

REPORT

Project 3 – FUSE 기반 File System Implementation

| | |
|--------|----------------|
| 과 목 | 운영체제론 |
| 담당교수 | 엄영익 교수님 |
| 학 과 | 컴퓨터공학과 |
| 조 원 1 | 3 학년 |
| 이 름 | 2012314092 김효상 |
| 조 원 2 | 4 학년 |
| 이 름 | 2013312164 오수진 |
| 제 출 일 | 2016.12.9 |

A. FUSE 조사 및 분석

a. FUSE 소개

- Filesystem in Userspace (FUSE)는 UNIX 나 LINUX 계열 컴퓨터에 적재할 수 있는 kernel 모듈로, 사용자가 kernel 코드를 작성, 수정하지 않고 자신의 파일 시스템을 작성하는 것을 허용하는 시스템이다. FUSE 는 사용자 공간에서 실행 중인 파일 시스템 코드에 의해 수행되며, FUSE 모듈은 실제 kernel 인터페이스와 Userspace 사이를 연결하는 역할을 수행한다.
- FUSE 는 가상 파일 시스템 (Virtual File System: VFS)을 작성하는데 유리하다. 필수적으로 디스크에 데이터를 저장하고 찾아오는 방식을 사용하는 전통적인 파일 시스템과는 달리, 가상 파일 시스템은 실제로 데이터 자체를 저장하지 않는다. 따라서 가상 파일 시스템에서의 FUSE 은 존재하는 파일 시스템이나 저장 장치에 대한 견해나 해석의 역할을 한다.
- FUSE 는 원래 A Virtual Filesystem (AVFS)의 한 부분이었지만, 현재는 SourceForge.net 의 프로젝트로 쪼개져 나왔으며 Linux, FreeBSD, NetBSD, Open Solaris, Mac OS X 등에서 다양한 운영체제가 FUSE를 지원하기 때문에 FUSE 을 기반으로 작성된 file system은 여러 운영체제에서도 사용이 가능하다.
- FUSE 의 단점으로는 계층이 추가됨에 따라 속도저하가 발생할 수도 있다는 점이 있다.

b. Structure of FUSE

- FUSE 의 구조는 아래의 그림과 같다.

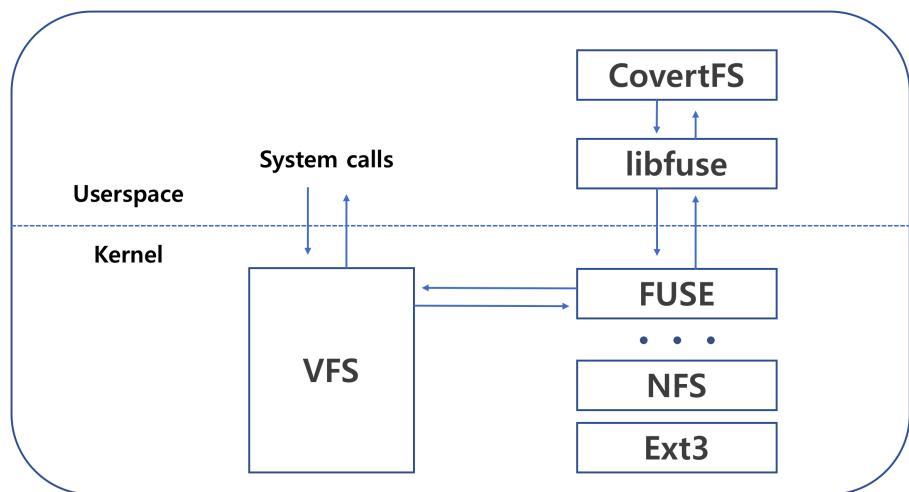


그림 1. FUSE 의 구조

- File system 을 구현하기 위해서는 kernel 내에서 구현되어야 한다. 하지만 kernel 내에서의 구현이 그 개발이 어렵다. FUSE 는 kernel 공간에 존재하며, kernel 인터페이스와 Userspace 사이를 연결하는 역할을 한다. 따라서 개발자가 kernel 이 아닌 Userspace 내에서 개발을 하며, 이 때에 요구되는 모든 작업은 FUSE에서 담당하기 때문에, 개발자는 File System을 구현하는 데 있어서 kernel 의 코드를 수정할 필요가 없다.

c. Virtual File System (VFS)

- File System Interface 을 Userspace 에 제공하기 위해 구현된 kernel 의 서브 시스템으로써, 실제 File System 의 구현과 운영체제 사이에 존재하는 추상화 계층을 의미한다.
- 운영체제가 각기 다른 종류의 file system 에 접근할 때, 그들의 차이를 느끼지 못하도록 완충, 번역의 역할을 하기 때문에, 모든 File System 은 공존하는 것뿐만이 아니라, 상호 동작하기 위해서 VFS 에 의존한다.

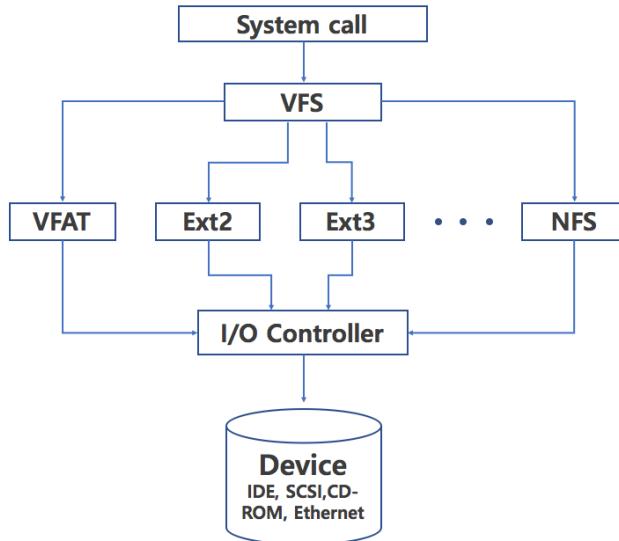


그림 2. VFS 의 구조

d. Usage of FUSE

- FUSE 을 기반으로 하는 다양한 파일 시스템이 존재한다.

| File System | Description |
|----------------------------|--|
| SSHFS | SSH를 통해 원격의 파일 시스템에 접근하는 것을 제공하는 파일 시스템. |
| GmailFS | Gmail에서 데이터를 메일의 형태로 저장하게 해주는 파일 시스템. |
| EncFS | 암호화된 가상 파일 시스템. |
| TweetFS | 트위터 유저의 상태정보들을 파일의 형식으로 관리하게 해주는 FUSE 기반 파일 시스템. |
| InterPlanetary File System | 분포, 분산된 인터넷 파일 시스템. |
| Virtualbox-FUSE | VirtualBox VDI 이미지 파일의 마운팅을 하용하는 파일 시스템. |
| Fuse-Zip | zip 파일을 파일 시스템으로 사용하는 것을 허용하는 파일 시스템. |
| Google-Drive-Ocamlfuse | 구글 드라이브에서 사용되는 FUSE 파일 시스템. |
| PNGDrive | 데이터가 암호화되어 이미지에 저장되게끔 하는 FUSE 파일 시스템. |
| DBFS | Oracle Database 파일 시스템. |

표 1. FUSE 기반의 파일 시스템 예

- 이 외에도 FUSE 는 개발자가 File system 을 개발하는 과정에서 중간 중간에 test 을 진행할 때, 굳이 kernel 을 사용하여 testing 을 진행하지 않고 FUSE 을 사용하여 Userspace 내에서 testing 을 진행한다. .

B. FUSE Design

a. FUSE System layout

본 프로젝트의 목표는 FUSE 을 기반으로 Hierarchical Directory Structure 구조를 지원하는 File System 을 구현하는 것이다.

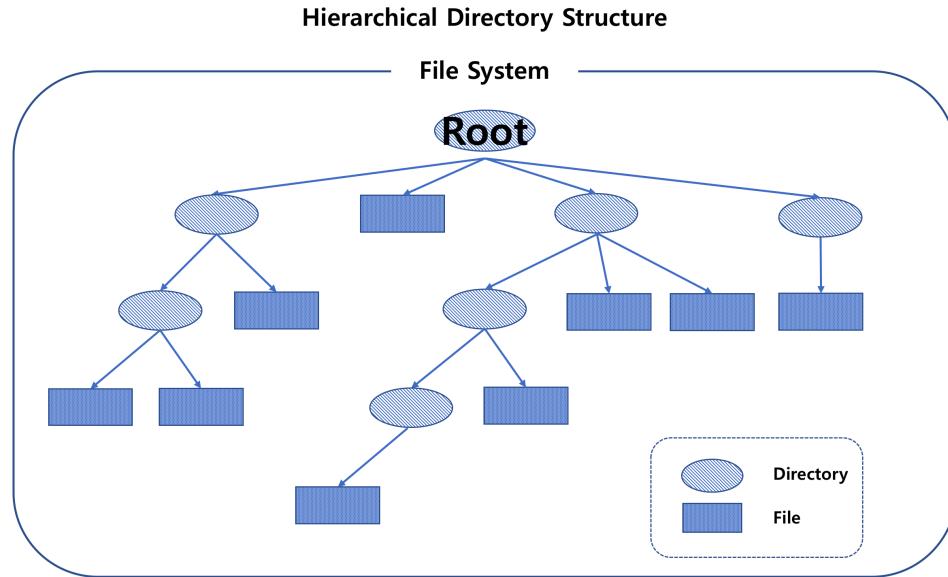


그림 3. Hierarchical Directory Structure

root 가 존재하고 계층 구조를 이루고 있으며, 2 개의 object, directory 와 file 로 구성되어 있다. Root 를 제외한 모든 object 의 parent 는 directory 이다. 즉, file 은 children 을 가지지 못한다.

b. Inode 의 구조

본 프로젝트에서는 Directory 와 File 마다 각각의 정보를 저장하는 Inode 가 필요하다. 이를 위해서 기본적인 Inode 의 구조를 먼저 정의하였다. 본 프로젝트 구현에 있어, Directory 와 File 의 Inode 구조가 매우 유사하기 때문에 이를 위한 구조체로 아래와 같이 하나의 구조체를 정의한다.

| Component | Description | Inode structure | |
|------------|------------------------|-----------------|------|
| | | Directory | File |
| name | name | use | use |
| mode | permission | use | use |
| uid | user id | use | use |
| gid | group id | use | use |
| size | file size | | use |
| ctime | change time | use | use |
| mtime | modification time | use | use |
| atime | access time | use | use |
| data | file data | | use |
| type | 0 : file 1 : directory | use | use |
| *parent | parent directory | use | use |
| **children | children | use | |
| cNum | number of children | use | |

표 2. 본 프로젝트 시스템의 Inode 의 구조

Directory 와 File 의 Inode 는 공통적으로 자신의 이름(name)과 권한 정보(permission), 사용자 id, 그룹 id, 파일과 관련된 시간, Inode type, parent directory 을 가진다. 권한 정보(permission)은 8 진수의 형식으로 저장되어 있으며, r(read)는 4, w(write)는 2 그리고 x(execute)는 1 로 저장된다. 예를 들어, rwxr-xr--의 경우 8 진수 형식의 754 퍼미션으로 변환되어 0754 로 저장된다. 사용자 id 와 그룹 id 的 경우 리눅스 운영체제의 user id 를 이용하며, Directory 나 File 이 처음 생성될 때 기본적으로 사용자 id 와 그룹 id 는 같은 값으로 들어간다. 파일과 관련된 시간으로는 change time 과 modification time 그리고 access time 이 있다. Change time 은 metadata 가 변하면 갱신되며, modification time 은 수정이 발생하면 갱신된다. 또한 access time 은 수정과 관계없이 Directory 혹은 file 에 접근이 발생하면 갱신된다. Inode type 은 해당 inode 가 Directory 인지 file 인지를 확인할 수 있는 요소로, Directory 의 경우 1, File 의 경우 0 이 저장된다. 마지막으로 Directory 와 File 모두 부모 directory 을 가리키는 포인터를 가진다.

Directory 의 경우, 자식을 가리키는 포인터 **children 과 자식의 수를 저장하는 cNum 을 가진다. 자식의 수는 정해지지 않고, 가변적이다. Linked list 을 이용하여 자식을 연결하여 하나의 일련의 linked list 로 구현할 수도 있지만, 본 프로젝트에서는 각각의 children 을 가리키는 포인터와 그 수를 저장하는 방식으로 구현하였다.

File 의 경우, file 의 크기와 해당 file 의 data 를 가진다.

c. Inode 간의 관계

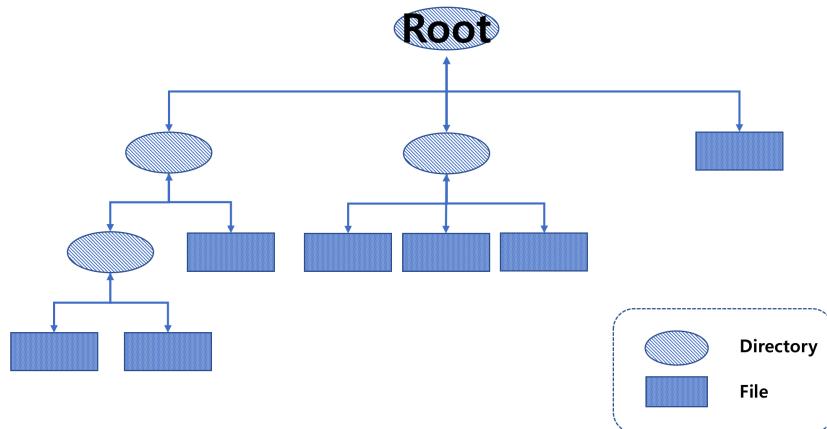


그림 4. Relation of Inode structure

<그림 4>는 2 개의 object, directory 와 file 로 계층 구조를 이루고 있는 본 프로젝트의 File System 을 도식적으로 표현한 것이다. 모든 object 은 각각의 고유한 Inode 을 가지고 있으며, 해당 Inode 은 주변 Inode 와 관계를 이루고 있다. Root 의 경우 parent 가 존재하지 않으며, root 는 모든 Inode 의 최상층의 parent 이다. Root 를 제외한 모든 object 의 parent 는 directory 이다. 즉, file 은 children 을 가지지 못한다. 각각의 Inode 에는 parent object 을 가리키는 pointer 와 children objects 을 가리키는 pointers 를 가진다. Parent object 의 경우 각 Inode 마다 단 하나밖에 존재할 수 없으며, children object 의 경우 그 수는 제한되어 있지 않다.

C. FUSE Implementation

a. 구현 환경

OS: LINUX UBUNTU 16.04 LTS / FUSE: Fuse Version 2.9.4

b. error 처리

<표 2>는 FUSE에서 제공하는 error 리스트와 본 프로젝트에서의 처리 여부를 나타낸다.

| Error name | Description | 처리 여부 |
|-----------------|--|-------|
| errno.ENOSYS | Function not implemented errno.EROFS: Read-only file system | △ |
| errno.EPERM | Operation not permitted | X |
| errno.EACCES | Permission denied | O |
| errno.ENOENT | No such file or directory | O |
| errno.EIO | I/O error | X |
| errno.EEXIST | File exists | O |
| errno.ENOTDIR | Not a directory | △ |
| errno.EISDIR | Is a directory | △ |
| errno.ENOTEMPTY | Directory not empty | O |

표 2. 본 프로젝트 시스템의 Inode의 구조

본 프로젝트에서는 errno.EACCES 와 error.ENOENT, errno.EEXIST 그리고 errno.ENOTEMPTY 의 처리를 지원한다. Object에 대한 접근 명령이 주어진 경우, 해당 Object의 permission을 확인하여 허용되지 않는 permission에 대한 명령이 주어진 경우, errno.EACCES error를 호출한다. Path의 입력이 주어졌을 때, 해당 Path에 file이나 directory가 존재하지 않는다면, error.ENOENT error를 호출한다. 또한, 파일의 생성을 위한 path가 주어졌을 때, 이미 해당 path에 file이 존재한다면, error.EEXIST error를 호출한다. 마지막으로, Directory의 삭제나 이동에 대한 명령이 주어진 경우, 그 Directory가 비어있지 않으면, errno.ENOTEMPTY error를 호출한다.

본 프로젝트에서는 위의 4 가지의 error 외에도 3 가지의 error, errno.ENOSYS, errno.ENOTDIR, errno.EISDIR의 처리를 다른 방식으로 지원한다. 우선 errno.ENOTDIR, errno.EISDIR의 경우, Inode에 directory와 file을 구분 짓는 변수로 'type'를 가지기 때문에 error의 호출 없이도 그것의 처리가 가능해진다. errno.ENOSYS의 경우, 명령을 실행하는 함수를 호출할 때마다 해당 명령과 object의 권한을 확인하여 그것이 가능한지를 확인하여 불가능할 경우, errno.EACCES error를 호출하는 방식으로 처리한다.

c. 구현 함수

본 프로젝트에서 FUSE 기반으로 구현한 관련 함수 리스트는 아래와 같다.

| Classification | Function name | Description |
|---------------------------|---------------|---|
| File attribute | OSPJ_getattr | Get attribute of file |
| Create & Remove Directory | OSPJ_mkdir | Create Directory |
| | OSPJ_rmdir | Remove Directory |
| Create & Remove File | OSPJ_mknod | Create File |
| | OSPJ_unlink | Remove File |
| Read Directory | OSPJ_readdir | Read objects in Directory and print out |
| Read & Write File | OSPJ_open | Open File |
| | OSPJ_read | Read File |
| | OSPJ_write | Write File |
| Change attribute | OSPJ_chmod | Change permission mode |
| | OSPJ_chown | Change owner |
| | OSPJ_rename | Rename of file or move |
| Management | OSPJ_utimens | Renew time |

표 4. 본 프로젝트의 FUSE 기반으로 구현한 관련 함수 리스트

File 과 Directory 의 생성과 삭제와 관련된 함수로는 OSPJ_mknod(File 의 생성), OSPJ_unlink(File 의 삭제), OSPJ_mkdir(Directory 의 생성), OSPJ_rmdir(Directory 의 삭제)가 있다.

또한 본 프로젝트에서 위의 함수를 구현하는 과정에서 필요하여 구현한 함수 리스트는 아래와 같다.

Directory 열기 및 닫기와 관련된 함수로는 OSPJ_readdir, OSPJ_getattr 가 있다.

OSPJ_readdir 는 해당 Directory 의 object 의 정보를 읽어 그것을 출력하는 함수이며, OSPJ_getattr 함수는 file 의 정보를 읽어오는 함수이다.

File 열기와 닫기, 읽기 및 쓰기와 관련된 함수로는 OSPJ_open, OSPJ_read, OSPJ_write 함수가 있다.

다음의 함수는 추가 구현된 기능의 함수로 File 및 directory 의 Inode 정보 수정과 관련된 함수이다.

OSPJ_utimens, OSPJ_rename, OSPJ_chown, OSPJ_chmod 가 있으며, OSPJ_utimens 는 시간과 관련된 함수이다. OSPJ_chmod 와 OSPJ_chown 은 각각 permission 과 owner 을 바꾸는 함수이다. OSPJ_rename 는 parameter 로 file 이 들어온 경우 해당 file 의 이름을 rename 하고, parameter 로 directory 가 들어온 경우 해당 위치로 move 을 한다.

| Function name | Description |
|-----------------|--|
| makeFP | Find path of parent which has a given node as child. |
| search | Find node coincided with given path. |
| RemoveDirectory | Remove Directory |
| RemoveFile | Remove File |

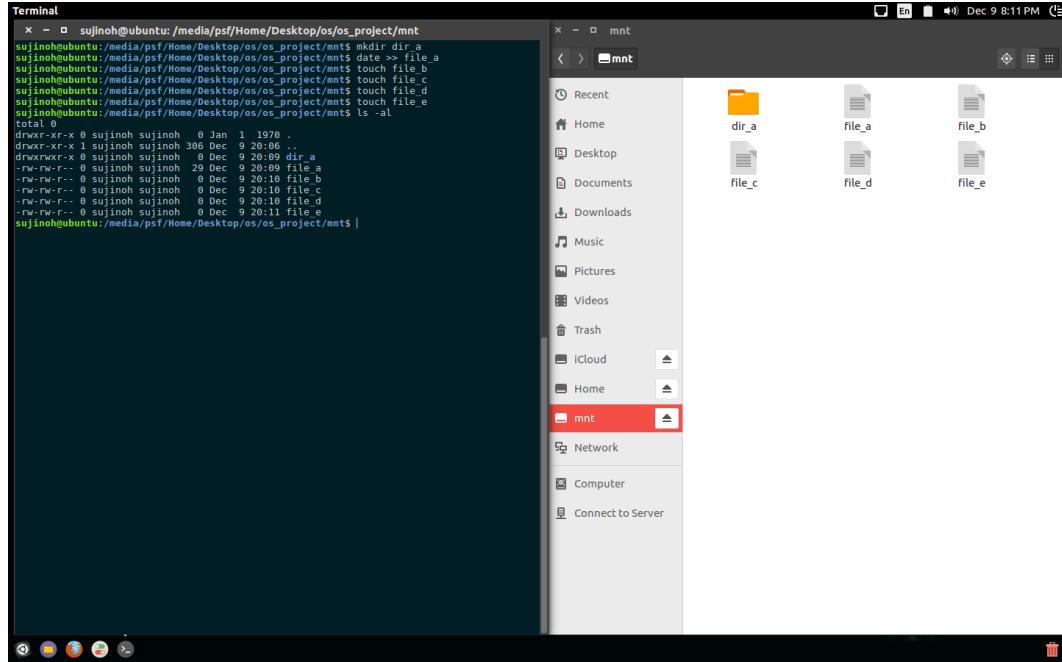
표 5. 본 프로젝트에 추가적으로 구현된 함수 리스트

makeFP 함수는 parameter 로 주어진 path 를 가진 node 의 부모 directory 의 path 를 찾는 함수이다. 또한 search 함수는 parameter 로 주어진 path 와 mode 을 가진다. Mode 는 directory 와 file 을 구분짓는 parameter 로 해당 path 와 일치하는 node 를 찾아 그것이 mode 로 들어온 것과 같은 type(directory or file)인지를 확인한다. RemoveDirectory 함수는 OSPJ_rmdir 함수에서 사용하는 함수로 parameter 로 주어진 path 의 directory 을 삭제한다. RemoveFile 함수는 OSPJ_unlink 함수에서 사용하는 함수로 parameter 로 주어진 path 의 file 을 삭제한다.

d. FUSE 실행 화면

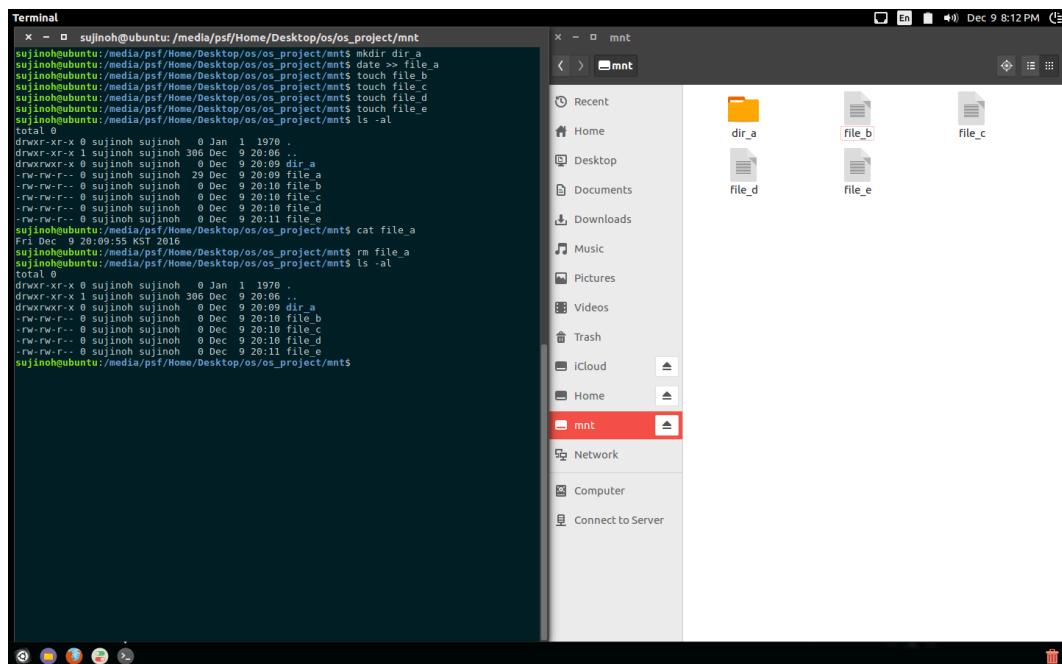
4 개의 시나리오를 정하여 이를 이용하여 FUSE 실행 화면을 보인다.

1) 시나리오 (1): File 과 Directory 생성 및 chmod 을 이용한 권한 변경



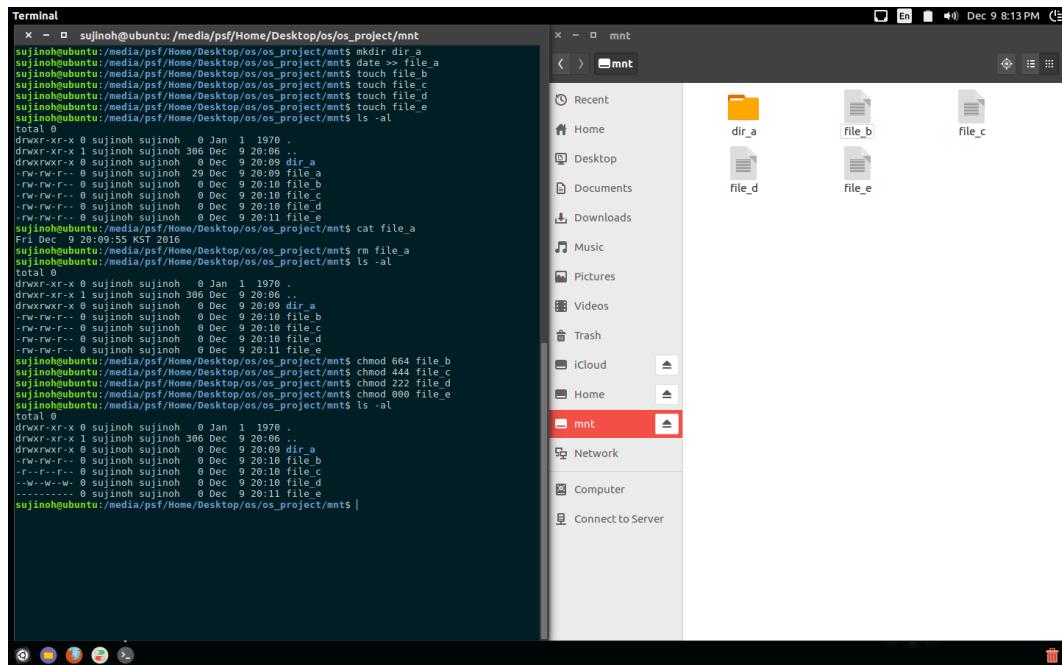
Screenshot 1. 시나리오 (1) Dummy directory 와 file 생성 화면

Dummy data 로 directory 'dir_a'와 file 'file_a', 'file_b', 'file_c', 'file_d', 'file_e'를 생성한 화면이다. 이 때, 'file_a'는 data 로 현재 날짜와 시간을 가지는 file 이며, 나머지 file 은 빈 파일로 생성한다.



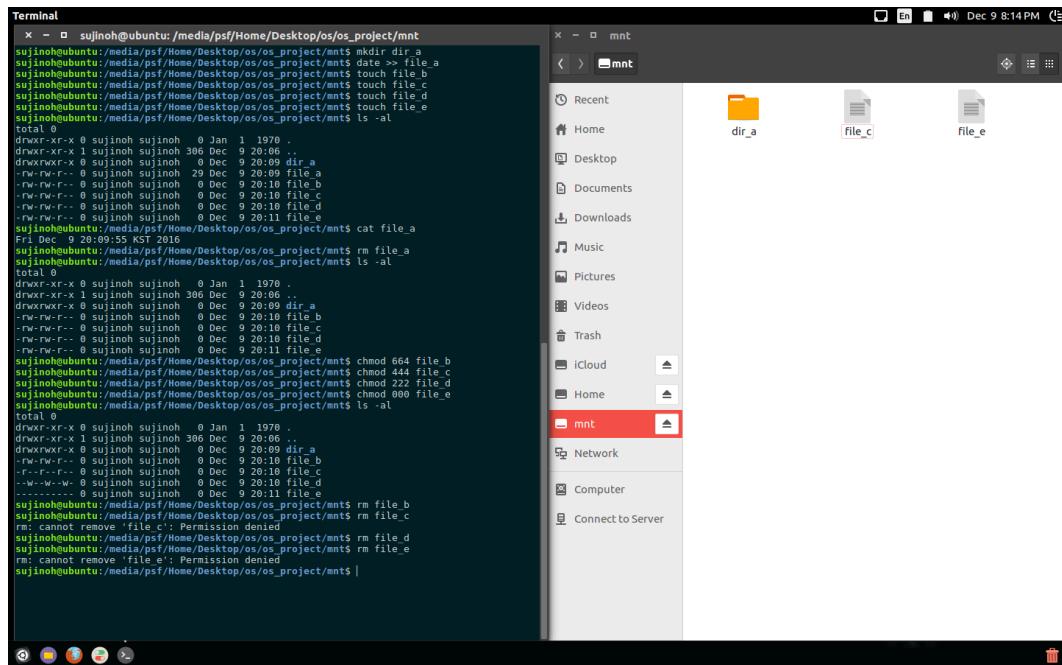
Screenshot 2. 시나리오 (1) 'file_a'의 실행 및 삭제 화면

'file_a'를 open 하여 data 로 현재 날짜와 시간을 가지는 것을 확인한다. 그리고 'file_a'를 삭제하는 명령을 수행한다.



Screenshot 3. 시나리오 (1) File 의 권한(permission) 변경 화면

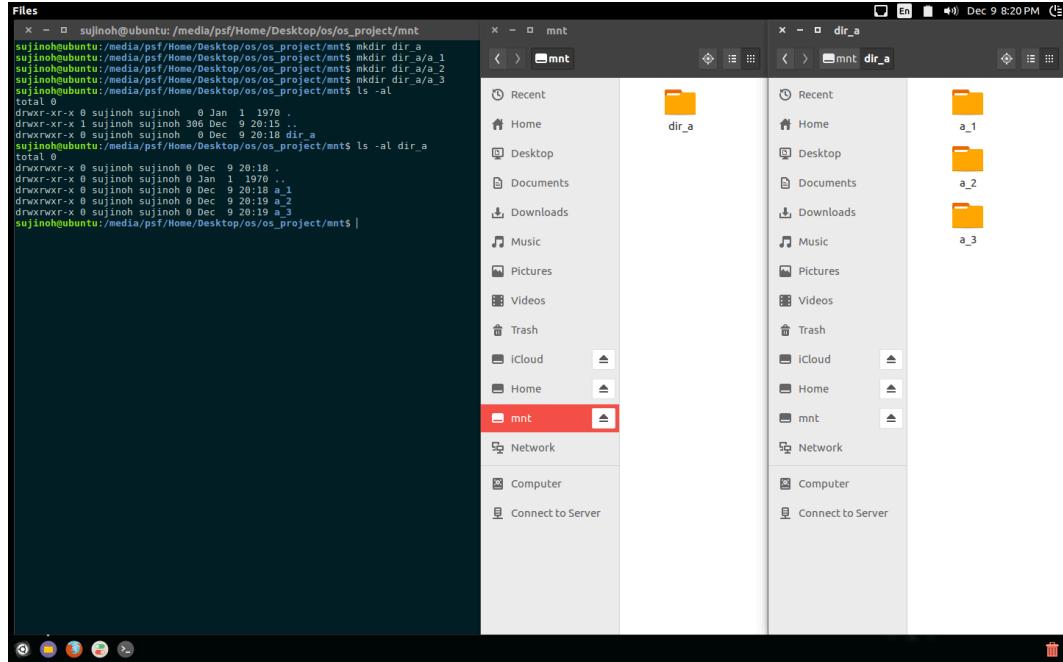
file 'file_b', 'file_c', 'file_d', 'file_e'의 권한(permission)을 변경하는 명령을 수행한 화면이다. 기존의 file 들은 'rw-rw-r--', 즉 664 permission 을 가지도록 설정되어 있었다. 여기서 'file_b'는 664 permission 을 그대로 가지고, 'file_c', 'file_d' 그리고 'file_e'는 각각 'r--r--r--'(444 permission), '--w--w--w'(222 permission), '-----'(000 permission)으로 권한을 변경하였다.



Screenshot 4. 시나리오 (1) 권한에 따른 file 삭제 화면

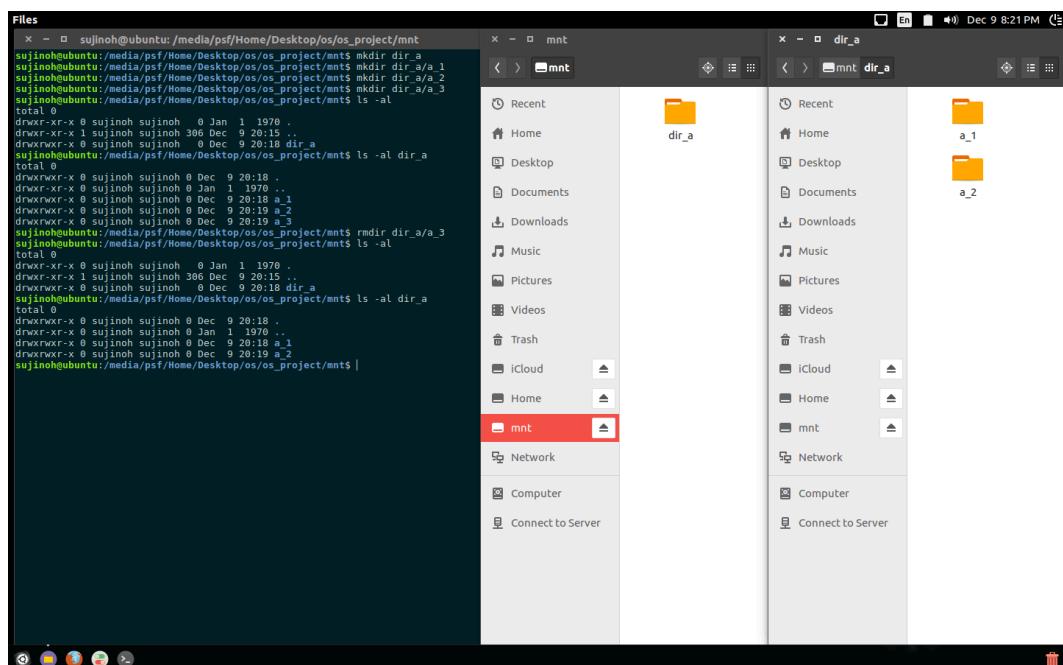
file 'file_b', 'file_c', 'file_d', 'file_e'에 삭제 명령을 수행하였다. 이 때 앞 단계에서 설정된 권한에 따라 삭제 명령이 수행되기도 거절되기도 하였다. 'file_c'와 'file_e'의 경우, w(write) 권한이 없어 삭제 명령이 거절되었으며, 'file_b'와 'file_d'의 경우 w(write) 권한이 있어 삭제 명령이 수행되었다.

2) 시나리오 (2): 다단계 Directory 생성 및 삭제



Screenshot 5. 시나리오 (2) Dummy directory 생성 화면

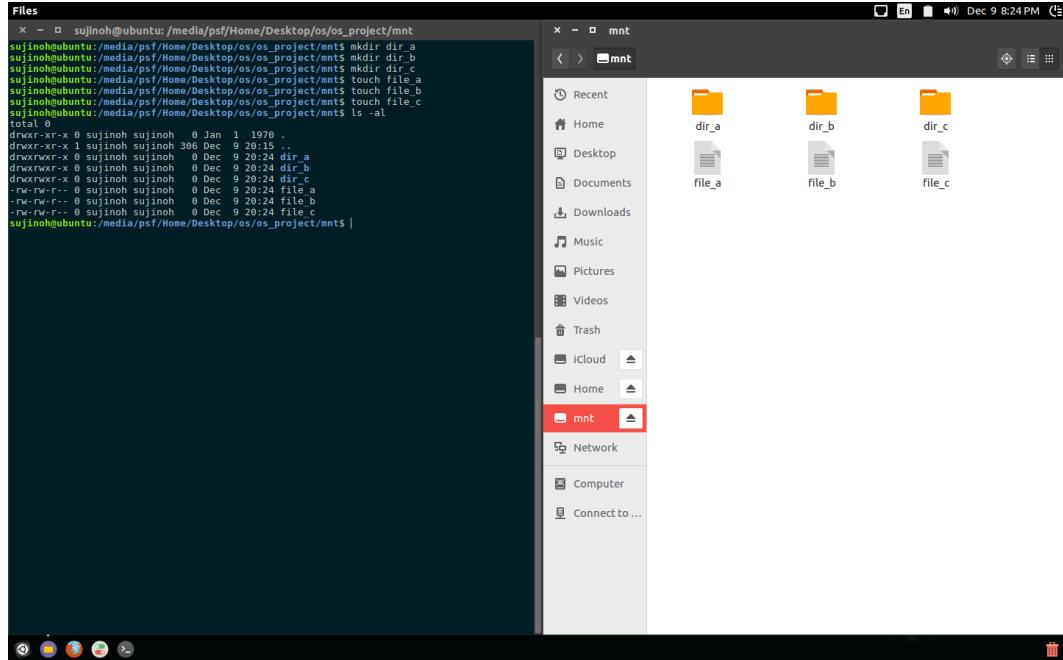
Dummy data로 directory 'dir_a'와 그 하위 폴더로 directory 'a_1', 'a_2', 'a_3'을 생성한다.



Screenshot 6. 시나리오 (2) directory 'a_3'의 삭제 화면

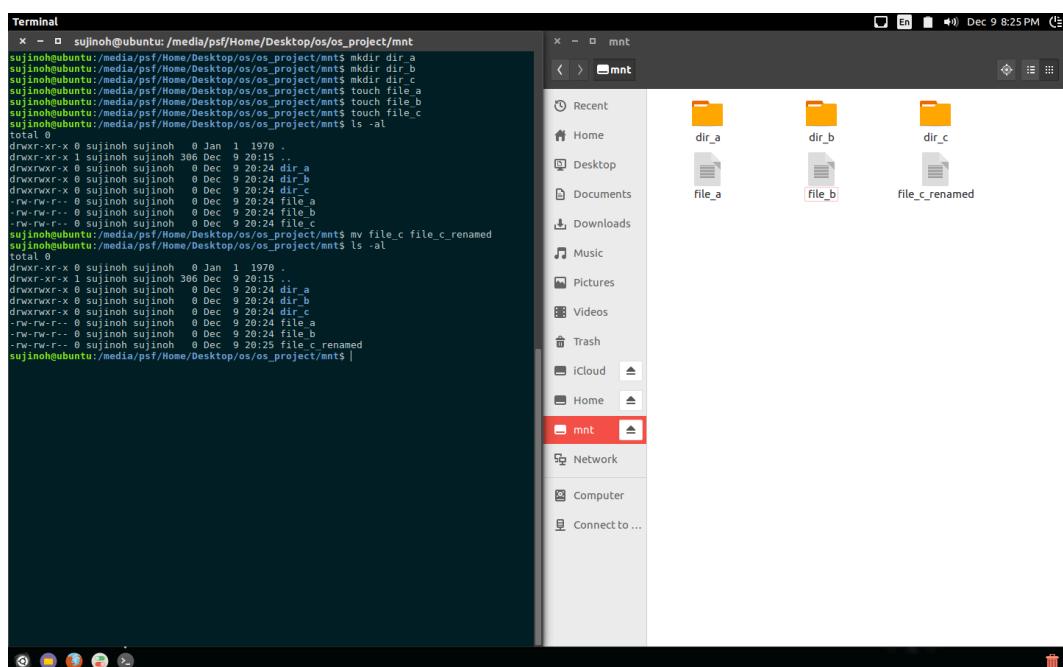
directory dir_a의 하위 폴더 directory 'a_3'을 삭제하는 명령을 수행한다.

3) 시나리오 (3): File의 rename을 이용한 이름 변경



Screenshot 7. 시나리오 (3) Dummy directory와 file 생성 화면

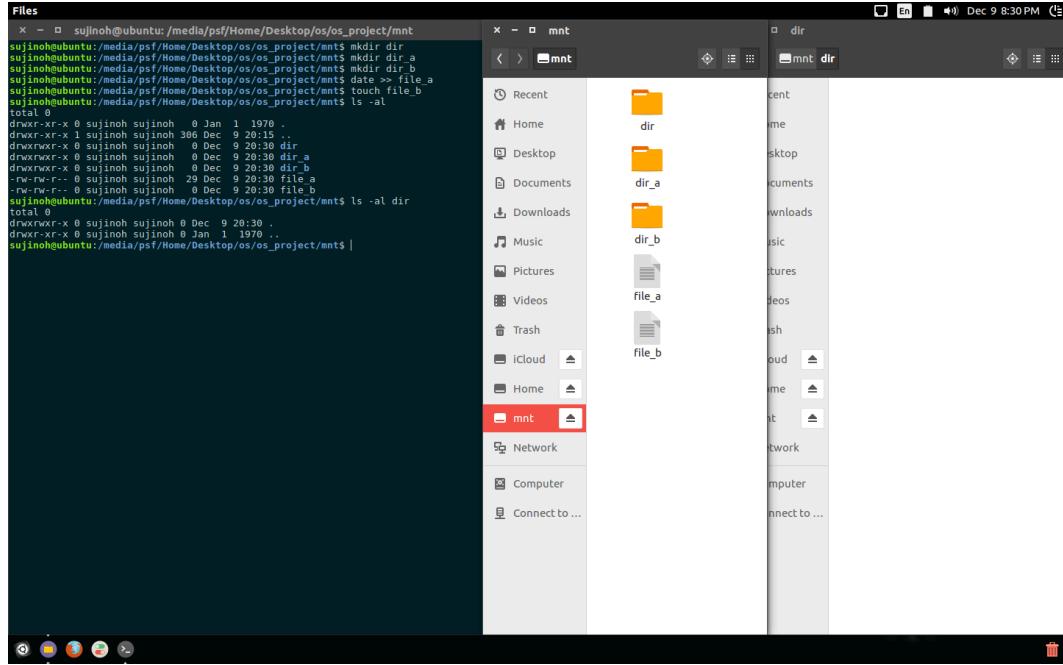
Dummy data로 directory 'dir_a', 'dir_b', 'dir_c'와 file 'file_a', 'file_b', 'file_c'를 생성한 화면이다.



Screenshot 8. 시나리오 (3) 'file_c'의 이름 변경 화면

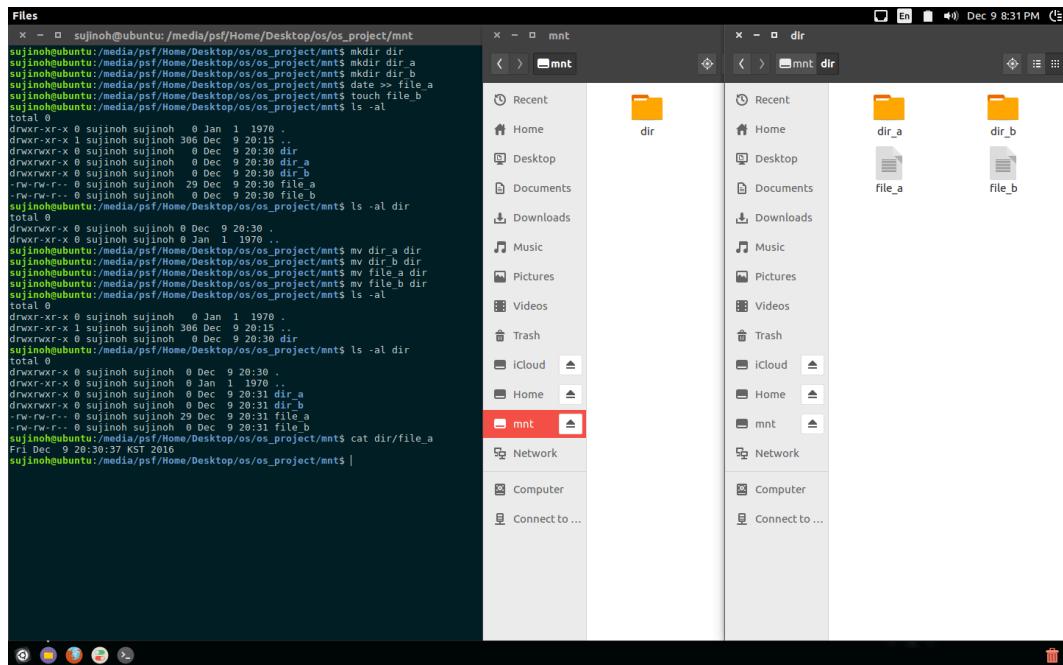
'file_c'의 이름을 rename을 통하여 'file_c_renamed'으로 변경하는 명령을 수행한다.

4) 시나리오 (4): directory 와 file 의 directory 이동



Screenshot 9. 시나리오 (4) Dummy directory 와 file 생성 화면

Dummy data로 directory 'dir', 'dir_a', 'dir_b'와 file 'file_a', 'file_b'을 생성한 화면이다. 이 때, 'file_a'는 data로 현재 날짜와 시간을 가지는 file이며, 'file_b'은 빈 파일로 생성한다.



Screenshot 10. 시나리오 (4) Dummy directory 와 file 의 이동 명령 실행 화면

Dummy directory 'dir_a', 'dir_b'와 file 'file_a', 'file_b'을 directory 'dir'로 이동하는 명령을 수행한다. 이동이 제대로 이루어졌는지를 확인하기 위해 'file_a'를 open 하여 그 내용을 확인한다.

D. Conclusion

a. 한계

본 프로젝트에서 FUSE 라이브러리를 기반으로 file system 이 작성되었다. 이로 인해, 해당 file system 을 마운트하는 유저 외에는 Fuse 의 접근이 차단되어 소유주 이외의 사용자는 해당 파일에 접근할 수 없었다. Chown 함수의 구현을 통해, 그룹 사용자와 기타 사용자를 변경할 수 있도록 되어있지만 실질적으로는 FUSE 라이브러리 한계상 그 기능이 실행되지는 않는다.

또한, permission 에서도 FUSE 라이브러리의 한계로 문제가 발생하였다. W(write)의 권한이 없는 경우 파일의 삭제나 permission 의 변경을 제한하였는데, 이렇게 되면 한번 w(write) 권한을 잊은 경우 다시는 그 권한을 얻을 수가 없었다. 기존의 리눅스의 경우, 이러한 경우에도 'sudo' 명령어를 통해 접근하는 것이 가능하였지만 현 프로젝트에서는 'sudo' 명령어의 접근을 확인하지 못하여, 이를 구현할 수가 없었다.

Error 처리 한계로 인해, 몇 개의 error 의 경우, 다른 방식으로 error 을 처리하였다.

File system 은 kernel 안에서 개발되기 때문에 쉽게 접하기 힘든 주제의 개발이다. 하지만 본 프로젝트에서 FUSE 를 통해 kernel 이 아닌 Userspace 에서도 file system 의 구현이 가능하였다. 본 프로젝트를 통해 file system 을 구현하여 보니, 그것에 대한 전반적인 이해가 잘 되었다.

E. Reference

- 1 https://en.wikipedia.org/wiki/Filesystem_in_Userspace
- 2 FUSE Official Webpage: <http://fuse.sourceforge.net/>
- 3 Libfuse API documentation: <http://libfuse.github.io/doxygen/>
- 4 FUSE Documentation:
https://www.cs.hmc.edu/~geoff/classes/hmc.cs135.201001/homework/fuse/fuse_doc.html
- 5 A. Silberschatz, et. Al., Operating System Concepts, 8-ed., Wiley, 2010.
- 6 Documentation: The Mode Bits for Access Permission
http://www.gnu.org/software/libc/manual/html_node/Permission-Bits.html
- 7 The Open Group Base Specifications Issue 7. IEEE Std 1003.1™-2008, 2016 Edition
<http://pubs.opengroup.org/onlinepubs/9699919799/>