File System

Project Assignment

Group: Segmentation Fault

Member	Github Usernames	Student ID
Christian Francisco	chrisf725	920603057
David Ye Luo	DavidYeLuo	917051959
Marc Castro	mcastro16	921720147
Rafael Sunico	Gravemind1142	920261261

Table Of Contents

Implementation

File Control Block Directory Entry (struct b_fcb)

File: b_io.h

Fields	Purpose
char *buf	Buffer that is in multiple blocks. Since we work with blocks instead of bytes to write to disk, the buffer prepares the data for the process.
int index	Index holds the current position of the char *buffer.
int buflen	Buflen indicates how many valid bytes are in the buffer.
dir_entry *dirEntry	DirEntry is a * pointer that is passed to b_fcb through the function b_open().
int flags	Flags variable is a bit wise value that contains Read/Write permission flags.
uint64_t currentBlockNum	CurrentBlockNum is a uint64_t variable that contains the current block number in the FCB.
bool isDirty	isDirty is a boolean variable that is true or false depending on whether the buffer is committed or not. Runs “true” if uncommitted.

File: b_io.c

The C file is the basic file system that handles key I/O Operations, the header file contains the definitions for the functions within the C file.

Fields	Purpose
b_io_fd b_open(char *filename, int flags)	A function that is an interface to open a buffered file. This function only performs modification of interface for this assignment and the flags match the Linux flags found in the open function.
int b_seek(b_io_fd fd, off_t offset, int whence)	Function that returns the offset location in bytes from the start of the file.
int b_write(b_io_fd fd, char *buffer, int count)	Write data in bytes of the buffer to the outer buffer of the fcb and write the bytes in disk.
int b_read(b_io_fd fd, char *buffer, int count)	This function is used to read a buffer that is passed into it and returns the blocks read.
void b_close(b_io_fd fd)	The function closes the file by freeing memory of the file descriptor and sets the buffer pointer to NULL. This also enables the file descriptor to handle any upcoming I/O operations.
uint64_t getBlockXofFile(b_fcb *fcb, uint64_t relativeFileBlockNum, uint64_t block_count)	A helper function for b_read() that is encapsulating the traverse of the FAT process. It is essentially the same as readAllocated but with relative file location.
uint64_t putBlockXofFile(b_fcb *fcb, uint64_t relativeFileBlockNum, uint64_t block_count)	A helper function for b_write() that is encapsulating the traverse of the fat process. It is essentially the same as overwriteAllocated but with relative file location.

Directory Entry

File: dirEntry.h

Fields	Purpose
name[NAME_SIZE]	Stores the name of the directory entry.
isDir	A boolean variable that runs true if it is a valid directory.
location	Stores the location of the directory entry.
size	Stores the size of the directory entry.
reclen	Number of used bytes of the directory

File: dirEntry.c

The C file contains basic directory management functions, and the header file contains basic directory information.

Fields	Purpose
uint64_t createDir(uint64_t parentLocation)	This function is used to create a directory by establishing how many blocks are needed, creating the directory pointer, setting a known state, allocating free space, and returning the start block to where the function was called.
bool pathParser(const char *path, int last_token_flag, dir_entry *temp_dir, char *last_token_in_path)	A function to check if a path is valid as either a file or a directory.
int fs_mkdir(const char *pathname, mode_t mode)	Function called upon when the mkdir Linux command is called in the command line.

	redirects to createDir to create the directory at a given location. If created successfully, the function will return 0.
int fs_delete(char *filename)	This removes the associated directory entry and frees the allocated blocks in the volume.
int fs_rmdir(const char *pathname)	Function that is called to remove a directory by checking if the path is allowed to be deleted or if it is valid. If those two conditions apply, it will then delete the path from the volume.
fdDir *fs_opendir(const char *name)	This opens a directory by giving a directory name and returns a pointer to the structure that stores the info of the directory given that the directory exists.
struct fs_diriteminfo *fs_readdir(fdDir *dirp)	Function that reads the directory descriptor and returns the directory information.
int fs_stat(const char *path, struct fs_stat *buf)	General details about a file system. Metadata like number of blocks, block size, size, access time, time modified, create time are provided here.
int fs_closedir(fdDir *dirp)	Closes the directory by freeing the directory pointer.
char *fs_getcwd(char *buf, size_t size)	Returns the path of the current working directory.
int fs_setcwd(char *buf)	Used to set the current working directory upon change of directory.
int fs_isFile(char *path)	Check the path if it is a file or not.
int fs_isDir(char *path)	Checks the path if it is a directory or not.

Free Space

File: fsFree.c

The C file contains functions needed for free space management, and the header file contains the definitions for free space management.

Fields	Purpose
uint64_t initFreeSpace(vcb *my_vcb)	Function that initiates the free space
int reloadFreeSpace(vcb *my_vcb)	Function that uses malloc and LBAread to retrieve the FAT array from the volume, assuming the volume is already formatted.
uint64_t allocateFreeSpace(uint64_t block_count)	Function that is used to allocate more free space for saving in the volume when called and returns the new start location.
uint64_t overwriteAllocated(void *buffer, uint64_t block_count, uint64_t block_location)	This behaves the same as LBAwrite, but it splits LBAwrite calls for fragmented FAT entries.
uint64_t readAllocated(void *buffer, uint64_t block_count, uint64_t block_location)	This behaves the same as LBAread, but splits the LBAread calls for fragmented FAT entries.
int releaseFreeSpace(uint64_t block_location)	Traverses FAT from block_location and marks subsequent blocks as free.

File System

File: fsInit.c

The file that contains the file system management.

int initFileSystem (uint64_t numberOfBlocks, uint64_t blockSize)	Initializes the file system with an amount of blocks along with a set amount of block size.
void exitFileSystem()	Not Implemented
void initVCB()	Not Implemented

File: mfs.h

This is the header file that contains the file system interface which is needed by the driver to interact with the filesystem.

struct fs_diriteminfo

Fields	Purpose
unsigned short d_reclen	Used to store the length of the record.
unsigned char fileType	Used to store the file type
char d_name[256]	Used to store the file name

struct fdDir

Fields	Purpose
char name[NAME_SIZE]	Stores the name of the fdDir
unsigned char fileType	Stores the file type.
unsigned short d_reclen	Stores the length of the structure.
unsigned short dirEntryPosition	Stores the file position of the directory
uint64_t directoryStartLocation	Stores the LBA starting location of the directory.

struct fs_stat

Fields	Purpose
off_t st_size	Stores the total size of fs_stat structure in bytes.
blksize_t st_blksize	Stores the block size for the file system I/O.
blkcnt_t st_blocks	Stores the number of 512 Byte blocks allocated.
time_t st_accesstime	Stores the time the structure was last accessed.
time_t st_modtime	Stores the time the structure was last modified.
time_t st_createtime	Stores the last status change.

Volume Control Block

File: vcb.h

The header file contains the volume control block.

Fields	Purpose
char disk_sig[4]	Disk signature that is defined as 4 chars.
uint64_t block_count	Number of blocks in the volume.
uint64_t block_size	Size in bytes per block.
uint64_t vcb_location	Stores the vcb location in volume.
uint64_t fs_start_location	Location where free space starts in volume.
uint64_t fs_next_location	Location of the next free block.
uint64_t root_location	Location where the root directory starts in volume.
uint64_t root_size	The size of the root in volume in blocks.
uint64_t fat_location	Location where FAT is stored in volume.
uint64_t fat_size;	The size of FAT in volume in blocks.

Issues and Resolutions

Unknown memory leak.

We currently have a problem that didn't cause issues with our program until the final few days of submission. There is a memory leak somewhere in our code, dating back to git commits since milestone 2, but only caused noticeable issues after implementing milestone 3 functions. It currently causes portions of the root directory to unexpectedly be overwritten by random bytes, which causes the block location/size values to be lost, which prevents consecutive volume read/writes to the root's descendant files and directories.

The make directory command didn't work with any other path that included "..".

This problem occurred because the pathParser function was only iterating over directory entries from index 2 onwards, which means that it would ignore the "." and ".." directory entries. I fixed this by changing the pathParser function to start iterating over directory entries from index 0.

This problem was also related to another issue where the ls -a command would not list "." or "..", which occurred because the fs_stat function was only iterating over directory entries from index 2 onwards, which means that it would ignore the "." and ".." directory entries. I fixed this by changing the pathParser function to start iterating over directory entries from index 0.

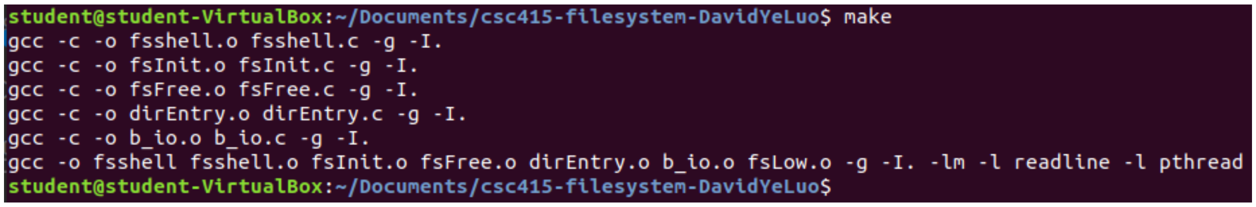
initFreeSpace was not initializing FAT array blocks correctly.

We worked on this together and realized we were initializing the first block to the correct limiter. After further review of how the fat array works, we established we have to go to the very end by adding 1 to the sffs_vcb->fat_size to hit the end. We then applied this to setting the two blocks after the FAT to be reserved.

Detail of how your driver program works

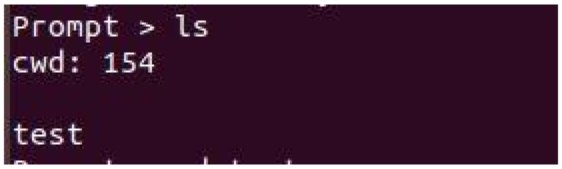
Commands	Working?	Description
LS	🟩	List directory
CP	🟥	Copy file - source destination
MV	🟥	Move file - source destination
MD	🟥	Make new directory
RM	🟥	Removes a file or directory
HISTORY	🟩	History of command calls.
CP2L	🟥	Copies a file from the test file system to the linux file system.
CP2FS	🟥	Copies a file from the Linux file system to the test file system.
CD	🟩	Changes Directory
PWD	🟩	Prints Working Directory
HELP	🟩	Displays brief summary of commands available

Screenshot of compilation:

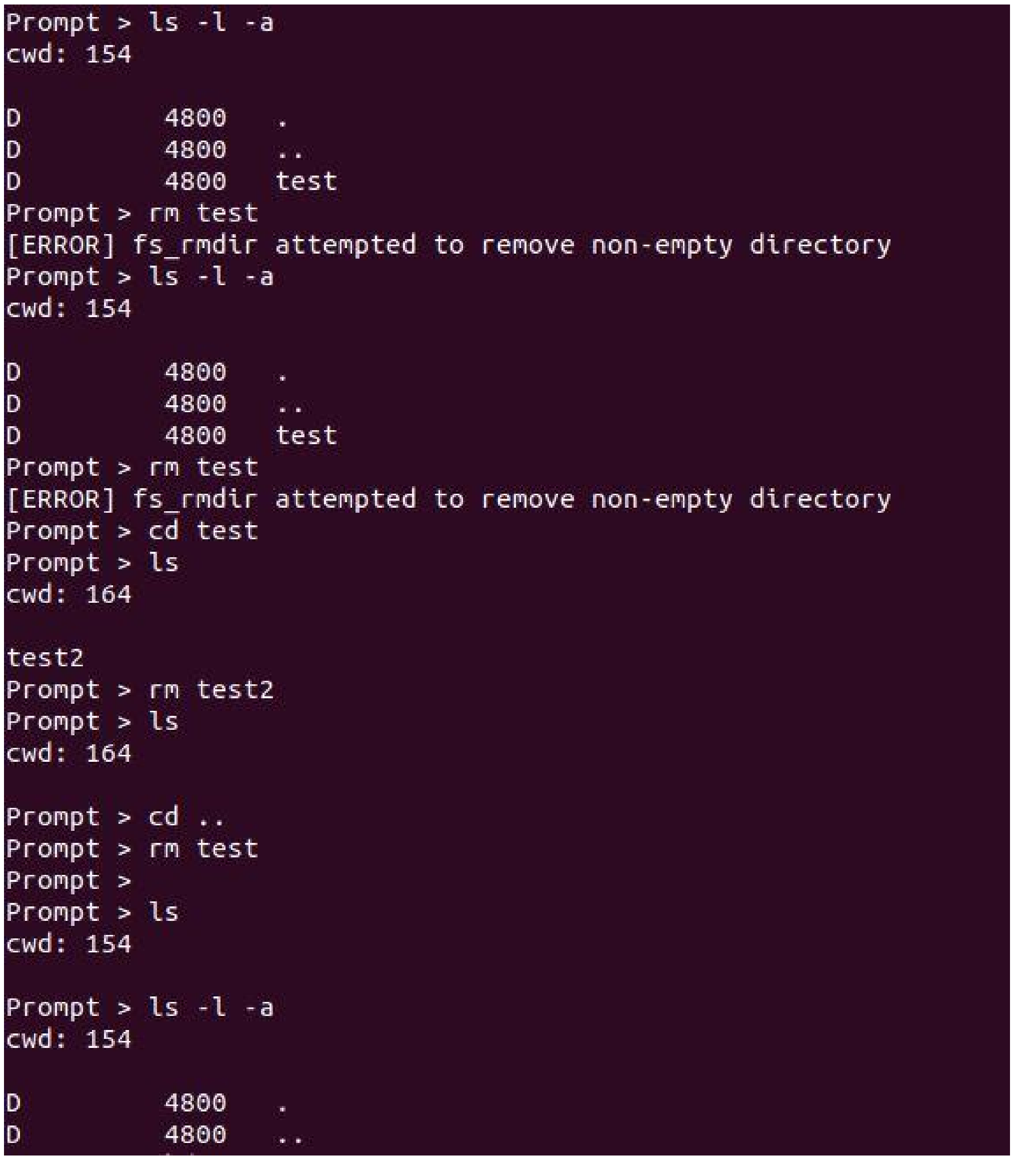


```
student@student-VirtualBox:~/Documents/csc415-filesystem-DavidYeLuo$ make
gcc -c -o fsshell.o fsshell.c -g -I.
gcc -c -o fsInit.o fsInit.c -g -I.
gcc -c -o fsFree.o fsFree.c -g -I.
gcc -c -o dirEntry.o dirEntry.c -g -I.
gcc -c -o b_io.o b_io.c -g -I.
gcc -o fsshell fsshell.o fsInit.o fsFree.o dirEntry.o b_io.o fsLow.o -g -I. -lm -l readline -l pthread
student@student-VirtualBox:~/Documents/csc415-filesystem-DavidYeLuo$
```

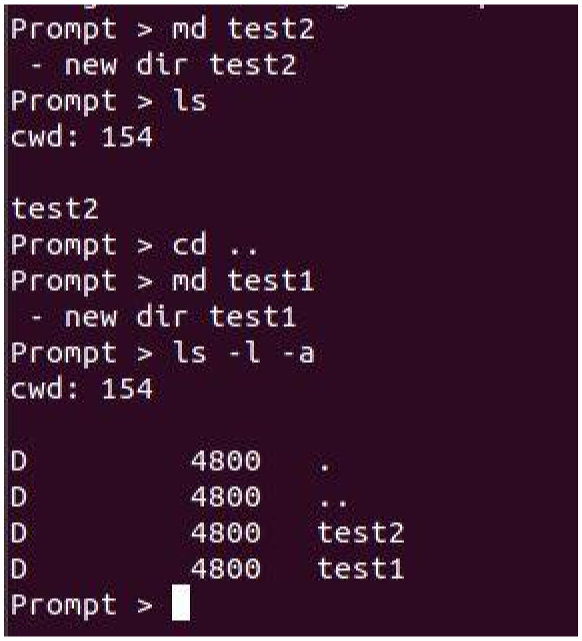
Screenshot(s) of commands:



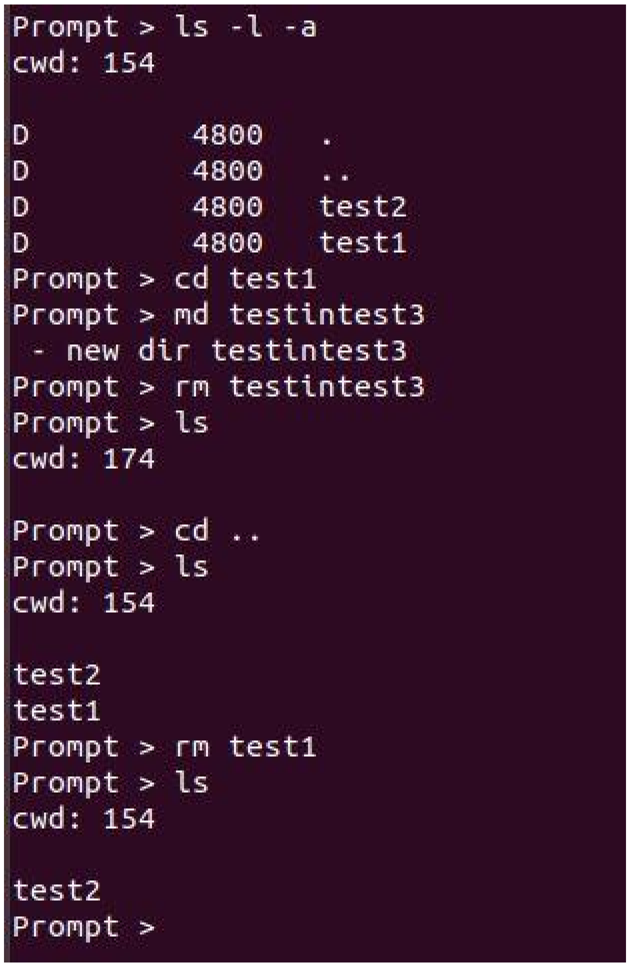
```
Prompt > ls
cwd: 154

test
```



```
Prompt > ls -l -a
cwd: 154

D         4800   .
D         4800   ..
D         4800   test
Prompt > rm test
[ERROR] fs_rmdir attempted to remove non-empty directory
Prompt > ls -l -a
cwd: 154

D         4800   .
D         4800   ..
D         4800   test
Prompt > rm test
[ERROR] fs_rmdir attempted to remove non-empty directory
Prompt > cd test
Prompt > ls
cwd: 164

test2
Prompt > rm test2
Prompt > ls
cwd: 164

Prompt > cd ..
Prompt > rm test
Prompt >
Prompt > ls
cwd: 154

Prompt > ls -l -a
cwd: 154

D         4800   .
D         4800   ..
```



```
Prompt > md test2
 - new dir test2
Prompt > ls
cwd: 154

test2
Prompt > cd ..
Prompt > md test1
 - new dir test1
Prompt > ls -l -a
cwd: 154

D         4800   .
D         4800   ..
D         4800   test2
D         4800   test1
Prompt >
```



```
Prompt > ls -l -a
cwd: 154

D         4800   .
D         4800   ..
D         4800   test2
D         4800   test1
Prompt > cd test1
Prompt > md testintest3
 - new dir testintest3
Prompt > rm testintest3
Prompt > ls
cwd: 174

Prompt > cd ..
Prompt > ls
cwd: 154

test2
test1
Prompt > rm test1
Prompt > ls
cwd: 154

test2
Prompt >
```