Assignment 7 -- File Allocation
(100 points)

This assignment involves maintaining a File Allocation Table.

We will simulate "FAT-12" on a (very small) disk. Each block will be 512 bytes in size. With FAT-12, we have 4096 blocks, so the disk is only 2 MB in size.

Starting with an empty root directory, we will carry out a series of transactions. From time to time we will print the contents of the directory and (part) of the FAT.

Write a program in C or C++ on the turing system to accomplish this. Designing your program is largely up to you.

Input File

The data in the input file is in lines. Each line begins with 1 letter indicating the type of transaction involved. After that are more items (file name, file size) as described below.

The file ends with a line starting with '?'. This is present as a delimiter.

The file can be found here:

/home/turing/t90hch1/csci480/Assign7/data7.txt

What to do

The main loop of the program does the following:

Read the letter indicating which transaction is next. After that, depending on the kind of transaction, read the rest of the data for that transaction.

Type 'C': Copy File transaction. The line also contains the file name and the name of the new file.

Search for the file name. If it does not exist, we have an error.
Search for the new file name. If it already exists, we have an error.
Create a new directory entry with the new name and the same size and then allocate space for it.

Type 'D': Delete File transaction. The line also contains the file name. Search for the file name. If it does not exist, we have an error. Remove the directory entry for this file and deallocate the space it was using.

Type 'N': New File transaction. The line also contains the file name and the file size.

Search for the file name. If it already exists, we have an error. Create a directory entry with this name and size and then allocate space for it.

Type 'M': Modify File transaction. The line also contains the file name and the new file size.

Search for the file name. If it does not exist, we have an error. After that, the easiest way to do a Modify transaction is to use the other kinds of transactions: New to make a new file with a temporary file name and the new size; Delete to get rid of the old file; and Rename to change the temporary file name to the name of the old file.

Type 'R': Rename File transaction. The line also contains the old file name and the new file name.

Search for the old file name. If it does not exist, we have an error. Search for the new file name. If it already exists, we have an error. Change the name.

Type '?": The run ends.

Items you will probably need

You will need a table of short integers to be the FAT itself, all initialized to 0.

You will need a data structure to contain the directory entries. I suggest defining a class or struct called Entry and then having a linked list or array (or whatever) of Entry instances. Maintain the order (chronological) of the entries. A directory entry contains (at least) the following items:

--- the file name
--- the file size
--- the number of the first block for that file

Define a constant to represent the value 512, the number of bytes in a block.

Define a constant called HOWOFTEN with the value 6.

Define a constant to represent 12, the maximum number of entries in a directory block.

You may want to define a constant to indicate how much of the FAT to print each time. The eventual value should be 240, but it may be convenient to use smaller values for this constant and for HOWOFTEN in developing your program.

You may want to define a function for each kind of transaction and perhaps use a switch statement. You will also need functions to print the directory and the FAT. You may also need an assortment of little utility functions for purposes such as:

-- searching for a file name in the directory
-- deciding how many blocks a file needs, based on its size
-- finding the first free block
-- finding the last block used by a file

Notes

You may assume that all of the transactions are valid.

The file names are no more than 20 characters long and might include a period somewhere. They do not include blanks or other white space or other punctuation symbols.

At the beginning, the directory is empty. At that point, it should contain only two entries named "." (the directory itself, size = 512 initially, starting block 0) and ".." (its parent directory, size 0, starting block 0). We will assume a directory block can contain up to 12 entries.

When you allocate blocks to files, start with the low-numbered blocks. We will probably never use most of the high-numbered blocks. The last block of a file is indicated by the value -1 in the File Allocation Table. A block not in use is indicated by the value 0 in the File Allocation Table.

Print the directory at the beginning, at the end, and after every HOWOFTEN transactions. This should include a count of the number of files listed and the sum of their sizes.

When you print the directory, print each file's name and size and a list of the numbers of the blocks allocated to the file.

When you print the directory, also print the first 240 entries in the File Allocation Table, 12 per line.

Notice that it is legal to have a file of length 0. For such a file, the starting block number is -1 and the directory listing should indicate "(none)" as the list of blocks.

The root directory itself should have starting block 0. Notice that this implies FAT[0] is -1, initially.

Whenever you create or remove an entry from the directory, the directory may itself need to add a block or lose one.

Notice that we are not creating subdirectories. We have just one directory. The parent directory ".." does not actually exist.

Please don't print out the entire FAT, as most of it will be empty.

Comments

Copy the input file into your own directory.

You may use other variable names if you want, but they should be reasonable names reflecting their purposes.

Your program should be appropriately indented and well documented. You can find style guidelines on the web sites of the CSCI 240 and 241 courses.

You may use the standard template libraries if you wish.

You should have a makefile. The name of the executable file should be "Assign7".

When you are done, you need to submit your work on Blackboard. As in the other assignments, you should create a tar file containing the files involved: the program file(s), header file(s) and the makefile. To do this, you need the "tar" utility.

Do the following (replacing "Znumber" with your own Z-ID):

(a) Create a subdirectory named Znumber_A7_dir.

(b) Copy the files into it (source code files, headers, input file and makefile).

(c) In the parent directory of Znumber_A7_dir, use this command:

```
tar -cvf Znumber_A7.tar Znumber_A7_dir
```

Use an FTP program to retrieve the tar file and then submit it on Blackboard. The TA will move it to turing, extract the files and run your makefile, as in:

```
tar -xvf Znumber_A7.tar
cd Znumber_A7_dir
make
```

If your makefile does not run (on the turing system) or your program does not compile and run (on the turing system), you will receive no credit.