CSc 360: Operating Systems (Summer 2012)

Assignment 3: A Simple File System (SFS)

Due Date: 3:30 pm, July 27, 2012

1 Introduction

In this assignment, you will implement utilities that perform operations on a simple file system, FAT12, used by MS-DOS.

1.1 Sample File Systems

You will be given a file system image: `disk.IMA` for self-testing, but your submission may be tested against other disk images following the same specification.

You should get comfortable examining the raw, binary data in the file system images using the program `xxd`.

2 Requirements

2.1 Part I

In part I, you will write a program that displays information about the file system. In order to complete part I, you will need to understand the file system structure of MS-DOS, including FAT Partition Boot Sector, FAT File Allocation Table, FAT Root Folder, FAT Folder Structure, and so on.

Your program for part I will be invoked as follows:

```
./diskinfo disk.IMA
```

Your output should include the following information:

```
OS Name:
Label of the disk:
Total size of the disk:
Free size of the disk:

==============
The number of files in the root directory (not including subdirectories):

=============
Number of FAT copies:
Sectors per FAT:
```

2.2 Part II

In part II, you will write a program, with the routines already implemented for part I, that displays the contents of the **root** directory in the file system.

Your program for part II will be invoked as follows:

```
./disklist disk.IMA
```

The directory listing should be **formatted** as follows:

1. The first column will contain:

   (a) `F` for regular files, or
   (b) `D` for directories;

   followed by a single space

2. then 10 characters to show the file size in bytes, followed by a single space

3. then 20 characters for the file name, followed by a single space

4. then the file creation date and creation time.

2.3 Part III

In part III, you will write a program that copies a file from the file system to the current directory in Linux. If the specified file is not found in the root directory of the file system, you should output the message `File not found.` and exit.

Your program for part III will be invoked as follows:

```
./diskget disk.IMA ANS1.PDF
```

If your code runs correctly, ANS1.PDF should be copied to your current Linux directory, and you should be able to read the content of ANS1.PDF.

2.4 Part IV

You will write a program that copies a file from the current Linux directory into the root directory of the file system. If the specified file is not found, you should output the message `File not found.` on a single line and exit. If the file system does not have enough free space to store the file, you should output the message `No enough free space in the disk image.` and exit.

Your program will be invoked as follows:

```
./diskput disk.IMA foo.txt
```

Note that a correct execution should update FAT and related allocation information in disk.IMA accordingly. To validate, you can use `diskget` implemented in Part III to check if you can correctly read `foo.txt` from the file system.

3 File System Specification

The specification of FAT12 and related information could be found in Connex.

4 Byte Ordering

Different hardware architectures store multi-byte data (like integers) in different orders. Consider the large integer: 0xDEADBEEF

On the Intel architecture (Little Endian), it would be stored in memory as:

`EF BE AD DE`

On the PowerPC (Big Endian), it would be stored in memory as:

`DE AD BE EF`

Since the FAT was developed for IBM PC machines, the data storage is in Little Endian format, i.e. the least significant byte is placed in the lowest address. This will mean that you have to convert all your integer values to Little Endian format before writing them to disk.

5   Submission Requirements

What to hand in: You need to hand in a `.tar.gz` file containing all your source code and a Makefile that produces the executables (i.e., diskinfo, disklist, diskget, and diskput).

The file is submitted through `connex.csc.uvic.ca` site.

6   Marking Scheme

7   Marking

We will mark your code submission based on correct functionality and code quality.

7.0.1   Functionality

1. Your programs must correctly output the required information in Part I, II, and III. One sample disk image is provided to you for self-learning and self-testing. Nevertheless, your code may be tested with other disk images of the same file system. We will not test your code with a damaged disk image. We will not disclose all test files before the final submission. This is very common in software engineering.

2. You are required to catch return errors of important function calls, especially when a return error may result in the logic error or malfunctioning of your program.

7.0.2   Code Quality

We cannot specify completely the coding style that we would like to see but it includes the following:

1. Proper decomposition of a program into subroutines (and multiple source code files when necessary)—A 1000 line C program as a single routine would fail this criterion.

2. Comment—judiciously, but not profusely. Comments also serve to help a marker, in addition to yourself. To further elaborate:

   (a) Your favorite quote from Star Wars or Douglas Adams' Hitch-hiker's Guide to the Galaxy does not count as comments. In fact, they simply count as anti-comments, and will result in a loss of marks.

   (b) Comment your code in English. It is the official language of this university.

3. Proper variable names—`leia` is not a good variable name, it never was and never will be.

4. Small number of global variables, if any. Most programs need a very small number of global variables, if any. (If you have a global variable named `temp`, think again.)

5. **The return values from all system calls and function calls listed in the assignment specification should be checked and all values should be dealt with appropriately.**

7.1   Detailed Test Plan

The detailed test plan for the code submission is as follows.

Components	Weight
Make file	2
diskinfo	8
disklist	9
diskget	10
diskput	10
Readme	1
Total Weight	40

Note that the score of 40/40 means that you obtain 100% of this assignment.

8 Warning

1. You are required to use C. Any other language is **not acceptable.**

2. Your code should output the required information specified in Parts I, II, III, and IV. Failing to do so will result in the deduction of scores.

3. You should use the Linux machines in ECS242 to test your work.

4. The due date of this assignment is set to be very late.
   THERE WILL BE NO FURTHER EXTENSION!

9 Plagiarism

This assignment is to be done individually. You are encouraged to discuss the design of the solution and the FAT 12 specification with your classmates, but each student must implement their own assignment.