ITP 30002 Operating System

Homework 5. JsonFS using FUSE

last update: 2 PM, 5 June 2023

Shin Hong

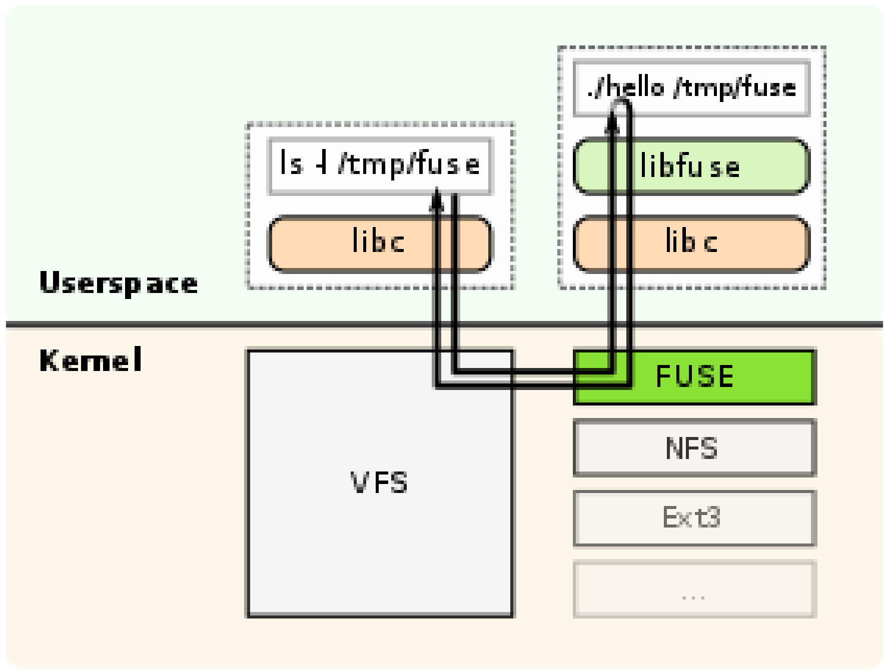
Overview

- This homework asks you to write a FUSE program that constructs a user-level file system based on the structure and data defined in a JSON file
  - FUSE is a framework to handle file-related system calls by calling corresponding handlers defined in a user-level program
  - The information about directories and regular files of a target file system is given as a JSON file

- Attributes
  - individual work: you are allowed to study the sample program with reading group members, but not allowed to collaborate on the homework tasks
  - video demo: submit the video for reporting your results
  - submission deadline: 9 PM, Tue 20 June (this is strict)

Background - FUSE

- Filesystem in UserSpacE (FUSE) allows non-privileged users to create a special-purpose file system as an application program (not kernel module)
  - FUSE was merged into the Linux kernel mainstream since 2016
  - FUSE is widely used for implementing file-system-like interfaces for various system programs
- Using libfuse, a FUSE program must provide a set of callback functions for file operation handlers



Example

- Hello world example in FUSE
  - https://github.com/fntlnz/fuse-example/tree/master
  - https://engineering.facile.it/blog/eng/write-filesystem-fuse/

FUSE File Operations

- http://libfuse.github.io/doxygen/structfuse__operations.html

- `void *(* `**`init`**`)(struct fuse_conn_info *conn, struct fuse_config *cfg)`
- `void(* `**`destroy`**`)(void *private_data)`

- `int(*  `**`getattr`**`)(const char *, struct stat *, struct fuse_file_info *fi)`

- `int(* `**`mkdir`**`)(const char *, mode_t)`
- `int(* `**`rmdir`**`)(const char *)`

- `int(* `**`rename`**`)(const char *, const char *, unsigned int flags)`
- `int(* `**`open`**`)(const char *, struct fuse_file_info *)`
- `int (* `**`create`**`) (const char *, mode_t, struct fuse_file_info *);`
- `int (* `**`unlink`**`) (const char *) ;`
- `int(* `**`read`**`)(const char *, char *, size_t, off_t, struct fuse_file_info *)`
- `int(* `**`write`**`)(const char *, const char *, size_t, off_t, struct fuse_file_info *)`

JsonFS

- Initialize a file system according to a given JSON file, and store the updated files back to the JSON file at unmount

- Users can read and write text file, create a new file and a new directory, and remove an existing file and an existing directory

- Data format
  - a JSON file is a list of files each of which has a unique `inode` number
  - a file has a type: directory (“`dir`”) or regular text file (“`reg`”)
  - a directory has `entries` that enumerates pairs of filename and inode numbers
  - the file with inode 0 is representing the root directory of the file system
  - a regular file has `data` that stores the text data of the file

Example

```
[
 {
  “inode”: 0,
  “type”:”dir”,
  “entries”:
   [
    {“name”: “hello”, “inode”: 1},
    {“name”: “d1”, “inode”: 2}
   ]
 },
 {
  “inode”: 1,
  “type”:”reg”,
  “data”: “Hello world!”
 },
 {
  “inode”: 2,
  “type”:”dir”,
  “entries”:
  [
   {“name”: “d2”, “inode”: 3}
  ]
 },
```

```
 {
  “inode”: 3,
  “type”:”dir”,
  “entries”:
   [
    {“name”: “bye”, “inode”: 4},
    {“name”:”hello”, “inode”:1}
   ]
 },
 {
  “inode”: 4,
  “type”:”reg”,
  “data”: “Goodbye!”
 }
]
```

Program Requirements

- Command-line interface
  - `$./jsonfs [input JSON file]`

- Assumptions
  - every regular file is a text file (containing only ASCII characters), and the size of the content does not exceed 4098 bytes.
  - each directory has no more than 16 files
  - a file may have multiple paths (i.e., links)
  - a given JSON file is always valid
  - total number of files in the system does not exceeds 128

- Multiple callback functions may be invoked concurrently, thus they must be properly synchronized

Submission Instruction

- Video demo
  - Explain your program design and implementation, and demonstrate that it works correctly
    - especially, describe how callback functions are synchronized
  - The video must be no more than 8 minutes
  - Upload your video to YouTube or other streaming services, and write down the URL at submission

- Submit all results to HDLMS by 9 PM, 20 June (Tue)
  - write the URL of your demo video to the submission note
  - create a zip file that archives all files of the resulting artifact including source code files, a build script ,README.md and libraries

Notes

- You can use a JSON library for reading and updating a JSON file
- Make sure that your artifact works correctly on the peace server
- Ask questions at the `#hw5-jsonfs` channel