Assignment 3
FAT32 File System Utility

Language Restrictions: Java 11
Additional Restrictions: system() and exec*() system calls may not be used
Work Environment: You may work in groups of no more than two students per group.

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.

Rules

The only file system-related source you may use is the FAT32 specification attached to the assignment description in Canvas. You may search for information about how to solve general problems in Java (setting up a Java List, etc.), but you MAY NOT search for anything related to solving file system problems or working with file systems.

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.

Starting your Utility

Do not hard-code the image name into your program. Once your utility is running, display a prompt of the form `WorkingDirectory]` so that the user knows that he or she is now in the utility. The current working directory should be displayed in the utility prompt relative to the image's root directory for ease of debugging and testing.

```
Java fat32_reader fat32.img
/]
```

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:

- stop
- info
- ls
- stat
- size
- cd
- read

- **`stop`**

  Description: exits your shell-like utility

- **`info`**

  Description: prints out information about the following fields in both hex and base 10. Be careful to use the proper endian-ness:
  - `BPB_BytesPerSec`
  - `BPB_SecPerClus`
  - `BPB_RsvdSecCnt`
  - `BPB_NumFATS`
  - `BPB_FATSz32`

  Sample execution:

```
/] info
BPB_BytesPerSec is 0x200, 512
BPB_SecPerClus is 0x1, 1
BPB_RsvdSecCnt is 0x20, 32
BPB_NumFATs is 0x2, 2
BPB_FATSz32 is 0x3f1, 1009
/]
```

  Note: Do not assume and hard-code these values into your reader! Your reader will be tested with images with different sector sizes, different number of sectors per cluster, etc., and everything should still work correctly.

- **`stat <FILE_NAME/DIR_NAME>`**

  Description: For the file or directory at the relative or absolute path specified in FILE_NAME or DIR_NAME, prints the size of the file or directory, the attributes of the file or directory, and the first cluster number of the file or directory. Return an error if FILE_NAME/DIR_NAME does not exist (see example below). (Note: The size of a directory will always be zero.)

  If a file has more than one `Attributes`, print them space delimited, in descending order of the bit value of the attributes.

  Sample execution:

  Directory or file exists:
  ```
  /] stat CONST.TXT
  Size is 45119
  Attributes ATTR_ARCHIVE
  Next cluster number is 0x0004
  /]
  ```

  Directory or file doesn't exist:
  ```
  /] stat NOTHERE.TXT
  Error: file/directory does not exist
  /]
  ```

- **`ls <DIR_NAME>`**

  Description: For the directory at the relative or absolute path specified in DIR_NAME, lists the contents of `DIR_NAME`, including "." and "..", and including hidden files (in other words, it should behave like the real "ls -a"). Display an error message if DIR_NAME is not a directory.

  Like the "real" **ls -a**, "." and ".." are shown for all directories, even the root directory (despite the fact that ".." is meaningless for the root directory).

  Like the "real" **ls -a**, your output should be alphabetically sorted

  NOTE: This assignment has you "stuck" in the root directory because there's no **cd** command. Both of these should work:

  `ls DIR` (relative path from where we are (the root directory))
  `ls /DIR` (absolute path)

  Sample execution:

  ```
  /] ls .
  . .. A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
  /]

  /] ls bob.txt
  Error: bob.txt is not a directory
  /]
  ```

- **`size <FILE_NAME>`**

  Description: For the file at the relative or absolute path specified in FILE_NAME, prints the size of file. Return an error if `FILE_NAME` does not exist or is not a file.

  Sample execution:

  ```
  /] size FOLDER1/FILE.TXT
  Size of FOLDER1/FILE.TXT is 42376 bytes
  /]

  /] size /DIR/NOT_HERE.TXT
  Error: /DIR/NOT_HERE.TXT is not a file
  /]

  /] size FOLDER1
  Error: FOLDER1 is not a file
  /]
  ```

- **`cd <DIR_NAME>`**

  Description: For the directory at the relative or absolute path specified in DIR_NAME, changes the current directory to DIR_NAME. The prompt is updated to show the new current directory. Return an error if `DIR_NAME` does not exist or is not a directory.

  Sample execution:

  ```
  /] cd FOLDER1
  /FOLDER1]

  /FOLDER1] cd /FOLDER2
  /FOLDER2]

  /] cd /FOLDER1/FILE.TXT
  Error: /FOLDER1/FILE.TXT is not a directory
  /]

  /] cd MSNGFLDR
  Error: MSNGFLDR is not a directory
  /]
  ```

- **`read <FILE_NAME> <OFFSET> <NUMBYTES>`**

  Description: For the file at the relative or absolute path specified in FILE_NAME, reads from the file starting `OFFSET` bytes from the beginning of the file and prints `NUM_BYTES` bytes of the file's contents, interpreted as ASCII text (for each byte, if the byte is less than decimal 127, print the corresponding ascii character. Else, print " 0xNN ", where NN is the hex value of the byte).

  Return an error when trying to read an unopened file, a nonexistent file, or read data outside the file.

  Sample execution:

  Successful read
  ```
  /] read CONST.TXT 0 15
  Provided by USC
  /]
  ```

  Unsuccessful reads
  ```
  /] read 10BYTES.TXT 5 5
  Error: attempt to read data outside of file bounds
  /]

  /] read 10BYTES.TXT -1 5
  Error: OFFSET must be a positive value
  /]

  /] read 10BYTES.TXT 1 -5
  Error: NUM_BYTES must be a greater than zero
  /]

  /] read 10BYTES.TXT 1 0
  Error: NUM_BYTES must be a greater than zero
  /]

  /] read NOTOPEN.TXT 5 5
  Error: file is not open
  /]

  /] read FOLDER1 5 5
  Error: FOLDER1 is not a file
  /]

  /] read /DIR/NOT_HERE.TXT 5 5
  Error: /DIR/NOT_HERE.TXT is not a file
  /]
  ```

Allowed Assumptions

- File and directory names will not contain spaces.

Suggestions

- For two of the three commands, the ability to drill down through a path and return the correct directory entry needs to be implemented. It may be efficient to have one member of the team implement this while the other team member implement the functionality listed above, testing on files in the root directory until the “path traverser” is complete. This will require some design work to insure that the “path traverser” can be easily “dropped in” to the functions above when it becomes available.

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

Submission Procedure

Zip up your README file, your source code, and anything that is needed to compile it, and submit the entire zip to Canvas by the due date.

Your submission will be built and tested on the CompOrg server, so be sure it works there.

Notes

- To indicate end-of-file, the entry in the FAT can be <u>any value</u> from 0x0FFFFFF8 to 0x0FFFFFFF
- When handling arguments for subdirectories, use a forward slash (UNIX-style)
- *short file names* are 1 byte per character in ASCII / UTF-8 encoding.
- FAT32 is case-insensitive, so commands on a file or directory that exists, but in a different case than that entered by the user, should still work.
- Commands that receive a deleted file as an argument should return the generic error messages shown above
- Hex editors:
  - MacOS: HexEdit or HexFiend
  - Windows: HxD
- For this assignment, we are ignoring FAT32 long file names, so:
  - For output of ls, etc., you'll be outputting the FAT32 short file names, which are stored in all caps.
- FAT32 file systems are composed of sectors, each of which has a fixed size in bytes. It is <u>not</u> the case that the file system has to completely occupy its last sector.