File System Project

CSC 415.01

Fall 2020

Professor: Robert Bierman

Developers: Emin Musayev
Hareesh Pedireddi
Pramod Khatri
Vito Gano

Github link:

https://github.com/CSC415-Fall2020/group-term-assignment-file-system-harmless4you

Contents

Introduction

In this project our main goal as a group was to create a Linux File System in the C language using only C incorporated Linux calls . In order to achieve this, we as a group of 4 created and formatted a disk in which we were able to allocate 1024 bytes through directory by root . After designing the components of the file system; directory entry, volume structure, free space, we were able to implement a file system by formatting volume, creating and maintaining directory information, initializing root directory as well as maintaining information and finally then creating, reading, writing and deleting files and lastly displaying the information. **The program was able to implement basic Linux commands: ls, cp, mv, md, rm, cp2l, cp2fs, cd, pwd, history and help.**

How to Run the Project

fsshell.c is our driver program. Assuming that we are running the program through virtual studio box:

1. Make remove
2. Make clean (if the program was running before)
3. Make format
4. Make run

Implementation Details

1. **b_io.c:**

   b_io.c allows us to read and write data from our file system to linux. Compared to Assignment 2 and Assignment 5, we are tasked to create our b_io.c functions such as b_open, b_read, b_write, b_close, and b_seek. We intended to have the functions in b_io.c be able to locate the file's name and be able to open, write and store the information of that specific file into an open Logical Block Address that is implemented through the use of LBAread and LBAwrite instead of using the system call functions that Linux has.

2. **linkedListFreeSpace.c:**

   In the linkedListFreeSpace.c file, it contains the function to be able to initialize our root, as well as being able to set up our free space management. In addition, there are some functions inside the linkedListFreeSpace.c that have helper functions to be able to obtain specific information that the mfs.c needs such as being able to obtain the volume size in bytes per block as well as being able to get the root location, being able to check for free space in our volume and being able to make changes to the free space head location.

3. **volumeFormatter.c :**

   volumeFormatter.c formats the disk volume. It has one function main (), that checks for the arguments counts and copies the string pointed and then formats (create, open, close) the volume for the disk with name, size and its size.

4. **fsLow.C:**

   fsLow.c was given as a resource by the professor. This file allows reading and writing of logical blocks and acts as the main interface for the file system. First it initializes the partition and sets the file as volume. Then, it starts partitioning the file system which is called before the program starts and returns values for success or errors. After closing the partition, it does LBA write to check to see if the reading and writing functionality is beyond the capacity of the volume.

5. **fsLowDriver.c**

   fsLowDriver.c file is used as our test purpose to be able to have a better understanding of how reading and writing into blocks is implemented. By experimenting the LBAread and LBAwrite inputs and being able to figure out how it is being written/read from the block positions with the block counts, we are able to optimize our other files to

6. **fsShell.c**

   fsShell.c is our driver program for the file system project. It was one of the resources given by the professor for the project which we did not modify. displayfiles () displays the file used by ls command. Similarly, cmd_ls () list the files in the directory, cmd_cp () function copies file from one directory to another, cmd_mv () moves files from one directory to another, cmd_md () function makes directory in the present directory, cmd_rm () removes the directory either be parent or children from root and pwd. Cmd_cp2l () function copies file our file system to linux whereas cmd_cp2fs () method copies files from linux to our files system, cmd_cd () changes the working directory, cmd_pwd () tells present working directory in the shell, cmd_history () gives the history of actions performed in the shell and lastly cmd_help () displays the details of the all other commands and their functions in the shell program.

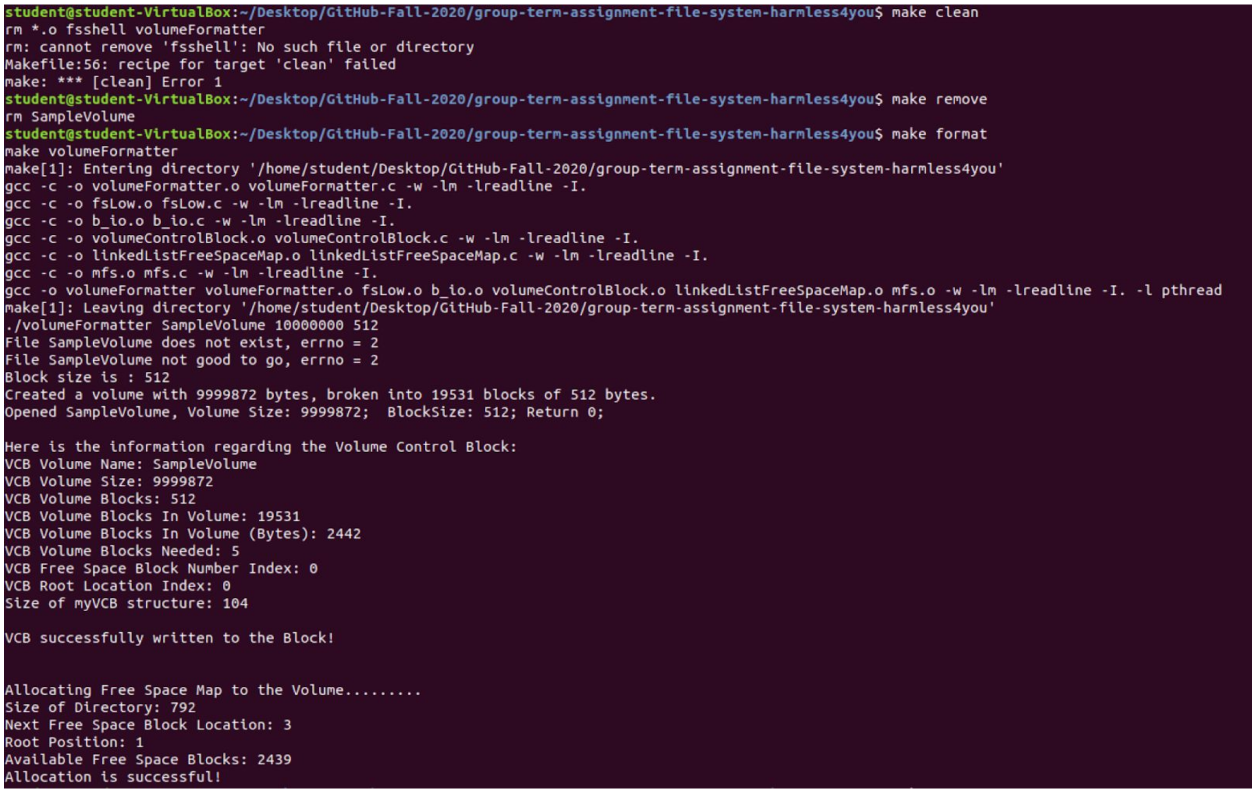
7. **Mfs.c**

In the mfs.c file, it contains all of the commands that the fsshell.c uses. For our implementation we were only able to implement the open directory function, the set current working directory function, the get current working directory function, and the make directory function. In addition, we were able to have some of the functions that the ls command uses such as the readdir function as well as the open_dir function, and the close_dir function.

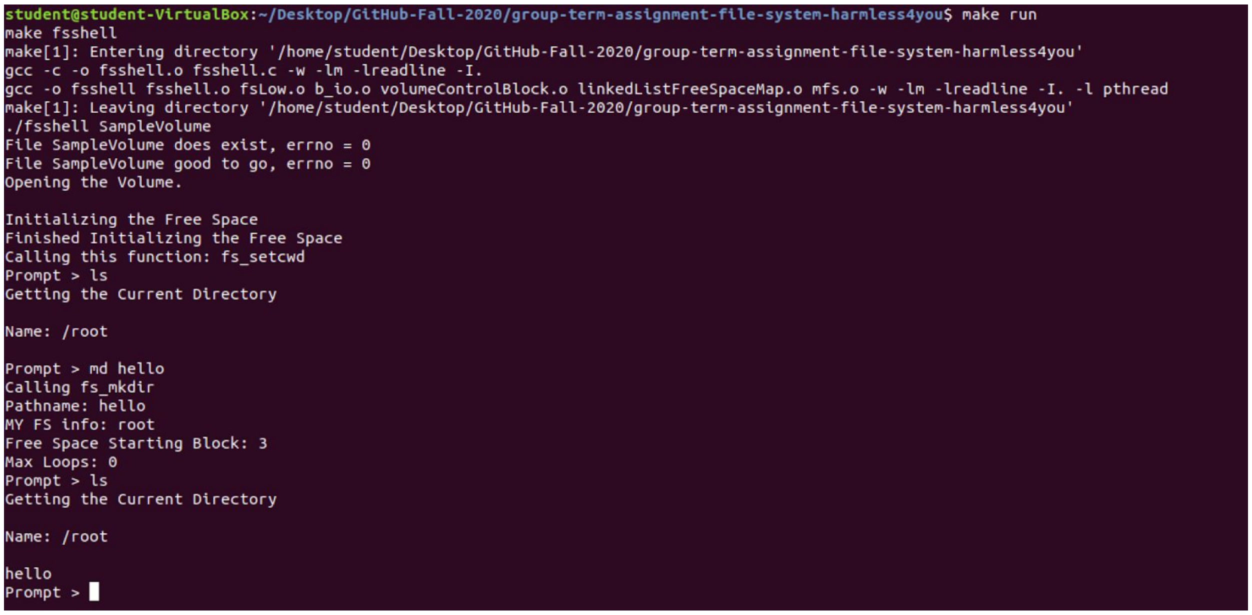
8. **volumeControlBlock.c:**

The volumeControlBlock.c is the one that stores all of the information regarding the Volume Control Block which is located starting on Block 0 of the Logical Block Addresses. In this function, we can initialize our Volume Control Block.

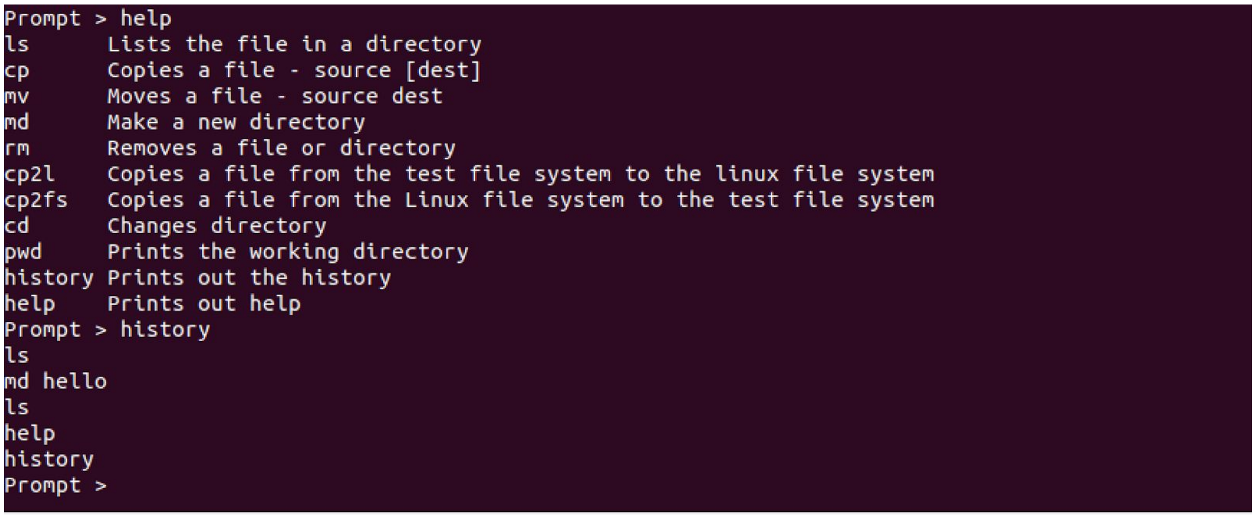
Screenshot Output



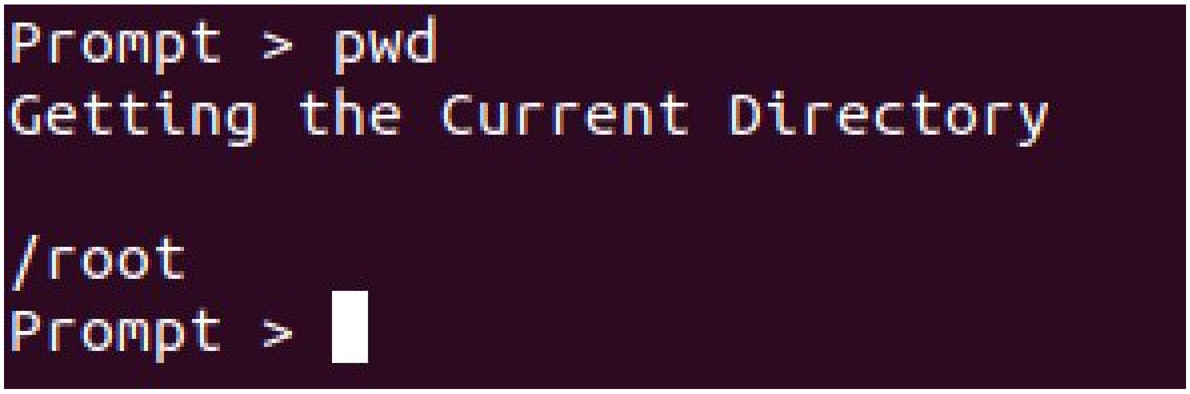
```
student@student-VirtualBox:~/Desktop/GitHub-Fall-2020/group-term-assignment-file-system-harmless4you$ make clean
rm *.o fsshell volumeFormatter
rm: cannot remove 'fsshell': No such file or directory
Makefile:56: recipe for target 'clean' failed
make: *** [clean] Error 1
student@student-VirtualBox:~/Desktop/GitHub-Fall-2020/group-term-assignment-file-system-harmless4you$ make remove
rm SampleVolume
student@student-VirtualBox:~/Desktop/GitHub-Fall-2020/group-term-assignment-file-system-harmless4you$ make format
make volumeFormatter
make[1]: Entering directory '/home/student/Desktop/GitHub-Fall-2020/group-term-assignment-file-system-harmless4you'
gcc -c -o volumeFormatter.o volumeFormatter.c -w -lm -lreadline -I.
gcc -c -o fsLow.o fsLow.c -w -lm -lreadline -I.
gcc -c -o b_io.o b_io.c -w -lm -lreadline -I.
gcc -c -o volumeControlBlock.o volumeControlBlock.c -w -lm -lreadline -I.
gcc -c -o linkedListFreeSpaceMap.o linkedListFreeSpaceMap.c -w -lm -lreadline -I.
gcc -c -o mfs.o mfs.c -w -lm -lreadline -I.
gcc -o volumeFormatter volumeFormatter.o fsLow.o b_io.o volumeControlBlock.o linkedListFreeSpaceMap.o mfs.o -w -lm -lreadline -I. -l pthread
make[1]: Leaving directory '/home/student/Desktop/GitHub-Fall-2020/group-term-assignment-file-system-harmless4you'
./volumeFormatter 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;

Here is the information regarding the Volume Control Block:
VCB Volume Name: SampleVolume
VCB Volume Size: 9999872
VCB Volume Blocks: 512
VCB Volume Blocks In Volume: 19531
VCB Volume Blocks In Volume (Bytes): 2442
VCB Volume Blocks Needed: 5
VCB Free Space Block Number Index: 0
VCB Root Location Index: 0
Size of myVCB structure: 104

VCB successfully written to the Block!


Allocating Free Space Map to the Volume.........
Size of Directory: 792
Next Free Space Block Location: 3
Root Position: 1
Available Free Space Blocks: 2439
Allocation is successful!
```



```
student@student-VirtualBox:~/Desktop/GitHub-Fall-2020/group-term-assignment-file-system-harmless4you$ make run
make fsshell
make[1]: Entering directory '/home/student/Desktop/GitHub-Fall-2020/group-term-assignment-file-system-harmless4you'
gcc -c -o fsshell.o fsshell.c -w -lm -lreadline -I.
gcc -o fsshell fsshell.o fsLow.o b_io.o volumeControlBlock.o linkedListFreeSpaceMap.o mfs.o -w -lm -lreadline -I. -l pthread
make[1]: Leaving directory '/home/student/Desktop/GitHub-Fall-2020/group-term-assignment-file-system-harmless4you'
./fsshell SampleVolume
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opening the Volume.

Initializing the Free Space
Finished Initializing the Free Space
Calling this function: fs_setcwd
Prompt > ls
Getting the Current Directory

Name: /root

Prompt > md hello
Calling fs_mkdir
Pathname: hello
MY FS info: root
Free Space Starting Block: 3
Max Loops: 0
Prompt > ls
Getting the Current Directory

Name: /root

hello
Prompt > █
```



```
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 > history
ls
md hello
ls
help
history
Prompt >
```



```
Prompt > pwd
Getting the Current Directory

/root
Prompt >
```

Issues we Had

This was not an easy project to start and it took quite a time for us to get the bigger picture of approaching the project as well as designing the project structure. At the beginning of the project, we had a different approach with our implementation of Free Space Management. We noticed that most people implemented the BitMap to check whether a block is active or not. Although the implementation of the Bit Map since practically straight forward given that it only works with 1s or 0s when allocating a specific map for directories and files, however just trying to completely understand the concept really took quite a considerable amount of time thus making our progress of being to implement a BitMap in our free space to be quite redundant. Eventually, we were able to go to linked lists since in a way, most of us learned linked lists pretty much during our CSC340 and our CSC220 classes. The concepts are relatively similar and the implementations of it can also be correlated to the way we wanted to implement our free space map. In addition to also having issues with implementing BitMaps in our linked list, there were also some issues regarding figuring out the starting point of the project. At first, we were even having trouble with understanding the LBAread and LBAwrite functions that were implemented in the fsLow.c function because of how it is relatively used everywhere in this file system project, in addition to it being used in the fsLowDriver demo. After having clarifications by discussing with the professor, we then had a rough idea of how it is and we were able to figure it out eventually. With all the parts of the project like trying to understand exactly what the Volume Control Block or the VCB as well as what exactly is the purpose of the Linked List for our Free Space Management, those were the only technical and conceptual issues that we faced when doing the project.

Conclusion and Overall Thoughts of the Project

To sum up , as a group we learned a lot and failed a lot many times. One of the big lessons we learned was time management skill; it is very important when working on a project which we are unfamiliar with. In fact this project showed us as a whole how complicated it must be to create a terminal , even though we didn't achieve the perfect terminal we intended we are still proud of the work as a group of 4 achieved overall . In fact, I would say our terminal did more than 50 percent of its capabilities with some minor bugs . In the short time, we had to create this terminal as a group , we believe we truly achieved a lot. In addition, there were also some conflicts within our classes as well as we had projects as well as other assignments that conflicted with the project overall, as well as personal issues such as time management, due to some of us having either family and personal obligations. But in the end, we learned a lot from the project and we will ensure that the next time we do a project we will be more prepared and ready.