Project 7: FAT12 File System

CS 311

In this project, you'll implement several file system operations in user space for the FAT12 file system. You're given a floppy diskette image file, driver-level source code, file system-level constant and type definitions, and a test suite application program. You'll add six file system-level functions.

References

The following are all available on, or linked to from, the course web site, or in the textbook:

1. *An Overview of FAT12*. This is your FAT12 file system specification document. Read it. Read it again. Read it a third time. Put it under your pillow at night when you're sleeping.
2. `floppyData.img`. This is your FAT12 floppy diskette image.
3. `driver.h` and `driver.c`. This is your driver-level code. Use it as-is, making no modifications.
4. `fstypes.h`. File system-level constant and type definitions. Use of this file is optional — you may, or may not, find some of the definitions useful. `direntry_t` should be extremely useful. `fat_t` and `root_t`, well, YMMV.
5. `fsops.h` and `fsops.c`. The code you write goes into these two files. Your system may be compiled:

   ```
   % gcc -o exercise driver.c fsops.c exercise.c
   ```

   Private global variables and private helper functions should be defined in `fsops.c` and declared `static`.
6. `exercise.c`. The test suite application.

Background

After reading *An Overview of FAT12* note the following subtleties:

1. FAT table entries refer to logical blocks. Add 31 to the logical block number to get the physical block number.
2. FAT table entries are 12 bits, hence the name FAT12. Two FAT table entries are packed into three consecutive bytes. One implication of this is that FAT entries can straddle disk blocks.

   Consider FAT entry `i`, where `i` is even. Its least significant eight bits are the byte at byte offset `(3 * i) / 2` into the FAT table. Its most significant four bits are the least significant four bits of the byte at byte offset `(3 * i) / 2 + 1` into the FAT table.

Now, consider FAT entry `i`, where `i` is odd. Its least significant four bits are the most significant four bits of the byte at byte offset `(3 * i) / 2` into the FAT table. Its most significant eight bits are the byte at byte offset `(3 * i) / 2 + 1` into the FAT table.

Bit-wise shift and mask operations — aren't you loving it?!?

3. The first entry in a non-root directory is '`.`', describing the current directory. The second entry in a non-root directory is '`..`', describing the parent directory. The root directory does not contain these entries. If the first logical block member in a '`..`' directory entry is 0, the parent directory is the root directory.

4. The date format in a directory entry is:

   (a) Bits 0–4: Day of month, valid value range 1–31 inclusive.

   (b) Bits 5–8: Month of year, 1 = January, valid value range 1–12 inclusive.

   (c) Bits 9–15: Count of years from 1980, valid value range 0–127 inclusive (1980–2107).

5. The time format in a directory entry is:

   (a) Bits 0–4: 2-second count, valid value range 0–29 inclusive (0–58 seconds).

   (b) Bits 5–10: Minutes, valid value range 0–59 inclusive.

   (c) Bits 11–15: Hours, valid value range 0–23 inclusive.

Tools

Read The Fine Manual for details about the following tools:

1. On phoenix, the Mtools suite may be used to examine a FAT12 image file:

```
% mdir -/ -i floppyData.img

% mdir -i floppyData.img ::/NEW/SUB

% mtype -i floppyData.img ::/NEW/SUB/FILE1.TXT
```

2. As root on your virtual machine, you can use the loop device to mount a FAT12 image file as part of the Linux file system tree:

```
% mkdir MountPoint

% sudo mount -r -t msdos -o loop floppyData.img MountPoint

# Use the mounted FAT12 file system.  The '-r' switch mounts the file
# system read-only.

% umount MountPoint
```

3. Again, as root on your virtual machine, if you have the image file mounted as above, you can run the following command to check the integrity of your file system following the use of the `fd_del()` function:

```
% sudo /sbin/dosfsck -v -V /dev/loop0
```

If, for some reason, the file system is on some other loop device run `mount` with no arguments to determine which loop device to use with `dosfsck`.

The three things you're looking for after deleting a file are that the file no longer appears in a directory listing, the free space on the file system has increased by the size of the blocks previously occupied by the file, and that `dosfsck` doesn't detect any integrity issues. The first two of these can be checked with Mtools' `mdir` command.

Description

Implement the following file system operations:

1. `int fd_mount(char *img)` — `img` should be the name of a file containing a valid FAT12 image. The FAT12 file system should be mounted read-write. If caching the FAT table and the root directory, this function should read these data structures into memory and make them available. Upon mounting, the root directory should be made the current working directory. This function will make use of `fdimgopen()` in `driver.c`

   On success, this function returns a device descriptor. On failure, it returns -1.

2. `int fd_unmount(int dev)` — Unmount the device `dev`. If the FAT table and root directory have been cached in memory, they should be flushed to disk before unmounting. Following the unmount operations, both on-disk copies of the FAT should match. This function will make use of `fdimgclose()` in `driver.c`.

   On success, this function returns 0. Otherwise, it returns -1.

3. `int fd_dir(int showAll)` — list the files in the current working directory. Use this format for producing the listing:

   ```
   .            <DIR>     2014-04-24  13:00
   ..           <DIR>     2014-04-24  13:00
   SUBSUB       <DIR>     2014-04-24  13:00
   FILE1    TXT      1538 2014-04-24  13:01
           4 files               1 538 bytes
   ```

   If `showAll` is 0, hidden files should not be listed. Otherwise, hidden files should be listed.

   This function return the number of files (and directories) listed.

4. `int fd_cd(char *dir)` — Change the current working directory. `dir` should name a sub-directory of the current working directory or be the string `".."`, indicating the parent directory. Assume that the parent of the root directory is the root directory.

   Return 0 on success. Return -1 if `dir` is not a sub-directory of the current working directory or the string `".."`.

5. `int fd_type(char *file)` — Display the contents of the file `file` in the current working directory. For the purposes of this function, a directory is not a file.

   Returns the number of characters typed on success. Returns -1 if `file` is not a file in the current working directory.

6. `int fd_del(char *file)` — Deletes the file `file` in the current working directory. For the purposes of this function, a directory is not a file. The blocks that had been in use by this file are marked free, as is the directory entry that had been in use by this file.

   Returns the number of blocks freed on success. Returns -1 if `file` is not a file in the current working directory.

Testing

As described earlier, the code you write should be put into `fsops.h` and `fsops.c` and can be compiled:

```
gcc -o exercise driver.c fsops.c exercise.c
```

When I test your code, I will be using the original versions of `driver.h`, `driver.c`, `fsops.h`, and `exercise.c`. In addition, I have my own, proprietary, test application and image file that I will use for testing.

Deliverables

The following files should be uploaded to the course GoucherLearn site by 5:00 pm on the due date:

1. Your source code files, `fstypes.h` (if you have made changes to this file) and `fsops.c`. Your source code should work with the original versions of the other files, as described in the previous section. **Do not include any binary files.**

2. A README file. This file should list any functionality missing from your project, as well as anything else you think I should know.