**Project 3 Specification**

**FAT32 File System Utility**

**Assigned:** March 30, 2020

**Due:** April 24(part 1), May 1 (part 2)

**Language Restrictions:** Use C or C++ for full credit. If you do the project in Java or Python, you can earn a maximum of 90% credit. The finished project *must* run on the class server.

**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, 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 (i.e., converting data structures into raw bytes for storage or network transmissions), and de-serialization (i.e., converting serialized bytes into data structures). Familiarity with these concepts is necessary for advanced file-system programming.

**Problem Statement**

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 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.

**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. Implement the following functionality:

* info

Description: prints out information about the following fields in both hex and base 10:

o BPB\_BytesPerSec

o BPB\_SecPerClus

o BPB\_RsvdSecCnt

o BPB\_NumFATS

o BPB\_FATSz32

* stat <FILE\_NAME/DIR\_NAME>

Description: prints the sizeof the fileor directory name, the attributes of the file or directory name, and the first cluster number of the file or directory name if it is in the present working directory. Return an error if FILE\_NAME/DIR\_NAME does ot exist. (Note: The size of a directory will always be zero.)

* size <FILE\_NAME>

Description: prints the size of file FILE\_NAME in the present working directory. Log an error if FILE\_NAME does not exist.

* cd <DIR\_NAME>

Description: changes the present working directory to DIR\_NAME. Log an error if the directory does not exist. DIR\_NAME may be “.” (here) and “..” (up one directory). You don't have to handle a path longer than one directory.

* ls <DIR\_NAME>

Description: lists the contents of DIR\_NAME, including “.” and “..”.

* read FILE\_NAME POSITION NUM\_BYTES

Description: reads from a file named FILE\_NAME, starting at POSITION, and prints NUM\_BYTES. Return an error when trying to read an unopened file.

* volume

Description: Prints the volume name of the file system image. If there is a volume name it will be found in the root directory. If there is no volume name, print “Error: volume name not found.”

* mkdir <SUBDIR\_NAME>

Description: make a new subdirectory in the current directory. This may require the allocation of additional space if the current directory has all entries full in all sectors.

It is acceptable for you to only add the subdirectory if there is space to do so without allocating an additional sector. If you are unable to create the subdirectory for any reason, you must print an error message on stderr (fd 2).

* rmdir <SUBDIR\_NAME>

Description: delete a subdirectory in the current directory, but only if it is empty! If you are unable to delete the subdirectory for any reason, you must print an error message on stderr (fd 2). Follow the FAT32 rules for deleting stuff; do not do a full overwrite or zero anything out!

* quit

Quit the utility.

**Allowed Assumptions**

* File and directory names will not contain spaces.
* Because you are not writing any data, you can treat the FAT32 free list superficially.
* Your C/C++ program ***must*** work on Ubuntu Linux 16.04. If it does not, you can only earn a maximum of 50%.
* Java and Python programs must work with the latest Java 8/9 and Python 3.

**Create a README file**

Please create a README text file that contains the following:

* The names of all the members in your group
* A listing of all files/directories in your submission and a brief description of each
* Instructions for compiling your programs
* Instructions for running your programs/scripts
* Any challenges you encountered along the way
* Any sources you used to help you write your programs/scripts

**Part 1 Submission**

Proper endian-ness functions for numbers must be used. Also the following commands should be completed and work with the fat32.img provided to you:

* info
* ls
* stat

**Grading**

Please refer to the grading sheet.

**Submission Procedure**

You must zip up your README file, your source code, and anything that is needed to compile it (such as a Makefile), and submit the entire zip to Canvas by the due date.

**Hints:**

Since this is a real filesystem, you can mount it as such in either MacOS or Linux. For MacOS, just click on the fat32.img file to open it in Finder.

On your own Linux virtual machine, you can do the following:

sudo mount -o loop fat32.img /mnt

cd /mnt

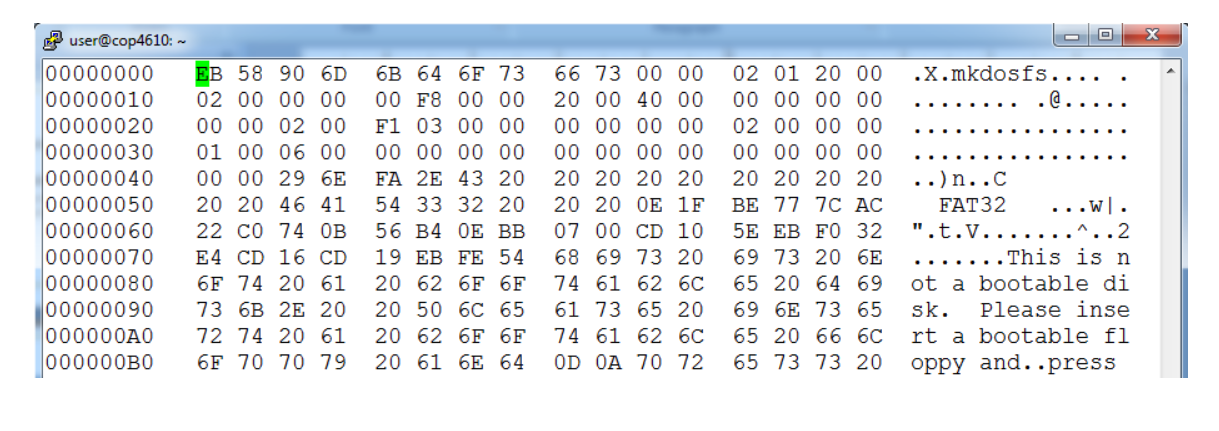
Once the file is mounted, you can go into the /mnt directory and issue all your normal file system commands like: ls, cat, cd, .. That will show you what the content is all about.

To verify the file format you will need a good hexadecimal editor, since the file format is binary. hexedit is an editor that is available on Mac, Windows, and Linux. You can download it from <https://sourceforge.net/projects/hexedit/> for Mac and Windows. You should be able to add it to a Linux VM using your normal sequence:

sudo apt-get update

sudo apt-get install hexedit

Hexedit will display your file contents in 3 columns. The offset from the beginning of the file is in the left columns (in base 16), a hex dump of the file in the middle column, and an ASCII (text) dump in the right column with non-printable characters replaced with periods.



Remember that integer values in the file are writing in little-endian order.