Project 2: File System

Due: Friday, April 18, 10pm. Pre-submission due on **Tuesday, April 8, 10pm**. See Deliverables.

Starter code: See Project 2 on Canvas for the Github link.

Submission: You can work on this assignment in a **pair**, or alone.

Submit the contents of your repository via Gradescope. See Deliverables below for what to submit. If you are working with a partner, do not forget to include their name with the submission.

Note: There will be no autograder for this assignment as we have no way of running a custom filesystem on Gradescope. We have provided tests.

Note: Clone and start studying the starter code as soon as possible. This assignment will likely require more programming effort than previous assignments.

A File System

In this assignment you will build a FUSE filesystem driver that will let you mount a 1MB disk image (data file) as a filesystem.

We also provide an ungraded lab that will lead you through installing FUSE and familiarizing yourself with parts of the starter code.

Step 1: Install FUSE

For this assignment you will need to have the FUSE framework installed. The Docker container we provided to you at the beginning of the semester already comes with FUSE installed, so you shouldn't have to do anything else. If you want to use WSL2 or Codespaces, you'll need to install the following packages:

- `libfuse-dev`
- `pkg-config`

Running

```
$ sudo apt-get update
```

followed by

```
$ sudo apt-get install libfuse-dev pkg-config
```

should do the trick.

After completing this step, go to Pre-submission and submit the repository as requested by Tuesday, April 8, 10pm.

Step 2: Implement a basic filesystem

You should extend the provided starter code so that it lets you do the following:

- Create files, supporting files with names at least 10 characters long
- List the files in the filesystem root directory (where you mounted it)
- Write to small files (under 4k)
- Read from small files (under 4k)
- Rename files
- Delete files

You will need to extend the functionality in `nufs.c`, which only provides a simulated filesystem to begin with. This will require that you come up with a structure for how the file system will store data in it's 1MB "disk". See these File system slides and OSTEP, Chapter 40 for inspiration.

We have provided some helper code in the `helpers/` directory. You can use it if you want, but you don't have to. However, `blocks.{c,h}` and `bitmap.{c,h}` might save you some time as these implement block manipulation over a binary disk image. Feel free to extend the functionality if needed.

Some additional header files that might be useful are provided in the `hints` directory. These are just some data definitions and function prototypes to serve as an inspiration for abstraction layers. They are provided "as-is", meaning you'll need to make sense of them. Reading up on file system implementation in the OSTEP book might help. If they don't seem helpful, you are free to implement your own abstractions.

Step 3: Directories

In this step, implement support for arbitrarily nested directories. The filesystem should support the following operations on directories:

- Creation (`mkdir`)
- Renaming (`rename`)
- Listing the contents of directories (`readdir`)
- Deleting (`rmdir`)
- Creating files contained in directories, moving files between directories

Step 4: Big files

Extend the filesystem to support files larger than 4K. The files must fit into the free blocks on disk. This must include proper allocation and deallocation as the file grows or shrinks. The file system needs to be able to handle the following example situations:

- at least 100 files with size 4K (part of the base assignment)
- at least 5 files with size 100K
- at least 1 file with size 500K

Deliverables

Please read the instructions carefully and ask questions. If you do not submit to the correct assignment on Gradescope, we cannot guarantee that we will grade your assignment before the end of semester.

Pre-submission

After doing Step 1, that is, after cloning the repository and installing FUSE,

1. Execute the following commands:

```
$ make nufs
$ ./nufs --version data.nufs > fuse_version
```

2. Commit the file `fuse_version` to your repo

3. Submit your repository to Gradescope under **Project 2: Pre-submission** by **Tuesday, April 8, 10pm**. No late submissions are allowed for this step.

Main submission

Modify the starter code to implement the requested functionality (steps 2, 3 and 4).

Commit the code to your repository. Do not include any executables, `.o` files, or other binary, temporary, or hidden files (unless they were part of the starter code). Do not include any disk images.

Once you are done, remember to submit your solution to Gradescope and do not forget to include your partner. Submit under **Project 2: Main Submission**

Provided Makefile and Tests

The provided Makefile should simplify your development cycle. It provides the following targets:

- `make nufs` - compile the `nufs` binary. This binary can be run manually as follows:

  ```
  $ ./nufs [FUSE_OPTIONS] mount_point disk_image
  ```

- `make mount` - mount a filesystem (using `data.nufs` as the image) under `mnt/` in the current directory
- `make unmount` - unmount the filesystem
- `make test` - run some tests on your implementation. This is a subset of tests we will run on your submission. It should give you an idea whether you are on the right path.
- `make gdb` - same as `make mount`, but run the filesystem in GDB for debugging
- `make clean` - remove executables and object files, as well as test logs and the `data.nufs`.

Rubric

The grade is broken down into three categories:

- 5% Completing the pre-submission by Tuesday, April 8, 10pm
- 45% Basic functionality and FS design
  - Based on manual testing and using a grading script
  - Does the filesystem correctly and efficiently implement the requested functionality?
  - Do operations complete in a reasonable time? We put a 30s timeout on most test cases.
  - Is the file system able to store at least 100 small ($\leq$ 4K) files?
- 15% Directory functionality
- 10% Big file support
- 25% Style
  - Via manual code review
  - Basics: meaningful purpose statements; explanation of arguments and return values
  - Explicitly stated assumptions
  - Short, understandable functions (generally, < 50 lines)
  - Consistent indentation and use of whitespace
  - Explanatory comments for complex blocks of code
  - No extra binaries (.o, executable files, etc.) or superfluous files committed to your repo

Hints & Tips

- There are no man pages for FUSE. Instead, the documentation is in the header file: `/usr/include/fuse/fuse.h` (available online at
- Also: https://www.cs.hmc.edu/~geoff/classes/hmc.cs135.201109/homework/fuse/fuse_doc.html
- The sources for libfuse contain a few further examples. Start with `hello.c`.
- The basic development / testing strategy for this assignment is to run your program (e.g., using `make mount`) in one terminal window and try file system operations on the mounted filesystem in another separate terminal window.
- Read the manual pages for the system calls you're implementing.
- To return an error from a FUSE callback, you return it as a negative number (e.g. return `-ENOENT`). **Some things don't work if you don't return the right error codes.**
- Read and write, on success, return the number of bytes they actually read or wrote.
- You need to implement `getattr` early and make sure it's correct. Nothing works without `getattr`. The modes for the root directory and `hello.txt` in the starter code are good default values for directories and files respectively.
- The functions `dirname` and `basename` exist, but may mutate their argument.