Files System Project

Team: Bug Master

Team member:

Tun-Ni Chiang 921458769 tunni-chiang

Jiasheng Li 916473043 jiasheng-li

Christopher Ling 918266861 dslayer1392

Shixin Wang 918663491 uyguyguy

1. **The GitHub Link for Our Group Submission:**

   https://github.com/CSC415-Fall2021/csc415-filesystem-jiasheng-li/tree/main

2. **A Description of Our File System**

   a) **A Brief Introduction of Our File System**

   We design a simple file system based on what we have learned from the components of a file system in Linux. Our file system has such components including volume control block, free space, directory entry structures with various metadata, functions that create and maintain directory entries, and functions that manage file operations. Our file system firstly checks the signature of the volume control block to decide whether the system needs to format the volume. After that, if the volume control block has been formatted, our file system reloads the free space. Otherwise, our file system implements bitmap to create and maintain a free space management system. Then, our file system initializes a root directory. The initialization of our file system is completed in the file fsInit.c.The bitmap stores binary data that use '1' to represent a used block and '0' to represent a free block. In addition, our file system has some implementations that can set a used bit to be free or a free bit to be used. The complete implementations of managing free space are in the fsFree.c file. Besides, our file system offers an interactive interface for users to list directories(ls), create directories(md), add and remove files (md, rm), copy files(cp), move files(mv), change directory (cd), copy a file from our file system to the Linux file system (cp21), copy a file from the Linux file system to our file system (cp2fs), print the working directory (pwd), prints out the history(history), and print out help options (help). The complete implementations of directory managements are in fsDir.c, and the complete implementations of file I/O operations are in b_io.c.

b) Core Functions of Our File System

fsDir.c: (Directory Management)

1) **int fs_mkdir(const char *pathname, mode_t mode):** creates a new directory that has a name from the last token of a given path and updates the directory' parent information into the disk.
2) **int fs_rmdir(const char *pathname):** removes the directory with a valid given path or print out the invalid path error message with a invalid given path
3) **fdDir *fs_opendir(const char *name):** open a directory by giving a directory name; return a pointer to the structure that stores the information of the directory if the directory exists. If it does not exist, print out a invalid path message.
4) **struct fs_diriteminfo *fs_readdir(fdDir *dirp):**return a pointer to the structure that stores information for each directory entry of a directory or return Null on reaching the end of the directory.
5) **int fs_closedir(fdDir *dirp):** deallocate the memory for the opened directory and reset the opened directory pointer to NULL
6) **char *fs_getcwd(char *buf, size_t size):** malloces a temp working Dir and make a cope of each temp working dir to an token array by looping back to root directory; concatenate a path by looping the token array; finally, returns a whole absolve path
7) **int fs_setcwd(char *buf):** take a path and check if the path is valid and the name of a directory from the last token of the path exits; if not, return invalid path error message; if yes, set the cwd location to the location of the found directory

8) **int fs_isFile(char *path):** take a given path and check if the path valid with a given condition, EXIST_FILE, for the pathParser function and return the true or false

9) **int fs_isDir(char *path):** take a given path and check if the path valid with a given condition, EXIST_DIR, for the pathParser function and return the true or false

10) **int fs_delete(char *filename):** take a give path and check if the file or directory exist by calling the pathParser function; if the given path is not valid, print out the invalid path error message; if the given path is valid, got the index of the file or directory in temp working directory to release the free space and set the data field of that directory entry index to default values.

11) **int fs_stat(const char *path, struct fs_stat *buf):** take a given path and find out the index of the name of that directory entry and fill in the data files of a fs_star structure

12) **int createDir(int parentLocation):** a helper function for making a directory to initialize each directory entry in a directory

13) **bool pathParser(char *path, unsigned char condition, DE *tempWorkingDir, char *lastToken):** a helper function to check a give path is valid as a file, a directory, or either a file or directory

B_io.c: (file I/O operation)

14) **b_io_fd b_open(char *filename, int flags):** returns a free file descriptor to an individual file; the file descriptor will be used by other file I/O operation functions to refer that file

15) **int b_read(b_io_fd fd, char *buffer, int count):** reads data if a file from the volume by taking the file descriptor of that file; read up to the size in bytes of the buffer of the structure fcb

16) **int b_write(b_io_fd fd, char *buffer, int count):** write data up to the size in bytes of the buffer into out buffer of the fcb and write them into the disk

17) **int b_seek(b_io_fd fd, off_t offset, int whence):** returns the offset location as measure in bytes from the beginning of the file by take a file descriptor of a file when read or write the file

18) **void b_close(b_io_fd fd):** deallocate the memory for the buffer of the file descriptor and set the buffer pointer to NULL; also, free the file descriptor to future file I/O operations

c) Core Structures of Our File System

Mfs.h: ()

1) A struct to manage our bitmap implementation

```
typedef struct
{
        int blockSize;
        int totalOfBlock;
        unsigned char *bitMap;
        int location;
        int lastAllocBlock; //temp
        int usedCount;          //for our purpose
        int freeCount;          //for our purpose
} freeSpaceManager;
```

2) A struct to store metadata for each directory entry

```
typedef struct
{
```

```
        char name[64];
        int size;
        int pointingLocation;
        bool isDir;
        time_t createTime;
        time_t lastModTime;
        time_t lastAccessTime;
} DE;
```

3) A structure to store the information for our volume control block

```
typedef struct
{
        char signature[64];
        int numberOfBlocks;
        int blockSize;
        int location;
        int bitMapLocation;
        int rootLocation;
        int rootSize;
        int freeSpaceStartLocation;
        int nextFreeBlock;
} VCB;
```

3. **Issue We Had**
   a) How do we know whether we need to initialize the volume control block or we know the volume control block is valid?
   Solution: we had a hint from Professor Robert Bierman during our first office hour with him. The volume control block is a struct that has random data. We need to check the signature of the volume control which is the value assigned by us. The signature is a large random number that can be 4 bytes or 8 bytes. We hard code the signature of our volume control block. When we initialize our file system, we need to check the signature of that volume control block first. If the signature matches our signature, we don't need to format the volume control block which was already formatted. If not matched, the volume control needs to be formatted and written for us.

   b) Is the size of the root directory arbitrarily assigned by us?
   Solution: most file systems have a fixed root directory size, but we don't have to make it fixed. A root directory can be sufficiently larger and global. It is up to us. Some DEs may be global.

   c) How to flip a bit of the bitmap from free to use or from used to free?

```
int setBitUsed(char *array, int bitIndex);
int setBitFree(char *array, int bitIndex);
int checkBitUsed(char *array, int bitIndex);
```

```
int setBitUsed(unsigned char *array, int bitIndex);
int setBitFree(unsigned char *array, int bitIndex);
int checkBitUsed(unsigned char *array, int bitIndex);
void printManager(freeSpaceManager *manager);
```

```
//return 1 -> success, -1 -> failed
int setBitUsed(unsigned char *array, int bitIndex)
{
    printf("[debug] setting bit to used...\n");
    array[bitIndex] = '1';

    //double check if we set the bit successfully and return result
    return checkBitUsed(array, bitIndex);
}
```

   Solution: we need to use an unsigned char* array which is the array of bytes to manage our bitmap. However, the char * array is the array working for byte, but the bitmap is working for bits. array[bitIndex] just touch the type of 8 bits, not each individual bit. As a result, we need the working byte corresponding to the bit that we want to flip by using the

block number dividing by 8. After that, we need to calculate the bit index of that byte by modding the block number by 8. After we get the working byte index and the bit index of that byte, we need to use bitwise operators '|' and '&' to flip that bit. For the setBitUsed function, we declare unsigned char setArray[8] = {0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01} to flip a certain free bit to be used by the or operation , '|'. It will be array[bitIndex / 8] = array[bitIndex / 8] | setArray[bitIndex % 8]. For the setBitFree function, we declare unsigned char clearArray[8] = {0x7F, 0xBF, 0xDF, 0xEF, 0xF7, 0xFB, 0xFD, 0xFE} to flip a certain used bit to be free by the and operation, '&'. It will be array[bitIndex / 8] = array[bitIndex / 8] & clearArray[bitIndex % 8].

d) How should we declare the pointer of the bitmap? Should it be inside the mfsh.h? Should it be a global variable?
Solution: since only the free space functions will touch the pointer of the bitmap, we should only declare it inside fsFree.c which is the implementation file for free space management. It also should be global to outside any functions of fsFree.c.

e) What is the way to initialize the time variables of the directory entry struct?

```
72  typedef struct
73  {
74          char name[64];
75          int size;
76          int pointingLocation;
77          bool isDir;
78          time_t createTime;
79          time_t lastModTime;
80          time_t lastAccessTime;
81  } DE;
```

Solution: since variables of time_t is not the variable of pointer type, they should be initialized as value 0 which is the initialized value for integer-type variables. Also, the initialized value for pointer-type variables should be Null.

f) What is the location of directory entry? Is it the address currently in memory location?
Solution: The location stored in the disk is a logic block number where the directory entry item is stored. An address will not be the same when we reboot our file system. A directory entry array is stored in certain logic blocks, and the directory entry in that directory entry array is just a pointer that is pointing to another directory entry array that is stored in other logic blocks.

g) How to define different conditions that will be passed in the function bool pathParser(char *path, unsigned char condition, DE *tempWorkingDir, char *lastToken)?

Solution: Initially, we planned to use integers to represent four conditions including valid file, valid directory, valid file and directory, and not found. The integer '1' represents the condition 'valid file'; the integer '2' represents the condition 'valid directory'; the integer '3' represents the condition 'valid file and directory'; the integer '0' represents the condition 'not found. However, the implementation was not the good implementation as commented by Professor Rober. Under his hint during one of his office hours, we should use the binary bits to represent the four conditions. As a result, we have:

```
#define NOT_EXIST 0x00000000
#define EXIST_FILE 0x00000001
#define EXIST_DIR 0x00000002
#define EXIST 0x00000004
```

h) We were passing the DE pointer tempWokringDir in the function bool pathParser() and printing out the information of tempWokringDir. The information of tempWokringDir was correct, but the information of tempWokringDir was lost when the function pathParser() was invoked inside the function fs_mkdir()
DE* tempWokringDir inside the function pathParser():

```
85  bool pathParser(char *path, unsigned char condition, DE *tempWorkingDir, char *lastToken)
86  {
```

```
    printf("[debug] print out the tempWorkingDir info (this is the info for second last tok
    printDEInfo(tempWorkingDir[0]);
    unsigned char thisCondition = NOT_EXIST;
```

DE* tempWokringDir inside the function fs_mkdir():

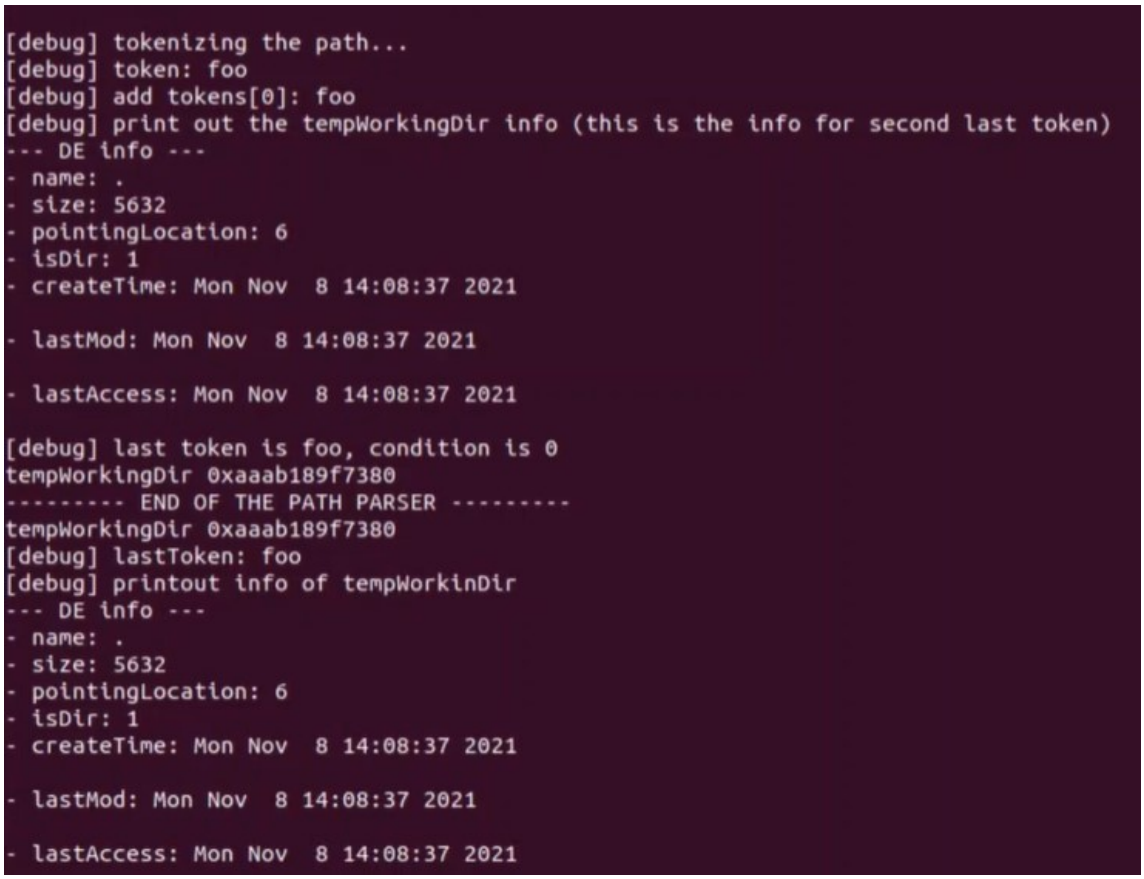
```
//finished implementing, still need to be tested
int fs_mkdir(const char *pathname, mode_t mode)
{
    printf("\n--------- INSIDE THE MKDIR ---------\n");

    char path[strlen(pathname)];
    strcpy(path, pathname);

    //1
    DE *tempWorkingDir = malloc(mfs_cwd[0].size);
    char *lastToken = malloc(256); //TODO hardcode

    //2
    bool valid = pathParser(path, NOT_EXIST, &tempWorkingDir, lastToken);

    printf("[debug] lastToken: %s\n", lastToken);
    printf("[debug] printout info of tempWorkinDir\n");
    printDEInfo(tempWorkingDir[0]);
```

We printed out the DE pointer tempWokringDir inside the pathParser() and the function fs_mkdir(), so we found that the addresses of tempWokringDir are different.

```
[debug] last token is foo, condition is 0
tempWorkingDir 0xaaaadae10380
--------- END OF THE PATH PARSER ---------
tempWorkingDir 0xaaaadae45e90
[debug] lastToken: foo
[debug] printout info of tempWorkinDir
--- DE info ---
```

We finally make a pointer to a pointer as a parameter for the function pathParser(), which is DE ** temWorkingDir. As a result, the addresses of temWorkingDir inside the function pathParser() and function fs_mkdir() were the same.

```
bool pathParser(char *path, unsigned char condition, DE **tempWorkingDir, char *lastToken)
{
    printf("\n--------- INSIDE THE PATH PARSER ---------\n");
    printf("[debug] arguments:\n- path: %s\n- condition: %d\n", path, condition);

    if (path[0] != '/')
    {
        printf("[debug] User passed in relative path... \n");
        tempWorkingDir = mfs_cwd;
    }
    else
    {
        printf("[debug] User passed in absolute path... \n");
        printf("[debug] root has %d blocks at %d location\n", mfs_vcb->rootSize / mfs_blockS
        LBAread(tempWorkingDir, mfs_vcb->rootSize / mfs_blockSize, mfs_vcb->rootLocation);
    }

    printf("[debug] print out the tempWorkingDir info\n");
    printDEInfo(*tempWorkingDir[0]);
```



i) When we entered the command “md foo/bar/folder1/folder2” to make directories for four layers, our file system had malloc () invalid size () unsorted which means the file system overruns something in memory. We found the error was from the implementation before an invocation of the function createDir() inside the function fs_mkdir(). With assistance from Professor Robert, we found that the mallocSize error was from the function pathParser(), so we looked into the implementation of the function pathParser(). The error was from char** tokens = malloc(strlen(path)) which should allocate memory for the number of pointers. However, char** tokens = malloc(strlen(path)) allocate the number of characters that the path has, so the malloc size was not enough for the number of pointers of the char array. For example, the path “foo/bar/folder1/folder2” will be a char array {“foo”, “bar”, “folder1”, “folder2”} that needs memory allocation for four pointers. Four pointers need 32 bytes of memory, but strlen(path) only has 24 bytes. Finally, we the code to be char **tokens = malloc((strlen(path) / 2) * sizeof(char *)), and the error disappeared.

Showing the error, malloc () invalid size () unsorted:

```
[debug] last token is folder2, condition is 0
[debug] tempWorkingDir @ 0xaaaaff8300c0
[debug] comparing the given condition with this condition, condition: 0, this condition: 0
--------- END OF THE PATH PARSER ---------
[debug] valid: 1
[debug] lastToken: folder2
[debug] tempWorkingDir @ 0xaaaaff8300c0
[debug] printout info of tempWorkinDir
--- DE info ---
- name: .
- size: 5632
- pointingLocation: 39
- isDir: 1
- createTime: Wed Nov 17 13:56:22 2021

- lastMod: Wed Nov 17 13:56:22 2021

- lastAccess: Wed Nov 17 13:56:22 2021

[debug] tempWorkingDir[2].name:
[debug] parent location: 39

--------- INSIDE THE CREATE DIR ---------
[debug] mallocSize: 5632
[debug] err issue
malloc(): invalid size (unsorted)
make: *** [Makefile:67: run] Aborted (core dumped)
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$
```

```
bool pathParser(char *path, unsigned char condition, DE *tempWorkingDir, char *lastToken)
{
    printf("\n--------- INSIDE THE PATH PARSER ---------\n");
    printf("[debug] arguments:\n- path: %s\n- condition: %d\n", path, condition);

    if (path[0] != '/')
    {
        printf("[debug] User passed in relative path... \n");
        int mallocSize = sizeof(DE) * mfs_defaultDECount;
        int numOfBlockNeeded = (mallocSize / mfs_blockSize) + 1;
        LBAread(tempWorkingDir, numOfBlockNeeded, mfs_cwd_location);
        printf("[debug] print out tempWorkingDir info\n");
        printDEInfo(tempWorkingDir[0]);
        // tempWorkingDir = mfs_cwd;
    }
    else
    {
        printf("[debug] User passed in absolute path... \n");
        printf("[debug] root has %d blocks at %d location\n", mfs_vcb->rootSize / mfs_block
        LBAread(tempWorkingDir, mfs_vcb->rootSize / mfs_blockSize, mfs_vcb->rootLocation);
    }

    printf("[debug] print out the tempWorkingDir info\n");
    printDEInfo(tempWorkingDir[0]);

    int tokenCount = 0;
    char **tokens = malloc(strlen(path)); //TODO check malloc
    char *theRest;
    char *token = strtok_r(path, "/", &theRest);
```

Fixing the error:

```
--------- INSIDE THE CREATE DIR ---------
[debug] mallocSize: 5632
[debug] err issue
[debug] next freeblock: 50

--------- END THE CREATE DIR ---------
[debug] printout directory info for new directory
--- DE info ---
- name: folder2
- size: 5632
- pointingLocation: 50
- isDir: 1
- createTime: Wed Nov 17 14:11:47 2021

- lastMod: Wed Nov 17 14:11:47 2021

- lastAccess: Wed Nov 17 14:11:47 2021

Prompt >
```

j) We had a warning for calling fs_delete(pathname) in the functionfs_rmdir() . the pathname is const Char * type. The parameter of the function fs_delete() should be char*, but we were passing the const char * into the function fs_delete(). Const char * allows read-only in a memory location, but char* allows reading and writing in a memory location. As a result, we cast Const char * to char * by coding int ret = fs_delete((char *)pathname);

```
File  Edit  View  Search  Terminal  Help
student@student-VirtualBox:~/Documents/CSC-415-Group-Projects/csc415-filesystem-jiasheng-li$ 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 fsDir.o fsDir.c -g -I.
fsDir.c: In function ‘fs_rmdir’:
fsDir.c:367:25: warning: passing argument 1 of ‘fs_delete’ discards ‘const’ qualifier from pointer target type [-Wdiscarded-qualifiers]
     int ret = fs_delete(pathname);
                         ^~~~~~~~
In file included from fsFree.h:15:0,
                 from fsDir.h:15,
                 from fsDir.c:15:
mfs.h:105:5: note: expected ‘char *’ but argument is of type ‘const char *’
 int fs_delete(char *filename); //removes a file return the deleted file's location, -1 means failed
     ^~~~~~~~~
gcc -o fsshell fsshell.o fsInit.o fsFree.o fsDir.o fsLow.o -g -I. -lm -l readline -l pthread
student@student-VirtualBox:~/Documents/CSC-415-Group-Projects/csc415-filesystem-jiasheng-li$
```

4. Detail of How Our Driver Program Works

The implementations of our driver program are in fsshell.c. The fsshell.c file has the main function to run the entire file system and all command functions to take user inputs to output information and complete operations of the file system that users preferred. The main function starts partition by invoking the function startPartitionSystem() with file name, volume size, and block size extracted from command line arguments. After that, the main function invokes the function initFileSystem() to reload the free space management if the volume has been formatted. Otherwise, if the volume has not been formatted, the driver program initializes the volume along with initializations of free space management and root directory and writes the volume into the disk. As a result, the driver program can read the volume and access the data at the memory level. Then, the driver program would run as an interactive interface that prompts users to enter commands to output information of the file system or process the file and directory I/O Operations. Besides, the driver program can exit and terminate the shell by invoking the functions exitFileSystem() and closePartitionSystem() in the main function. To conclude, our driver program is a simple file system that can perform various tasks and handle files in a volume like a file system of the Linux operating system. Also, our driver program takes the most common commands of the Linux operating system.

Introduction of Commands of Our Driver Program:

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

5. **Screen Shots Showing Each of the Commands Listed in the README**
   Note: included hexdump screenshot in case you needed :)

```
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$ 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 fsDir.o fsDir.c -g -I.
gcc -c -o b_io.o b_io.c -g -I.
gcc -o fsshell fsshell.o fsInit.o fsFree.o fsDir.o b_io.o fsLowM1.o -g -I. -lm -l readline -l pthread
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$
```

ls & md - terminal

```
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$ make run
gcc -c -o fsInit.o fsInit.c -g -I.
gcc -o fsshell fsshell.o fsInit.o fsFree.o fsDir.o b_io.o fsLowM1.o -g -I. -lm -l readline -l pthread
./fsshell 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
Prompt > md foo
Prompt > md bar
Prompt > md folder
Prompt > ls

foo
bar
folder
Prompt > exit
System exiting
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$
```

ls & md - hexdump of root

```
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$ Hexdump/hexdump.linuxM1 --count 1 --start 7 SampleVolume
Dumping file SampleVolume, starting at block 7 for 1 block:

000E00: 2E 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E40: 00 16 00 00 30 02 00 00  0B 00 00 00 06 00 00 00 | ....0...........
000E50: 01 00 00 00 00 00 00 00  9D 36 A9 61 00 00 00 00 | .........6.a....
000E60: 9D 36 A9 61 00 00 00 00  9D 36 A9 61 00 00 00 00 | .6.a....6.a....
000E70: 2E 2E 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000EA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000EB0: 00 16 00 00 00 00 00 00  00 00 00 00 06 00 00 00 | ................
000EC0: 01 00 00 00 00 00 00 00  9D 36 A9 61 00 00 00 00 | .........6.a....
000ED0: 9D 36 A9 61 00 00 00 00  9D 36 A9 61 00 00 00 00 | .6.a....6.a....
000EE0: 66 6F 6F 00 00 00 00 00  00 00 00 00 00 00 00 00 | foo.............
000EF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000F00: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F20: 00 16 00 00 E0 00 00 00  0B 00 00 00 11 00 00 00 | ....`...........
000F30: 01 00 00 00 00 00 00 00  A1 36 A9 61 00 00 00 00 | .........6.a....
000F40: A1 36 A9 61 00 00 00 00  A1 36 A9 61 00 00 00 00 | .6.a....6.a....
000F50: 62 61 72 00 00 00 00 00  00 00 00 00 00 00 00 00 | bar.............
000F60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F90: 00 16 00 00 E0 00 00 00  0B 00 00 00 1C 00 00 00 | ....`...........
000FA0: 01 00 00 00 00 00 00 00  A2 36 A9 61 00 00 00 00 | .........6.a....
000FB0: A2 36 A9 61 00 00 00 00  A2 36 A9 61 00 00 00 00 | .6.a....6.a....
000FC0: 66 6F 6C 64 65 72 00 00  00 00 00 00 00 00 00 00 | folder..........
000FD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000FE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000FF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$
```

ls & md - hexdump of foo (similar to bar and folder)

```
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$ Hexdump/hexdump.linuxM1 --count 1 --start 18 SampleVolume
Dumping file SampleVolume, starting at block 18 for 1 block:

002400: 2E 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002410: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002420: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002430: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002440: 00 16 00 00 E0 00 00 00  0B 00 00 00 11 00 00 00 | ....`...........
002450: 01 00 00 00 00 00 00 00  A1 36 A9 61 00 00 00 00 | .........6.a....
002460: A1 36 A9 61 00 00 00 00  A1 36 A9 61 00 00 00 00 | .6.a....6.a....
002470: 2E 2E 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002480: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002490: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0024A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0024B0: 00 16 00 00 00 00 00 00  00 00 00 00 06 00 00 00 | ................
0024C0: 01 00 00 00 00 00 00 00  A1 36 A9 61 00 00 00 00 | .........6.a....
0024D0: A1 36 A9 61 00 00 00 00  A1 36 A9 61 00 00 00 00 | .6.a....6.a....
0024E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0024F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

002500: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002510: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002520: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002530: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002540: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002550: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002560: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002570: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002580: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002590: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0025A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0025B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0025C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0025D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0025E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0025F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$
```

rm - terminal

```
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$ 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
Prompt > rm foo
Prompt > ls

bar
folder
Prompt > exit
System exiting
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$
```

rm - hexdump of root

```
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$ Hexdump/hexdump.linuxM1 --count 1 --start 7 SampleVolume
Dumping file SampleVolume, starting at block 7 for 1 block:

000E00: 2E 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E40: 00 16 00 00 30 02 00 00  0B 00 00 00 06 00 00 00 | ....0...........
000E50: 01 00 00 00 00 00 00 00  9D 36 A9 61 00 00 00 00 | .........6.a....
000E60: 9D 36 A9 61 00 00 00 00  9D 36 A9 61 00 00 00 00 | .6.a....6.a....
000E70: 2E 2E 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000EA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000EB0: 00 16 00 00 00 00 00 00  00 00 00 00 06 00 00 00 | ................
000EC0: 01 00 00 00 00 00 00 00  9D 36 A9 61 00 00 00 00 | ........6.a....
000ED0: 9D 36 A9 61 00 00 00 00  9D 36 A9 61 00 00 00 00 | .6.a....6.a....
000EE0: 00 6F 6F 00 00 00 00 00  00 00 00 00 00 00 00 00 | .oo.............
000EF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000F00: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F20: 00 00 00 00 E0 00 00 00  0B 00 00 00 00 00 00 00 | .................
000F30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F50: 62 61 72 00 00 00 00 00  00 00 00 00 00 00 00 00 | bar.............
000F60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000F90: 00 16 00 00 E0 00 00 00  0B 00 00 00 1C 00 00 00 | ....#...........
000FA0: 01 00 00 00 00 00 00 00  A2 36 A9 61 00 00 00 00 | ........6.a....
000FB0: A2 36 A9 61 00 00 00 00  A2 36 A9 61 00 00 00 00 | .6.a....6.a....
000FC0: 66 6F 6C 64 65 72 00 00  00 00 00 00 00 00 00 00 | folder..........
000FD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000FE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000FF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$
```

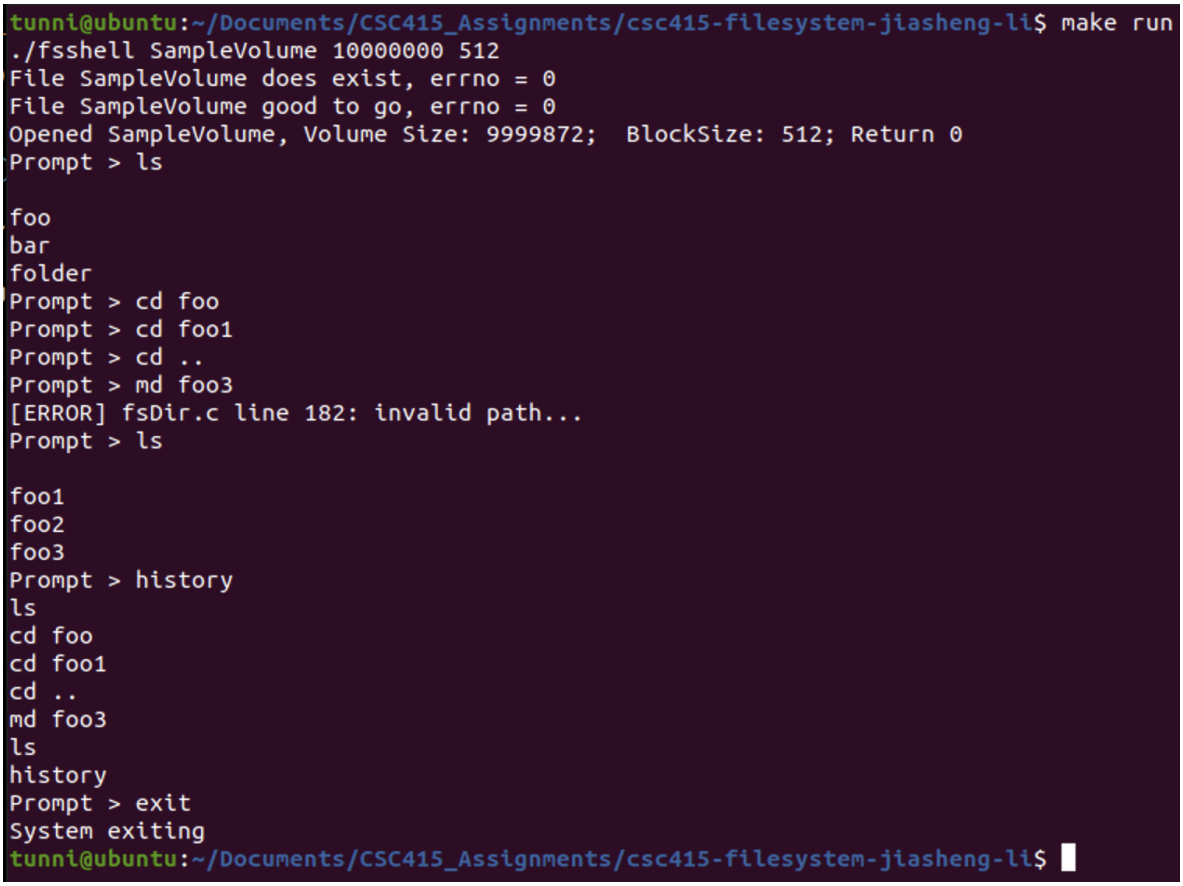
cd - terminal

```
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$ 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
Prompt > md foo/foo1
Prompt > md foo/foo2
Prompt > md foo/foo3
Prompt > ls

foo
bar
folder
Prompt > cd foo
Prompt > ls

foo1
foo2
foo3
Prompt > exit
System exiting
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$
```

pwd - terminal

```
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$ 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
Prompt > pwd
/
Prompt > cd foo
Prompt > pwd
/foo
Prompt > cd foo1
Prompt > pwd
/foo/foo1
Prompt > exit
System exiting
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$
```

history - terminal



help - terminal

```
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$ 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
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 > exit
System exiting
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$
```

cp2fs - terminal

```
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$ make run
./fsshell 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
Prompt > cp2fs a.txt b.txt
Prompt > exit
System exiting
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$
```

cp2fs - Hexdump

```
tunnl@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$ Hexdump/hexdump.linuxM1 --count 3 --start 18 SampleVolume
Dumping file SampleVolume, starting at block 18 for 3 blocks:

002400: 4C 6F 76 65 20 69 73 20  61 20 73 65 74 20 6F 66 | Love is a set of
002410: 20 65 6D 6F 74 69 6F 6E  73 2C 20 62 65 68 61 76 |  emotions, behav
002420: 69 6F 72 73 2C 20 61 6E  64 20 62 65 6C 69 65 66 | iors, and belief
002430: 73 20 77 69 74 68 20 73  74 72 6F 6E 67 20 66 65 | s with strong fe
002440: 65 6C 69 6E 67 73 20 6F  66 0A 61 66 66 65 63 74 | elings of.affect
002450: 69 6F 6E 2E 20 53 6F 2C  20 66 6F 72 20 65 78 61 | ion. So, for exa
002460: 6D 70 6C 65 2C 20 61 20  70 65 72 73 6F 6E 20 6D | mple, a person m
002470: 69 67 68 74 20 73 61 79  20 68 65 20 6F 72 20 73 | ight say he or s
002480: 68 65 20 6C 6F 76 65 73  20 68 69 73 20 6F 72 20 | he loves his or
002490: 68 65 72 0A 64 6F 67 2C  20 6C 6F 76 65 73 20 66 | her.dog, loves f
0024A0: 72 65 65 64 6F 6D 2C 20  6F 72 20 6C 6F 76 65 73 | reedom, or loves
0024B0: 20 47 6F 64 2E 20 54 68  65 20 63 6F 6E 63 65 70 |  God. The concep
0024C0: 74 20 6F 66 20 6C 6F 76  65 20 6D 61 79 20 62 65 | t of love may be
0024D0: 63 6F 6D 65 20 61 6E 0A  75 6E 69 6D 61 67 69 6E | come an.unimagin
0024E0: 61 62 6C 65 20 74 68 69  6E 67 20 61 6E 64 20 61 | able thing and a
0024F0: 6C 73 6F 20 69 74 20 6D  61 79 20 68 61 70 70 65 | lso it may happe

002500: 6E 20 74 6F 20 65 61 63  68 20 70 65 72 73 6F 6E | n to each person
002510: 20 69 6E 20 61 20 70 61  72 74 69 63 75 6C 61 72 |  in a particular
002520: 0A 77 61 79 2E 0A 4C 6F  76 65 20 68 61 73 20 61 | .way..Love has a
002530: 20 76 61 72 69 65 74 79  20 6F 66 20 66 65 65 6C |  variety of feel
002540: 69 6E 67 73 2C 20 65 6D  6F 74 69 6F 6E 73 2C 20 | ings, emotions,
002550: 61 6E 64 20 61 74 74 69  74 75 64 65 2E 20 46 6F | and attitude. Fo
002560: 72 20 73 6F 6D 65 6F 6E  65 20 6C 6F 76 65 0A 69 | r someone love.i
002570: 73 20 6D 6F 72 65 20 74  68 61 6E 20 6A 75 73 74 | s more than just
002580: 20 62 65 69 6E 67 20 69  6E 74 65 72 65 73 74 65 |  being intereste
002590: 64 20 70 68 79 73 69 63  61 6C 6C 79 20 69 6E 20 | d physically in
0025A0: 61 6E 6F 74 68 65 72 20  6F 6E 65 2C 20 72 61 74 | another one, rat
0025B0: 68 65 72 20 69 74 20 69  73 20 61 6E 0A 65 6D 6F | her it is an.emo
0025C0: 74 69 6F 6E 61 6C 20 61  74 74 61 63 68 6D 65 6E | tional attachmen
0025D0: 74 2E 20 57 65 20 63 61  6E 20 73 61 79 20 6C 6F | t. We can say lo
0025E0: 76 65 20 69 73 20 6D 6F  72 65 20 6F 66 20 61 20 | ve is more of a
0025F0: 66 65 65 6C 69 6E 67 20  74 68 61 74 20 61 20 00 | feeling that a .
```

```
002600: 70 65 72 73 6F 6E 0A 66  65 65 6C 73 20 66 6F 72 | person.feels for
002610: 20 61 6E 6F 74 68 65 72  20 70 65 72 73 6F 6E 2E |  another person.
002620: 20 54 68 65 72 65 66 6F  72 65 2C 20 74 68 65 20 |  Therefore, the
002630: 62 61 73 69 63 20 6D 65  61 6E 69 6E 67 20 6F 66 | basic meaning of
002640: 20 6C 6F 76 65 20 69 73  20 74 6F 20 66 65 65 6C |  love is to feel
002650: 0A 6D 6F 72 65 20 74 68  61 6E 20 6C 69 6B 69 6E | .more than likin
002660: 67 20 74 6F 77 61 72 64  73 20 73 6F 6D 65 6F 6E | g towards someon
002670: 65 2E 0A 57 65 20 6B 6E  6F 77 20 74 68 61 74 20 | e..We know that
002680: 74 68 65 20 64 65 73 69  72 65 20 74 6F 20 6C 6F | the desire to lo
002690: 76 65 20 61 6E 64 20 63  61 72 65 20 66 6F 72 20 | ve and care for
0026A0: 6F 74 68 65 72 73 20 69  73 20 61 20 68 61 72 64 | others is a hard
0026B0: 2D 77 69 72 65 64 20 61  6E 64 0A 64 65 65 70 2D | -wired and.deep-
0026C0: 68 65 61 72 74 65 64 20  62 65 63 61 75 73 65 20 | hearted because
0026D0: 74 68 65 20 66 75 6C 66  69 6C 6C 6D 65 6E 74 20 | the fulfillment
0026E0: 6F 66 20 74 68 69 73 20  77 69 73 68 20 69 6E 63 | of this wish inc
0026F0: 72 65 61 73 65 73 20 74  68 65 20 68 61 70 70 69 | reases the happi

002700: 6E 65 73 73 0A 6C 65 76  65 6C 2E 20 45 78 70 72 | ness.level. Expr
002710: 65 73 73 69 6E 67 20 6C  6F 76 65 20 66 6F 72 20 | essing love for
002720: 6F 74 68 65 72 73 20 62  65 6E 65 66 69 74 73 20 | others benefits
002730: 6E 6F 74 20 6A 75 73 74  20 74 68 65 20 72 65 63 | not just the rec
002740: 69 70 69 65 6E 74 20 6F  66 0A 61 66 66 65 63 74 | ipient of.affect
002750: 69 6F 6E 2C 20 62 75 74  20 61 6C 73 6F 20 74 68 | ion, but also th
002760: 65 20 70 65 72 73 6F 6E  20 77 68 6F 20 64 65 6C | e person who del
002770: 69 76 65 72 73 20 69 74  2E 20 54 68 65 20 6E 65 | ivers it. The ne
002780: 65 64 20 74 6F 20 62 65  20 6C 6F 76 65 64 20 63 | ed to be loved c
002790: 61 6E 0A 62 65 20 63 6F  6E 73 69 64 65 72 65 64 | an.be considered
0027A0: 20 61 73 20 6F 6E 65 20  6F 66 20 6F 75 72 20 6D |  as one of our m
0027B0: 6F 73 74 20 62 61 73 69  63 20 61 6E 64 20 66 75 | ost basic and fu
0027C0: 6E 64 61 6D 65 6E 74 61  6C 20 6E 65 65 64 73 2E | ndamental needs.
0027D0: 0A 4F 6E 65 20 6F 66 20  74 68 65 20 66 6F 72 6D | .One of the form
0027E0: 73 20 74 68 61 74 20 74  68 69 73 20 6E 65 65 64 | s that this need
002800: 6E 74 61 63 74 20 63 6F  6D 66 6F 72 74 2E 20 49 | ntact comfort. I
002810: 74 20 69 73 20 74 68 65  20 64 65 73 69 72 65 0A | t is the desire.
002820: 74 6F 20 62 65 20 68 65  6C 64 20 61 6E 64 20 74 | to be held and t
002830: 6F 75 63 68 65 64 2E 20  53 6F 2C 20 74 68 65 72 | ouched. So, ther
002840: 65 20 61 72 65 20 6D 61  6E 79 20 65 78 70 65 72 | e are many exper
002850: 69 6D 65 6E 74 73 20 73  68 6F 77 69 6E 67 20 74 | iments showing t
002860: 68 61 74 0A 62 61 62 69  65 73 20 77 68 6F 20 61 | hat.babies who a
002870: 72 65 20 6E 6F 74 20 68  61 76 69 6E 67 20 63 6F | re not having co
002880: 6E 74 61 63 74 20 63 6F  6D 66 6F 72 74 2C 20 65 | ntact comfort, e
002890: 73 70 65 63 69 61 6C 6C  79 20 64 75 72 69 6E 67 | specially during
0028A0: 20 74 68 65 20 66 69 72  73 74 20 73 69 78 0A 6D |  the first six.m
0028B0: 6F 6E 74 68 73 2C 20 67  72 6F 77 20 75 70 20 74 | onths, grow up t
0028C0: 6F 20 62 65 20 70 73 79  63 68 6F 6C 6F 67 69 63 | o be psychologic
0028D0: 61 6C 6C 79 20 64 61 6D  61 67 65 64 2E 0A 74 00 | ally damaged..t.
0028E0: 62 2E 74 78 74 00 00 00  00 00 00 00 00 00 00 00 | b.txt...........
0028F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

002900: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002910: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002920: 00 14 00 00 DC 04 00 00  0A 00 00 00 11 00 00 00 | ....◆...........
002930: 00 00 00 00 00 00 00 00  65 5C A9 61 00 00 00 00 | ........e\◆a....
002940: 65 5C A9 61 00 00 00 00  65 5C A9 61 00 00 00 00 | e\◆a....e\◆a....
002950: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002960: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002970: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002980: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002990: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0029A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0029B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0029C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0029D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0029E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0029F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

cp2l - terminal

```
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$ make run
./fsshell 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
Prompt > cp2fs a.txt b.txt
Prompt > cp2l b.txt c.txt
Prompt > exit
System exiting
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$
```

```
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$ cat a.txt
Love is a set of emotions, behaviors, and beliefs with strong feelings of
affection. So, for example, a person might say he or she loves his or her
dog, loves freedom, or loves God. The concept of love may become an
unimaginable thing and also it may happen to each person in a particular
way.
Love has a variety of feelings, emotions, and attitude. For someone love
is more than just being interested physically in another one, rather it is an
emotional attachment. We can say love is more of a feeling that a person
feels for another person. Therefore, the basic meaning of love is to feel
more than liking towards someone.
We know that the desire to love and care for others is a hard-wired and
deep-hearted because the fulfillment of this wish increases the happiness
level. Expressing love for others benefits not just the recipient of
affection, but also the person who delivers it. The need to be loved can
be considered as one of our most basic and fundamental needs.
One of the forms that this need can take is contact comfort. It is the desire
to be held and touched. So, there are many experiments showing that
babies who are not having contact comfort, especially during the first six
months, grow up to be psychologically damaged.
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$ cat c.txt
Love is a set of emotions, behaviors, and beliefs with strong feelings of
affection. So, for example, a person might say he or she loves his or her
dog, loves freedom, or loves God. The concept of love may become an
unimaginable thing and also it may happen to each person in a particular
way.
Love has a variety of feelings, emotions, and attitude. For someone love
is more than just being interested physically in another one, rather it is an
emotional attachment. We can say love is more of a feeling that a person
feels for another person. Therefore, the basic meaning of love is to feel
more than liking towards someone.
We know that the desire to love and care for others is a hard-wired and
deep-hearted because the fulfillment of this wish increases the happiness
level. Expressing love for others benefits not just the recipient of
affection, but also the person who delivers it. The need to be loved can
be considered as one of our most basic and fundamental needs.
One of the forms that this need can take is contact comfort. It is the desire
to be held and touched. So, there are many experiments showing that
babies who are not having contact comfort, especially during the first six
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$
```

cp - terminal

```
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$ 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
Prompt > cp b.txt c.txt
Prompt > exit
System exiting
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$
```

cp - Hexdump

```
tunni@ubuntu:~/Documents/CSC415_Assignments/csc415-filesystem-jiasheng-li$ Hexdump/hexdump.linuxM1 --count 3 --start 29 SampleVolume
Dumping file SampleVolume, starting at block 29 for 3 blocks:

003A00: 00 70 65 72 73 6F 6E 0A  66 65 65 6C 73 20 66 6F | .person.feels fo
003A10: 72 20 61 6E 6F 74 68 65  72 20 70 65 72 73 6F 6E | r another person
003A20: 2E 20 54 68 65 72 65 66  6F 72 65 2C 20 74 68 65 | . Therefore, the
003A30: 20 62 61 73 69 63 20 6D  65 61 6E 69 6E 67 20 6F |  basic meaning o
003A40: 66 20 6C 6F 76 65 20 69  73 20 74 6F 20 66 65 65 | f love is to fee
003A50: 6C 0A 6D 6F 72 65 20 74  68 61 6E 20 6C 69 6B 69 | l.more than liki
003A60: 6E 67 20 74 6F 77 61 72  64 73 20 73 6F 6D 65 6F | ng towards someo
003A70: 6E 65 2E 0A 57 65 20 6B  6E 6F 77 20 74 68 61 74 | ne..We know that
003A80: 20 74 68 65 20 64 65 73  69 72 65 20 74 6F 20 6C |  the desire to l
003A90: 6F 76 65 20 61 6E 64 20  63 61 72 65 20 66 6F 72 | ove and care for
003AA0: 20 6F 74 68 65 72 73 20  69 73 20 61 20 68 61 72 |  others is a har
003AB0: 64 2D 77 69 72 65 64 20  61 6E 64 0A 64 65 65 70 | d-wired and.deep
003AC0: 2D 68 65 61 72 74 65 64  20 62 65 63 61 75 73 65 | -hearted because
003AD0: 20 74 68 65 20 66 75 6C  66 69 6C 6C 6D 65 6E 74 |  the fulfillment
003AE0: 20 6F 66 20 74 68 69 73  20 77 69 73 68 20 69 6E |  of this wish in
003AF0: 63 72 65 61 73 65 73 20  74 68 65 20 68 61 70 70 | creases the happ

003B00: 69 6E 65 73 73 0A 6C 65  76 65 6C 2E 20 45 78 70 | iness.level. Exp
003B10: 72 65 73 73 69 6E 67 20  6C 6F 76 65 20 66 6F 72 | ressing love for
003B20: 20 6F 74 68 65 72 73 20  62 65 6E 65 66 69 74 73 |  others benefits
003B30: 20 6E 6F 74 20 6A 75 73  74 20 74 68 65 20 72 65 |  not just the re
003B40: 63 69 70 69 65 6E 74 20  6F 66 0A 61 66 66 65 63 | cipient of.affec
003B50: 74 69 6F 6E 2C 20 62 75  74 20 61 6C 73 6F 20 74 | tion, but also t
003B60: 68 65 20 70 65 72 73 6F  6E 20 77 68 6F 20 64 65 | he person who de
003B70: 6C 69 76 65 72 73 20 69  74 2E 20 54 68 65 20 6E | livers it. The n
003B80: 65 65 64 20 74 6F 20 62  65 20 6C 6F 76 65 64 20 | eed to be loved
003B90: 63 61 6E 0A 62 65 20 63  6F 6E 73 69 64 65 72 65 | can.be considere
003BA0: 64 20 61 73 20 6F 6E 65  20 6F 66 20 6F 75 72 20 | d as one of our
003BB0: 6D 6F 73 74 20 62 61 73  69 63 20 61 6E 64 20 66 | most basic and f
003BC0: 75 6E 64 61 6D 65 6E 74  61 6C 20 6E 65 65 64 73 | undamental needs
003BD0: 2E 0A 4F 6E 65 20 6F 66  20 74 68 65 20 66 6F 72 | ..One of the for
003BE0: 6D 73 20 74 68 61 74 20  74 68 69 73 20 6E 65 65 | ms that this nee
003BF0: 64 20 63 61 6E 20 74 61  6B 65 20 69 73 20 63 00 | d can take is c.
```

```
003C00: 6F 00 6E 74 61 63 74 20  63 6F 6D 66 6F 72 74 2E | o.ntact comfort.
003C10: 20 49 74 20 69 73 20 74  68 65 20 64 65 73 69 72 |  It is the desir
003C20: 65 0A 74 6F 20 62 65 20  68 65 6C 64 20 61 6E 64 | e.to be held and
003C30: 20 74 6F 75 63 68 65 64  2E 20 53 6F 2C 20 74 68 |  touched. So, th
003C40: 65 72 65 20 61 72 65 20  6D 61 6E 79 20 65 78 70 | ere are many exp
003C50: 65 72 69 6D 65 6E 74 73  20 73 68 6F 77 69 6E 67 | eriments showing
003C60: 20 74 68 61 74 0A 62 61  62 69 65 73 20 77 68 6F |  that.babies who
003C70: 20 61 72 65 20 6E 6F 74  20 68 61 76 69 6E 67 20 |  are not having
003C80: 63 6F 6E 74 61 63 74 20  63 6F 6D 66 6F 72 74 2C | contact comfort,
003C90: 20 65 73 70 65 63 69 61  6C 6C 79 20 64 75 72 69 |  especially duri
003CA0: 6E 67 20 74 68 65 20 66  69 72 73 74 20 73 69 78 | ng the first six
003CB0: 0A 6D 6F 6E 74 68 73 2C  20 67 72 6F 77 20 75 70 | .months, grow up
003CC0: 20 74 6F 20 62 65 20 70  73 79 63 68 6F 6C 6F 67 |  to be psycholog
003CD0: 69 63 61 6C 6C 79 20 64  61 6D 61 67 65 64 6E 00 | ically damagedn.
003CE0: 62 2E 74 78 74 00 00 00  00 00 00 00 00 00 00 00 | b.txt...........
003CF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

003D00: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003D10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003D20: 00 14 00 00 DC 04 00 00  0A 00 00 00 11 00 00 00 | ....▯...........
003D30: 00 00 00 00 00 00 00 00  85 B4 A9 61 00 00 00 00 | ........▯▯▯a....
003D40: 85 B4 A9 61 00 00 00 00  85 B4 A9 61 00 00 00 00 | ▯▯▯a....▯▯▯a....
003D50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003D60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003D70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003D80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003D90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003DA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003DB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003DC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003DD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003DE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003DF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

```
003E00: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003E10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003E20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003E30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003E40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003E50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003E60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003E70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003E80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003E90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003EA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003EB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003EC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003ED0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003EE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003EF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

003F00: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003F10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003F20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003F30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003F40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003F50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003F60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003F70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003F80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003F90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003FA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003FB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003FC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003FD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003FE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
003FF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

Planning Process (The steps that we planned for implementation)

fsDir.h

[Global Variables]

- mfs_DefaultDECount = 50;

[Functions]

- int createDir(int parentLocation);
    - Returning value: -1 means failed, positive means success
- bool pathParser(char* path, unsigned char condition, DE*tempWorkingDir);
    - Returning value: 0 means failed, 1 means success
- bool allocateDirectory(DE *directory)
    - Returning 1 on success, 0 on failed
- void printDEInfo
-

fsFree.h

[Global Variables]

- unsigned char *bitmap;
- unsigned char setArray[8] = {0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01};
- unsigned char clearArray[8] = {0x7F, 0xBF, 0xDF, 0xEF, 0xF7, 0xFB, 0xFD, 0xFE};

[Functions]

- int initFreeSpace(VCB* vcb);
    - Return 1 on success, -1 on failed
- int reloadFreeSpace();
    - Return 1 on success, -1 on failed
- int allocateBlocks(int blockCount);
    - Return index of LBA, -1 on failed
- int releaseFreeSpace();
    - Return 1 on success, -1 on failed

mfs.h

[Global Variables]

- int mfs_cwd_location;
- int mfs_blockSize;

[DE structure]

- char name[64]
- int size
- int actualSize
- int blockCount
- int location
- bool isDir
- time_t createTime
- time_t lastModTime
- time_t lastAccessTime

[Functions]

- int fs_mkdir(const char *pathname, mode_t mode);
    - Condition: NOT_EXIST
- int fs_rmdir(const char *pathname);
    - Condition: EXIST_DIR
- fdDir *fs_opendir(const char *name);
    - Condition: EXIST_DIR
- struct fs_diriteminfo *fs_readdir(fdDir *dirp);
- int fs_closedir(fdDir *dirp);
- char *fs_getcwd(char *buf, size_t size);
- int fs_setcwd(char *buf);
    - Return 0 means success
- int fs_isFile(char *path);
    - Condition: EXIST_FILE
- int fs_isDir(char *path);
    - Condition: EXIST_DIR
- int fs_delete(char *filename);
    - Condition: EXIST_DIR | EXIST_FILE
- int fs_stat(const char *path, struct fs_stat *buf);
    - Condition: EXIST_DIR

b_io.h

[Global Variable]

[Functions]

- b_io_fd b_open(char *filename, int flags);
    - Returns fd (index of fcb), or -1 when failed
- int b_read(b_io_fd fd, char *buffer, int count);

    - Returns the number of bytes read, or -1 when failed
- int b_write(b_io_fd fd, char *buffer, int count);
    - Returns the number bytes written, or -1 when failed
- int b_seek(b_io_fd fd, off_t offset, int whence);
    - Returns new offset, or -1 when failed
- void b_close(b_io_fd fd);
    - Return 0 on success, or -1 on failed

fsDir.c

---

int createDir(int parentLocation);

1. Malloc space for directory
    a. We will have to determine how much size we need to allocate for DefaultDECount
    b. Turn the size in unit of block
    c. Update the size that we will actually malloc for
    d. Since the block size might can contained more DE, we have to update the count
    e. Malloc with mallocSize
2. Loop through directory and init each DE fields of each DE
3. Allocate blocks from our free space
4. Set first DE as itself
5. Set second DE as its parent (if parent is NULL, set to itself)
6. Write into disk

bool pathParser(char* path, unsigned char condition, DE*tempWorkingDir);

1. Check if path is relative or absolute, so we can set the tempWorkingDir correctly
    a. If is relative, then tempWorkingDir will be mfs_cwd
2. Tokenize the path and store in tokens (also count the number of token)
3. Find each token (traverse the directories and update the tempWorkingDir), except for the last one
4. Check condition for last token and store its status in thisCondition
5. Free allocated elements
6. Compare thisCondition and given condition to check validation

int fs_mkdir(const char *pathname, mode_t mode)

1. Convert path to char array (to avoid the const warning)
2. Malloc space for tempWorkingDir and last token
3. Check validation by calling pathParser()

4. If invalid, return -1 and print error message
5. If valid, get the directory index of tempWorkingDir (if index = -1, means directory full. Print msg and return -1)
6. Create directory for lastToken
7. Set the DE to the index
8. Write into disk
9. Free tempworkingDir and last token
10. Return 1

int fs_rmdir(const char *pathname);

1. Convert path to char array (to avoid the const warning)
2. Malloc space for tempWorkingDir and last token
3. Check validation by calling pathParser()
4. If invalid, return -1 and print error message
5. If valid, check if the tempWorkingDir is all empty except for index 0 and index 1
6. If not, print error message and return -1
7. If yes, call fs_delete(), pass in lastToken
8. Check return value
9. Free tempWorkingDir and last token
10. Return 1

fdDir *fs_opendir(const char *name);

1. Convert path to char array (to avoid the const warning)
2. Malloc space for tempWorkingDir and last token
3. Check validation by calling pathParser()
4. If invalid, return -1 and print error message
5. If valid, loop through tempWorkingDir and find the DE
6. Malloc space for fdDir struct
7. Fill in the elements
8. Return fdDir

struct fs_diriteminfo *fs_readdir(fdDir *dirp);

1. Check if dirp is NULL
2. If not, malloc space for fs_diriteminfo
3. Fill in elements
4. Return diriteminfo

int fs_closedir(fdDir *dirp); https://man7.org/linux/man-pages/man3/closedir.3.html

1. Check if dirp is NULL
2. If not, free it
3. Return 1

char *fs_getcwd(char *buf, size_t size); https://man7.org/linux/man-pages/man3/getcwd.3.html

1. Malloc space for char path and a token array and tokenCount
2. Loop through the cwd and copy each DE name
    a. Copy each name into token array
    b. Also keep track of tokenCount
3. Loop through the token array from last to first
    a. Copy it into char path with a "/" at the front
4. Copy path into buf
5. Free token array
6. Return path

int fs_setcwd(char *buf);

1. Malloc space for tempWorkingDir and last token
2. Check validation by calling pathParser()
3. If invalid, return -1 and print error message
4. If valid, loop through tempWorkingDir to find the last token.
5. Set mfs_cwd to the location of last token
6. Return

int fs_isFile(char *path);

1. Convert path to char array (to avoid the const warning)
2. Malloc space for tempWorkingDir and last token
3. Check validation by calling pathParser()
4. If invalid, return -1 and print error message
5. If valid, loop through the tempWorking to find lastToken DE
6. Return !isDir

int fs_isDir(char *path);

1. Convert path to char array (to avoid the const warning)
2. Malloc space for tempWorkingDir and last token
3. Check validation by calling pathParser()

4. If invalid, return -1 and print error message
5. If valid, loop through the tempWorking to find lastToken DE
6. Return isDir

int fs_delete(char *filename);

1. Convert path to char array (to avoid the const warning)
2. Malloc space for tempWorkingDir and last token
3. Check validation by calling pathParser()
4. If invalid, return -1 and print error message
5. If valid, find the DE in the tempWorkingDir
6. Set the bit to be free
7. Set the DE to not in use
8. Return

int fs_stat(const char *path, struct fs_stat *buf);

1. Convert path to char array (to avoid the const warning)
2. Malloc space for tempWorkingDir and last token
3. Check validation by calling pathParser()
4. If invalid, return -1 and print error message
5. If valid, loop through tempWorkingDir and fill in the buf's field
6. Return 1

Milestone 3

---

b_fcb structure

- DE *fi
- char *buf
- int buflen //size of our buf
- int bufIndex //current index of our buf
- int currentBlock
- int accessMode
- int currentFileIndex

b_io_fd b_open(char *filename, int flags)

- Return value: index of the file descriptor for this file
- Flags: O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_TRUNC, O_APPEND
- Malloc empty space, empty space, LBA write

1. Malloc space for *tempWorkingDir* and *lastToken(filename)*
2. Use pathparser and check the validation
    a. If invalid, check if the flag has O_CREAT. If yes, **create a new file**.
    b. Allocate the default amount of blocks for the new file -> so will get the *LBAindex*
    c. Find the next DE index in the tempWorkingDir -> so will get the *DEindex*
    d. Fill in DE info at the DE index in the temWorkingDir
        i. char name[64] -> lastToken (file name)
        ii. int size -> default
        iii. int actualSize -> 0
        iv. int blockCount -> default amount of blocks
        v. int location -> LBAindex
        vi. bool isDir -> false
        vii. time_t createTime
        viii. time_t lastModTime
        ix. time_t lastAccessTime
    e. Malloc buffer for file
    f. LBA write buffer into disk -> default amount of blocks at LBAindex
    g. LBA write tempWorkingDir to disk -> default amount of blocks at tempWorkingDir[0].location

    ---------- we have a existing file, tempWorkingDir & lastToken
3. Get fcb index -> FCBindex

4. init fcb at FCBindex as no other flags
    a. DE *fi -> loop through the tempWorkingDir to find the file
    b. char *buf -> malloc the chunk size space for buf
    c. ~~int fileRemains: for counting how many byte has left in the file -> actual size of the file~~
    d. ~~int bufferRemains: for counting how many byte has left in the myBuffer -> chunk size~~
    e. int ~~currentBufferRead~~ bufIndex: record the current index of the buffer -> 0
    f. int currentBlock -> fi.location
    g. int accessMode -> flags
5. Update info based on flags
    a. If O_TRUNC, check if the file has write access, if yes then clear the data and set the related fields to zero (size, current position, etc.) and update time fields (do LBAwrite)
        i. Create a same size of empty buffer
        ii. LBAwrite it to the disk for fi.blockCount at fi.location
        iii. Set the fi.actualSize to 0
        iv. Set the fcb.fileRemain to 0
        v. Set the fcb.currentBufferRead = 0
        vi. Set the bufferRemains to chunk size
    b. If O_APPEND, use b_seek, to set the offset to the end of the file
        i. b_seek(FCBindex, fcb.currentRead, SEEK_END)
6. Free tempWorkingDir, lastToken, buf
7. Return the index number

int b_read(b_io_fd fd, char *buffer, int count)

- Return the 0 when reach the EOF, otherwise the number of bytes that we read

1. Check if the access mode has read, if no, print error message and return failed
2. Calculate the bufferRemainingBytes
3. Calculate how many bytes have been processed to check EOF
4. Update the count if user is asking more than we can process
5. Safety check if count is smaller than zero
6. Calculate part 1, 2, and 3
    a. Case 1: count <= bufferRemainingBytes
        i. Part 1 = count
        ii. Part 2 = 0
        iii. Part 3 = 0
    b. Else cases
        i. Part 1 = bufferRemainingBytes
        ii. Part 3 = count - bufferRemainingBytes

iii. transferBlocks = part 3 / chunk size
iv. Part 2 = transferBlocks * chunk size
v. Part 3 = part 3 - part 2

7. Check if Part 1 is larger than zero
    a. Copy data from our buffer to user's buffer
    b. Update the buffer index
8. Check if part 2 is larger than zero
    a. Read the blocks directory from disk (current block + file location) to user buffer at index part 1 for transferBlocks
    b. Update the current block
    c. Update the number of part 2 to byteRead
9. Check if part 3 is larger than zero
    a. Reload our buffer
    b. Update the current block
    c. Update the buffer index to 0
    d. Set the buffer length to bytes that we read
    e. Check if bytes that we read is smaller than part 3, if yes set part 3 = bytesRead
    f. Then double check if part 3 > 0
    g. If yes, then memcpy the bytes in our buf to user's buffer at part1+part2 location, with part 3 bytes
    h. Update index
10. Calculate returned bytes
11. Return bytes

int b_write(b_io_fd fd, char *buffer, int count)

1. Check if the access mode has write, if no, print error message and return failed
2. Check if we have enough space for count, if not, then update the count (after this step, we have enough space to write into the file)
    a. If fileActualSize + count > fileMaxSize, then reallocate blocks for write
        i. Get new LBA location with passing in blockCount + 5
        ii. Create an empty buffer with size of new blockCount * block size
        iii. LBAread the old content to the bigger buffer
        iv. Update the fcb information
            1. Declare tempWorkingDir and lastToken
            2. Use pathParser
            3. Find the DE that has the same name as lastToken in tempWorkingDir -> DEindex

4. tempWorkingDir[DEindex].location -> new location
5. tempWorkingDir[DEindex].size -> original blockCount + 5 * blockSize
6. tempWorkingDir[DEindex].blockCount += 5
7. fi.location -> new location
8. fi.size -> new size
9. fi.blockCount -> new blockCount
10. fcb.currentBlock -> increment by the difference between new blockCount and old blockCount

v. LBAwrite the tempWorkingDir for tempWorkingDir[0].blockCount to tempWorkingDir[0].location

vi. Free tempWorkingDir and lastToken and buffer

3. Calculate the bufferRemainingBytes = chunk size - bufIndex
4. Calculate part 1, 2, and 3
    a. Case 1: count <= bufferRemainingBytes
        i. Part 1 = count
        ii. Part 2 = 0
        iii. Part 3 = 0
    b. Else cases
        i. Part 1 = bufferRemainingBytes
        ii. Part 3 = count - bufferRemainingBytes
        iii. transferBlocks = part 3 / chunk size
        iv. Part 2 = transferBlocks * chunk size
        v. Part 3 = part 3 - part 2
5. Part 1
    a. Copy the user given buffer to our buffer
    b. Update the index
    c. If the bufRemain = 0, write to Disk and update block count, index
6. Part 2
    a. LBAwrite the user given data at part 1 index to the disk
    b. LBAwrite(buffer + part 1, current block + location, transferBlocks)
    c. Update the current block
    d. Update the part 2 to byteWrite
7. Part 3
    a. Copy the user given buffer to our buffer
    b. Update index
    c. If the bufRemain = 0, write to Disk and update block count, index
8. Calculate the returned byte

9. Return byte

int b_seek(b_io_fd fd, off_t offset, int whence)

- Repositions the file offset of the open file description
- Returns new offset, or -1 when there's an error
- Whence: option for offset()
    - SEEK_SET: fd = offset (0x00) (The file offset is set to offset bytes.)
    - SEEK_CUR: fd += offset (0x01) (The file offset is set to its current location plus offset bytes.)
    - SEEK_END: fd = (start block position + file size / 512) + offset (if positive then error) (0x02) (The file offset is set to the size of the file plus offset bytes.)

void b_close(b_io_fd fd)

1. Release any acclocation space (buf)
2. Set fi of the fd index in fcbArray to null

mv commands

- Four situation
    - Renaming a file (existing filename, non-existing filename)
    - Renaming a directory (existing directory, non-existing directory)
        - Loop through the DE to find first argument then change the name
    - Moving a file (existing filename, existing directory)
    - Moving a directory (existing directory, existing directory)
        - Loop through the DE to find first argument
        - Loop through the DE to find second argument and get the location
        - Set the location to the new location

1. Check if the argcnt is more than three (command, arg1, arg2), if not, print usage and return -1
2. Create two tempWorkingDir(1, 2) and two lastToken(1, 2)
3. Check first argument with EXIST_DIR | EXIST_FILE condition, if invalid, print error message and return -1
4. Check second argument with EXIST_DIR | EXIST_FILE condition
    a. If invalid -> rename the file
        i. Loop through the tempWorkingDir1 to find lastToken1
        ii. Set the DE.name to lastToken2

        iii. LBAwrite the tempWorkingDir1
    b. If valid -> change the location
        i. Loop through the tempWorkingDir1 to find lastToken1
        ii. Loop through the tempWorkingDir2 to find lastToken2
        iii. LBAread the lastToken2 DE array to tempWorkingDir2
        iv. Loop through tempWorkingDir2 to find new DEindex
        v. Create a temp DE pointer to store lastToken1 info
        vi. Set the DE in tempWorkingDir1 to NULL
        vii. Set the tempWorkingDir2[DEindex] to temp DE
        viii. LBAwrite tempWorkingDir1
        ix. LBAwrite tempWorkingDir2
        x. Free temp DE
5. Free tempWorkingDir1&2 and lastToken1&2

bool pathParser(char *path, unsigned char condition, DE *tempWorkingDir, char *lastToken);

- Return 0 when failed, 1 when success

1. Malloc space for tempWorkingDir and lastToken
    a. allocateDirectory();
    b. Allocate 256 bytes for lastToken
    c. Check if malloc successfully
2. Check if the path is relative or absolute (use '/')
    a. Relative: read the blocks at mfs_currentLocation to the tempWorkingDir
    b. Absolute: read the root to the tempworkingDir
3. Tokenizing the path
    a. Declare tokenCount, token array, theRest, token
    b. Add token to the token array and increment the tokenCount by 1
4. Update the tempWorkingDir to the second last token (nested for loop)
    a. Declare bool found as a flag
    b. If found token DE in tempWorkingDir, read that into the tempWorkingDir, set found to true, then break this loop
    c. After running through the inner loop, check if found. If yes, reset found to 0. If not, print err message and return 0.
5. Get lastToken from the token array if the array size is not 0
6. Loop through the tempWorkingDir to see if it is an existed dir, file or non existing (special case, "/" which array size would be 0, should returned EXIST_DIR immediately if we are looking at root)
7. Compare the condition and given condition, then return validation

Helper Functions

bool allocateDirectory();

1. Calculate the malloc size
2. Declare and allocate the return value
3. Return the malloc failed or success

```
void printDEInfo(DE de)
{
  printf("--- DE info ---\n");
  printf("- name: %s\n- size: %d\n- pointingLocation: %d\n", de.name, de.size, de.location);
  printf("- isDir: %d\n- createTime: %s\n- lastMod: %s\n- lastAccess: %s\n", de.isDir,
ctime(&de.createTime), ctime(&de.lastModTime), ctime(&de.lastAccessTime));
}
```

fsDir.h

[Global Variables]

- mfs_DefaultDECount = 50;

[Functions]

- int createDir(int parentLocation);
  - Returning value: -1 means failed, positive means success
- bool pathParser(char* path, unsigned char condition, DE*tempWorkingDir);
  - Returning value: 0 means failed, 1 means success
- bool allocateDirectory(DE *directory)
  - Returning 1 on success, 0 on failed
- void printDEInfo
-

fsFree.h

[Global Variables]

- unsigned char *bitmap;
- unsigned char setArray[8] = {0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01};
- unsigned char clearArray[8] = {0x7F, 0xBF, 0xDF, 0xEF, 0xF7, 0xFB, 0xFD, 0xFE};

[Functions]

- int initFreeSpace(VCB* vcb);
  - Return 1 on success, -1 on failed
- int reloadFreeSpace();
  - Return 1 on success, -1 on failed

- int allocateBlocks(int blockCount);
    - Return index of LBA, -1 on failed
- int releaseFreeSpace();
    - Return 1 on success, -1 on failed

mfs.h

[Global Variables]

- int mfs_cwd_location;
- int mfs_blockSize;

[DE structure]

- char name[64]
- int size
- int actualSize
- int blockCount
- int location
- bool isDir
- time_t createTime
- time_t lastModTime
- time_t lastAccessTime

[Functions]

- int fs_mkdir(const char *pathname, mode_t mode);
    - Condition: NOT_EXIST
- int fs_rmdir(const char *pathname);
    - Condition: EXIST_DIR
- fdDir *fs_opendir(const char *name);
    - Condition: EXIST_DIR
- struct fs_diriteminfo *fs_readdir(fdDir *dirp);
- int fs_closedir(fdDir *dirp);
- char *fs_getcwd(char *buf, size_t size);
- int fs_setcwd(char *buf);
    - Return 0 means success
- int fs_isFile(char *path);
    - Condition: EXIST_FILE
- int fs_isDir(char *path);
    - Condition: EXIST_DIR
- int fs_delete(char *filename);

    - Condition: EXIST_DIR | EXIST_FILE
- int fs_stat(const char *path, struct fs_stat *buf);
    - Condition: EXIST_DIR

b_io.h

[Global Variable]

[Functions]

- b_io_fd b_open(char *filename, int flags);
    - Returns fd (index of fcb), or -1 when failed
- int b_read(b_io_fd fd, char *buffer, int count);
    - Returns the number of bytes read, or -1 when failed
- int b_write(b_io_fd fd, char *buffer, int count);
    - Returns the number bytes written, or -1 when failed
- int b_seek(b_io_fd fd, off_t offset, int whence);
    - Returns new offset, or -1 when failed
- void b_close(b_io_fd fd);
    - Return 0 on success, or -1 on failed

fsDir.c

---

int createDir(int parentLocation);

1. Malloc space for directory
    a. We will have to determine how much size we need to allocate for DefaultDECount
    b. Turn the size in unit of block
    c. Update the size that we will actually malloc for
    d. Since the block size might can contained more DE, we have to update the count
    e. Malloc with mallocSize
2. Loop through directory and init each DE fields of each DE
3. Allocate blocks from our free space
4. Set first DE as itself
5. Set second DE as its parent (if parent is NULL, set to itself)
6. Write into disk

bool pathParser(char* path, unsigned char condition, DE*tempWorkingDir);

1. Check if path is relative or absolute, so we can set the tempWorkingDir correctly
    a. If is relative, then tempWorkingDir will be mfs_cwd
2. Tokenize the path and store in tokens (also count the number of token)

3. Find each token (traverse the directories and update the tempWorkingDir), except for the last one
4. Check condition for last token and store its status in thisCondition
5. Free allocated elements
6. Compare thisCondition and given condition to check validation

int fs_mkdir(const char *pathname, mode_t mode)

1. Convert path to char array (to avoid the const warning)
2. Malloc space for tempWorkingDir and last token
3. Check validation by calling pathParser()
4. If invalid, return -1 and print error message
5. If valid, get the directory index of tempWorkingDir (if index = -1, means directory full. Print msg and return -1)
6. Create directory for lastToken
7. Set the DE to the index
8. Write into disk
9. Free tempworkingDir and last token
10. Return 1

int fs_rmdir(const char *pathname);

1. Convert path to char array (to avoid the const warning)
2. Malloc space for tempWorkingDir and last token
3. Check validation by calling pathParser()
4. If invalid, return -1 and print error message
5. If valid, check if the tempWorkingDir is all empty except for index 0 and index 1
6. If not, print error message and return -1
7. If yes, call fs_delete(), pass in lastToken
8. Check return value
9. Free tempWorkingDir and last token
10. Return 1

fdDir *fs_opendir(const char *name);

1. Convert path to char array (to avoid the const warning)
2. Malloc space for tempWorkingDir and last token
3. Check validation by calling pathParser()
4. If invalid, return -1 and print error message

5. If valid, loop through tempWorkingDir and find the DE
6. Malloc space for fdDir struct
7. Fill in the elements
8. Return fdDir

struct fs_diriteminfo *fs_readdir(fdDir *dirp);

1. Check if dirp is NULL
2. If not, malloc space for fs_diriteminfo
3. Fill in elements
4. Return diriteminfo

int fs_closedir(fdDir *dirp); https://man7.org/linux/man-pages/man3/closedir.3.html

1. Check if dirp is NULL
2. If not, free it
3. Return 1

char *fs_getcwd(char *buf, size_t size); https://man7.org/linux/man-pages/man3/getcwd.3.html

1. Malloc space for char path and a token array and tokenCount
2. Loop through the cwd and copy each DE name
    a. Copy each name into token array
    b. Also keep track of tokenCount
3. Loop through the token array from last to first
    a. Copy it into char path with a "/" at the front
4. Copy path into buf
5. Free token array
6. Return path

int fs_setcwd(char *buf);

1. Malloc space for tempWorkingDir and last token
2. Check validation by calling pathParser()
3. If invalid, return -1 and print error message
4. If valid, loop through tempWorkingDir to find the last token.
5. Set mfs_cwd to the location of last token
6. Return

int fs_isFile(char *path);

1. Convert path to char array (to avoid the const warning)

2. Malloc space for tempWorkingDir and last token
3. Check validation by calling pathParser()
4. If invalid, return -1 and print error message
5. If valid, loop through the tempWorking to find lastToken DE
6. Return !isDir

int fs_isDir(char *path);

1. Convert path to char array (to avoid the const warning)
2. Malloc space for tempWorkingDir and last token
3. Check validation by calling pathParser()
4. If invalid, return -1 and print error message
5. If valid, loop through the tempWorking to find lastToken DE
6. Return isDir

int fs_delete(char *filename);

1. Convert path to char array (to avoid the const warning)
2. Malloc space for tempWorkingDir and last token
3. Check validation by calling pathParser()
4. If invalid, return -1 and print error message
5. If valid, find the DE in the tempWorkingDir
6. Set the bit to be free
7. Set the DE to not in use
8. Return

int fs_stat(const char *path, struct fs_stat *buf);

1. Convert path to char array (to avoid the const warning)
2. Malloc space for tempWorkingDir and last token
3. Check validation by calling pathParser()
4. If invalid, return -1 and print error message
5. If valid, loop through tempWorkingDir and fill in the buf's field
6. Return 1

Milestone 3

---

b_fcb structure

- DE *fi
- char *buf
- int buflen //size of our buf

- int bufIndex //current index of our buf
- int currentBlock
- int accessMode
- int currentFileIndex

b_io_fd b_open(char *filename, int flags)

- Return value: index of the file descriptor for this file
- Flags: O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_TRUNC, O_APPEND
- Malloc empty space, empty space, LBA write

1. Malloc space for *tempWorkingDir* and *lastToken(filename)*
2. Use pathparser and check the validation
    a. If invalid, check if the flag has O_CREAT. If yes, **create a new file**.
    b. Allocate the default amount of blocks for the new file -> so will get the *LBAindex*
    c. Find the next DE index in the tempWorkingDir -> so will get the *DEindex*
    d. Fill in DE info at the DE index in the temWorkingDir
        i. char name[64] -> lastToken (file name)
        ii. int size -> default
        iii. int actualSize -> 0
        iv. int blockCount -> default amount of blocks
        v. int location -> LBAindex
        vi. bool isDir -> false
        vii. time_t createTime
        viii. time_t lastModTime
        ix. time_t lastAccessTime
    e. Malloc buffer for file
    f. LBA write buffer into disk -> default amount of blocks at LBAindex
    g. LBA write tempWorkingDir to disk -> default amount of blocks at tempWorkingDir[0].location

    ---------- we have a existing file, tempWorkingDir & lastToken
3. Get fcb index -> FCBindex
4. init fcb at FCBindex as no other flags
    a. DE *fi -> loop through the tempWorkingDir to find the file
    b. char *buf -> malloc the chunk size space for buf
    ~~c. int fileRemains: for counting how many byte has left in the file -> actual size of the file~~
    ~~d. int bufferRemains: for counting how many byte has left in the myBuffer -> chunk size~~
    e. int ~~currentBufferRead~~ bufIndex: record the current index of the buffer -> 0

    f. int currentBlock -> fi.location
    g. int accessMode -> flags
5. Update info based on flags
    a. If O_TRUNC, check if the file has write access, if yes then clear the data and set the related fields to zero (size, current position, etc.) and update time fields (do LBAwrite)
        i. Create a same size of empty buffer
        ii. LBAwrite it to the disk for fi.blockCount at fi.location
        iii. Set the fi.actualSize to 0
        iv. Set the fcb.fileRemain to 0
        v. Set the fcb.currentBufferRead = 0
        vi. Set the bufferRemains to chunk size
    b. If O_APPEND, use b_seek, to set the offset to the end of the file
        i. b_seek(FCBindex, fcb.currentRead, SEEK_END)
6. Free tempWorkingDir, lastToken, buf
7. Return the index number

int b_read(b_io_fd fd, char *buffer, int count)

- Return the 0 when reach the EOF, otherwise the number of bytes that we read

1. Check if the access mode has read, if no, print error message and return failed
2. Calculate the bufferRemainingBytes
3. Calculate how many bytes have been processed to check EOF
4. Update the count if user is asking more than we can process
5. Safety check if count is smaller than zero
6. Calculate part 1, 2, and 3
    a. Case 1: count <= bufferRemainingBytes
        i. Part 1 = count
        ii. Part 2 = 0
        iii. Part 3 = 0
    b. Else cases
        i. Part 1 = bufferRemainingBytes
        ii. Part 3 = count - bufferRemainingBytes
        iii. transferBlocks = part 3 / chunk size
        iv. Part 2 = transferBlocks * chunk size
        v. Part 3 = part 3 - part 2
7. Check if Part 1 is larger than zero
    a. Copy data from our buffer to user's buffer
    b. Update the buffer index

8. Check if part 2 is larger than zero
    a. Read the blocks directory from disk (current block + file location) to user buffer at index part 1 for transferBlocks
    b. Update the current block
    c. Update the number of part 2 to byteRead
9. Check if part 3 is larger than zero
    a. Reload our buffer
    b. Update the current block
    c. Update the buffer index to 0
    d. Set the buffer length to bytes that we read
    e. Check if bytes that we read is smaller than part 3, if yes set part 3 = bytesRead
    f. Then double check if part 3 > 0
    g. If yes, then memcpy the bytes in our buf to user's buffer at part1+part2 location, with part 3 bytes
    h. Update index
10. Calculate returned bytes
11. Return bytes

int b_write(b_io_fd fd, char *buffer, int count)

1. Check if the access mode has write, if no, print error message and return failed
2. Check if we have enough space for count, if not, then update the count (after this step, we have enough space to write into the file)
    a. If fileActualSize + count > fileMaxSize, then reallocate blocks for write
        i. Get new LBA location with passing in blockCount + 5
        ii. Create an empty buffer with size of new blockCount * block size
        iii. LBAread the old content to the bigger buffer
        iv. Update the fcb information
            1. Declare tempWorkingDir and lastToken
            2. Use pathParser
            3. Find the DE that has the same name as lastToken in tempWorkingDir -> DEindex
            4. tempWorkingDir[DEindex].location -> new location
            5. tempWorkingDir[DEindex].size -> original blockCount + 5 * blockSize
            6. tempWorkingDir[DEindex].blockCount += 5
            7. fi.location -> new location
            8. fi.size -> new size
            9. fi.blockCount -> new blockCount

10. fcb.currentBlock -> increment by the difference between new blockCount and old blockCount

v. LBAwrite the tempWorkingDir for tempWorkingDir[0].blockCount to tempWorkingDir[0].location

vi. Free tempWorkingDir and lastToken and buffer

3. Calculate the bufferRemainingBytes = chunk size - bufIndex
4. Calculate part 1, 2, and 3
    a. Case 1: count <= bufferRemainingBytes
        i. Part 1 = count
        ii. Part 2 = 0
        iii. Part 3 = 0
    b. Else cases
        i. Part 1 = bufferRemainingBytes
        ii. Part 3 = count - bufferRemainingBytes
        iii. transferBlocks = part 3 / chunk size
        iv. Part 2 = transferBlocks * chunk size
        v. Part 3 = part 3 - part 2
5. Part 1
    a. Copy the user given buffer to our buffer
    b. Update the index
    c. If the bufRemain = 0, write to Disk and update block count, index
6. Part 2
    a. LBAwrite the user given data at part 1 index to the disk
    b. LBAwrite(buffer + part 1, current block + location, transferBlocks)
    c. Update the current block
    d. Update the part 2 to byteWrite
7. Part 3
    a. Copy the user given buffer to our buffer
    b. Update index
    c. If the bufRemain = 0, write to Disk and update block count, index
8. Calculate the returned byte
9. Return byte

int b_seek(b_io_fd fd, off_t offset, int whence)

- Repositions the file offset of the open file description
- Returns new offset, or -1 when there's an error
- Whence: option for offset()

- SEEK_SET: fd = offset (0x00) (The file offset is set to offset bytes.)
- SEEK_CUR: fd += offset (0x01) (The file offset is set to its current location plus offset bytes.)
- SEEK_END: fd = (start block position + file size / 512) + offset (if positive then error) (0x02) (The file offset is set to the size of the file plus offset bytes.)

void b_close(b_io_fd fd)

1. Release any acclocation space (buf)
2. Set fi of the fd index in fcbArray to null

mv commands

- Four situation
    - Renaming a file (existing filename, non-existing filename)
    - Renaming a directory (existing directory, non-existing directory)
        - Loop through the DE to find first argument then change the name
    - Moving a file (existing filename, existing directory)
    - Moving a directory (existing directory, existing directory)
        - Loop through the DE to find first argument
        - Loop through the DE to find second argument and get the location
        - Set the location to the new location

1. Check if the argcnt is more than three (command, arg1, arg2), if not, print usage and return -1
2. Create two tempWorkingDir(1, 2) and two lastToken(1, 2)
3. Check first argument with EXIST_DIR | EXIST_FILE condition, if invalid, print error message and return -1
4. Check second argument with EXIST_DIR | EXIST_FILE condition
    a. If invalid -> rename the file
        i. Loop through the tempWorkingDir1 to find lastToken1
        ii. Set the DE.name to lastToken2
        iii. LBAwrite the tempWorkingDir1
    b. If valid -> change the location
        i. Loop through the tempWorkingDir1 to find lastToken1
        ii. Loop through the tempWorkingDir2 to find lastToken2
        iii. LBAread the lastToken2 DE array to tempWorkingDir2
        iv. Loop through tempWorkingDir2 to find new DEindex
        v. Create a temp DE pointer to store lastToken1 info
        vi. Set the DE in tempWorkingDir1 to NULL
        vii. Set the tempWorkingDir2[DEindex] to temp DE
        viii. LBAwrite tempWorkingDir1
        ix. LBAwrite tempWorkingDir2
        x. Free temp DE
5. Free tempWorkingDir1&2 and lastToken1&2

bool pathParser(char *path, unsigned char condition, DE *tempWorkingDir, char *lastToken);

- Return 0 when failed, 1 when success

1. Malloc space for tempWorkingDir and lastToken
    a. allocateDirectory();
    b. Allocate 256 bytes for lastToken
    c. Check if malloc successfully
2. Check if the path is relative or absolute (use '/')
    a. Relative: read the blocks at mfs_currentLocation to the tempWorkingDir
    b. Absolute: read the root to the tempworkingDir
3. Tokenizing the path
    a. Declare tokenCount, token array, theRest, token
    b. Add token to the token array and increment the tokenCount by 1
4. Update the tempWorkingDir to the second last token (nested for loop)
    a. Declare bool found as a flag
    b. If found token DE in tempWorkingDir, read that into the tempWorkingDir, set found to true, then break this loop
    c. After running through the inner loop, check if found. If yes, reset found to 0. If not, print err message and return 0.
5. Get lastToken from the token array if the array size is not 0
6. Loop through the tempWorkingDir to see if it is an existed dir, file or non existing (special case, "/" which array size would be 0, should returned EXIST_DIR immediately if we are looking at root)
7. Compare the condition and given condition, then return validation

Helper Functions

bool allocateDirectory();

1. Calculate the malloc size
2. Declare and allocate the return value
3. Return the malloc failed or success

```
void printDEInfo(DE de)
{
  printf("--- DE info ---\n");
  printf("- name: %s\n- size: %d\n- pointingLocation: %d\n", de.name, de.size, de.location);
  printf("- isDir: %d\n- createTime: %s\n- lastMod: %s\n- lastAccess: %s\n", de.isDir,
ctime(&de.createTime), ctime(&de.lastModTime), ctime(&de.lastAccessTime));
}
```