CSC 501 (001) Fall 2018 Operating Systems Principles

Resource containers -- file

Overview

In the previous project, we have applied the concept of containers on processor and memory. In this project, you will be building a file system that isolate files within a file system using containers. The file system must be compatible with conventional file system operations that existing applications can easily access it without modifying any line of code.

This project relies on "file_container" pseudo device that WE implemented. Instead of implementing a file system from scratch, you will be using libfuse 2.9. Fuse is a library that hides most low-level details of file systems and allows programmers to design a file system running in the user-space. With the kernel module and the library of fuse take care of all low-level system interactions, you simply need to focus on those data structures and operations that maintain the abstraction of a file system using containers.

You are strongly encouraged to work in a group of 2. Groups do the same project as individuals. All members will receive the same grade. Note that working in groups may or may not make the project easier, depending on how the group interactions work out. If collaboration issues arise, contact your instructor as soon as possible: flexibility in dealing with such issues decreases as the deadline approaches.

In this project, you will be given the prototype of the file system. the src/fcfuse.c file contains the some basic functions that are mainly for initialization as well as the declaration of supported operations. The src/fcfuse_function.c that only contains empty functions -- you need to implement those to make the file system work. You may also define your own file system status structures and other important file system structures in the src/fcfuse_extra.h header file.

Objective

* Learning essential operations of a file system
* Learning how system library interact with file systems
* Designing a file system for innovative storage devices

How to start

To begin, you need to first form a group and setup the environment for developing your project. You should set up a machine or a VMWare virtual machine (CS students should have free license for that https://www.csc.ncsu.edu/vmap/) with clean Ubuntu 16.04 installation. You also may use the VCL service maintained by NCSU through https://vcl.ncsu.edu/ . You may reserve one virtual machine and connect to this machine remotely by selecting reservations. We will use the "Ubuntu 16.04 Base" to test your kernel module. However, the VCL service will reset once your reservation timeouts. The default image in VCL service already has libfuse-2.9 installed. In you don't have one in your environment, please make sure that you get one through "sudo apt-get install libfuse-dev" command.

You also need to fetch the code from https://github.ncsu.edu/htseng3/CSC501_Container_File.git. After fetching the code, there are two things you have to do first. One is compile the kernel_module in the fetched repo and install, insmod as you experienced in project #1 and project #2. Then, navigate to the library directory, you will also need to compile and install the library in your system. Finally, you can type

./configure

In the fetched repo.

If the configure works successfully, you may type

make

to compile the code. Upon success, you should find an "fcfuse" executable in the src directory. To test the file system, you will have to first insert the file_container kernel module by performing "sudo insmod file_container.ko" in the kernel_module directory and "sudo chmod 777 /dev/fcontainer" as in previous projects. After you have done these, you can launch the file system by using

./src/fcfuse /dev/fcontainer {data_location} {mount_point}

where {data_location} is a directory stores your data for each container and {mount_point} is an empty directory serve as the mount point. We strongly recommend you to add "-s -d" arguments when you launch the fcfuse program since this will show you more debugging information.

At this point, your file system won't work at all. You may not see anything when you do "ls". You will see errors when you create files. You will see error messages when you try to do "df" that shows the free space on each partition. -- Yes, it's your job to make all of these plus some other features that make this file system working.

After you have some initial understanding about this project, we strongly encourage you to review the overview, those references, and then discuss with your teammate(s). You and your team should obtain a high-level system architecture before you start writing any idea. You should discuss with TAs and the instructor during their office hours **as early as possible**.

No matter you're using VMWare, real machine, or VCL, you should always use https://github.ncsu.edu to control/maintain/backup your work.

Your tasks

1. You are responsible for completing at least the following functions in the given fcfuse_functions.c to support the required features that make the file system work with the concept of container. You may use existing system calls and fuse library functions to accomplish your task if appropriate. You may design any data structure that supports the required features. Here is the list of functions you need to accomplish at least:

1. int fcfuse_getattr(const char *path, struct stat *statbuf);
2. int fcfuse_readlink(const char *path, char *link, size_t size);
3. int fcfuse_mknod(const char *path, mode_t mode, dev_t dev);
4. int fcfuse_mkdir(const char *path, mode_t mode);
5. int fcfuse_unlink(const char *path);
6. int fcfuse_rmdir(const char *path);
7. int fcfuse_symlink(const char *path, const char *link);
8. int fcfuse_rename(const char *path, const char *newpath);
9. int fcfuse_link(const char *path, const char *newpath);
10. int fcfuse_chmod(const char *path, mode_t mode);
11. int fcfuse_chown(const char *path, uid_t uid, gid_t gid);
12. int fcfuse_truncate(const char *path, off_t newsize);
13. int fcfuse_utime(const char *path, struct utimbuf *ubuf);
14. int fcfuse_open(const char *path, struct fuse_file_info *fi);
15. int fcfuse_read(const char *path, char *buf, size_t size, off_t offset, struct fuse_file_info *fi);
16. int fcfuse_write(const char *path, const char *buf, size_t size, off_t offset, struct fuse_file_info *fi);
17. int fcfuse_statfs(const char *path, struct statvfs *statv);
18. int fcfuse_release(const char *path, struct fuse_file_info *fi);
19. int fcfuse_opendir(const char *path, struct fuse_file_info *fi);
20. int fcfuse_readdir(const char *path, void *buf, fuse_fill_dir_t filler, off_t offset, struct fuse_file_info *fi);
21. int fcfuse_releasedir(const char *path, struct fuse_file_info *fi);
22. int fcfuse_fsyncdir(const char *path, int datasync, struct fuse_file_info *fi);
23. void *fcfuse_init(struct fuse_conn_info *conn);
24. void fcfuse_destroy(void *userdata);
25. int fcfuse_access(const char *path, int mask);
26. int fcfuse_ftruncate(const char *path, off_t offset, struct fuse_file_info *fi);
27. int fcfuse_fgetattr(const char *path, struct stat *statbuf, struct fuse_file_info *fi);

2. Test the developed program: It's your responsibility to test the developed file system. You may use the programs in benchmark directory to experience the usage of file containers. The producer program takes inputs from stdin and write the content to a file in a container. The validate program reads data from a container as well as stdin and compare their content.
* To test your program, you should try the following
```
# create data in different containers, for example:
./producer {mount_point}/a.bin 1 < input_file_for_container_1
./producer {mount_point}/a.bin 2 < input_file_for_container_2
# this will generate two files with the same name in each container. However, since each container is isolated, they should all exist with different content.
# to validate this, you can use the validate program
./validate {mount_point}/a.bin 1 < input_file_for_container_1
./validate {mount_point}/a.bin 2 < input_file_for_container_2
If you receive Pass on both, it's a good sign now!
```

*** We will unmount the file system and remount to test if everything works still.**

Turn ins

You **only need to (or say you can only) turn in the fcfuse_functions.c and fcfuse_extra.h** file in the src directory. All your modifications should be limited within these two files. **Exactly 1 member** of each group should submit the source code.

All group members' names and Unity IDs should be easily found in a single line comment at the beginning of the code in the following format:

// Project 3: 1st member's name, 1st member's Unity ID; 2nd member's name, 2nd member's Unity ID;

You need to name the tarball as {1st_members_unityid}_{2nd_members_unityid}_fcontainer.tar.gz

Reference and hints

1. You should go through the example and docs in libfuse-2.9 github repo https://github.com/libfuse/libfuse/tree/fuse-2_9_bugfix

2. You may reference this tutorial https://www.cs.nmsu.edu/~pfeiffer/fuse-tutorial/ to see how fuse interact with the system. However, we have real device -- npheap, you cannot really use the way how they implement the file system.

3. Here is another good reference regarding fuse https://engineering.facile.it/blog/eng/write-filesystem-fuse/

4. You will find out that chapter 39 and chapter 40 of the textbook are really useful.

Submission status

Submission status	No attempt
Grading status	Not graded
Due date	Friday, November 30, 2018, 12:00 AM
Time remaining	Assignment is overdue by: 98 days 21 hours
Last modified	-

Submission comments

➕ Comments (0)

Add submission

Make changes to your submission

◀ Resource containers -- memory

Jump to...

Reading quiz for Arpaci-Dusseau Chapter 2, 4, 6, "THE", and "Hydra" ▶