CSc 360: Operating Systems
(Summer 2024)

Programming Assignment 3
A Simple File System (SFS)
Public: July 5, 2024
Due: August 2, 2024 (23:55 pm)

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

./diskinfo disk.IMA

The 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
(including all files in the root directory and files in all subdirectories):
=============
Number of FAT copies:
Sectors per FAT:
```

- Note 1: when you list the total number of files in the disk, a sub-directory name is not considered as a normal file name and thus should not be counted.
- Note 2: For a directory entry, if the field of "First Logical Cluster" is 0 or 1, then this directory entry should not be counted.
- Note 3: Total size of the disk = total sector count * bytes per sector
- Note 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.

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

./disklist disk.IMA

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

- Directory Name, followed by a line break, followed by "==========", followed by a line break.
- List of files or subdirectories:
  - The first column will contain:
    - F for regular files, or
    - D for directories

    then, followed by a single space
  - then 10 characters to show the file size in bytes, followed by a single space
  - then 20 characters for the file name, followed by a single space
  - then the file creation date and creation time.
  - 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.

2.3 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:

./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 specified directory (i.e., the root directory or a sub-directory) 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:

./diskput disk.IMA /subdir1/subdir2/foo.txt

where subdir1 is a sub-directory of the root directory and subdir2 is a sub-directory of subdir1, and foo.txt is the file name. If no specified directory is given, then the file is copied to the root directory of the file system, e.g.,

./diskput disk.IMA foo.txt

will copy foo.txt 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.
- Note 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 foo.txt from the file system.

3 File System Specification

The specification of FAT12 and related information could be found in Brightspace – content – Week 10.

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 submit **FOUR** source code, readme.txt, and **ONE** Makefile that produces the executables (i.e., diskinfo, disklist, diskget, and diskput).

6 Marking Scheme

Your submission will be graded based on correct functionality and code quality.

6.1 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.

6.2 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.
3. Proper variable names
4. Small number of global variables, if any. Most programs need a very small number of global variables, if any.

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.

6.3 Detailed Test Plan

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

Components	Weight
Make file	5
diskinfo	15
disklist	20
diskget	25
diskput	30
Readme	5
Total Weight	100

7 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.

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