Exercise Sheet 4 for Lecture Parallel Storage Systems

Deadline: 2026-05-24, 23:59

Prof. Dr. Michael Kuhn (michael.kuhn@ovgu.de)
Michael Blesel (michael.blesel@ovgu.de)
Parallel Computing and I/O • Institute for Intelligent Cooperating Systems
Faculty of Computer Science • Otto von Guericke University Magdeburg
https://parcio.ovgu.de

---

1. FUSE File System (240 Points)

In the materials, you will find template code for a FUSE file system. It contains some important functions that you will have to implement. The file system can be compiled with `make` and run with `./dummyfs $mnt`, where `$mnt` stands for any existing directory. You will need to load the `libfuse` module using `module load libfuse` to be able to use FUSE.
Hint: We recommend to mount your file system in a directory in `/tmp` on one of the computing nodes for a better debugging experience. To get more than one active shell on an allocated node we recommend using `tmux`.

FUSE is written in C but bindings for many other languages exist. If your group has a strong preference to use a different language, you are allowed to do so. If you choose this option, we require you to submit clear instructions on how to build and run your file system and what dependencies need to be installed.

Inform yourself about the important functions and data structures by using the corresponding man pages and the FUSE documentation. The FUSE API documentation and other useful information can be found below. Implement the functions that are necessary to run the `checkpoint` application without any errors. The complete contents of the file system shall be stored in memory and be lost after unmounting the file system.
Hint: You can use the `fusermount` command to unmount your mounted file system.

Limit the file system to support one single file (`matrix.out`) with a maximum size of 5 MiB. Creating, writing to or reading from other files or directories shall return the correct error code (see man pages for the operations and FUSE documentation to find the correct error codes). For simplicity's sake, you can use a static data structure.
Hint: You can just allocate a buffer with size 5 MiB whose size does not change for the whole runtime of the file system.

While implementing, think about the extensibility of your file system because you will build on it further in the coming exercises. In the next sheet, you will extend the functionality of your file system to allow more flexibility (more and larger files etc.). Following that, you will design and then implement a persistent file system.

Compare the values for throughput and IOPS with the `checkpoint` application. You can modify its syncing behavior using the third argument. Take a look at the values for the version without sync (`sync = 0`), the version with `fsync` after every write operation (`sync = 1`) and the version with one `fsync` per iteration (`sync = 2`). What catches your attention? What influence does the usage of many small operations and of `fsync` have on the performance of your file system? Write down your observations in a file called `performance.md`.

Make sure that you can enter the mount point of your file system (`cd $mnt`) and are able to execute `ls -l` correctly there. Furthermore, it should be possible to delete the created checkpoint (`rm`) and to change its size (`truncate`).

2. Stucture of File System Operations (60 Points)

Describe the internal workflow of your file system for creating, deleting and listing a file. Enumerate in chronological order all called file system operations and explain why each of them is required.

For this, activate the debug mode for your FUSE file system with `-d`. It can also useful to disable multi-threading with `-s`.

Additional Resources

The FUSE GitHub repository contains an example directory that can be helpful for understanding how the FUSE API can be used. A simple "hello world" example can be found at `https://github.com/libfuse/libfuse/blob/master/example/hello.c`. A more elaborate tutorial can be found at `https://www.cs.nmsu.edu/~pfeiffer/fuse-tutorial/`.

The FUSE documentation should be your first place to go to look up information about FUSE functions or data structures. It can be found at `http://libfuse.github.io/doxygen/`.

The `stat` struct contains important information about a file or directory and will be necessary to implement the functions of your file system. You can familiarize yourself with with `stat` at `https://linux.die.net/man/2/stat`.

Submission

We will count your last commit on the `main` branch of your repository before the exercise deadline as your submission. In the root directory of the repository, we expect a `PSS-2026-Exercise-04-Materials` directory with the following contents:

- A file `group.md` with your group members (one per line) in the following format:

  ```
  Erika Musterfrau <erika.musterfrau@example.com>

  Max Mustermann <max.mustermann@example.com>
  ```

- Task 1: The modified source code for your file system `dummyfs.c` and the corresponding Makefile. If you chose a different language than C, please include all source code and a way to build the application easily.
- Task 1: A text file `performance.md` with your observations.
- Task 2: A text file with the operations performed by your file system as well as your explanations called `structure.md`.