Computer Science 360
Introduction to Operating Systems
Spring 2023

Programming Assignment 3

A Simple File System (SFS)

Code Due: Thursday, April 6, 11:55 pm by Brightspace submission
(Late submissions **not** accepted)

Goals

This assignment is designed to help you:

1. Understand the construction of file systems,
2. build a simple file system,
3. perform operations on file systems.

You are required to implement your solution in C (other languages are not allowed). Your work will be tested on linux.csc.uvic.ca.

Note: linux.csc.uvic.ca is a particular machine at the UVic Department of Computer Science. It does not mean "any Linux machine" at UVic. Even more importantly, it does not mean any "Unix-like" machine, such as a Mac OS X machine—many students have developed their programs for their Mac OS X laptops only to find that their code works differently on linux.csc.uvic.ca resulting in a substantial loss of marks.

You can remote access linux.csc.uvic.ca by ssh username@linux.csc.uvic.ca. SSH clients are available for a wide variety of operating systems including Linux, Mac OS and Windows.

Introduction

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

Sample File System

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.

Requirements

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. For example, your program for part I will be invoked as follows:

```
linux.csc.uvic.ca:/home/user$ ./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 disk:
Number of FAT copies:
Sectors per FAT:

Note:

1) when you list the total number of files in the disk, you should count all files in the root directory and files in all subdirectories. A subdirectory name is not considered as a normal file name and thus should not be counted.
2) For a directory entry, if the field of "First Logical Cluster" is 0 or 1, then this directory entry should not be counted.
3) Total size of the disk = total sector count * bytes per sector
4) Free size of the disk = total number of sectors that are unused (i.e., 0x000 in an FAT entry means unused) * bytes per sector. Remember that the first two entries in FAT are reserved.

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 and all sub-directories (possibly multi-layers) in the file system. Your program for part II will be invoked as follows:

```
linux.csc.uvic.ca:/home/user$ ./disklist disk.IMA
```

Starting from the root directory, the directory listing should be formatted as follows:

1) Directory Name, followed by a line break, followed by "==========", followed by a line break.
2) List of files or subdirectories:
    a. The first column will contain:
        i. F for regular files, or
        ii. D for directories;

        followed by a single space
    b. then 10 characters to show the file size in bytes, followed by a single space
    c. then 20 characters for the file name, followed by a single space
    d. then the file creation date and creation time.
    e. then a line break.

Note:

For a directory entry, if the field of "First Logical Cluster" is 0 or 1, then this directory entry should be skipped and not listed.

Part III

In part III, you will write a program that copies a file from the **root directory** of the file system to the current directory in Linux. If the specified file cannot be 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:

```
linux.csc.uvic.ca:/home/user$ ./diskget disk.IMA <filename>
```

If your code runs correctly, the file <*filename*> should be copied to your current Linux directory, and you should be able to read the content of copied file.

Part IV

You will write a program that copies a file from the current Linux directory into specified directory (i.e., the root directory or a subdirectory) of the file system. If the specified file is not found, you should output the message File not found. and exit. If the specified directory is not found in the file system, you should output the message The directory not found. 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:

```
linux.csc.uvic.ca:/home/user$ ./diskput disk.IMA
[destination path] <filename>
```

where optional `[destination path]` specifies the destination path within the file system starting from the root of the file system. If no `[destination path]` provided, then the file is copied to the root directory of the file system, e.g.,

```
linux.csc.uvic.ca:/home/user$ ./diskput disk.IMA <filename>
```

will copy *filename* to the root directory of the file system.

Note:

1) Since most linux file systems do not record the file creation date & time (it is called birth time, and it is mostly empty), let's set the creation time and the last write time the same in the disk image, which is the last write time in the original file in linux.
2) 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 a file from the file system.

File System Specification

Please refer to FAT12Description.pdf and FATsearchsteps.pdf attached to this assignment.

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 into disk.

Submission Requirements

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

Instructions for Uploading your assignment:

- Remove all generated files such as: *.o *.bak *~ and executable files. Usually, your Makefile should have clean target (make clean) option.
- Rename the folder containing your solutions as V<number>_SFS where V<number> is your student V number without <>. For example: V00000_SFS.
- From inside linux.csc.uvic.ca machine, tar and compress your solution folder by executing the following command:

  tar -zcf V<number>_SFS.tar.gz V<number>_SFS/*

  Please note: any other compressing formats will not be accepted!

- Test your generated file by executing the following:

  tar -ztf V<number>_SFS.tar.gz
  This should test and list of all files contained in the V<number>_SFS.tar.gz file.

  Or, copy the compressed tar file (V<number>_SFS.tar.gz) into a different folder and execute:

  tar -zxf V<number>_SFS.tar.gz
  This should extract your compressed file and creates a folder named V<number>_SFS and you should find your files inside the folder.
- Upload V<number>_SFS.tar.gz file into BrightSpace.

Marking Scheme

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

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.

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 system calls and function calls, particularly those related to the exceptions listed in Section 2, should be checked and all values should be dealt with appropriately.

Detailed Test Plan

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

COMPONENTS	WEIGHT
Makefile	5
diskinfo	15
disklist	20
diskget	25
diskput	30
Readme.txt	5
TOTAL WEIGHT	100

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 server linux.csc.uvic.ca to test your work.

Plagiarism

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

Your markers will submit the code to an automated plagiarism detection program. We add archived solutions from previous semesters (a few years worth) to the plagiarism detector, in order to catch "recycled" solutions.

The End
