Raza Ali

Amy Li

Rhetta Mae Sevilla

Naweeda Qeyam

CSC 415-03

December 4, 2020

File System Project

Team CSC415-GroupProject

Github Link: https://github.com/CSC415-Fall2020/group-term-assignment-file-system-rmsevilla

Description:

The purpose of this assignment was for students to work together to design and create a working file system. Students' file systems required to have a formatted volume, maintain a free space management system, initialize a root directory, maintain directory information, create, read, write, and delete files, and display info.

For maintaining free space, we planned to use a bitmap to implement the directory, and allocate free space due to its simplicity in comparison to using a linked list, grouping, or counting. The bitmap keeps track of the free disk blocks that are utilized to allocate space when files are created. When files are deleted, the disk blocks related to that file, will then be released for it to be reused by another new file. The bitmap scans a block of bits for a non-zero sector. A sector that has a 1, indicates that a disk block is in use, while a zero would indicate a free disk block. There are three functions within the bitmap.c file: setBitMap, clearBitMap, and findBitMap. The setBitMap function allocates a file to a free disk block, clearBitMap releases a disk block that has been associated to a deleted file, and findBitMap checks to see if a disk block is in use if the bits equate to a non-zero number.

After setting up our bitmap for free space allocation, we began to format the file system's disk. To format the disk, we started off by implementing the VCB.c (volume control block allocation file) first. We initialized variables and stored all the necessary data. Then we started implementing reading, writing, VCB calling the LBAread/LBAwrite which takes a buffer, a count of LBA blocks and the starting LBA block number, so it is important to check if the buffer is large enough for the number of blocks and the block size. These functions return the number of blocks read or written. FreeVcb, getFreeBlock which returns the first free block available, readVCB into memory by calling the LBAread.

B_io files: B_io.c is used to open, read, and write our file. There are various flags that are used in certain functions, and a struct of the file control block (FCB). We also created an array of size 20 that will hold all the files being created with a struct of the FCB at each index. The file holds the functions; b_init(), b_getFCB(), b_open(), b_write(), b_read(), and b_close().

The function b_init() will simply initialize each item in the array of files we created with default file descriptors, the mode the file is in, and set the buffer size for each file. b_getFCB will get the free file in the FCB array. If there are no free files available it will return -1.

B_open() will try to open a file, but before it can, it has to check whether or not the FCB array is initialized. Once initialized it then checks if there is already another FCB open. If there is, it will return -1. Otherwise, it will continue and check if the file we are opening exists. If it doesn't exist, it will then check the flag to see if we are creating a file. If we are making a new file, it will set all default values of the struct for that file. Once we know we have an existing file, we can allocate space for the buffer. But then we will have to check if malloc() was successful or not. If it fails, then we close the file and mark it as a free file again. Otherwise, it will set the number of bytes read and index to 0 to indicate it hasn't been read yet, and we return the file descriptor.

The b_write() function copies bytes into the buffer while keeping track of where the buffer last left off, and how many bytes can fit at the end of the current buffer to account for overflow. If the amount of bytes to write is greater than the free space left in the current buffer, it will copy a certain amount of bytes until the current buffer is full. Then it will write out everything currently in the buffer, reset the buffer, and then check if there is enough free space left in the file for the second chunk. If there is enough space, it will copy in everything from the second chunk into the buffer. Then it will return the amount of bytes that have been processed.

In b_read(), we keep track of the index to know where we are in the buffer and the total amount of bytes being copied into the buffer. B_read() first checks if the amount of free bytes remaining in the file buffer is greater than the amount requested to be read in. If the amount of free bytes remaining in the file buffer is greater, it will simply copy the amount being requested into the current buffer and return the updated amount of bytes requested. Otherwise, it will copy until the current buffer is full. After copying, it will check whether or not we have reached the end of the file. If we have reached the end of the file, we return the total amount of bytes written. It will also check if our current block is still within our block limit. We do an LBAread to increment the block index. Then we check how many bytes we currently have in the file, and see if it is less than the default buffer size. This is to check if we have reached the end of the file. If we are not at the end of the file, we check if the amount of bytes in the file is greater than the amount of remaining bytes left to be read. If there are more bytes in the file than the remaining amount of bytes, we copy the remaining amount of bytes to our file. Otherwise, copy a certain amount of bytes, with those amount of bytes equal to the size of the file, to the buffer. Then we finally return the total amount of bytes read.

B_close() ultimately closes the file, as the name implies. First, it checks if the mode is currently in write mode. If it is, check if there is enough space left to write to block. If there is not enough space it will exit b_close(). If there is enough space, it will write the remaining bytes left in the buffer into the file. After this, it will begin by closing the file handler, then free the buffer, and lastly change the file descriptor back to default to indicate it is a free file.

Issues we had:

While creating our file system, we have encountered many challenges that lead to a few setbacks. One of the challenges we faced was figuring out how to design our file system. We struggled with figuring out how our directory entry would look like. After doing some research, we settled on using indexed allocation for the free space allocation, because it supports direct access to file blocks and could prevent external fragmentation. We also came up with a rough idea of some structs that we may need such as a struct for metadata, super blocks, and inodes. Although we had a rough design plan, our plan was so broad that it was hard to clearly imagine how to implement it, and how files would connect to each other. As a group, we had to do more research for a couple days to get a better idea of how to set up our file system.

Another problem that we came across at the beginning of the project was the occurrence of merge conflicts. Since creating branches and merging in Github was still new to us, we had to learn how to merge without causing any merge conflicts. Whenever we had a merge conflict, we had to put aside time to come together as a group to fix it. We also had to take the time to learn how to pull requests from the master branch to ensure that our branches were up to date.

Once we came up with our file system design and learned more about git, we began to focus on creating our volume control block (VCB). Similarly to our problem when planning our design, we initially had trouble visualizing what we had to do for the VCB. We understood that the purpose of the VCB's was for it to contain volume details of files, and to manage and count free block space. However, we needed a better understanding of what kinds of functions that were required to make the VCB work. We had to take the time to research more about the VCB before making any further progress in writing code.

In addition to our problem with figuring out how to set up the VCB, we came across a problem with our MFS. When running the code, we would get errors that involved our usage of the "->" operator within the mfs.c file. The Virtual Machine returned multiple messages of the same error, "Invalid type argument of '->' (have 'int')." Although it took a bit of time, we were able to figure out why we were getting that error. At first, we thought that our error was caused by something complicated, but for this issue, it ended up being a simple fix. We forgot to add the VCB.h file into our mfs.h file, which is why the program was not recognizing the "->" operator.

As we continued working on the file system, we also came across a few segmentation faults while testing. For example during the process of using pointers there were quite a few instances where we were having difficulty having our pointers initialized correctly. One instance was when we were using a char pointer to deal with our inodes and we seemed to be dealing with a segmentation fault that we couldn't seem to figure out, later to figure out that this was due to a bad pointer. Furthermore we also had some segmentation faults related specifically to how our VCB was being initialized, making the silly mistakes of forgetting to include the correct libraries as well as including the correct header file that works with the corresponding C file.

One of the last problems that we encountered during the project involved our md and cp21 command in our file system. Although our file system ended up working, there was a slight problem when it came to creating a new directory. Once the user creates their first directory, they can't make another directory right after. When we tried to make a new directory with a different name, the program says that the directory already exists. The only way we could make a new directory was to enter the first directory, then create the new directory in it. This pattern continues whenever the user wants to create another directory, they have to enter the most recently created directory to make a new directory. Furthermore another problem we encountered in our file system is with copying over text file data to be displayed using the cp2l command. The error that keeps coming up when we use this command is that the block is out of range, and while debugging we could not seem to figure out why this was (it may have been something to do with how our b_open logic is working, but our blocksize has been initialized appropriately but still seems to not output the correct results).

<u>**How we resolved our issues:**</u>

- **Google**

Since it is not the first time people are making a file system, we were able to find resources for similar problems that other people came across. We looked at how they resolved the issue or how other people would go at resolving the issue.

- **Testing and debugging**

Printing out each individual value for our inodes to track why certain values were going off significantly in the wrong direction (some of them were significantly wrong resulting in negative values followed by segmentation dumps)

- **Lecture material**

Reviewing the posted lectures and the Thursday open secession was super helpful for listening to other students' questions, and understanding whatever we were confused about.

- **Textbook**

We read part six of the book which covers the topic about file system implementation. It helped us to get an idea about directory implementation, allocation methods, and free space management.

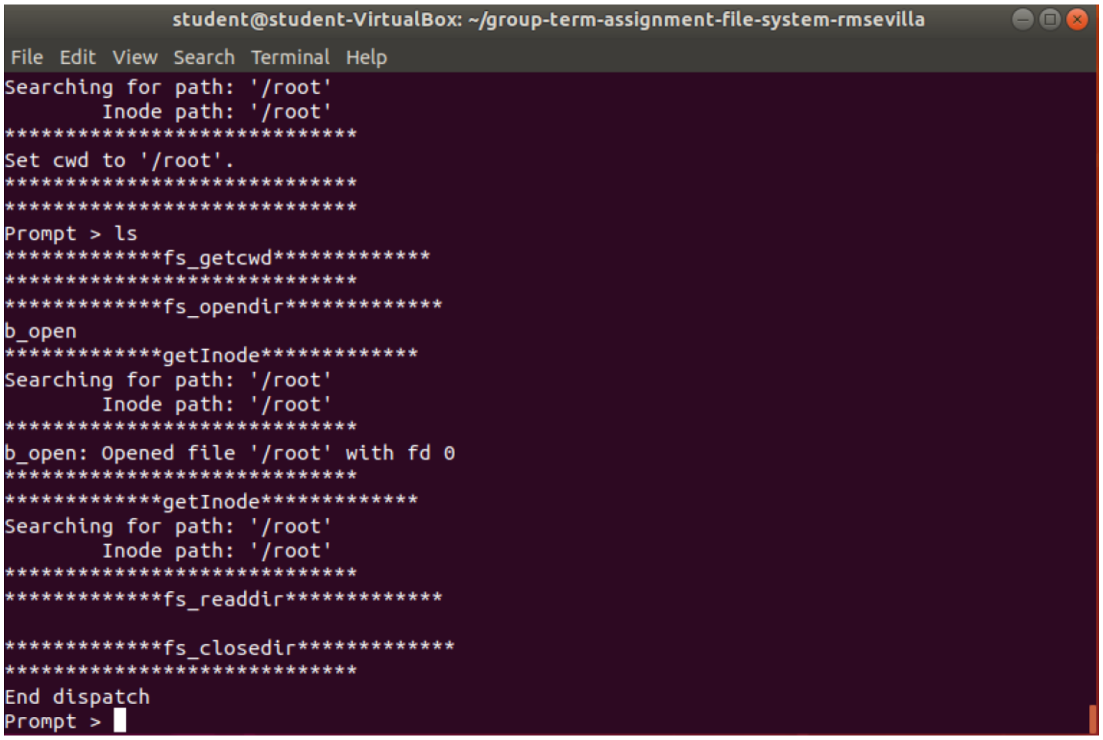
How our driver program works:

As for running and debugging our FS we made use of creating a Tester.c file with the intention of opening our volume control block, initializing all the different values for our blocks for the inodes, followed by printing out our inode values and closing the volume. The whole purpose of this was to test to see if our inodes were initialized correctly while also returning the values corresponding to those specific variables. The driver program was made easy to test our FS since we had all our values appropriately allocated earlier in our two files, (the VCB and MFS), and because of this the tester.c file only required that we properly initialized the parent, children, block pointers, and the sizes for our blocks so we can see how it all works. This was achieved through multiple print statements which included the values for the different attributes.

In order to run our program a make wipe needs to be done first to wipe our SampleVolume, followed by a make format to format the volume control block, and from there you can do a make run from in which you'll receive the prompt for the FS.

<u>Screenshots of all Commands:</u>

ls - list the files in the directory:

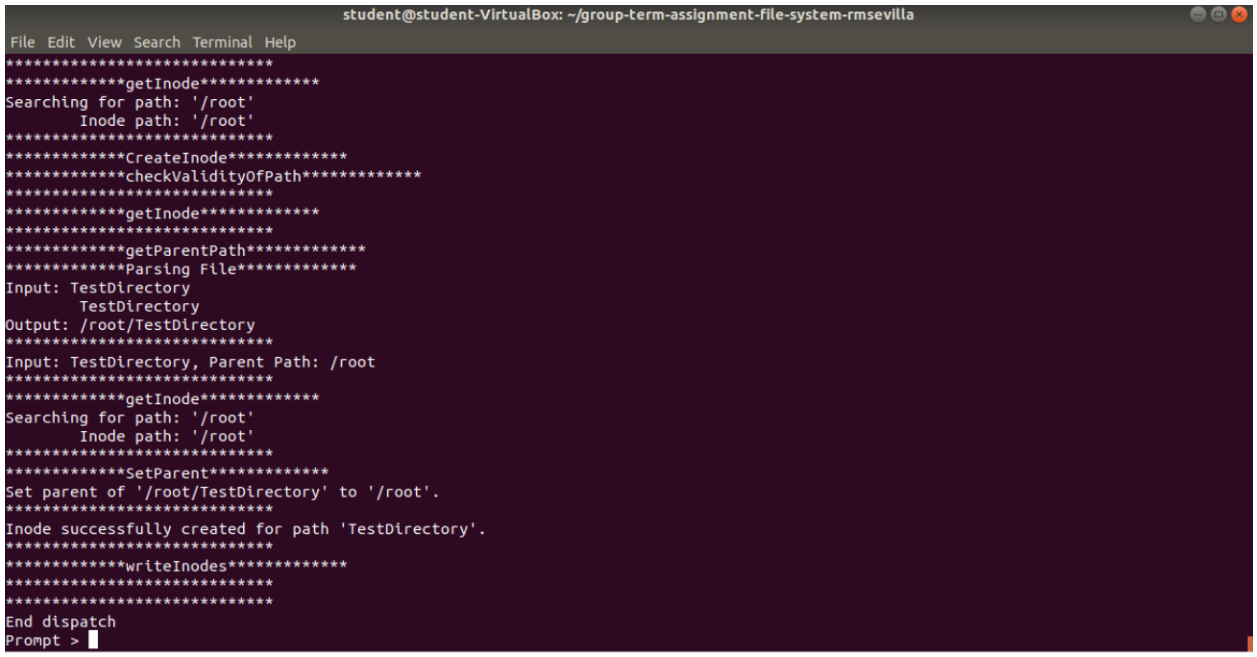

student@student-VirtualBox: ~/group-term-assignment-file-system-rmsevilla
File Edit View Search Terminal Help

```
Searching for path: '/root'
        Inode path: '/root'
*****************************
Set cwd to '/root'.
*****************************
*****************************
Prompt > ls
*************fs_getcwd*************
*****************************
*************fs_opendir*************
b_open
*************getInode*************
Searching for path: '/root'
        Inode path: '/root'
*****************************
b_open: Opened file '/root' with fd 0
*****************************
*************getInode*************
Searching for path: '/root'
        Inode path: '/root'
*****************************
*************fs_readdir*************

*************fs_closedir*************
*****************************
End dispatch
Prompt > 
```

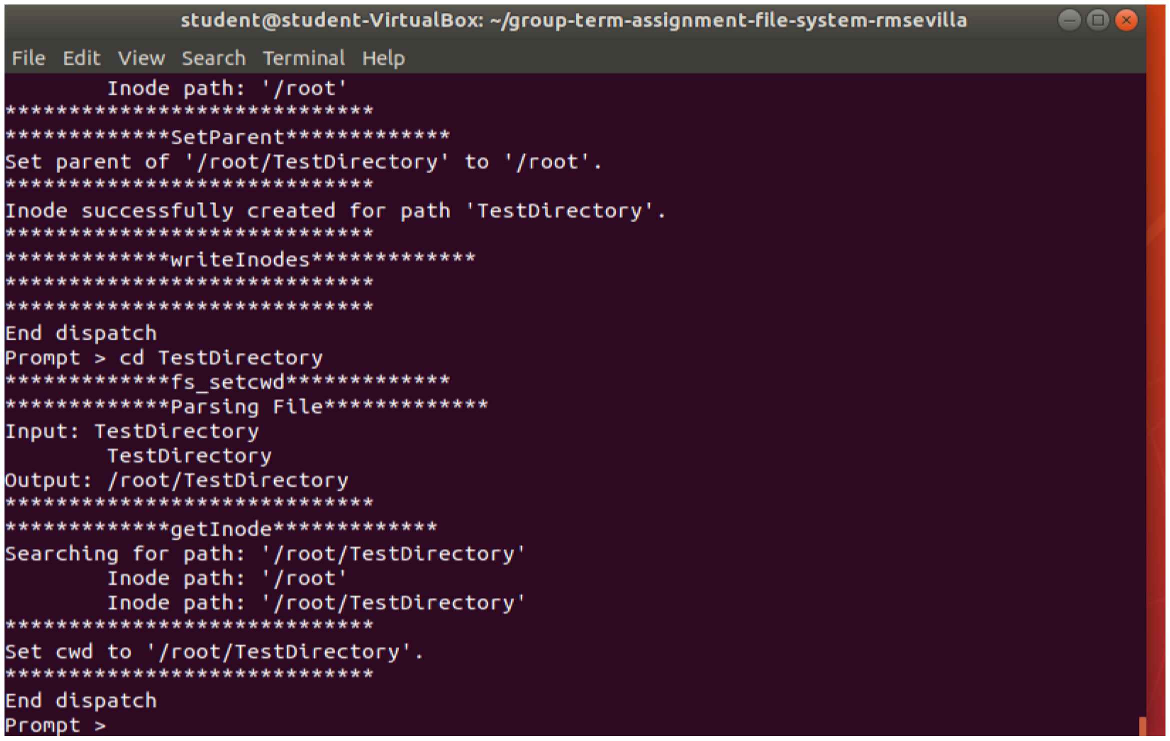
md - Make a new directory:

The command used was md TestDirectory with this corresponding output:


student@student-VirtualBox: ~/group-term-assignment-file-system-rmsevilla
File Edit View Search Terminal Help

```
*****************************
*************getInode*************
Searching for path: '/root'
        Inode path: '/root'
*****************************
*************CreateInode*************
*************checkValidityOfPath*************
*****************************
*************getInode*************
*****************************
*************getParentPath*************
*************Parsing File*************
Input: TestDirectory
        TestDirectory
Output: /root/TestDirectory
*****************************
Input: TestDirectory, Parent Path: /root
*****************************
*************getInode*************
Searching for path: '/root'
        Inode path: '/root'
*****************************
*************SetParent*************
Set parent of '/root/TestDirectory' to '/root'.
*****************************
Inode successfully created for path 'TestDirectory'.
*****************************
*************writeInodes*************
*****************************
*****************************
End dispatch
Prompt > 
```

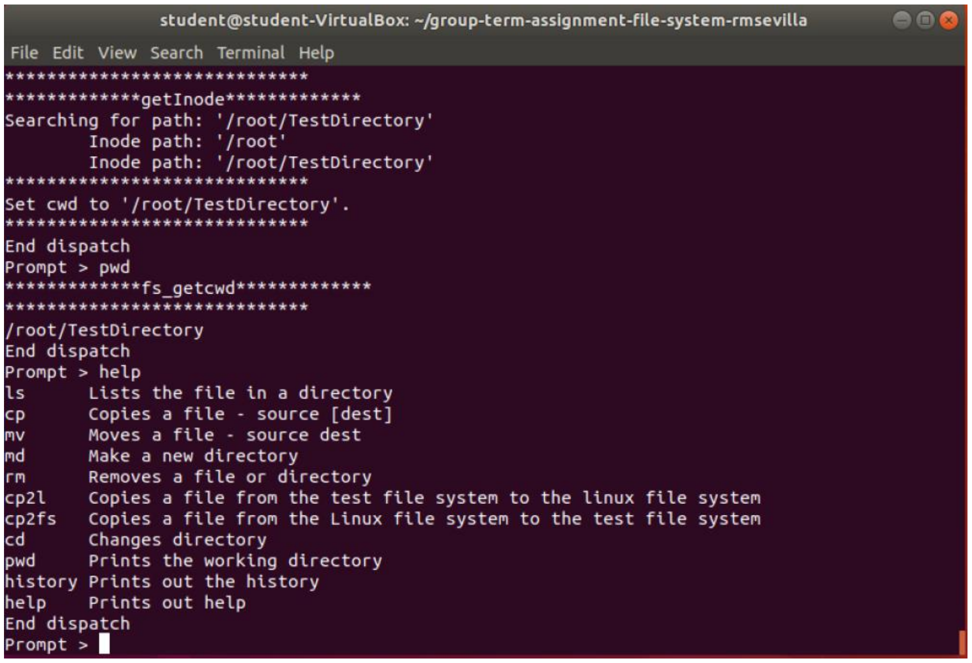
cd - Changes directory:


student@student-VirtualBox: ~/group-term-assignment-file-system-rmsevilla
File Edit View Search Terminal Help

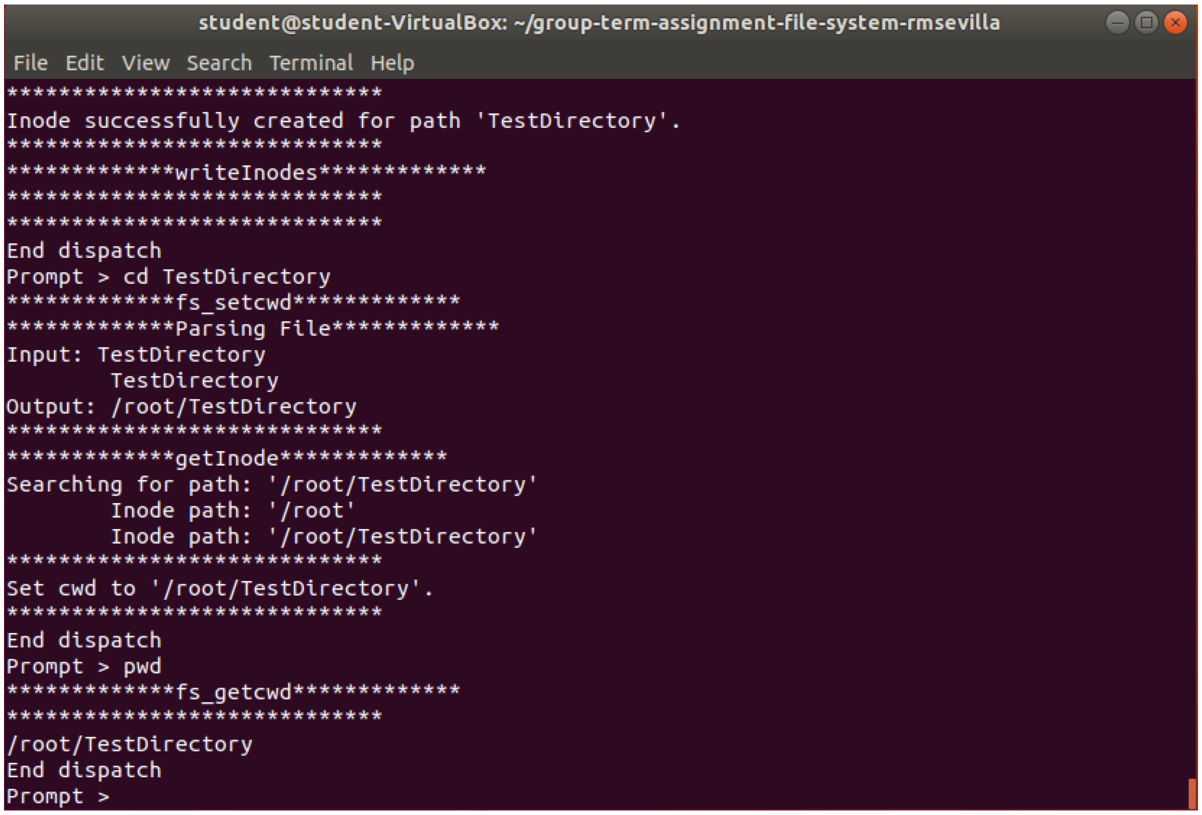
```
        Inode path: '/root'
*****************************
*************SetParent*************
Set parent of '/root/TestDirectory' to '/root'.
*****************************
Inode successfully created for path 'TestDirectory'.
*****************************
*************writeInodes*************
*****************************
*****************************
End dispatch
Prompt > cd TestDirectory
*************fs_setcwd*************
*************Parsing File*************
Input: TestDirectory
        TestDirectory
Output: /root/TestDirectory
*****************************
*************getInode*************
Searching for path: '/root/TestDirectory'
        Inode path: '/root'
        Inode path: '/root/TestDirectory'
*****************************
Set cwd to '/root/TestDirectory'.
*****************************
End dispatch
Prompt >
```

pwd - Prints the working directory (near the bottom of the terminal run you can see that pwd displays /root/TestDirectory)

Help - prints out help

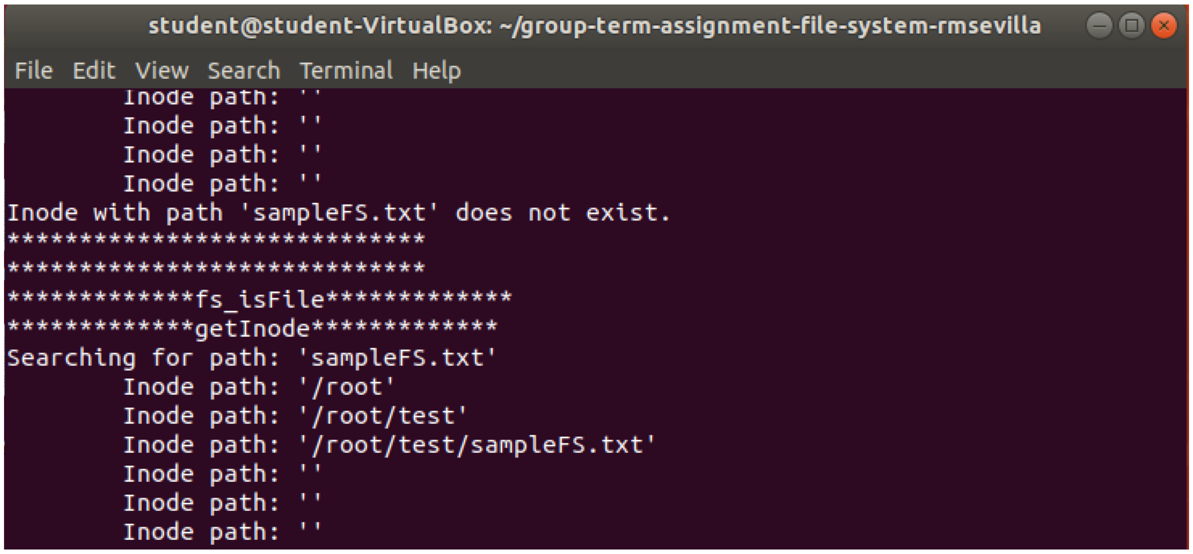

student@student-VirtualBox: ~/group-term-assignment-file-system-rmsevilla
File Edit View Search Terminal Help

```
*****************************
*************getInode*************
Searching for path: '/root/TestDirectory'
        Inode path: '/root'
        Inode path: '/root/TestDirectory'
*****************************
Set cwd to '/root/TestDirectory'.
*****************************
End dispatch
Prompt > pwd
*************fs_getcwd*************
*****************************
/root/TestDirectory
End dispatch
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
End dispatch
Prompt >
```


student@student-VirtualBox: ~/group-term-assignment-file-system-rmsevilla
File Edit View Search Terminal Help

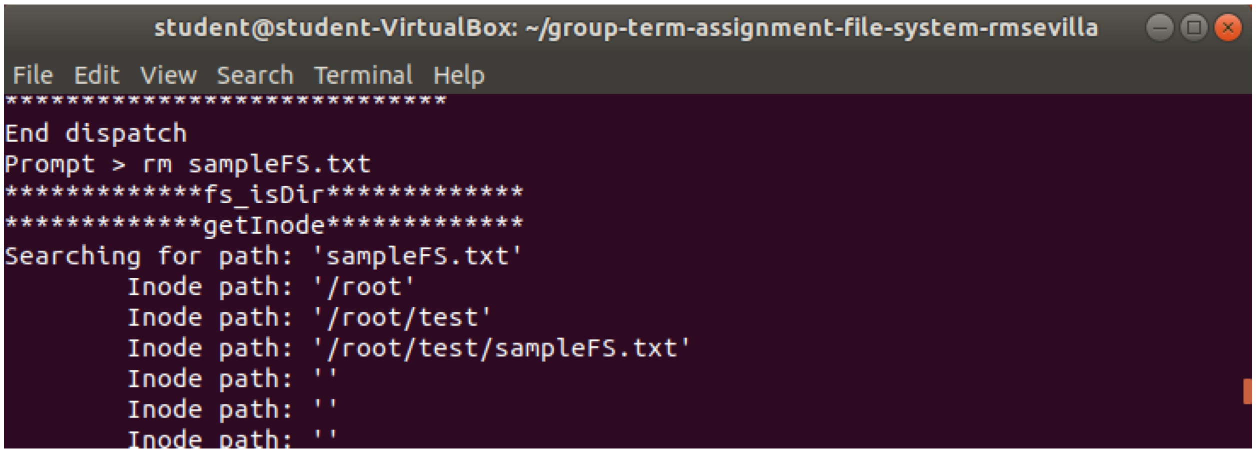
```
*****************************
Inode successfully created for path 'TestDirectory'.
*****************************
*************writeInodes*************
*****************************
*****************************
End dispatch
Prompt > cd TestDirectory
*************fs_setcwd*************
*************Parsing File*************
Input: TestDirectory
        TestDirectory
Output: /root/TestDirectory
*****************************
*************getInode*************
Searching for path: '/root/TestDirectory'
        Inode path: '/root'
        Inode path: '/root/TestDirectory'
*****************************
Set cwd to '/root/TestDirectory'.
*****************************
End dispatch
Prompt > pwd
*************fs_getcwd*************
*****************************
/root/TestDirectory
End dispatch
Prompt >
```

rm - removes file or directory:



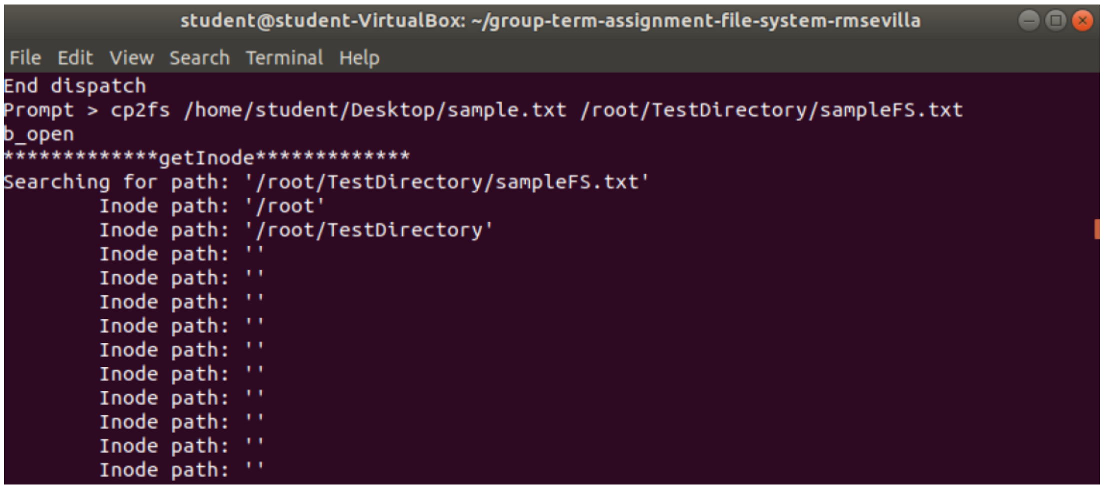
```
student@student-VirtualBox: ~/group-term-assignment-file-system-rmsevilla
File  Edit  View  Search  Terminal  Help
        Inode path: ''
        Inode path: ''
        Inode path: ''
        Inode path: ''
Inode with path 'sampleFS.txt' does not exist.
*****************************
*****************************
*************fs_isFile*************
*************getInode*************
Searching for path: 'sampleFS.txt'
        Inode path: '/root'
        Inode path: '/root/test'
        Inode path: '/root/test/sampleFS.txt'
        Inode path: ''
        Inode path: ''
        Inode path: ''
```

The file in question sampleFS.txt's inode path is tracked, and then deleted. After it's done so it returns to us that it does not exist.

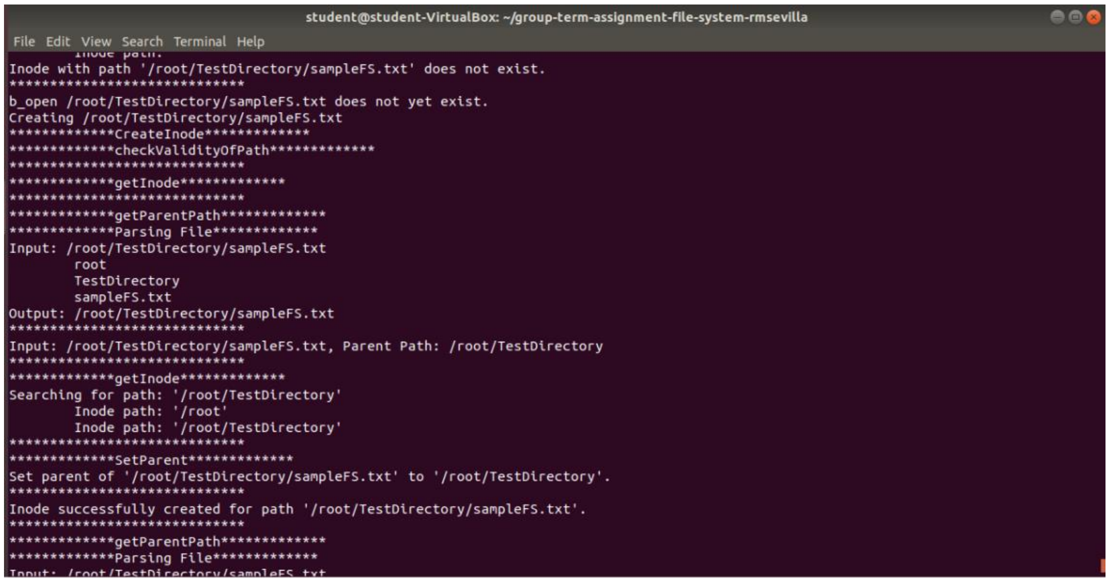


```
student@student-VirtualBox: ~/group-term-assignment-file-system-rmsevilla
File  Edit  View  Search  Terminal  Help
*****************************
End dispatch
Prompt > rm sampleFS.txt
*************fs_isDir*************
*************getInode*************
Searching for path: 'sampleFS.txt'
        Inode path: '/root'
        Inode path: '/root/test'
        Inode path: '/root/test/sampleFS.txt'
        Inode path: ''
        Inode path: ''
        Inode path: ''
```

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

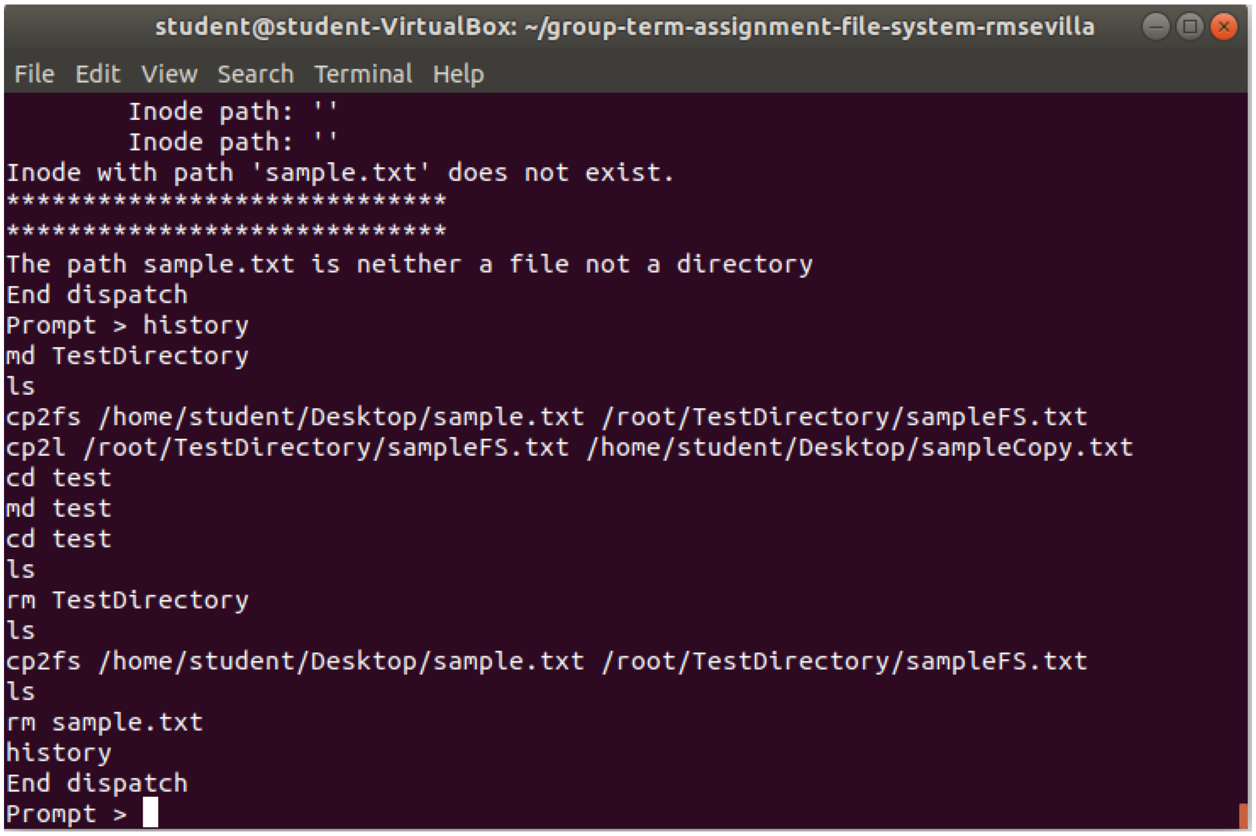


```
End dispatch
Prompt > cp2fs /home/student/Desktop/sample.txt /root/TestDirectory/sampleFS.txt
b_open
*************getInode*************
Searching for path: '/root/TestDirectory/sampleFS.txt'
        Inode path: '/root'
        Inode path: '/root/TestDirectory'
        Inode path: ''
        Inode path: ''
        Inode path: ''
        Inode path: ''
        Inode path: ''
        Inode path: ''
        Inode path: ''
        Inode path: ''
        Inode path: ''
        Inode path: ''
```



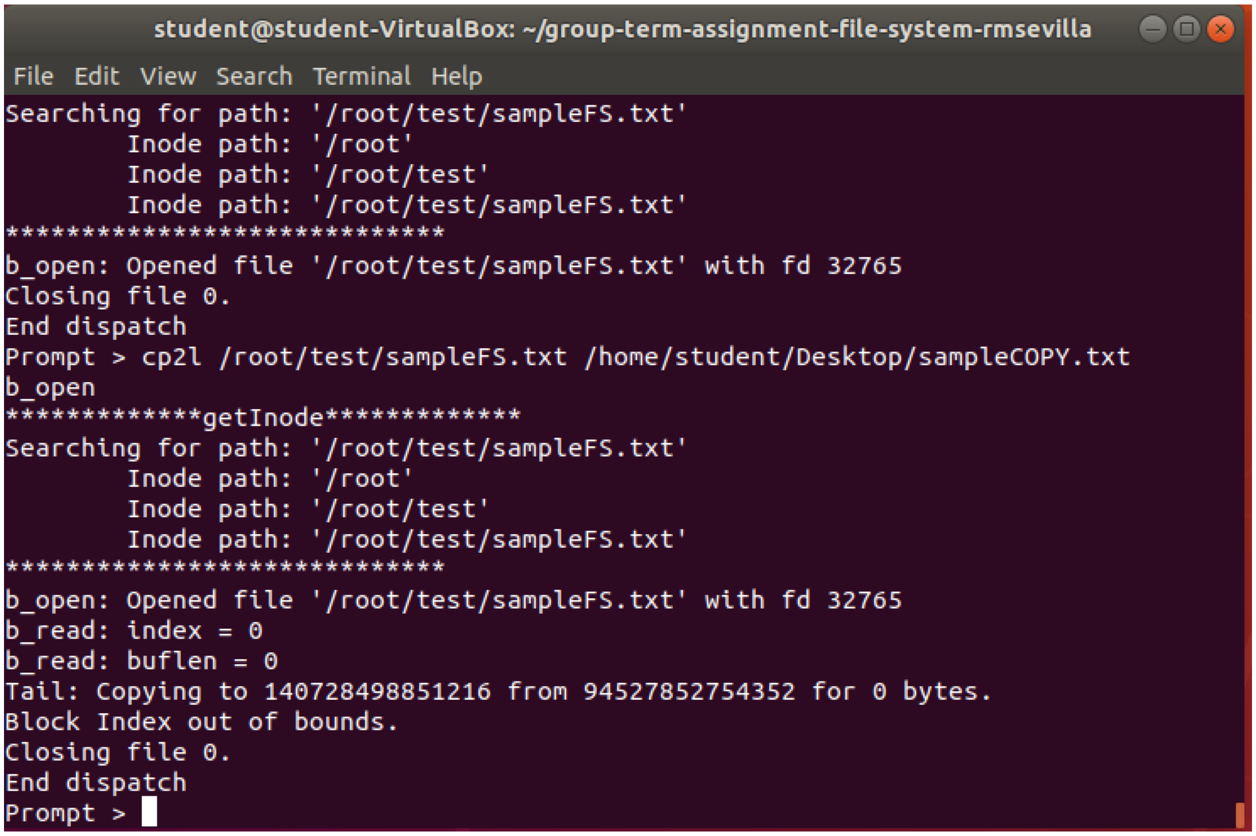
```
        Inode path:
Inode with path '/root/TestDirectory/sampleFS.txt' does not exist.
*****************************
b_open /root/TestDirectory/sampleFS.txt does not yet exist.
Creating /root/TestDirectory/sampleFS.txt
*************CreateInode*************
*************checkValidityOfPath*************
*****************************
*************getInode*************
*****************************
*************getParentPath*************
*************Parsing File*************
Input: /root/TestDirectory/sampleFS.txt
        root
        TestDirectory
        sampleFS.txt
Output: /root/TestDirectory/sampleFS.txt
*****************************
Input: /root/TestDirectory/sampleFS.txt, Parent Path: /root/TestDirectory
*****************************
*************getInode*************
Searching for path: '/root/TestDirectory'
        Inode path: '/root'
        Inode path: '/root/TestDirectory'
*****************************
*************SetParent*************
Set parent of '/root/TestDirectory/sampleFS.txt' to '/root/TestDirectory'.
*****************************
Inode successfully created for path '/root/TestDirectory/sampleFS.txt'.
*****************************
*************getParentPath*************
*************Parsing File*************
Input: /root/TestDirectory/sampleFS.txt
```

Testing the history command:


student@student-VirtualBox: ~/group-term-assignment-file-system-rmsevilla
File Edit View Search Terminal Help

```
        Inode path: ''
        Inode path: ''
Inode with path 'sample.txt' does not exist.
*****************************
*****************************
The path sample.txt is neither a file not a directory
End dispatch
Prompt > history
md TestDirectory
ls
cp2fs /home/student/Desktop/sample.txt /root/TestDirectory/sampleFS.txt
cp2l /root/TestDirectory/sampleFS.txt /home/student/Desktop/sampleCopy.txt
cd test
md test
cd test
ls
rm TestDirectory
ls
cp2fs /home/student/Desktop/sample.txt /root/TestDirectory/sampleFS.txt
ls
rm sample.txt
history
End dispatch
Prompt >
```

Cp2l (getting errors running it and we couldn't seem to figure out the issues)


student@student-VirtualBox: ~/group-term-assignment-file-system-rmsevilla
File Edit View Search Terminal Help

```
Searching for path: '/root/test/sampleFS.txt'
        Inode path: '/root'
        Inode path: '/root/test'
        Inode path: '/root/test/sampleFS.txt'
*****************************
b_open: Opened file '/root/test/sampleFS.txt' with fd 32765
Closing file 0.
End dispatch
Prompt > cp2l /root/test/sampleFS.txt /home/student/Desktop/sampleCOPY.txt
b_open
*************getInode*************
Searching for path: '/root/test/sampleFS.txt'
        Inode path: '/root'
        Inode path: '/root/test'
        Inode path: '/root/test/sampleFS.txt'
*****************************
b_open: Opened file '/root/test/sampleFS.txt' with fd 32765
b_read: index = 0
b_read: buflen = 0
Tail: Copying to 140728498851216 from 94527852754352 for 0 bytes.
Block Index out of bounds.
Closing file 0.
End dispatch
Prompt >
```

When running the cp2l command we keep getting an error for being out of bounds. Furthermore there seems to be some sort of issue with reading the file that we input in, which is strange in that the cp2fs seems to work perfectly fine.