Spring 2023
FINAL VERSION
File System Group Project

Team Name: Chinese
GitHub Name: yuqiao1205
Group Member:

Yan Peng	ID: 921759056
Zhenyu Lin	ID: 920225329
Zaijing Liu	ID: 921394952
Zicheng Tan	ID: 920261872

The GitHub Link for Group Submission

https://github.com/CSC415-2023-Spring/csc415-filesystem-yuqiao1205

Description of File System:

Our file system can manage the files and directories based on a hierarchical filesystem, typified by the file system in Linux. This file system uses a contiguous allocation scheme to allocate disk space to files, each file occupies a contiguous series of blocks on disk. We used a free space bitmap that tracks what blocks on the disk have been allocated. With this system, storage devices must be partitioned and formatted before the first use. We initialize a volume control block, free space map and root directory. Beyond that, the file system is typically composed of two main components: a directory structure and a set of data blocks containing files and directories. The directory structure is organized as a tree with the root directory serving as the anchor. Each directory consists of a table of directory entries, which provides information on files and directories and which data blocks store the actual file contents. Lastly, our file system provides an interactive interface for users to use commands including (ls, cp, mv, md, rm, touch, cat, cp2l, cp2fs, cd, pwd, history, help) to manage the file system. The file system uses a main driver fsshell.c to dispatch commands entered into the shell. These in turn call on functions in mfs.c and b_io.c which utilize lower-level implementations in fsVol (volume), fsFree (free space map), and fsDir (directory and path management).

About Design

1. Volume

A volume is represented by the Volume Control Block which knows which blocks are used by the file system and the block size. It also references key structures, namely the free space map and the root directory, serving as the root data structure for the system.

2. Free space management

In this implementation free space is allocated in contiguous runs. The free space is tracked using a bitmap approach. Each bit in the byte array corresponds to a block on the volume, and a value of 1 indicates that the block is free, while a value of 0 indicates that it is in use. The interface between the free space manager and the rest of the system is a few functions that produce arrays of logical block positions. This interface was chosen so that if the free space management approach changed the rest of the filesystem code would not have to change. In any case we maintained the bitmap approach.

3. Directories

Directories are represented by a table of directory entries on disk which are linked into a tree by references to other directories by the logical block position on disk. When loaded into memory a tuple of (b_dcb) (pointer to directory entry in parent directory, pointer to table of directory entries, starting block) is used to tie all the pieces of information together. Directories are indexed in memory by the starting block on disk which is unique.

4. Files

On disk files are represented by a directory entry and a contiguous series of blocks. In memory they are represented by a structure (b_fcb) which binds the information about the file with an offset and other information needed for operations (open, seek, read, write and close).

Issue We Had

1. When we were working on the “mv” command, we initially thought that we only needed to move the directory entry to another location. However, this caused an issue where even if the file was moved to another directory, it

was not able to read the content of the file and was not able to remove the file. To resolve this issue, I physically created a new file in a target directory and copied the content of the old file to the new file. Finally, I removed the file from the old source directory.

2. Another issue arose when we were working on the “cd” command. We used the stroke function initially; however, this function changed the original string. Hence, we created a deep copy of the path string and used the stroke function so that it wouldn’t affect the original global path string.

3. Another cd issue: Initially, the multi-layer directory structure combined with ".." and "." could not be implemented, only works in the case of common examples is implemented. We rewrote the original code to first combine the PWD and path if necessary to form a compound and then removed any extra . and .. with the corresponding path segment.

4. pwd issue: In some complicated cases, for example, /foo/bar/qux/../../bar, the correct current path information cannot be displayed. We rewrote fs_setcwd(), uses a function absolute path that was rewritten into a more general form to handle many . and .. and avoid trailing slashes.

5. cp had a few issues when copying to directories where the filename must be appended to the destination path to get the right full pathname. This required some updates to the logic.

6. Segmentation Fault: sometimes when we change to a third directory to do some operation will cause a segmentation fault. To find the root cause, I modified Makefile, and added AddressSanitizer as so #CFLAGS= -g -I. -fsanitize=address -fno-omit-frame-pointer. This tells me where the segmentation fault occurred and helped me to solve this problems. For example, when we allocate stack for directory entries, based on the debug information, we added 1 to the token count because the stack needs an extra slot to store the root directory entry. Finally, we resolved the issue.

Detail of How Driver Program Works

The driver uses a dispatch table to match the name of commands to the functions that implement them, e.g. “cd” to the function cmd_cd. These functions in turn check and populate arguments and then draw upon the functions in mfs.c and b_io.c to do their work, finally they check the return values and message the user in failure cases. The following table lists all the functions and their dependencies on the implementation layer of the filesystem.

command	dependencies
ls	fs_isDir() ,fs_opendir(), fs_isFile, fs_getcwd
cp:	b_open(), b_read(), b_write(), b_close()
md	mk_dir()
rm	fs_isDir(),fs_rmDir, fs_isFile(), f_delete()
touch	b_open(),b_close()
cat	b_open(),b_read()
cp2l	b_open(),b_read(),b_close()
cp2fs	b_open(), b_write(),b_close()
cd	fs_setCwd()
pwd	fs_getCwd()

For example, cd invokes cmd_cd which checks the argument and prints a usage message if it was not supplied. It also transforms the argument, if necessary, in this case stripping quotes from it if it is quoted. It calls fs_setcwd which is implemented in mfs.c, and checks the return value, messaging the user if there is some issue with the function.

How to Run Our File System

Type “make run” into command line to run our file system shell.
Type the commands we implemented below to perform different tasks.

1. ls	**2. cp**
3. mv	**4. rm**
5. touch	**6. cat**
7. cp2l	**8. cp2fs**
9. cd	**10. md**
11. pwd	**12. history**
13. help	

Main Structures of Our File System

1. **A struct to hold the information about a file while it is open and being used for operations.**

```
typedef struct b_fcb {
        char *path;  // path of the file
        uint64_t offset;  // offset in the file
        char *buf;        //holds the open file buffer
        int64_t block_index; // block index in the file (blocks)
        int index;        //holds the current position in the buffer (bytes)
        int buflen;     //holds how many valid bytes are in the buffer (bytes)
        DirectoryEntry *dirent;   // a point to directory entry
```

```
        int flags;   // flags for open
};
```

2. **A single entry in a directory representing either a file or directory.**

```
typedef struct {
        char name[88];       // name of the file or directory
        int type;
        int size;            // size of the file in bytes
        int startingBlock;    // starting location of the file on disk
        int numBlocks;        // number of blocks used by the file
time_t accessTime;    // time of last access
        time_t modTime;       // time of last modification
        time_t createTime;    // time of last status change
} DirectoryEntry;
```

3. **The VCB to identify, size and track key structures of a volume on a physical device, in particular the free space map and root directory.**

```
typedef struct VolumeControlBlock {

        long int magicNumber;
        long int magicNumber2;

        uint64_t numberOfBlocks;
        uint64_t blockSize;

        uint64_t freeSpaceMapPosition;
        uint64_t rootDirectoryPosition;

        // internal use only
        uint64_t freeSpaceBlocks;
} VolumeControlBlock;
```

4. **A structure for managing directories in memory, providing a way to store and access directory-related information, and ensure that a directory is only in memory in one place.**

```
typedef struct b_dcb {
        int startingBlock; // serves as unused (-1) or LB position 1+
        DirectoryEntry * dirEntry; // DirectoryEntry for this directory
        DirectoryEntry * dirTable; // Pointer to the first entry _in_ the
```

```
directory
        int index;
} b_dcb;
```

Screen shot of compilation:

```
student@student-VirtualBox:~/src/csc415-filesystem-yuqiao1205$ make
gcc -c -o fsshell.o fsshell.c -g -I.
gcc -c -o fsInit.o fsInit.c -g -I.
gcc -c -o fsVol.o fsVol.c -g -I.
gcc -c -o fsFree.o fsFree.c -g -I.
gcc -c -o fsDir.o fsDir.c -g -I.
gcc -c -o mfs.o mfs.c -g -I.
gcc -c -o b_io.o b_io.c -g -I.
gcc -o fsshell fsshell.o fsInit.o fsVol.o fsFree.o fsDir.o mfs.o b_io.o fsLow.o -g -I. -lm -l readline -l pthread
student@student-VirtualBox:~/src/csc415-filesystem-yuqiao1205$
```

Showing Each of the Commands Listed:

1. **"ls" command:**
   - **ls:** Lists the file in a directory. (The example below shows that the **current directory foo** contains a file test1.txt and a directory bar).
   - **ls -l:** Displays the information for each file and directory in the current directory.
   - **ls -la:** Displays the same information as ls -l command but includes hidden files (whose names start with a dot (“.”), (“..”) and directories in the output.

Screen short for ls command with some options

```
Prompt > pwd
/foo
Prompt > ls

test1.txt
bar
Prompt > ls -l

-           1250    2023-04-20 11:34    test1.txt
D           1536    2023-04-20 11:44    bar
Prompt > ls -la

D           1536    2023-04-20 11:33    .
D           1536    2023-04-20 11:33    ..
-           1250    2023-04-20 11:34    test1.txt
D           1536    2023-04-20 11:44    bar
Prompt >
```

Screen short for ls that list the contents of a directory by

specifying its path.

```
Prompt > ls /foo/bar

qux
Prompt >
```

Screen short for ls that list the contents of a directory by specifying its name.

```
Prompt > pwd
/
Prompt > cp2fs testfile.txt 1.txt
Prompt > ls

foo
1.txt
Prompt > cp 1.txt foo
Prompt > ls foo

bar
1.txt
Prompt >
```

Screen short for ls that list the detail of directory.

```
Prompt > ls -l foo

D          1536    2023-04-27 18:31    bar
Prompt > cp 3.txt foo/3.txt
Prompt > ls -l foo

D          1536    2023-04-27 18:31    bar
-          1250    2023-04-29 12:37    3.txt
Prompt >
```

2. **“md” command: make a new directory.**

Screen short for making two directories foo1 and foo2 within root (/) directory.

```
Prompt > ls

foo
bar
Prompt > md foo1
Prompt > md foo2
Prompt > ls

foo
bar
foo1
foo2
Prompt >
```

Screen short for making a new directory within another directory.

```
Prompt > md foo/foo1
Prompt > md foo/foo1/foo2
Prompt > ls -l foo/foo1

D         1536   2023-04-29 12:58   foo2
Prompt > ls -l foo

D         1536   2023-04-27 18:31   bar
D         1536   2023-04-29 12:58   foo1
Prompt >
```

3. **"rm" command: remove a directory or file.**

Screen short for removing a directory.

```
Prompt > cd /bar/bar1
Prompt > ls

bar2
bar3
Prompt > rm bar3
Prompt > ls

bar2
Prompt >
```

Screen short for removing directory by specify a path.

```
Prompt > ls

bar
bar1
Prompt > pwd
/foo
Prompt > cd /
Prompt > rm /foo/bar
Prompt > ls foo

bar1
Prompt >
```

```
Prompt > ls foo/bar

qux
Prompt > ls foo/bar/qux

Prompt > rm foo/bar/qux
Prompt > ls foo/bar

Prompt >
```

Screen short for removing a file test1.txt.

```
Prompt > ls

foo
bar
foo1
foo2
test1.txt
test2.txt
Prompt > rm test1.txt
Prompt > ls

foo
bar
foo1
foo2
test2.txt
Prompt >
```

Screen short for removing a file by specifying their paths.

```
Prompt > cd /
Prompt > ls foo

bar
1.txt
2.txt
Prompt > rm /foo/1.txt
Prompt > ls foo

bar
2.txt
Prompt >
```

4. **"pwd" command: print working directory**

When working directory is foo.

```
Prompt > cd foo
Prompt > pwd
/foo
Prompt >
```

When working directory is root (*/*).

```
Prompt > pwd
/
Prompt >
```

Screen short for showing current working multi-level directory.

```
Prompt > cd foo/foo1/foo2
Prompt > pwd
/foo/foo1/foo2
Prompt >
```

5. **"cd" command: change directory**

Screen short for cd to many levels' directory combined by using "..", "."

```
Prompt > md foo
Prompt > cd foo
Prompt > md bar
Prompt > cd bar
Prompt > md qux
Prompt > cd qux
Prompt > pwd
/foo/bar/qux
Prompt > cd /
Prompt > pwd
/
Prompt > cd /foo/bar/qux/../../bar
Prompt > pwd
/foo/bar
Prompt >
```

Screen short for using cd../..

```
Prompt > cd /foo/bar/qux
Prompt > cd ../..
Prompt > pwd
/foo
Prompt >
```

Screen short: When cd followed by the name of a directory.

```
Prompt > ls

foo
bar
Prompt > cd bar
Prompt > pwd
/bar
Prompt >
```

Screen short for using cd .. command.

1. **Screen short for going back to the parent directory (root "/").**

```
Prompt > pwd
/bar
Prompt > cd ..
Prompt > pwd
/
Prompt >
```

2. **In this case, go back to the parent directory foo.**

```
Prompt > pwd
/foo/foo1
Prompt > cd ..
Prompt > pwd
/foo
Prompt >
```

Screen short for using "cd /" back to root.

```
Prompt > pwd
/foo/bar
Prompt > cd /
Prompt > pwd
/
Prompt >
```

Screen short for using "cd ." to change to current directory

```
Prompt > md foo
Prompt > md bar
Prompt > ls

foo
bar
Prompt > cd foo
Prompt > pwd
/foo
Prompt > cd .
Prompt > pwd
/foo
Prompt >
```

6. **"cp2fs" command: Copies a file from the Linux file system to the test file system.**

In this case, we have a file **testfile.txt** in Linux file system.

```
student@student-VirtualBox:~/src/csc415-filesystem-yuqiao1205$ ls
bigger.txt  fsDir.c   fsFree.h  fsLow.h    fsshell.c  fsVol.o   mfs.h                  SampleVolume
b_io.c      fsDir.h   fsFree.o  fsLowM1.o  fsshell.o  Hexdump   mfs.o                  testfile.txt
b_io.h      fsDir.o   fsInit.c  fsLow.o    fsVol.c    Makefile  milestone1_report_1.0.pdf  test_scripts
b_io.o      fsFree.c  fsInit.o  fsshell    fsVol.h    mfs.c     README.md              tools
student@student-VirtualBox:~/src/csc415-filesystem-yuqiao1205$
```

Screen short for using cp2fs command.

```
Prompt > ls

foo1
bar1
Prompt > cp2fs testfile.txt fs.txt
Prompt > ls -l

D          1536   2023-04-26 12:00   foo1
D          1536   2023-04-26 11:37   bar1
-          1250   2023-04-26 12:08   fs.txt
Prompt >
```

7. **"cp2l" command: Copy a file from our file system to the Linux file system.**

Screen short for using cp2l command.

```
Prompt > ls

fs.txt
Prompt > cp2l fs.txt fs1.txt
Prompt >
```

Now, we can check, fs1.txt is now in Linux system.

```
student@student-VirtualBox:~/src/csc415-filesystem-yuqiao1205$ ls
bigger.txt  b_io.o   fsDir.h   fsFree.h  fsInit.o   fsLow.o    fsshell.o  fsVol.o   mfs.c  milestone1_report_1.0.pdf  testfile.txt
b_io.c      fs1.txt  fsDir.o   fsFree.o  fsLow.h    fsshell    fsVol.c    Hexdump   mfs.h  README.md                  test_scripts
b_io.h      fsDir.c  fsFree.c  fsInit.c  fsLowM1.o  fsshell.c  fsVol.h    Makefile  mfs.o  SampleVolume               tools
student@student-VirtualBox:~/src/csc415-filesystem-yuqiao1205$
```

8. **"cp" command: copy a file source**

Screen short for copying a source file to destination file.

```
Prompt > ls

foo
1.txt
Prompt > cp 1.txt 2.txt
Prompt > ls

foo
1.txt
2.txt
Prompt >
```

Screen short for copying a source file to another directory file.

```
Prompt > ls

foo
1.txt
2.txt
Prompt > cp 2.txt /foo/3.txt
Prompt > cd foo
Prompt > ls -la

D          1536    2023-04-26 11:35    .
D          1536    2023-04-26 11:35    ..
D          1536    2023-04-26 12:00    foo1
D          1536    2023-04-26 11:37    bar1
-          1250    2023-04-26 12:13    fs.txt
-          1250    2023-04-26 12:19    3.txt
Prompt >
```

Screen short for copying one directory file to another directory file.

```
Prompt > pwd
/
Prompt > ls

foo
1.txt
5.txt
Prompt > cp /foo/3.txt /3.txt
Prompt > ls

foo
1.txt
5.txt
3.txt
Prompt >
```

Screen short for copying a source file to directory.

```
Prompt > ls

foo
1.txt
2.txt
Prompt > cp 2.txt foo
Prompt > ls foo

1.txt
2.txt
Prompt >
```

Screen short for the case that if cp only used one file name.

```
Prompt > ls

3.txt
foo
1.txt
Prompt > cp 1.txt
Missing required destination!
Prompt > cp 1.txt 2.txt
Prompt > ls -l

-          1250    2023-04-29 12:37    3.txt
D          1536    2023-04-27 18:31    foo
-          1250    2023-04-29 12:50    1.txt
-          1250    2023-04-29 12:50    2.txt
Prompt >
```

9. **"touch" command: Creates/ Touches a file.**

Screen short
case 1: when the file does not exist, will create a new file.

```
Prompt > ls

Prompt > touch touchtest.txt
Prompt > ls -l

-           0   2023-04-20 22:04   touchtest.txt
Prompt >
```

case 2: when the file exists, update the modification timestamp of an existing file.

```
Prompt > touch touchtest.txt
Prompt > ls -l

-           0   2023-04-20 22:09   touchtest.txt
Prompt >
```

10. **"cat" command: Displace the file to the console.**

Screen short for cat command

```
Prompt > ls

touchtest.txt
README1.md
test1.txt
Prompt > cat test1.txt
A0A1A2A3A4A5A6A7A8A9AAABACADAEAFB0B1B2B3B4B5B6B7B8B9BABBBCBDBEBFC0C1C2C3C4C5C6C7C8C9CACBCCCDCECFD0D1D2D3D4D5D6D7D8D9DADBDCDDDEDFE0E1
E2E3E4E5E6E7E8E9EAEBECEDEEEFF0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF0G1G2G
Prompt >
```

11. **"mv" command: Moves a file.**

Screen short for moving a file to a file = rename the file to a new file.

```
Prompt > ls -l

D          1536    2023-04-26 12:26    foo
-          1250    2023-04-26 12:27    1.txt
-          1250    2023-04-26 12:31    2.txt
Prompt > mv 1.txt 3.txt
Prompt > ls -l

D          1536    2023-04-26 12:26    foo
-          1250    2023-04-26 12:31    2.txt
-          1250    2023-04-26 12:38    3.txt
Prompt >
```

Screen short for moving a file to another directory.

```
Prompt > ls

foo
test3.txt
test2.txt
Prompt > cd /foo/foo1
Prompt > pwd
/foo/foo1
Prompt > cd /
Prompt > mv test3.txt /foo/foo1
Prompt > cd /foo/foo1
Prompt > ls -la

D           2560    2023-04-23 23:26    .
D           2560    2023-04-23 22:58    ..
-           1250    2023-04-23 23:30    test1.txt
-           1250    2023-04-23 23:32    test3.txt
Prompt >
```

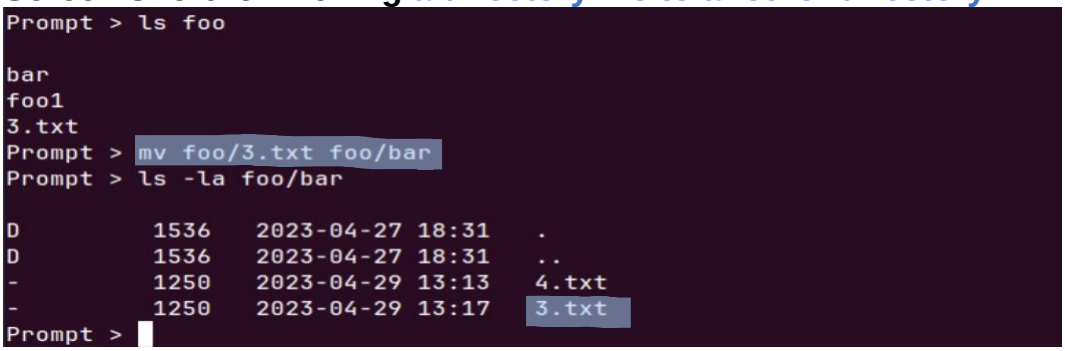
Screen short for moving a file to another directory and rename a new file.

```
Prompt > pwd
/foo/foo1
Prompt > cp test1.txt test5.txt
Prompt > ls

test1.txt
test5.txt
Prompt > mv test5.txt /foo/test6.txt
Prompt > cd ..
Prompt > ls

foo1
test4.txt
test6.txt
Prompt >
```

Screen short for moving a directory file to another directory file.

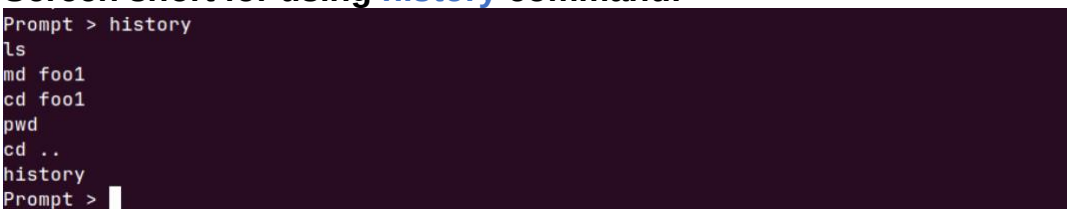
```
Prompt > pwd
/foo
Prompt > ls

bar
foo1
1.txt
Prompt > cd ..
Prompt > mv foo/1.txt foo/bar/4.txt
Prompt > ls /foo/bar

4.txt
Prompt >
```

Screen short for moving a directory file to another directory.



12. **"history" command: Prints out the history.**

Screen short for using history command.



13. **"help" command: Prints out help.**

Screen short for using help command.