Dale Armstrong

920649883

Professor Robert Bierman

CSC 415 - Section 03

Assignment 3 - File System

1. Program Goals

The goal of this project is to create a basic filesystem that will help us to better understand how a file system works, how it is put together, the design decisions that go into creating a file system and the various tradeoffs, and getting more experience working on larger projects. A file system is a part of every operating system as there must be a way to store, read, and copy data.

2. Description of my File System

SuperBlock	Inodes	Bit Vector	Data Blocks

(How the data is stored on disk)

This file system is designed around many of the ideas in the Linux filesystem. The super block stores volume information and takes up a single block at the beginning of the volume. The system will check the signature at the start and end of the super block for a match, before loading the volume and is loaded when the file system starts.

Directories/Files are made up of metadata and data blocks. Inodes are the file metadata containing various file information such as size, reserved block size, type, date modified, and contains 10 direct data block pointers, and 2 indirect data block pointers. The first indirect data block pointer in an inode has one level of indirection, while the 2nd has 2 levels of indirection.

Originally I created the file system with 4096 block size in mind, so the max file size could be upwards of 1GB. Because of the limit of a 512 block size, the file size limit is 2MB. Because I originally planned for a 4096 block size, I left out the third indirect block. The filesystem is capable of different block sizes and it only needs that argument set when creating the partition.

Directory data blocks are filled with <name, Inode ID> pairs known as the file control blocks (FCB). This minimizes the amount of space each file takes up within a directory and allows for links from various files to the same inode.

The number of inodes is first calculated based on the number of blocks there are and then the nearest prime number that is larger than the calculated number is used. This is to create a more efficient hash function to quickly search for a free inode, rather than iterating through the inodes and checking if any are free.

A bit vector is used to store the free and used blocks. 1 is considered in use, while 0 is unused. The bit vector gets loaded into memory and whenever allocation or deallocation occurs, the bit vector is written to disk.

Before I implemented the file descriptor, I had already created readFile and writeFile, that both contain internal buffers to read/write individual blocks. Currently my fileOpen, fileRead, and fileWrite are wrappers for utilizing the file descriptor with my read and write functions.

With these structures and additional functions the file system is able to store/retrieve files, create and manage directories, moves files/directories, change names, copy files, and keep track of all information needed for various functions. There are many helper functions that support the main functions of the file system, each having a particular goal, with the main ones being allocating blocks, deallocating blocks, parsing paths, creating files, reading and writing files, copying directories and files, reading and writing inodes, and various bit vector functions.

3. Issues

I didn't have too many issues other than the small bug here and there that I had to track down. Most of the bugs dealt with accidently using a wrong variable name or I forgot to change something in the inode when moving a file. I normally spend a lot of time researching and going over things before I start to code, then rereading my code several times to minimize the amount of debugging required.

The hardest part was making sure that my logic was correct when allocating, deallocating, finding free blocks, writing a file, and reading a file. Each of these parts are similar in that you have to traverse not only the direct blocks, but also the indirect blocks and the logic used takes a long time to get just right. Even though they are similar, each part accesses those blocks slightly differently, so each part equally takes a long time to figure out.

My file system is not very efficient, especially as it reads each block one at a time. It then has to retrieve the pointer to the next block, read that block, etc… until all requested blocks are read. I believe it would be more efficient if it were to create a large enough buffer to read contiguous blocks all at once. Maybe even read ahead into a cache incase a file needed more data in a future request.

Another place where it isn't very efficient is the bit vector, perhaps I should have randomized or stored where the last blocks were written to, instead of iterating from the start each time. It does attempt to find contiguous blocks, but if there is not a large enough space to hold the entire file, it will take the largest hole and it will search again from the beginning. A way to work around that might be to switch to another searching algorithm once the drive hits a certain threshold filled and/or based on the requested block size ratio of blocks free to requested blocks. Another way would be to incorporate some sort of defrag or compression of data during idle times. Luckily the bit vector is read into memory where it is far faster than if it had to be referenced from disk each time.

4. Driver Instructions

To compile, use the included makefile, by having make installed and typing make. It will create the myfs executable file. I have included a 9MB 512 block size test file volume named "testfile" that is already partitioned and formatted.

- To run the program utilizing the testfile just type "./myfs"
- To create a new file type ./myfs <filename> <volumesize> <blocksize>
    - Once created, the program will present the option to format the volume.
    - The file system will automatically calculate the required inodes, bit vector size, and data blocks. The minimum volume size is currently set to 20 blocks, which the file system will automatically set the volume size to if the requested number is below 20.

Driver Commands:

- **help [command]**
    - **help** by itself will display all commands and what they do.
    - **help <command>** will display usage information about a particular command
- **format** - Formats the partition and installs the filesystem. Will delete any current filesystems that are installed.
- **ls** - Lists the files in the current directory and their accompanying information. If the file is a directory, the size and blocks reserved are a sum of the directory size and reserved plus the sum of all files and directories residing inside that directory.
- **cd <directoryname>** - Lists files in the current directory.
    - **cd** or **cd /** will go straight to root
    - **cd ..** will move up one directory
    - It is possible to traverse several directories in one command
      eg "**cd ../home/etc/../etc**"
- **pwd** - Prints the full working directory
- **cp <source> <destination>** - Copies the file from the source to the destination
- **mv <source> <destination>** - Moves the file from the source to the destination. You may also use this function to change a filename.
- **rm <filename>** - Deletes the file. This is only for file types, directories require rmdir.
- **rmdir <directoryname>** - Deletes the directory and all files and folders in the directory, freeing up used blocks.
- **mkdir <directoryname>** - Creates the given directory
- **mkfile <filename> [size]** - Creates an empty file of the given filename, with an optional reserve size in bytes.
- **lsfs** - Displays various information about the filesystem. Free blocks, used blocks, block size, volume size, which block each part of the filesystem starts at, the maximum file size this filesystem could potentially support at this block size, and the maximum block size that this filesystem could potentially support based on the number of indirect blocks and direct blocks.
- **resize <filename> <size>** - Resizes the file. If the number of bytes exceeds the reserved block size, then new empty blocks will be allocated to the file. If the size is decreased, the reserved blocks will not be reduced. This only works on files and not directories.

- **reserve <filename> <size>** - Resizes the reserved blocks. The minimum reserved blocks is either one block or the number of blocks required to hold the size of the file. Ie: if size is 0, then blocks reserved will be 1, if size is between 1-2 block sizes, reserved size will be 2.
- **cpin <source> <destination>** - Copies a file from the linux filesystem into this filesystem
- **cpout <source> <destination>** - Copies a file from this filesystem to the linux filesystem
- **exit** - exits the file system

5. Driver Demonstration

This screenshot shows the **makefile**, the creation of the partition with the given arguments, **format**, **exit**, then loading the provided "testfile" without requiring any arguments.

```
student@student-VirtualBox:~/src$ make
gcc -c -o obj/FileSystem.o FileSystem.c -lm
gcc -c -o obj/fsdriver3.o fsdriver3.c -lm
gcc -c -o obj/fsLow.o fsLow.c -lm
gcc -o myfs obj/FileSystem.o obj/fsdriver3.o obj/fsLow.o -lm
student@student-VirtualBox:~/src$ ./myfs testfile 9000000 512
File testfile does not exist, errno = 2
File testfile not good to go, errno = 2
Block size is : 512
Created a volume with 8999936 bytes, broken into 17578 blocks of 512 bytes.
Opened testfile, Volume Size: 8999936;  BlockSize: 512; Return 0
Partition is not formatted.
You must format the partition to continue.
Format now? (y/n): y
Please wait, wiping partition....Done!
/>mkfile test
/>mkdir folder
/>ls
| Type | File Size | Reserved | Last Modified | File Name
   2              0        512   May 8 00:08    test
   1              0        512   May 8 00:08    folder/
/>format
Warning: This will delete the current filesystem!
Do you want to continue? (y/n): y
Please wait, wiping partition....Done!
Format success!
/>ls
| Type | File Size | Reserved | Last Modified | File Name
/>exit
student@student-VirtualBox:~/src$ ./myfs
File testfile does exist, errno = 0
File testfile good to go, errno = 0
Opened testfile, Volume Size: 8999936;  BlockSize: 512; Return 0
/>ls
| Type | File Size | Reserved | Last Modified | File Name
/>
```

help , help <command>

```
/>help
Type help <function> to get more information about a function
Commands:
format - formats the partition
lsfs   - lists the information of the current filesystem
ls     - lists files in the directory
cd     - changes current directory
pwd    - prints the full working directory
mkdir  - creates a directory
mkfile - creates a file with size
rmdir  - removes a directory
resize - changes the size of the file
reserve- increases or reduces the reserved blocks
cp     - copies a file from source to destination
mv     - moves a file from source to destination
rm     - deletes a file
cpin   - copy a file in from another filesystem
cpout  - copies a file to another filesystem
exit   - exit shell
/>help lsfs
Usage: lsfs
Lists the information about the current filesystem.
This includes: Volume name, volume ID, block size, number of blocks,
        free blocks, and space used.
/>help reserve
Usage: reserve <filename> <size>
Resizes the reserved blocks to hold the number of bytes requested.
Minimum reserved is one block or the minium blocks needed to hold the current size of the file.
Maximum blocks are limited by the number of available free blocks to be allocated.
/>help cd
Usage: cd <dirname>
Changes current directory to the given directory
/>
```

lsfs

```
/>lsfs
Volume name:        Untitled
Volume size:        8999936
Block size:         512
Blocks:             17578
Inodes:             8807
Free Inodes:        8806
Used Inodes:        1
Inode index:        1
Inode Blocks:       2477
BitVector index:    2478
BitVector Blocks:   4
Total Data Blocks:  15097
Free Data Blocks:   15096
Used Data Blocks:   1
Root Data index:    2482
FS Max File Size:   2135040
FS Max Blocks/File: 4170
/>
```

mkdir <directoryname>, mv <source> <destination>, cd <path>, pwd

```
/>mkdir new
/>mkdir old
/>mkdir folder2
/>mkdir folder
/>ls
| Type | File Size | Reserved | Last Modified | File Name
   1              0        512   May 8 00:10    new/
   1              0        512   May 8 00:10    old/
   1              0        512   May 8 00:10    folder2/
   1              0        512   May 8 00:10    folder/
/>mv folder folder2
/>mv old folder2/folder
/>mv folder2 new
/>ls
| Type | File Size | Reserved | Last Modified | File Name
   1            408       2048   May 8 00:11    new/
/>cd new
/new>ls
| Type | File Size | Reserved | Last Modified | File Name
   1            272       1536   May 8 00:11    folder2/
/new>cd folder2
/new/folder2>ls
| Type | File Size | Reserved | Last Modified | File Name
   1            136       1024   May 8 00:11    folder/
/new/folder2>cd folder
/new/folder2/folder>ls
| Type | File Size | Reserved | Last Modified | File Name
   1              0        512   May 8 00:11    old/
/new/folder2/folder>cd old
/new/folder2/folder/old>ls
| Type | File Size | Reserved | Last Modified | File Name
/new/folder2/folder/old>mkfile newfile 20000
/new/folder2/folder/old>cd /
/>ls
| Type | File Size | Reserved | Last Modified | File Name
   1            544      22528   May 8 00:11    new/
/>cd new/folder2/folder/old/../../folder/
/new/folder2/folder>pwd
/new/folder2/folder
/new/folder2/folder>
```

ls, cp <src directory> <dest directory>, rmdir <directory>

```
/>ls
| Type | File Size | Reserved | Last Modified | File Name
   1            544      22528   May 8 00:11    new/
/>cp new newcopy
/>ls
| Type | File Size | Reserved | Last Modified | File Name
   1            544      22528   May 8 00:11    new/
   1            544      22528   May 8 00:12    newcopy/
/>cd newcopy
/newcopy>ls
| Type | File Size | Reserved | Last Modified | File Name
   1            408      22016   May 8 00:12    folder2/
/newcopy>cd
/>rmdir new
This will delete all files located within the directory.
Do you want to continue? (y/n): y
/>ls
| Type | File Size | Reserved | Last Modified | File Name
   1            544      22528   May 8 00:12    newcopy/
/>
```

mkfile <filename> [reservesize], cp <src file> <dest file>, resize <file> <size>

```
/>mkfile testfile 2000
/>ls
| Type | File Size | Reserved | Last Modified | File Name
   1            544      22528   May 8 00:12    newcopy/
   2              0       2048   May 8 00:13    testfile
/>cp testfile testcopy
/>ls
| Type | File Size | Reserved | Last Modified | File Name
   1            544      22528   May 8 00:12    newcopy/
   2              0       2048   May 8 00:13    testfile
   2              0       2048   May 8 00:14    testcopy
/>resize testfile 2000
/>ls
| Type | File Size | Reserved | Last Modified | File Name
   1            544      22528   May 8 00:12    newcopy/
   2           2000       2048   May 8 00:14    testfile
   2              0       2048   May 8 00:14    testcopy
/>resize testcopy 4000
/>ls
| Type | File Size | Reserved | Last Modified | File Name
   1            544      22528   May 8 00:12    newcopy/
   2           2000       2048   May 8 00:14    testfile
   2           4000       4096   May 8 00:14    testcopy
/>
```

reserve <filename> <size>, **rm <filename>**

```
/>ls
| Type | File Size | Reserved | Last Modified | File Name
   1            544       22528   May 8 00:12    newcopy/
   2           2000        2048   May 8 00:14    testfile
   2           4000        4096   May 8 00:14    testcopy
/>reserve testfile 5000
/>ls
| Type | File Size | Reserved | Last Modified | File Name
   1            544       22528   May 8 00:12    newcopy/
   2           2000        5120   May 8 00:14    testfile
   2           4000        4096   May 8 00:14    testcopy
/>rm testfile
/>ls
| Type | File Size | Reserved | Last Modified | File Name
   1            544       22528   May 8 00:12    newcopy/
   2           4000        4096   May 8 00:14    testcopy
/>
```

cpout <source> <destination>, **cpin <source> <destination>**, also works on larger file sizes, **exit**

```
/>cpout testcopy acopy.txt
/>cpin acopy.txt acopycopy
/>ls
| Type | File Size | Reserved | Last Modified | File Name
   1            544       22528   May 8 00:12    newcopy/
   2           4000        4096   May 8 00:14    testcopy
   2           4000        4096   May 8 00:15    acopycopy
/>cpin test.txt warandpeace.txt
/>ls
| Type | File Size | Reserved | Last Modified | File Name
   1            544       22528   May 8 00:12    newcopy/
   2           4000        4096   May 8 00:14    testcopy
   2           4000        4096   May 8 00:15    acopycopy
   2         780701      780800   May 8 00:16    warandpeace.txt
/>mkdir home
/>mkdir etc
/>mkdir bin
/>ls
| Type | File Size | Reserved | Last Modified | File Name
   1            544       22528   May 8 00:12    newcopy/
   2           4000        4096   May 8 00:14    testcopy
   2           4000        4096   May 8 00:15    acopycopy
   2         780701      780800   May 8 00:16    warandpeace.txt
   1              0         512   May 8 00:16    home/
   1              0         512   May 8 00:16    etc/
   1              0         512   May 8 00:16    bin/
/>exit
student@student-VirtualBox:~/src$
```