FA25 CSC310 Final Project

Due: 5:00PM, 12/12/25

Objectives

At the end of this assignment, you should be able

- To write C programs that use command line arguments to enter data
- To read and write binary files in C
- To modify a program to format a disk image to the QFS specification provided on Blackboard
- To write a program to print information about a QFS disk image, and list the directory contents
- To write a program to read a file from a QFS disk image
- To write a program to recover files from a QFS disk image
- To (optionally) write a program to delete a file from a QFS disk image
- To (optionally) write a program to add a file to a QFS disk image

Background

You have been provided with the description of a new kind of file system called QFS (Quinnipiac File System). The QFS file system is a simple file system to store files on small disks and disk images. The QFS file system has been designed to be simple to implement and understand, while still providing basic file system functionality. It supports a single-level directory structure, and blocks of data are fixed in size, based on the size of the disk image file. Each file block that does not include the end of a file has a pointer to the next block of data in the file. This pointer is its uint16_t block number (starting with block 0).

A QFS disk is divided into several key sections:

- **Superblock:** Contains metadata about the file system, such as block size, total number of blocks, free block count, total number of directory entries, and number of free directory entries.
- **Directory Entries:** A fixed-size table that holds information about files and directories.
- **File Data Blocks:** The area where actual file data is stored.

Assignment

In this assignment you will be writing C programs to work with QFS disk images. The programs you will be writing are in two categories: ***REQUIRED*** and ***OPTIONAL***. Completing all four required programs will earn you 100% on this assignment. Completing the optional programs will earn you 10% extra credit per program. The programs you will be writing come from the following list:

mkfs_qfs.c [***REQUIRED***] You will be provided with a version of this program that will format a disk with 512 byte blocks. The QFS standard states that the size of data block is determined by the overall size of the disk or disk image. Your first task will be to modify this program so that it correctly formats all disks or images up to 120MB. Note that the process of formatting a disk involves setting the superblock values, emptying the directory entries, and marking all data blocks as free (byte 1 of each block). No data blocks should be cleared during the formatting process beyond marking them free. The disk label is optional and should be set to an empty string if not provided. It should be no longer than 14 characters and have no spaces.

list_information.c [***REQUIRED***] Your next task will be to write a program that will read in a QFS disk image and list information about the disk image from the superblock, and its directory contents. It should print out the following information:

- Block size
- Total number of blocks
- Number of free blocks
- Total number of directory entries
- Number of free directory entries
- For each file in the directory, print the file name, file size, file type, and starting block number.

read_file.c [***REQUIRED***] Your third task will be to read in a QFS disk image and extract a specified file from the disk image, writing it to the current working directory. Your program should take three command line argument: The first is the name of the disk image, the second is the name of the file to be extracted, and the third is the name to call it in the current working directory. It should search the directory entries for the specified file, and if found, read the file data blocks and write the contents to a new file in the current working directory. If the file does not exist on the disk, an appropriate message should be printed.

recover_files.c [***REQUIRED***] Your fourth task will be to read in a QFS disk image and recover all files stored in the disk image, writing them to the current working directory. For this program you will be provided with a disk image that had jpg files on it, and has been subsequently formatted with QFS. Your program should search the data blocks for any jpg file data, and reconstruct the files based on the data found. You should write each recovered file to the current working directory. Name the recovered files as `recovered_file_X.jpg`, where X is a unique number for each file. Here is a link to the jpg file signature information you will need to identify jpg files (Note that jpg files start with the byte sequence `0xFF 0xD8` and end with the byte sequence `0xFF 0xD9`):

`https://github.com/corkami/pics/blob/master/binary/JPG.png`

delete_file.c [***OPTIONAL***] Your fifth program will read in a QFS disk image and delete a specified file from the disk image. Your program should take two command line arguments, one for the name of the disk image and one for the file to be deleted. It should search the directory entries for the specified file, and if found, mark the file's directory entry as empty. It should then mark all blocks used by the file as free by writing a 0 in the first byte of each data block storing the file. It should also change the superblock as needed.

write_file.c [***OPTIONAL***] Your final program, should you choose to implement it will read in a QFS disk image and add a specified file from the current working directory to the disk image. Your program should take one command line argument for the image, and one for the name of the file to be added. It should find a free directory entry and enough free data blocks to store the file's contents.

All of the programs should read input from the user as command line arguments (no file redirection). For example, to make a QFS file system on a disk image, your program would be run as follows:

```
./mkfs.qfs <image> [<label>]
```

To list the contents of a QFS disk image, your program would be run as follows:

```
./list_information <image>
```

To copy a file from a QFS disk image to the local filesystem, your program would be run as follows:

```
./read_file <image> <stored_file_name> <local_file_name>
```

To recover files from a QFS disk image, your program would be run as follows:

```
./recover_files <image>
```

To delete a file from a QFS disk image, your program would be run as follows:

```
./delete_file <image> <stored_file_name>
```

To write a local file to a QFS disk image, your program would be run as follows:

```
./write_file <image> <local_file_name>
```

You can find example programs and data structures for both the Minix and FAT16 file systems in the sample code (files.zip) in the Chapter 8 folder on Blackboard. You may find these examples helpful in writing your own programs for this assignment. Also the chapter 8 folder has links to descriptions of both the Minix and FAT16 file systems. I cannot stress enough the importance of understanding the file system structure you are working with before attempting to write code to manipulate it.

Starter code for each part of this assignment, including a header file with the QFS data structures will be provided on Blackboard. You should use this starter code as a starting point for your programs. You should modify the provided `mkfs_qfs.c` program to support variable block sizes as needed. You should not need to modify the other provided files beyond adding your own code to implement the required functionality.

Submission

Please pass in a zipped folder named with the QU login shortname of one of your team members containing at a minimum the following five files for this assignment:

Please pass in a zipped folder named with the QU login shortname of one of your team members containing the following files for this assignment:

- Makefile [Provided in the starter code]
- mkfs_qfs.c
- list_information.c
- read_file.c
- recover_files.c
- delete_file.c [OPTIONAL]
- write_file.c [OPTIONAL]

Testing

Test files will be provided separately on Blackboard. You should test your programs with these files to ensure that they work correctly.

Grading

- **mkfs_qfs.c** - 25 points
- **list_information.c** - 25 points
- **read_file.c** - 25 points
- **delete_file.c** - 25 points
- **recover_files.c** [OPTIONAL] - 10 points extra credit
- **write_file.c** [OPTIONAL] - 10 points extra credit

For each program, points will be awarded equally based on correctness and code quality.

Notes

- As mentioned in class, this is a group project. You may work in groups of up to four students.
- Each group should submit only one copy of the assignment, with all group members' names included in a comment at the top of each source file.
- All input to your programs will be well-formed. You do not need to handle invalid input.
- You should use a consistent coding style, and include comments in your code to explain your logic.
- You should comment your code appropriately to explain your logic. Include a header comment at the top of each file with your name, the date, and a brief description of the file's purpose.
- If you are not attempting the optional programs, please do not include them in your submission.
- The following hints may be useful as you work on this assignment:
  - Use `fopen()` with the "rb+" mode to open files for both reading and writing in binary mode.
  - Use `fseek()` to navigate to specific locations in the disk image file.
  - Use `fread()` and `fwrite()` to read from and write to the disk image file.
  - Remember to update the superblock and directory entries as needed when modifying the file system.
  - Modifying `mkfs_qfs.c` to support variable block sizes does not need to happen first. You can implement and test the other programs using the original 512-byte block size version of `mkfs_qfs.c` and the sample files provided, then modify it later to support variable block sizes.
  - `read_file.c` and `recover_files.c` are extremely similar. Both will require reading multiple data blocks to reconstruct a file. You may find it helpful to implement one of these programs first, then use similar logic for the other.
- Due to the complexity and timing of this assignment, I will not be checking for memory leaks.