Programming Assignment 2

Very Simple File System

Jeffrey Tan, S3851781

Tutor: Paul

Level attempted: HD+ **(base 64, MC Compliance, .gz)**

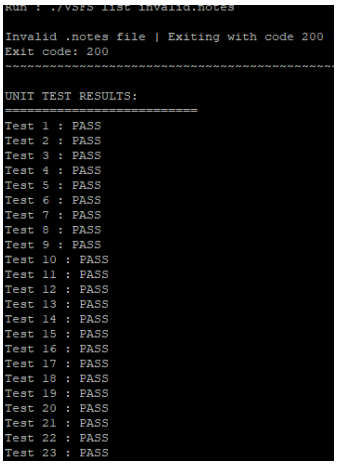
Files Summary

- VSFS.cpp – source code for program
- Makefile – makefile to compile VSFS.cpp into VSFS (executable)
- tests.sh – shell file with 23 unit tests
- results.txt – the results tests.sh on my end
- mcvsfs – shell file for mc compliance
- mc.ext – config file for mc to recognise .notes extension
- abc64.notes – a sample notes file with dummy files and directories preloaded, contents encoded in base64
- abc64.notes.gz – ^ but zipped to .gz
- abc.notes – if you wanted to test with base64 global variable set to false
- abc.notes.gz – ^ but zipped to .gz

Testing Instructions

1. Use `make` to compile the program.
2. Use `./tests.sh > results.txt 2>&1`
3. This will run the tests and put stdout and stderr into the results.txt file.
4. The end of results.txt will have a list of pass/fail results for each unit test index.
5. Feel free to test the program without ./test.sh using the commands:
   ./VSFS [command]... filesystem ... term1 ... term2

FYI: tests.sh will create all the notes and internal/external files and directories for you to do the tests (and also deletes/replaces these files on start-up so if you run the tests again it gives the same result).



TEST SUITE	
COPYIN	
1	Copying an empty file.
2	Replacing existing file and adding content.
3	When external file doesn't exist.
4	Creating intermediate directories.
5	Copying to a subdirectory.
6	File named "/" , "." , "..".
7	File starts or ends in "/".
COPYOUT	
8	Copy out with intermediate paths also created.
9	Internal file doesn't exist.
10	Copy out to existing file. Check text has changed.
MKDIR	
11	Directory already exists.
12	Directory must end in "/" .
13	Directory cannot start with "/".
14	Create a new directory.
15	Create a subdirectory to an existing directory.
16	Create a subdirectory to a non-existent directory.
RM	
17	File to be removed not found.
18	Remove a file.
RMDIR	
19	Directory doesn't exist.
20	Remove a directory and recursively, all subdirectories and files.
DEFRAG	
21	Remove deleted files and sort into file-tree sequence.
LIST	
22	List files into ls -lR format

OTHER	
23	Validation of notes file according to rules.
24	Using a .gz file that doesn't exist.
25	Using a .gz file that exists and should work the same.

MC Compliance Instructions

1. The shell file mcvsfs needs to be changed. At the moment, there is a variable:

```
EXE=/home/sh1/S3851781/VSFS
```

Change this to the directory where you have my executable compiled.

2. Place mcvsfs into ~/.local/share/mc/extfs.d/
3. Place mc.ext in ~/.config/mc/
   This associates .notes to cd file.notes/mcvsfs://
   Alternatively, you may want to just `cd abc64.notes/mcvsfs:// ` instead of adding mc.ext to the config.
4. Create a '.notes' file with "NOTES V1.0".
   Don't forget the newline after "NOTES V1.0"
   Or go with the abc64.notes file I've provided.

   NOTES V1.0

5. When your cd into the file, there may be a (or many) parsing error(s). But for some reason it doesn't affect the functionality after you get past the errors, and you can proceed with browsing the virtual filesystem.

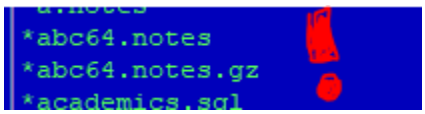
Important: abc64.notes won't open properly (invalid notes file error exit 200) in some directories, for some reason, it definitely doesn't open properly if it is still in the same directory as the source code for VSFS.cpp . Please move the abc64.notes (or whatever notes file you are trying to open in MC) outside of the directory with the source code.

I've tested abc.notes.gz to work the exact same as abc64.notes, however you would need to:

cd abc.notes.gz/mcvsfs://

As I have not associated .notes.gz in the mc.ext config file.

Warning:, if you have say abc64.notes and abc64.notes.gz in the same directory and you cd into the .gz version, MC will crash because the gzip program prompts a "do you want to replace abc64.notes" message. So please keep the notes and their .gz counterparts in different directories if you are testing the .gz versions.

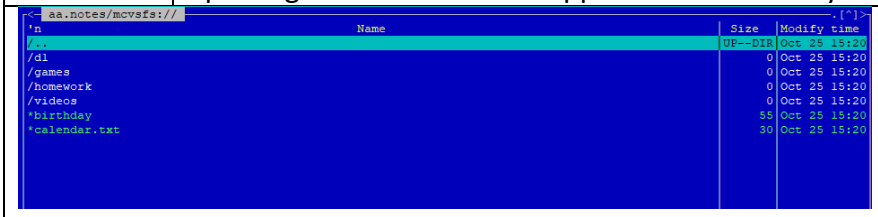
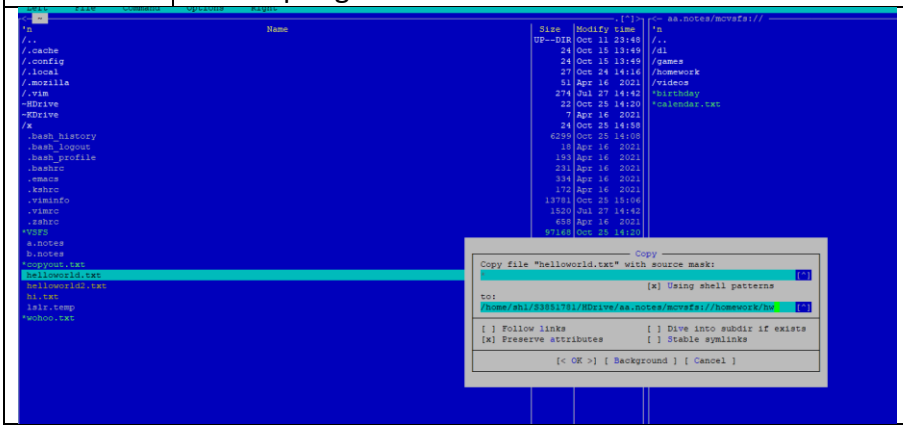


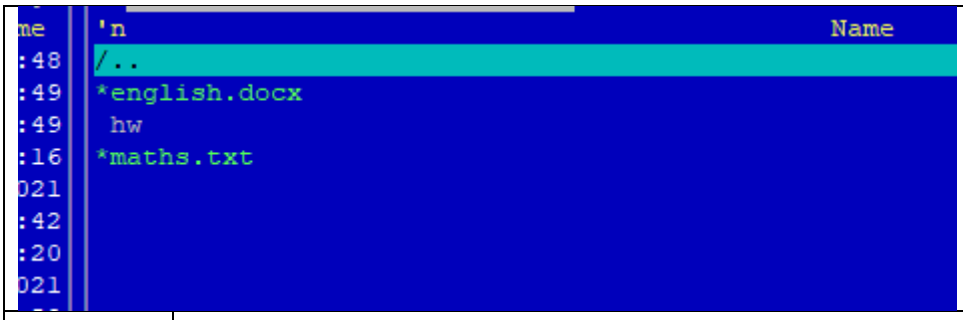
What's working and what's not (+ denotes working, - denotes an issue)

I have tested these claims myself. Please see the screenshots I've provided.
.notes file used in testing: abc.notes (included in submission).

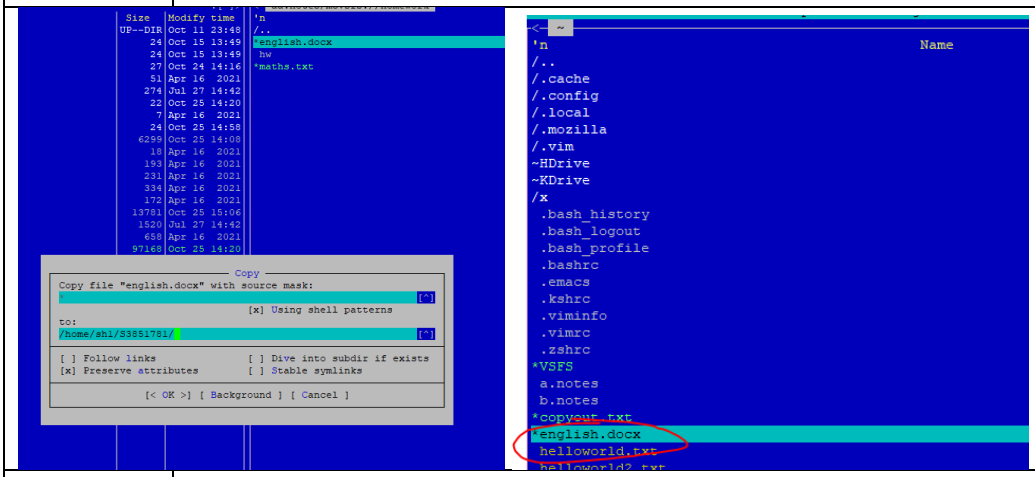
```
NOTES V1.0
@calendar.txt
 Tomorrow, I will go for a run.
@birthday
 Happy birthday Tom!
 I hope you had a nice day :)
 - Jeff
=homework/
@homework/maths.txt
 x+y=z
@homework/english.docx
 Shakespeare was not a real person.
=games/
@games/halo.exe
 binary stuff
=videos/
=videos/tvshows/
@videos/tvshows/simpsons.mp4
 video data
=videos/movies/
@videos/movies/interstellar.mp4
 video data
=d1/
@d1/ihavenosecrets
=d1/d2/
@d1/d2/imserious
=d1/d2/d3/
@d1/d2/d3/secrets
```

<<< abc.notes used to demonstrate the below

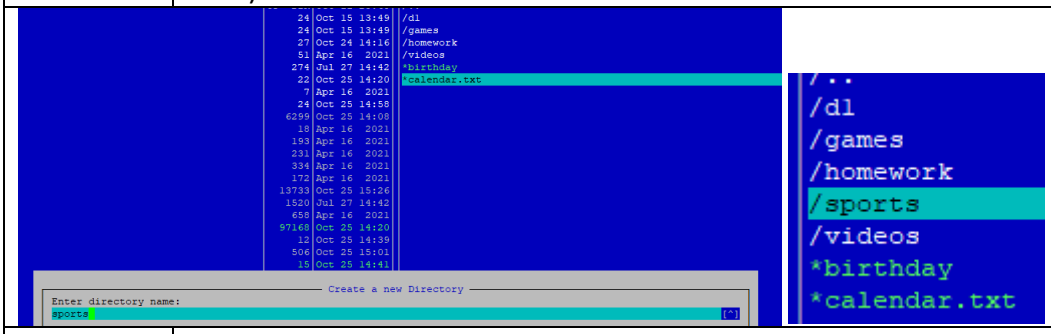
Interface command	Working status
list	+ you can view the virtual filesystem inside mc – files, subdirectories and all. - parsing error sometimes happens the first time you enter a .notes file.
	
copyin	+ you can copyin a file to the filesystem, even to subdirectories (that exist) - attempting to have VSFS create intermediate directories results in an error
	



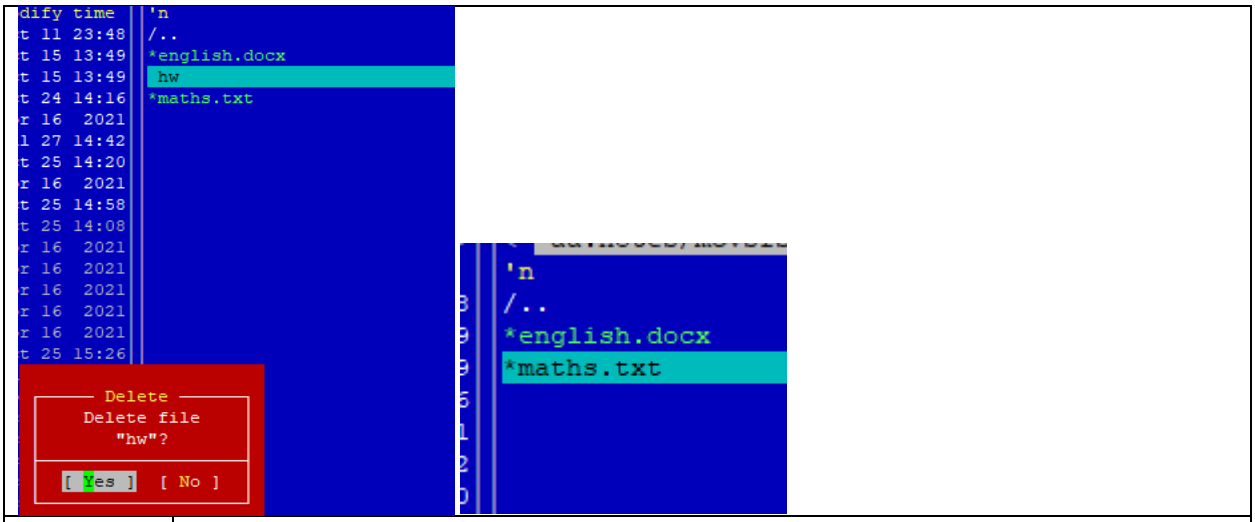
copyout	+ you can copyout a file to an external file, including to subdirectories (that exist) - intermediate directories cannot be created



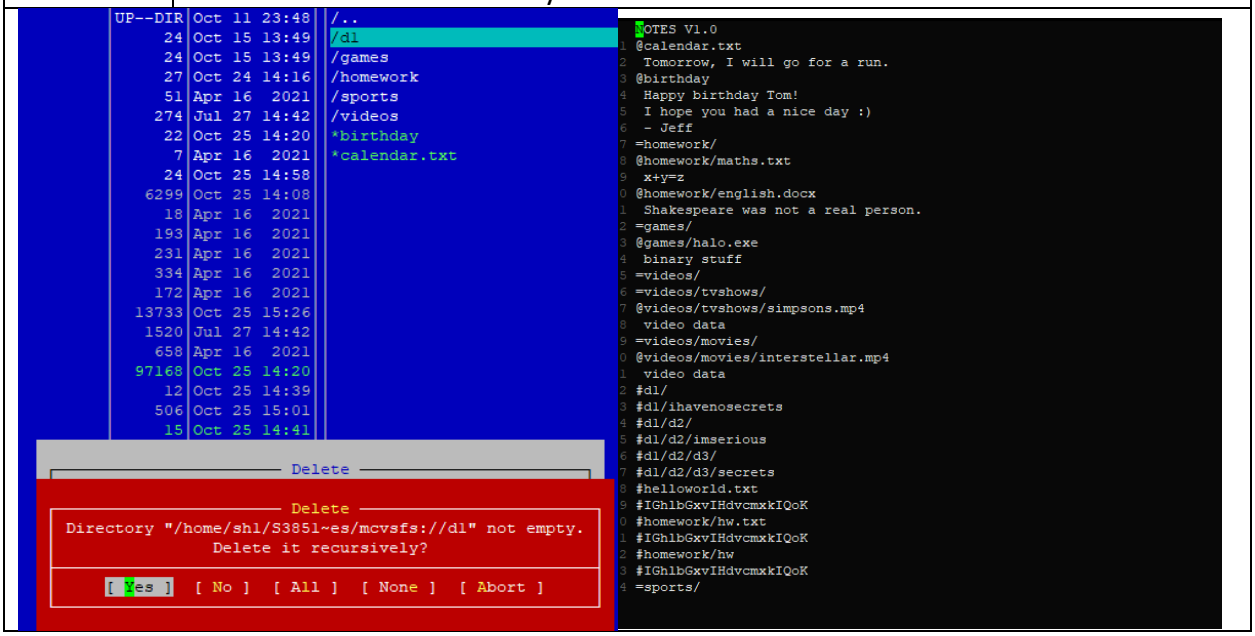
Mkdir	+ works as expected + you can make directories, and also subdirectories (of directories that already exist)



rm	+ works as expected + you may delete individual files



rmdir	+ works as expected + deletion occurs recursively for all subdirectories and files



Titbits on the Code : VSFS.cpp

Global variable “base64” is set to true by default. Make this false to not use base64 en/decoding.

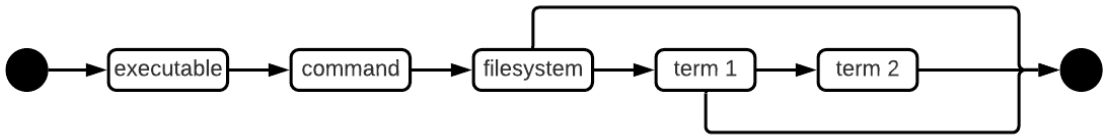
For the “size” parameter in list(), I count the number of characters in the content, and each character = 1 byte, as that is how C++ defines char pointers. I understand that this may not mean that the file will take up that number of bytes in a real file system – metadata and data structure overheads but I hope it is okay for the VSFS.

See first few lines of main(): For gz, if the file ends in .gz and exists, it is unzipped, goes through the program, then zipped again. Otherwise, it is treated as a .notes file.

REPORT QUESTIONS 1-5

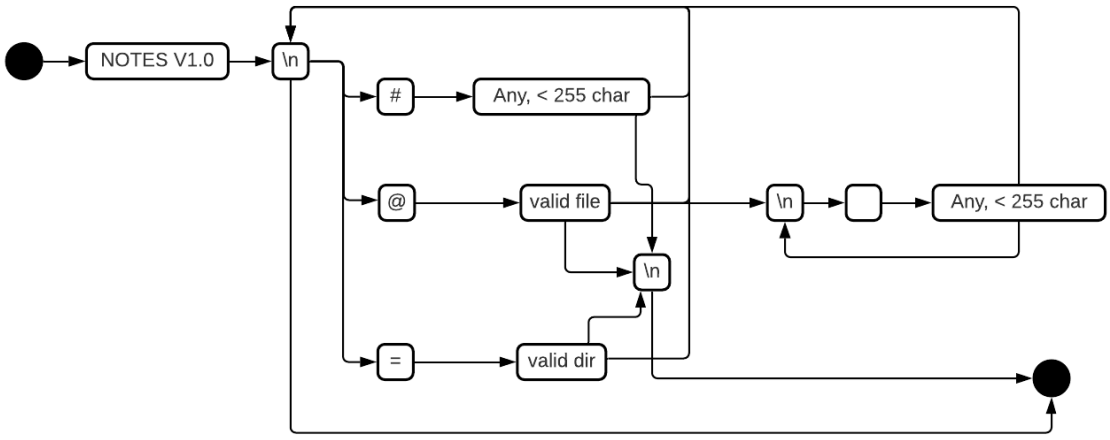
1.

a) Syntax Diagram Command Line



Term	Value
executable	VSFS (if executable is in the current path then ./VSFS)
command	*list, copyin, copyout, rm, rmdir, mkdir, defrag*
filesystem	A file with a .notes extension adhering to the conventions outline in the assignment specification
term 1	If command is: (1) copyin The external file to be copied in (2) copyout The internal file to be copied out (3) rm The file to be removed (4) rmdir The directory to be removed (5) mkdir The directory to be created
term 2	If command is: (1) copyin Name of the internal file copied to. (2) copyout Name of the external file copied to.

b) Syntax Diagram .notes



The notes file always stars with “NOTES V1.0”.

Each line following must begin with “#”, “@”, or “=”.

1. “#” can be followed by anything that is no more than 255 characters including the “\n” character.
2. “=” can be followed by a valid directory name, so not starting with “/” but ending in “/”
3. “@” can be followed by a valid file name, so not “.” “..” “/” and does not start or end with “/”. Immediately following this line there can be as many content lines as required beginning with a space “ “ that is no more than 255 characters including the “\n” character.

Each of 1, 2, and 3 end in a newline, and we either reach the end of the file, or we go again with a new instance of 1, 2, or 3.

2. Man Page

VSFS(1) User Commands VSFS(1)

NAME

VSFS – a very simple file system

SYNOPSIS

./VSFS [command]... filesystem ... term1 ... term2

DESCRIPTION

Performs standard filesystem operations on a '.notes' filesystem.

OPTIONS

list <filesystem>

Lists the contents of the filesystem in a ls -lR format. That is, all files, directories, subdirectories, and files of subdirectories are displayed in a hierarchical and alphabetical order.

copyin <filesystem> <external path> <internal path>

Copies the contents of the external file into the internal file path specified. If the internal file already exists, then it is deleted and replaced. Otherwise, it is created. Any intermediate directories are also created.

copyout <filesystem> <internal path> <external path>

Copies the contents of the internal file into the external file path specified. If the external file already exists, then it is replaced. Otherwise, it is created. Any intermediate directories are also created.

mkdir <filesystem> <directory path>

Creates a new directory (provided it does not already exist). The path of the subdirectories must already exist.

rm <filesystem> <internal path>

Deletes an existing file in the filesystem. Replaces the first character of the deleted records with a '#' that is ignored by the filesystem.

October 2021 VSFS(1)

rmdir <filesystem> <internal path>

Deletes an existing directory in the filesystem. If the directory has subdirectories and files, these file and directory records are also recursively deleted. Deleted directories have the first character replaced with '#' which is ignored by the filesystem.

defrag <filesystem>

Removes all records marked with '#' and sorts the filesystem in ls -lR order.

EXIT STATUS

0	**If okay**
22 (EINVAL)	If - Too few arguments for VSFS - Command given does not exist - Copyin: internal file name given is "/", ".", or ".." - Copyin: file starts or ends in "/" - Mkdir: directory name starts with "/" or does not end with "/"
200 (Custom)	If the .notes file is invalid, (see the rules outlined in the assignment specification).
2 (ENOENT)	If - Copyin: External file does not exist - Rm & Copyout: Internal file does not exist - Rm: Internal directory does not exist - Mkdir: The subdirectories for given directory path don't exist
1 (EPERM)	If there is a permissions issue with creating files, directories, or renaming files outside the filesystem i.e. Linux user permissions.
21 (EISDIR)	If directory already exists and user attempts to create the same directory.

AUTHOR

Written by Jeffrey Tan (S3851781)

3. Language Rationale

Language of choice: C++ for implementation, Shell for testing and MC compliance.

C++ is considered to be fast executing language as it is relatively low-level, allowing compilers to better optimise the code. Filesystems should perform operations as fast as possible to avoid bottlenecking the overall system.

C++ has a singular entry and exit point in main(). When writing the program, I ensured that exit code state was passed back and exited on the last line of main(). The arguments are taken in main, and the appropriate command's function is called. That function executes and returns a 0 if okay, or another code indicating an error. The return value is captured in main and exits with said value. The benefit to this approach (as opposed to calling exit() in each function) is that it makes flow of execution more maintainable, especially if one needed to add new functionality, with it's own set of potential errors.

C++ compiles directly into an executable. It is expected that the user call this program from the command line, so a single executable of the entire program is required.

Unit testing the program would require using command line calls with different arguments. Shell is one such language which does this. I chose Shell over other scripting languages because it is simple, readable, and behaves exactly like the command line when you want it to. Similarly, writing the script for MC compliance requires the program itself to be called to do the operations required by the extfs.d interface, I once again chose Shell as it was easiest to understand.

4. What would I change in order to…

a)

At the moment, almost all the commands in my VSFS require a sequential scan throughout, in some cases, the entire file when only a small section of information is required. This is particularly wasteful when the program must read through many lines of content to get to the next file/directory record.

The following describes the functionality that currently exists vs. the changes if a table were to exist.

Command	Current	Changes
Copyin	Scan filesystem to see if internal file exists and delete it if it exists.	Check the table to see if internal file exists. If yes, pass the index to the rm command for fseek() deletion.
Copyout	Scan filesystem for location of internal file, then copy contents.	Find the internal file in the table, and fseek() to the index, then copy the contents out.
Rm	Scan filesystem for location of internal file, then delete.	Find the internal file in the table, and fseek() to the index, then delete the file and its contents.

Rmdir	Scan filesystem for directory, then delete. For each subdirectory (recursive) do more scans to delete its children.	Use the table to build a recursive list of the indexes of records to be deleted. Pass these indexes to rm.
Mkdir	Scan filesystem to see if directory already exists.	Use the table to find whether the directory already exists.
List and defrag	Scan the entire file to build an ls -lR formatted output of the filesystem.	Use the table to build the output.

Of course, the table will need to be maintained. The following are ways to do so for each command.

Command	Changes
Copyin	Add row to the table with the new appended file record. If the old record is deleted, the table is updated in the rm command.
Copyout	Do nothing.
Rm	Erase the row in the table for the file to be deleted.
Rmdir	Erase rows for all directories and files deleted.
Mkdir	Add row to the table with new appended directory record.
List	Do nothing.
Defrag	Completely rebuild the table from scratch after the filesystem has been sorted.

b)

In a fixed length record system, we would have very similar functionality to the system that already exists. That is, for each operation, the file pointer may have to traverse the entire filesystem. The difference being that in the current implementation, the pointer must read through many lines of content records. If each record was of a fixed length, then the sequential scan can skip to just file and directory names.

Command	Current	Changes
Copyin	Scan every line	Scan every file/directory line by moving the head a fixed length down the filesystem.
Copyout		
Rm		
Rmdir		
Mkdir		
List and defrag		

5.

We could use .notes to store a database of human DNA. Each directory is the genome of one person. Inside that directory is 46 files, containing the genetic code (written in Gs Ts As and Cs) for each of the 23 pairs of chromosomes. One could use this for research in biology, or for criminal investigations. This .notes file will become very large as just one person's genome is estimated to be around 1.5GB. You would definitely want to do lossless compression on this filesystem.