CPSC 313: Computer Hardware and Operating Systems
Assignment #5, due Wednesday, November 21st, 2012 at 22:00

Introduction and Objectives

This assignment is all about file systems. While completing this assignment,

1. You will gain additional practical experience writing code in the C programming language.

2. You will gain a better understanding of the implementation of at least one file system (specifically, the MS-DOS file system that is used by most digital cameras and MP3 players).

The assignment is divided into three parts. The division is meant to provide guidelines to help you allocate time when you are working on the assignment, and to help you make steady progress on it instead of leaving it to the last minute. Although the assignment is broken into three parts you are not required to handin the intermediate parts and will handin only the completed assignment. For some people this is a challenging assignment, so you should get started early. If you want you can work with one other person. Groups of three or more people are not permitted.

The program you are to write, named **fat12info** is to be written from scratch using C and it must compile and run on the undergraduate Linux machines provided by the department. If you are you using your own machine make sure to test that it compiles and runs as expected before handing it in. Keep in mind that the department machines are all running a 64 bit version of Linux and as a result there could be differences in prototypes provided by different include files and of course there are always the potential issues that arise when porting from a machine with one native word size to another with a different size. Also, you are not allowed to use C++ to complete this assignment. You are only allowed to use functions from the standard C library. In particular, this means that you are *not* allowed to use a library of functions that deal with FAT file systems from somewhere on the Web.

When writing this program you are to use good software design practices for the naming of functions, variables, and the organization of functions into separate files. In particular this means that you are not to put all of your functions into a single file. Instead, put related functions into the same file. For example, putting functions dealing with block I/O together, and then putting directory functions together would make a lot of sense. To deal with compiling these separate files into a single executable you will want to use a makefile. The zip file associated with this assignment contains a suggested collection of files to place your code, function prototypes, structure definitions etc., into along with a makefile. These files are just suggestions and you are free to delete, rename, or add to the collection of files that comprise this assignment.

Part 1

You should aim at completing this part of the assignment by Wednesday November 14th, 2012. The goal of this part is to help you become comfortable using random access techniques to access the data in a file and then interpreting some of the files contents. You will find on the course web site a file `fat_volume.dat` that contains the image of a small MS-DOS file system. The task for this first part is to write code to output the following information about this file system:

- its sector size.
- its cluster size in sectors.
- the number of entries in its root directory.
- the number of sectors per file allocation table.
- the number of reserved sectors on the disk.
- the number of hidden sectors on the disk.
- the sector number of the first copy of the file allocation table.
- the sector number of the first sector of the root directory.
- the sector number of the first sector of the first usable data cluster.

All of this information can be computed from the information stored in the first sector of the file system. Full details about the MS-DOS file systems can be found here:

http://www.win.tue.nl/∼aeb/linux/fs/fat/fat-1.html

One of the sentences on this page is somewhat confusing. Under the heading "FAT12", you will find the statement:

> Since 12 bits is not an integral number of bytes, we have to specify how these are arranged. Two FAT12 entries are stored into three bytes; if these bytes are uv, wx, yz then the entries are xuv and yzw.

Here is a simpler interpretation of the second sentence: Suppose you have the bytes b_1, b_2 and b_3 in that order. You can extract the two FAT12 entries by treating these three bytes as a single little-endian 24 bit unsigned integer x. To get the FAT12 entry that occupies the low-order 12 bits *and* x with 0x0FFF. To the get the FAT12 entry that occupies bits 12 to 23 bits shift the x value to the right by 12 so that bits 12 to 23 are now bits 0 to 11. Unless you are 100% sure that the top order 8 bits were 0, *and* this value with 0x0FFF to ensure that only the 12 bits of interest are captured.

When writing your code for part 1, be sure to make it general enough to use again for parts 2 and 3. For instance it might be useful to define a `struct` like `filesystem_info` that contains most of the information about the file system that part 1 asks you to determine. It is strongly recommend that you write functions to:

• Read a 512 byte sector into a buffer in memory since the approach of reading one byte at a time from a file is extremely inefficient.

• Read a single cluster (In the DOS world the term cluster corresponds to the term block that we have used in class) into a buffer in memory.

• Extract from a memory buffer an unsigned integer value from two (or three, or four) specific bytes treated as a little-endian value.

You may find the manual pages for the C library functions `open`, `lseek`, and `read` helpful. To access one of them, simply type

man 2 open

(for instance) in a terminal window[1].

Your test program must take a single argument, the name of the file containing the file system it is supposed to interpret. That is, your `main` function should look at its parameters `argc` and `argv` to determine which file should be used.

Part 2

You should aim at completing this part of the assignment by Thursday November 15, 2012. The goal of this part is to be able to deal with directory entries, and extract information about the files a directory contains. More specifically, you should add to your code from part 1 so it can output the following information about each file in the root directory:

• Its name, including the extension if one is present.

• Whether or not this file is a directory.

• The number of the first cluster containing its data.

• Its size (note that directories have a size of 0).

This part of the assignment should be relatively easy once you have completed part 1.

Part 3

To complete this part, the program is to print out the information from part 2 about every file in the file system, **including files in subdirectories**. The name printed for each file is to include its full path name (for instance, `ASS2/MCHEME.TXT`). The program is also to print out the list of clusters used for each file, in whatever format you prefer. This part of the assignment is to be completed by adding to the solution from Part 2.

[1] the `2` is necessary because there are several references to things called `open`, and you want the manual page for the C library function, instead of the Tcl built-in command, the Perl pragma, or something else

This part of the assignment will likely be the most challenging because it deals directly with the File Allocation Table. Note that because the file system is relatively small, it is stored using the FAT12 format. It is recommended that you write functions that:

- determine if a given cluster number is the last cluster in its file.
- retrieve the next cluster number in a file, given the current cluster number.

Deliverables

You are to use the `handin` program to submit the assignment. The assignment name is `a5`, and the files to submit for this part are:

1. All of your C source and include files. They `must` be commented appropriately, and marks will be taken off it they do not contain enough comments to allow the TAs to figure out what your code is doing.

2. A makefile such that when make is typed in the directory containing your solution the program **fat12info** is built. This program takes a single argument, the name of the file containing the sample file system the program is to decode. Your code must compile by typing make and there must be no errors or warnings. You are not allowed to change the gcc options in the makefile to alter how and when any warning messages are printed.

3. A file in text format that contains the following information:

   - Your name and student number.
   - The output from your program.
   - How long it took you to complete this assignment.
   - **DO NOT** handin a partner.txt file

If you are part of a group, only **one** of you are to submit the files as listed above. The other member of the group must submit a single file called `partner.txt` that contains the undergraduate login ID of your **other** partner (for instance, `c3p0`) and nothing else. This will allow us to properly associate you with the work that was done with your partner.

Makefiles

In the batch of files for this assignment is included a Makefile to be used when building the program. If you add any new files you will need to edit the Makefile so that the line that starts **"SRC ="** contains the name of the new .c file. After modifying and saving the makefile you will need to run makedepend to update the makefile with any .h dependencies. Any time you add the include of a .h file to either a .c file or another .h file you need to rerun makedepend. A more detailed description on how to use make and how makefiles work can be found at:

http://www.ugrad.cs.ubc.ca/~cs219/CourseNotes/Make/intro.html