System Programming

Week 5: Ctags

Yeonjin Noh

Updated: 2016-09-27

f2fs_ctags

nyg0813@gmail.com

http://esos.hanyang.ac.kr

Esos Lab. Hanyang University

Table of contents

Ctags란?

프로그래밍 소스코드에서 함수의 정의, 매크로 선언들과 소스 코드의 태그들의 **Database(tags file)**를 생성하는 Unix 명령어

다시말해,

소스코드 내부의 함수 및 변수의 위치를 쉽게 인식 할 수 있도록
인덱스(index) 를 만드는 명령어라 정의할 수 있다.

Ctags의 장점

- 함수의 검색 및 함수가 정의된 곳으로 이동이 가능
- 커널 코드와 같이 큰 규모의 프로젝트의 소스를 분석할 때 유용
- Vim 및 emacs와 호환 가능

CTAGS 환경설정 및 사용법

Ctags 설치

먼저 Ctags를 설치하기 전,

```
$ ctags -help
```

명령어로 Ctags가 설치되었는지 확인하고, 없다면 아래와 같이 명령어를 입력한다.

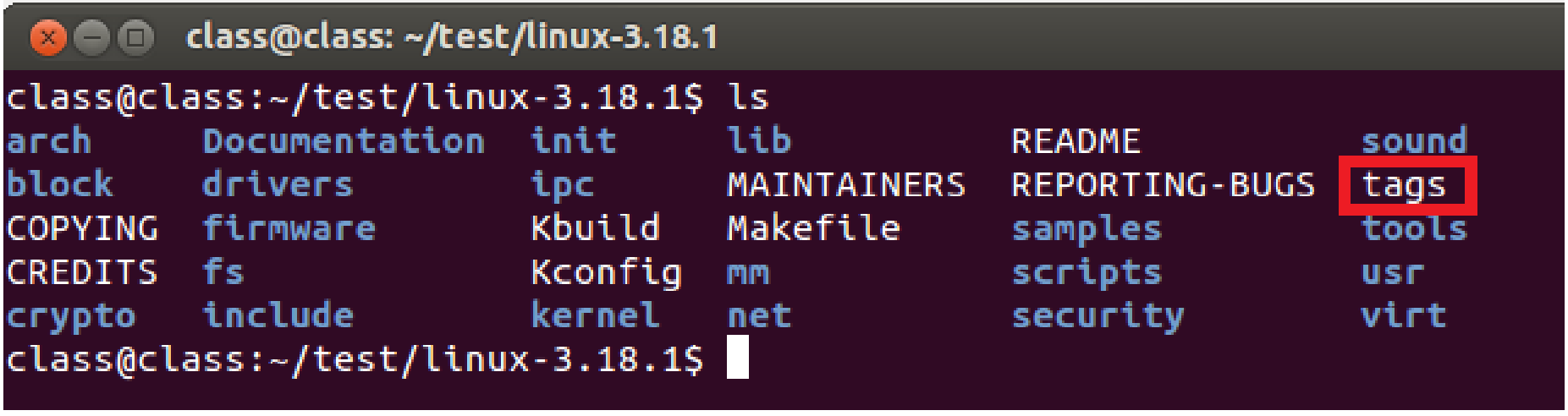
```
$ sudo apt-get install ctags
```

tags 파일 생성(Skip)

Ctags를 사용하기 위해서는 'tags 파일'을 생성할 필요가 있다.
'tags 파일'을 생성하기 위해서는 분석을 원하는 폴더로 이동한 후 아래의 명령을 입력한다.

```
$ catgs -R
```

생성된 tags 파일은 ls 명령어를 통해 다음과 같이 확인할 수 있다.

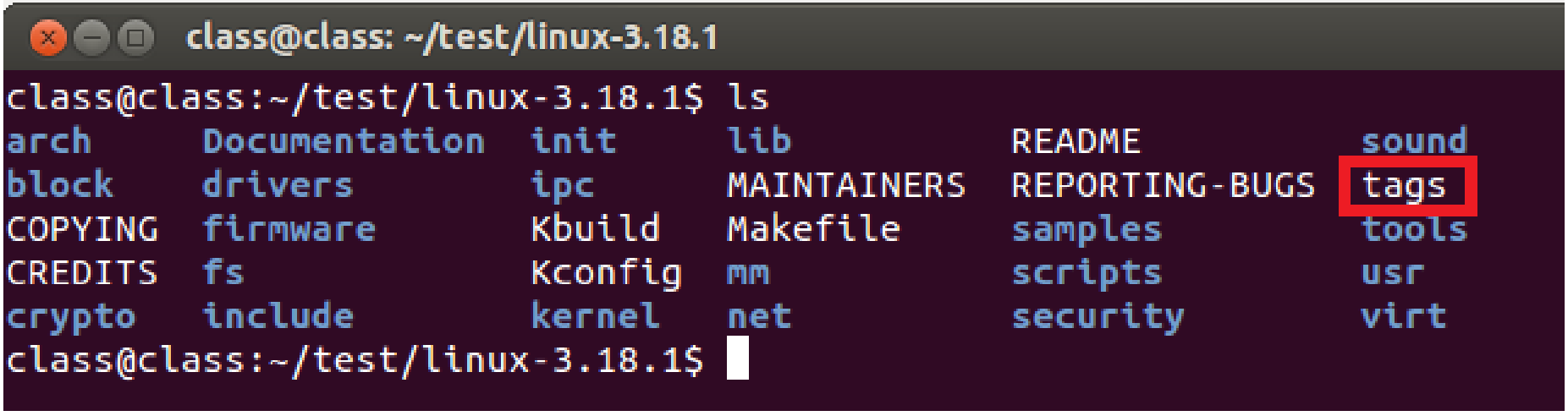


tags 파일 복사(Skip)

Ctags에서 생성한 tags 파일은 절대경로가 아닌 상대경로이기 때문에,
기존에 생성된 tags 파일을 복사하거나 이동시켜서 사용할 수 있다.
아래의 명령어를 통해 'tags파일'을 커널코드로 복사한다.

```
$ cp ../codes/tags /'linux소스 코드 경로'/
```

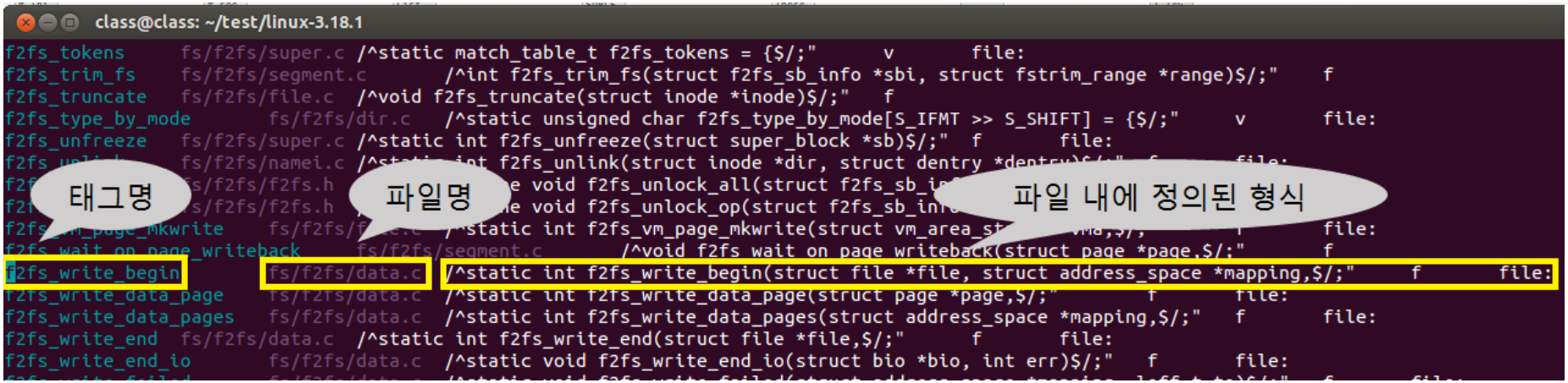
복사된 tags 파일은 ls 명령어를 통해 다음과 같이 확인할 수 있다.



tags 파일 열기

```
$ vi tags
```

위 명령어를 통해 tags 파일을 열어보면, tags 파일은 아래 그림과 같은 구성을 가진다.



이를 통해 각 함수, 변수에 따라 태그가 생성된 것을 알 수 있다.

Ctags 사용법

태그 파일을 연 상태에서 vi를 command line 모드로 변경하고,

```
:tj ‘함수(tag명)’
```

명령어를 입력하여 원하는 함수의 위치로 이동할 수 있으며, :tj를 입력하지 않고도 함수나 변수 이름에서 **‘Ctrl +]’**를 입력하면 원하는 함수의 위치로 이동할 수 있다.

```
:po
```

명령어를 통해 이전 태그로 돌아갈 수 있으며, :po 대신 **‘Ctrl + t’**를 사용할 수도 있다.

Ctags 사용법

또한, **태그 파일을 연 상태**에서,

```
:stj '함수(tag명)'
```

명령어를 입력하면 아래와 같이 분할된 창에서 태그를 볼수 있다.

```
class@class: ~/test/linux-3.18.1
#define PAGE_CACHE_SIZE         PAGE_SIZE
#define PAGE_CACHE_MASK         PAGE_MASK
#define PAGE_CACHE_ALIGN(addr)  (((addr)+PAGE_CACHE_SIZE-1)&PAGE_CACHE_MASK)

#define page_cache_get(page)            get_page(page)
#define page_cache_release(page)        put_page(page)
void release_pages(struct page **pages, int nr, bool cold);

/*
 * speculatively take a reference to a page.
 * If the page is free ( count == 0), then  count is untouched, and 0
include/linux/pagemap.h
f2fs_unlock_all fs/f2fs/f2fs.h  /^static inline void f2fs_unlock_all(struct f2fs_sb_info *sbi)$/;"      f
f2fs_unlock_op  fs/f2fs/f2fs.h  /^static inline void f2fs_unlock_op(struct f2fs_sb_info *sbi)$/;"       f
f2fs_vm_page_mkwrite    fs/f2fs/file.c  /^static int f2fs_vm_page_mkwrite(struct vm_area_struct *vma,$/;"       f       file:
f2fs_wait_on_page_writeback     fs/f2fs/segment.c       /^void f2fs_wait_on_page_writeback(struct page *page,$/;"       f
f2fs_write_begin        fs/f2fs/data.c  /^static int f2fs_write_begin(struct file *file, struct address_space *mapping,$/;"
f2fs_write_data_page    fs/f2fs/data.c  /^static int f2fs_write_data_page(struct page *page,$/;"        f       file:
f2fs_write_data_pages   fs/f2fs/data.c  /^static int f2fs_write_data_pages(struct address_space *mapping,$/;"   f       file:
f2fs_write_end  fs/f2fs/data.c  /^static int f2fs_write_end(struct file *file,$/;"      f       file:
f2fs_write_end_io       fs/f2fs/data.c  /^static void f2fs_write_end_io(struct bio *bio, int err)$/;"   f       file:
f2fs_write_failed       fs/f2fs/data.c  /^static void f2fs_write_failed(struct address_space *mapping, loff_t to)$/;"   f
tags
"include/linux/pagemap.h" 674L, 19878C
```

Vim 연동

Vim에서도 Ctags를 동작시키기 위해서는 아래의 과정이 필요하다.

```
$ vi  /.vimrc
```

먼저, vimrc 파일을 열고 아래와 같이 tags 파일의 경로를 추가한다.

```
$ set tags=/'source코드 경로'/tags
```

위 과정이 완료되면 Vim에서도 Ctags를 사용할 수 있다.

Ctags 명령어

:ta [tags] or Ctrl +]	Tag가 정의된 위치를 나열하고 선택한 위치로 점프, 현재 위치는 stack에 push 된다.
:ts [tags] or :tj [tags]	
:po or Ctrl + t	Stack 을 pop 하고 그 위치로 점프
:sts [tags]	[tags]가 정의된 위치를 나열하고 선택한 위치로 창을 수평 분할하여 새로 생성된 창에 표시.
:stj [tags]	
:tn	tj나 ts로 점프했을 때 다음 tag로 점프
:tp	tj나 ts로 점프했을 때 이전 tag로 점프
:tr	tj나 ts로 점프했을 때 처음 tag로 점프
:tl	tj나 ts로 점프했을 때 마지막 tag로 점프
:pts [tags]	[tag]가 정의된 위치를 나열하고 선택한 위치로 창을 수평 분할하여 새로 생성된 창에 표시하지만 현재 위치에 표시
:ptj [tag]	미리보기 윈도우에 tag가 정의된 형식을 보임
:ptn	ptj나 pts로 점프했을 때 다음 tag로 점프
:ptp	ptj나 pts로 점프했을 때 이전 tag로 점프
:ptr	ptj나 pts로 점프했을 때 처음 tag로 점프
:ptl	ptj나 pts로 점프했을 때 마지막 tag로 점프

F2FS 커널수정

f2fs-tool 다운로드 및 설치

f2fs-tools는 리눅스의 파티션을 F2FS 형식으로 포맷할 수 있게 하며,

```
$sudo apt-get install f2fs-tools
```

위 명령어를 통해 설치할 수 있다, 만약 위 명령어대로 설치가 안될 경우

```
$ git clone https://git.kernel.org/pub/scm/linux/kernel/git/jaegeuk/
f2fs-tools.git
$ sudo apt-get install uuid-dev pkg-config autoconf libtool libselinux1-dev
$ autoreconf --install
$ ./configure
$ make
```

명령어를 차례로 입력하여 설치한다.

f2fs 커널 수정

먼저, cd명령어를 통해 linux-3.18.1 폴더로 이동한다.

```
$ cd /'경로'/linux-3.18.1/
```

그 후, 아래 명령어를 통해 미리 컴파일된 '**원본**' f2fs.ko 파일을 복사한다.

```
$ cp fs/f2fs/f2fs.ko ../modules/f2fs_ori.ko
```

복사가 완료되면 vi를 실행해 command line 모드로 진입한다.

```
$ vi
```

f2fs 커널 수정 - 전역 변수 선언

아래 명령어처럼 Ctags를 이용해 함수가 정의된 곳(segement.h)으로 이동한다.

```
:tj nr_pages_to_write
```

nr_pages_to_write 함수 상단에 아래와 같이 전역 변수를 선언한다.

```
////////////////////////////////////////////////////////////////////////////
/* 실습용 코드 : 전역 변수들을 선언해준다 */
extern unsigned long long len_user_data;
extern unsigned long long len_fs_write;
////////////////////////////////////////////////////////////////////////////
static inline long nr_pages_to_write(struct f2fs_sb_info *sbi, int type,
                                     struct writeback_control *wbc)
{
   ...
```

f2fs 커널 수정 - 전역 변수 초기화

아래 명령어처럼 Ctags를 이용해 변수가 정의된 곳(statement.c)으로 이동한다.

```
:tj inmem_entry_slab
```

inmem_entry_slab 변수 아래에서 선언한 변수의 값을 초기화해준다.

```
...
static struct kmem_cache *inmem_entry_slab;
////////////////////////////////////////////////////////////////////////
/* 실습용 코드 : 선언한 전역 변수들을 초기화 한다 */
unsigned long long len_user_data = 0;
     // User에서 File System에 쓰여진 데이터의 양
unsigned long long len_fs_write = 0;
     // File System에서 Block Layer에 쓰여진 데이터의 양
////////////////////////////////////////////////////////////////////////
/*
 * __reverse_ffs is copied from include/asm-generic/bitops/__ffs.h since
 * MSB and LSB are reversed in a byte by f2fs_set_bit.
 */
static inline unsigned long __reverse_ffs(unsigned long word)
```

f2fs 커널 수정 - User to FS

아래 명령어처럼 Ctags를 이용해 함수가 정의된 곳으로 이동한다.

```
:tj f2fs_write_begin
```

f2fs_write_begin 함수 내부에 아래와 같이 코드를 입력한다.

```
static int f2fs_write_begin(struct file *file, struct address_space *mapping,
                loff_t pos, unsigned len, unsigned flags,
                struct page **pagep, void **fsdata)
{
        ...
        trace_f2fs_write_begin(inode, pos, len, flags);
//////////////////////////////////////////////////////////////////////////
/* 실습용 코드 : User영역에서 File system에 쓰는 Data의 크기를 기록 */
        len_user_data += len;
//////////////////////////////////////////////////////////////////////////
        f2fs_balance_fs(sbi);
        ...
}
```

f2fs 커널 수정 - FS to Block Layer 1

아래 명령어처럼 Ctags를 이용해 함수가 정의된 곳으로 이동한다.

```
:tj __submit_merged_bio
```

__submit_merged_bio 함수 내부에 아래와 같이 코드를 입력한다.

```
static void __submit_merged_bio(struct f2fs_bio_info *io)
{
        ...
////////////////////////////////////////////////////////////////////////
/* 실습용 코드 : File system에서 Block Layer에 쓰는 Data의 크기를 기록 */
        if(!is_read_io(fio->rw)){
                len_fs_write += io->bio->bi_vcnt * 4096;
        }
////////////////////////////////////////////////////////////////////////
        io->bio = NULL;
}
```

f2fs 커널 수정 - FS to Block Layer 2

아래 명령어처럼 Ctags를 이용해 함수가 정의된 곳으로 이동한다.

```
:tj f2fs_submit_page_bio
```

f2fs_submit_page_bio 함수 내부에 아래와 같이 코드를 입력한다.

```
int f2fs_submit_page_bio(struct f2fs_sb_info *sbi, struct page *page,
                                        block_t blk_addr, int rw)
{
        ...
////////////////////////////////////////////////////////////////////////////
/* 실습용 코드 : File system에서 Block Layer에 쓰는 Data의 크기를 기록 */
        if(!is_read_io(fio->rw)){
                len_fs_write += io->bio->bi_vcnt * 4096;
        }
////////////////////////////////////////////////////////////////////////////
        submit_bio(rw, bio);
        return 0;
}
```

f2fs 커널 수정 - Data의 양 표시

아래 명령어처럼 Ctags를 이용해 함수가 정의된 곳으로 이동한다.

```
:tj stat_show
```

stat_show 함수 내부에 아래와 같이 코드를 입력한다.

```
static int stat_show(struct seq_file *s, void *v)
{
        ...
        seq_printf(s, "(OverProv:%d Resv:%d)]\n\n",
                        si->overp_segs, si->rsvd_segs);
////////////////////////////////////////////////////////////////////////
/* 실습용 코드 : User영역, File System, Block Layer 간 쓰여진 data의 양 표시 */
        seq_printf(s, "Buffered Write Information :\n");
        seq_printf(s, " - User to FS: %llu bytes\n", len_user_data);
        seq_printf(s, " - FS to Block Layer: %llu bytes\n\n", len_fs_write);
////////////////////////////////////////////////////////////////////////
        ...
}
```

f2fs 모듈 컴파일

cd명령어를 통해 다시 linux-3.18.1 폴더로 이동한다.

```
$ cd /'경로'/linux-3.18.1/
```

그 후, 아래 명령어를 통해 수정된 f2fs 커널의 모듈을 컴파일한다.

```
$ make modules
```

컴파일이 완료되면 '**새로운**' f2fs.ko 파일을 복사한다.

```
$ cp fs/f2fs/f2fs.ko ../modules/f2fs_new.ko
```

F2FS 실습

f2fs 모듈 적재(Load)

rmmod명령어를 통해 f2fs 모듈을 제거한다.

```
$ sudo rmmod f2fs
```

만약 F2FS 모듈이 없는 경우에는 아래와 같이

```
'ERROR: Module f2fs does not exist in /proc/modules'
```

경고문이 출력된다.
모듈이 제거 후 f2fs_ori.ko, f2fs_new.ko파일이 있는 폴더로 이동 한다.

```
$ cd ../modules
```

모듈은 아래와 같이 insmod명령어로 적재(Load) 할 수 있다.

```
$ sudo insmod f2fs_new.ko
```

파티션 F2FS 포맷

cd명령어를 통해 codes 폴더로 이동한다.

```
$ cd ../codes
```

codes폴더 내부의 mkfs.f2fs 파일로 /dev/sdb1 파티션을 포맷한다.

```
$ sudo ./mkfs.f2fs l mnt /dev/sdb1
```

성공적으로 포맷 시 아래 문구가 출력된다.

```
Info: format successful
```

F2FS 파일시스템 마운트

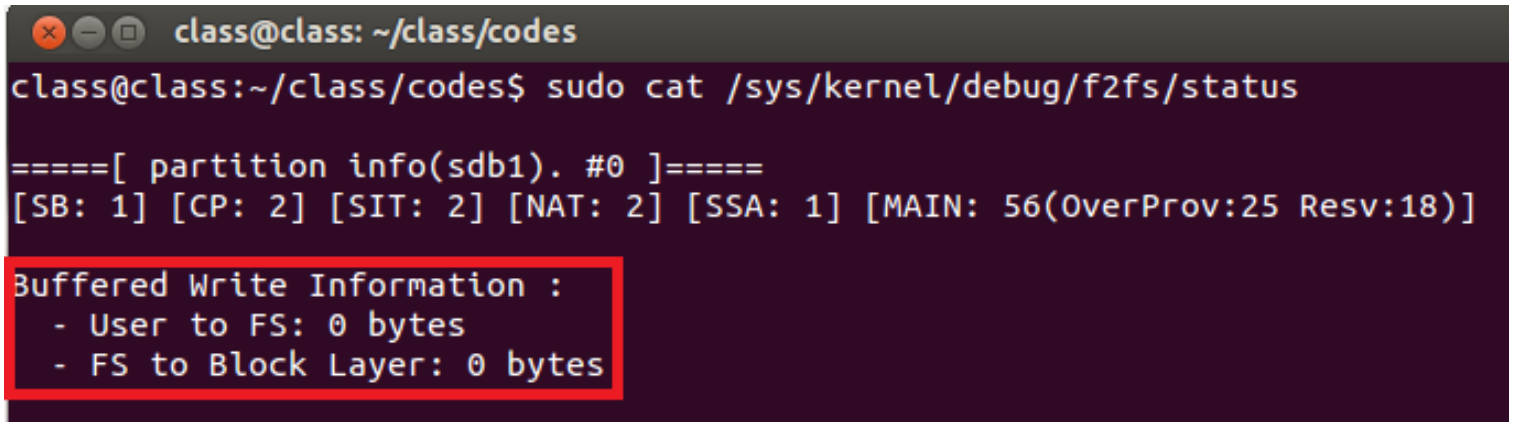
F2FS으로 포맷된 파티션을 다음과 같은 명령어로 마운트시킨다.

```
$ sudo mount -t f2fs /dev/sdb1 ./mnt/
```

폴더를 마운트 한 후 F2FS의 상태를 보여주는

```
$ sudo cat /sys/kernel/debug/f2fs/status
```

명령어를 입력하여 아래와 같은 화면이 출력되면,



F2FS 환경에서 File System 및 Block Layer에 쓰여지는 Data의 양을 확인할 수 있다.

F2FS 파일시스템 실습 1

F2FS의 상태를 보여주는 아래 명령어를 이용하여

```
$ sudo cat /sys/kernel/debug/f2fs/status
```

F2FS 마운트 폴더인 mnt 내부에서 다음과 같은 명령어들을 실행하며 Data 양의 변화를 살펴보자.

```
$ sudo mkdir test
$ sudo cat /proc/modules >> a.txt
$ sudo rm -r test
$ sudo cp a.txt b.txt
$ sudo rm a.txt
$ sudo rm b.txt
```

F2FS 파일시스템 실습 2

F2FS의 상태를 보여주는 아래 명령어를 이용하여

```
$ sudo cat /sys/kernel/debug/f2fs/status
```

F2FS 마운트 폴더인 mnt 내부에 테스트 프로그램인 'sysp'를 실행한 후 Data 양의 변화를 살펴보자.

```
$ cd mnt
$ cp ../codes/sysp .
$ sudo ./sysp
```

sysp 테스트 프로그램

sysp 테스트 프로그램은 f_0.txt부터 f_9.txt까지 총 10개의 3Kbyte 파일을 Buffered write한 후,마지막 f_9.txt 파일에 대해서만 fsync() 함수를 실행하는 프로그램이다.