REPORT

보고서 작성 서약서

1. 나는 타학생의 보고서를 복사(Copy)하지 않았습니다.
2. 나는 타학생의 보고서를 인터넷에서 다운로드 하여 대체하지 않았습니다.
3. 나는 타인에게 보고서 제출 전에 보고서를 보여주지 않았습니다.
4. 보고서 제출 기한을 준수하였습니다.

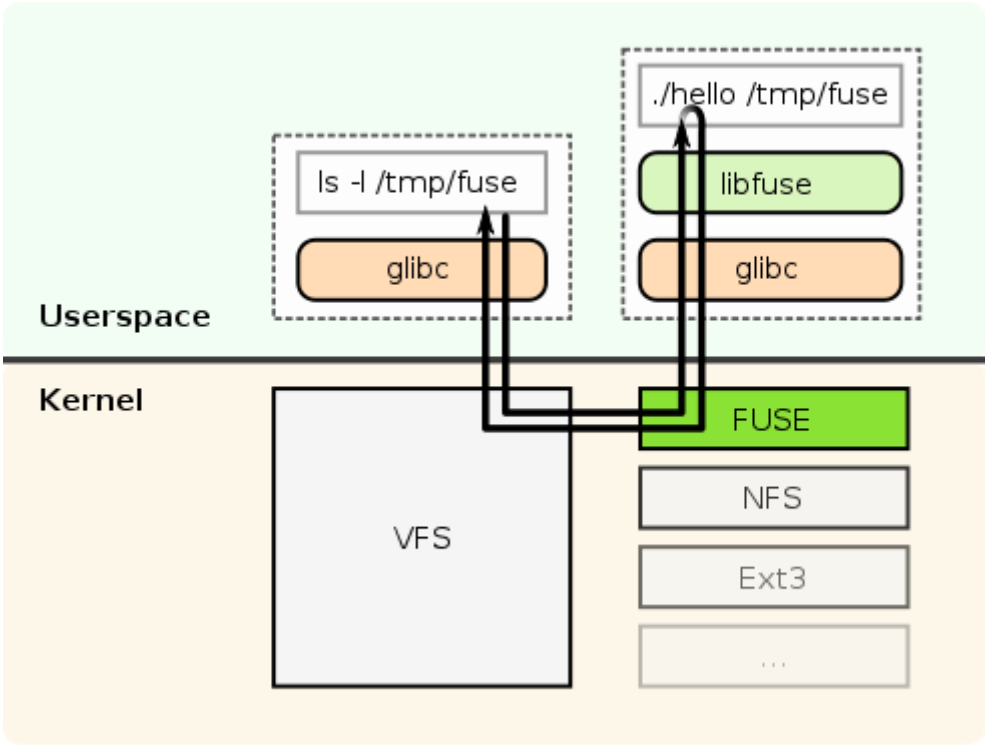
나는 보고서 작성시 위법 행위를 하지 않고,
성.균.인으로서 나의 명예를 지킬 것을 약속합니다.

과 목	: 운영체제론
과 제 명	: Project #3
담 당 교 수	: 엄 영 익 교수님
이 름	: 권현준, 조영도
제 출 일	: 2016-12-12

목차

1. 개요

리눅스 운영체제중 하나인 Ubuntu 에서 사용자 공간 파일시스템인 FUSE 를 이용하여 현재 실행 중인 프로세스 정보를 디렉터리 목록으로 보여주는 파일 시스템을 구현한다. C 언어와 기본 C 라이브러리인 glibc, FUSE 라이브러리만 사용하고 소스파일에 모든 기능을 구현한다.



Structural diagram of Filesystem in Userspace

1.1.FUSE 소개

FUSE 는 Filesystem in USEr space 의 약자로 Unix 계열 컴퓨터 운영체제에서 사용가능한 소프트웨어 인터페이스입니다. Loadable Kernel Module 로 설치가 간단하며, 루트권한이 없는 사용자도 커널 코드를 수정하지 않고 자신만의 File system 을 만들 수 있는 기능을 제공합니다. FUSE 는 file system code 를 사용자 공간에서 실행할 수 있도록 도와주기에, FUSE module 은 실제 커널 인터페이스의 '중개자' 역할을 한다고 볼 수 있습니다.
FUSE 는 초기부터 free software 로 시작되었고, 라이선스는 GNU GPL 과 GNU LGPL 에 따라 release 되고 있습니다. FUSE 는 리눅스 커널 2.6.14 부터 공식적으로 흡수되었습니다.
FUSE 는 VFS(Virtual FIle System)을 사용하는데 있어 특히 유용합니다. 디스크에 데이터를 읽고 쓰는 것을 기본으로 한 전통적인 파일 시스템과 달리, VFS 는 실제로 데이터를 파일 시스템에 저장하지 않습니다. FUSE 는 어플리케이션 레벨에서 작업이 이루어질 수

있도록 안정성 등 더 나은 장점을 제공하며, 리눅스 이외의 운영체제에서도 FUSE 가 사용 가능하기 때문에 한 번 작성된 사용자 파일 시스템을 여러 운영체제에서 작동시킬 수 있습니다. 그러나 계층이 추가됨으로 인해 속도저하가 발생할 수 있습니다.

1.2. How FUSE works - FUSE library

맨 처음 사용자가 작성한 user mode 의 프로그램을 실행시켜 함수 fuse_main() (lib/helper.c)을 호출하면, 그 함수는 자신에 입력된 인수를 분석한 뒤, 함수 fuse_mount() (lib/mount.c)를 호출합니다. 함수 fuse_mount()는 하나의 UNIX domain socket pair 를 생성하고 환경변수 FUSE_COMMFD_ENV 안의 socket 끝을 지나가는 fusermount (util/fusermount.c)을 나누고 실행합니다. 이 fusermount 는 fuse module 이 제대로 load 되었는지 확실하게 하며, 폴더 /dev/fuse 에서 file handle 을 UNIX domain socket 을 통해 다시 fuse_mount()로 전달합니다. 그리고 fuse_mount()는 받은 file handle 을 fuse_main()로 반환합니다. 함수 fuse_mount()는 파일시스템 데이터의 cache 이미지를 유지하고 저장할 수 있는 FUSE 구조체를 할당하는 함수 fuse_new() (lib/fuse.c)를 호출합니다. 최종적으로 fuse_main() 는 함수 fuse_loop_mt() (lib/fuse_mt.c)와 함수 fuse_loop() (lib/fuse.c) 둘 중 하나를 호출하는데, 두 함수 모두 파일시스템을 읽고 user mode 프로그램에 저장된 구조체 fuse_operations 호출하는 함수입니다.

1.3. How FUSE works - The kernel module

FUSE 의 kernel module 은 두 가지로 구성되어있습니다. 첫 번째는 kernel/dev.c 파일 안에 존재하며 UNIX 계열 운영 체제에서 프로세스와 다른 시스템 정보를 계층적 파일 구조 같은 형식으로 보여주는 특별한 파일시스템인 proc 파일시스템 컴포넌트이며, 두 번째는 세 개의 파일 kernel/file.c, kernel/inode.c, kernel/dir.c 안의 파일시스템 system call 들입니다. 세 개의 파일 속 모든 시스템 콜은 세 개의 함수 request_send(), request_send_noreply(), request_send_nonblock() 를 어느 것이든 하나를 호출합니다. 호출의 대부분은 request_send() 함수인데, 이 함수는 "list of requests" 구조(fc->pending)에 요청을 붙이며, 응답을 기다립니다. 나머지 두 함수 request_send_noreply(), request_send_nonblock()는 non-blocking 이라는 특성과 응답을 받아도 답신이 없다는 특성을 제외하고는 이전에 나온 함수와 비슷합니다. 파일 /dev/fuse 에서의 IO 요청에 대한 응답은 위의 proc 파일시스템 컴포넌트가 합니다. 함수 fuse_dev_read()는 파일 읽기와 "list of requests"의 구조부터 호출 프로그램까지의 커맨드 반환을 다룹니다. 함수 fuse_dev_write()는 파일 쓰기와 쓰여진 데이터를 req->out 자료구조에 옮기는 것을 다루는데, 이 자료구조는 "list of requests" 구조체와 함수 request_send()를 통해 시스템 콜이 반환되는 곳입니다.

1.4. FUSE Operations

FUSE 에는 다양한 기능이 존재합니다. 아래의 각 함수들을 모두 구조체 fuse_operation (fuse.h)에 선언되어 있으며, 각 함수의 기능을 간략히 요약하였습니다.

int(* getattr)(const char *, struct stat *) - 파일의 속성을 읽는 함수
int(* readlink)(const char *, char *, size_t) - Symbolic 노드를 읽는 함수
int(* mknod)(const char *, mode_t, dev_t) - 파일(혹은 device special)을 생성하는 함수
int(* mkdir)(const char *, mode_t) - 디렉토리를 생성하는 함수
int(* unlink)(const char *) - 파일을 삭제하는 함수
int(* rmdir)(const char *) - 디렉토리를 삭제하는 함수
int(* symlink)(const char *, const char *) - Symbolic 노드를 생성하는 함수
int(* rename)(const char *, const char *) - 파일 또는 디렉토리 이름을 변경하는 함수
int(* link)(const char *, const char *) - Hard link 를 생성하는 함수
int(* chmod)(const char *, mode_t) - 파일 또는 디렉토리의 권한을 변경하는 함수
int(* chown)(const char *, uid_t, gid_t) - 파일 또는 디렉토리의 소유자를 변경하는 함수
int(* truncate)(const char *, off_t) - 파일의 크기를 변경하는 함수
int(* open)(const char *, struct fuse_file_info *) - System call open 을 위한 함수
int(* read)(const char *, char *, size_t, off_t, struct fuse_file_info *) - 파일 데이터를 읽는 함수
int(* write)(const char *, const char *, size_t, off_t, struct fuse_file_info *) - 데이터를 파일에 쓰는 함수
int(* statfs)(const char *, struct statvfs *) - 파일시스템에 대한 정보를 반환하는 함수
int(* flush)(const char *, struct fuse_file_info *) - 쓰여지지 않은 데이터를 쓰는 함수
int(* release)(const char *, struct fuse_file_info *) - open 과 반대 역할을 하는 함수
int(* fsync)(const char *, int, struct fuse_file_info *) - 데이터와 유저 데이터를 디스크에 쓸어보내는 함수
int(* setxattr)(const char *, const char *, const char *, size_t, int) - 확장자 속성 이름과 값을 지정하는 함수
int(* getxattr)(const char *, const char *, char *, size_t) - 경로의 확장자 속성 이름 가져오는 함수
int(* listxattr)(const char *, char *, size_t) - 이용할 수 있는 확장자 속성의 리스트를 반환하는 함수
int(* removexattr)(const char *, const char *) - 확장자 속성 이름을 제거하는 함수
int(* opendir)(const char *, struct fuse_file_info *) - 디렉토리를 여는 함수

int(* readdir)(const char *, void *, fuse_fill_dir_t, off_t, struct fuse_file_info *) - 디렉토리를 읽는 함수
int(* releasedir)(const char *, struct fuse_file_info *) - 디렉토리를 릴리즈하는 함수
int(* fsyncdir)(const char *, int, struct fuse_file_info *) - 디렉토리 컨텐츠를 동기화하는 함수
void *(* init)(struct fuse_conn_info *conn) - 파일시스템을 초기화하는 함수
void(* destroy)(void *) - 파일시스템을 종료하는 함수
int(* access)(const char *, int) - 사용자가 해당 경로를 접근할 수 있는지 알려주는 함수
int(* create)(const char *, mode_t, struct fuse_file_info *) - 해당 경로를 생성하여 열거나 또는 단순히 여는 함수
int(* ftruncate)(const char *, off_t, struct fuse_file_info *) - 오픈 파일의 크기를 바꿔주는 함수
int(* fgetattr)(const char *, struct stat *, struct fuse_file_info *) - 오픈 파일의 속성을 가져오는 함수
int(* lock)(const char *, struct fuse_file_info *, int cmd, struct flock *) - POSIX 의 파일 locking 기능을 수행하는 함수
int(* utimens)(const char *, const struct timespec tv[2]) - 파일의 접근시간과 수정시간 변경하는 함수
int(* bmap)(const char *, size_t blocksize, uint64_t *idx) - 파일과 디바이스 사이의 Block 인덱스 Mapping 하는 함수
int(* ioctl)(const char *, int cmd, void *arg, struct fuse_file_info *, unsigned int flags, void *data) - 사용자와 커널을 잇는 인터페이스 일부인 ioctl 와 관련된 함수
int(* poll)(const char *, struct fuse_file_info *, struct fuse_pollhandle *ph, unsigned *reventsp) - 기대되는 IO 이벤트들이 발생했는지 여부를 알 수 있는 함수
int(* write_buf)(const char *, struct fuse_bufvec *buf, off_t off, struct fuse_file_info *) - 오픈 파일에 버퍼에 있는 콘텐츠를 쓰는 함수
int(* read_buf)(const char *, struct fuse_bufvec **bufp, size_t size, off_t off, struct fuse_file_info *) - 버퍼에 있는 오픈 파일로 부터 데이터를 저장하는 함수
int(* flock)(const char *, struct fuse_file_info *, int op) - BSD 파일 locking 기능을 수행하는 함수
int(* fallocate)(const char *, int, off_t, off_t, struct fuse_file_info *) - 오픈 파일을 위한 공간을 할당하는 함수

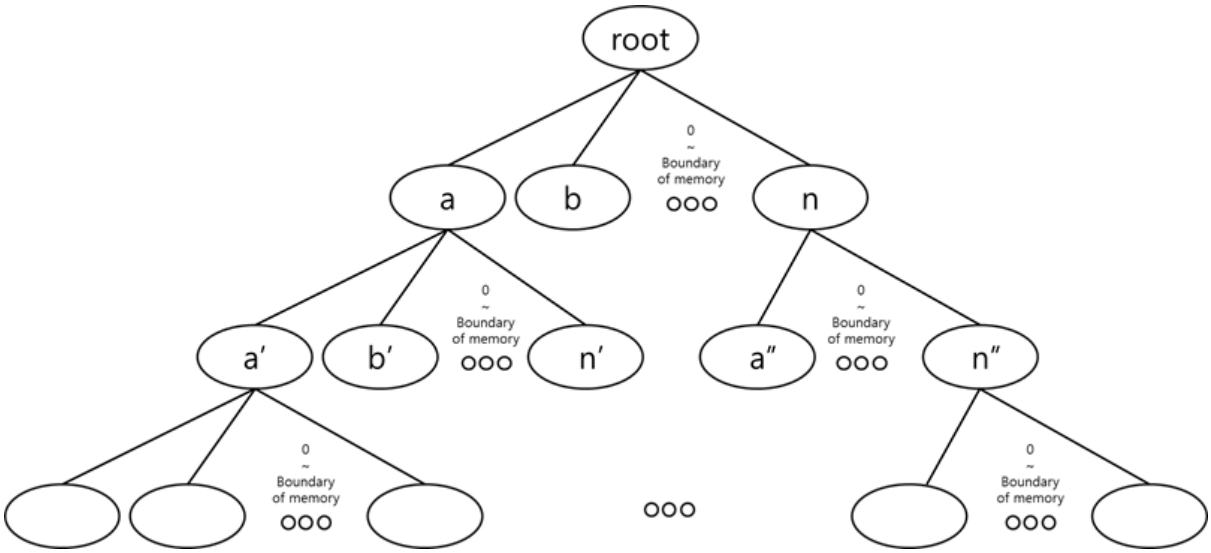
2. 분석

2.1 파일 시스템 설계 내용

파일 시스템을 설계하면서 가장 먼저 고려한 구조는 구조체를 이용한 Linked list 형식이었습니다. 설계 초기에는 디렉토리와 파일의 구조를 따로 구현하여 배열로도, 각각의 구조체로도 고려해보았으나 한 구조체안에 FILE 과 DIRECTORY 라는 정수형으로 type 을 저장해 구별할 수 있도록 하였습니다. 이렇게 통일화된 구조체를 사용하면서 구조체 포인터 변수로 상위의 경로를 찾아 갈 수 있도록 하였고, 구조체 더블 포인터 변수를 통해 하위의 여러 경로를 찾아 갈 수 있게 구현 하였습니다.
파일 시스템의 구조체는 다음과 같습니다.

```
typedef struct _MYFILESYS
{
        int type;   // Variable to assign type, 0 is file and 1 is directory
        char name[MAX]; // Name of file or directory
        long size; // Variable to assign file size
        char * data; // Variable to assign file data
        int lowerNum; // Number of lower
        mode_t mode; // Mode of file (permission)
        uid_t uid; // User id of file
        gid_t gid; // Group id of file
        time_t a_time;// Variable to notify access time
        time_t c_time; // Variable to notify changing time
        time_t m_time;// Variable to notify modification time
        struct _MYFILESYS * upper; // Upper directory, because upper directory is one
        struct _MYFILESYS ** lower;   // Lower directory, because lower directory is
                                                many
}MYFILESYS;
```

위의 구조체 형식으로 파일시스템의 자료구조는 아래와 그림과 같습니다.



위와 같은 구조는 계층적 구조로 가장 간단하게 구현 가능하고, 각 노드 간의 부모 자식관계가 실제 계층구조로 분명합니다. 또한 신규 노드의 추가는 O(1)이며, 실제로 간단합니다. 그러나 하나의 디렉터리에 수백 개 이상의 파일 또는 디렉터리가 있는 경우 n 을 자식 노드의 개수라 할 때, O(n)의 탐색시간을 가지고, 기존 노드의 삭제 경우 O(n)의 시간이 걸리는 점은 단점이라 할 수 있습니다.

2.2 구현 기능 설명

이번 파일 시스템 구현 프로젝트에서 다음과 같은 기능들을 구현 하였습니다.

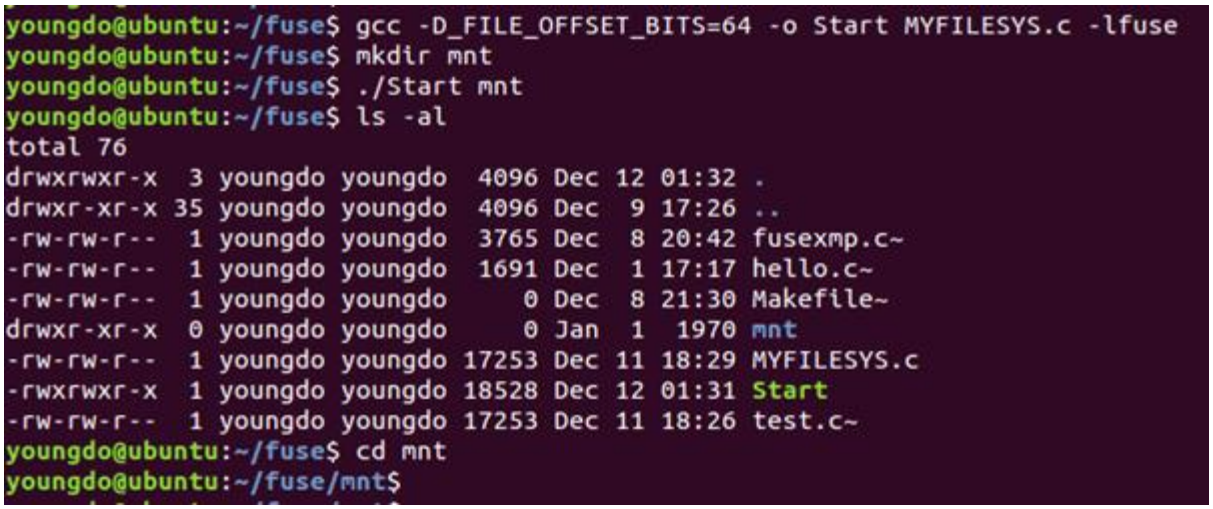
```
static struct fuse_operations MYFILESYS_OPER =
{
        .getattr = MYFILESYS_getattr,
        .readdir = MYFILESYS_readdir,
        .mkdir = MYFILESYS_mkdir,
        .rmdir = MYFILESYS_rmdir,
        .mknod = MYFILESYS_mknod,
        .read = MYFILESYS_read,
```

```
        .utimens = MYFILESYS_utimens,
        .open = MYFILESYS_open,
        .write = MYFILESYS_write,
        .unlink = MYFILESYS_unlink,
        .chmod = MYFILESYS_chmod,
        .chown = MYFILESYS_chown,
        .rename = MYFILESYS_rename,
};
```

getattr 와 readdir, mkdir, rmdir, mknod, read 그리고 open, write, unlink 들 같이 기존의 hello.c 예제와 쉽게 접근할 수 있는 기능들 이외에 chmod 와 chown 함수를 통해 파일의 권한을 변경하고, rename 함수를 통해 이름을 재 설정하는 기능을 추가로 구현 하였습니다. 또한 utimes 함수를 통해 파일이 변경, 접근, 수정되었던 시간을 알 수 있도록 구현하였습니다.

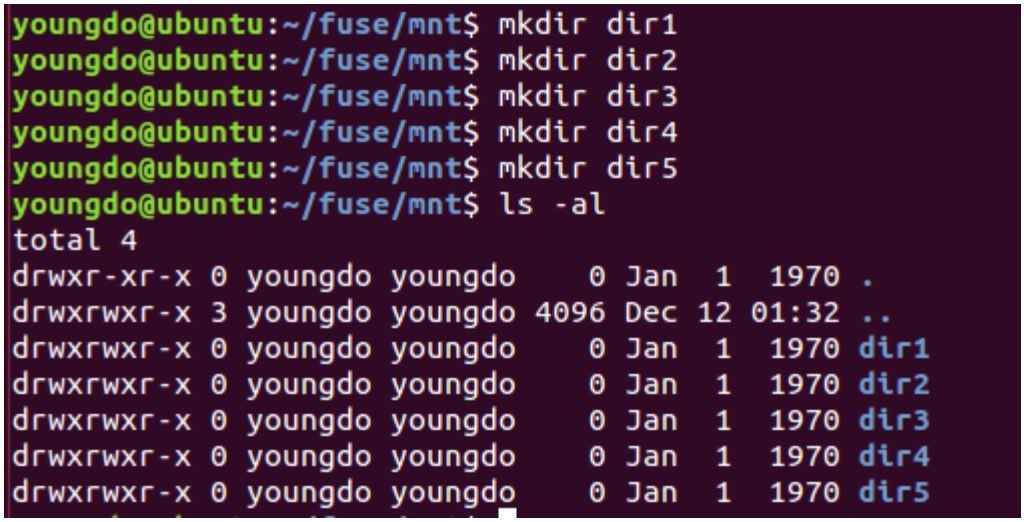
3. 실행 결과 화면 설명

3.1. Mount 과정



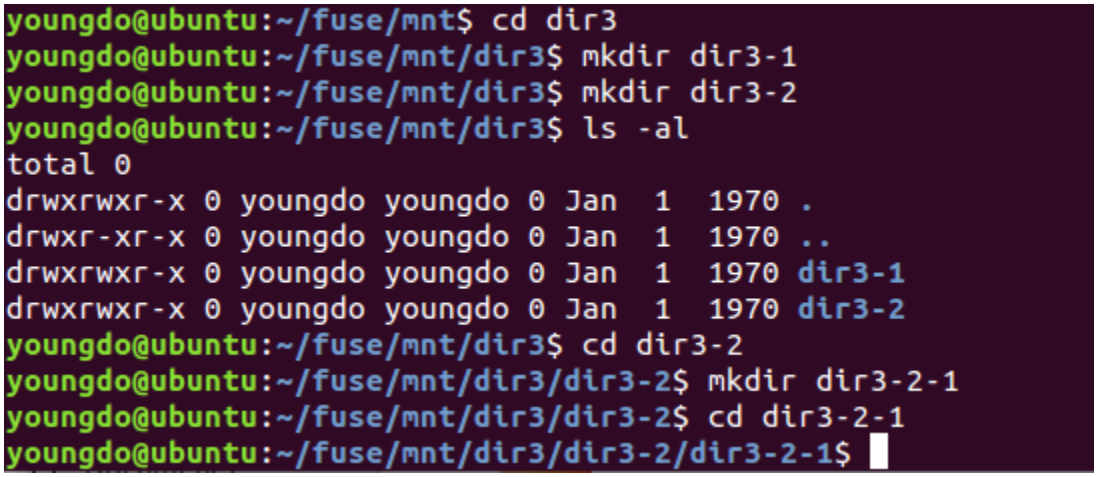
위의 그림과 같이 MYFILESYS_kwon_cho.c 파일을 컴파일 후 mnt 이름을 가진 디렉토리를 생성, 그곳에 mount 시키는 것을 볼 수 있습니다. 또한 cd 명령어를 통해 해당 디렉토리로 이동하였습니다.

3.2. 다중 및 다단계 디렉토리 생성



위의 그림은 하나의 mnt 디렉토리 안에 이름이 dir1 ~ dir5 인 5 개의 디렉토리를 mkdir 명령어를 통해 생성한 것을 디렉토리 속성 확인하는 ls 명령어를 통해 확인할 수

있습니다. 이를 통해 직접 구현한 두 함수 MYFILESYS_readdir()와 MYFILESYS_mkdir()가 정상적으로 작동함을 알 수 있습니다.



위 그림은 mkdir 명령어를 통해 다단계 디렉토리를 구성할 수 있음과 cd 명령어를 통해 각 디렉토리를 이동할 수 있음을 보여주며, 문제없다는 것을 확인할 수 있습니다.

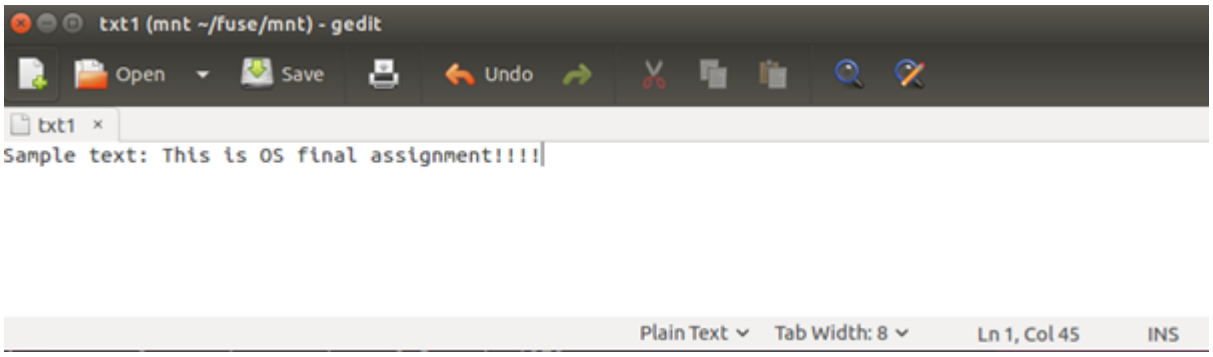
위 그림은 cd 명령어를 호출했을 때, 작성한 함수에서 존재하지 않는 경로에 대한 예외처리가 되고 있다는 것을 보여줍니다.

3.3. 파일 생성

```
youngdo@ubuntu:~/fuse/mnt$ touch txt1
youngdo@ubuntu:~/fuse/mnt$ touch txt2
youngdo@ubuntu:~/fuse/mnt$ touch txt3
youngdo@ubuntu:~/fuse/mnt$ ls -al
total 4
drwxr-xr-x 0 youngdo youngdo    0 Jan  1  1970 .
drwxrwxr-x 3 youngdo youngdo 4096 Dec 12 03:25 ..
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir1
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir2
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir3
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir4
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir5
-rw-rw-r-- 0 youngdo youngdo    0 Dec 12 03:30 txt1
-rw-rw-r-- 0 youngdo youngdo    0 Dec 12 03:30 txt2
-rw-rw-r-- 0 youngdo youngdo    0 Dec 12 03:30 txt3
```

위 그림은 mnt 디렉토리에서 이름이 txt1 ~ txt3 인 세 개의 비어있는 파일을 touch 명령어를 통해 생성된다는 것과 그 파일의 속성을 ls 명령어를 통해 보여줍니다. 이 과정에서 구현한 함수 MYFILESYS_mknod()와 MYFILESYS_getattr()가 정상적으로 작동함을 알 수 있습니다.

3.4. gedit 을 통한 파일 열기, 닫기, 수정



위의 그림은 위의 과정 3 에서 생성한 비어있던 파일에 내용을 추가하여 저장했다는 것을 볼 수 있습니다.

```
youngdo@ubuntu:~/fuse/mnt$ cat txt1
Sample text: This is OS final assignment!!!!
youngdo@ubuntu:~/fuse/mnt$ ls -al
total 4
drwxr-xr-x 0 youngdo youngdo    0 Jan  1  1970 .
drwxrwxr-x 3 youngdo youngdo 4096 Dec 12 03:25 ..
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir1
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir2
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir3
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir4
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir5
-rw-rw-r-- 0 youngdo youngdo   45 Jan  1  1970 txt1
-rw-rw-r-- 0 youngdo youngdo    0 Dec 12 03:30 txt1~
-rw-rw-r-- 0 youngdo youngdo    0 Dec 12 03:30 txt2
-rw-rw-r-- 0 youngdo youngdo    0 Dec 12 03:30 txt3
youngdo@ubuntu:~/fuse/mnt$
```

위의 그림에서와 같이 gedit 으로 저장한 것이 실제 정상적으로 저장되었음을 cat 명령어와 ls 명령어를 사용해 확인할 수 있습니다. 또한, 이 과정에서 작성한 함수 MYFILESYS_open(), MYFILESYS_write(), MYFILESYS_read()가 정상적으로 작동함을 알 수 있습니다.

3.5. 파일 및 디렉토리 이름 변경

```
youngdo@ubuntu:~/fuse/mnt$ mv dir1 directory1
youngdo@ubuntu:~/fuse/mnt$ mv txt2 sample2
youngdo@ubuntu:~/fuse/mnt$ ls -al
total 4
drwxr-xr-x 0 youngdo youngdo    0 Jan  1  1970 .
drwxrwxr-x 3 youngdo youngdo 4096 Dec 12 03:25 ..
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir2
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir3
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir4
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir5
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 directory1
-rw-rw-r-- 0 youngdo youngdo    0 Dec 12 03:30 sample2
-rw-rw-r-- 0 youngdo youngdo   45 Jan  1  1970 txt1
-rw-rw-r-- 0 youngdo youngdo    0 Dec 12 03:30 txt1~
-rw-rw-r-- 0 youngdo youngdo    0 Dec 12 03:30 txt3
youngdo@ubuntu:~/fuse/mnt$
```

위의 그림은 앞서 생성한 각 파일과 디렉토리의 이름을 mv 명령어를 통해 변경되었음을 나타냅니다. 이 과정에서 작성한 함수 MYFILESYS_rename()이 정상적으로 작동함을 알 수 있습니다.

3.6. 권한 변경 및 소유자 변경

```
youngdo@ubuntu:~/fuse/mnt$ chmod 700 txt1
youngdo@ubuntu:~/fuse/mnt$ ls -al
total 4
drwxr-xr-x 0 youngdo youngdo    0 Jan  1  1970 .
drwxrwxr-x 3 youngdo youngdo 4096 Dec 12 03:25 ..
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir2
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir3
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir4
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir5
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 directory1
-rw-rw-r-- 0 youngdo youngdo    0 Dec 12 03:30 sample2
-rwx------ 0 youngdo youngdo   45 Jan  1  1970 txt1
-rw-rw-r-- 0 youngdo youngdo    0 Dec 12 03:30 txt1~
-rw-rw-r-- 0 youngdo youngdo    0 Dec 12 03:30 txt3
```

위의 그림은 chmod 명령어를 통해 파일 속성에서 권한이 소유자에게만 있도록 권한 변경함을 볼 수 있습니다. 이를 통해 함수가 MYFILESYS_chmod() 원하는 결과를 나타낸다는 것을 알 수 있습니다.

```
youngdo@ubuntu:~/fuse/mnt$ chown root:staff txt1
youngdo@ubuntu:~/fuse/mnt$ ls -al
total 4
drwxr-xr-x 0 youngdo youngdo    0 Jan  1  1970 .
drwxrwxr-x 3 youngdo youngdo 4096 Dec 12 03:25 ..
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir2
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir3
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir4
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir5
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 directory1
-rw-rw-r-- 0 youngdo youngdo    0 Dec 12 03:30 sample2
-rwx------ 0 root    staff     45 Jan  1  1970 txt1
-rw-rw-r-- 0 youngdo youngdo    0 Dec 12 03:30 txt1~
-rw-rw-r-- 0 youngdo youngdo    0 Dec 12 03:30 txt3
```

위의 그림은 chown 명령어를 통해 파일의 원하는 소유자와 그룹으로 변경이 가능하다는 것을 알 수 있고, 구현한 MYFILESYS_chown() 함수가 원하는 결과를 나타내고 있다는 것을 알 수 있습니다.

3.7. 파일 및 디렉토리 삭제

```
youngdo@ubuntu:~/fuse/mnt$ rm *
rm: cannot remove ‘dir2’: Is a directory
rm: cannot remove ‘dir3’: Is a directory
rm: cannot remove ‘dir4’: Is a directory
rm: cannot remove ‘dir5’: Is a directory
rm: cannot remove ‘directory1’: Is a directory
youngdo@ubuntu:~/fuse/mnt$ ls -al
total 4
drwxr-xr-x 0 youngdo youngdo    0 Jan  1  1970 .
drwxrwxr-x 3 youngdo youngdo 4096 Dec 12 03:51 ..
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir2
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir3
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir4
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir5
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 directory1
youngdo@ubuntu:~/fuse/mnt$
```

위의 그림은 rm 명령어를 통해 해당 디렉토리의 파일들을 삭제할 수 있다는 것을 보여줍니다. 이 결과는 구현한 MYFILESYS_unlink() 함수가 정상 작동함을 알 수 있습니다.

```
youngdo@ubuntu:~/fuse/mnt$ rmdir dir2
youngdo@ubuntu:~/fuse/mnt$ ls -al
total 4
drwxr-xr-x 0 youngdo youngdo    0 Jan  1  1970 .
drwxrwxr-x 3 youngdo youngdo 4096 Dec 12 03:51 ..
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir3
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir4
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir5
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 directory1
youngdo@ubuntu:~/fuse/mnt$ rmdir *
rmdir: failed to remove ‘dir4’: Directory not empty
youngdo@ubuntu:~/fuse/mnt$ ls -al
total 4
drwxr-xr-x 0 youngdo youngdo    0 Jan  1  1970 .
drwxrwxr-x 3 youngdo youngdo 4096 Dec 12 03:51 ..
drwxrwxr-x 0 youngdo youngdo    0 Jan  1  1970 dir4
youngdo@ubuntu:~/fuse/mnt$
```

위의 그림은 rmdir 명령어를 통해 해당 mnt 디렉토리에서 존재하는 하부 디렉토리를 삭제하는 것을 볼 수 있습니다. 여기서 디렉토리 안에 하위 폴더 또는 디렉토리가 존재하는 경우 해당 디렉토리만 삭제되지 않음을 볼 수 있습니다. 또한, 이 결과는 구현한 함수 MYFILESYS_rmdir()가 정상 작동함을 알 수 있습니다.

4. 참고자료 / 문헌

4.1. FUSE 이해:

1) Develop your own filesystem with FUSE
(https://www.ibm.com/developerworks/library/l-fuse/)
2) [Fuse] FUSE 의 기본 작성법
(http://m.blog.naver.com/xogml_blog/130141202822)
3) CS135 FUSE Documentation
(https://www.cs.hmc.edu/~geoff/classes/hmc.cs135.201001/homework/fuse/fuse_doc.html)

4) fuse_file_info Struct Reference
(https://fossies.org/dox/fuse-2.9.7/structfuse__file__info.html)
5) hello.c File Reference
(https://lastlog.de/misc/fuse-doc/doc/html/hello_8c.html)
6) Class Fuse.Operations
(http://pike.lysator.liu.se/generated/manual/modref/ex/predef_3A_3A/Fuse/Operations.html)

4.2. 파일권한:

Linux 에서 chmod 을 이용한 디렉토리/파일 권한 설정/변경
(http://oooobang.tistory.com/18)

5. 소스코드

[MYFILESYS_kwon_cho.c 로 첨부하였습니다.]