Files System Project

CSC 415-01 Fall 2020

Team Segmentation Fault

Russell Azucenas - Henry Kang - Rinay Kumar

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

Description

This is a file system which manages the files and directories in the storage disk. To manage free space, the system uses a bitmap that stores a binary value to indicate whether the block is used or free. The bitmap is set to 1 if the block has content written in it, otherwise, it is set to 0. The directory entries are stored as inodes, which are pointed by the indices of an array of inodes. Each inode has a name, path, type, parent path, children paths, number of children, last access time, last modification time, its size in blocks and bytes, array of pointers to data blocks, and number of pointers. The file system supports I/O operations including open, read, write, seek, and close. The main driver of the system, the shell, supports ls, copy, move, make directory, remove directory, copy to linux, copy to file system, change directory (cd), print current working directory (pwd), history, and help commands that enables the user to work around the file system.

Issues

Team Member Changes:

The original Segmentation Fault team consisted of four members. One member requested to be transferred to another team, as he felt the pace and progress of the team's work was not as he had expected. Another team member dropped the class, leaving two members, Russell and Henry. Rinay was originally on The Walking Dead team, but due to team members missing meetings and breakout sessions, the team was split up and Rinay was added to the Segmentation Fault team. The final team arrangement for Segmentation Fault was Russell, Henry, and Rinay. The task then was to take parts that were already implemented, some of which were done by team members who left or were moved, and build on them to progress on the project.

Task Communication:

Our team consists of a merger of two other teams. Russell and Henry came from the original Segmentation Fault team, and Rinay was added from the The Walking Dead team. Both prior teams had some communication issues, which led to the change in teams in the first place.

For the original Segmentation Fault team, it was not always clearly communicated what was to be worked on and by whom, and by what time frame to complete it. There were team meetings, but they were undirected and sometimes confusing.

For The Walking Dead team, in the beginning there was steady communication and semi-frequent team meetings covering issues of research, planning, and organization, but as the weeks went on, the team meetings and participation became less frequent. Rinay missed a couple of meetings and in class breakout sessions, which led to the breakup of The Walking Dead team.

The new merged Segmentation Fault team was better at communication. There were regular team meetings attended by all three members, and summaries of the meetings and the tasks to be tackled by each member were posted in the team's Discord chat. When there was a coding issue that needed help, it was posted on the Discord, and help or clarification was provided in a timely manner by the other team members.

Integrating With Fsshell:

After the initial implementation of the directory functions in mfs.c, the next task was to integrate with fsshell to test each of the shell commands. The first issue was in modifying the Make file to accommodate fshell and the various object files needed. Although appearing straightforward after the fact, at the time it took several attempts to get it right. After the Make file was implemented correctly, running 'make' in the terminal led to a series of errors and warnings throughout the project and failed to compile. Each of the red errors were investigated by the team to try and resolve them and get to the Prompt> screen. Several of the issues were related #include statements of the various header files. At some spots the header files were chained, and this was changed to include each the necessary headers in each of the .c files. After a few hours of modifying the files to remove the errors, 'make run' finally completed and ran the Prompt >.

Volume Control Block and Accessing Inodes:

The file system utilized an inode structure. After integrating the mfs.c file with all directory functions to the fsshell and testing some of the commands, we ran into several errors and issues that appeared to be related to the getInode function implemented in fsInode.c. It was not able to access inodes that were created from the volume control block.

Tests were run on the volume control block to ensure it was generating the volume needed by the file system. This involved modifying the Make file to include a 'make volume', which would format and create the volume, and a 'make test', which would test that the sample volume was created and had read/write access. These two make

commands were to be run before calling ‘make run’ and accessing the fsshell / filesystem.

After ensuring that the volume control block was working as intended, the next step was to determine if the LBA read and write functions in fsLow were working correctly. It was found the LBAread method was supposed to return the number of blocks read, but was returning 0 in every logic block. This was modified to return the value returned by the Linux read call in the form of ‘return retRead / partInfo->blocksize’. This change allowed the test of the volume control block that was implemented to pass.

File Paths:

Several of the issues around the functionality of the directory functions in mfs.c seemed to be caused by the pathname that was passed as a parameter to them. Passing just the pathname given to directory functions was not working. Some helper functions were then created to parse the given pathnames and extract the exact information that was needed to pass into the directory functions. The path for the parent directory was needed to use some inode functions and the path for child and target directories were needed for other inode functions.

Driver Information

For the initial testing of starting a partition, checking read/write functionality, then closing the partition, fsLowDriver.c was used as the driver. This file contained the main function that ran the test. After the file system and directory functions were implemented, the Make file was modified to include fsshell.c as the driver, with the main function in there acting as the complete system’s main.

The formatting of the volume control block, however, is not done in fsshell’s main function. Instead, a separate file called fsVolume.c was created with its own main function, and the Make file was modified to include a ‘make volume’ to initialize the volume. In order to run the file system from terminal, the necessary steps are:

1. Change directory into the folder of the project
2. Type ‘make volume’ to create and format the volume
3. Type ‘make run’ to run the fsshell and get ‘Prompt> ‘

Shell Commands

Shell commands were split into two groups, Group 1 consisting of the commands directly related to the directory functions in mfs.c, including pwd, ls, md, cd, rm, and mv, and Group 2 consisting of the commands directly related to b_io functions, including cp, cp2l, and cp2fs.

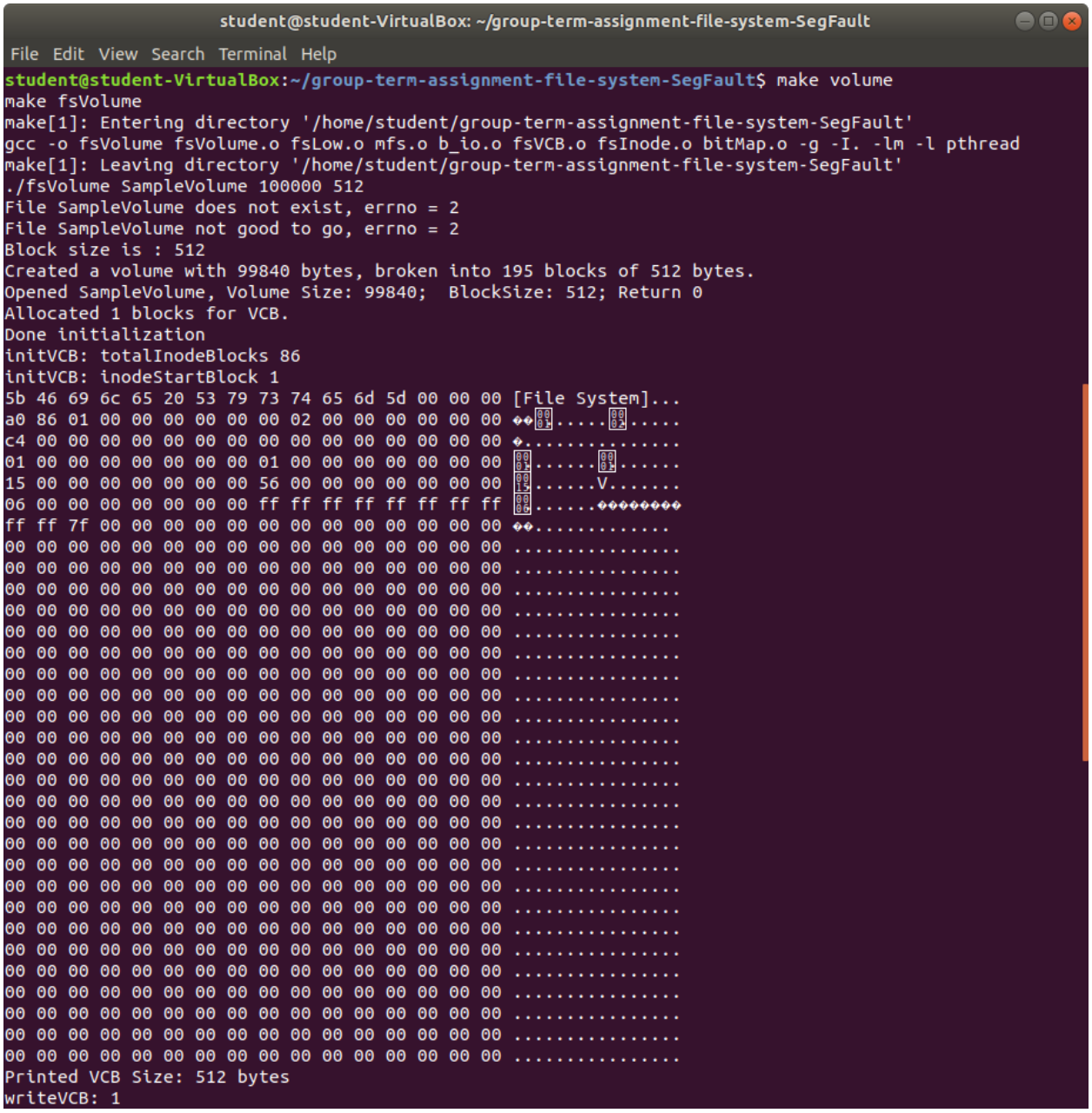
Group 1 Testing:

- **pwd**: Functions as expected, and prints to terminal the current working directory.
- **ls**: Functions as expected. Prints out all the children of the given directory. Flags were not tested however.
- **md**: Functions as expected. Creates new directory of the given name.
- **cd**: Functions as expected. Only changes current working directory if the requested directory exists. Dot commands ‘cd ..’ and ‘cd .’ also function as expected.
- **rm**: Functions as expected. Removes the specified file/directory. Also removes the contents of the directory being removed. This differs from Linux ‘rmdir’, which requires the requested directory first to be emptied before removing.
- **mv**: Functions sporadically. Causes segmentation faults. Not fully debugged.

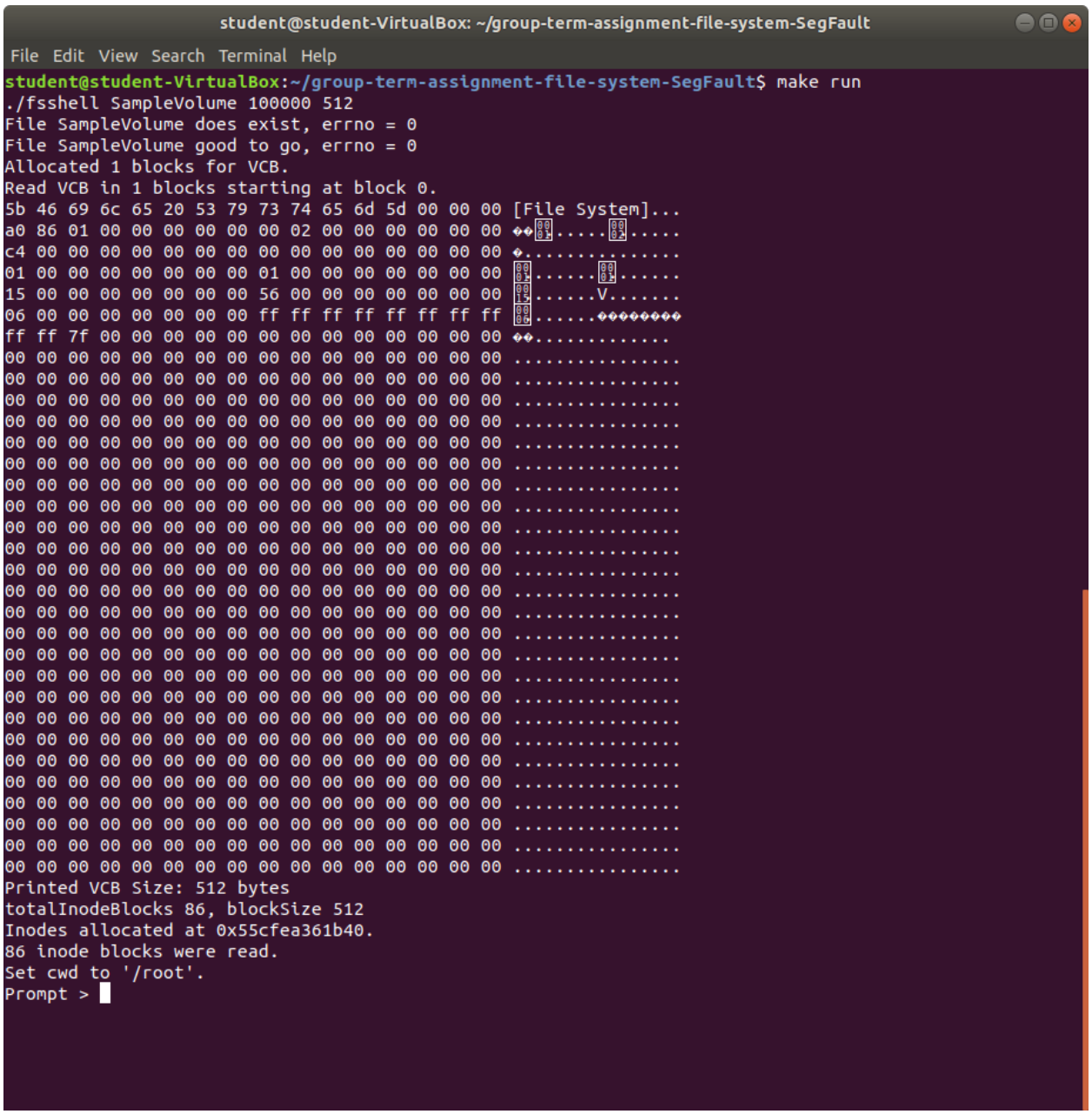
Group 2 Testing:

- **cp**: Partial functionality. Creates destination file, however data doesn’t appear to be copied over correctly.
- **cp2l**: Partial functionality. Takes text file from file system, creates and writes to text file in Linux, however adds excess data after writing complete file.
- **cp2fs**: Functions as expected. Takes a text file from Linux and adds it to the file system.

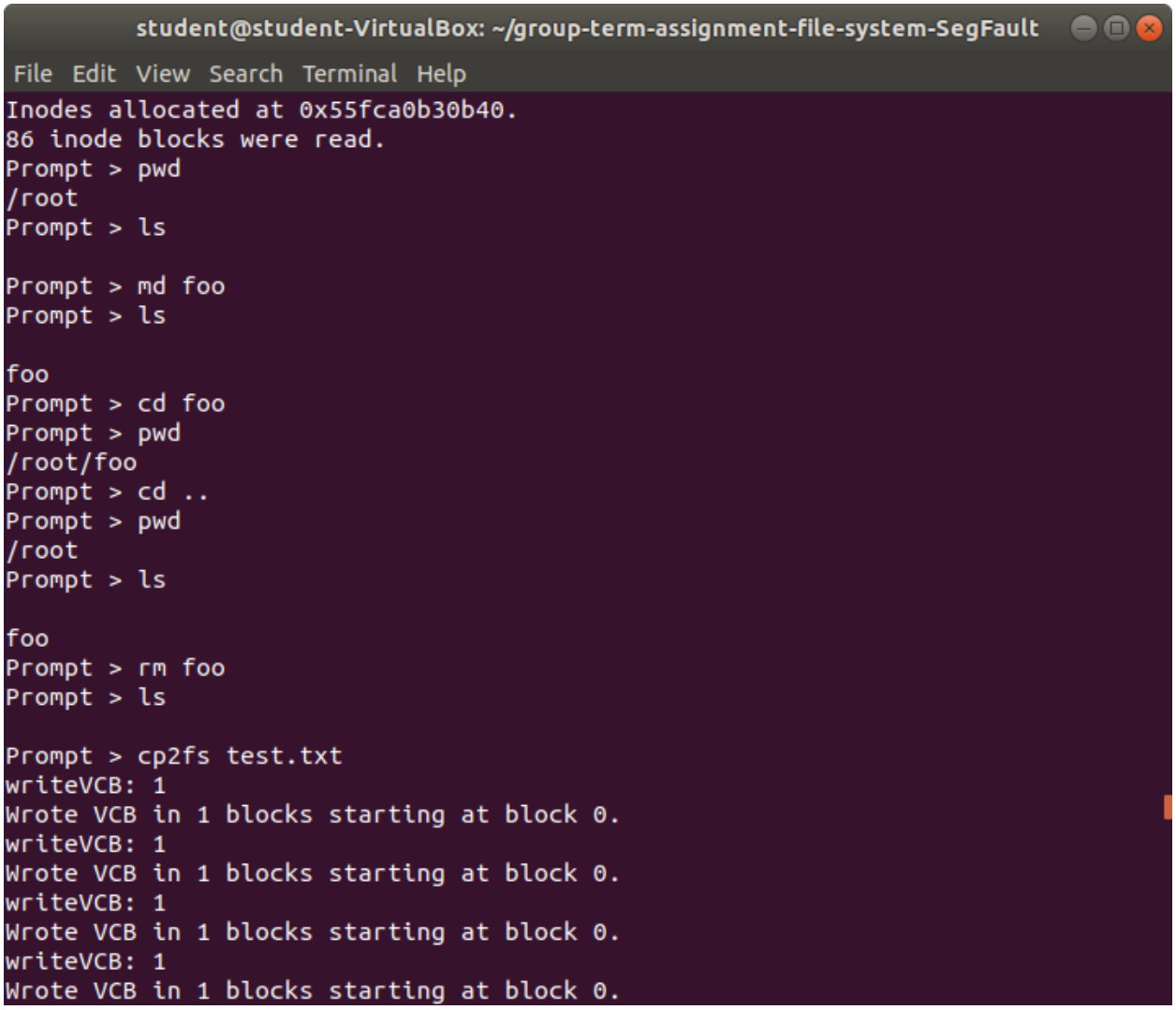
Screenshots



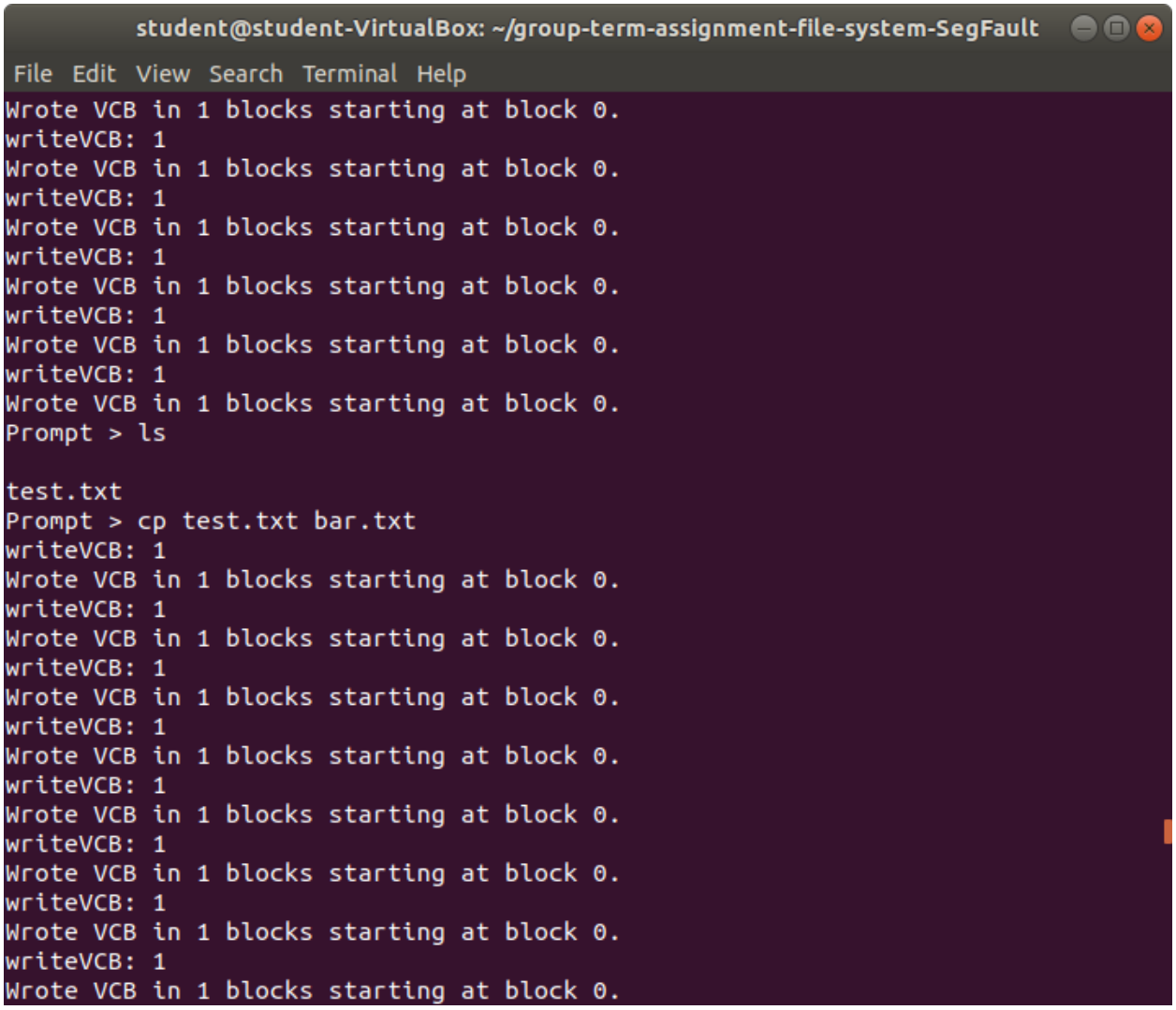
Shows the ‘make volume’ command, which creates the volume.



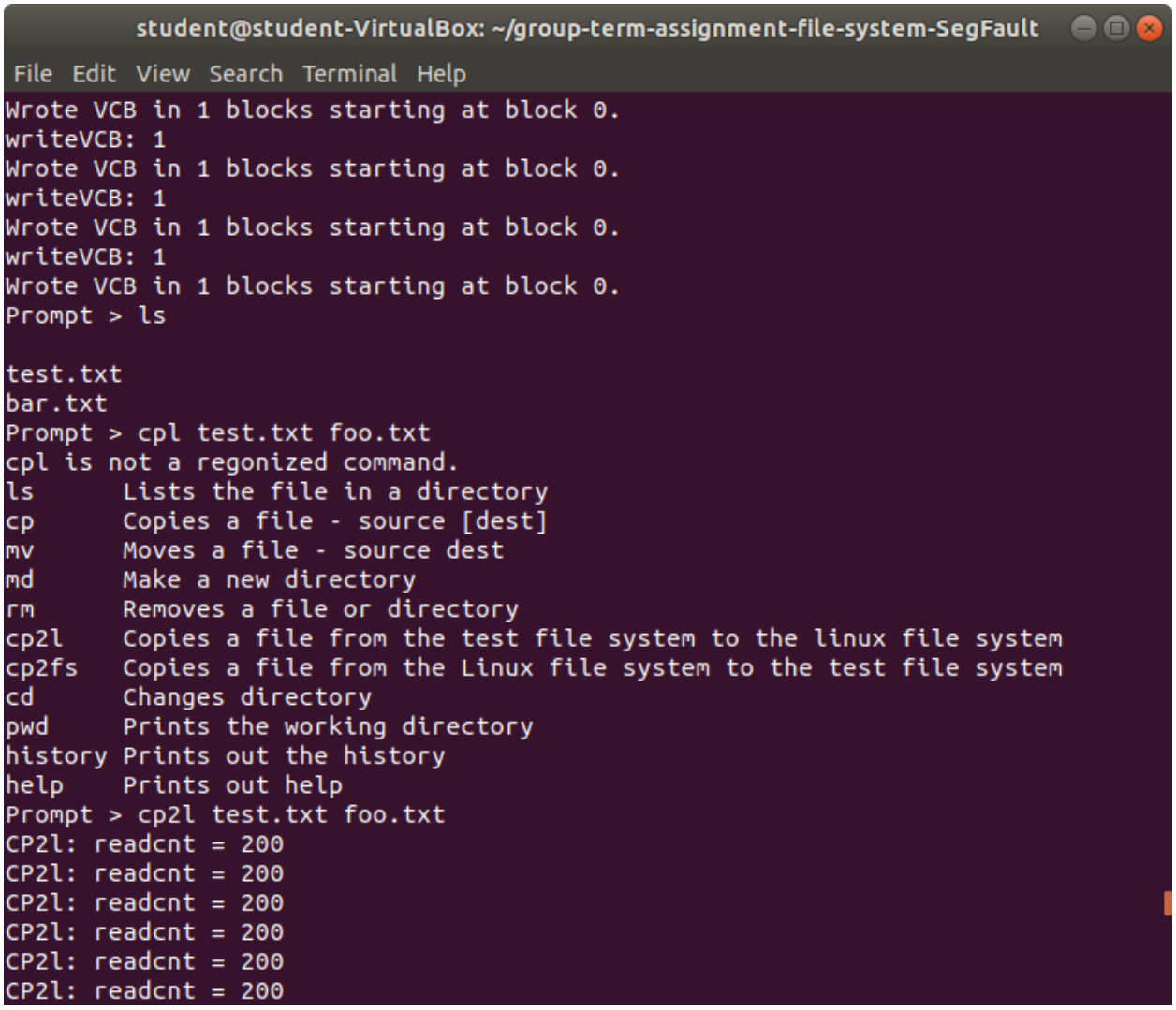
Shows ‘make run’ which initializes and starts the shell.



Shows pwd, ls, md, rm, and cp2fs with test.txt being added to file system



Shows that test.txt is added using ls, shows cp functionality creating bar.txt



Shows cp created text file, shows help command, shows cp2l writing file to Linux