File System Final Group Submission

Names: Danial Tahir, Chris Camano, Mahek Delawala, Savjot Dhillon

Student IDs: 920838929, 921642160, 922968394, 918239054

GitHub Name: DanTahir

Team Name: Segfault

https://github.com/CSC415-2023-Spring/csc415-filesystem-DanTahir

Table of Contents

§ 0. Shell command functionality table

Shell Command	Working as Intended?
LS	**Yes**
CP	**Yes**
MV	**Yes**
MD	**Yes**
RM	**Yes**
CP2L	**Yes**
CP2FS	**Yes**
CD	**Yes**
PWD	**Yes**
TOUCH	**Yes**
CAT	**Yes**

§ 1. Description of the file system

Our file system has a one-block vcb, a freespace map based on a bitmap, and a directory structure that allows unlimited entries in each directory (subject to filespace).

1.1 Description of the VCB Structure

The VCB is a struct of uint64_t variables which fit into a single 512 byte block. The first is the signature, nominally set to 29339303911 (although it could be changed slightly if the volume needs to be reset). Second is the location of the start of the freespace map. Third is the location of the start of the root directory. Fourth is the block size. Fifth is the volume size. Sixth is the block count, the total number of blocks.

1.2 Description of the Free Space Map Structure

The free space map is a bitmap, an array of bytes (typedef u_int8_t) with a size of (rounded up) numberOfBlocks / 8 bytes which fits into that value rounded-up divided by blockSize number of blocks. It always starts in block 1 immediately after the VCB. It allocates one bit for each block in the volume. The initial bits set are the bits for the VCB, the free space map and the root directory. All other bits in the map are set to 0. Free space is allocated contiguously.

Space for files and folders is allocated and freed using the bitmapAllocFilespace and bitmapFreeFilespace functions. Space is allocated based on the results of the bitmapFirstFreeFilespace function, which takes a size in bytes and returns the next contiguous open space of that size, which is then allocated by bitmapAllocFilespace.

1.3 Description of the Directory Structure

A directory in our system is an infinitely expanding array of DirEntry structs. The DirEntry struct is made up of a char array of size 25 for the name(nominally set to NAMELEN which is defined in the code as 25), a uint64_t for the location on disk of the entry, a uint64_t for the size in bytes of the entry, and a byte (a self-defined data type that represents a u_int8_t) for whether or not the file is a directory. The first two entries are initialized to “.” for the current file and “..” for the parent file. New entries are added by the dirAddEntry function which resizes the array and writes the new entry to disk, moving the directory if necessary to allocate more blocks to the directory. Entries are removed by the dirRemoveEntry function which resizes the array downwards but doesn’t move directories since they’ll

always fit in their existing space. Directories know their size by storing their size in bytes and then calculating their number of entries based on their size in bytes.

For the root directory, the location of the parent is initialized to the same location as the root directory. The root directory is always allocated immediately after the freespace map, but may change position later, in which case the VCB is updated.

1.4 Description of the directory functions in mfs.h and mfs.c

fs_mkdir() takes a pathname, instantiates a directory, then copies the working directory to that directory. Then it uses dirTraversePath to advance that directory to the right position on the path. It then checks the last node in the path for whether it's present in the directory. If not, it creates the directory using dirInitNew, adding it to the entries in the parent directory using dirAddEntry.

fs_rmdir() takes a pathname, instantiates a directory, then turns it into a copy of the working directory. Then it uses dirTraversePath to advance that directory to the appropriate position on the path, returning the last node in the path to work with. The function checks if the last node is found in the names in the directory, checks whether the entry is a directory, and finally, checks whether the directory is empty. If all of those conditions are true, the directory is removed from its parent using dirRemoveEntry and its space freed from the freespace map using bitmapFreeFilespace.

fs_opendir() takes a pathname, instantiates a directory, copies the working directory to the directory, then uses the directory to traverse the path. It outputs the last pathnode as a string and searches the directory for an entry with the string as its name. It checks whether the entry is a directory, and if it is, the function creates an fdDir object that it returns. If the entry isn't found or isn't a directory, or the path is invalid, the function returns null.

fs_readdir takes the fdDir pointer created by opendir and uses dirRead to open the folder location pointed to by the fdDir's directoryStartLocation. It checks whether the fdDir pointer (dirp)'s entry position is equal to or greater than the count of entries in the opened directory, and if so, it returns null. Otherwise, it reads the entry item pointed to by the entry position into an fs_diriteminfo struct which it returns.

fs_closedir frees the fdDir pointer created by fs_opendir, returning -1 if the pointer is already null.

fs_getcwd instances a dir and copies the working directory. It then stores its current location before iterating back to its parent. It checks the parent for the matching location and uses that name to write to the path, after first copying the path so it can rewrite the stored path after the directory name. It then does this again and again until it reaches root. If the directory starts out at root, it simply outputs a / as the pathname.

fs_setcwd takes a pathname, then traverses the working directory along the path. dirTraversePath outputs the last node in the path as a string, then setcwd searches the working directory for the final node as a filename. If the name is found and its entry is a directory, dirSetWorking sets the working directory to the entry's location. If the entry isn't found or isn't a directory, the function returns -1.

fs_isDir and fs_isFile work the same way, they take a filename (actually a path) and instance an fs_stat struct and call fs_stat on it, then return the fileType variable we added to the fs_stat struct, except fs_isFile returns the opposite of the variable.

fs_delete instances a directory, copies the working directory, then traverses the passed-in path. It uses the last node in the path to search the directory for a filename, checks to see whether the result is a file or a directory, and if it is a file, it calls dirRemoveEntry on the directory at the entry's position to erase the file and resize the directory.

fs_stat copies the working dir, traverses the path, then searches the directory for a filename, and if it finds the filename, it fills out the passed-in fs_stat object and returns 0.

1.5 Description of the file functions in b_io.h and b_io.c

b_open opens a file control block, then creates a directory and copies the working directory. dirTraversePath traverses the "filename" (actually a path) to the end directory and outputs the last node in the path as a filename. b_open searches the directory for the filename. If the filename is not found, b_open checks whether the create flag is set. If it is, b_open creates a new empty file using dirAddEntry, then fills out the FCB with the directory, directory position, and other important data, and also sets the buffer to the size of one block so that b_getFCB() will recognize it, then returns the file descriptor. If the filename is not found and the create flag is not set, b_open outputs "file not found" and returns -1. If the filename is found but is a directory, b_open returns -1. If the truncate flag is set, b_open zeroes out the file. Then b_open allocates a buffer of filesize (minimum size 1) and reads the file from volume into the buffer. By storing the entire file in the buffer we greatly simplify b_write and b_read while reducing the necessary number of reads and writes. b_open sets the remaining information in the FCB and returns the file descriptor.

b_seek takes a file descriptor and sets the related file control block's index based on the whence int. If whence is SEEK_SET it sets the index to the beginning of the file, SEEK_CUR is the current position, and SEEK_END sets the index to the end of the file. The file is then incremented by the offset. The function returns the position the index has been incremented to.

b_write first checks if the write or read/write flags are set, and if not, returns -1. If the appropriate flags are set, b_write checks the append flag and, if its set, sets the index to the end of the file. dirRead rereads the directory pointed to by the FCB's dir, in case it's changed since the file was opened.

b_write then checks whether the index plus the count overrun the filesize, in which case b_write needs to write new filespace. If so, bitmapFreeFilespace releases the old filespace, bitmapFirstFreeFilespace finds filespace for the new filesize, and bitmapAllocFileSpace allocates the new filespace. Then the FCB buffer is freed and resized to the new filesize. The old file is read into the resized buffer, then the buffer is overwritten with the passed-in user buffer from the index to the count, filling the resized buffer. Then the file is written to its new location. b_write then returns the bytes written. If the index + count do not overrun the filesize, then there is no need to resize the buffer, so b_write simply memcopies the user buffer over the FCB buffer at the appropriate index, then writes the file and returns the bytes written.

b_read checks whether the read or read/write flags are set, and if not, returns -1. If they are set, b_read makes sure the requested count doesn't overrun the file, and if it does, truncates the count. Then it uses memcpy to copy from the FCB buffer to the user buffer, and returns the total bytes copied.

b_close frees and zeroes everything in the FCB.

§ 2. Issues we encountered

One issue we encountered was that we didn't know how to write a bitmap. We resolved the issue by using code we found from a source online, https://codereview.stackexchange.com/questions/8213/bitmap-implementation.

Another issue we encountered was that we lost a number of points on Milestone 1 for various reasons, including that our init file instantiated our bitmap and our directory functions took the system blocksize and block count, and also our directories were a Dir struct containing an array of DirEntry structs. We resolved the issue by refactoring so the instantiation of the bitmap is contained within the bitmap file and the directory functions no longer require the blocksize and blockcount, and also so that directories are an array of DirEntry structs.

Another issue we encountered was when we decided to implement expanding-size folders, this was much more complex than any previous subsystem we'd added and introduced a number of bugs. We resolved the issue with lots of printfs and laborious bug squashing.

Debugging the b_io files proved to be quite difficult since some of the functions were quite sensitive and so many shell commands depended on them working. Of all of the testing that we did these functions required some of the most care to ensure that the interconnected parts of the codebase were working properly.

For implementing some of the directory related functions, as in case of setcwd() and getcwd() we had discussion on how to start the implementation as different people in the team thought of different implementations and then after having discussion within the team we finalized one approach.

§ 3. Details of the Driver

fsshell.c includes 12 functions. cmd_ls, cmd_cp, cmd_mv, cmd_md, cmd_rm, cmd_touch, cmd_cat, cmd_cp2l, cmd_cp2fs, cmd_cd, cmd_pwd, cmd_history, and cmd_help. It calls these functions based on commands entered into the prompt. Commands entered into the prompt are processed by processprompt, which hands them off to their functions.

3.1 cmd_ls

First cmd_ls checks for whether the initial arguments in the argument vector are arguments for the ls command. Then it reads for a remaining argument in the argument vector. If there is one, it reads in that argument as a pathname and checks if the pathname ends with a directory. If not, it calls getcwd and reads in the pathname from that function. In both cases it then calls opendir on the path before handing off its fdDir pointer to a displayFiles function. displayFiles displays the results of fs_readdir. If the long flag is set, it temporarily sets the working directory to the fdDir's pointed-to directory so that fs_stat can read the filename pointed to by fs_readdir.

3.2 cmd_touch

cmd_touch opens the first argument as a pathname which it passes to b_open with the write and create flags set. b_open works as previously described, resulting in a file of size 0. Then cmd_touch calls b_close to close the file control block as previously described.

3.3 cmd_cat

cmd_cat uses its first argument as a path. It hands the path to b_open which works as previously described and returns a file descriptor that points to a file control block. Then cmd_cat calls b_read to read 200 bytes from the file in a loop until b_read returns less than 200 bytes, at which point the file is read and b_close is called to clear the file control block.

3.4 cmd_cp

cmd_cp reads in two arguments as paths, one as source and one as destination. It opens the source path with b_open set to read and the destination path with b_open set to write, create, and truncate. b_open works as previously described, setting file control blocks for both files with the appropriate flags and other data set, and returning a file descriptor pointing to each file control block. cmd_cp uses b_read to read BUFFERLEN bytes from file 1 and uses b_write to write them to file 2 in a loop until b_read returns less than BUFFERLEN, then it calls b_close to close both files.

3.5 cmd_mv

cmd_mv works exactly like cmd_cp except that if both file descriptors are valid and the source and destination are different, cmd_mv calls fs_delete on the source path to delete the file.

3.6 cmd_md

cmd_md calls fs_mkdir with its argument as the path. fs_mkdir works as previously described, adding a new directory to the parent directory.

3.7 cmd_rm

cmd_rm uses a single argument as a path, which it checks against fs_isFile and fs_isDir, which work as previously described. If fs_isDir returns true, cmd_rm calls fs_rmdir on the path, which works as previously described and deletes the directory if it is empty. If fs_isFile returns true, cmd_rm calls fs_delete on the path to delete the file.

3.8 cmd_cp2l

cmd_cp2l uses two arguments as paths, one as a source path on the volume, one as a destination path on the linux filesystem. It calls b_open on the source path and linux open on the destination path, then reads BUFFERLEN from b_read on the volume file descriptor and writes them with linux write using the linux file descriptor, in a loop until b_read returns less than BUFFERLEN bytes. Then b_close and close are called on the file descriptors.

3.9 cmd_cp2fs

cmd_cp2fs uses two arguments as paths, one as a source path on the linux filesystem, one as a destination path on the volume. It calls b_open on the destination path set to write, create and truncate and calls linux open on the source path set to read only. Then cmd_cp2fs reads from the linux file using the linux read function pointed to the linux file descriptor, then it writes from the read buffer to the volume file using the b_write with the volume file descriptor, which accesses the file pointed to by that descriptor's file control block. It writes BUFFERLEN bytes in a loop until read returns less than BUFFERLEN, then it closes both descriptors.

3.10 cmd_cd

cmd_cd takes a single argument as a path which it sends to fs_setcwd. fs_setcwd works as previously described and sets the working directory to the passed-in path.

3.11 cmd_pwd

cmd_pwd creates a char pointer buffer of DIRMAX_LEN +1 and a char pointer and calls fs_getcwd with the buffer and length of DIRMAX_LEN. fs_getcwd works as previously described, then cmd_pwd prints the returned char pointer from fs_getcwd.

3.12 cmd_history

cmd_history returns command history from history_get and prints it to the screen.

3.13 cmd_help

cmd_help returns the entries from the dispatchTable.

§ 4. Who worked on what

The VCB	**Danial Tahir, Chris Camano**
The free space map	**Danial Tahir, Mahek Delawala**
The root directory	**Danial Tahir, Savjot Dhillon**
fs_mkdir	**Danial Tahir**
fs_rmdir	**Mahek Delawala, Danial Tahir**
fs_opendir	**Savjot Dhillon, Danial Tahir**
fs_readdir	**Savjot Dhillon, Danial Tahir**
fs_closedir	**Savjot Dhillon, Danial Tahir**
fs_isDir	**Savjot Dhillon**
fs_isFile	**Savjot Dhillon**
fs_getcwd	**Mahek Delawala, Danial Tahir**
fs_setcwd	**Mahek Delawala, Danial Tahir**
fs_stat	**Chris Camano, Danial Tahir**
fs_delete	**Chris Camano, Danial Tahir**
b_open	**Danial Tahir, Chris Camano**
b_read	**Danial Tahir, Chris Camano**
b_write	**Danial Tahir, Chris Camano**
b_seek	**Danial Tahir, Chris Camano**
b_close	**Danial Tahir, Chris Camano**
mv	**Danial Tahir**
Unit testing	**Danial Tahir, Chris Camano, Mahek Delawala, Savjot Dhillon**
Final writeup	**Danial Tahir, Chris Camano, Mahek Delawala, Savjot Dhillon**

§ 5. Command Screenshots

5.1 ls and md

```
student@student-VirtualBox:~/assignments/group/csc415-filesystem-DanTahir$ make
run RUNOPTIONS="SampleVolume2 10000000 1024"
./fsshell SampleVolume2 10000000 1024
File SampleVolume2 does not exist, errno = 2
File SampleVolume2 not good to go, errno = 2
Block size is : 1024
Created a volume with 9999360 bytes, broken into 9765 blocks of 1024 bytes.
Opened SampleVolume2, Volume Size: 9999360;  BlockSize: 1024; Return 0
Initializing File System with 9765 blocks with a block size of 1024
signature not a match
isvcbset returns 0
running setVCB
writing bitmap
getting VCB
writing root directory
Prompt > ls
```

```
Prompt > ls

Prompt > md 1
dir to make - 1
Prompt > md 2
dir to make - 2
Prompt > md 3
dir to make - 3
Prompt > md 4
dir to make - 4
Prompt > md 5
dir to make - 5
Prompt > md 6
dir to make - 6
Prompt > md 7
dir to make - 7
Prompt > md 8
dir to make - 8
Prompt > md 9
dir to make - 9
Prompt > md 10
dir to make - 10
Prompt > md 11
dir to make - 11
Prompt > md 12
dir to make - 12
Prompt > md 13
dir to make - 13
Prompt > md 14
dir to make - 14
Prompt > md 15
dir to make - 15
Prompt > md 15
dir to make - 15
dirToMake found in dir
Prompt > md 16
```

```
Prompt > md 16
dir to make - 16
Prompt > md 17
dir to make - 17
Prompt > md 18
dir to make - 18
Prompt > touch 18
directory selected
Prompt > md 19
dir to make - 19
Prompt > md 20
dir to make - 20
Prompt > md 21
dir to make - 21
Prompt > ls

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
Prompt > ls -l
```

```
Prompt > ls -l

D          112   1
D          112   2
D          112   3
D          112   4
D          112   5
D          112   6
D          112   7
D          112   8
D          112   9
D          112   10
D          112   11
D          112   12
D          112   13
D          112   14
D          112   15
D          112   16
D          112   17
D          112   18
D          112   19
D          112   20
D          112   21
Prompt > ls -l -a
```

```
Prompt > ls -l -a

D         1288   .
D         1288   ..
D          112   1
D          112   2
D          112   3
D          112   4
D          112   5
D          112   6
D          112   7
D          112   8
D          112   9
D          112   10
D          112   11
D          112   12
D          112   13
D          112   14
D          112   15
D          112   16
D          112   17
D          112   18
D          112   19
D          112   20
D          112   21
Prompt > ls -l -a 1
```

```
Prompt > ls -l -a 1

D          112   .
D         1288   ..
Prompt >
```

```
Prompt > md 1/1
dir to make - 1
Prompt > md 1/2
dir to make - 2
Prompt > md 1/3
dir to make - 3
Prompt > md 1/4
dir to make - 4
Prompt > md 1/5
dir to make - 5
Prompt > md 1/6
dir to make - 6
Prompt > md 1/7
dir to make - 7
Prompt > md 1/8
dir to make - 8
Prompt > md 1/9
dir to make - 9
Prompt > md 1/10
dir to make - 10
Prompt > md 1/11
dir to make - 11
Prompt > md 1/12
dir to make - 12
Prompt > md 1/13
dir to make - 13
Prompt > md 1/14
dir to make - 14
Prompt > md 1/15
dir to make - 15
Prompt > md 1/16
dir to make - 16
Prompt > md 1/17
dir to make - 17
Prompt > md 1/18
dir to make - 18
Prompt > md 1/19
```

```
Prompt > md 1/19
dir to make - 19
Prompt > md 1/20
dir to make - 20
Prompt > ls -l -a 1

D         1232   .
D         1288   ..
D          112   1
D          112   2
D          112   3
D          112   4
D          112   5
D          112   6
D          112   7
D          112   8
D          112   9
D          112   10
D          112   11
D          112   12
D          112   13
D          112   14
D          112   15
D          112   16
D          112   17
D          112   18
D          112   19
D          112   20
Prompt >
```

5.2 cd, ls and pwd

```
Prompt > cd 1
Prompt > ls -l -a

D          1232   .
D          1288   ..
D           112   1
D           112   2
D           112   3
D           112   4
D           112   5
D           112   6
D           112   7
D           112   8
D           112   9
D           112   10
D           112   11
D           112   12
D           112   13
D           112   14
D           112   15
D           112   16
D           112   17
D           112   18
D           112   19
D           112   20
Prompt > pwd
/1
Prompt >
```

```
Prompt > pwd
/1
Prompt > cd ..
Prompt > pwd
/
Prompt > cd 1/1
Prompt > pwd
/1/1
Prompt > cd ../..
Prompt > pwd
/
Prompt > cd 1/1
Prompt > ls -l -a

D           112   .
D          1232   ..
Prompt > cd ../..
Prompt > pwd
/
Prompt >
```

5.3 Touch

```
Prompt > touch 1/1/file1
Prompt > touch 1/1/file2
Prompt > touch 1/1/file3
Prompt > touch 1/1/file4
Prompt > touch 1/1/file5
Prompt > touch 1/1/file6
Prompt > touch 1/1/file7
Prompt > touch 1/1/file8
Prompt > touch 1/1/file9
Prompt > touch 1/1/file10
Prompt > md 1/1/1
dir to make - 1
Prompt > touch 1/1/file11
Prompt > touch 1/1/file12
Prompt > touch 1/1/file13
Prompt > touch 1/1/file14
Prompt > touch 1/1/file15
Prompt > touch 1/1/file16
Prompt > touch 1/1/file17
Prompt > touch 1/1/file18
Prompt > ls -l -a 1/1
```

```
Prompt > ls -l -a 1/1

D         1176   .
D         1232   ..
-            0   file1
-            0   file2
-            0   file3
-            0   file4
-            0   file5
-            0   file6
-            0   file7
-            0   file8
-            0   file9
-            0   file10
D          112   1
-            0   file11
-            0   file12
-            0   file13
-            0   file14
-            0   file15
-            0   file16
-            0   file17
-            0   file18
Prompt >
```

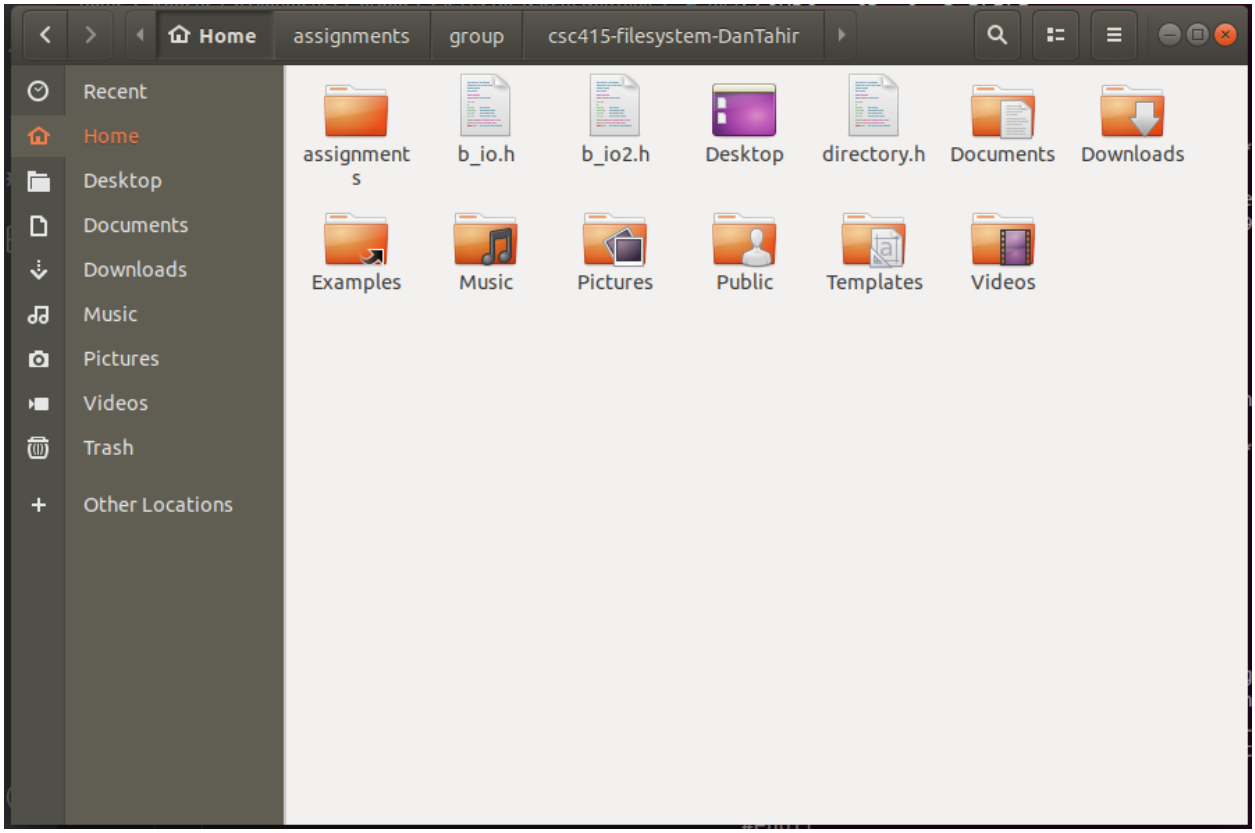
5.4 cp2fs and cat

```
Prompt > cp2fs /home/student/b_io.h 1/1/1/b_io.h
Prompt > ls -l -a 1/1/1

D          168   .
D         1176   ..
-          760   b_io.h
Prompt > cat 1/1/1/b_io.h
```

```
Prompt > cat 1/1/1/b_io.h
/**************************************************************
* Class:  CSC-415-03 Spring 2023
* Names: Danial Tahir, Chris Camano, Mahek Delawala, Savjot Dhillon
* Student IDs: 920838929, 921642160, 922968394, 918239054
* GitHub Name: DanTahir
* Group Name: Segfault
* Project: Basic File System
*
* File: b_io.h
*
* Description: Interface of basic I/O functions
*
**************************************************************/

#ifndef _B_IO_H
#define _B_IO_H
#include <fcntl.h>
#include "file.h"

typedef int b_io_fd;

b_io_fd b_open (char * filename, int flags);
int b_read (b_io_fd fd, char * buffer, int count);
int b_write (b_io_fd fd, char * buffer, int count);
int b_seek (b_io_fd fd, off_t offset, int whence);
int b_close (b_io_fd fd);

#endif

Prompt >
```

5.5 cp2l

```
Prompt > cp2l 1/1/1/b_io.h /home/student/b_io2.h
Prompt >
```




```
student@student-VirtualBox:~$ diff /home/student/b_io.h /home/student/b_io2.h
student@student-VirtualBox:~$
```

b_io2.h - Visual Studio Code

Edit Selection View Go Run Terminal Help

b_io.c | file.h | file.c | fsInit.c | mfs.h | mfs.c | b_io2.h 2

home > student > b_io2.h > ...

```c
/*************************************************************
* Class:  CSC-415-03 Spring 2023
* Names: Danial Tahir, Chris Camano, Mahek Delawala, Savjot Dhillon
* Student IDs: 920838929, 921642160, 922968394, 918239054
* GitHub Name: DanTahir
* Group Name: Segfault
* Project: Basic File System
*
* File: b_io.h
*
* Description: Interface of basic I/O functions
*
*************************************************************/

#ifndef _B_IO_H
#define _B_IO_H
#include <fcntl.h>
#include "file.h"

typedef int b_io_fd;

b_io_fd b_open (char * filename, int flags);
int b_read (b_io_fd fd, char * buffer, int count);
int b_write (b_io_fd fd, char * buffer, int count);
int b_seek (b_io_fd fd, off_t offset, int whence);
int b_close (b_io_fd fd);

#endif
```

5.6 cp2fs and cat – greater than blocksize file

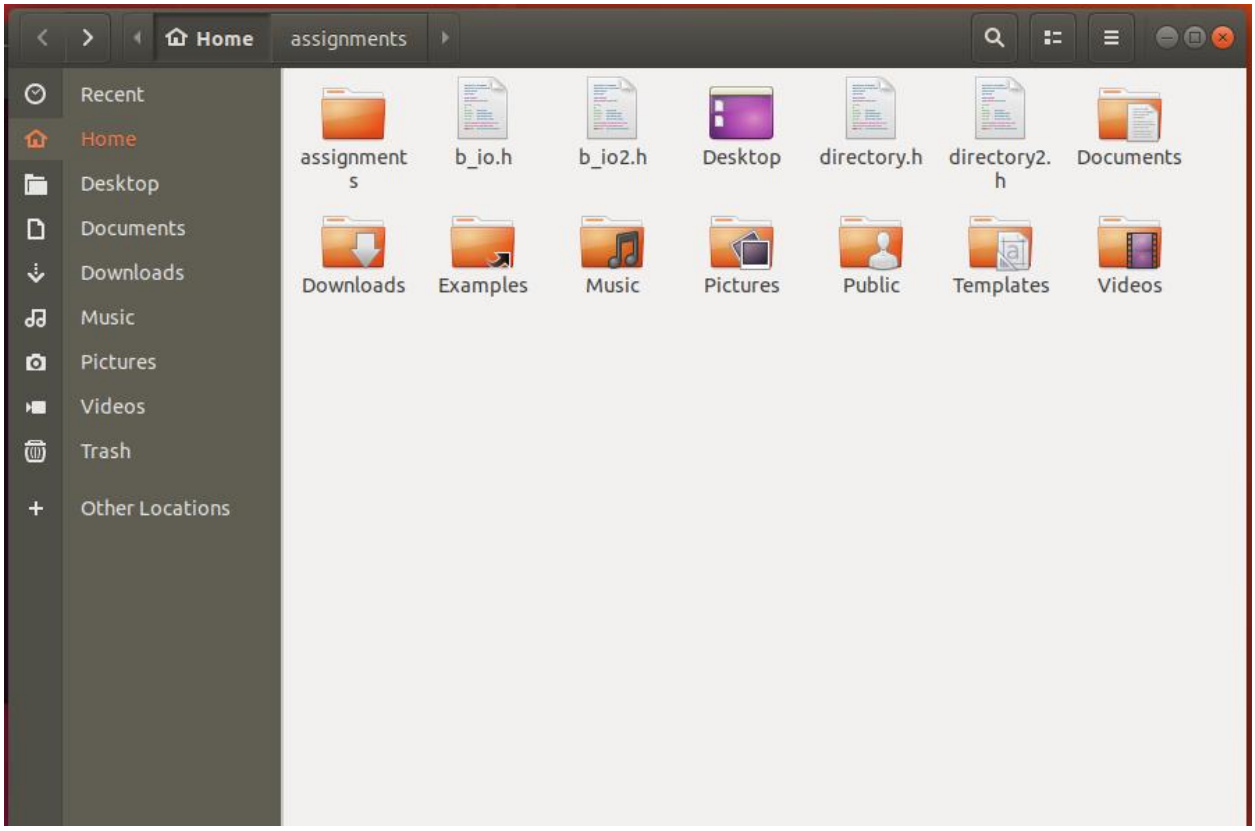
```
Prompt > cp2fs /home/student/directory.h 1/1/1/directory.h
Prompt > ls -l -a 1/1/1

D          224   .
D         1176   ..
-          760   b_io.h
-         2241   directory.h
Prompt >
```

```
Prompt > cat 1/1/1/directory.h
/*****************************************************************
* Class:  CSC-415-03 Spring 2023
* Names: Danial Tahir, Chris Camano, Mahek Delawala, Savjot Dhillon
* Student IDs: 920838929, 921642160, 922968394, 918239054
* GitHub Name: DanTahir, chriscamano, Mahek-Delawala, dsav99
* Group Name: Segfault
* Project: Basic File System
*
* File: directory.h
*
* Description: Declare the structure of the directory entry
* and directory and declare functions related to writing and
* reading directories
*
*****************************************************************/

#include <stdio.h>
#include <string.h>
#include <sys/types.h>

#include "freeSpaceMap.h"


#define NAMELEN 25

typedef struct DirEntry {

    char name[NAMELEN];
    uint64_t location;
    uint64_t size;
    byte isDir;
    bool used;

} DirEntry;
```

```
extern DirEntry * workingDir;

// mallocs a Dir
DirEntry * dirInstance();
// creates a new directory
uint64_t dirInitNew(uint64_t parentDirLoc);
// writes a directory to volume
void dirWrite(DirEntry * dir, uint64_t location);
// reads a directory from volume
void dirRead(DirEntry ** dirp, uint64_t location);
// set the working directory to a given volume location
void dirSetWorking(uint64_t location);
// reread the working directory from the volume
void dirResetWorking();
// allocate memory for the working directory and set it
void dirInitWorking(uint64_t location);
// free the working directory's memory
void dirFreeWorking();
// advance a directory to a given position on the directory tree, returning the
last node in the
// path as a string (which must be allocated memory)
int dirTraversePath(DirEntry ** dirp, const char * pathName, char * endName);
// this copies the working directory to the passed-in directory so
// the passed-in directory can be traversed without changing the
// working directory
void dirCopyWorking(DirEntry ** dirp);
// this adds a new entry to the directory, expanding the size of the directory
int dirAddEntry(DirEntry ** dirp, char * name, uint64_t location, uint64_t size,
 byte isDir);
// this removes an entry from the directory, shrinking the size of the directory
// it's very important that this never gets called on . or ..
int dirRemoveEntry(DirEntry ** dirp, int index);
Prompt >
```

5.7 cp2l - greater than blocksize file

```
Prompt > cp2l directory.h /home/student/directory2.h
Prompt >
```




```
student@student-VirtualBox:~$ diff /home/student/directory.h /home/student/direc
tory2.h
student@student-VirtualBox:~$
```

5.8 cp and mv

```
Prompt > ls 1/1/1

b_io.h
directory.h
Prompt > cp 1/1/1/directory.h file1
Prompt > ls -l -a 1/1/1

D          224   .
D         1176   ..
-          760   b_io.h
-         2241   directory.h
Prompt > cp 1/1/1/directory.h 1/1/1/file1
Prompt > ls -l -a 1/1/1

D          280   .
D         1176   ..
-          760   b_io.h
-         2241   directory.h
-         2241   file1
Prompt > cp 1/1/1/file1 1/1/1/file2
Prompt > cp 1/1/1/file1 1/1/1/file3
Prompt > cp 1/1/1/file1 1/1/1/file4
Prompt > cp 1/1/1/file1 1/1/1/file5
Prompt > ls -l -a 1/1/1
```

```
Prompt > ls -l -a 1/1/1

D          504   .
D         1176   ..
-          760   b_io.h
-         2241   directory.h
-         2241   file1
-         2241   file2
-         2241   file3
-         2241   file4
-         2241   file5
Prompt > mv 1/1/1/file5 1/1/1/file6
Prompt > ls -l -a 1/1/1

D          504   .
D         1176   ..
-          760   b_io.h
-         2241   directory.h
-         2241   file1
-         2241   file2
-         2241   file3
-         2241   file4
-         2241   file6
Prompt > cat 1/1/1/file6
```

```
Prompt > cat 1/1/1/file6
/*************************************************************
* Class:  CSC-415-03 Spring 2023
* Names: Danial Tahir, Chris Camano, Mahek Delawala, Savjot Dhillon
* Student IDs: 920838929, 921642160, 922968394, 918239054
* GitHub Name: DanTahir, chriscamano, Mahek-Delawala, dsav99
* Group Name: Segfault
* Project: Basic File System
*
* File: directory.h
*
* Description: Declare the structure of the directory entry
* and directory and declare functions related to writing and
* reading directories
*
*************************************************************/

#include <stdio.h>
#include <string.h>
#include <sys/types.h>

#include "freeSpaceMap.h"


#define NAMELEN 25

typedef struct DirEntry {

    char name[NAMELEN];
    uint64_t location;
    uint64_t size;
    byte isDir;
    bool used;

} DirEntry;
```

```
} DirEntry;

extern DirEntry * workingDir;

// mallocs a Dir
DirEntry * dirInstance();
// creates a new directory
uint64_t dirInitNew(uint64_t parentDirLoc);
// writes a directory to volume
void dirWrite(DirEntry * dir, uint64_t location);
// reads a directory from volume
void dirRead(DirEntry ** dirp, uint64_t location);
// set the working directory to a given volume location
void dirSetWorking(uint64_t location);
// reread the working directory from the volume
void dirResetWorking();
// allocate memory for the working directory and set it
void dirInitWorking(uint64_t location);
// free the working directory's memory
void dirFreeWorking();
// advance a directory to a given position on the directory tree, returning the
last node in the
// path as a string (which must be allocated memory)
int dirTraversePath(DirEntry ** dirp, const char * pathName, char * endName);
// this copies the working directory to the passed-in directory so
// the passed-in directory can be traversed without changing the
// working directory
void dirCopyWorking(DirEntry ** dirp);
// this adds a new entry to the directory, expanding the size of the directory
int dirAddEntry(DirEntry ** dirp, char * name, uint64_t location, uint64_t size,
 byte isDir);
// this removes an entry from the directory, shrinking the size of the directory
// it's very important that this never gets called on . or ..
int dirRemoveEntry(DirEntry ** dirp, int index);
Prompt >
```

5.9 rm

```
Prompt > ls -l -a 1/1/1

D          504   .
D         1176   ..
-          760   b_io.h
-         2241   directory.h
-         2241   file1
-         2241   file2
-         2241   file3
-         2241   file4
-         2241   file6
Prompt > rm 1/1/1/file6
Prompt > rm 1/1/1/file4
Prompt > ls -l -a 1/1/1

D          392   .
D         1176   ..
-          760   b_io.h
-         2241   directory.h
-         2241   file1
-         2241   file2
-         2241   file3
Prompt > rm 1/1/1
Dir to Delete - 1
directory not empty
```

```
Prompt > md 1/1/2
dir to make - 2
Prompt > ls 1/1

file1
file2
file3
file4
file5
file6
file7
file8
file9
file10
1
file11
file12
file13
file14
file15
file16
file17
file18
2
Prompt > rm 1/1/2
```

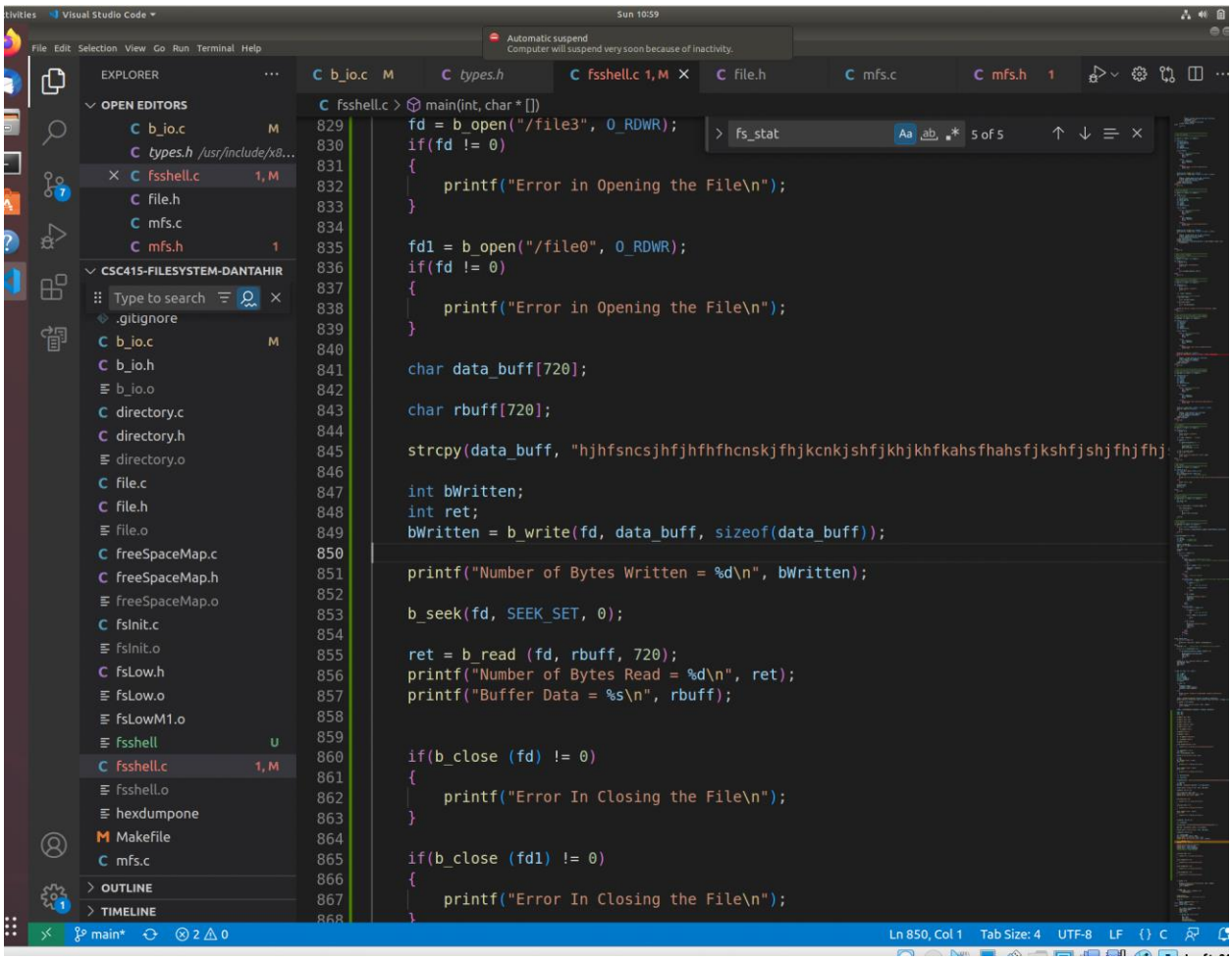
```
Prompt > rm 1/1/2
Dir to Delete - 2
Prompt > ls 1/1

file1
file2
file3
file4
file5
file6
file7
file8
file9
file10
1
file11
file12
file13
file14
file15
file16
file17
file18
Prompt >
```

5.10 md and touch with large filenames

```
Prompt > md 12345678901234567890123456789 0
dir to make - 1234567890123456789012 34
Prompt > ls

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
file1
file2
1234567890123456789012 34
Prompt >
```

```
student@student-VirtualBox:~/assignments/group/csc415-filesystem-DanTahir$ make
run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Initializing File System with 19531 blocks with a block size of 512
signature matched
isvcbset returns 1
Prompt > ls

Prompt > touch 1234567890123456789012345678 90
Prompt > ls

1234567890123456789012 34
Prompt > touch 0123456789012345678901234567890
Prompt > ls

1234567890123456789012 34
0123456789012345678901 23
Prompt >
```

5.11 Output of b_read, b_write function with use of b_seek

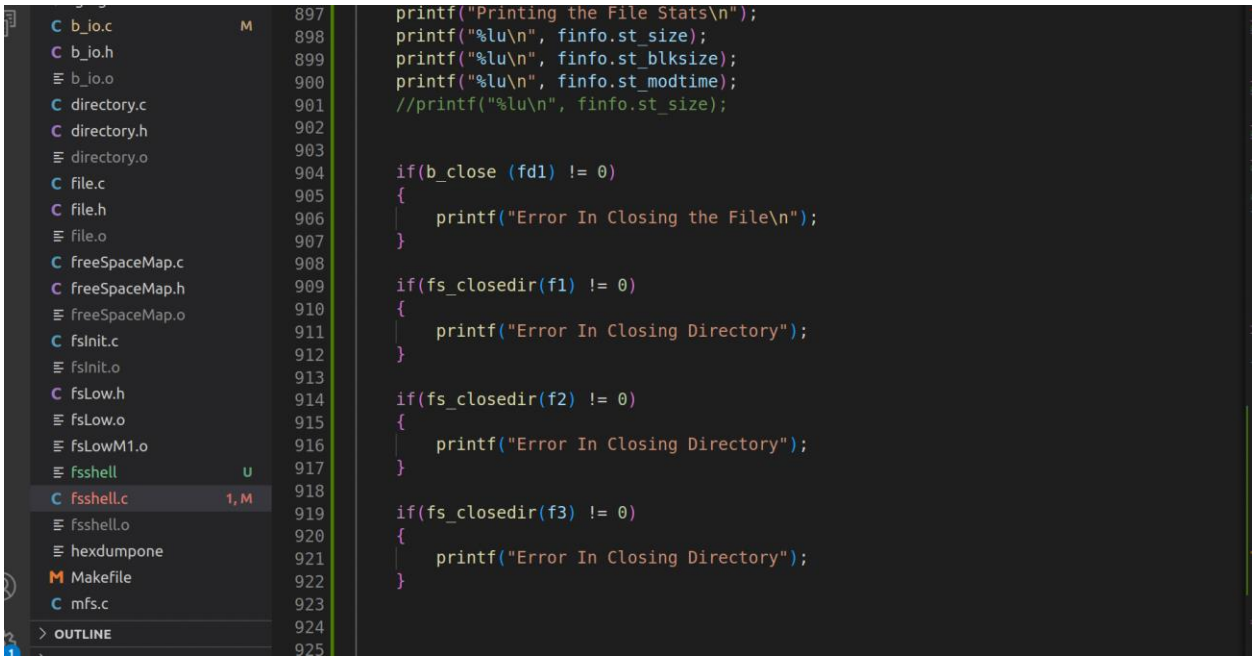


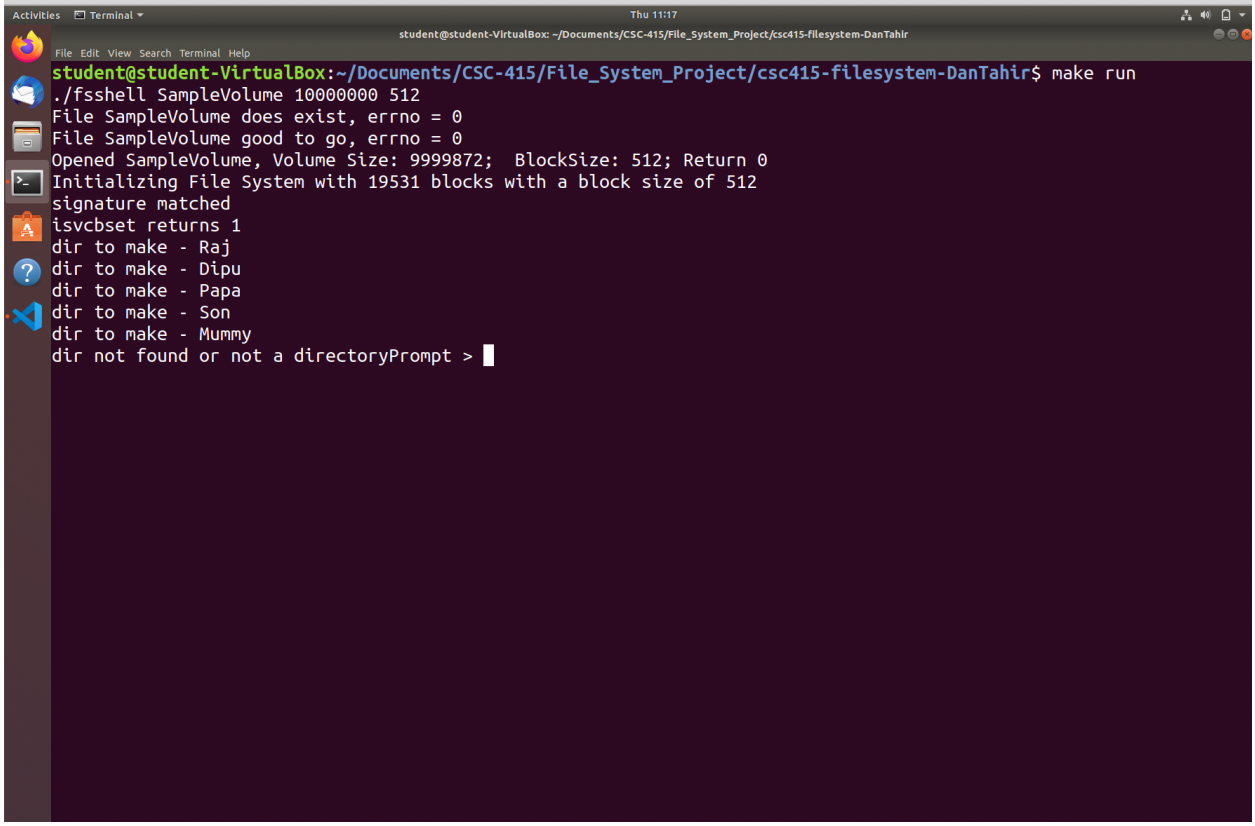
```c
fd = b_open("/file3", O_RDWR);
if(fd != 0)
{
    printf("Error in Opening the File\n");
}

fd1 = b_open("/file0", O_RDWR);
if(fd != 0)
{
    printf("Error in Opening the File\n");
}

char data_buff[720];

char rbuff[720];

strcpy(data_buff, "hjhfsncsjhfjhfhfhcnskjfhjkcnkjshfjkhjkhfkahsfhahsfjkshfjshjfhjfhj

int bWritten;
int ret;
bWritten = b_write(fd, data_buff, sizeof(data_buff));

printf("Number of Bytes Written = %d\n", bWritten);

b_seek(fd, SEEK_SET, 0);

ret = b_read (fd, rbuff, 720);
printf("Number of Bytes Read = %d\n", ret);
printf("Buffer Data = %s\n", rbuff);


if(b_close (fd) != 0)
{
    printf("Error In Closing the File\n");
}

if(b_close (fd1) != 0)
{
    printf("Error In Closing the File\n");
}
```

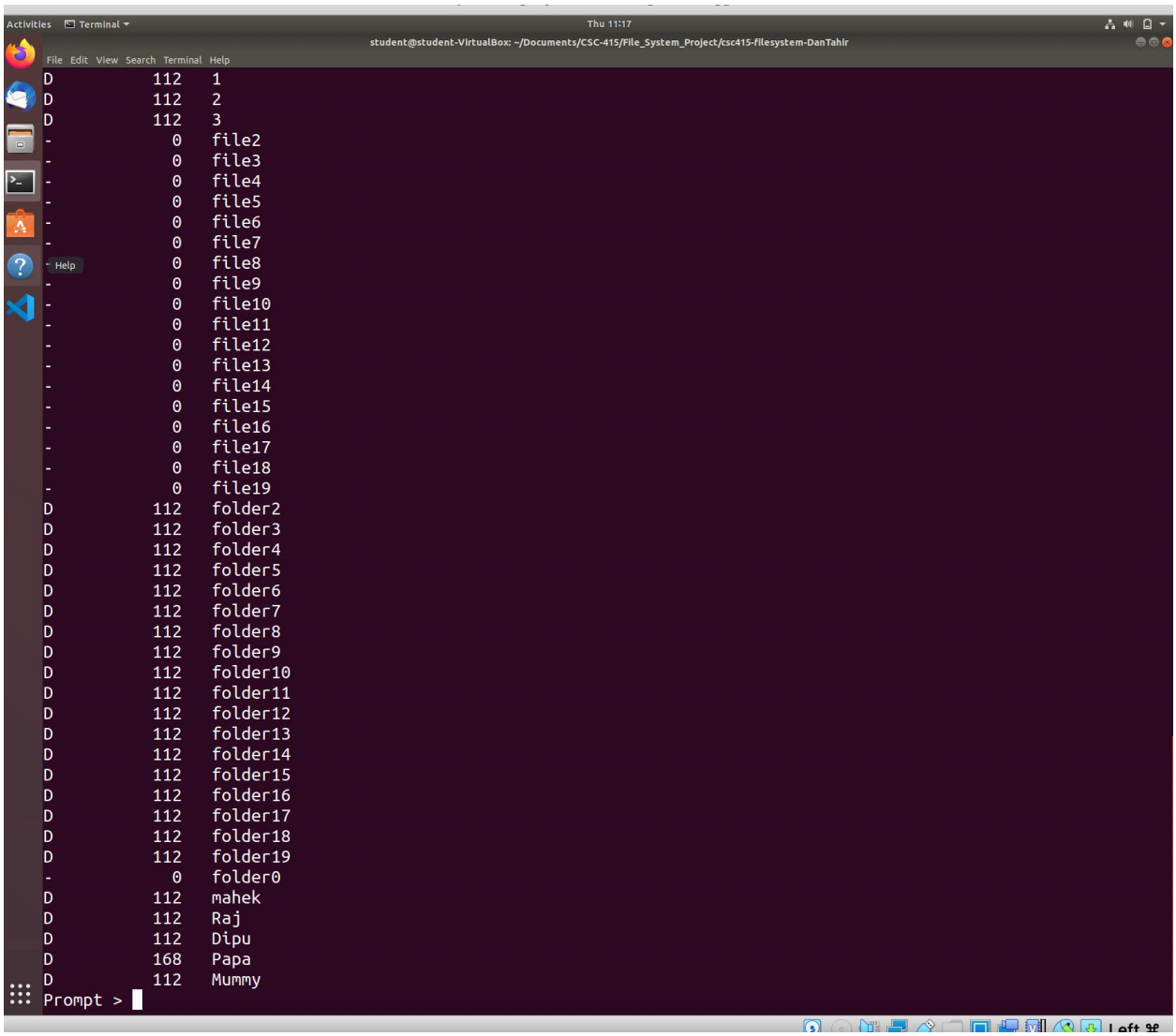
```
student@student-VirtualBox:~/Documents/CSC-415/File_System_Project/csc415-filesystem-DanTahir$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Initializing File System with 19531 blocks with a block size of 512
signature matched
isvcbset returns 1
dir to make - Raj
dirToMake found in dir
dir to make - Dipu
dirToMake found in dir
dir to make - Papa
dirToMake found in dir
dir to make - Son
dirToMake found in dir
dir to make - Mummy
dirToMake found in dir
dir not found or not a directoryCurrent Directory: /Papa/Son
Number of Bytes Written = 720
Number of Bytes Read = 720
Buffer Data = hjhfsncsjhfjhfhfhcnskjfhjkcnkjshfjkhjkhfkahsfhahsfjkshfjshjfhjfhjshhjhfsjakhfwawiurirfhjkshjkshfjkhfjhfncjkshfjkshfj
kshfjkashfjhajfhhwfjhawfhncjskfnvjkhjvsbjehvjkesbjhfjkhweiffhjkfsjcnnvhejhjhfsklfjklejfklsjkljaklwiwuirurrjjfkjsfkhgskjhkfjjkjflfk
jksjkljfkajwlwjfailjfklsajfklhfkghkshfklsajklfjkljfglgkjkjgklsjfgklshjgfklgjhieurifhgjvkfjskljkgjhksglsgvnvlskehgklhklehgijeigjies
gklhegkshgklsegilewiosghgielhsgies384844444444444444444444444444444444ksdfnkdjfsjkljksdjkgjjesnfnffnsnfsfnsfnsnfvnvnvnvkesnkjfkjes
lkjfjeowioitotjgkgnvdnvdsmsmdmdksjgnvdsmnvjksdjgjdjgsgndnmskljfwjwoi389u495urgjdnvldnvdslsjvsmvnv nfkljefjeskjesfkjgkesjgskljgklesj
gkgjksejgkgjklsgjklvsjvlkjvkljskgfjksegji3920203!!@@@!nkjdfnkfnnfnfknd@@#
Enter
Exit
Enter
Exit
Prompt >
```

5.12 opendir() and closedir() output

```
fdDir *f1;
fdDir *f2;
fdDir *f3;

fs_mkdir("/Raj", 077);

fs_mkdir("/Dipu", 077);

fs_mkdir("/Papa", 077);

fs_mkdir("/Papa/Son", 077);

fs_mkdir("Mummy", 077);

f1 = fs_opendir("/Raj");

fs_opendir("/Dipu");

fs_opendir("/Papa");

f2 = fs_opendir("/Papa/Son");

f3 = fs_opendir("Mummy");

fs_opendir("Hero");
```


b_io.c M
b_io.h
b_io.o
directory.c
directory.h
directory.o
file.c
file.h
file.o
freeSpaceMap.c
freeSpaceMap.h
freeSpaceMap.o
fsInit.c
fsInit.o
fsLow.h
fsLow.o
fsLowM1.o
fsshell U
fsshell.c 1, M
fsshell.o
hexdumpone
Makefile
mfs.c
OUTLINE
printf("Printing the File Stats\n");
printf("%lu\n", finfo.st_size);
printf("%lu\n", finfo.st_blksize);
printf("%lu\n", finfo.st_modtime);
//printf("%lu\n", finfo.st_size);
if(b_close (fd1) != 0)
{
printf("Error In Closing the File\n");
}
if(fs_closedir(f1) != 0)
{
printf("Error In Closing Directory");
}
if(fs_closedir(f2) != 0)
{
printf("Error In Closing Directory");
}
if(fs_closedir(f3) != 0)
{
printf("Error In Closing Directory");
}


Activities
Terminal
Thu 11:17
student@student-VirtualBox: ~/Documents/CSC-415/File_System_Project/csc415-filesystem-DanTahir
File Edit View Search Terminal Help
student@student-VirtualBox:~/Documents/CSC-415/File_System_Project/csc415-filesystem-DanTahir$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872; BlockSize: 512; Return 0
Initializing File System with 19531 blocks with a block size of 512
signature matched
isvcbset returns 1
dir to make - Raj
dir to make - Dipu
dir to make - Papa
dir to make - Son
dir to make - Mummy
dir not found or not a directoryPrompt >


Activities
Terminal
Thu 11:17
student@student-VirtualBox: ~/Documents/CSC-415/File_System_Project/csc415-filesystem-DanTahir
File Edit View Search Terminal Help
D 112 1
D 112 2
D 112 3
- 0 file2
- 0 file3
- 0 file4
- 0 file5
- 0 file6
- 0 file7
Help
0 file8
- 0 file9
- 0 file10
- 0 file11
- 0 file12
- 0 file13
- 0 file14
- 0 file15
- 0 file16
- 0 file17
- 0 file18
- 0 file19
D 112 folder2
D 112 folder3
D 112 folder4
D 112 folder5
D 112 folder6
D 112 folder7
D 112 folder8
D 112 folder9
D 112 folder10
D 112 folder11
D 112 folder12
D 112 folder13
D 112 folder14
D 112 folder15
D 112 folder16
D 112 folder17
D 112 folder18
D 112 folder19
- 0 folder0
D 112 mahek
D 112 Raj
D 112 Dipu
D 168 Papa
D 112 Mummy
Prompt >