Project 3 Specification

FAT32 File System Utility

Assigned: October 30, 2015

Due: November 30, 11:59 pm, 2015

You can use the reminder of the slack days. -10 late penalty for each 24-hour period after the due time. Absolutely no submission after December 6 11:59 pm.

Language restrictions: C only

Additional restrictions: system() and exec() system calls may not be used

Purpose:

The purpose of this project is to familiarize you with three concepts: basic file-system design and implementation, file-system image testing, and data serialization/de-serialization. You will need to understand various aspects of the FAT32 file system such as cluster-based storage, FAT tables, sectors, directory structure, and byte-ordering (endianness). You will also be introduced to mounting and un-mounting of file system images onto a running Linux system, data serialization (converting data structures into raw bytes for storage or network transmissions), and de-serialization (converting serialized bytes into data structures again).

Problem Statement:

For this project, you will design and implement a simple, user-space, shell-like utility that is capable of interpreting a FAT32 file system image. The program must understand the basic commands to manipulate the given file system image. The utility must not corrupt the file system image and should be robust. You may not reuse kernel file system code, and you may not copy code from other file system utilities (or from anywhere else over the internet, to be precise).

Project Tasks:

You are tasked with writing a program that supports file system commands. For good modular coding design, please implement each command in a separate function. Please implement the following functionality:

- *open <FILE_NAME> <MODE>*

Opens a file named *FILE_NAME* in the present working directory. A file can only be read from or written to if it is opened first. You may need to maintain a table of opened files and add *FILE_NAME* to it when *open* function is called.

MODE may be:

- *r* – read-only
- *w* – write-only
- *rw* – read and write
- *wr* – read and write

Return an error if the file is already opened.

- *close <FILE_NAME>*

  Closes a file. Return an error if the file is already closed. Depending on your implementation, you may need to remove *FILE_NAME* from the table of opened files.

- *create <FILE_NAME>*

  Creates a file in the present working directory of 0 bytes with name *FILE_NAME*. Return an error if a file with that name already exists.

- *rm <FILE_NAME>*

  Deletes *FILE_NAME* from the current working directory. Return an error if *FILE_NAME* does not exist. This only removes the link from a directory to a file, not the actual data. (That is, the file no longer shows in the output of *ls,* and the space is reclaimed).

- *size <FILE_NAME>*

  Prints the size of file *FILE_NAME* in the current working directory, in bytes. Return an error if FILE_NAME does not exist.

- *cd <DIR_NAME>*

  Changes the current working directory to *DIR_NAME*. Return an error if the specified directory does not exist. Should support "." (here) and ".." (up one directory) directories.

- *ls <DIR_NAME>*

  Lists the contents of *DIR_NAME*, including the "." (here) and ".." (up one directory) directories. Return an error if the specified directory does not exist

- *mkdir <DIR_NAME>*

Creates a new directory with the name *DIR_NAME*. Return an error if the directory already exists or if a file already exists by that name.

- *rmdir <DIR_NAME>*

  Removes a directory called *DIR_NAME*. Return an error if the directory is not empty or the directory does not exist. (Directory is empty if it contains only "." And "..").

- *read FILE_NAME POSITION NUM_BYTES*

  Reads from a file *FILE_NAME*, starting at *POSITION*, and prints *NUM-BYTES*. Return an error when trying to read from an unopened file or from a file opened only for writing, or if *POSITION* is greater than the size of the file *FILE_NAME.*

- *write FILE_NAME POSITION NUM_BYTES STRING*

  Writes to a file *FILE_NAME*, starting at *POSITION*, for *NUM-BYTES*, the quote-enclosed *STRING*. Over-write the existing bytes at the specified position. Return an error when trying to write an unopened file, or a file opened only for reading.

  If *POSITION* is outside the bounds of the file, grow the file enough to make *POSITION* valid. File holes are supported. If the length of the data exceeds the bounds of the file, grow the file as necessary.

In case of all the functions above, print an appropriate error message if the function returns an error.

Allowed Assumptions:

- File and directory names will not contain spaces.
- You may assume that *STRING* always begins and ends with *QUOTES*
- *FILE_NAME* and *DIR_NAME* will only contain the names, not multi-level paths. For example, *"create afile"* will create a file named afile in the current directory. It does not have to support commands like *"create adirectory/afile"*
- Unless specified, you may choose any reasonable data type for the arguments and return values for your functions.

Grading Criteria:

a) **Documentation:** 30%
   - 5: A Readme file listing group members' names, fsu ids, contents of the folder, and how to compile and execute.
   - 5: Makefile
   - 10: A report explaining how the program works, the basic modules, important data structures and variables, any known limitations/errors/incompleteness. Basically, make the grader's job easier.
   - 5: Code: Readable
   - 5: Code: Naming

b) **Coding:** 70%
   - 55: open, close, create, rm, size, cd, ls, mkdir, rmdir, read, write (5 points each)
   - 15: input-output handling

c) **Deductions:**
   - 70: for not compiling (So make sure it compiles at least.)
   - 10: for crashing at any point while running
   - 20: for corrupting the image file

Submission and Grading Guidelines:

1. This is a group project. Each group will have 3 people.

2. One person from each team will submit a tar and gzip'd archive containing the submission to Blackboard. He source code and Makefile should be placed inside a "src" directory (Do not submit binaries). Other files (e.g. Readme) should be placed at the root of the archive. Upload your submission using Blackboard and verify that you can download and extract it successfully. The filename of the submission file is to be formatted as:

   P3_<member_name1>_<member_name2>_<member_name3>.tar.gz

   At a minimum, the submitted archive should contain:

   - Documentation
   - Source File
   - Makefile

3. For this project, no demonstration is necessary. The TA will contact you if he thinks a meeting is needed.

4. Everyone in the group will get the same grade. Because it is a group project, everyone is expected to share the workload equally. If anyone has a major disagreement about workload distribution, (s)he needs to contact the TA beforehand.