Assignment 4
FAT32 File System Utility

Language Restrictions: Java openjdk 17.0.10
Additional Restrictions: File-system specific calls that do portions of the assignment 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.

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 a file or directory in the current directory, 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 using an eight-digit hex value as shown below. 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.

  If a file has no `Attributes` set, print `Attributes NONE`

  Sample execution:

  Directory or file exists:

```
/] stat CONST.TXT
Size is 45119
Attributes ATTR_ARCHIVE
Next cluster number is 0x00000004
/]
```

  Directory or file doesn't exist:

```
/] stat NOTHERE.TXT
Error: file/directory does not exist
/]
```

- **ls**

  Description: For the current directory, lists the contents of `the directory`, including "." and "..", and including hidden files (in other words, it should behave like the real "ls -a").

  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

  Both of these types should work:

  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
/]
```

- **size <FILE_NAME>**

  Description: For a file in the current directory, 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 FILE.TXT
Size of FILE.TXT is 42376 bytes
/]

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

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

- **cd <DIR_NAME>**

  Description: From the current directory, changes to the subdirectory DIR_NAME (including current directory using "." and up one directory using ".."). You **do not** need to handle something like "cd dir1/dir2". 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
/FOLDER1/FOLDER2]

/FOLDER1/FOLDER2] cd .
/FOLDER1/FOLDER2]

/FOLDER1/FOLDER2] cd ..
/FOLDER1]

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

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

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

  Description: For a file in the current directory, 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 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 6
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 greater than zero
/]

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


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

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

Allowed Assumptions

- File and directory names will not contain spaces.

Testing Procedure

My script will do the following in your submission directory:

```
javac *.java
java fat32_reader fat32.img < inputcommands.txt > readeroutput.txt
```

Submission Procedure

Zip up your java .class files in the root directory of your zip file and submit the zip file to Canvas by the due date.

Your submission will be tested on the VM, so be sure it works there.

Notes

- To indicate end-of-file, the entry in the FAT can be any value 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 not the case that the file system has to completely occupy its last sector.
- Java's integer types are signed. There are no unsigned types available in Java. Be sure to pick data types that won't overflow.
- If you want to explore fat32.img using the mount command, and mount returns to the error " mount failed: Operation not permitted", run the mount command using the flag "vers=2.0".
- Your submission will only be tested using valid FAT32 image files.
- Attribute bits are your friends, like when you're trying to figure out if an entry is a file, subdirectory, or something else.
- The disk image can easily be larger than available memory, so you should **not** read the entire image file into an array. Instead, do something like read the file allocation table and the boot sector into memory, then read individual data sectors into memory as needed.
- Your program is a shell, so the user can issue commands (or not) in whatever sequence they desire.

- A goal for this assignment is to build values from multiple bytes using your knowledge of endian. Therefore, you may use java libraries that act on individual bytes, but **not** libraries that convert multiple bytes into ints, longs, etc.
- . and .. are valid for any command that deals with directories. They could also be used with commands that only work on files, in which case the error shown in the specification should be returned.
- I will test your submission with image files containing file systems with different bytes-per-sector, FAT size, etc. All the values in the boot sector may change, so it's important not to hard code those values.
- For filenames in FAT32, the first 8 bytes, filled in from left to right, contain the up-to-eight characters of the name before the period. The next three bytes, filled in from left to right, contain the up-to-three characters of the name after the period. When your shell prints a filename to the screen, the period between the filename and extension, not stored in the name field, is rendered to the screen only if there are characters in those last three bytes.