

Programming Assignment 5

CSE 3320.002

Due: May 2nd, 2018 11:59PM

Description

In this assignment you will write the beginning of a dropbox-like file system. You will create an ‘in-memory’ file system. Your program will provide the user with ~33MB of “virtual” drive space. Users will have the ability to list the files currently in the file system, add files, and remove files. Files will not persist in the file system when the program exits.

1.0 File System Requirements

1.1: Your program will print out a prompt of mfs> when it is ready to accept input.

1.2: The following commands shall be supported:

Command	Usage	Description
put	put <filename>	Copy the file to the file system.
get	get <filename>	Retrieve the file from the file system.
get	get <filename> <newfilename>	Retrieve the file from the file system and place it in the file named <newfilename>.
del	del <filename>	Delete the file from the file system.
list	list	List the files in the file system.
df	df	Display the amount of disk space left in the file system.

1.3: Your program will be allocating 4226 blocks for the file system.

- 1.4: The file system shall support files up to 259,072 bytes in size.
- 1.5: The file system shall support up to 128 files.
- 1.6: The file system block size shall be 8096 bytes.
- 1.7: The file system shall use an indexed allocation scheme.
- 1.8: The file system shall support file names of up to 32 characters.
- 1.9: Supported file names shall only be alphanumeric with “.”. There shall be no restriction to how many characters appear before or after the “.”. There shall be support for files without a “.”
- 1.10: The file system will store the directory in the first (block 0) disk block.
- 1.11: The file system shall allocate blocks 1-129 for inodes
- 1.12 The directory structure shall be a single level hierarchy with no subdirectories.

2.0 Command Requirements

2.1 The put command - The put command shall allow the user to put a new file into the file system.

2.1.1 The command shall take the form:

```
put <filename>
```

2.1.2 If the filename is too long an error will be returned stating:

```
put error: File name too long.
```

2.1.3 If there is not enough disk space for the file an error will be returned stating:

```
put error: Not enough disk space.
```

2.2 The get command - The get command shall allow the user to retrieve a file from the file system and place it in the current working directory.

2.2.1 The command shall take the form:

```
get <filename>
```

```
mfs> get main.c
mfs>
```

and

get <filename> <newfilename>

```
mfs> get main.c main.c.backup  
mfs>
```

2.2.2 If no new filename is specified the get command shall copy the file to the current working directory using the old filename.

2.2.3 If the file does not exist in the file system an error will be printed that states:

get error: File not found.

```
mfs> get doesntexist.txt  
mfs> get error: File not found.
```

2.3 The del command - The del command shall allow the user to delete a file from the file system

```
mfs> del main.cc  
mfs> del error: File not found.
```

2.3.1 If the file does not exist in the file system an error will be printed that states:

del error: File not found.

```
mfs> del main.cc  
mfs> del error: File not found.
```

2.3.2 If the file does exist in the file system it shall be deleted and all the space available for additional files.

2.3 The list command - The list command shall display all the files in the file system, their size in bytes and the time they were added to the file system

```
mfs> list
13764 Oct 28 21:13 a.out
20465 Oct 28 21:13 main.c
134   Oct 27 19:23 input.txt
mfs>
```

2.3.1 If no files are in the file system a message shall be printed:

list: No files found.

```
mfs> list
list: No files found.
mfs>
```

2.4 The df command - The df command shall display the amount of free space in the file system in bytes,

```
mfs> df
8423718 bytes free.
mfs>
```

3.0 Nonfunctional Requirements

3.1: You may code your solution in C or C++.

3.2: C files shall end in .c . C++ files shall end in .cpp

3.3: Your source files must be ASCII text files. No binary submissions will be accepted.

3.4: Tabs or spaces shall be used to indent the code. Your code must use one or the other. All indentation must be consistent.

3.5: No line of code shall exceed 100 characters.

3.6: Each source code file shall have the following header filled out:

3.7: All code must be well commented. This means descriptive comments that tell the intent of the code, not just what the code is executing. The following explains the intent:

3.8: Keep your curly brace placement consistent. If you place curly braces on a new line , always place curly braces on a new end. Don't mix end line brace placement with new line brace placement.

3.9: Each function should have a header that describes its name, any parameters expected, any return values, as well as a description of what the function does. For example :

Grading

The assignment will be graded out of 100 points. Compiler warnings are there to tell you something is not correct. Pay attention to them and you will save yourself a lot of late nights debugging code. Code that does not compile will earn 0.

Category	Points
put <filename>	15
get <filename>	15
get <filename> <filename>	15
list	15
df	5
Functional requirements met	5
inodes / directory correct	20
Output format as specified/ No extra output	5
Non-functional requirements met	5

Submitting your Assignment

Submit your assignment on blackboard

Academic Integrity

This assignment must be 100% your group's work. No code may be copied from friends, previous students, books, web pages, etc. All code submitted is checked against a database of previous semester's graded assignments, current student's code and common web sources. Code that is copied from an external source will result in a 0 for the assignment and referral to the Office of Student Conduct.

Hints and Suggestions

Reuse the code from the mav shell. Your parser and main loop logic are already done. It will allow you to concentrate on the new functionality you have to implement and not on reimplementing code you've already written.

Look at the man page for stat. `man 2 stat`. You will need the metadata it gives you.

I have placed a file call `main.c` under this assignment on blackboard. That code will read a file, place the data into blocks, then open a new file and write those blocks to a new file.

Compile it with:

```
gcc main.c -o fcopy
```

You can then run it by:

```
./fcopy <filename1> <filename2>
```

This code shows how to take a file, chop it up into block size chunks, store it in a “disk block”-like data structure and then reform the file. Modify it to select free blocks.

This code is how your put and get command should work. USE IT.

This program WILL take the entire time allotted.