Exercise Sheet 6 for Lecture Parallel Storage Systems

Deadline: 2024-07-02 and 2024-07-09, 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. File System Design (180 Points)

In the second part of this exercise, you will extend your implemented FUSE file system to work with a storage device. In the first part of the exercise, you will therefore modify the design of your current file system accordingly.

Your file system is no longer supposed to only keep data temporarily in memory but to store it persistently on a storage device. This storage device can be a real block device as well as a file in a different file system.

Furthermore, the already implemented file system shall be extended in a way that it is no longer dependent on external memory management (as provided by `malloc` up to this point). Since this means that it will no longer be possible to easily reserve continuous memory regions with the help of `malloc`, you will have to handle the management and allocation of free and used memory regions by yourself.

Additionally, the limitations for the file system from the last exercise sheet are relaxed:

1. The file system shall support a maximum size of 16 GiB.
2. A single file shall have a maximum size of 256 MiB.
3. The storage space for a file must not be allocated from the beginning.

Think about what changes are necessary to your file system design to achieve this. For this, describe at least the following aspects:

- Segmentation of the storage device (visual representation and textual description)
  - Where on the storage device are the metadata and data located?
  - What additional information is needed?
  - In what form is the storage device segmented?
  - How do you deal with the fragmentation of files?
- Management of free and used storage regions
  - What are the differences between data and metadata?
  - What data structures would you use for efficient accesses?
  - How are concurrent accesses handled?

- Structure and storage format of metadata (for example, in the form of inodes)
  - How are the data regions of a file referenced?
- Data integrity in case of crashes
  - Is a file system check needed after a crash? If yes, how would this work?

2. File System Implementation (720 Points)

In this task, you shall implement your file system design from the last task. Note that the reason for the design document from the last task is to give you early feedback for your ideas and that you might have to improve your design after it was discussed in the exercises.

For the underlying storage device, a real block device as well as a file in a different file system shall be supported. You can create such a file with for example `fallocate` or `truncate`. The storage device shall be given as an argument to your FUSE file system. The command shall look like the following:

```
$ ./pssfs device mountpoint
```

Make sure that changes are actually written to the storage device. This shall happen no later than at a call of `fsync` by an application. The file system shall additionally be as resistant to crashes as possible.

Hint: To make the implementation easier, it makes sense to use `mmap`. With this, the whole block device or the whole file can be mapped as one continuous memory region into the address space of your file system. This is possible even if the block device or file is larger than the available memory of the system.

3. Performance Evaluation (180 Points)

Analyze your file system with the benchmarks in the materials. To compile it, you will have to load the `glib` module:

```
$ . /opt/spack/pss-2024/env.sh
$ module load glib
```

Create suitable plots and write down your conclusions. The benchmark can be called as follows, where `$mnt` stands for the mount point of your file system:

```
$ ./data --path=$mnt --sync-on-close --iterations=3
```

Further options can be displayed with the `--help` argument. Measure the performance of your implementation with the option `--sync-on-close`, 1–12 threads (`--threads`) and suitable block sizes (`--block-size`) of 4 KiB and 4 MiB. Vary the block count (`--block-count`) so that the benchmark runs for several minutes and that the maximum file size of 256 MiB is not exceeded. Conduct the measurements for thread-local files as well as for a shared file (`--shared`).

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-2024-Exercise-06-Materials` directory with the following contents:

- A file `group.txt` 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: An extensive design document detailing the inner workings of your file system called `design.pdf`. The document should discuss all aspects mentioned in the task. The design document has to be submitted once until the first deadline and then again for the second deadline (with possible improvements/changes).
- Task 2: The source code for your file system `pssfs.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. The implementation has to be submitted until the second deadline.
- Task 3: A document containing your visualized performance measurements and explanations called `benchmark.pdf`. The evaluation has to be submitted until the second deadline.