2017

Operating System Lab 3

Embedded System Lab.

LAB 3 – FUSE FILE SYSTEM

JUHYUNG SON, CHOI JONG MOO

[Lab 3 File System]

운영체제 수업을 통해 File System 과, 주요 자료 구조인 super block, inode, directory entry 및, 초기 UNIX File System 에 대해 숙지하였다. 이를 바탕으로 본 과제에서는 FUSE 라는 user level file system interface 를 활용해 간단한 UNIX File System 을 구현해본다. 아래 실습 과제 위치와 설명을 적어두었다, 두번째 실습 관련 직접 수정 및 구현해야 하는 파일은 아래 빨간색으로 표기하였다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls
Makefile  bmap.c  dir.c  example  file.c  fuse_main.c  include  inode.c  lab3_mkfs  mnt  mount.c
```

✓ 과제 구성

- Makefile
  - 실습 자료들을 컴파일 하기 위한 파일
- include/lab2_fs_types.h
  - 실습에 사용할 구조체 및 구현할 함수에 대한 헤더 파일.
- **bmap.c**
  - **inode bitmap, data bitmap 관련 함수 관련 파일.**
- **file.c**
  - **write, read, create, delete 등 file 연산 관련 파일**
- **dir.c**
  - **makedir ,rmdir, rename 등 directory 연산 관련 파일**
- **inode.c**
  - **inode table, inode 관련 연산 파일**
- fuse_main.c
  - lab3_mount 실행파일을 생성하기 위한 소스코드. FUSE 관련 함수 호출되는 부분.
- mount.c
  - format 한 ramdisk 를 mount 하기 위한 파일.
- lab3_mkfs
  - ramdisk 를 format 하기 위한 실행 파일.
  -

A. UNIX File System

UNIX operating system 이 처음 출시 되었을 때, UNIX 개발자인 Ken Tompson 이 개발한 첫 파일 시스템인 UNIX file system 으로 파일시스템의 meta data 에 대한 정보를 가진 inode 개념 data block 및 inode 의 할당 정보를 기록하는 bitmap 과 multi-level indexing 구조를 가지고 있다. 본 실습에서는 이러한 UNIX File System 을 구현 할 것이므로 실습 전 간단히 되돌아 보도록 한다.

1. ondisk layout

i. super block

파일시스템의 크기, 각 영역의 시작 주소, free inode 의 개수, free block 의 개수 등의 파일시스템을 운용하기위한 기본 정보를 가지고 있다.

ii. inode bitmap

현재 inode 의 할당 정보를 나타내며 1bit 로 inode 의 할당 정보를 나타낸다. 할당이 되어 있으면 해당 inode 번호의 bit 가 1 로 setting 되며, 할당이되지 않았거나 삭제된 상태이면 0 으로 setting 된다. 아래와 같은 경우, 파일시스템을 포멧 한 상태의 inode bitmap 모습이다.

```
root@fslecture-VirtualBox:/home/fs-lecture/Simple_FS# xxd -g 4 -l 128 -s+0x1000 /dev/rxd0
0001000: 07000000 00000000 00000000 00000000  ................
0001010: 00000000 00000000 00000000 00000000  ................
0001020: 00000000 00000000 00000000 00000000  ................
0001030: 00000000 00000000 00000000 00000000  ................
0001040: 00000000 00000000 00000000 00000000  ................
0001050: 00000000 00000000 00000000 00000000  ................
0001060: 00000000 00000000 00000000 00000000  ................
0001070: 00000000 00000000 00000000 00000000  ................
```

위 그림에서 하나의 숫자는 16 진수로 표현되어 있으므로 4bit 를 나타낸다. 따라서 16 진수인 00 00 00 07 을 2 진수로 나타내면 아래와 같다.

0000 0000 0000 0000 0000 0000 0000 0111

따라서 오른쪽부터 0 번, 1 번, 2 번 inode 가 할당되어 있는 상태임을 알 수 있다.

아래 그림은 해당 파일시스템에서 몇가지 directory 및 파일을 생성, 삭제 한 후의 결과이다. 아래 그림에서 16 진수인 00 01 a0 07 을 2 진수로 나타내면 아래와 같다.

0000 0000 0000 0001 1010 0000 0000 0111

0, 1, 2, 13, 15, 16 번 inode 가 할당되어 있는 상태임을 알 수 있다.

```
root@fslecture-VirtualBox:/home/fs-lecture/Simple_FS/mnt# xxd -g 4 -l 128 -s+0x1000 /dev/rxd0
0001000: 07a00100 00000000 00000000 00000000  ................
0001010: 00000000 00000000 00000000 00000000  ................
0001020: 00000000 00000000 00000000 00000000  ................
0001030: 00000000 00000000 00000000 00000000  ................
0001040: 00000000 00000000 00000000 00000000  ................
0001050: 00000000 00000000 00000000 00000000  ................
0001060: 00000000 00000000 00000000 00000000  ................
0001070: 00000000 00000000 00000000 00000000  ................
```

iii. data bitmap

data block 에 대한 할당 정보를 bitmap 으로 나타낸다. 할당 정보를 나타내는 방식은 inode bitmap 과 동일하며, data block 의 위치를 계산하는 방법은 아래와 같다. data 영역의 시작부분에 data block 의 크기를 곱하여 구할 수 있다.

iv. data area

root directory 부터 directory entry, data 등이 기록되는 영역이다.

2. 주요 자료구조

/home/fs-lecture/DKU_OS_Lab/lab3_fuse/include/lab4_fs_types.h 위치에 주요 자료구조의 예시가 있으며 이를 구현에 그대로 사용하거나 조작한다.

i. super block

```
108 /*
109  * superblock structure
110  */
111 struct lab3_super_block{
112     u32 sb_signature;   /* file system signature*/
113     u32 sb_student_id;  /* your student id  */
114
115     s64 sb_sector_size; /* sector size */
116     s64 sb_block_size_shift; /* block size shift bit */
117     s64 sb_no_of_sectors;    /* number of sectors in ramdirk  */
118     s64 sb_no_of_used_blocks; /* used block count  */
119
120     u32 sb_root_ino; /* reserved inode count  */
121     u32 sb_reserved_ino; /* inode table max inode count  */
122
123     s32 sb_max_dir_num; /* msx directoy in block      */
124     s32 sb_max_inode_num; /* max inode count in block */
125
126     u32 sb_reserved;    /* reserved area count in cluster size */
127     u32 sb_sb_start;    /* super block start address in cluster size */
128     u32 sb_sb_size;     /* super block area size in cluster size */
129     u32 sb_ibitmap_start; /* inode bitmap start address in cluster size */
130     u32 sb_ibitmap_size; /* inode bitmap size in blocknr in cluster size */
131     u32 sb_dbitmap_start; /* data bitmap start address in cluster size */
132     u32 sb_dbitmap_size; /* data bitmap area size in cluster size */
133     u32 sb_itable_start; /* inode table start address in cluster size*/
134     u32 sb_itable_size; /* inode table size in cluster size */
135     u32 sb_darea_start; /* data area start address in cluster size*/
136     u32 sb_darea_size; /* data area size in cluster size */
137
138     s32 sb_free_inodes; /* free inode count  */
139     s64 sb_free_blocks; /* free block count  */
140
141     u32 sb_last_allocated_ino;/* last allocated inode number  */
142     u32 sb_last_allocated_blknr; /* last allocated blknr  */
143 };
```

ii. inode

```
172 struct lab3_inode{
173     inode_t i_ino;       /* inode number  */
174     u32 i_type;          /* file type  */
175     s64 i_size;          /* file size  */
176     u32 i_deleted;       /* is fil edeleted ?  */
177     u32 i_links_count;   /* Links count */
178     u32 i_ptr;           /* for directory entry */
179     u32 i_atime;         /* Access time */
180     u32 i_ctime;         /* Inode change time */
181     u32 i_mtime;         /* Modification time */
182     u32 i_dtime;         /* Deletion Time */
183     u16 i_gid;           /* Low 16 bits of Group Id */
184     u16 i_uid;           /* Low 16 bits of Owner Uid */
185     u16 i_mode;          /* File mode */
186     u32 resv1[1];        /* null paddings to make inode size to 128 byte size */
187     u32 i_blocks[TINDIRECT_BLOCKS + 1];
188     u32 resv2[2];        /* null padding to make inode size to 128 byte size */
189 };
```

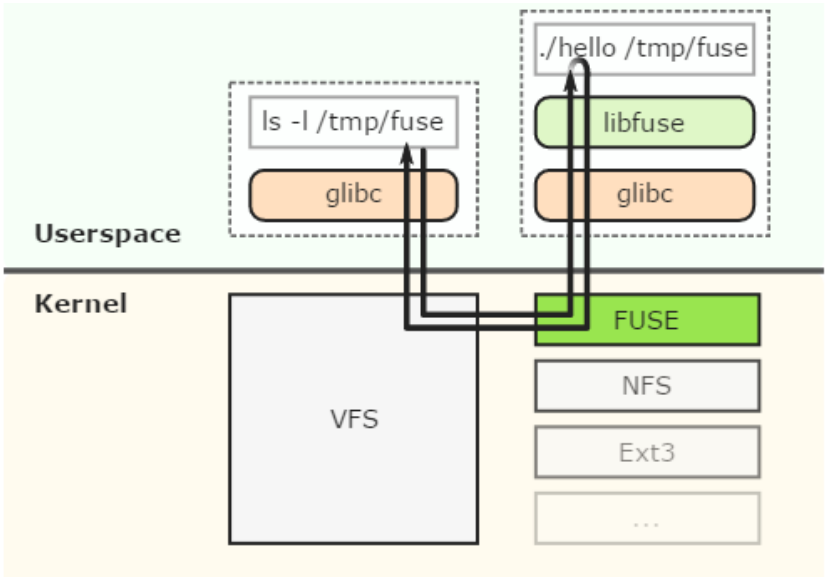
iii. directory entry

```
145 /*
146  * Directory entry structure
147  */
148 struct lab3_dir_entry{
149     inode_t d_ino; /* inode number  */
150     u32 d_flag; /*directory flag (e.g. LAB2_DIR_USDED , LAB2_DIR_DELETED ..)*/
151     s8  d_filename[FNAME_SIZE]; /* file nanme  */
152     u32 resv; /* null padding to make dir_entry to 128 byte size  */
153 };
```

B. FUSE File System

Kernel 영역에서 File System 을 개발 하기란 매우 힘든 작업이다. 때문에 이를 보완하기 위한 FUSE 라는 User Level File System 이 있다. 본 실습에서는 FUSE 를 사용하여 자신만의 파일시스템을 구현하며 실습 전 FUSE 에 대해 간단히 살펴본다.

1. FUSE 파일 시스템이란?



다양한 스토리지 장치의 속도를 높이기 위한 많은 파일시스템들이 존재하지만 파일시스템은 커널에서 구현되어 있기 때문에 디버깅하기 어려움, 커널의 복잡한 자료구조들과의 연관성 등의 이유로 구현하기 매우 힘들다. FUSE 는 개발자들이 Kernel Level 이 아닌, User Level 에서 다양한 언어로 자신의 파일시스템을 개발하기 위한 API 를 제공하고, Unix-like operating system 의 loadable kernel module 를 통해 일반 파일시스템 연산 시, 연산의 핵심 동작 부분의 구현을 User Level 에서 Kernel code 의 수정 없이 할 수 있도록 하게 해 줌으로써 Kernel, low-level operation 에 대한 깊은 이해가 없이도 file system 을 만들 수 있게 해준다.

2. FUSE 파일 시스템의 장단점

i. 장점

- clean and easy interface
- no need to advanced kernel development skill
- comes user isolation, more secure

ii. 단점

- libfuse need to be installed
- slower than low-level implementation
- Not the best option if you need multiple users to access your FS

3. FUSE file system 의 예시

FUSE 파일 시스템은 개발의 용이성 때문에 아래와 같이 다양한 파일시스템에서 사용이 되고 있다.

- LoggedFS : filesystem that logs operations that happens in it.
- GlusterFS : scalable network file system.
- SSHFS : allows mounting a remote filesystem over SSH.
- GMailFS : allows to user Gmail storage as a file system.

과제 관련 FUSE 파일 시스템 개발 전 이해를 돕기 위해 아래와 같이 FUSE 파일 시스템의 예시를 아래 경로에 첨부하였다.

```
root@fslecture-VirtualBox:/home/fs-lecture/DKU_OS_LAB/lab3_fuse/example# ls -al
합계 20
drwxr-xr-x 3 root root 4096  3월  7 06:44 .
drwxr-xr-x 5 root root 4096  3월  7 06:36 ..
-rw-r--r-- 1 root root  730  3월  7 03:12 Makefile
-rw-r--r-- 1 root root 3546  3월  7 06:44 lab3_example.c
drwxr-xr-x 2 root root 4096  3월  7 03:12 mnt
```

예시 자료는 아래와 같이 컴파일 및 실행할 수 있다.

```
root@fslecture-VirtualBox:/home/fs-lecture/DKU_OS_LAB/lab3_fuse/example# make
Compiling lab3_example.c ...
gcc -c -D_GNU_SOURCE -g -W -D_FILE_OFFSET_BITS=64 -DFUSE_USE_VERSION=30 -I/usr/include/fuse
o lab3_example.o lab3_example.c
gcc -o lab3_example lab3_example.o -lfuse
root@fslecture-VirtualBox:/home/fs-lecture/DKU_OS_LAB/lab3_fuse/example# ./lab3_example mnt/
root@fslecture-VirtualBox:/home/fs-lecture/DKU_OS_LAB/lab3_fuse/example# ls -al mnt/
합계 4
drwxr-xr-x 2 root root    0  3월  7 06:48 .
drwxr-xr-x 3 root root 4096  3월  7 06:48 ..
-rw-r--r-- 1 root root 1024  3월  7 06:48 file349
-rw-r--r-- 1 root root 1024  3월  7 06:48 file54
root@fslecture-VirtualBox:/home/fs-lecture/DKU_OS_LAB/lab3_fuse/example# cat mnt/file349
Hello World From File 349!
root@fslecture-VirtualBox:/home/fs-lecture/DKU_OS_LAB/lab3_fuse/example# cat mnt/file54
Hello World From File54!
```

관련 자료 설명.

http://www.maastaar.net/fuse/linux/filesystem/c/2016/05/21/writing-a-simple-filesystem-using-fuse/

4. FUSE file interface

i. FUSE file handles

```
/**
 * Information about open files
 */
struct fuse_file_info {
    /** Open flags.  Available in open() and release() */
    int flags;

    /** In case of a write operation indicates if this was caused by a
        writepage */
    unsigned int writepage : 1;

    /** Can be filled in by open, to use direct I/O on this file. */
    unsigned int direct_io : 1;

    /** Can be filled in by open, to indicate that currently
        cached file data (that the filesystem provided the last
        time the file was open) need not be invalidated. */
    unsigned int keep_cache : 1;

    /** Indicates a flush operation.  Set in flush operation, also
        maybe set in highlevel lock operation and lowlevel release
        operation. */
    unsigned int flush : 1;

    /** Can be filled in by open, to indicate that the file is not
        seekable. */
    unsigned int nonseekable : 1;

    /* Indicates that flock locks for this file should be
       released.  If set, lock_owner shall contain a valid value.
       May only be set in ->release(). */
    unsigned int flock_release : 1;

    /** Padding.  Do not use*/
    unsigned int padding : 27;

    /** File handle.  May be filled in by filesystem in open().
        Available in all other file operations */
    uint64_t fh;

    /** Lock owner id.  Available in locking operations and flush */
    uint64_t lock_owner;

    /** Requested poll events.  Available in ->poll.  Only set on kernels
        which support it.  If unsupported, this field is set to zero. */
    uint32_t poll_events;
};
```

FUSE 함수들은 parameter 로, path 를 받아 해당 파일을 찾고 관련 연산을 수행할 수 있도록 한다. 하지만, file 에 대한 read, write 의 경우 각 file descriptor 및 어디까지 read, write 하였는지등의 자료를 해당 file descriptor 가 열려있는 동안 유지 할 수 있다면 구현이 더 편할 것이다. 이를 위해 fuse 는 open 된 파일에 대해 fuse_file_info 라는 자료구조를 open, create, read, write, release, opendir, readdir, releasedir 등의 연산에 parameter 로 제공하여 구현을 좀더 쉽게 해준다. 이중, fuse_file_info 에서 fh 라는 변수를 사용자 정의 값으로 조작 가능하게 한다. 본 과제에서는 open, create 시, file descriptor, handler 등의 정보를 전달하고, read, write 등의 연산 시에 이를 활용하며, release, releasedir 등의 연산 시에 file descriptor 를 닫는 등의 연산에 활용 할 수 있다.

ii. FUSE context

```
struct fuse_context {
    /** Pointer to the fuse object */
    struct fuse *fuse;

    /** User ID of the calling process */
    uid_t uid;

    /** Group ID of the calling process */
    gid_t gid;

    /** Thread ID of the calling process */
    pid_t pid;

    /** Private filesystem data */
    void *private_data;

    /** Umask of the calling process */
    mode_t umask;
};
```

FUSE 는 또한 각 FUSE 관련 연산 시, 공통적으로 사용할 수 있는 context 를 위와 같이 제공하여 어떤 함수에서도 fuse_get_context 를 통해 context 를 불러와 구현에 사용할 수 있도록 편의성을 제공한다. 본 과제에서는 이를 ramdisk 를 open 한 file descriptor 를 fuse_context->private_data 에 저장하는 등으로 사용할 수 있다. fuse_context 는 파일시스템 mount 시에 처음 호출되는 init 함수에서 설정할 수 있으며, 파일시스템이 umount 될 때 호출되는 release 함수에서 free 할 수 있다.

iii. FUSE functions

```
static struct fuse_operations prefix_oper = {
    .init        = prefix_init,
    .destroy     = prefix_destroy,
    .getattr     = prefix_getattr,
    .fgetattr    = prefix_fgetattr,
    .access      = prefix_access,
    .readlink    = prefix_readlink,
    .readdir     = prefix_readdir,
    .mknod       = prefix_mknod,
    .mkdir       = prefix_mkdir,
    .symlink     = prefix_symlink,
    .unlink      = prefix_unlink,
    .rmdir       = prefix_rmdir,
    .rename      = prefix_rename,
    .link        = prefix_link,
    .chmod       = prefix_chmod,
    .chown       = prefix_chown,
    .truncate    = prefix_truncate,
    .ftruncate   = prefix_ftruncate,
    .utimens     = prefix_utimens,
    .create      = prefix_create,
    .open        = prefix_open,
    .read        = prefix_read,
    .write       = prefix_write,
    .statfs      = prefix_statfs,
    .release     = prefix_release,
    .opendir     = prefix_opendir,
    .releasedir  = prefix_releasedir,
    .fsync       = prefix_fsync,
    .flush       = prefix_flush,
    .fsyncdir    = prefix_fsyncdir,
    .lock        = prefix_lock,
    .bmap        = prefix_bmap,
    .ioctl       = prefix_ioctl,
    .poll        = prefix_poll,
#ifdef HAVE_SETXATTR
    .setxattr    = prefix_setxattr,
    .getxattr    = prefix_getxattr,
    .listxattr   = prefix_listxattr,
    .removexattr = prefix_removexattr,
#endif
    .flag_nullpath_ok = 0,                /* See below */
};
```

FUSE 는 사용자의 구현을 위해 위와 같이 다양한 operation을 제공한다. 파일시스템을 만들 때 위 그림의 모든 연산이 필수적인 것은 아니며 본 과제에서는 flush, lock 등 ioctl 등 많은 기능을 배제하고 파일 시스템에 필수적인 연산 만을 구현하도록 한다. 구현해야 하는 최소의 필수적인 연산들은 아래와 같다. 아래 연산들에서 required 라고 명시된 함수는

파일시스템 동작을 위해 반드시 구현해야 하는 함수이며, optional 이라고 명시된 함수는 필수적으로 구현하지 않아도 되는 함수이다.

```
static struct fuse_operations lab3_oper = {
    .init       = lab3_init,        // optional
    .getattr    = lab3_getattr,     // required
    .access     = lab3_access,      // optional
    .readdir    = lab3_readdir,     // optional
    .mkdir      = lab3_mkdir,       // required
    .unlink     = lab3_unlink,      // required
    .rmdir      = lab3_rmdir,       // requried
    .rename     = lab3_rename,      // required
    .truncate   = lab3_truncate,    // optional
    .open       = lab3_open,        // required
    .read       = lab3_read,        // required
    .write      = lab3_write,       // required
    .release    = lab3_release,     // optional
    .releasedir = lab3_releasedir,  // optional
    .destroy    = lab3_destroy,     // optional
    .opendir    = lab3_opendir,     // optional
    .create     = lab3_create,      // required
    .utimens     = lab3_utimens     // required
};
```

- **init 함수**

```
void *lab3_init(struct fuse_conn_info *conn)
{
    // You need to Implement here.
}
```

파일시스템이 mount 될 때 맨 처음 호출되는 함수로, 함수 내에서 return하는 값을 FUSE 내부적으로 return 하는 값을 fuse_context 의 private 에 저장해 추후 다른 연산에서 fuse_get_context 함수를 통해 init 함수에서 설정한 값을 활용할 수 있도록 해준다.

- **getattr 함수**

```
static int lab3_getattr(const char *path, struct stat *stbuf)
{
    // You need to Implement here.
}
```

file의 metadata 관련 정보들. 즉 파일속성값들을 반환하며. 매개변수로 주어진 struct stat에 관련 metadata들을 채워 주어야 한다. 파일시스템에서 stat 연산 시 호출되는 함수이며, ls, mkdir 등 다른 모든 연산에도 기본적으로 해당 파일의 metadata를 가져오기 위해 호출하는 함수이다.

- 참고 자료(stat 관련)

http://man7.org/linux/man-pages/man2/stat.2.html

https://en.wikipedia.org/wiki/Stat_(system_call)

- **readdir 함수**

```
static int lab3_readdir(const char *path, void *buf, fuse_fill_dir_t filler,
        off_t offset, struct fuse_file_info *fi)
{
    // You need to Implement here.
}
```

디렉터리의 내용을 읽고 그 내용을 반환하는 함수이며, FUSE는 손쉬운 구현을 위해 아래와 같은 filler 함수를 parameter로 제공하여 directory 로부터 읽어 들인 directory내에 존재하는 파일 이름을 name에, 그 파일의 metadata를 stat에 채워 filler 함수를 호출함으로써 읽어 들인 내용을 반환할 수 있다.

```
/** Function to add an entry in a readdir() operation
 *
 * @param buf the buffer passed to the readdir() operation
 * @param name the file name of the directory entry
 * @param stat file attributes, can be NULL
 * @param off offset of the next entry or zero
 * @param flags fill flags
 * @return 1 if buffer is full, zero otherwise
 */
typedef int (*fuse_fill_dir_t) (void *buf, const char *name,
                const struct stat *stbuf, off_t off,
                enum fuse_fill_dir_flags flags);
```

readdir 함수는 위의 filler함수의 offset parameter를 설정하는 방식에 따라 두가지 방식으로 구현할 수 있다. 첫번째 방법은 offset parameter를 무시하고 0으로 설정하여 한번의 readdir 함수 호출로 전체 directory를 읽어 들이는 방법이며, 두번째 방법은 offset에 directory에서 현재 읽어 들인 directory entry의 offset을 설정하여 directory의 directory entry를 읽어 들일 때마다 readdir함수가 호출되도록 하는 방법이다.

이중 첫번째 방법의 구현 알고리즘은 아래와 같이 구현하면 된다.

① parameter로 넘겨받은 path의 inode 를 읽어 들여 해당 directory가 저장된 block주소를 구한다.

② inode로부터 알아낸 block 주소에서 첫번째 directory entry를 읽어들여 struct stat를 생성해 st_ino, st_mode, st_nlink를 채우고, 읽어 들인 파일 이름과 함께 filler 함수로 반환하다.

③ 2번 과정을 directory의 끝까지 수행 한다.

fuse.h (https://github.com/libfuse/libfuse/blob/master/include/fuse.h)에 명시된 readdir 구현 방법은 아래와 같다.

```
/** Read directory
 *
 * The filesystem may choose between two modes of operation:
 *
 * 1) The readdir implementation ignores the offset parameter, and
 * passes zero to the filler function's offset.  The filler
 * function will not return '1' (unless an error happens), so the
 * whole directory is read in a single readdir operation.
 *
 * 2) The readdir implementation keeps track of the offsets of the
 * directory entries.  It uses the offset parameter and always
 * passes non-zero offset to the filler function.  When the buffer
 * is full (or an error happens) the filler function will return
 * '1'.
 */
int (*readdir) (const char *, void *, fuse_fill_dir_t, off_t,
                struct fuse_file_info *, enum fuse_readdir_flags);
```

- **mkdir 함수**

```
static int lab3_mkdir(const char *path, mode_t mode)
{
    // You need to Implement here.
}
```

path로 넘겨받은 위치에 directory를 mode의 권한으로 생성한다.

다. directory를 생성하게 되면 기본적으로 자신의 inode 번호 "." 라는 파일 이름을 가지는 directory entry와 상위 directory의 inode 번호 ".."라는 이름의 파일이름을 가지는 directory entry를 생성한다. path에 해당하는 이름이 이미 존재 한다면 –EEXIST 라는 error를 반환해야 한다. (linux error : http://www.virtsync.com/c-error-codes-include-errno 참조)

directory를 추가하려면 inode table에 inode추가, 부모 directory에 directory entry추가, inode bitmap, data bitmap 에 할당 표시(해당 bit를 1로 설정), directory block에 ".", ".."의 두가지 directory entry추가의 과정을 수행해야 한다.

- **rmdir 함수**

```
static int lab3_rmdir(const char *path)
{
    // You need to Implement here.
}
```

path 로 넘겨받은 위치에 해당하는 directory를 삭제한다. path 위치에 해당 파일이 존재 하지 않는다면, -ENOENT 라는 error를 반환해야 한다. (linux error : http://www.virtsync.com/c-error-codes-include-errno 참조)

directory를 삭제하려면 inode table에서 inode삭제, 부모 directory에 directory entry삭제, inode bitmap, data bitmap 에 해제 표시(해당 bit를 0으로 설정), directory 가 기록된 block의 삭제의 과정을 수행해야 한다.

- **open 함수**

```
static int lab3_open(const char *path, struct fuse_file_info *fi)
{
    // You need to Implement here.
}
```

path에 이미 존재하는 파일을 open하는 함수 fuse_file_info struct 의 fh 변수를 활용할 수 있다.

fh 에 현재까지 read/write 한 byte 의 offset 등의 정보를 가진 struct를 저장해 주면 쉽게 구현할 수 있다.

- **create 함수**

```
static int lab3_create (const char *path, mode_t mode, struct fuse_file_info *fi)
{
    // You need to Implement here.
}
```

path에 해당하는 파일을 mode 권한으로 생성하는 함수

파일을 생성하게 되면, 생성한 파일의 inode 를 indoe table에 추가, inode bitmap에 할당 정보 표시(해당 bit를 1으로 설정), 부모 directory 에 directory entry 추가 의 과정을 수행해야 한다.

- **unlink 함수**

```
static int lab3_unlink(const char *path)
{
    // You need to Implement here.
}
```

path에 해당하는 파일을 삭제하는 함수. 본 과제에서는 hard link, soft link를

고려하지 않기 때문에 단순히 파일을 삭제하는 기능만 구현하면 된다.

파일을 삭제하게 되면, 삭제할 파일의 inode를 indoe table에서 삭제,, inode bitmap에서 해당 inode번호 할당 해제 표시(해당 bit를 0으로 설정), 부모 directory 에서 directory entry 삭제, 파일의 data가 저장된 block삭제 의 과정을 수행해야 한다.

- **write 함수**

```
static int lab3_write(const char *path, const char *buf, size_t size,
         off_t offset, struct fuse_file_info *fi)
{
    // You need to Implement here.
}
```

path에 해당하는 파일에 buf의 내용을 size만큼 offset위치에 write하는 함수이다.

write를 수행하려면 path에 해당하는 inode를 읽어 들여 inode에 기록된 파일의 data 위치를 찾아가 buf의 정보를 추가해야 한다. 만약 data block을 더 할당해야 할 경우 data bitmap의 할당 정보를 추가해 주며 파일의 크기가 변경되었으므로 inode의 파일 크기 필드를 update해준다

- **read 함수**

```
static int lab3_read(const char *path, char *buf, size_t size, off_t offset,
         struct fuse_file_info *fi)
{
    // You need to Implement here.
}
```

path에 해당하는 파일의 내용을 offset위치로부터 size만큼 읽어 들여 buf에 저장하는 함수이다.

read를 수행하려면 path에 해당하는 inode를 읽어들여 inode에 기록된 파일의 data 위치를 찾아가 buf의 정보를 읽어 들여야 한다.

- **utimes 함수**

```
static int lab3_utimens(const char *path, const struct timespec ts[2])
{
    // You need to Implement here.
}
```

path에 해당하는 파일의 last accessed time, last modification time을 각각 fs[0], fs[1] 로 설정하는 함수이다.

파일에 대해 read, write등 수행 시 모두 파일에 대한 접근이 이루어 진 것이

므로 파일의 접근 시간 등을 수정해 주어야 한다. utimes를 수행하려면 path 에 해당하는 inode를 읽어 들여 inode에 기록된 파일의 last accessed time, last modified time을 update해준다

- **release 함수**

```
static int lab3_release(const char *path, struct fuse_file_info *fi)
{
    // You need to Implement here.
}
```

path에 해당하는 파일을 닫는 함수이다.

- **rename 함수**

```
static int lab3_rename(const char *from, const char *to)
{
    // You need to Implement here.
}
```

from에 해당하는 directory entry를 to로 바꾸어 주는 함수이다. from에 해당하는 파일이 존재하지 않을 경우, -ENOENT 라는 error를 반환해야 하며, to 위치에 이미 파일이 존재할 경우 –EEXIST 라는 error를 반환해야 한다. 또한 to 위치가 디렉터리가 아닌 파일로 설정된 경우 –ENODIR을 반환해야 한다. (linux error : http://www.virtsync.com/c-error-codes-include-errno 참조)

C. Lab3 Simple Linux File System based on FUSE

운영체제내에서 또는 다양한 disk 에서는 FAT, btrfs, xfs,f2fs 등 다양한 파일시스템이 사용되며, 이중 많이 사용되는 파일시스템들 중 하나인 ext 계열 파일시스템들은 UNIX 계열 파일시스템으로 muli-level indexing 등 초기 UNIX 파일시스템의 설계 디자인을 따른다. 본 과제에서는 가장 간단한 파일시스템인 초기 UNIX 파일시스템을 ramdisk 에서 구동 되도록 구현하여 파일시스템 내에서 data 및 metadata 를 찾아가는 원리를 실습해본다. (본 과제는 제한시간 내 과제 구현을 위해 현재 ext2, ext3, ext4 등 다른 ext 파일시스템들과 달리 block group 을 사용하지 않으며, xxd 를 통한 hex 값 확인 디버깅의 용이성을 위해 indoe table 의 크기를 4KB 로 잡았다.)

1. 기본 과제 구성 및 보너스 과제 구성

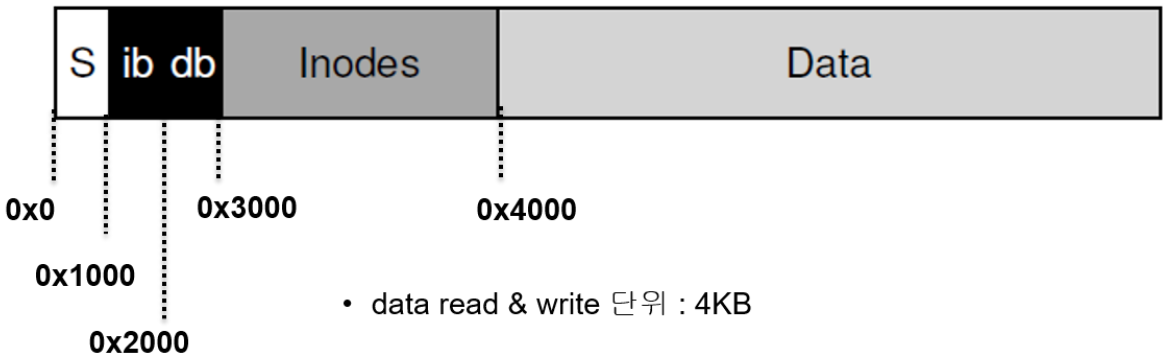
Lab2 FUSE File System 은 아래 그림과 같은 위치에 있으며 bmap.c, dir.c, file.c, inode.c, fuse_main.c 라는 파일의 함수들을 구현해야 한다. 함수 구현은 소스 파일 내에 정해진 형식대로 구현하지 않아도 되나 정해진 형식을 변경하게 되면 include/lab3_fs_types.h 의 함수 선언을 변경해 주어야 한다. 구현해야 하는 소스파일을 빨간색으로 표시하였다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls
Makefile  bmap.c  dir.c  example  file.c  fuse_main.c  include  inode.c  lab3_mkfs  mnt  mount.c
```

fuse_main.c 를 제외한 다른 구현해야 하는 파일들은 비어있는 상태이다. fuse_main.c 의 구현해야 할 함수에 사용될 것들 중 inode 관련된 것은 inode.c 에, file 관련된 것은 file.c 에, directory 관련된 것은 dir.c 에 bitmap 관련된 것은 bmap.c 에 추가하면 된다. 예를 들어 fuse_main.c 파일의 lab3_getatrr 함수에 lab3_read_inode 라는 이름으로 inode 를 읽는 함수를 정의하여 사용하고 싶다면, include/lab3_fs_types.h 에 해당 함수를 정의하고, inode.c 에 정의한 후, fuse_main.c 에서 사용하면 된다.

현재 파일시스템을 format 하는 부분은 실행파일로 제공하였고, super block, inode, directory entry 등의 기본적인 자료구조를 include/lab3_fs_types.h 에 제공하였다. 이를 자신이 직접 format 하는 프로그램 작성 및 자신이 정의한 자료구조를 사용하여 과제를 수행하는 것을 보너스 과제로 하도록 한다.

lab3_mkfs 라는 이름으로 제공한 format 파일은 아래 그림과 같이 format 을 구성하는 실행파일이다.



2. 과제 목표

- ✓ *FUSE 를 활용하여 실습자료에서 제공한 Ramdisk 를 기반으로 동작하는 간단한 UNIX file system 을 구현한다.*
- ✓ *제한시간 내 과제 구현을 위해 single thread 에서 동작하는 File System 을 작성한다.(race condition 등을 고려하지 않으며, FUSE 실행시 –s 옵션을 주어 single thread 에서 동작하도록 한다.)*
- ✓ *가능하다면 구현해도 좋지만, 제한시간 내 과제 구현을 위해 caching, buffering 등은 고려하지 않고 핵심 함수의 기능위주로 구현한다.*
- ✓ *제한시간 내 과제 구현을 위해 hard link, soft link, block device file, character device file 등은 고려하지 않으며, 위의 FUSE operation 에 명시한 연산만 구현한다.*

3. 실습 전 숙지 사항.

본 문서의 UNIX File System, FUSE 설명 및 아래 사항을 반드시 이해하고 구현한다.

- ✓ inode, file, directory 에 대한 이해

  http://pages.cs.wisc.edu/~remzi/OSTEP/file-intro.pdf
- ✓ "file, directory, 관련 기본 연산"

  http://pages.cs.wisc.edu/~remzi/OSTEP/file-implementation.pdf
- ✓ "UNIX file system 의 연산 구조"

  http://pages.cs.wisc.edu/~remzi/OSTEP/file-ffs.pdf

✓ FUSE interface, FUSE operation

https://github.com/libfuse/libfuse/blob/579c3b03f57856e369fd6db2226b77aba63b59ff/include/fuse.h#L102-L577

✓ “리눅스에서의 C Error

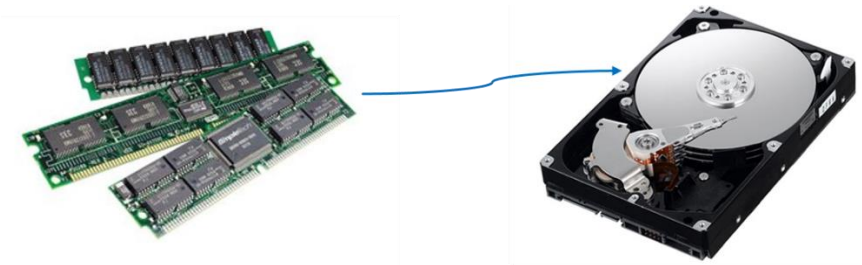
http://www.virtsync.com/c-error-codes-include-errno

inode.c, file.c, dir.c, bmap.c 파일에서 구현한 함수를 다른 소스파일에서 사용해야 할 경우, include/lab3_fs_types.h 에 함수를 선언하고 컴파일을 수행해야 한다.

리눅스 부팅 후 파일시스템 데이터, 메타데이터가 기록될 공간을 만들기 위해 아래와 같이 배포된 이미지의 ramdisk 를 활용하여 메모리의 일정 공간을 저장공간으로 만든다.

i. Ramdisk 란?

디스크가 아닌 RAM(DRAM)을 이용하여 디스크 드라이브를 구현하는 방식으로 “RapidDisk”라는 open source tool 을 이용해서 생성한 후, 이를 임의의 디렉토리에 mount 하여 사용한다. 본 실습은 실제 컴퓨터 상의 storage 를 사용하지 않고 ramdisk 를 생성하여 storage 로 이용한다.



ii. ramdisk 를 활용하여 스토리지 만들기

실습 전 아래 명령어를 통해 모듈을 적재 시킨다.

- # sudo modprobe rapiddisk
- # sudo modprobe rapiddisk_cache
- # sudo modprobe dm_crypt

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# modprobe rapiddisk
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# modprobe rapiddisk_cache
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# modprobe dm_crypt
```

모듈이 적재 되었는지 확인한다.

- # lsmod | grep rapidisk
- # lsmod | grep dm_crypt

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# lsmod | grep rapiddisk
rapiddisk_cache        20480  0
rapiddisk              20480  0
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# lsmod | grep dm_crypt
dm_crypt               28672  0
```

원하는 storage 의 크기만큼 ramdisk 를 생성한다. 자신의 노트북 또는 컴퓨터의 memory 보다 작도록 설정한다. (약 1G 정도로 설정 권장).

- # sudo rapiddisk --attach 1024

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# rapiddisk --attach 1024
rapiddisk 4.4
Copyright 2011-2016 Petros Koutoupis

Attached device rd0 of size 1024 Mbytes
```

ramdisk 디바이스 파일(rd0)이 생성 되었는지 확인한다.

- # sudo ls –al /dev | grep rd0

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al /dev/ | grep rd
brw-rw----   1 root disk    251,   0  3월 10 16:54 rd0
```

확인이 되면 스토리지로 활용할 ramdisk 가 정상적으로 생성된 것이다.

iii. ramdisk 해제 하기.

ramdisk 를 활용한 실습 도중 스토리지를 format 해야 할 일이 자주 생길 수 있다. 이때 ramdisk 를 detach 후 다시 attach 하면 format 된 스토리지에서 다시 과제를 수행 할 수 있다.

- # sudo rapiddisk --detach rd0

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# rapiddisk --detach rd0
rapiddisk 4.4
Copyright 2011-2016 Petros Koutoupis

Detached device rd0
```

iv. xxd 명령어를 통한 스토리지 hex 값 확인

본 과제는 ramdisk 를 활용하여 disk 에 자신이 정의한 자료구조를 기록함으로써 UNIX 파일시스템의 방식으로 동작 할 수 있도록 구현하는 과제이다. 따라서 구현 도중 ramdisk 에 기록되는 모습을 확인하며 구현해야 한다. 이를 위해 xxd 라는 shell 상에서 바이너리(이진 값)의 hexdump 를 보여주는 명령어를 사용한다.

xxd 명령은 다음과 같은 옵션을 가지고 있다.

g : 바이너리 값을 인자로 넘긴 숫자 단위로 묶어 출력한다.

s : 바이너리 값을 인자로 넘긴 주소부터 출력한다.

l : 바이너리 값을 인자로 넘긴 값만큼 출력한다.

- # sudo xxd –g 4 –l 128 –s+0x0 /dev/rd0

  위의 명령어는 /dev/rd0 스토리지의 0x0 위치부터 4byte 만큼 묶어, 128byte 만큼 출력하라는 의미이며 결과는 아래와 같다.

  아래 결과는 실습을 구현한 사항의 예시이며, 0x0 위치에 super block 을 기록해 놓은 모습이며, super block 의 한 필드로 수업 이름, 학번을 기재해 놓은 모습을 볼 수 있다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# xxd -g 4 -l 128 -s+0x0 /dev/rd0
00000000: 4c414234 32111860 00020000 00000000  LAB42..`........
00000010: 03000000 00000000 00002000 00000000  .......... .....
00000020: 00000000 00000000 02000000 03000000  ................
00000030: 00000000 00000000 04000000 00000000  ................
00000040: 01000000 01000000 01000000 02000000  ................
00000050: 01000000 03000000 01000000 04000000  ................
00000060: fcff0300 1d000000 fcff0300 00000000  ................
00000070: 03000000 01000000 00000000 00000000  ................
```

- 참고 자료.

  http://www.tutorialspoint.com/unix_commands/xxd.htm

4. 과제 컴파일 방법

기본적으로 ramdisk 를 format 하기 위한 lab3_mkfs 실행 파일은 제공되지만, 파일시스템을 mount 후, 동작과 관련된 실행 파일인 lab3_mount 를 컴파일을 통해 생성해야 한다. 아래와 같이 make 명령어를 통해 컴파일을 수행할 수 있다.

- ***# make***

```
root@fslecture-VirtualBox:/home/fs-lecture/DKU_OS_LAB/lab3_fuse# ls
Makefile  dir.c    file.c       include  lab3_mkfs  mount.c
bmap.c    example  fuse_main.c  inode.c  mnt
root@fslecture-VirtualBox:/home/fs-lecture/DKU_OS_LAB/lab3_fuse# make
Compiling Lab3 fuse_main.c ...
gcc -c -D_GNU_SOURCE -g -W -I/home/fs-lecture/DKU_OS_LAB/lab3_fuse/include/ -D_FILE_OFFSET_BIT
S=64 -DFUSE_USE_VERSION=30 -I/usr/include/fuse  -o fuse_main.o fuse_main.c
Compiling Lab3 file.c ...
gcc -c -D_GNU_SOURCE -g -W -I/home/fs-lecture/DKU_OS_LAB/lab3_fuse/include/ -D_FILE_OFFSET_BIT
S=64 -DFUSE_USE_VERSION=30 -I/usr/include/fuse  -o file.o file.c
Compiling Lab3 dir.c ...
gcc -c -D_GNU_SOURCE -g -W -I/home/fs-lecture/DKU_OS_LAB/lab3_fuse/include/ -D_FILE_OFFSET_BIT
S=64 -DFUSE_USE_VERSION=30 -I/usr/include/fuse  -o dir.o dir.c
Compiling Lab3 inode.c ...
gcc -c -D_GNU_SOURCE -g -W -I/home/fs-lecture/DKU_OS_LAB/lab3_fuse/include/ -D_FILE_OFFSET_BIT
S=64 -DFUSE_USE_VERSION=30 -I/usr/include/fuse  -o inode.o inode.c
Compiling Lab3 mount.c ...
gcc -c -D_GNU_SOURCE -g -W -I/home/fs-lecture/DKU_OS_LAB/lab3_fuse/include/ -D_FILE_OFFSET_BIT
S=64 -DFUSE_USE_VERSION=30 -I/usr/include/fuse  -o mount.o mount.c
Compiling Lab3 bmap.c ...
gcc -c -D_GNU_SOURCE -g -W -I/home/fs-lecture/DKU_OS_LAB/lab3_fuse/include/ -D_FILE_OFFSET_BIT
S=64 -DFUSE_USE_VERSION=30 -I/usr/include/fuse  -o bmap.o bmap.c
gcc -o lab3_mount fuse_main.o file.o dir.o inode.o mount.o bmap.o -lfuse
root@fslecture-VirtualBox:/home/fs-lecture/DKU_OS_LAB/lab3_fuse# ls
Makefile  bmap.o  dir.o    file.c  fuse_main.c  include  inode.o    lab3_mount  mount.c
bmap.c    dir.c   example  file.o  fuse_main.o  inode.c  lab3_mkfs  mnt         mount.o
```

컴파일 후, lab3_mount 실행 파일과, 관련 object file 이 생성되는 것을 확인 할 수 있다. 또한 컴파일 결과를 지워야 할 경우 아래와 같은 명령어를 통해 수행할 수 있다.

- ***# make clean***

```
root@fslecture-VirtualBox:/home/fs-lecture/DKU_OS_LAB/lab3_fuse# make clean
Cleaning Lab3  ...
rm -rf fuse_main.o file.o dir.o inode.o mount.o bmap.o lab3_mount
root@fslecture-VirtualBox:/home/fs-lecture/DKU_OS_LAB/lab3_fuse# ls
Makefile  dir.c    file.c       include  lab3_mkfs  mount.c
bmap.c    example  fuse_main.c  inode.c  mnt
```

5. 과제 실행 및 디버깅 방법

FUSE 파일 시스템 실행 시, s 옵션을 통해 single thread 로 동작하도록 하여 실행한다.

i. 과제 실행 방법

① ramdisk format 수행.

모듈을 올리고 생성한 ramdisk 공간을 실습 시 사용할 파일시스템으로 포맷한다. lab3_mkfs 실행파일을 이용하여 아래와 같이 수행한다.

- **./lab3_mkfs –d /dev/rd0**

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mkfs -d /dev/rd0
---------------------------------------------------------------------
Formatting target device to lab file system
  Warning: your data will be removed permanentlyi ...
    super block size     : 120
    inode size           : 128
    dir entry size       : 128


    sb area size         : 4096 B
    sb area pos          : 0

    ibmap area size      : 4096 B
    ibmap area pos       : 1000

    dbmap area size      : 4096 B
    dbmap area pos       : 2000

    itble area size      : 4096 B
    itble area pos       : 3000

    data area size       : 4096 B
    data area pos        : 4000


 lab3 fuse file system formatted successfully.
---------------------------------------------------------------------
```

② **mount 수행**

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/rd0  mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

-------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30
```

- ***# ./lab3_mount –s /dev/rd0 [mount 를 수행할 directory]***

ramdisk 를 포맷하였으면 구현한 파일시스템을 사용하기 위해 파일시스템을 mount 해야 한다. 이를 위해 lab3_mount 라는 실행 파일을 사용한다. 아래 실행 예시는 구현한 lab4_mount 라는 FUSE 파일 시스템 실행 파일을 수행한 결과이다. s 옵션을 주어 single thread 에서 동작하도록 하였으며, /dev/rd0 라는 ramdisk 위치와 mount 위치인 mnt 를 명시해 주어 해당 위치의 저장장치를 mnt 에 mount 하여 동작하도록 하였다. mount 위치는 다른 directory 에 수행하여도 된다.

파일시스템을 mount 한 후, 아래 명령어를 통해 파일시스템이 정상적으로 mount 되었는지 확인 할 수 있다. mount 명령어는 mount 된 파일시스템을 확인하는 명령어이다.

- ***# mount | grep lab***

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# mount | grep lab
/home/os-lecture/DKU_OS_LAB/lab3_fuse/lab3_mount on /home/os-lecture/DKU_OS_LAB/
lab3_fuse/mnt type fuse.lab3_mount (rw,nosuid,nodev,relatime,user_id=0,group_id=
0)
```

mount 된 이후, 실습에 다양한 명령어를 통해 실습에서 구현한 결과를 확인하고 gdb 를 통해 디버깅 해본다.

실습 구현 완료 후, 또는 구현 시 문제가 있어 디버깅 후 다시 사용하고자 할 때, 또는 사용을 완료하였을 때, mount 한 파일시스템을 unmount 해주어야 한다. umount 는 아래와 같은 명령어를 통해 mount 를 해제 한다.

- ***# umount mnt [mount 된 directory]***

```
root@fslecture-VirtualBox:/home/fs-lecture/DKU_OS_LAB/lab3_fuse# umount mnt/
root@fslecture-VirtualBox:/home/fs-lecture/DKU_OS_LAB/lab3_fuse# mount | grep lab
root@fslecture-VirtualBox:/home/fs-lecture/DKU_OS_LAB/lab3_fuse#
```

ii. 과제 디버깅 방법

FUSE 는 background 로 사용자가 구현한 파일시스템을 동작시킨다. 따라서 gdb 를 통해 debugging 수행 시, 작성한 FUSE 프로그램을 실행할 때, 몇가지 옵션을 주어야 한다.

✓ FUSE 디버깅 실행 옵션

- d : 디버깅 옵션으로 FUSE 내부에서 각 함수의 호출 단계를 보여준다.
- f : FUSE 구현 사항이 foreground 로 수행되도록 하여, 출력 스트림이 살아 있도록 한다. 이를 통해 printf 등을 디버깅에 사용할 수 있다.
- s : multi thread가 아닌, single thread 로 FUSE가 동작하도록 한다. 이를 통해 race dondition을 방지 할 수 있다.

✓ 디버깅 예시

- ***# gdb lab3_mount***

아래는 구현한 FUSE 파일시스템 실행 파일인 lab4_mount 를 dgb로 디버깅하기 위해 gdb를 실행한 모습이다.

```
root@fslecture-VirtualBox:/home/fs-lecture/DKU_OS_LAB/lab3_fuse# gdb lab3_mount
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/fs-lecture/DKU_OS_LAB/lab3_fuse/lab3_mount...done.
(gdb)
```

위와 같이gdb 실행 후, tracking 하려는 함수에 대해 breakpoint를 설정한다. 아래 예시는 directory를 읽어 들이는 readdir함수에 대해 디버깅하기 위해 fuse_main.c 파일의 lab3_readdir 함수에 break point를 설정한 모습이다.

- ***(gdb) b fuse_main.c:lab3_readdir***

```
(gdb) b fuse_main.c:lab3_readdir
Breakpoint 1 at 0x401343: file fuse_main.c, line 149.
```

아래와 같이 정규 표현식을 사용하여 해당 파일내의 특정 패턴을 가지는 모든 함수에 대해 breakpoint를 설정 할 수도 있다.

- ***(gdb) rb fuse_main.c:lab3****

```
(gdb) rb fuse_main.c:lab3*
Breakpoint 2 at 0x401278: file fuse_main.c, line 121.
void *lab3_init(struct fuse_conn_info *);
Breakpoint 3 at 0x401b5e: file fuse_main.c, line 392.
static int lab3_create(const char *, mode_t, struct fuse_file_info *);
Breakpoint 4 at 0x401299: file fuse_main.c, line 127.
static int lab3_getattr(const char *, struct stat *);
Breakpoint 5 at 0x401446: file fuse_main.c, line 186.
static int lab3_mkdir(const char *, mode_t);
Breakpoint 6 at 0x4019a4: file fuse_main.c, line 325.
static int lab3_open(const char *, struct fuse_file_info *);
Breakpoint 7 at 0x401a07: file fuse_main.c, line 337.
static int lab3_read(const char *, char *, size_t, off_t, struct fuse_file_info *);
Note: breakpoint 1 also set at pc 0x401343.
Breakpoint 8 at 0x401343: file fuse_main.c, line 149.
static int lab3_readdir(const char *, void *, fuse_fill_dir_t, off_t, struct fuse_file_info *);
Breakpoint 9 at 0x401af6: file fuse_main.c, line 372.
static int lab3_release(const char *, struct fuse_file_info *);
Breakpoint 10 at 0x4018de: file fuse_main.c, line 285.
static int lab3_rename(const char *, const char *);
Breakpoint 11 at 0x40175b: file fuse_main.c, line 247.
static int lab3_rmdir(const char *);
Breakpoint 12 at 0x4015f5: file fuse_main.c, line 219.
static int lab3_unlink(const char *);
Breakpoint 13 at 0x401bb3: file fuse_main.c, line 406.
static int lab3_utimens(const char *, const struct timespec *);
Breakpoint 14 at 0x401a7b: file fuse_main.c, line 352.
static int lab3_write(const char *, const char *, size_t, off_t, struct fuse_file_info *);
```

다음은 설정한 breakpoint 들을 확인해본 모습이다.

- ***(gdb) info b***

```
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000401343 in lab3_readdir at fuse_main.c:149
2       breakpoint     keep y   0x0000000000401278 in lab3_init at fuse_main.c:121
3       breakpoint     keep y   0x0000000000401b5e in lab3_create at fuse_main.c:392
4       breakpoint     keep y   0x0000000000401299 in lab3_getattr at fuse_main.c:127
5       breakpoint     keep y   0x0000000000401446 in lab3_mkdir at fuse_main.c:186
6       breakpoint     keep y   0x00000000004019a4 in lab3_open at fuse_main.c:325
7       breakpoint     keep y   0x0000000000401a07 in lab3_read at fuse_main.c:337
8       breakpoint     keep y   0x0000000000401343 in lab3_readdir at fuse_main.c:149
9       breakpoint     keep y   0x0000000000401af6 in lab3_release at fuse_main.c:372
10      breakpoint     keep y   0x00000000004018de in lab3_rename at fuse_main.c:285
11      breakpoint     keep y   0x000000000040175b in lab3_rmdir at fuse_main.c:247
12      breakpoint     keep y   0x00000000004015f5 in lab3_unlink at fuse_main.c:219
13      breakpoint     keep y   0x0000000000401bb3 in lab3_utimens at fuse_main.c:406
14      breakpoint     keep y   0x0000000000401a7b in lab3_write at fuse_main.c:352
```

디버깅을 하려면, breakpoint 설정을 통해 tracking 할 함수를 정했다면, 프로그램을 실행시켜 실제 프로그램 내부에서 해당 함수가 어떻게 사용이 되는지 확인해야 한다. FUSE 파일 시스템은 디버깅을 위해 d 옵션을 제공한다. 이를 아래와 같이 실행한다.

- ***(gdb) r –d –s /dev/rd0 mnt***

d 옵션을 주어 FUSE 파일 시스템을 실행하게 되면, FUSE 는 mount가 된 후 사용자의 입력이 들어오기 까지 아래 그림과 같이 대기하게 된다. 그 후 다른 terminal 창을 열어 해당 위치에서 명령어를 실행해 봄으로써 디버깅을 수행 할 수 있다.

```
(gdb) r -d -s /dev/rd0 mnt/
Starting program: /home/os-lecture/DKU_OS_LAB/lab3_fuse/lab3_mount -d -s /dev/rd0 mnt/
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

-------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

FUSE library version: 2.9.4
nullpath_ok: 0
nopath: 0
utime_omit_ok: 0
unique: 1, opcode: INIT (26), nodeid: 0, insize: 56, pid: 0
INIT: 7.23
flags=0x0003f7fb
max_readahead=0x00020000
   INIT: 7.19
   flags=0x00000011
   max_readahead=0x00020000
   max_write=0x00020000
   max_background=0
   congestion_threshold=0
   unique: 1, success, outsize: 40
```

아래 그림은 두개의 Terminal 창을 통해 디버깅을 하는 과정에서 아래

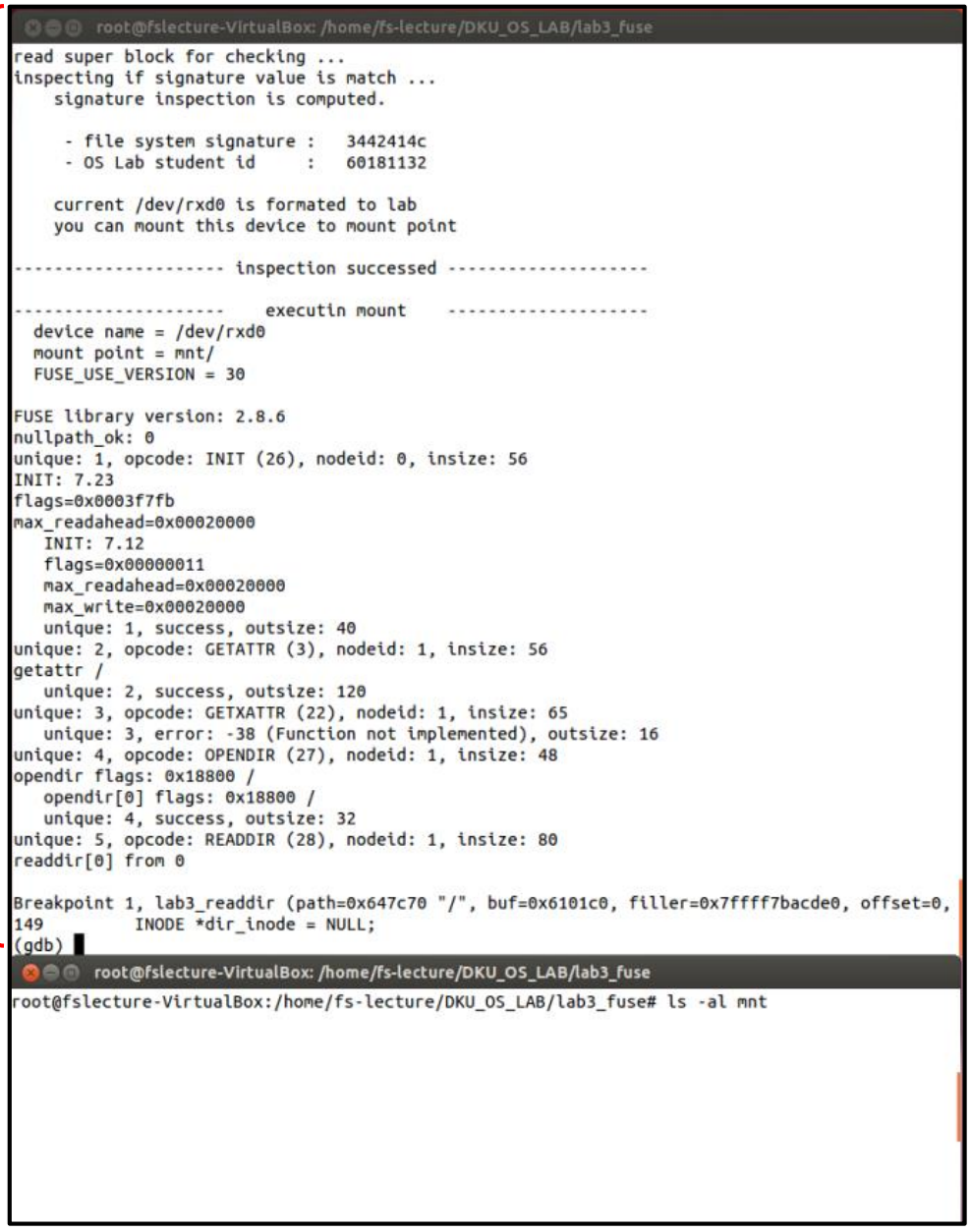
터미널에서 ls mnt 명령을 통해 readdir 함수를 호출하게 된 결과 위의 terminal 에서 readdir 에서 멈춘 결과이다.

Terminal A

gdb 가 실행중인 터미널, 파일시스템 명령 수행 터미널에서 ls, mv, mkdir 등의 명령에 따른 함수 동작 과정 디버깅 수행.

Terminal B

파일시스템 명령을 수행할 터미널, ls, mv, mkdir 등의 명령 수행



이 후, 일반 모드에서 디버깅을 수행하거나 TUI 모드에서 디버깅을 수행 할 수 있다.

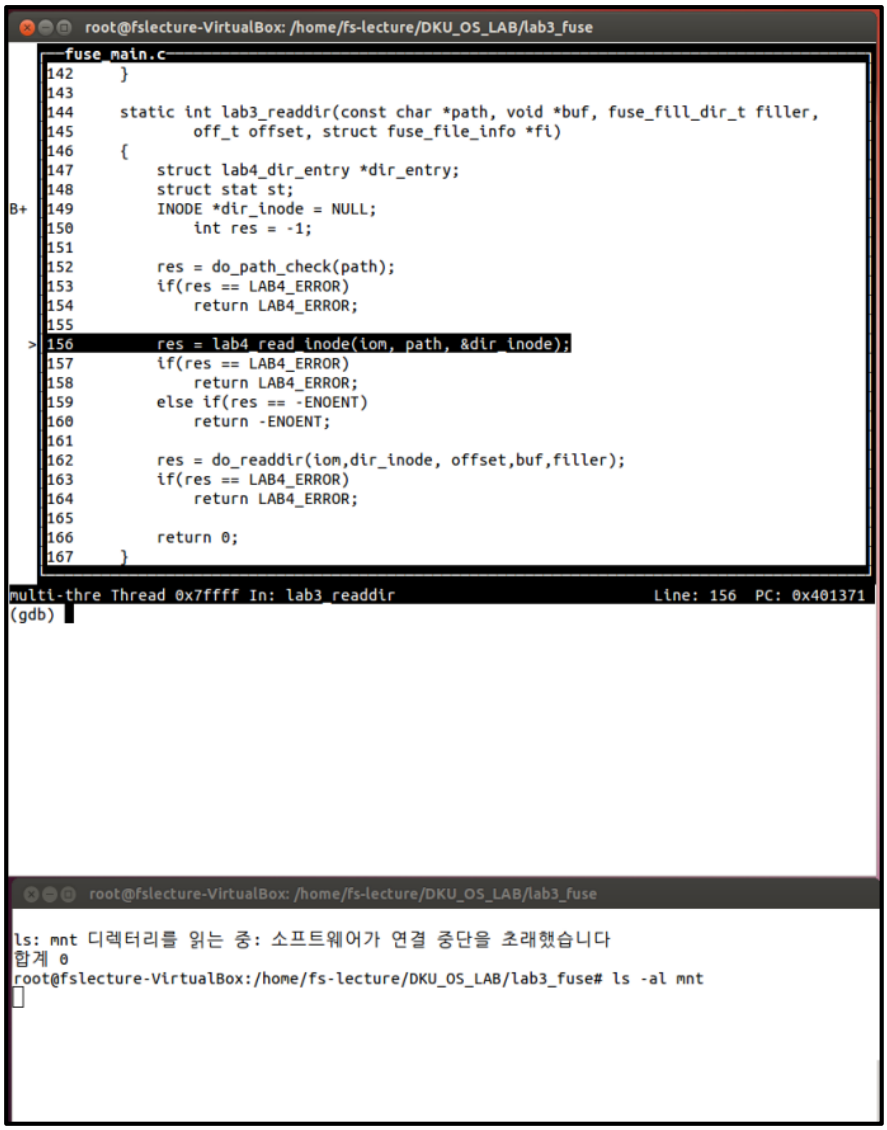
- ***(gdb) -***

위와 같이 – 명령을 통해 TUI 모드로 전환하게 되면 아래 그림과 같이 수행이 멈추게 된 위치의 코드를 보여준다. 이를 통해 현재 디버깅 수행중인 위치를 확인 하며 디버깅이 가능하며, n, s u, finish, c 등의 gdb 명령을 통해 해당 함수를 통해 디버깅을 수행 할 수 있다. (GDB 의 기

본 명령어는 Lab0 문서에서 설명하였으므로 본 문서에서는 설명을 생략하도록 한다.)

Terminal A

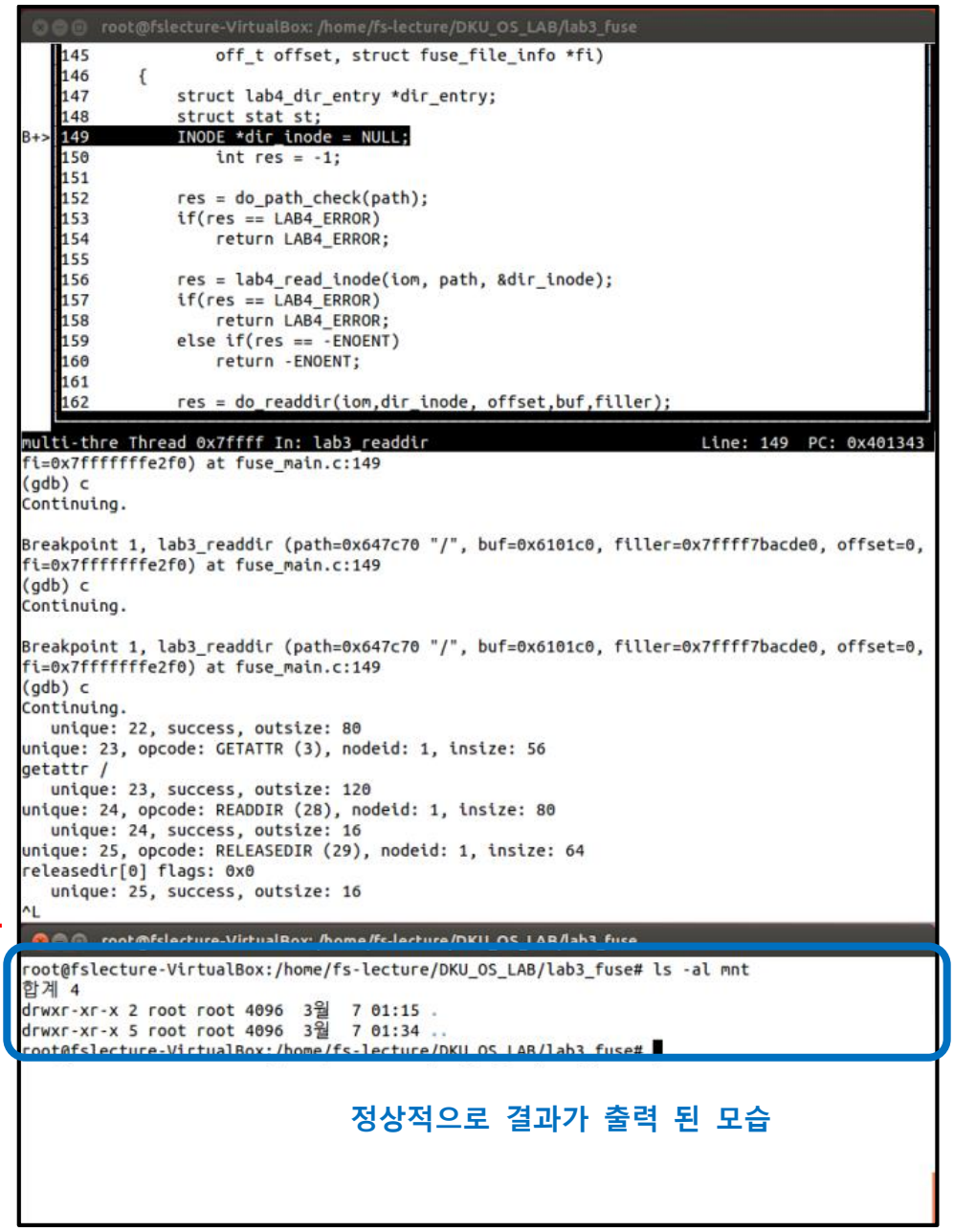
Terminal B



디버깅 중 해당 함수가 완료되면 정상적으로 구현이 되었다면, 아래 Terminal 에 결과가 출력되는 것을 확인 할 수 있다.

Terminal A

Terminal B



정상적으로 구현이 되지 않은 경우 아래와 같은 에러들이 출력되는 것을 볼 수 있다. 아래 결과는 lab3_readdir 함수에 아무런 내용을 구현하지 않은 출력 결과이다.

Terminal A

Terminal A

iii. 과제 ramdisk 초기화 포맷 방법.

본 과제는 파일시스템을 만들기 위해 ramdisk 를 포멧하고, ramdisk 에 실제로 wirte, read 를 수행하는 과제이다. 따라서, directory create, file create 관련 과제 중, 잘못된 위치에 write 수행 시, ramdisk 를 다시 처음으로 format 해야 할 수 있다. ramdisk 를 다시 초기화 하려면 먼저 ramdisk 가 mount 된 상태라면 unmount 수행 후, 아래와 같은 명령어를 통해 ramdisk 를 detach 후 다시 attach 를 수행한다.

- ***# rapiddisk --detach rd0***

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# umount mnt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# rapiddisk --detach rd0
rapiddisk 4.4
Copyright 2011-2016 Petros Koutoupis

Detached device rd0
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# rapiddisk --attach 1024
rapiddisk 4.4
Copyright 2011-2016 Petros Koutoupis

Attached device rd0 of size 1024 Mbytes
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls /dev/ | grep rd0
rd0
```

이렇게 다시 attach 된 ramdisk 는 0 으로 초기화 된 상태이므로, 다시 format 하여 실습을 수행 할 수 있다.

6. 과제 구현 과정.

구현된 mount.c 의 소스는 구현의 편의성을 위해 mount 될 때, 아래와 같은 include/lab3_fs_types.h 위치의 struct io_manager 라고 정의한 구조체 변수를 생성하여

```
struct io_manager{
    s8 *dev_path; /* device file path (e.g. /dev/rd0 )  */
    s8* mnt_path; /* mount path (e.g. mnt ) */
    int dev_fd; /* opened device file's file descriptor(used to write&read device file)  */
    int no_of_sectors; /* number of sectors in device file  */
    int blk_size; /* block size in device file  */
    int cluster_size; /* cluster size in device file (write, read unit)  */
    int format_type; /* format type (current file system format type)  */
    struct lab3_super_block *sb; /* super block which you read from debvice file  */
    struct lab3_sb_info *sbi; /* super block in  */
    char *iom_buf_ibmap; /* inode bitmap buffer  */
    char *iom_buf_dbmap; /* data bitmap buffer  */
    char *iom_buf_itble; /* inode table bitmap buffer  */
};
```

아래와 같은 fuse_main.c 의 전역 변수인 iom 에 채우도록 한다.

```
IOM *iom;
```

따라서 mount 후 이 변수를 이용하여 write, read 등을 수행할 수 있다. (보너스 과제를 위해 mount 부분 및 format 부분을 자체 구현 시 위의 구현 사항을 따르지 않아도 된다.)

본 과제의 구현을 위해 fuse_main.c 파일의 아래와 같은 연산들을 구현해야 한다.

```
static struct fuse_operations lab3_oper = {
    .init       = lab3_init,
    .getattr    = lab3_getattr,
    .readdir    = lab3_readdir,
    .mkdir      = lab3_mkdir,
    .unlink     = lab3_unlink,
    .rmdir      = lab3_rmdir,
    .rename     = lab3_rename,
    .open       = lab3_open,
    .read       = lab3_read,
    .write      = lab3_write,
    .release    = lab3_release,
    .create     = lab3_create,
    .utimens    = lab3_utimens
};
```

A. Step 1. init 함수 , metadata 관련 함수 구현.

lab3_init 함수는 file system 이 mount 될 때, 가장 먼저 호출되는 함수이다. 구현 의존적인 함수로 추가적으로 구현하지 않아도 과제 내용이 작동이 가능할 수 있다. getattr 함수는 파일시스템의 가장 기초적인 파일의 inode 정보를 읽어 들이는 함수이다. 본 과제에서는 lab3_getattr 함수를 통해 이를 구현 할 수 있다. 해당 함수를 구현하기 전에는 mount 후 아래와 같은 결과를 볼 수 있다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/
rd0 mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

-------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls
ls: 'mnt'에 접근할 수 없습니다: 범위를 벗어난 수치 결과
Makefile  dir.o    fsformat.c          fuse_main.c  inode.o     mount.c
bmap.c    example  fsformat.o          fuse_main.o  lab3_mkfs   mount.o
bmap.o    file.c   fuse_format_main.c  include      lab3_mount  tags
dir.c     file.o   fuse_format_main.o  inode.c      mnt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt
ls: 'mnt'에 접근할 수 없습니다: 범위를 벗어난 수치 결과
```

위 그림은 mnt 라는 direcctory 에 mount 수행 후, mnt directory 의 부모 directory 에서 ls 를 수행한 결과와, mnt directory 를 조회 한 결과이다. 하지만 getattr 함수가 구현되어 있지 않기 때문에, mount directory 인 mnt 의 inode 를 가져오지 못해 위와 같이 오류가 발생하는 것을 확인 할 수 있다. 정상적으로 getattr 함수를 구현하게 되면 아래와 같은 결과를 확인 할 수 있다. 하지만 mount directry 를 읽어 들이는 readdir 함수의 경우 아직 구현이 되지 않은 상태이기 때문에 ls mnt 연산에서 아무런 결과가 나오지 않는 것을 확인할 수 있다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/
rd0 mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

-------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls
Makefile  dir.o    fsformat.c          fuse_main.c  inode.o     mount.c
bmap.c    example  fsformat.o          fuse_main.o  lab3_mkfs   mount.o
bmap.o    file.c   fuse_format_main.c  include      lab3_mount  tags
dir.c     file.o   fuse_format_main.o  inode.c      mnt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 0
```

```
static struct fuse_operations lab3_oper = {
    .init       = lab3_init,
    .getattr    = lab3_getattr,
    .readdir    = lab3_readdir,
    .mkdir      = lab3_mkdir,
    .unlink     = lab3_unlink,
    .rmdir      = lab3_rmdir,
    .rename     = lab3_rename,
    .open       = lab3_open,
    .read       = lab3_read,
    .write      = lab3_write,
    .release    = lab3_release,
    .create     = lab3_create,
    .utimens    = lab3_utimens
};
```

Step1 을 통해 구현 해야 하는 함수 중 위와 같은 함수들을 해결하였다.

B. Step 2. directory read create, delete 관련 함수 구현

readdir 함수는 읽으려는 directory 의 inode 를 읽어 해당 directory 가 위치한 block 주소 위치를 알아내서 결과적으로 directory 내부에 있는 directory entry 를 읽어 들이는 함수이다. 해당 함수를 구현하지 않으면 아래와 같이 directory 에 내용이 존재해도 아무런 결과를 출력하지 않는 것을 확인 할 수 있다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/
rd0 mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

-------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt
합계 0
```

위 예시의 경우 root directory 에 대해 ls 명령어를 수행한 것이므로, root directory 의 inode 를 inode table 로부터 읽어 들여, root directory 가 위치한 block 주소를 알아내어, 그 위치에 존재하는 directory entry 를 읽어 들이도록 구현하면 아래와 같은 결과를 확인 할 수 있다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/
rd0 mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

-------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 2 root       root       4096  3월 10 06:59 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 07:38 ..
```

위 그림을 통해 정상적으로 구현 시, mount directory 에 대해 ls 연산을 수행하면 모든 root directory 에 포함되어 있는 “.” , “..” 의 두가지 directory entry 를 볼 수 있다.

directory 생성 함수는 생성하려는 directory 의 부모 directory 의 inode 로부터 block 위치를 알아내서 해당 block 에 새로운 directory 관련 directory entry 를 생성하는 함수이며 본 과제에서는 lab3_mkdir 함수를 통해 구현 할 수 있다. 이 함수가 정상적으로 구현이 되어 있지 않는다면 아래와 같은 결과를 볼 수 있다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/rd0 mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

-------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# mkdir mnt/test
mkdir: `mnt/test' 디렉토리를 만들 수 없습니다: 범위를 벗어난 수치 결과
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt
합계 4
drwxr-xr-x 2 root       root       4096  3월 10 06:59 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 07:42 ..
```

위 그림에서는 mount 후 mount directory 인 mnt 에 test 라는 이름의 directory entry 를 생성하려 하고 있다. 하지만 lab3_mkdir 함수가 정상적으로 구현이 되지 않아, 오류를 출력하며, 실제 directory 도 생성이 되지 않을 것을 볼 수 있다. lab3_mkdir 함수가 정상적으로 구현이 되었다면 아래와 같은 결과를 확인 할 수 있다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/rd0 mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

--------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# mkdir mnt/test
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt
합계 4
drwxr-xr-x 3 root       root       4096  3월 10 06:59 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 12:13 ..
drwxr-xr-x 2 root       root       4096  3월 10 12:14 test
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse#
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/test/
합계 0
drwxr-xr-x 2 root root 4096  3월 10 12:14 .
drwxr-xr-x 3 root root 4096  3월 10 06:59 ..
```

위 그림에서 mkdir 명령어를 통해 mount directory 내에 test 라는 directory 가 생성되었으며, test 라는 directory 에 “.”, “..” 라는 이름의 directory entry 라는 것이 존재하는 것을 알 수 있다. ls –al 결과 출력되는 message 에서 가장 왼쪽부터 접근 권한 (http://linuxcommand.org/lts0070.php), link count(http://www.theunixschool.com/2012/10/link-count-file-vs-directory.html), 파일 소유자, 그룹, 크기를 의미한다. 모든 directory 는 생성되면 기본적으로 자신의 inode 가리키는 “.”라는 이름의 directory entry, 부모 directory entry 의 inode 를 가리키는 “..” 라는 이름의 directory entry 를 기본적으로 가지게 된다. 따라서 이를 생성하도록 구현하여야 한다.

directory delete 함수는 삭제하려는 directory 의 부모 directory 로부터 부모 directory 의 inode 를 읽어 들여, 부모 directory 의 block 이 들어 있는 위치에 존재하는 삭제할 directory 관련 directory entry 를 삭제하고, 삭제할 directory entry 로부터 그 inode 를 알아내 삭제할 directory 의 block 에 위치한 내용을 지우는 함수이다. 본 과제에서는 lab3_rmdir 함수를 통해 구현이 될 수 있다. lab3_rmdir 함수가 정상적으로 구현이 되어있지 않다면 아래와 같은 결과를

출력하게 된다. 아래 그림을 통해 directory 삭제 명령을 내리면 오류를 출력하게 되며, 실제 directory 도 삭제되지 않고 남아있는 것을 확인 할 수 있다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/rd0 mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

-------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 3 root       root       4096  3월 10 06:59 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 13:02 ..
drwxr-xr-x 2 root       root       4096  3월 10 12:14 test
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# rmdir mnt/test/
rmdir: failed to remove 'mnt/test/': 범위를 벗어난 수치 결과
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 3 root       root       4096  3월 10 06:59 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 13:02 ..
drwxr-xr-x 2 root       root       4096  3월 10 12:14 test
```

해당 함수를 정상적으로 구현하게 되면 아래와 같은 결과를 확인 할 수 있다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/rd0 mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

-------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 3 root       root       4096  3월 10 06:59 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 13:06 ..
drwxr-xr-x 2 root       root       4096  3월 10 12:14 test
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# rmdir mnt/test/
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 2 root       root       4096  3월 10 06:59 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 13:06 ..
```

위의 결과를 통해 test 라는 directory 가 정상적으로 삭제 된 것을 확인 할 수 있다.

```
static struct fuse_operations lab3_oper = {
    .init       = lab3_init,
    .getattr    = lab3_getattr,
    .readdir    = lab3_readdir,
    .mkdir      = lab3_mkdir,
    .unlink     = lab3_unlink,
    .rmdir      = lab3_rmdir,
    .rename     = lab3_rename,
    .open       = lab3_open,
    .read       = lab3_read,
    .write      = lab3_write,
    .release    = lab3_release,
    .create     = lab3_create,
    .utimens    = lab3_utimens
};
```

Step 2 를 통해 구현 해야 하는 함수 중 위와 같은 함수들을 해결하였다.

C. Step 3. file create, read, write ,delete 관련 함수 구현.

먼저 file create 함수를 파일을 생성하는 함수이다. 파일의 생성, 접근, 삭제 등 파일 관련 기본 연산을 구현하기 위해서는 기본적으로 create, open, release 함수를 구현해야 한다. create 함수는 함수는 생성할 파일 관련 inode 를 inode table 에 추가, 생성 할 파일 관련 directory entry 를 부모 directory 에 추가하는 함수이다. 본 과제에서는 lab3_create 함수를 통해 구현 할 수 있다. 또한 open 함수는 생성한 파일에 대해 file descriptor 를 open 하여, 현재 어디까지 파일에 write & read 를 수행 하였는지를 관리하는 함수이다. open 함수는 lab3_open 함수를 통해 구현 할 수 있다. release 함수는 사용을 완료한 파일에 대해 file descriptor 를 닫는 함수이며 lab3_relaease 함수를 통해 구현 할 수 있다. create & open & release 함수가 정상적으로 구현이 되어 있지 않는다면 크기가 0 인 간단한 파일을 생성하는 명령어에서 아래와 같은 결과를 출력하게 된다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/rd0 mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

--------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls mnt/
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# touch mnt/test.txt
touch: 'mnt/test.txt'를 touch할 수 없음: 범위를 벗어난 수치 결과
```

lab3_create 함수와 lab3_open 함수를 정상적으로 출력하게 되면 아래와 같은 결과를 확인 할 수 있다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/rd0 mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

-------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# touch mnt/test.txt
touch: 'mnt/test.txt'의 시간을 설정: 범위를 벗어난 수치 결과
```

이를 통해 아직 무언가 정확히 구현이 되지 않았다는 것을 알 수 있다. 파일을 생성, read, write 등 파일에 대한 접근이 일어나게 되면, 마지막으로 해당 경로 파일의 접근 시간 등을 수정해 주어야 한다. 이는 본 과제의 lab3_utimens 함수를 통해 구현 할 수 있다. lab3_utmesns 함수를 구현하게 되면 아래와 같이 빈 파일이 생성되는 것을 확인 할 수 있다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/rd0 mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

-------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# touch mnt/test.txt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt
합계 4
drwxr-xr-x 2 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 13:30 ..
-rw-r--r-- 1 root       root          0  3월 10 13:26 test.txt
```

이제 빈 파일이 아니라 파일에 대해 wirte 를 수행하도록 해본다. write 함수는 본 과제의 lab3_write 함수에서 구현될 수 있으며, 해당 경로의 inode 를 읽어 들여

data 가 저장 된 block 주소에 값을 write 하는 함수이다. write 함수가 수행되기 전에는 아래와 같은 결과를 확인 할 수 있다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse#
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/rd0 mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

-------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 2 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 13:37 ..
-rw-r--r-- 1 root       root          0  3월 10 13:26 test.txt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# echo aaa >> mnt/test.txt
bash: echo: 쓰기 오류: 입력/출력 오류
```

정상적으로 write 함수를 구현 하게 되면 아래와 같은 결과를 확인 할 수 있다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/rd0  mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

-------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 2 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 13:44 ..
-rw-r--r-- 1 root       root          0  3월 10 13:26 test.txt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# echo aaa >> mnt/test.txt
```

아직 read 함수를 구현하지 않았기 때문에 xxd 명령어를 통해 test.txt 파일이 기록된 위치를 test.txt 의 inode 를 통해 확인하여 해당 위치의 hex 값을 확인 해 보면 아래 그림과 같이 aaa 가 기록된 모습을 확인 할 수 있다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# xxd -g 4 -l 128 -s+0x5000 /dev/rd0
00005000: 6161610a 00000000 00000000 00000000  aaa.............
00005010: 00000000 00000000 00000000 00000000  ................
00005020: 00000000 00000000 00000000 00000000  ................
00005030: 00000000 00000000 00000000 00000000  ................
00005040: 00000000 00000000 00000000 00000000  ................
00005050: 00000000 00000000 00000000 00000000  ................
00005060: 00000000 00000000 00000000 00000000  ................
00005070: 00000000 00000000 00000000 00000000  ................
```

이제 기록한 사항에 대해 읽어 들일 차례이다. read 함수는 읽어 들일 파일의 inode 정보를 통해 파일의 block 위치를 알아내고, block 위치로부터, 파일의 data 를 읽어 들인다. 본 과제에서 lab3_read 함수를 통해 구현할 수 있다. 해당 함수를 구현하기 전에는 read 명령에 대해 아래와 같은 결과를 확인 할 수 있다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/rd0 mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

-------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 2 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 14:09 ..
-rw-r--r-- 1 root       root          4  3월 10 13:26 test.txt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# cat mnt/test.txt
cat: mnt/test.txt: 입력/출력 오류
```

정상적으로 lab3_read 함수를 구현하게 되면 아래와 같은 결과를 확인 할 수 있다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/rd0 mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

-------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 2 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 14:11 ..
-rw-r--r-- 1 root       root          4  3월 10 13:26 test.txt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# cat mnt/test.txt
aaa
```

마지막으로 생성한 파일에 대해 삭제하는 연산을 구현하도록 해본다. delete 연산은 unlink 연산 이라고 하며, 삭제할 파일의 부모 directory 에서 삭제할 파일의 directory entry 를 읽어 inode 정보를 알아내고 파일의 data 가 저장된 block 의 내용을 삭제한다. 본 과제에서는 lab3_unlink 함수를 통해 구현할 수 있다. unlink 함수를 구현하기 전 결과는 아래 그림과 같다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/rd0 mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

-------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 2 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 14:41 ..
-rw-r--r-- 1 root       root          4  3월 10 13:26 test.txt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# rm -rf mnt/test.txt
rm: 'mnt/test.txt'를 지울 수 없음: 범위를 벗어난 수치 결과
```

lab3_unlink 함수를 정상적으로 구현 하게 되면 아래와 같은 결과를 볼 수 있다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/rd0 mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

-------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 2 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 14:43 ..
-rw-r--r-- 1 root       root          4  3월 10 13:26 test.txt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# rm -rf mnt/test.txt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 2 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 14:43 ..
```

```
static struct fuse_operations lab3_oper = {
    .init       = lab3_init,
    .getattr    = lab3_getattr,
    .readdir    = lab3_readdir,
    .mkdir      = lab3_mkdir,
    .unlink     = lab3_unlink,
    .rmdir      = lab3_rmdir,
    .rename     = lab3_rename,
    .open       = lab3_open,
    .read       = lab3_read,
    .write      = lab3_write,
    .release    = lab3_release,
    .create     = lab3_create,
    .utimens    = lab3_utimens
};
```

Step 3 를 통해 구현 해야 하는 함수 중 위와 같은 함수들을 해결하였다.

D. Step 4. rename 관련 함수 구현.

rename 함수는 기존 같은 경로에서 파일의 이름을 변경하거나 파일의 위치를 옮기는 함수로 경로에 해당하는 directory entry 의 file name 을 수정하거나, directory entry 를 다른 경로로 옮기는 함수이다. 본 과제에서 lab3_rename 함수를 통해 구현 할 수 있다. lab3_rename 함수를 구현하기 전 결과는 아래와 같다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/rd0 mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

-------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# touch aaa > mnt/test.txt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt
합계 4
drwxr-xr-x 2 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 14:55 ..
-rw-r--r-- 1 root       root          0  3월 10 14:56 test.txt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# mv mnt/test.txt mnt/rename.txt
mv: 'mnt/test.txt'를 'mnt/rename.txt'로 옮길 수 없음: 범위를 벗어난 수치 결과
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt
합계 4
drwxr-xr-x 2 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 14:55 ..
-rw-r--r-- 1 root       root          0  3월 10 14:56 test.txt
```

위 그림과 같이 test.txt 라는 파일의 이름이 정상적으로 변경이 되지 않은 것을 확인 할 수 있으며, 정상적으로 lab3_rename 함수를 구현하게 되면 아래와 같은 결과를 확인 할 수 있다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ./lab3_mount -s /dev/rd0 mnt/

 device /dev/sdc is selected

 check if it is formatted to lab or fat ...

--------------------  format inspection   --------------------
read super block for checking ...
inspecting if signature value is match ...
    signature inspection is computed.

     - file system signature :   3442414c
     - OS Lab student id     :   60181132

    current /dev/rd0 is formated to lab
    you can mount this device to mount point

-------------------- inspection successed --------------------

--------------------    executin mount    --------------------
  device name = /dev/rd0
  mount point = mnt/
  FUSE_USE_VERSION = 30

root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# touch aaa > mnt/test.txt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt
합계 4
drwxr-xr-x 2 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 14:54 ..
-rw-r--r-- 1 root       root          0  3월 10 14:54 test.txt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# mv mnt/test.txt mnt/rename.txt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt
합계 4
drwxr-xr-x 2 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 14:54 ..
-rw-r--r-- 1 root       root          0  3월 10 14:54 rename.txt
```

```
static struct fuse_operations lab3_oper = {
    .init       = lab3_init,
    .getattr    = lab3_getattr,
    .readdir    = lab3_readdir,
    .mkdir      = lab3_mkdir,
    .unlink     = lab3_unlink,
    .rmdir      = lab3_rmdir,
    .rename     = lab3_rename,
    .open       = lab3_open,
    .read       = lab3_read,
    .write      = lab3_write,
    .release    = lab3_release,
    .create     = lab3_create,
    .utimens    = lab3_utimens
};
```

Step4 를 통해 구현 해야 하는 함수를 모두 해결하였다.

7. 과제 평가 사항.

본 실습을 통해 getattr, readdir, writre, read 등의 파일시스템의 핵심 역할을 하는 함수들을 구현하였다. Step1~ 4 까지 걸쳐 구현한 사항들을 아래와 같이 평가 및 확인 한다.

i. Step 1 : file metadata (stat)

파일시스템의 metadata 정보를 가져오는 stat 명령어 정상적으로 동작해야 하며 ls 를 통해 mount directory 의 내용을 볼 수 있어야 한다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# stat mnt/
  File: 'mnt/'
  Size: 4096            Blocks: 0          IO Block: 4096   디렉토리
Device: 27h/39d Inode: 1           Links: 2
Access: (0755/drwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2017-03-10 13:19:47.000000000 +0900
Modify: 2017-03-10 13:19:47.000000000 +0900
Change: 2017-03-10 13:19:47.000000000 +0900
 Birth: -
```

ii. Step 2 : read directory (ls), directory create (mkdir), & directory delete (rmdir)

ls 명령어를 통해 mount directory 의 내용을 확인 할 수 있어야 하고, directory 를 생성하는 mkdir 명령어가 정상적으로 동작해야 하며, directory 생성 시 자신을 가리키는 “.” 와 상위 directory 를 가리키는 “..” 가 생성되어야 한다. 또한 같은 이름의 directory 를 생성하려 할 때, 에러를 메시지를 띄울 수 있어야 한다. 또한 directory 를 삭제하는 rmdir 명령어가 정상적으로 동작해야 하며, 비어있지 않은 directory 를 삭제 하려 할 때, directory 삭제할 수 없다는 에러 메시지를 띄울 수 있어야 한다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 2 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 15:06 ..
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# mkdir mnt/test1
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# mkdir mnt/test2
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# mkdir mnt/test1/test3
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 4 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 15:06 ..
drwxr-xr-x 3 root       root       4096  3월 10 15:06 test1
drwxr-xr-x 2 root       root       4096  3월 10 15:07 test2
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/test1/
합계 0
drwxr-xr-x 3 root root 4096  3월 10 15:06 .
drwxr-xr-x 4 root root 4096  3월 10 13:19 ..
drwxr-xr-x 2 root root 4096  3월 10 15:08 test3
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse#
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse#
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# mkdir mnt/test1/test3/
mkdir: `mnt/test1/test3/' 디렉토리를 만들 수 없습니다: 파일이 있습니다
```

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 4 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 15:06 ..
drwxr-xr-x 3 root       root       4096  3월 10 15:06 test1
drwxr-xr-x 2 root       root       4096  3월 10 15:09 test2
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# rmdir mnt/test2/
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 3 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 15:06 ..
drwxr-xr-x 3 root       root       4096  3월 10 15:06 test1
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# rmdir mnt/test1/
rmdir: failed to remove 'mnt/test1/': 디렉터리가 비어있지 않음
```

iii. Step 3 : file create (touch mnt/test.txt) , file write (echo >> mnt/test.txt), file read (cat)

파일에 대한 생성 및 read/write 를 수행하는 명령들이 정상적으로 동작해야 한다. touch mnt/test.txt 를 통해 test.txt 라는 파일 생성시, 정상적으로 파일이 생성되어야 하고, echo aaa >> mnt/test.txt 를 통해 aaa 라는 문자를 test.txt 에 기록하고 cat 명령을 통해 read 할 수 있어야 하며, echo bbb >> test.txt 명령을 통해 파일에 append 연산 수행 시, test.txt 에 aaa bbb 라는 내용을 cat 명령어를 통해 read 할 수 있어야 한다. 또한 파일 삭제가 가능해야 한다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 3 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 15:06 ..
drwxr-xr-x 3 root       root       4096  3월 10 15:06 test1
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# touch mnt/test.txt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 3 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 15:06 ..
-rw-r--r-- 1 root       root          0  3월 10 15:14 test.txt
drwxr-xr-x 3 root       root       4096  3월 10 15:06 test1
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# cat mnt/test.txt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# echo aaa >> mnt/test.txt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# cat mnt/test.txt
aaa
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# echo bbb >> mnt/test.txt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# cat mnt/test.txt
aaa
bbb
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse#
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# echo ccc > mnt/test1/test2.txt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/test1/
합계 0
drwxr-xr-x 3 root root 4096  3월 10 15:06 .
drwxr-xr-x 3 root root 4096  3월 10 13:19 ..
-rw-r--r-- 1 root root    4  3월 10 15:16 test2.txt
drwxr-xr-x 2 root root 4096  3월 10 15:08 test3
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# rm -rf mnt/test.txt
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 3 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 15:06 ..
drwxr-xr-x 3 root       root       4096  3월 10 15:06 test1
```

iv. Step 4 :fiile rename (mv)

파일의 이름변경 및 위치를 변경하는 rename 연산이 정상적으로 동작하여야 한다.

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse#
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 3 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 15:06 ..
drwxr-xr-x 3 root       root       4096  3월 10 15:06 test1
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# mv mnt/test1/ mnt/rename1
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 3 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 15:06 ..
drwxr-xr-x 3 root       root       4096  3월 10 15:06 rename1
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# mv mnt/rename1/test2.txt mnt/
root@os-lecture:/home/os-lecture/DKU_OS_LAB/lab3_fuse# ls -al mnt/
합계 4
drwxr-xr-x 3 root       root       4096  3월 10 13:19 .
drwxr-xr-x 5 os-lecture os-lecture 4096  3월 10 15:06 ..
drwxr-xr-x 3 root       root       4096  3월 10 15:06 rename1
-rw-r--r-- 1 root       root          4  3월 10 15:16 test2.txt
```

8. 보너스 과제.

본 과제 lab3_fuse 는 ramdisk 에 대한 format 과 mount 과정이 제공되어 있다. 이 부분까지 구현 및 super block, inode, directory entry 을 자신이 직접 작성하여 과제를 수행하는 경우에 한해 bonus 점수를 부여한다.

9. 과제 제출.

i. 제출 사항.

- 구현한 lab3_fuse 의 압축 파일.
- 구현한 lab3_fuse 의 구현 사항에 대한 설명 및 어려웠던 점에 대한 레포트.

ii. 제출 방법.

✓ lab3_fuse 구현 제출

아래와 같은 directory 에서 tar 명령을 통해 구현한 lab3_fuse directory 에 대해 압축을 수행한 후 압축 결과 생기는 lab3_fuse_학번.tar 파일을 tooson9010@gmail.com 으로 제출한다.

- ***# tar cvf lab3_fuse_학번.tar lab3_fuse***

```
root@os-lecture:/home/os-lecture/DKU_OS_LAB# tar cvf lab3_fuse_32111860.tar lab3_fuse
lab3_fuse/
lab3_fuse/file.c
lab3_fuse/fuse_format_main.c
lab3_fuse/Makefile
lab3_fuse/aaa
lab3_fuse/dir.c
lab3_fuse/tags
lab3_fuse/mount.c
lab3_fuse/include/
lab3_fuse/include/lab3_fs_types.h
lab3_fuse/fuse_main.c
lab3_fuse/bmap.c
lab3_fuse/mnt/
lab3_fuse/example/
lab3_fuse/example/Makefile
lab3_fuse/example/lab3_example.c
lab3_fuse/example/mnt/
lab3_fuse/fsformat.c
lab3_fuse/inode.c
root@os-lecture:/home/os-lecture/DKU_OS_LAB# ls -al
합계 160
drwxrwxr-x  5 os-lecture os-lecture   4096  3월 10 15:25 .
drwxr-xr-x 22 os-lecture os-lecture   4096  3월 10 13:16 ..
drwxr-xr-x  3 os-lecture os-lecture   4096  3월 10 04:49 lab1_sched
drwxr-xr-x  3 os-lecture os-lecture   4096  3월 10 04:31 lab2_sync
drwxr-xr-x  5 os-lecture os-lecture   4096  3월 10 15:24 lab3_fuse
-rw-r--r--  1 root       root       143360  3월 10 15:25 lab3_fuse_32111860.tar
```

✓ 레포트 제출

미디어 센터 506 호로 제출.

D.Reference

i. **file, directory, 관련 기본 연산**

http://pages.cs.wisc.edu/~remzi/OSTEP/file-intro.pdf

ii. **UNIX file system 의 연산 구조**

http://pages.cs.wisc.edu/~remzi/OSTEP/file-implementation.pdf

iii. **file mode 관련 사항**

http://www.gnu.org/software/libc/manual/html_node/Testing-File-Type.html#Testing-File-Type

iv. **linux file permission 관련 사항.**

https://www.linux.com/learn/understanding-linux-file-permissions

v. **file, directory 의 link count(directory 생성시 link count 가 2 인 이유)**

vi. **리눅스에서의 C Error**

http://www.virtsync.com/c-error-codes-include-errno

vii. **FUSE API**

https://github.com/libfuse/libfuse/blob/579c3b03f57856e369fd6db2226b77aba63b59ff/include/fuse.h#L102-L577