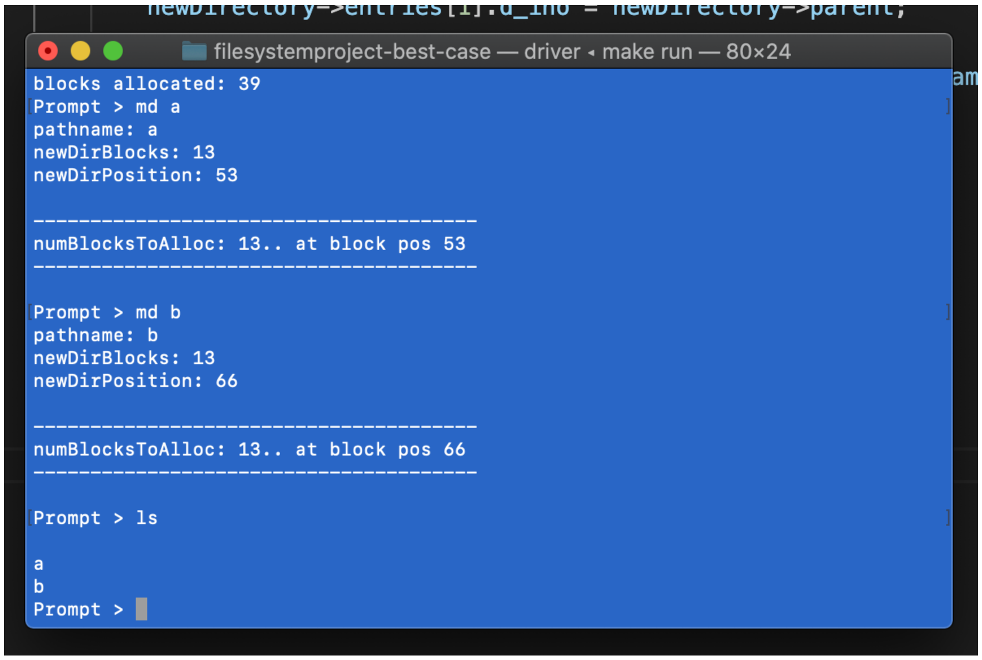
```
Prompt > pwd
/
Prompt > md a
pathname: a

---------------------------------------
numBlocksToAlloc: 13.. at block pos 53
---------------------------------------

Prompt > md b
pathname: b

---------------------------------------
numBlocksToAlloc: 13.. at block pos 66
---------------------------------------

Prompt > ls

a
b
Prompt > mv a ./b/a
Prompt > ls

b
Prompt > cd b
path=b cwd=40
entry ino: 66
Prompt > ls

a
Prompt >
```

Rm command:



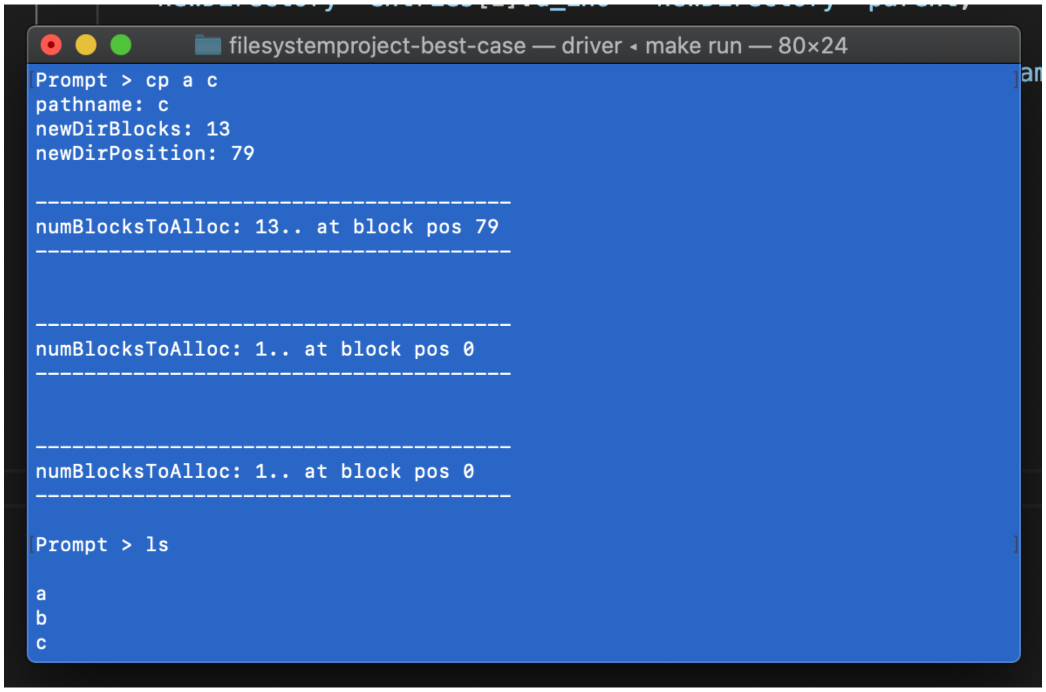
```
a
b
c
Prompt > mv b d
token: b
d_name: b
newpath: d
newDirBlocks: 13
Prompt > ls

a
c
Prompt > rm a
directory found
pathname: a, cwd: 40
Prompt > ls

c
Prompt > rm c
directory found
pathname: c, cwd: 40
Prompt > ls

Prompt >
```

Ls command:

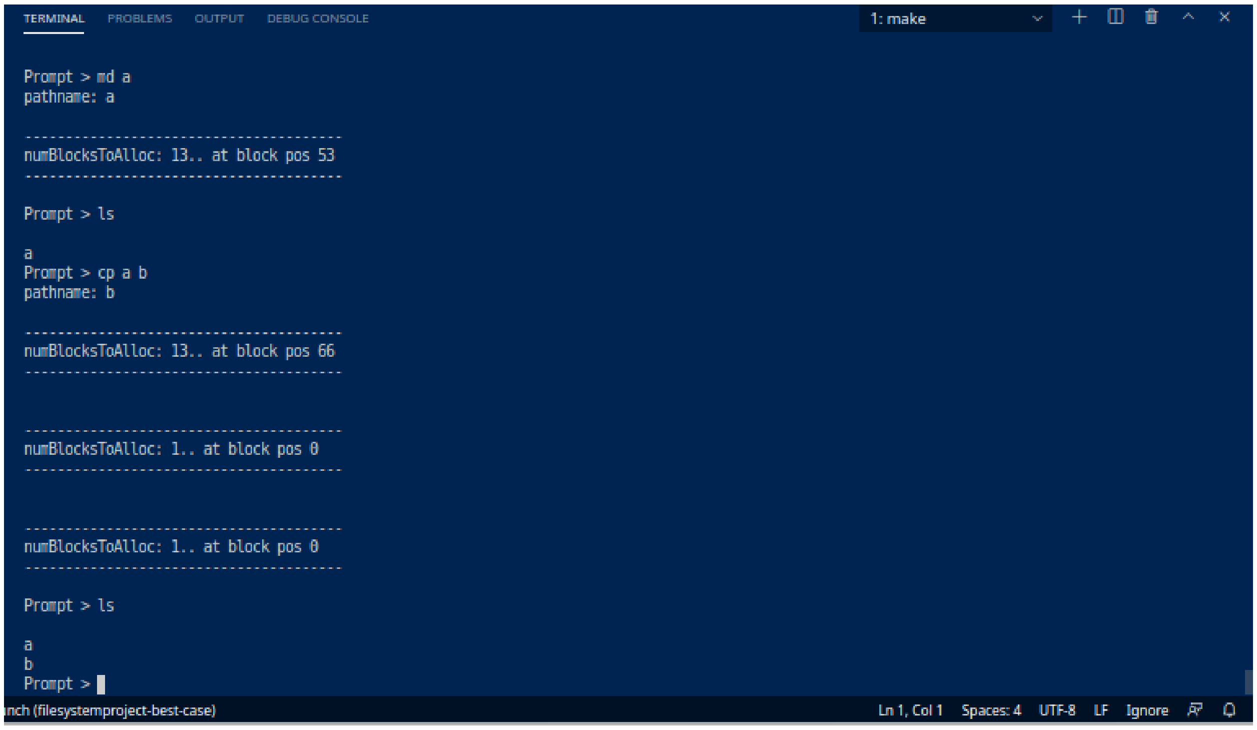


```
blocks allocated: 39
Prompt > md a
pathname: a
newDirBlocks: 13
newDirPosition: 53

---------------------------------------
numBlocksToAlloc: 13.. at block pos 53
---------------------------------------

Prompt > md b
pathname: b
newDirBlocks: 13
newDirPosition: 66

---------------------------------------
numBlocksToAlloc: 13.. at block pos 66
---------------------------------------

Prompt > ls

a
b
Prompt >
```

Cp Command:

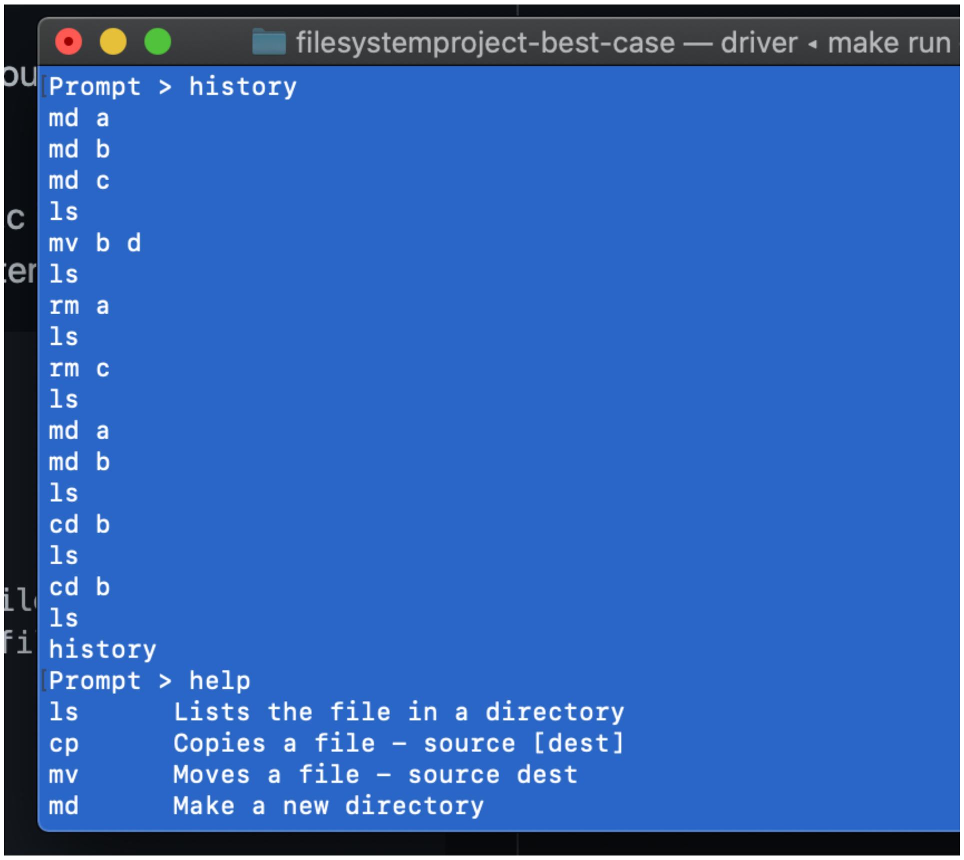


```
Prompt > cp a c
pathname: c
newDirBlocks: 13
newDirPosition: 79

----------------------------------------
numBlocksToAlloc: 13.. at block pos 79
----------------------------------------



----------------------------------------
numBlocksToAlloc: 1.. at block pos 0
----------------------------------------



----------------------------------------
numBlocksToAlloc: 1.. at block pos 0
----------------------------------------

Prompt > ls

a
b
c
```



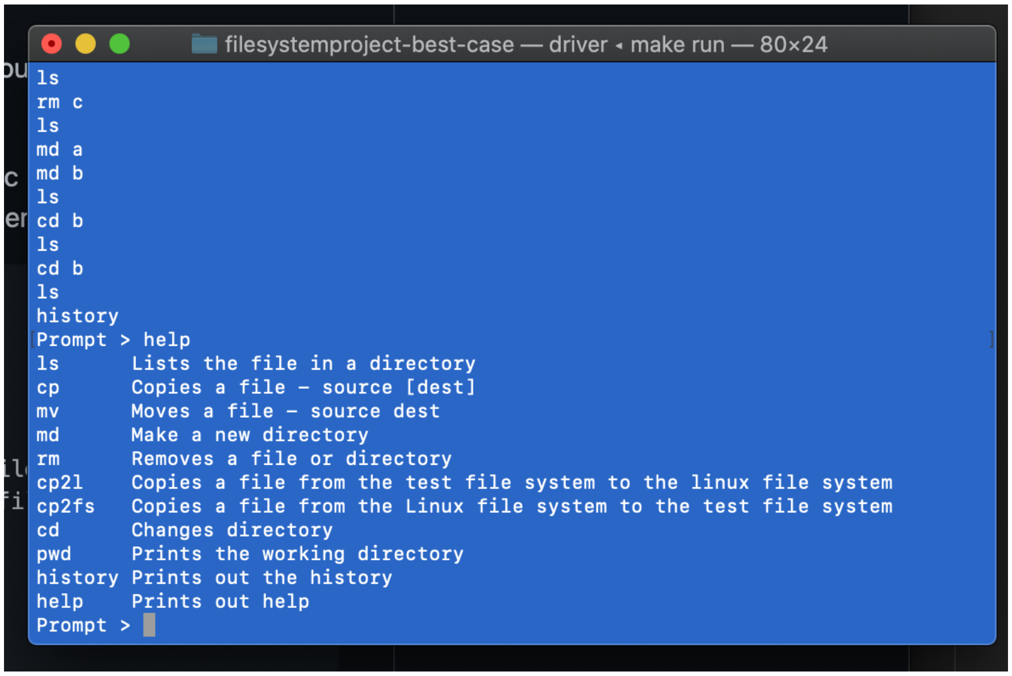
```
Prompt > md a
pathname: a

---------------------------------------
numBlocksToAlloc: 13.. at block pos 53
---------------------------------------

Prompt > ls

a
Prompt > cp a b
pathname: b

---------------------------------------
numBlocksToAlloc: 13.. at block pos 66
---------------------------------------


---------------------------------------
numBlocksToAlloc: 1.. at block pos 0
---------------------------------------


---------------------------------------
numBlocksToAlloc: 1.. at block pos 0
---------------------------------------

Prompt > ls

a
b
Prompt >
```

History:



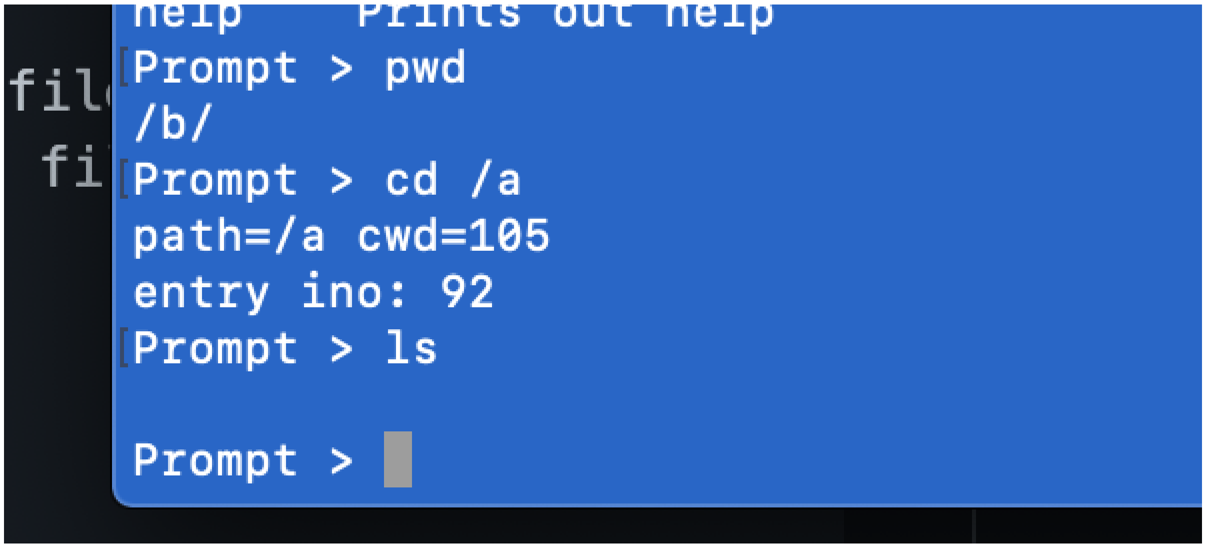
```
Prompt > history
md a
md b
md c
ls
mv b d
ls
rm a
ls
rm c
ls
md a
md b
ls
cd b
ls
cd b
ls
history
Prompt > help
ls       Lists the file in a directory
cp       Copies a file - source [dest]
mv       Moves a file - source dest
md       Make a new directory
```

Help:



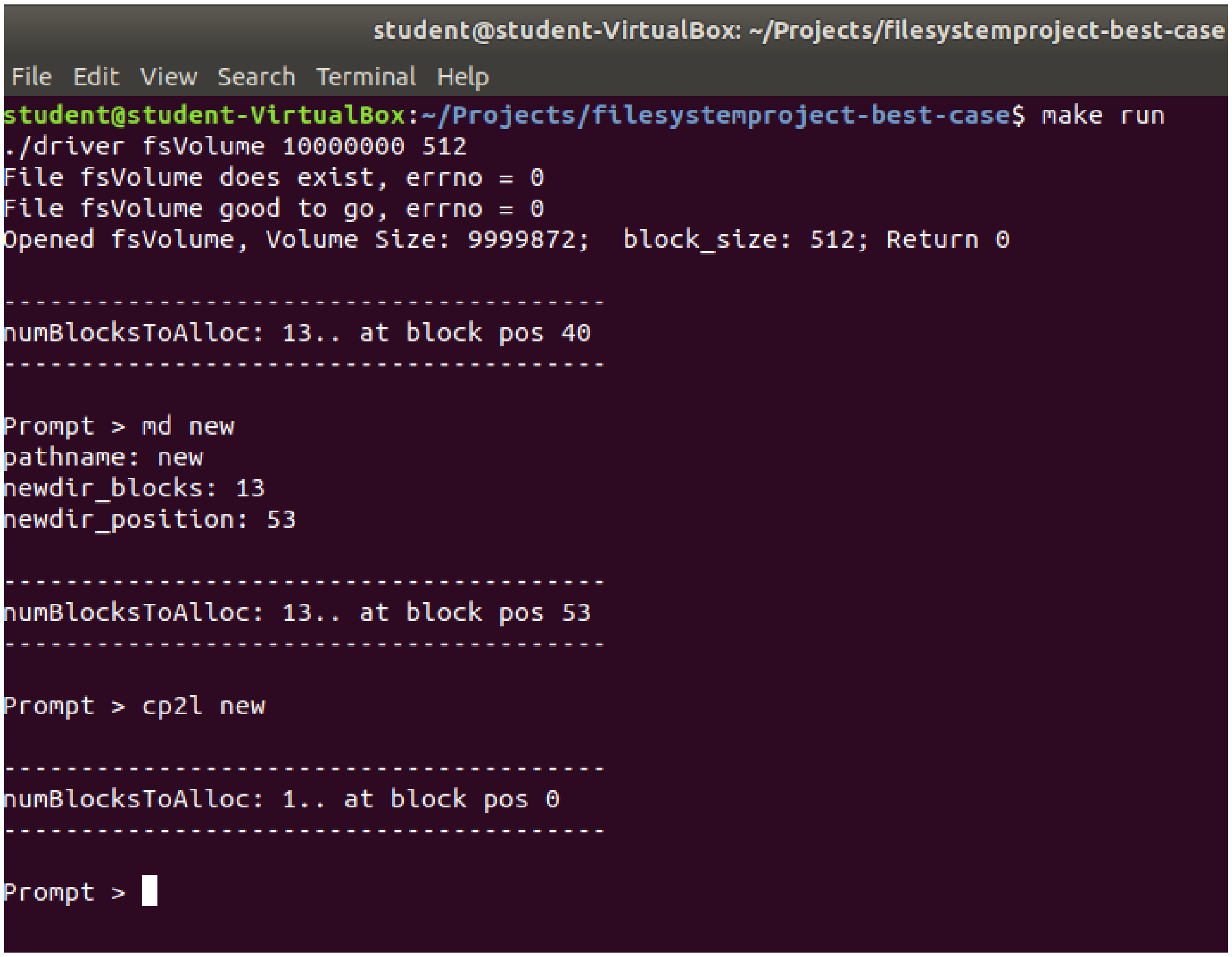
```
ls
rm c
ls
md a
md b
ls
cd b
ls
cd b
ls
history
Prompt > help
ls       Lists the file in a directory
cp       Copies a file - source [dest]
mv       Moves a file - source dest
md       Make a new directory
rm       Removes a file or directory
cp2l     Copies a file from the test file system to the 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 >
```

Pwd and Cd:



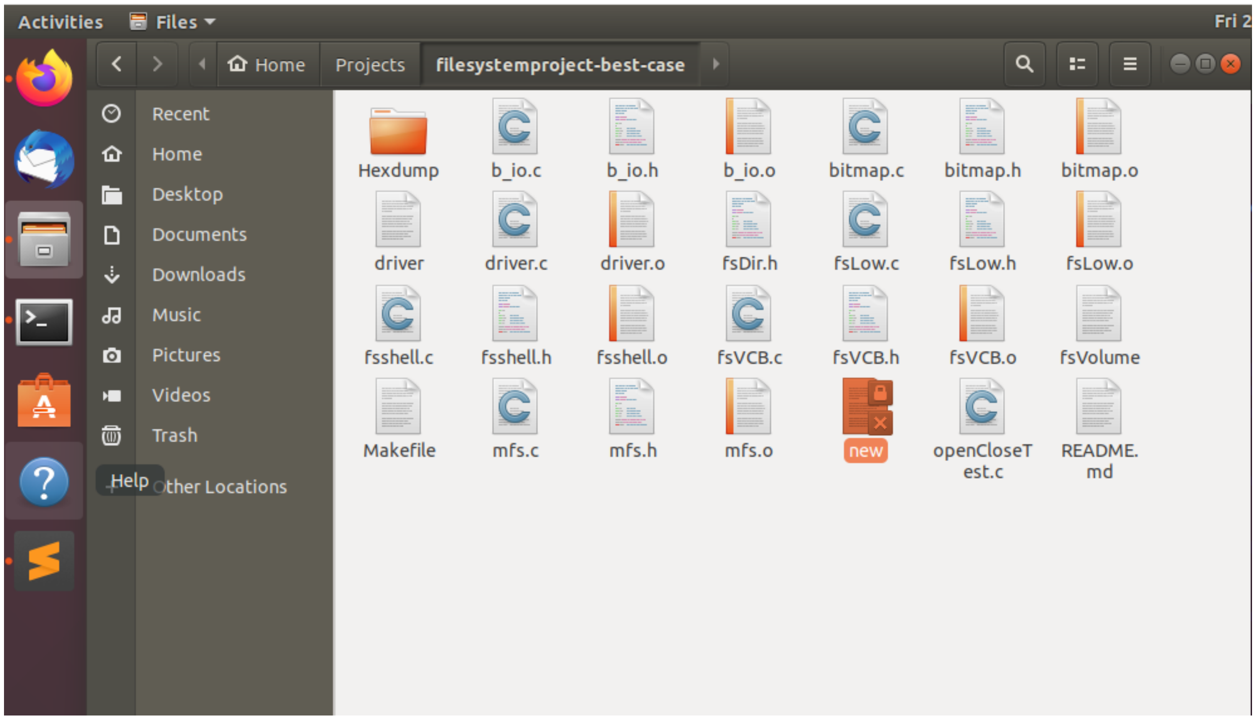
```
help      Prints out help
Prompt > pwd
/b/
Prompt > cd /a
path=/a cwd=105
entry ino: 92
Prompt > ls

Prompt > 
```

Cp2l:

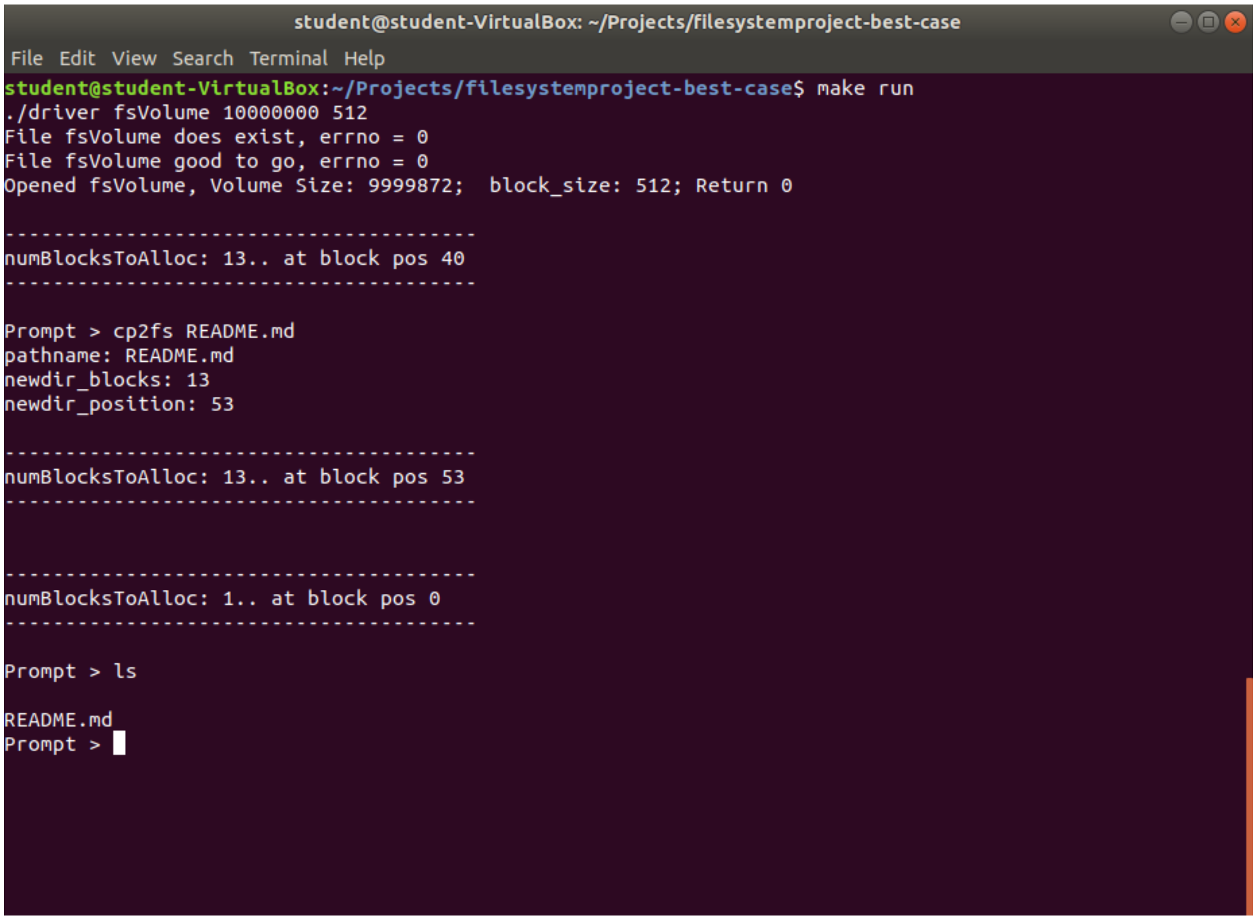


student@student-VirtualBox: ~/Projects/filesystemproject-best-case
File Edit View Search Terminal Help

```
student@student-VirtualBox:~/Projects/filesystemproject-best-case$ make run
./driver fsVolume 10000000 512
File fsVolume does exist, errno = 0
File fsVolume good to go, errno = 0
Opened fsVolume, Volume Size: 9999872;  block_size: 512; Return 0

---------------------------------------
numBlocksToAlloc: 13.. at block pos 40
---------------------------------------

Prompt > md new
pathname: new
newdir_blocks: 13
newdir_position: 53

---------------------------------------
numBlocksToAlloc: 13.. at block pos 53
---------------------------------------

Prompt > cp2l new

---------------------------------------
numBlocksToAlloc: 1.. at block pos 0
---------------------------------------

Prompt > 
```



Cp2fs:


```
student@student-VirtualBox: ~/Projects/filesystemproject-best-case
File Edit View Search Terminal Help
student@student-VirtualBox:~/Projects/filesystemproject-best-case$ make run
./driver fsVolume 10000000 512
File fsVolume does exist, errno = 0
File fsVolume good to go, errno = 0
Opened fsVolume, Volume Size: 9999872;  block_size: 512; Return 0

---------------------------------------
numBlocksToAlloc: 13.. at block pos 40
---------------------------------------

Prompt > cp2fs README.md
pathname: README.md
newdir_blocks: 13
newdir_position: 53

---------------------------------------
numBlocksToAlloc: 13.. at block pos 53
---------------------------------------


---------------------------------------
numBlocksToAlloc: 1.. at block pos 0
---------------------------------------

Prompt > ls

README.md
Prompt > 
```