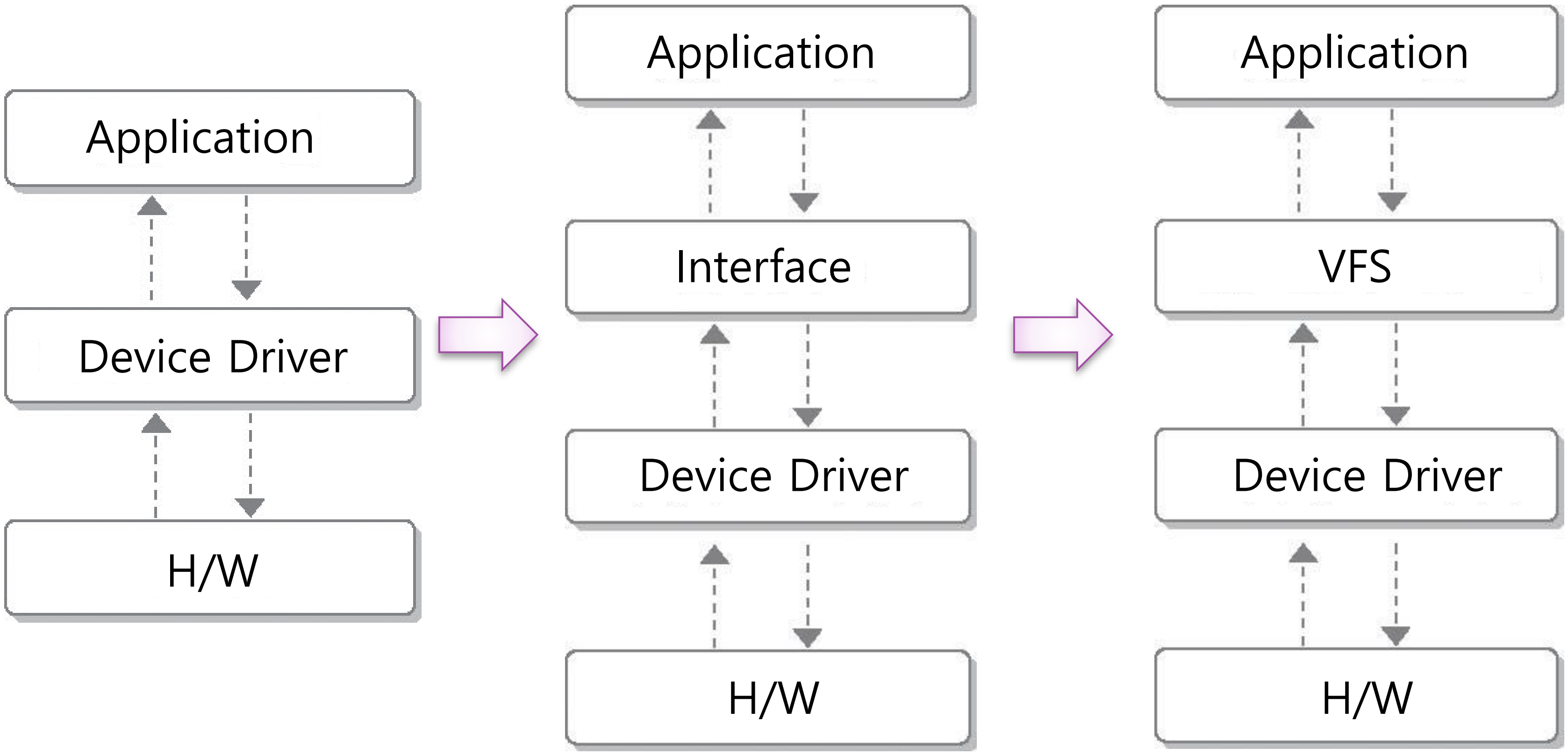
Character Device Programming

Agenda

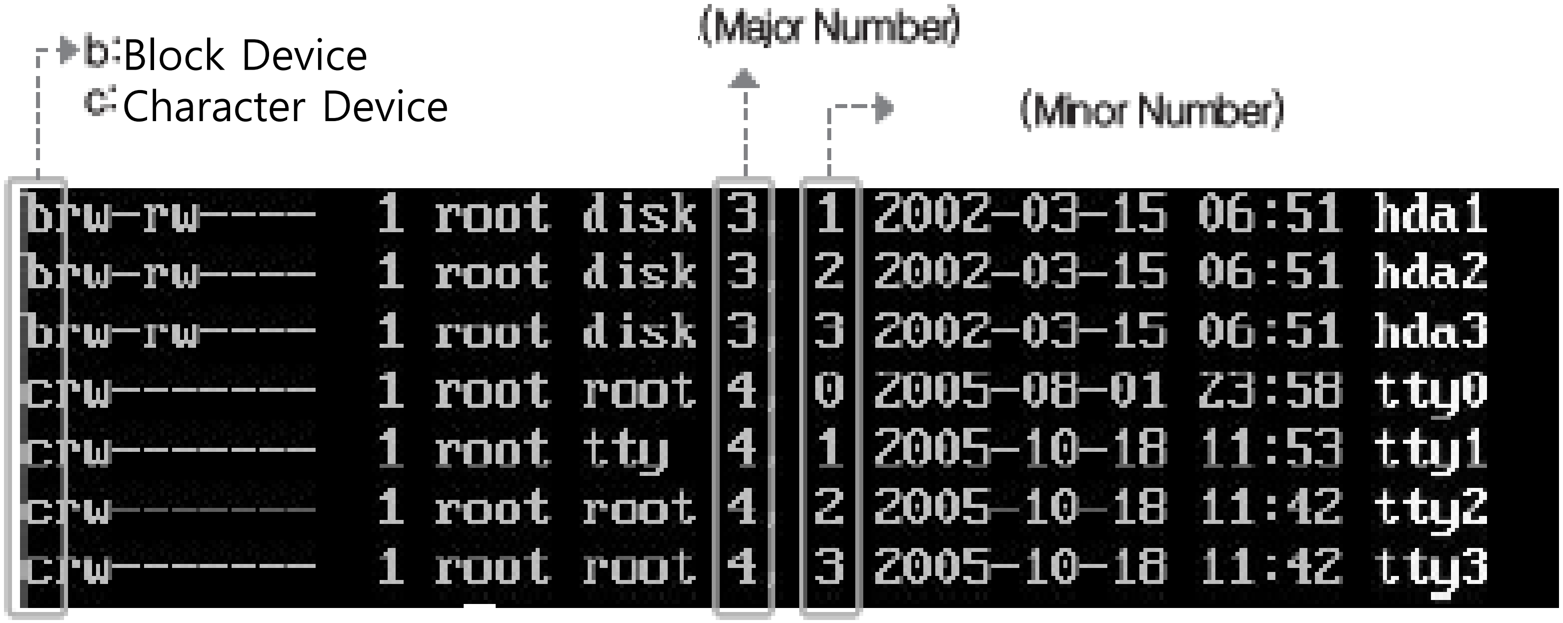
- Device Driver is …
- Kernel structure related to Device Driver
- Write a code of a character device driver
- kmalloc()
- IOCTL Programming

Device Driver

- Abstraction for HW
- In Linux, Virtual File Systems (VFS) is used.



Device File



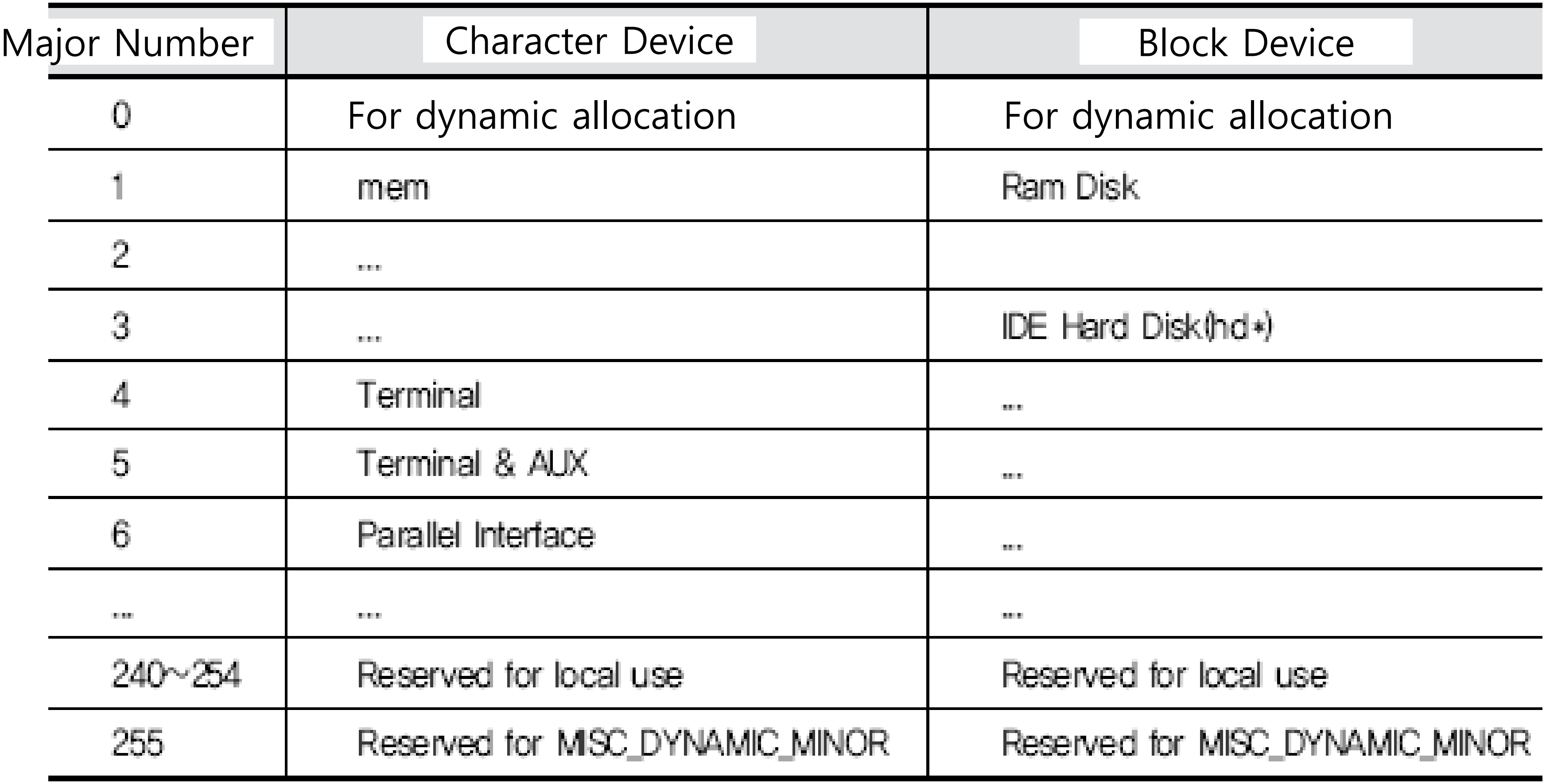
b:Block Device
c:Character Device

(Major Number)

(Minor Number)

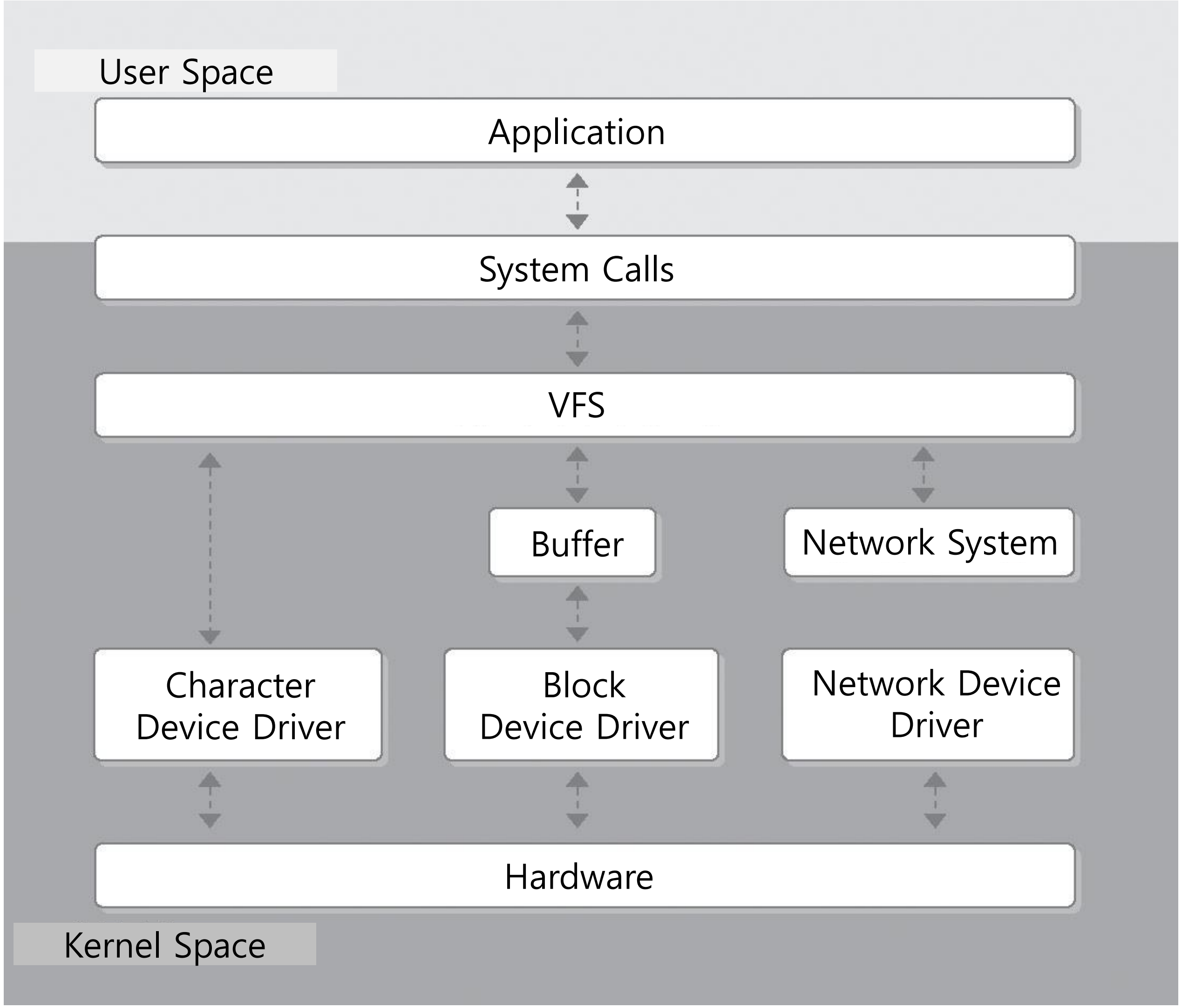
```
brw-rw----   1 root disk 3  1 2002-03-15 06:51 hda1
brw-rw----   1 root disk 3  2 2002-03-15 06:51 hda2
brw-rw----   1 root disk 3  3 2002-03-15 06:51 hda3
crw-------   1 root root 4  0 2005-08-01 23:58 tty0
crw-------   1 root tty  4  1 2005-10-18 11:53 tty1
crw-------   1 root root 4  2 2005-10-18 11:42 tty2
crw-------   1 root root 4  3 2005-10-18 11:42 tty3
```

Example of Major Numbers

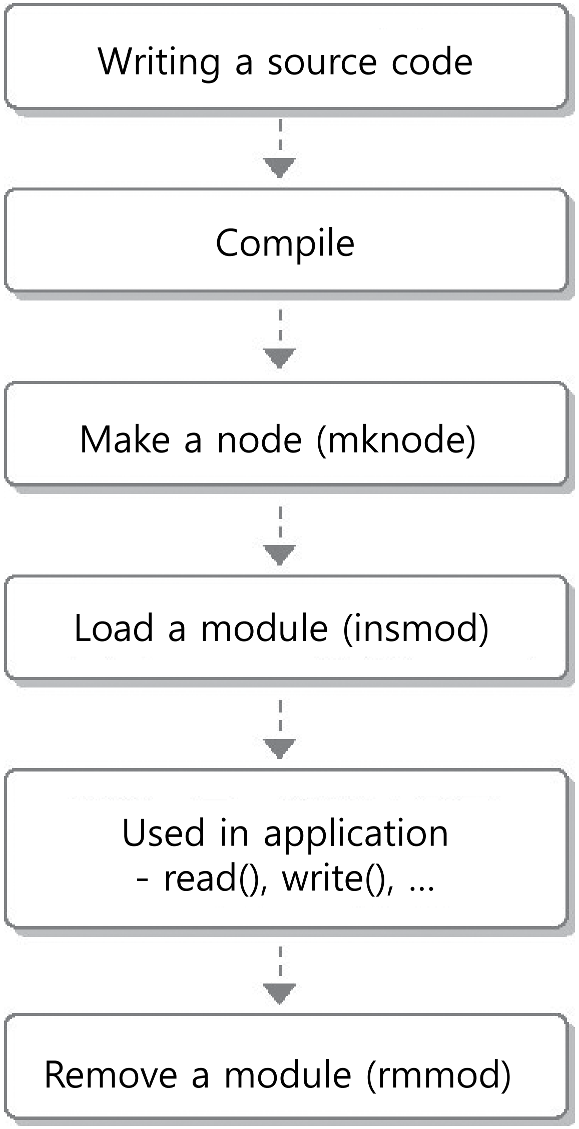


Major Number	Character Device	Block Device
0	For dynamic allocation	For dynamic allocation
1	mem	Ram Disk
2	...	
3	...	IDE Hard Disk(hd*)
4	Terminal	...
5	Terminal & AUX	...
6	Parallel Interface	...
...	...	...
240~254	Reserved for local use	Reserved for local use
255	Reserved for MISC_DYNAMIC_MINOR	Reserved for MISC_DYNAMIC_MINOR

Device Driver in Linux



Development Process



Writing a source code

Compile

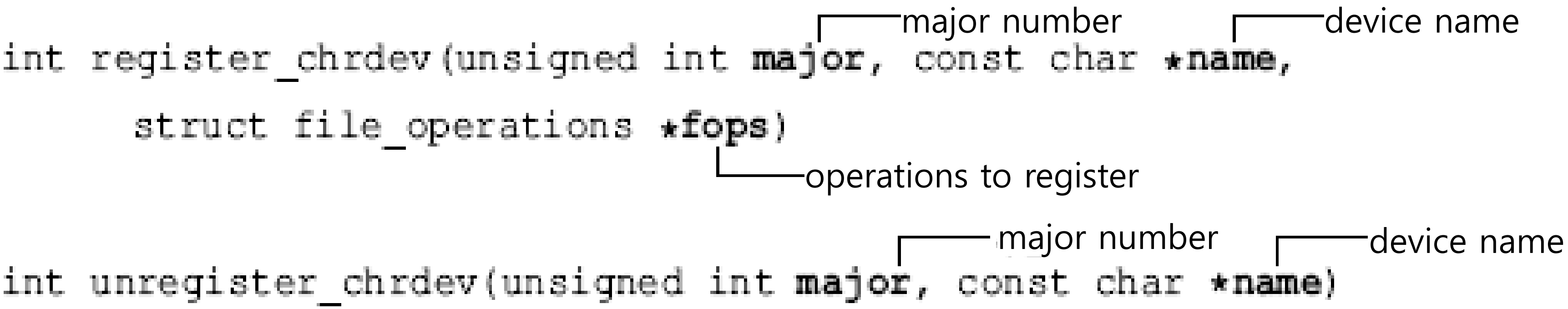
Make a node (mknode)

Load a module (insmod)

Used in application
- read(), write(), …

Remove a module (rmmod)

(Un)Register Character Device Driver



```
int register_chrdev(unsigned int major, const char *name,
      struct file_operations *fops)
```

major number

device name

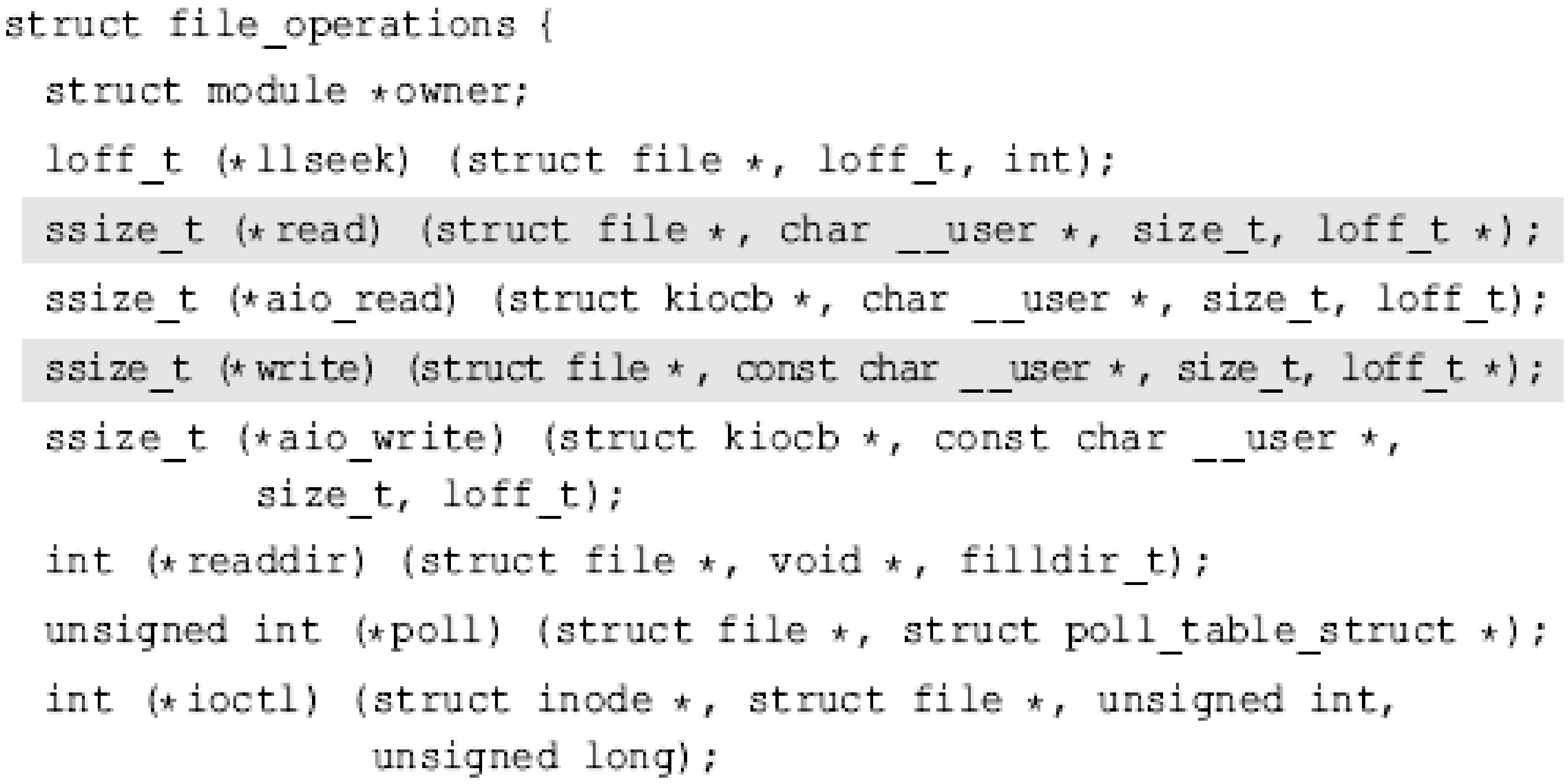
operations to register

```
int unregister_chrdev(unsigned int major, const char *name)
```

major number

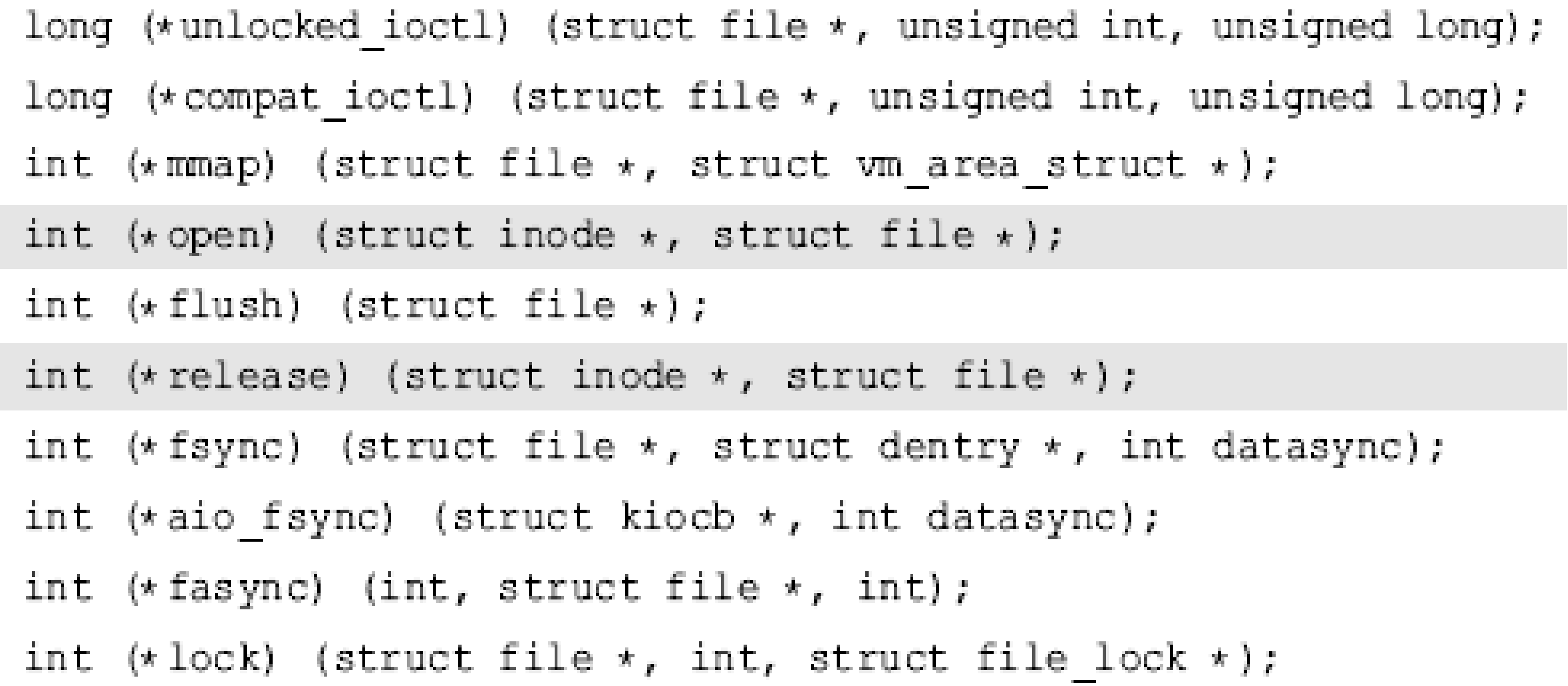
device name

file_operations Structure-1



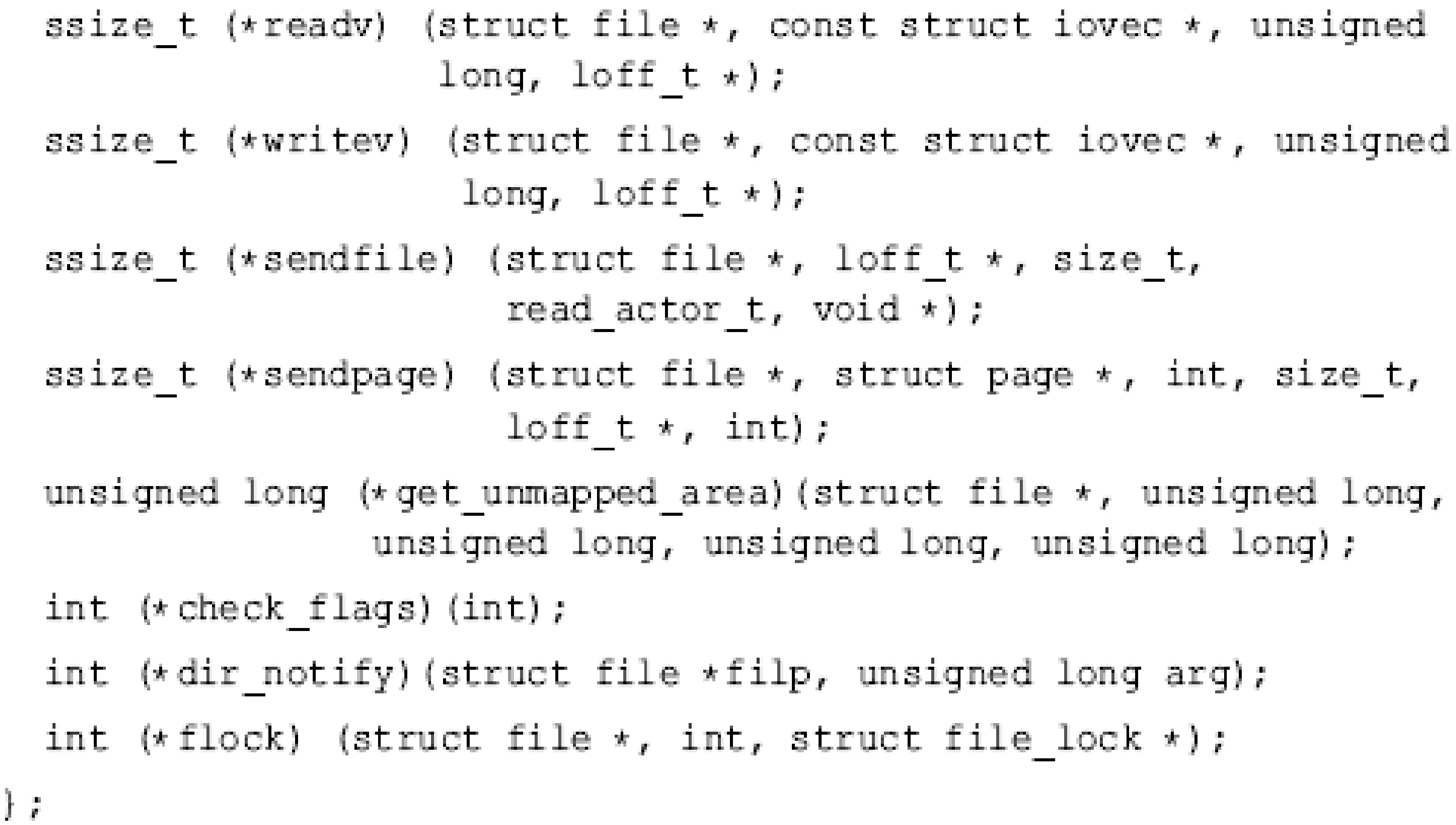
```
struct file_operations {
  struct module *owner;
  loff_t (*llseek) (struct file *, loff_t, int);
  ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
  ssize_t (*aio_read) (struct kiocb *, char __user *, size_t, loff_t);
  ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
  ssize_t (*aio_write) (struct kiocb *, const char __user *,
           size_t, loff_t);
  int (*readdir) (struct file *, void *, filldir_t);
  unsigned int (*poll) (struct file *, struct poll_table_struct *);
  int (*ioctl) (struct inode *, struct file *, unsigned int,
                unsigned long);
```

file_operations Structure-2



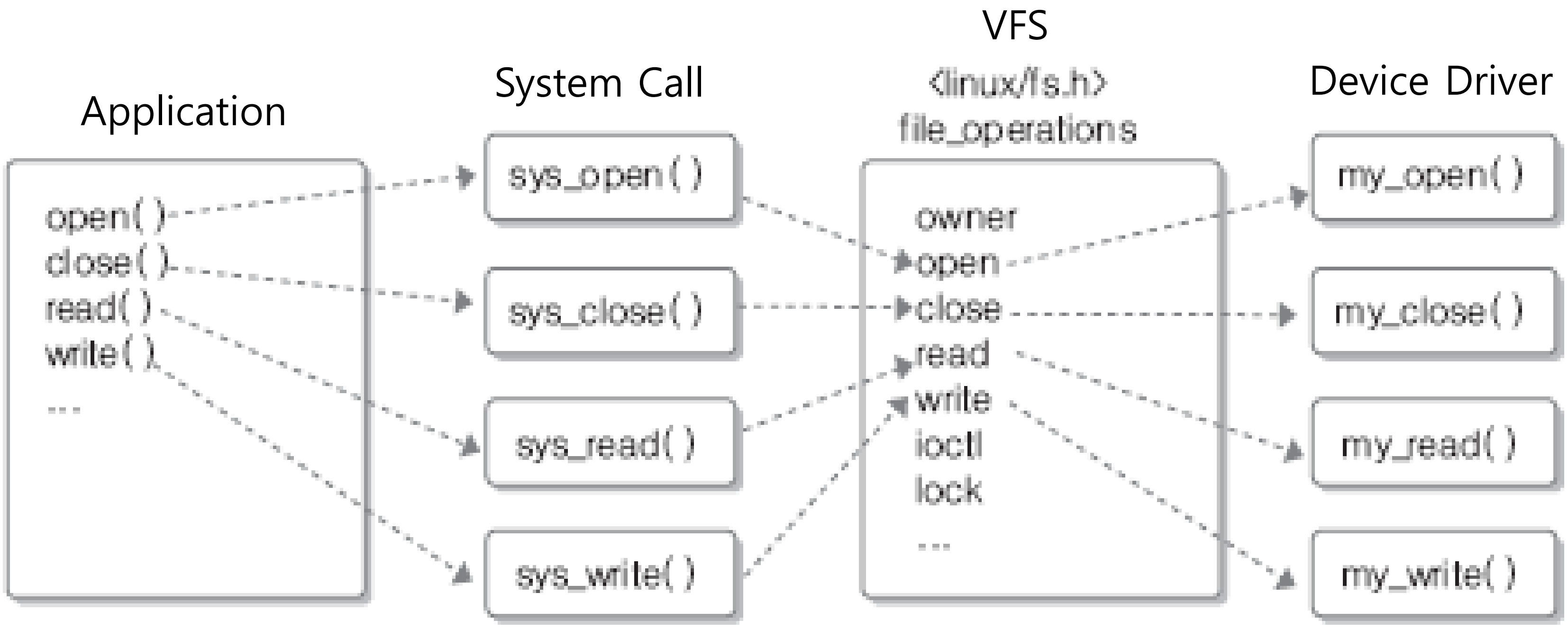
```
long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
int (*mmap) (struct file *, struct vm_area_struct *);
int (*open) (struct inode *, struct file *);
int (*flush) (struct file *);
int (*release) (struct inode *, struct file *);
int (*fsync) (struct file *, struct dentry *, int datasync);
int (*aio_fsync) (struct kiocb *, int datasync);
int (*fasync) (int, struct file *, int);
int (*lock) (struct file *, int, struct file_lock *);
```

file_operations Structure-3



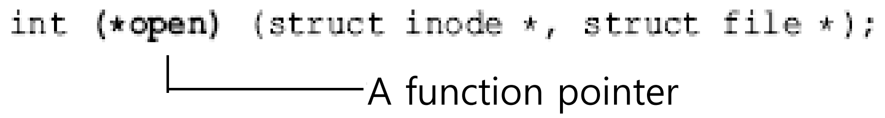
```
  ssize_t (*readv) (struct file *, const struct iovec *, unsigned
                     long, loff_t *);
  ssize_t (*writev) (struct file *, const struct iovec *, unsigned
                      long, loff_t *);
  ssize_t (*sendfile) (struct file *, loff_t *, size_t,
                        read_actor_t, void *);
  ssize_t (*sendpage) (struct file *, struct page *, int, size_t,
                        loff_t *, int);
  unsigned long (*get_unmapped_area)(struct file *, unsigned long,
                 unsigned long, unsigned long, unsigned long);
  int (*check_flags)(int);
  int (*dir_notify)(struct file *filp, unsigned long arg);
  int (*flock) (struct file *, int, struct file_lock *);
};
```

Function Calls



open()

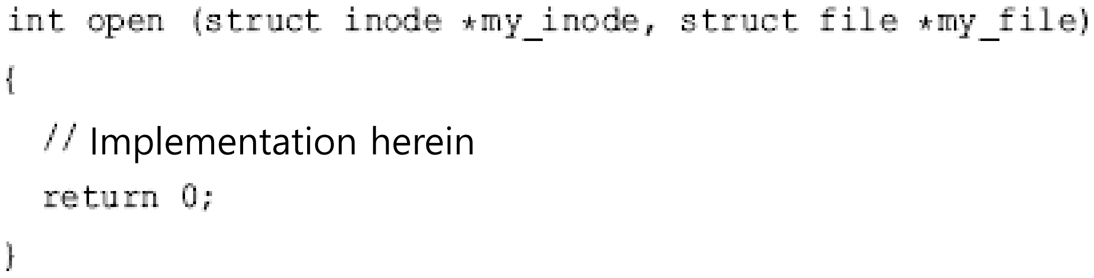
- Function Prototype



```
int (*open) (struct inode *, struct file *);
```

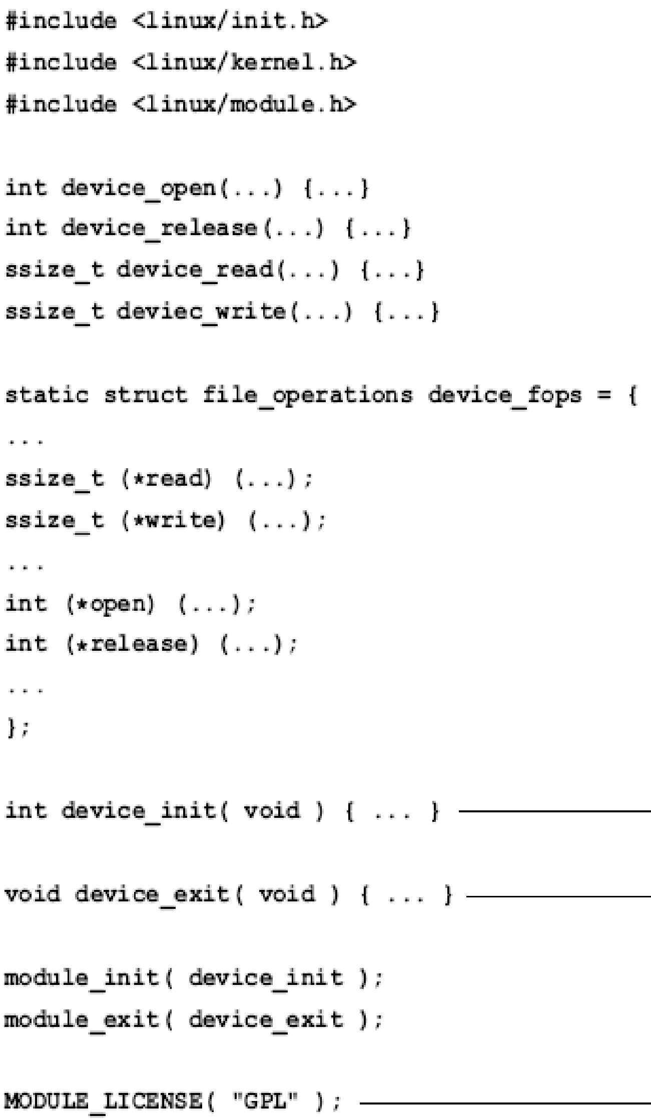
A function pointer

- Function Implementation



```
int open (struct inode *my_inode, struct file *my_file)
{
  // Implementation herein
  return 0;
}
```

Structure of Character Device Driver



```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>

int device_open(...) {...}
int device_release(...) {...}
ssize_t device_read(...) {...}
ssize_t deviec_write(...) {...}

static struct file_operations device_fops = {
...
ssize_t (*read) (...);
ssize_t (*write) (...);
...
int (*open) (...);
int (*release) (...);
...
};

int device_init( void ) { ... }

void device_exit( void ) { ... }

module_init( device_init );
module_exit( device_exit );

MODULE_LICENSE( "GPL" );
```

Header files

Implemented functions

File operations

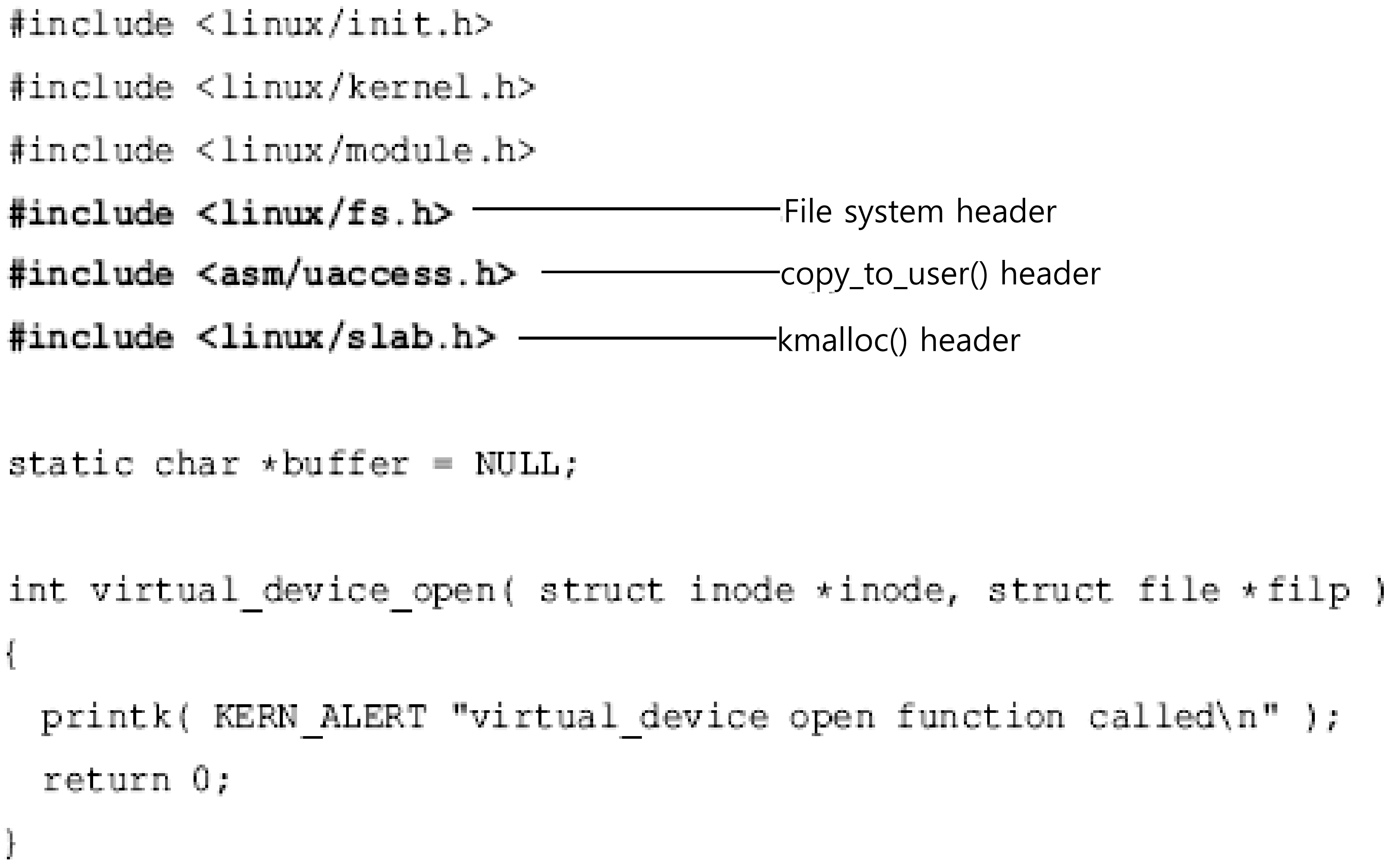
For module initialization

For module cleanup

Register initialization/cleanup functions

Licence

Device Driver Ex-1



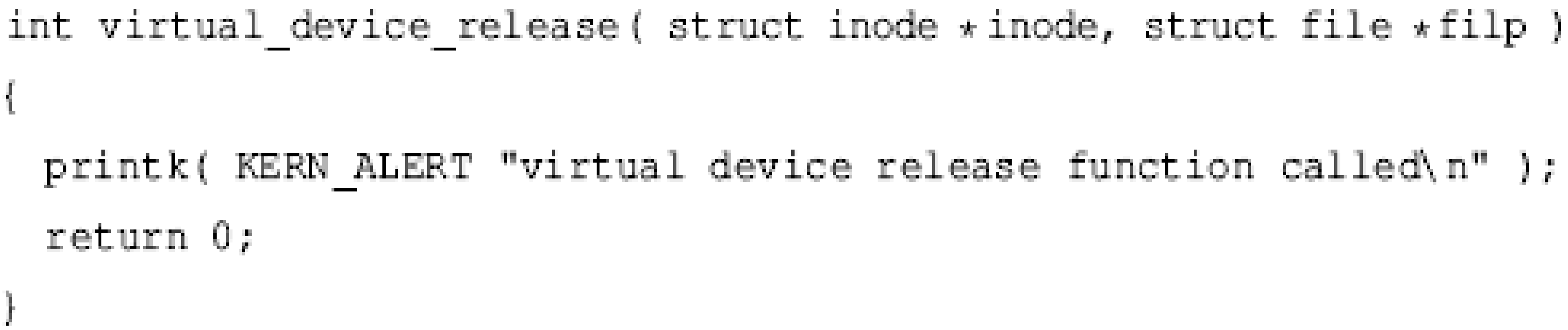
```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <asm/uaccess.h>
#include <linux/slab.h>

static char *buffer = NULL;

int virtual_device_open( struct inode *inode, struct file *filp )
{
  printk( KERN_ALERT "virtual_device open function called\n" );
  return 0;
}
```

File system header

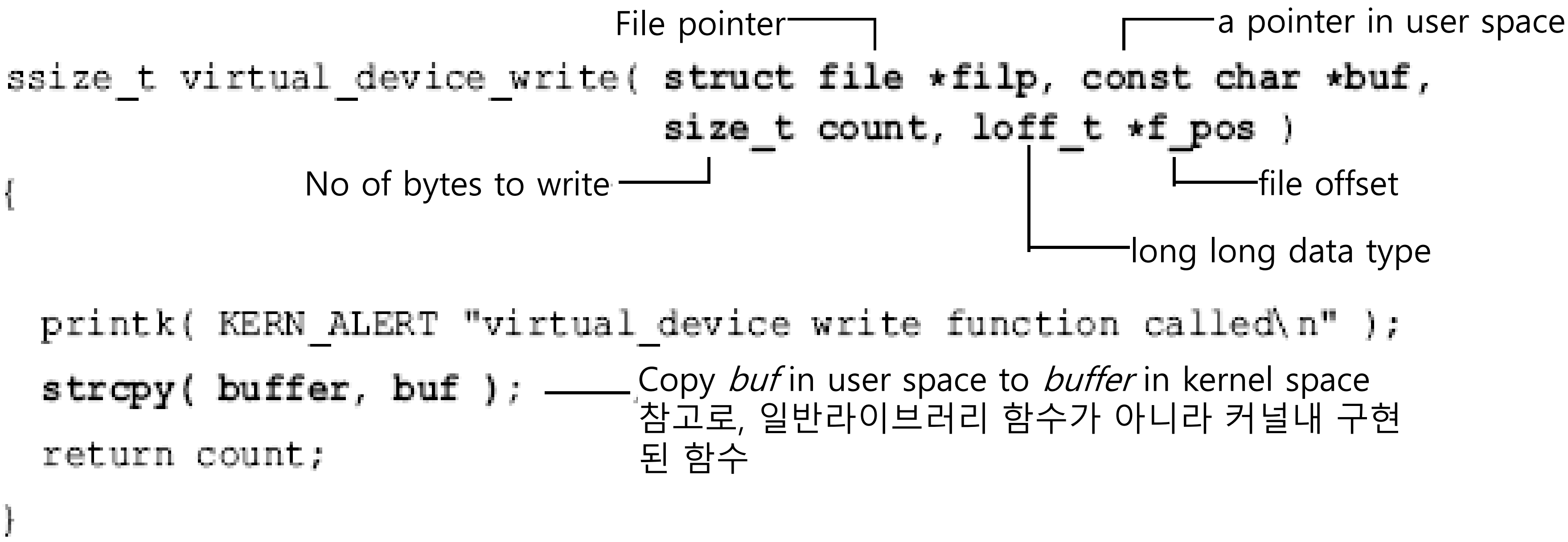
copy_to_user() header

kmalloc() header

Device Driver Ex-2



```
int virtual_device_release( struct inode *inode, struct file *filp )
{
  printk( KERN_ALERT "virtual device release function called\n" );
  return 0;
}
```



```
ssize_t virtual_device_write( struct file *filp, const char *buf,
                              size_t count, loff_t *f_pos )
{
  printk( KERN_ALERT "virtual_device write function called\n" );
  strcpy( buffer, buf );
  return count;
}
```

File pointer

a pointer in user space

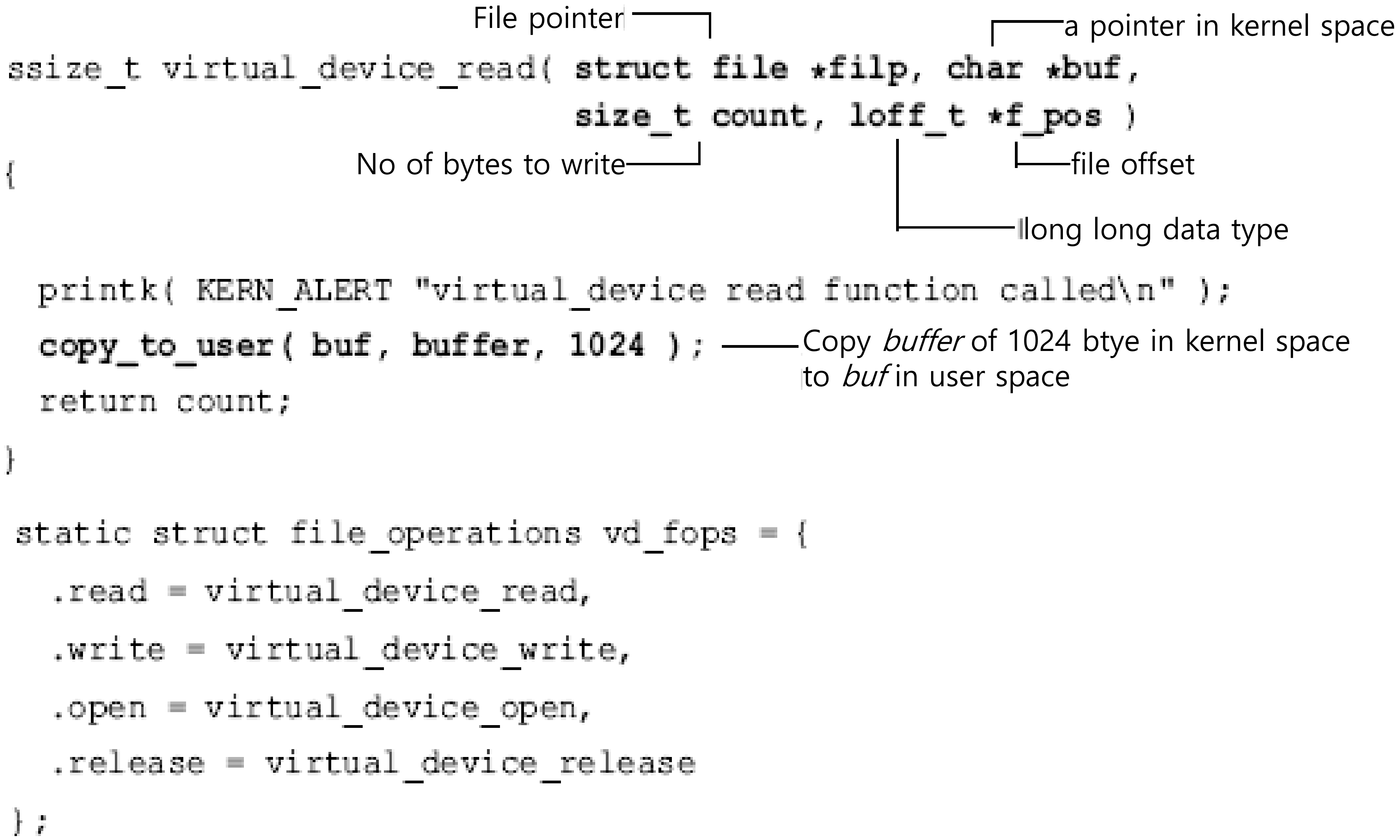
No of bytes to write

file offset

long long data type

Copy *buf* in user space to *buffer* in kernel space
참고로, 일반라이브러리 함수가 아니라 커널내 구현된 함수

Device Driver Ex-3



```
ssize_t virtual_device_read( struct file *filp, char *buf,
                             size_t count, loff_t *f_pos )
{

  printk( KERN_ALERT "virtual_device read function called\n" );
  copy_to_user( buf, buffer, 1024 );
  return count;
}

static struct file_operations vd_fops = {
   .read = virtual_device_read,
   .write = virtual_device_write,
   .open = virtual_device_open,
   .release = virtual_device_release
};
```

File pointer

a pointer in kernel space

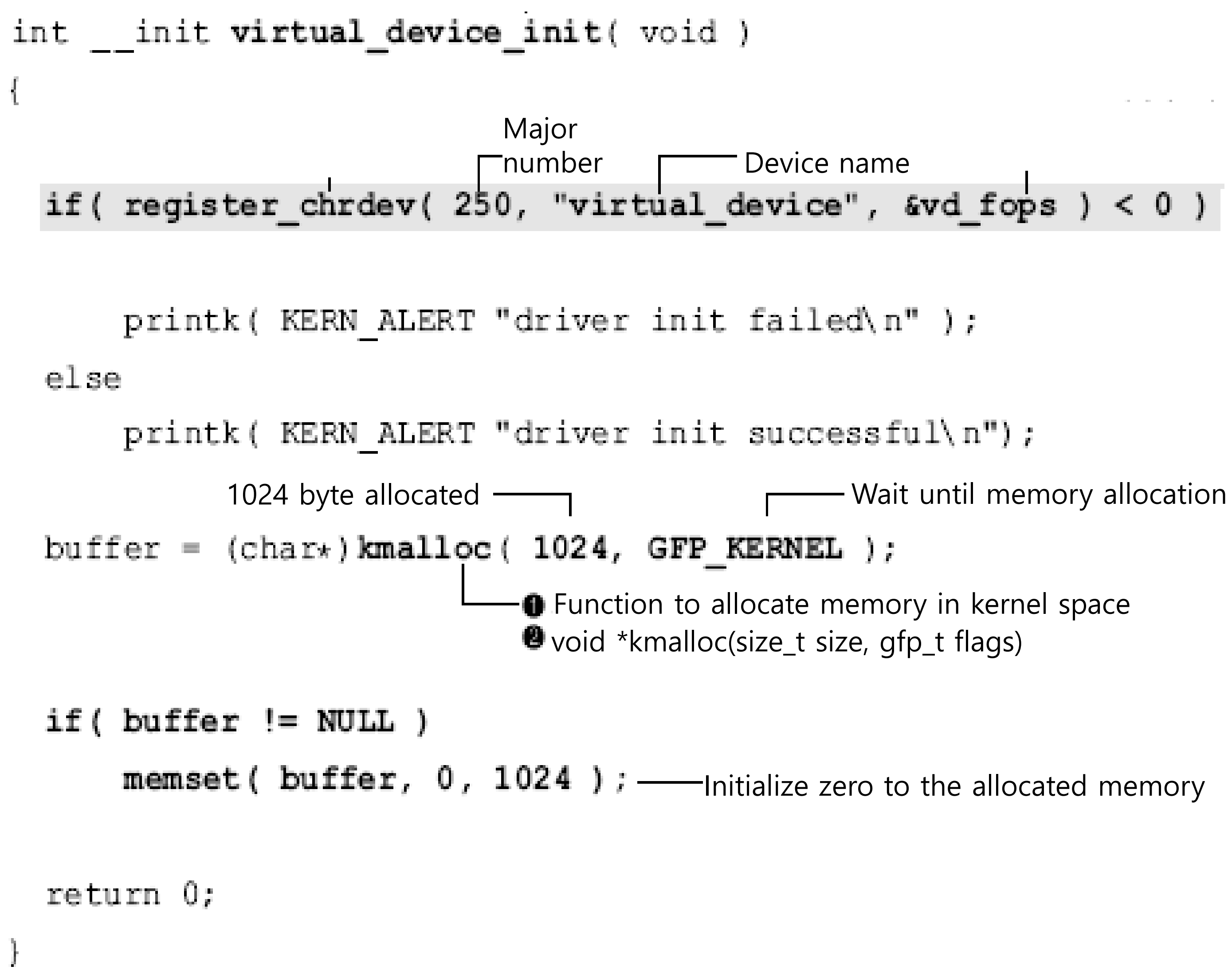
No of bytes to write

file offset

long long data type

Copy *buffer* of 1024 btye in kernel space to *buf* in user space

Device Driver Ex-4



```
int __init virtual_device_init( void )
{

  if( register_chrdev( 250, "virtual_device", &vd_fops ) < 0 )

      printk( KERN_ALERT "driver init failed\n" );
  else
      printk( KERN_ALERT "driver init successful\n");

  buffer = (char*)kmalloc( 1024, GFP_KERNEL );

  if( buffer != NULL )
      memset( buffer, 0, 1024 );

  return 0;
}
```

Major number

Device name

1024 byte allocated

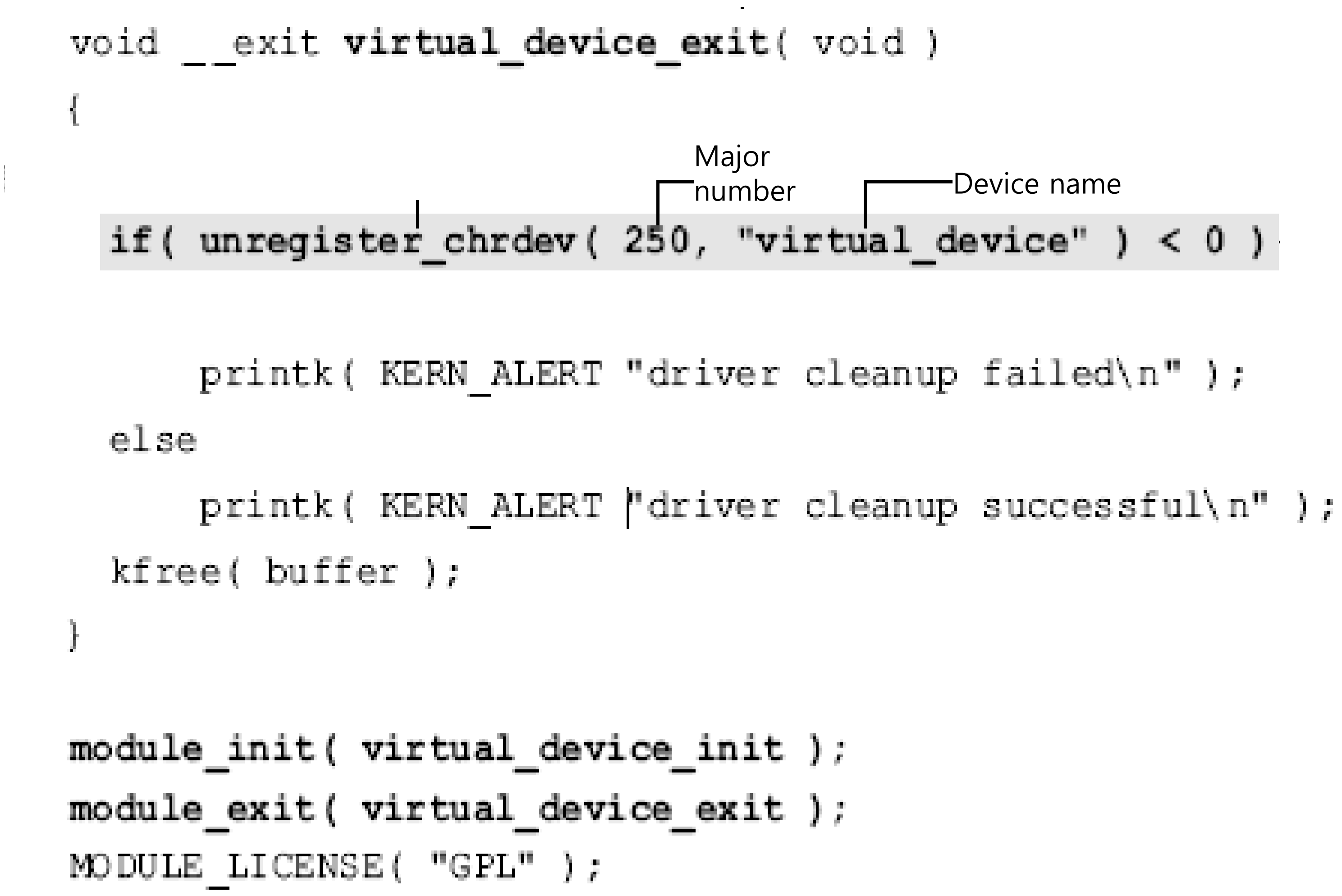
Wait until memory allocation

❶ Function to allocate memory in kernel space

❷ void *kmalloc(size_t size, gfp_t flags)

Initialize zero to the allocated memory

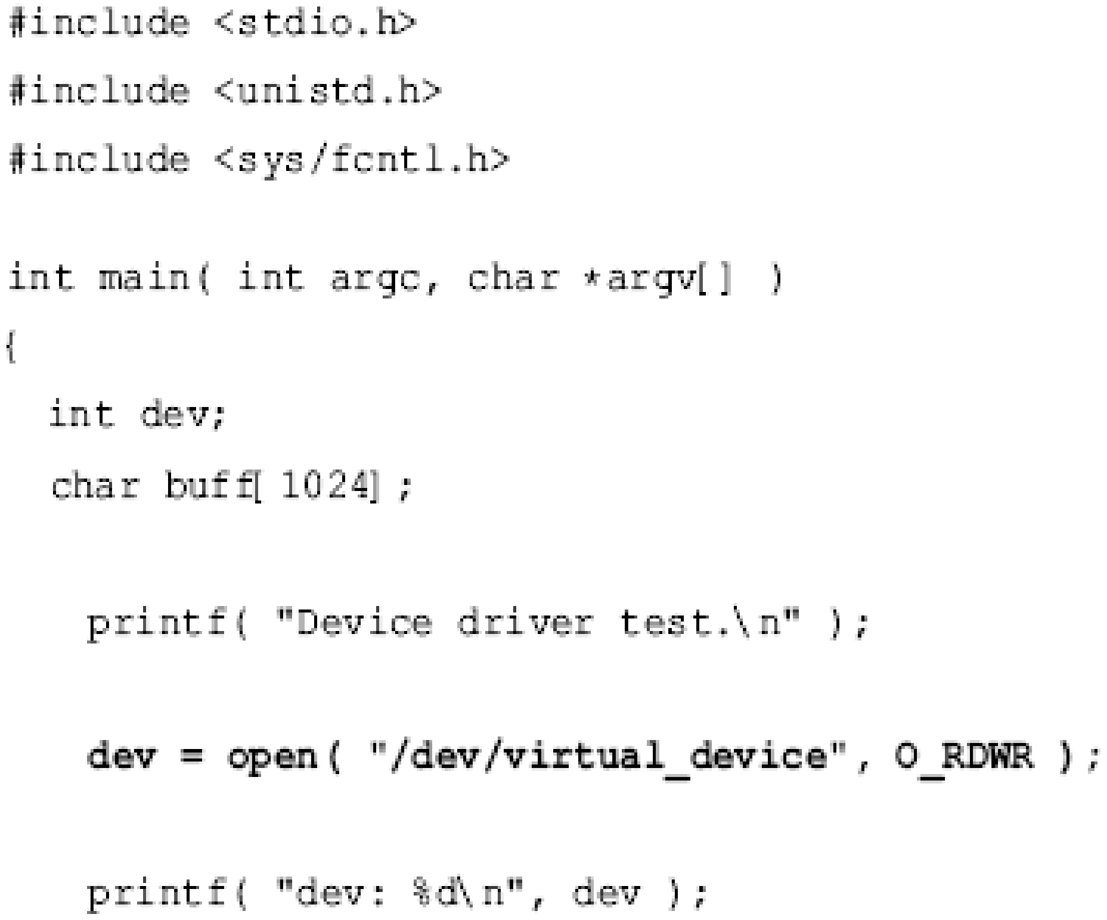
Device Driver Ex-5



```
void __exit virtual_device_exit( void )
{

  if( unregister_chrdev( 250, "virtual_device" ) < 0 )

      printk( KERN_ALERT "driver cleanup failed\n" );
  else
      printk( KERN_ALERT "driver cleanup successful\n" );
  kfree( buffer );
}

module_init( virtual_device_init );
module_exit( virtual_device_exit );
MODULE_LICENSE( "GPL" );
```

Application Ex-1

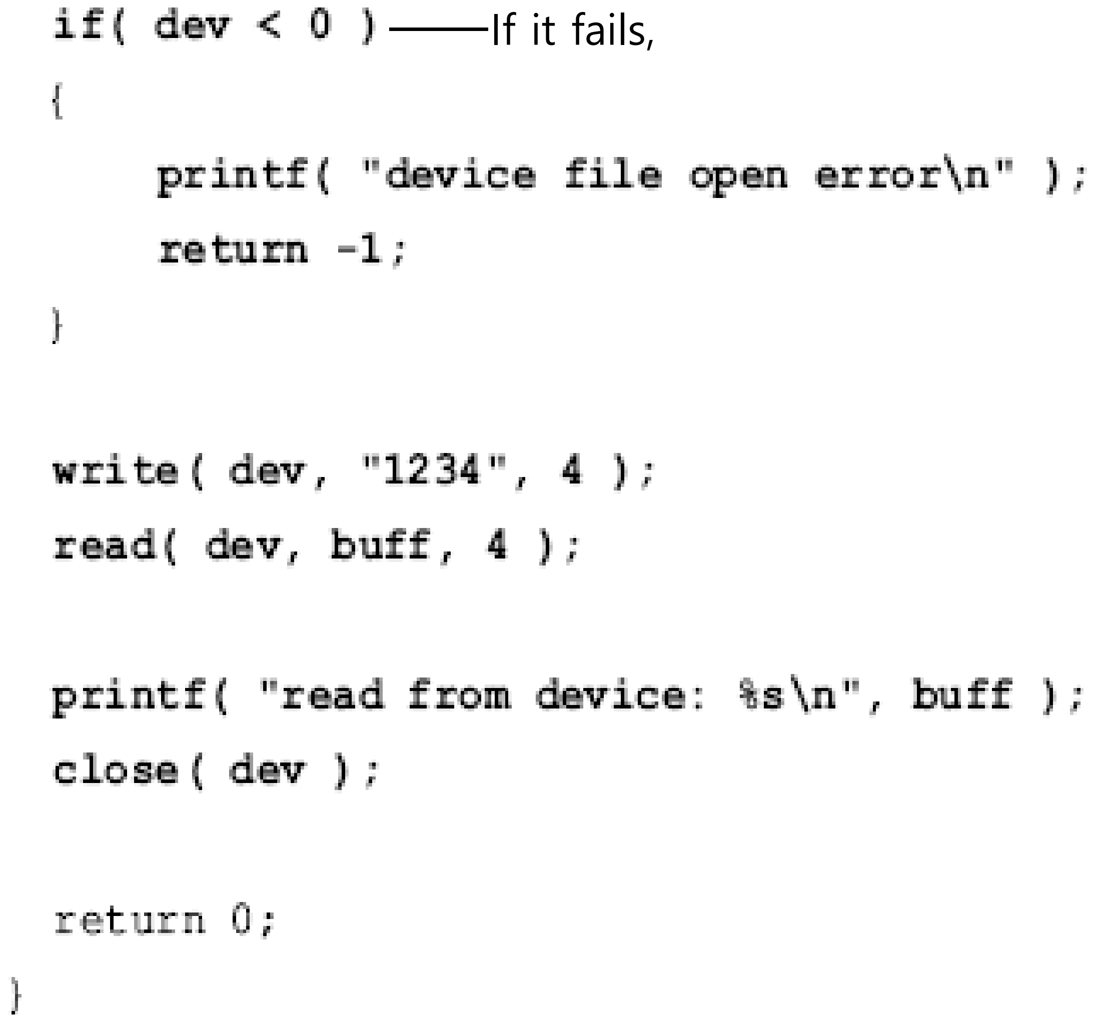
- Test program for write/read operation on a virtual device



```
#include <stdio.h>
#include <unistd.h>
#include <sys/fcntl.h>

int main( int argc, char *argv[] )
{
  int dev;
  char buff[ 1024] ;

    printf( "Device driver test.\n" );

    dev = open( "/dev/virtual_device", O_RDWR );

    printf( "dev: %d\n", dev );
```

Open the device of read & write mode

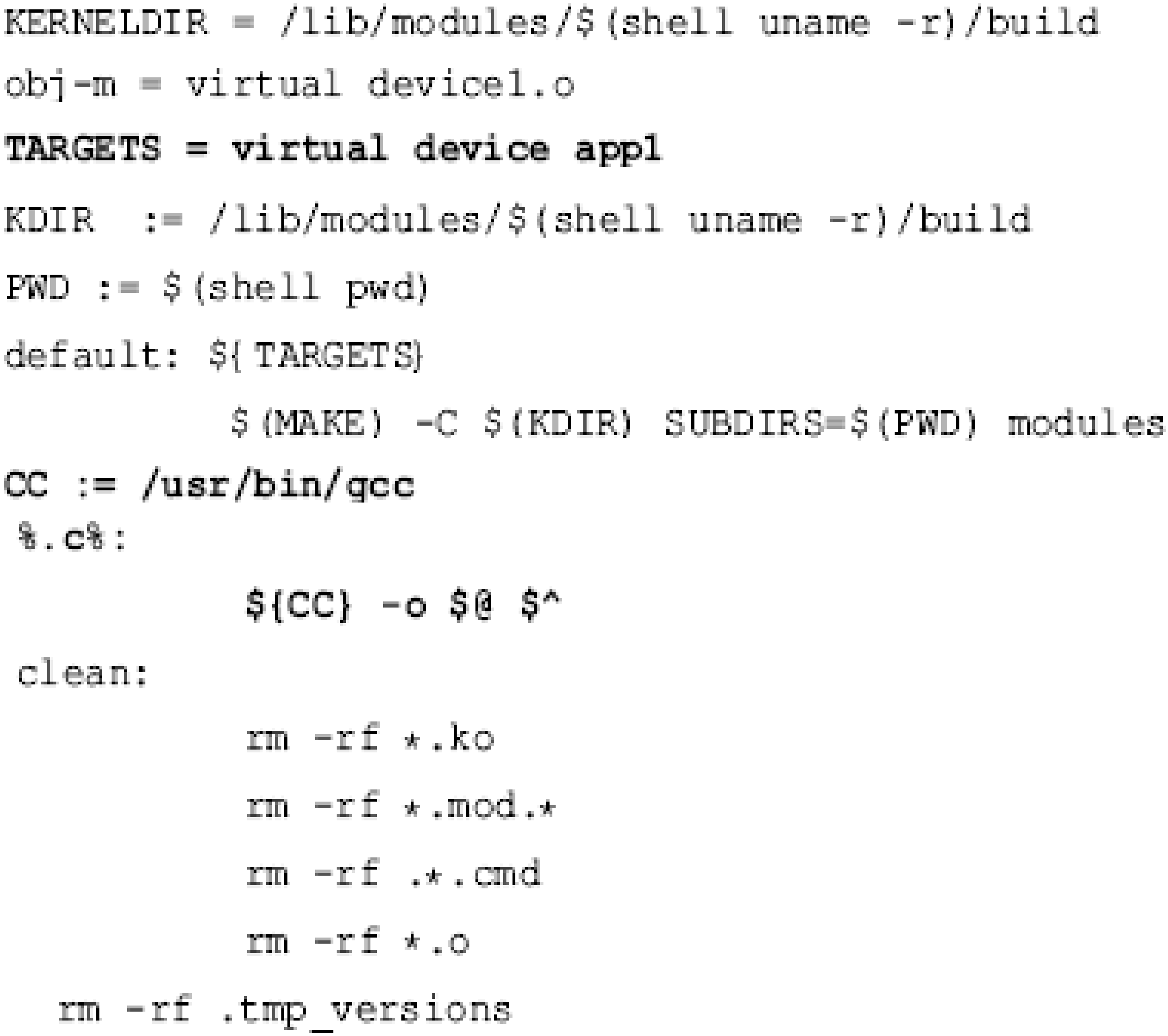
Application -2



```
    if( dev < 0 )
    {
        printf( "device file open error\n" );
        return -1;
    }

    write( dev, "1234", 4 );
    read( dev, buff, 4 );

    printf( "read from device: %s\n", buff );
    close( dev );

    return 0;
}
```

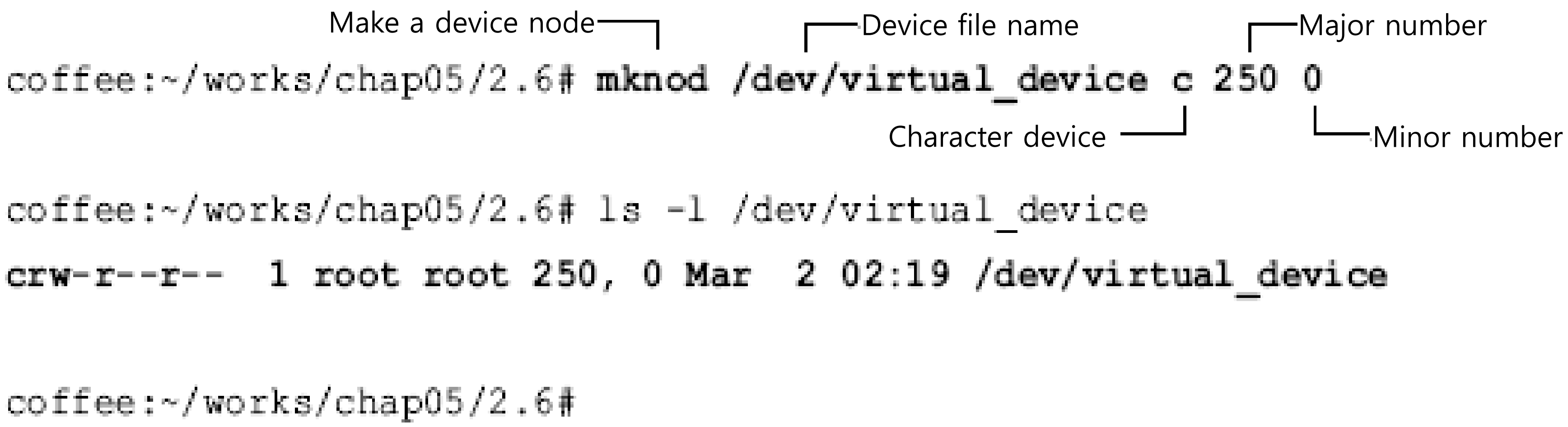
If it fails,

Makefile



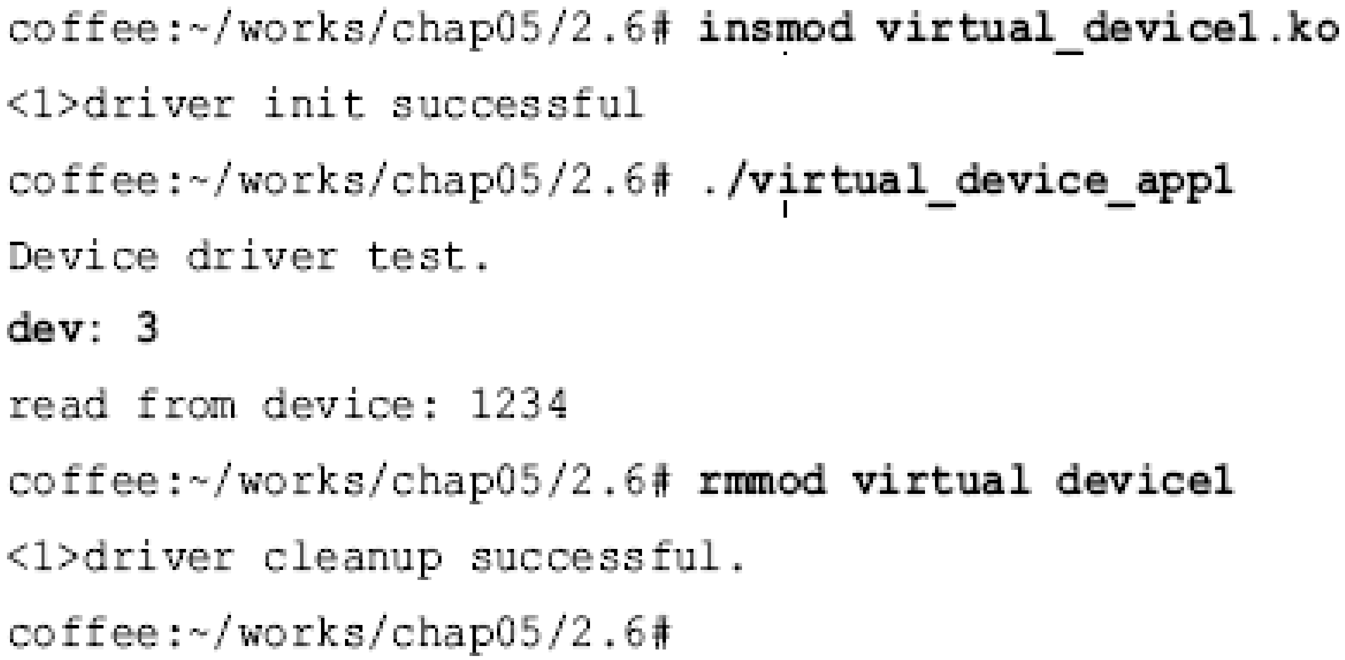
```
KERNELDIR = /lib/modules/$(shell uname -r)/build
obj-m = virtual device1.o
TARGETS = virtual device appl
KDIR  := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)
default: ${TARGETS}
        $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
CC := /usr/bin/gcc
 %.c%:
        ${CC} -o $@ $^
 clean:
        rm -rf *.ko
        rm -rf *.mod.*
        rm -rf .*.cmd
        rm -rf *.o
  rm -rf .tmp_versions
```

Device Node Creation



```
coffee:~/works/chap05/2.6# mknod /dev/virtual_device c 250 0

coffee:~/works/chap05/2.6# ls -l /dev/virtual_device
crw-r--r--  1 root root 250, 0 Mar  2 02:19 /dev/virtual_device

coffee:~/works/chap05/2.6#
```

Test



```
coffee:~/works/chap05/2.6# insmod virtual_device1.ko
<1>driver init successful
coffee:~/works/chap05/2.6# ./virtual_device_app1
Device driver test.
dev: 3
read from device: 1234
coffee:~/works/chap05/2.6# rmmod virtual device1
<1>driver cleanup successful.
coffee:~/works/chap05/2.6#
```

Function Pointer in Structure

- ANSI C90

```
static struct file_operations call_fops = {
  NULL, NULL, xxx_read, xxx_write, NULL, NULL, NULL, NULL, xxx_open, NULL, xxx_release,
  NULL, NULL, NULL, NULL, NULL, NULL, NULL };
```

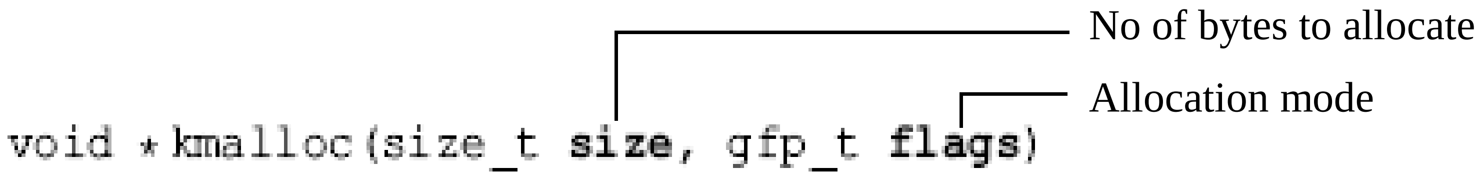
- GCC Extension

```
static struct file_operations call_fops = {
  read : xxx_read,
  write: xxx_write,
  open: xxx_open,
  release: xxx_release };
```

- ANSI C99

```
static struct file_operations call_fops = {
  .read = xxx_read,
  .write = xxx_write,
  .open = xxx_open,
  .release = xxx_release  };
```

kmalloc()



```
void *kmalloc(size_t size, gfp_t flags)
```

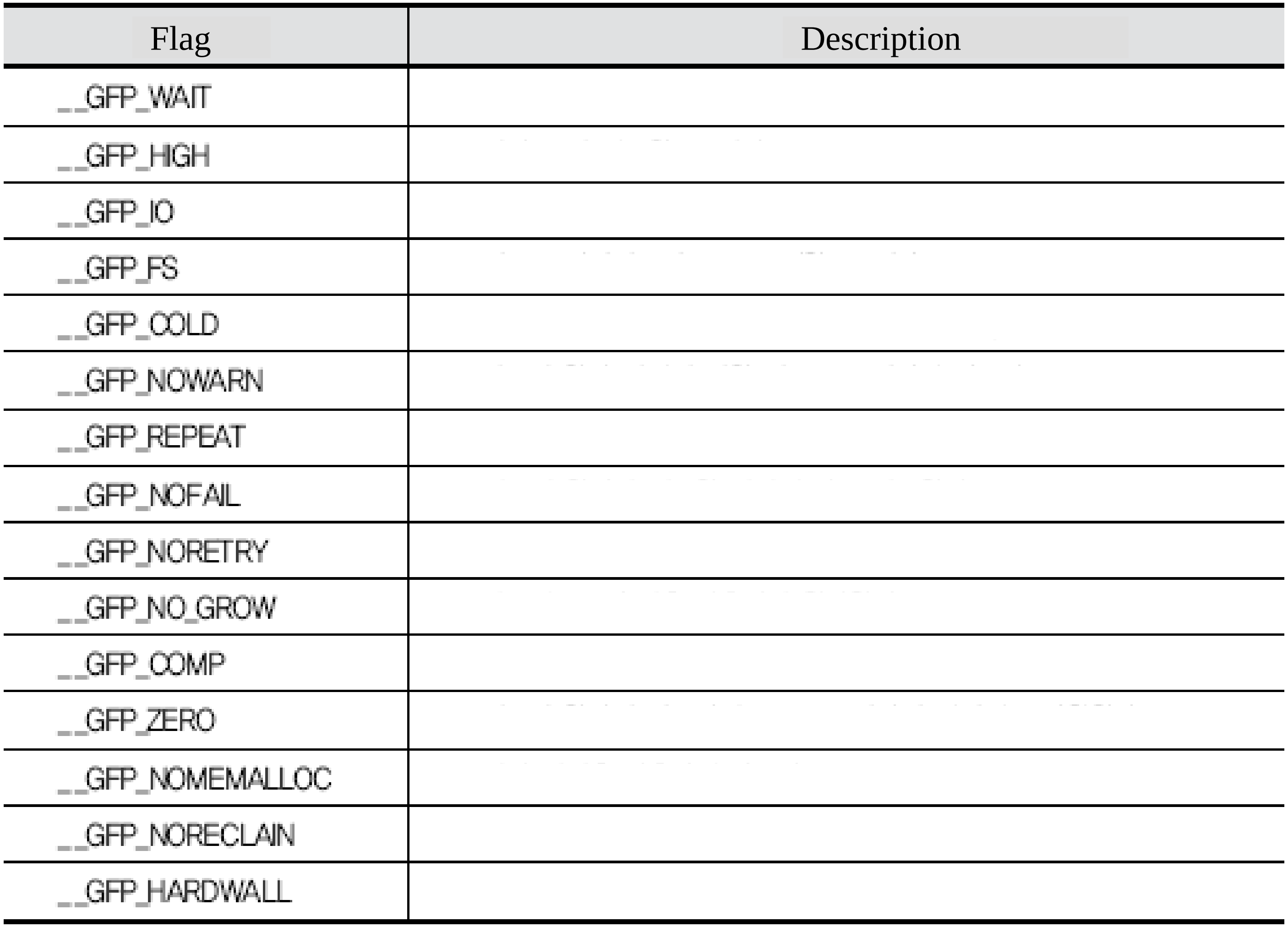
No of bytes to allocate

Allocation mode

GFP_ATOMIC : __GFP_HIGH

GFP_KERNEL : __GFP_WAIT|__GFP_IO|__GFP_FS

Flags

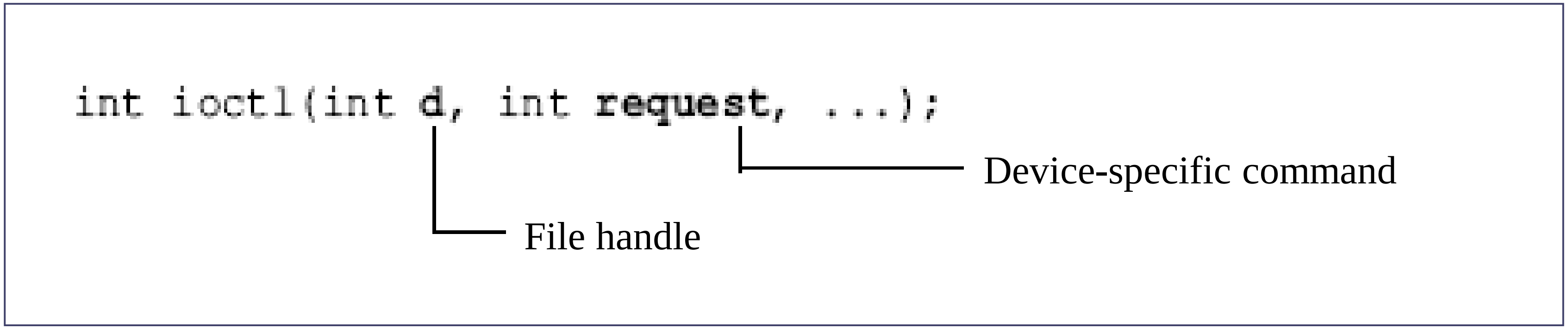


Flag	Description
__GFP_WAIT	
__GFP_HIGH	
__GFP_IO	
__GFP_FS	
__GFP_COLD	
__GFP_NOWARN	
__GFP_REPEAT	
__GFP_NOFAIL	
__GFP_NORETRY	
__GFP_NO_GROW	
__GFP_COMP	
__GFP_ZERO	
__GFP_NOMEMALLOC	
__GFP_NORECLAIN	
__GFP_HARDWALL	

IOCTL Programming

IOCTL?

- the Single UNIX specification only as an extension for dealing with STREAMS devices, [Rago, S.A. 1993. UNIX System V Network Programming. Addison-Wesley]
  - But, it is now used for various devices and regular files.
- In addition to read/write operations, used for device specific operations control
  - Device control or status
  - Read/Write operations
  - Depending the application commands, process parameters differently.



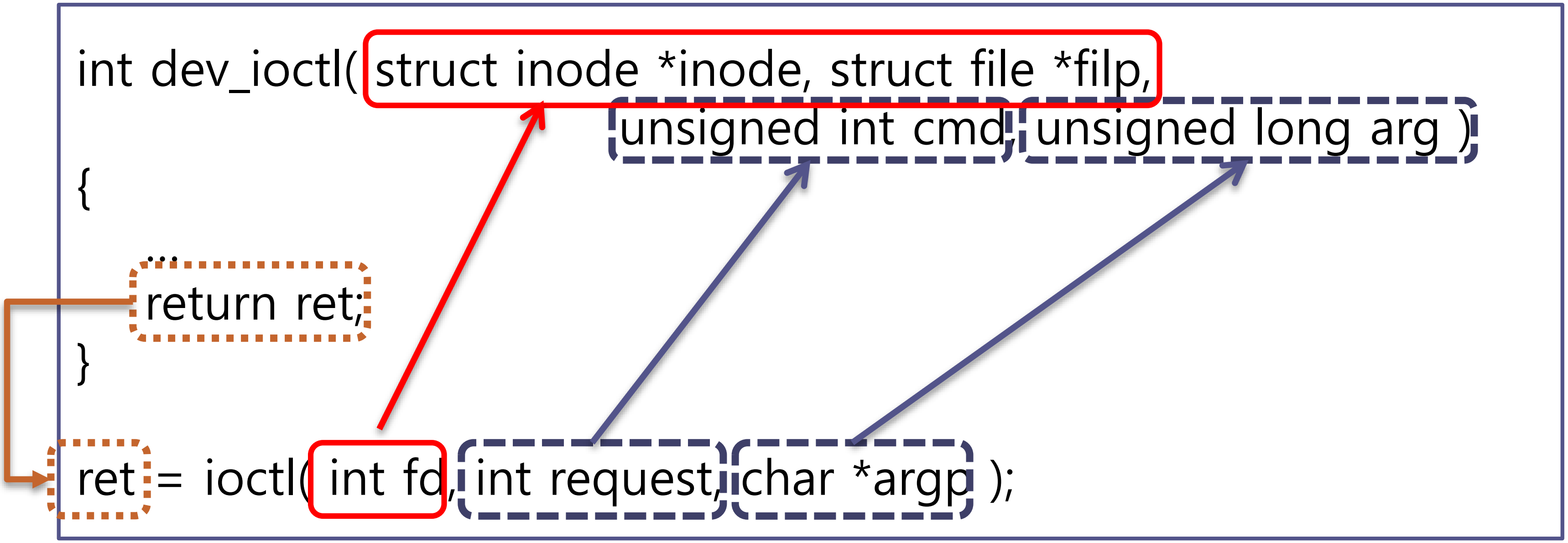
```
int ioctl(int d, int request, ...);
```

Device-specific command

File handle

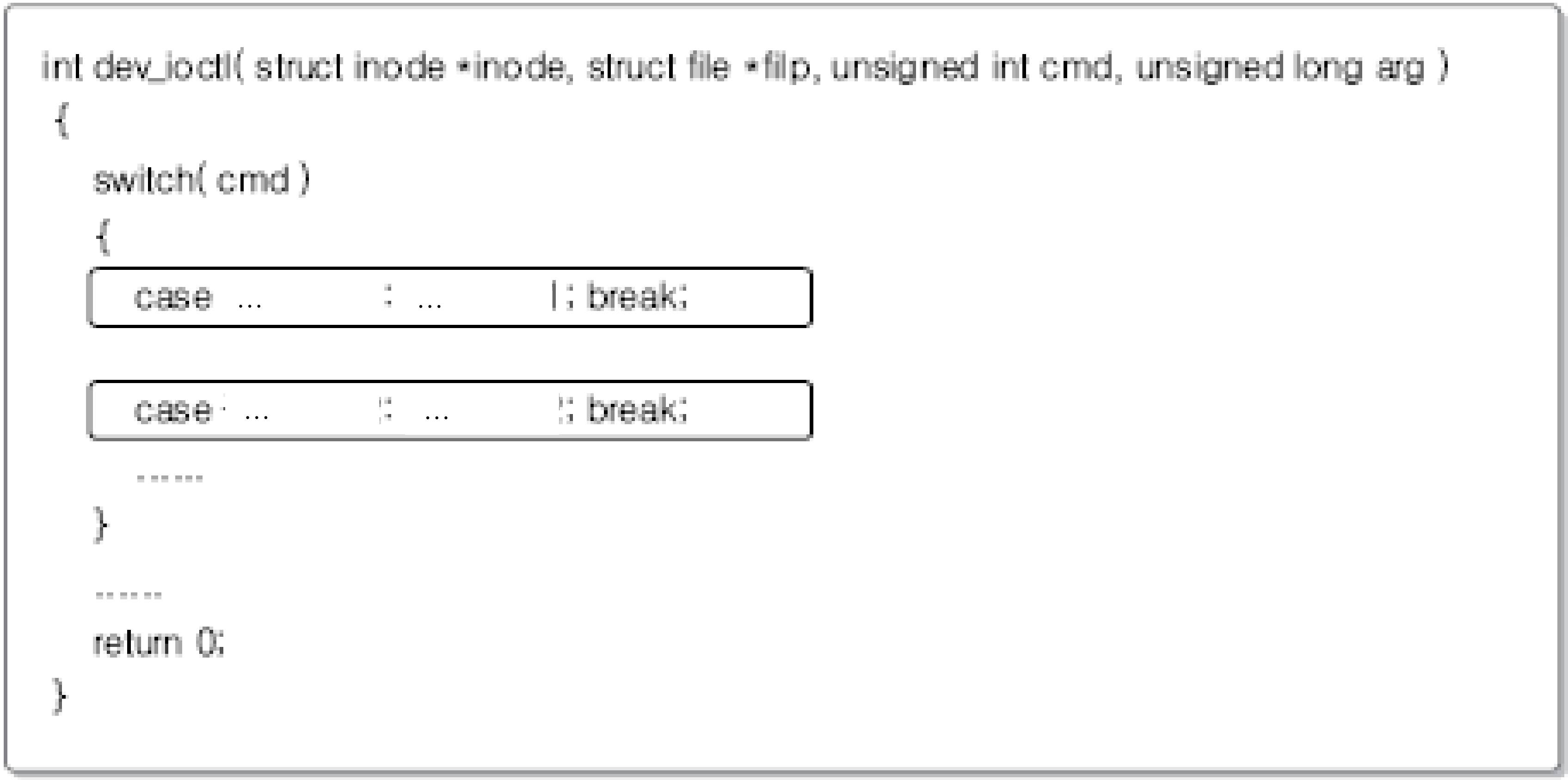
IOCTL Parameters

- ioctl() -> sys_ioctl() -> ioctl() in device driver



```
int dev_ioctl( struct inode *inode, struct file *filp,
                         unsigned int cmd, unsigned long arg )
{
    …
    return ret;
}
ret = ioctl( int fd, int request, char *argp );
```

IOCTL() Structure



```
int dev_ioctl( struct inode *inode, struct file *filp, unsigned int cmd, unsigned long arg )
 {
    switch( cmd )
    {
      case  ...          :  ...          ; break;

      case  ...          :  ...          ; break;
        ......
    }
    ......
    return 0;
 }
```

cmd in IOCTL()

```
int dev_ioctl( struct inode *inode, struct file *flip,
                              unsigned int cmd, unsigned long arg )
```

2 bit	8 bit	8 bit	14 bit
dir	type	nr	size
32	30	22	14

cmd structure

Field	Bit	Description
nr	8	Command number
type	8	Device (Magic) number
size	14	Data size of argument
dir	2	Read/Write attribute

IOCTL Macros-1

- cmd R/W Macros
  - _IO: No additional data
  - _IOR : Read command
  - _IOW : Write command
  - _IOWR : Read/Write command
- Macro types
  - _IO(magic_number, class_number)
  - _IOR(magic_number, class_number, variable_type)
  - _IOW(magic_number, class_number, variable_type)
  - _IOWR(magic_number, class_number, variable_type)

```
#define MY_IOCTL_READ            _IOR( MY_IOCTL_NUMBER, 0, int )
#define MY_IOCTL_WRITE           _IOW( MY_IOCTL_NUMBER, 1, int )
#define MY_IOCTL_STATUS           _IO( MY_IOCTL_NUMBER, 2 )
#define MY_IOCTL_READ_WRITE       _IOWR( MY_IOCTL_NUMBER, 3, int )
```

IOCTL Macros-2

- cmd information macros
  - _IOC_NR : get class_number
  - _IOC_TYPE: get magic_number
  - _IOC_SIZE: get data_size
  - _IOC_DIR : get read/write fields
- Macro types
  - _IOC_NR(cmd)
  - _IOC_TYPE(cmd)
  - _IOC_SIZE(cmd)
  - _IOC_DIR(cmd)

_IOC_DIR macro returns
- _IOC_NONE
- _IOC_READ
- _IOC_WRITE
- _IOC_READ | _IOC_WRITE

```
if( _IOC_TYPE( cmd ) != MY_IOCTL_NUMBER ) return -EINVAL;
if( _IOC_NR( cmd ) >= MY_IOCTL_NR ) return -EINVAL;
size = _IOC_SIZE( cmd );
```

IOCTL Example

```
#define MY_IOCTL_READ          _IOR( MY_IOCTL_NUMBER, 0, int )
#define MY_IOCTL_WRITE         _IOW( MY_IOCTL_NUMBER, 1, int )
……
int data = 0;
dev = open( “/dev/virtual_device”, O_RDWR );
ioctl( dev, MY_IOCTL_READ, &data );
ioctl( dev, MY_IOCTL_WRITE, &data );
```

Memory Verification

- Verify if memory in user space is valid or not
- #define access_ok(type,addr,size) : 2.4.20 later

Why Macro?

```
#include <asm/uaccess.h>
#define access_ok(type,addr,size) (__range_ok(addr,size) == 0)
```

To test irrespective of data type

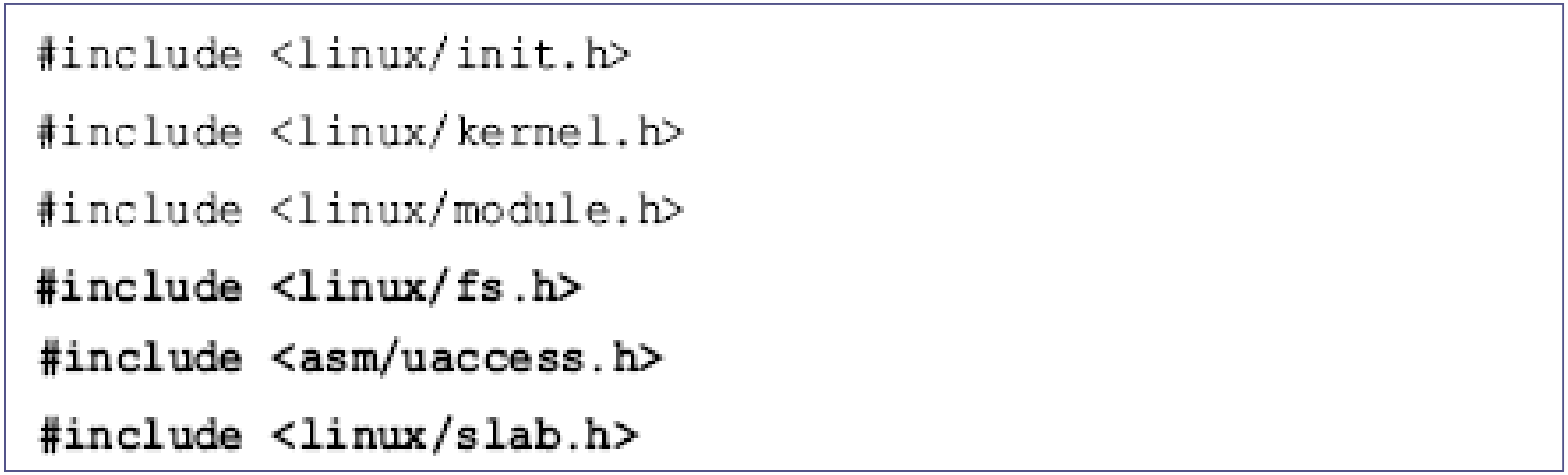
- type:
  - VERIFY_READ : readable?
  - VERIFY_WRITE : writable?

```
int res;
int error = access_ok( VERIFY_WRITE, res, sizeof( int ) );
if( error < 0 ) return error;
```

IOCTL Practice

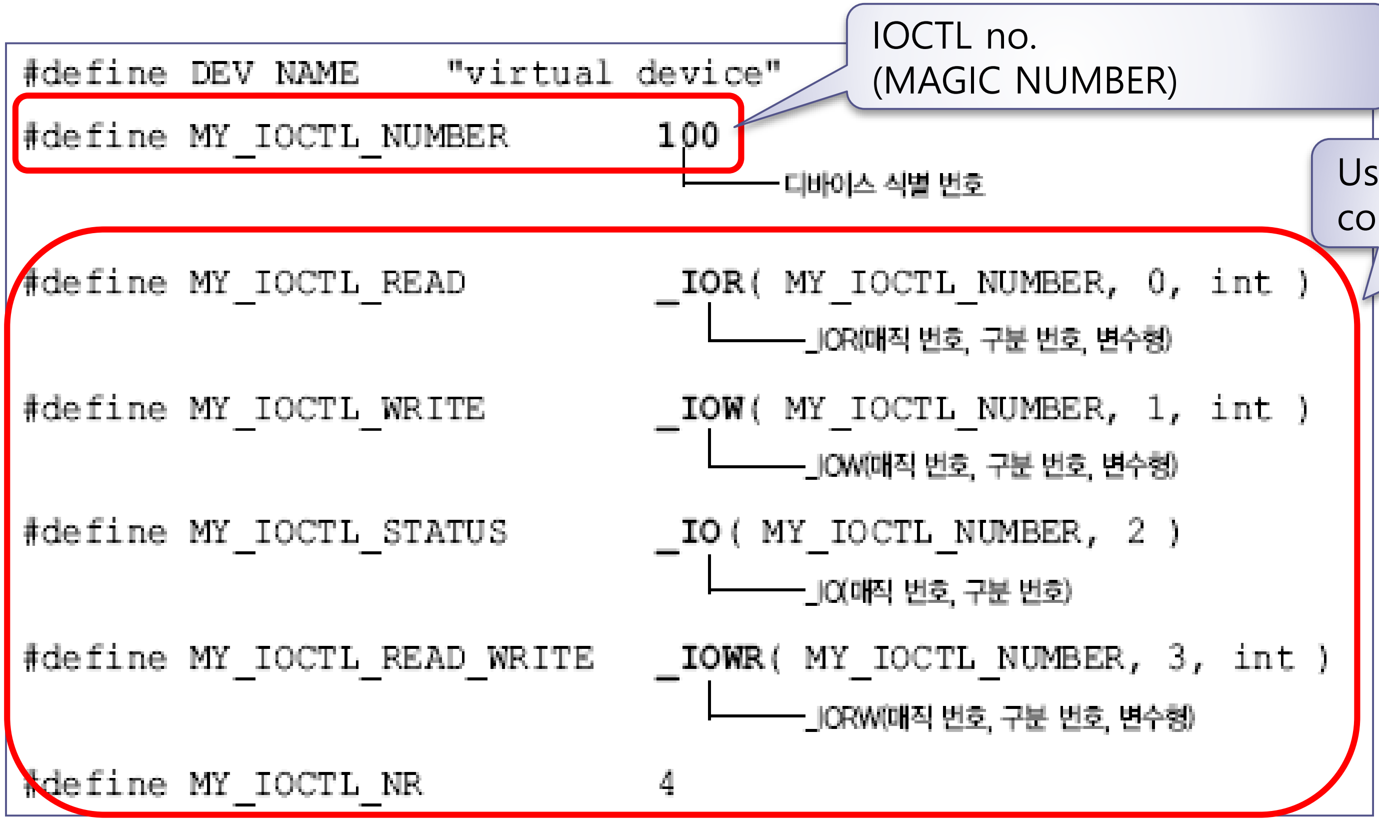
IOCTL device driver -1

- Header files



```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <asm/uaccess.h>
#include <linux/slab.h>
```

IOCTL device driver -2

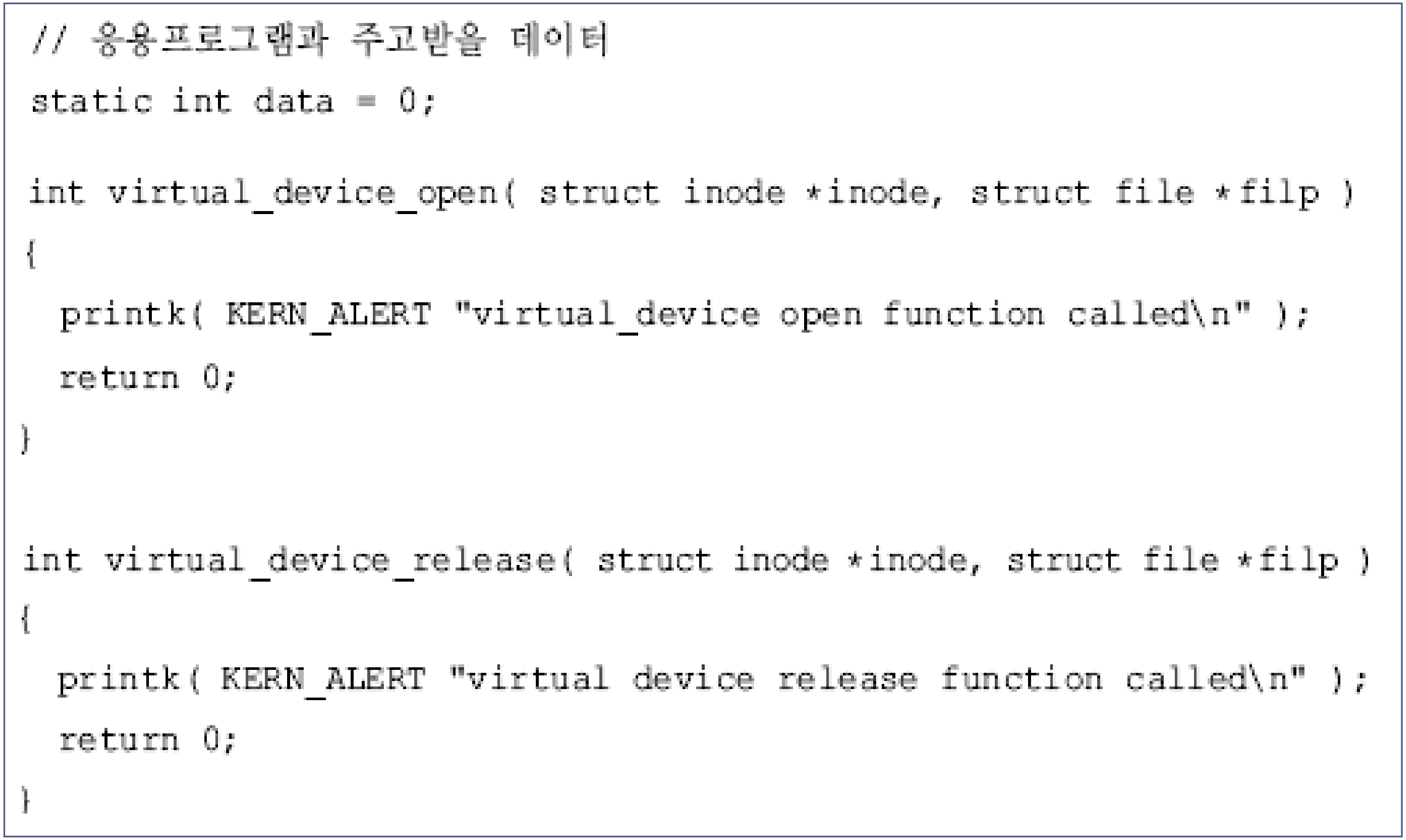


```
#define DEV_NAME        "virtual device"
#define MY_IOCTL_NUMBER         100
                                  디바이스 식별 번호

#define MY_IOCTL_READ           _IOR( MY_IOCTL_NUMBER, 0, int )
                                  _IOR(매직 번호, 구분 번호, 변수형)

#define MY_IOCTL_WRITE          _IOW( MY_IOCTL_NUMBER, 1, int )
                                  _IOW(매직 번호, 구분 번호, 변수형)

#define MY_IOCTL_STATUS         _IO( MY_IOCTL_NUMBER, 2 )
                                  _IO(매직 번호, 구분 번호)

#define MY_IOCTL_READ_WRITE     _IOWR( MY_IOCTL_NUMBER, 3, int )
                                  _IORW(매직 번호, 구분 번호, 변수형)

#define MY_IOCTL_NR             4
```

IOCTL no.
(MAGIC NUMBER)

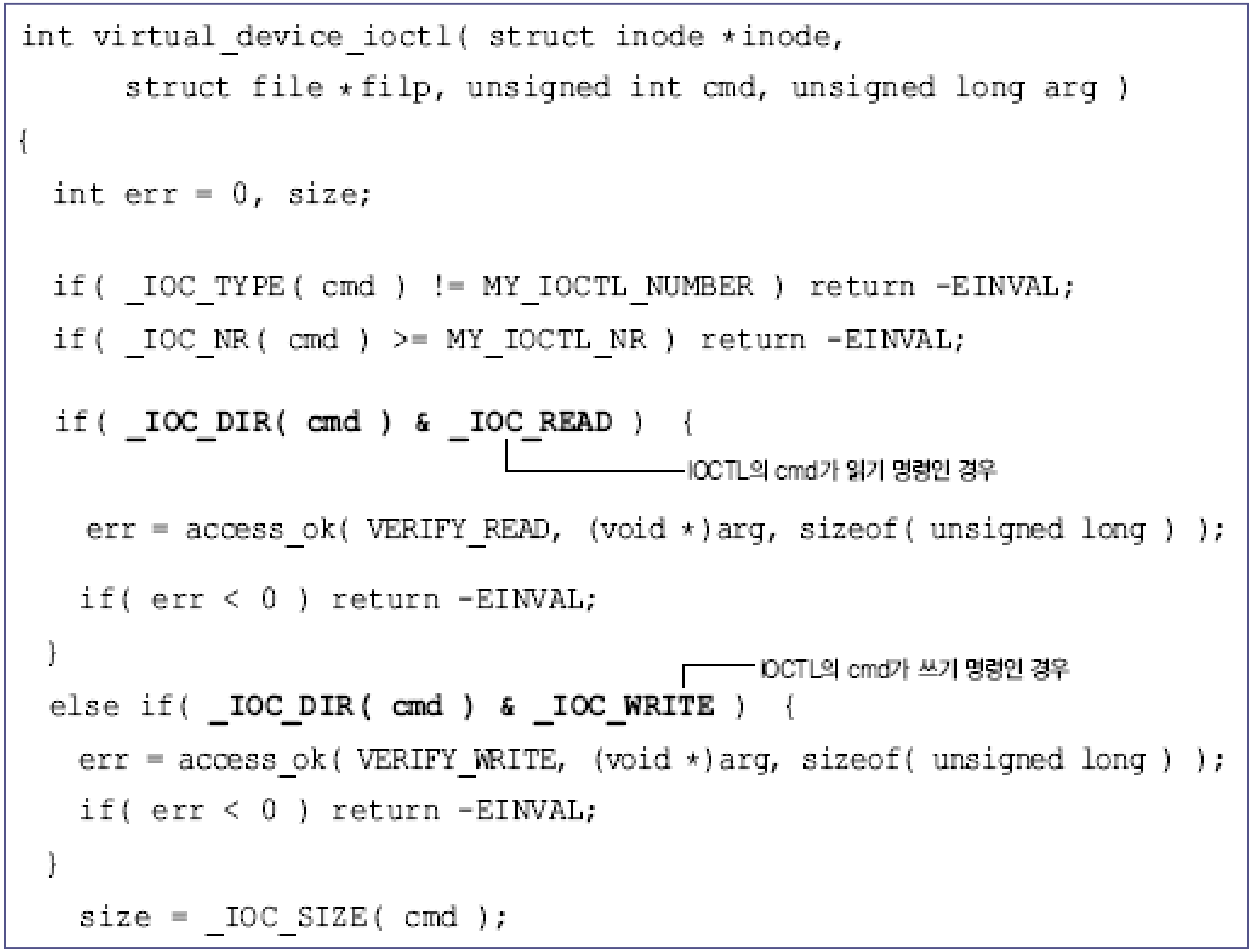
User-defined IOCTL command

IOCTL device driver -3



```
// 응용프로그램과 주고받을 데이터
static int data = 0;

int virtual_device_open( struct inode *inode, struct file *filp )
{
  printk( KERN_ALERT "virtual_device open function called\n" );
  return 0;
}

int virtual_device_release( struct inode *inode, struct file *filp )
{
  printk( KERN_ALERT "virtual device release function called\n" );
  return 0;
}
```

IOCTL device driver -4

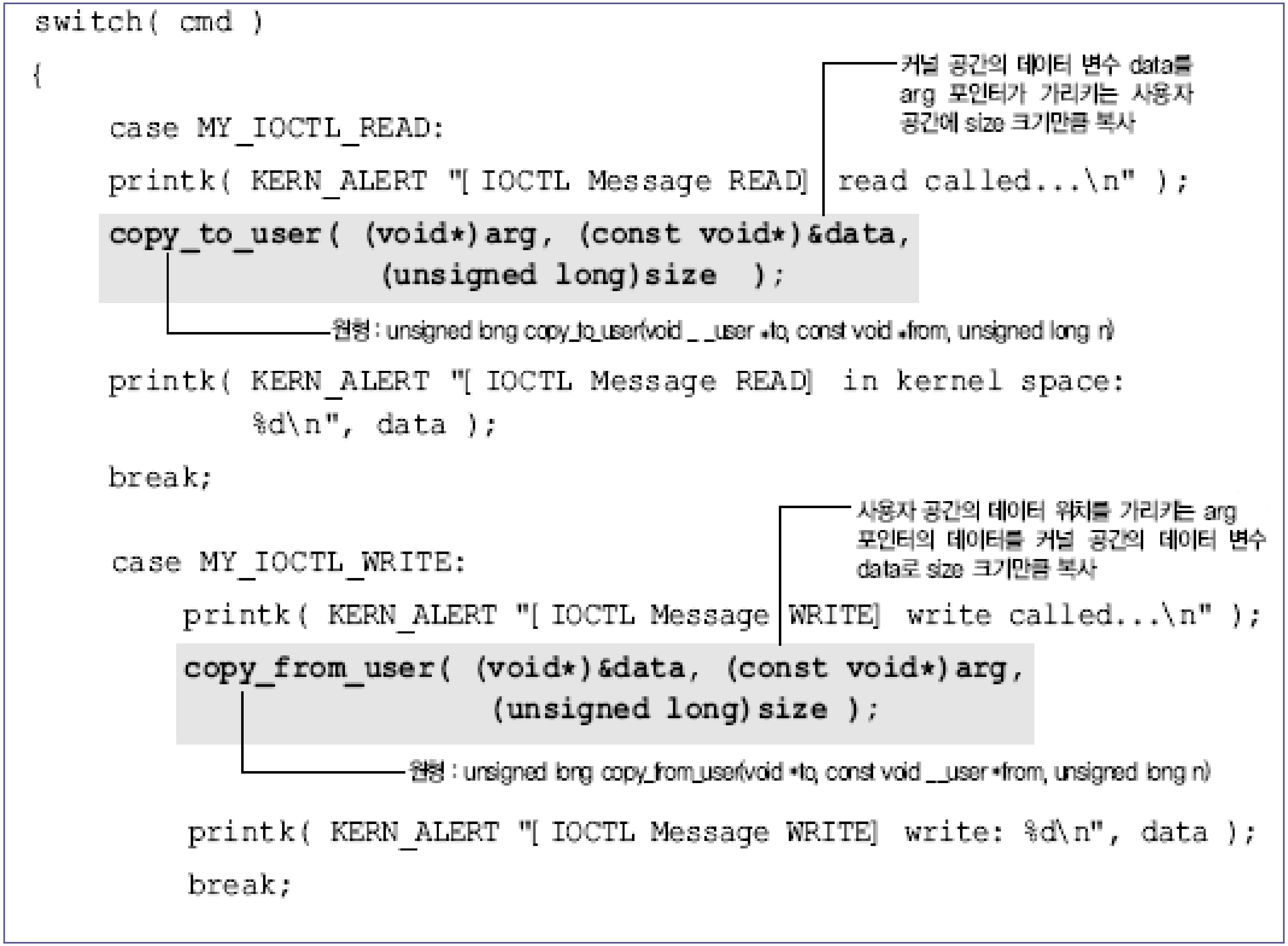


```
int virtual_device_ioctl( struct inode *inode,
      struct file *filp, unsigned int cmd, unsigned long arg )
{
  int err = 0, size;

  if( _IOC_TYPE( cmd ) != MY_IOCTL_NUMBER ) return -EINVAL;
  if( _IOC_NR( cmd ) >= MY_IOCTL_NR ) return -EINVAL;

  if( _IOC_DIR( cmd ) & _IOC_READ )  {

    err = access_ok( VERIFY_READ, (void *)arg, sizeof( unsigned long ) );

    if( err < 0 ) return -EINVAL;
  }
  else if( _IOC_DIR( cmd ) & _IOC_WRITE )  {
    err = access_ok( VERIFY_WRITE, (void *)arg, sizeof( unsigned long ) );
    if( err < 0 ) return -EINVAL;
  }
    size = _IOC_SIZE( cmd );
```

IOCTL의 cmd가 읽기 명령인 경우

IOCTL의 cmd가 쓰기 명령인 경우

IOCTL device driver -5



```
switch( cmd )
{
    case MY_IOCTL_READ:
    printk( KERN_ALERT "[ IOCTL Message READ] read called...\n" );
    copy_to_user( (void*)arg, (const void*)&data,
                  (unsigned long)size  );
    printk( KERN_ALERT "[ IOCTL Message READ] in kernel space:
              %d\n", data );
    break;

    case MY_IOCTL_WRITE:
        printk( KERN_ALERT "[ IOCTL Message WRITE] write called...\n" );
        copy_from_user( (void*)&data, (const void*)arg,
                        (unsigned long)size );
        printk( KERN_ALERT "[ IOCTL Message WRITE] write: %d\n", data );
        break;
```

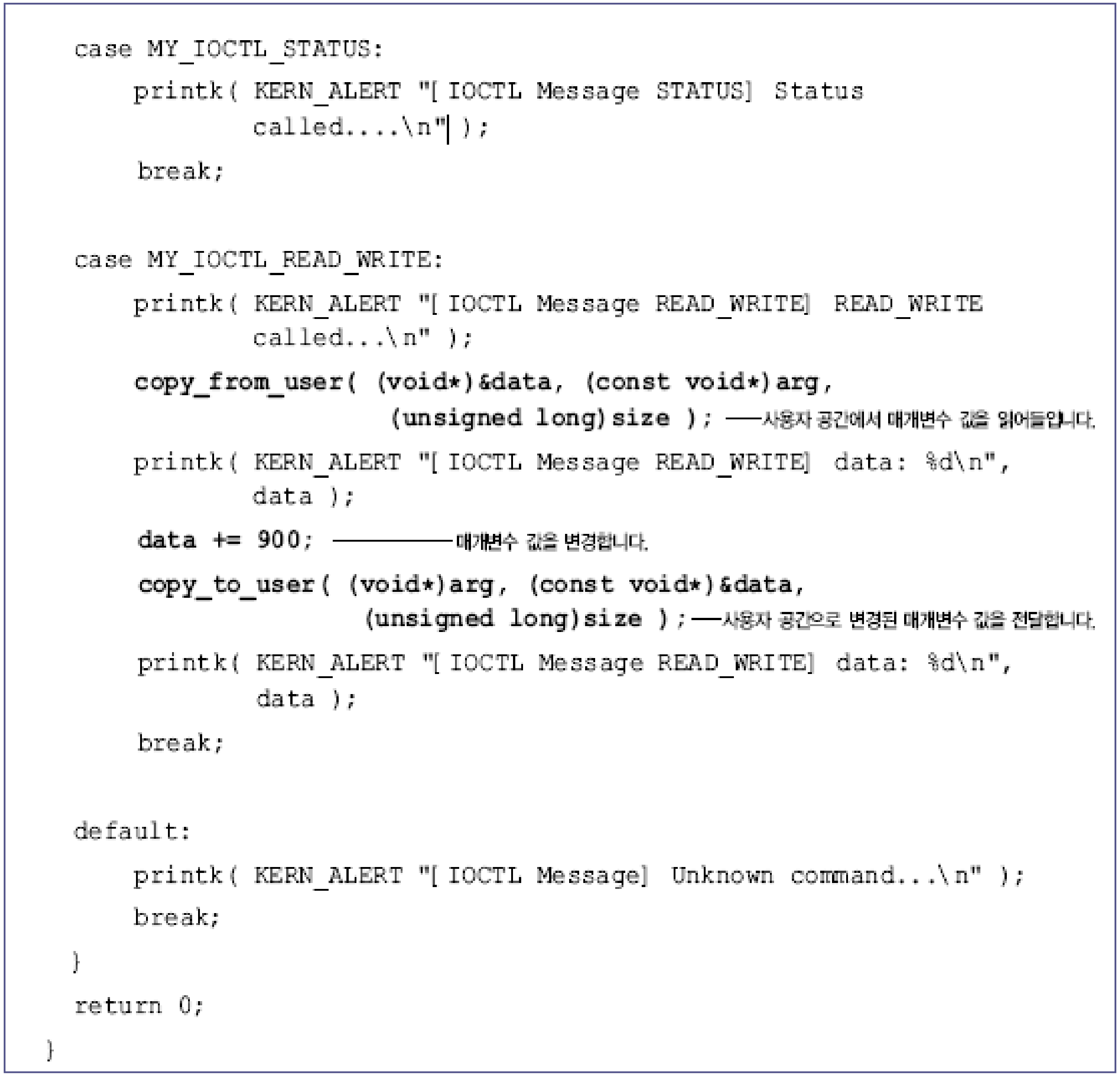
커널 공간의 데이터 변수 data를 arg 포인터가 가리키는 사용자 공간에 size 크기만큼 복사

원형 : unsigned long copy_to_user(void _ _user *to, const void *from, unsigned long n)

사용자 공간의 데이터 위치를 가리키는 arg 포인터의 데이터를 커널 공간의 데이터 변수 data로 size 크기만큼 복사

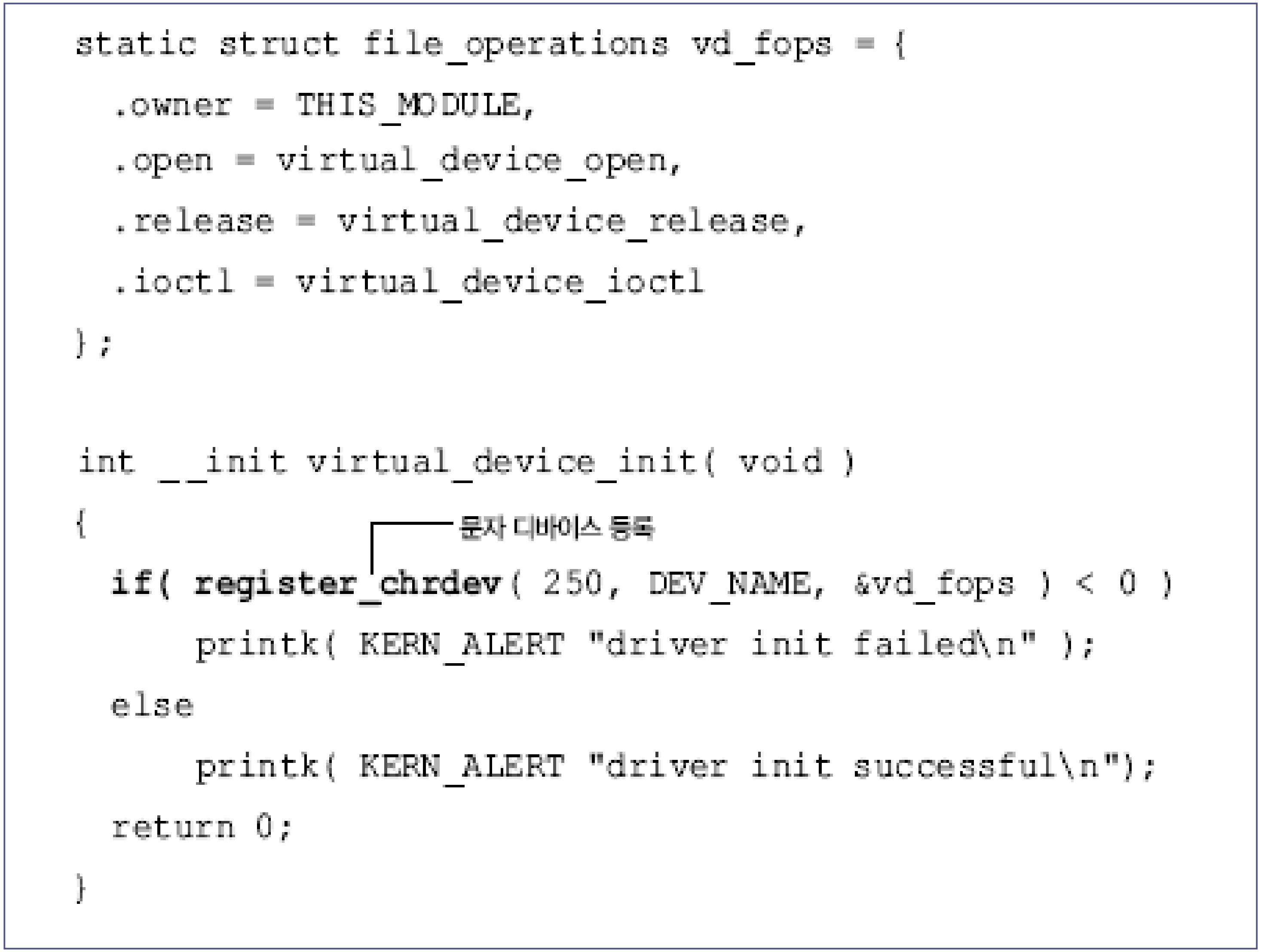
원형 : unsigned long copy_from_user(void *to, const void __user *from, unsigned long n)

IOCTL device driver -6



```
  case MY_IOCTL_STATUS:
      printk( KERN_ALERT "[ IOCTL Message STATUS] Status
              called....\n" );
      break;


  case MY_IOCTL_READ_WRITE:
      printk( KERN_ALERT "[ IOCTL Message READ_WRITE] READ_WRITE
              called...\n" );
      copy_from_user( (void*)&data, (const void*)arg,
                      (unsigned long)size ); ——사용자 공간에서 매개변수 값을 읽어들입니다.
      printk( KERN_ALERT "[ IOCTL Message READ_WRITE] data: %d\n",
              data );
      data += 900; ————매개변수 값을 변경합니다.
      copy_to_user( (void*)arg, (const void*)&data,
                    (unsigned long)size );——사용자 공간으로 변경된 매개변수 값을 전달합니다.
      printk( KERN_ALERT "[ IOCTL Message READ_WRITE] data: %d\n",
              data );
      break;


  default:
      printk( KERN_ALERT "[ IOCTL Message] Unknown command...\n" );
      break;
  }
  return 0;
}
```

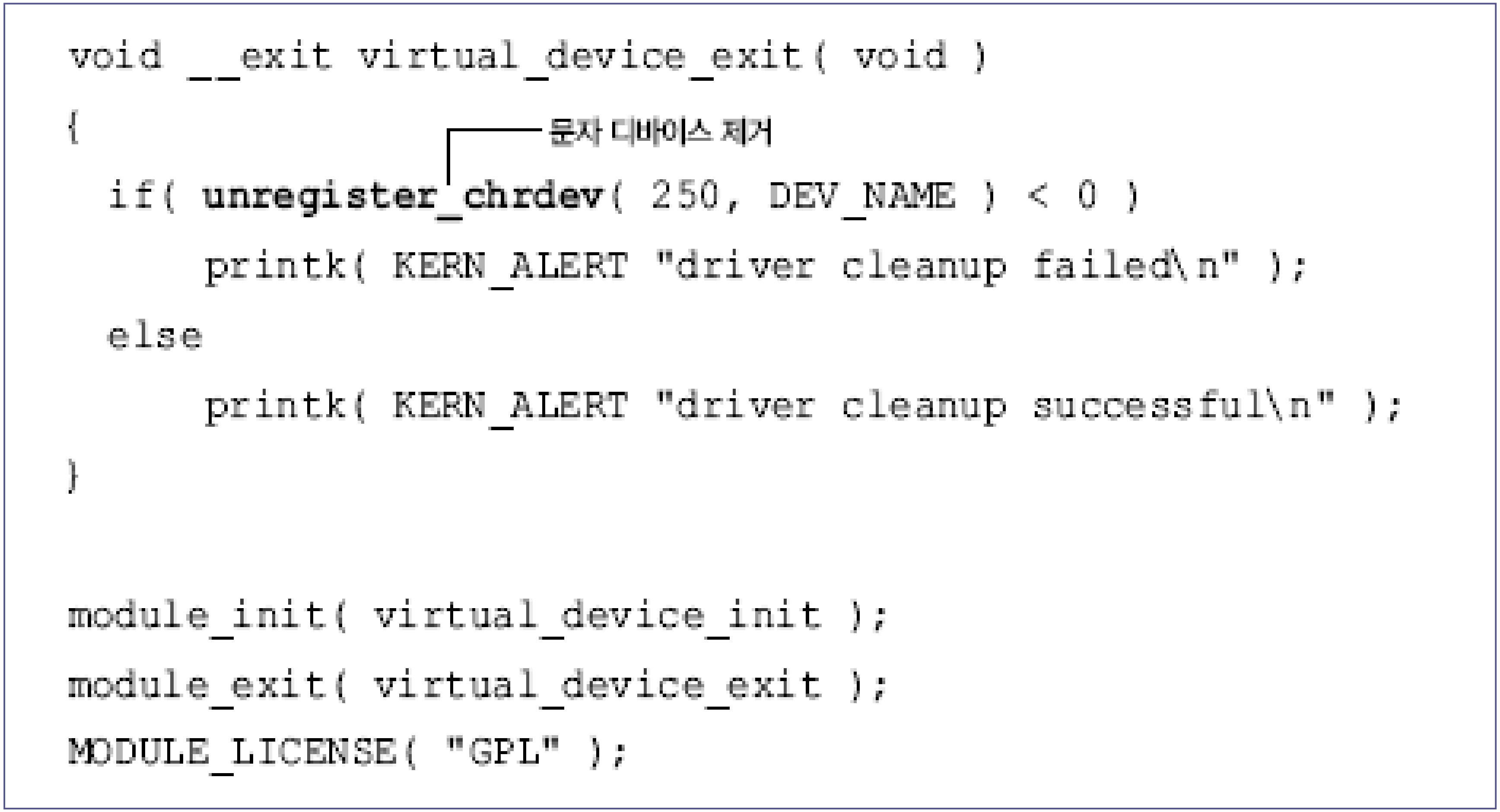
IOCTL device driver -7



```
static struct file_operations vd_fops = {
  .owner = THIS_MODULE,
  .open = virtual_device_open,
  .release = virtual_device_release,
  .ioctl = virtual_device_ioctl
};


int __init virtual_device_init( void )
{
  if( register_chrdev( 250, DEV_NAME, &vd_fops ) < 0 )
      printk( KERN_ALERT "driver init failed\n" );
  else
      printk( KERN_ALERT "driver init successful\n");
  return 0;
}
```

문자 디바이스 등록

IOCTL device driver-8

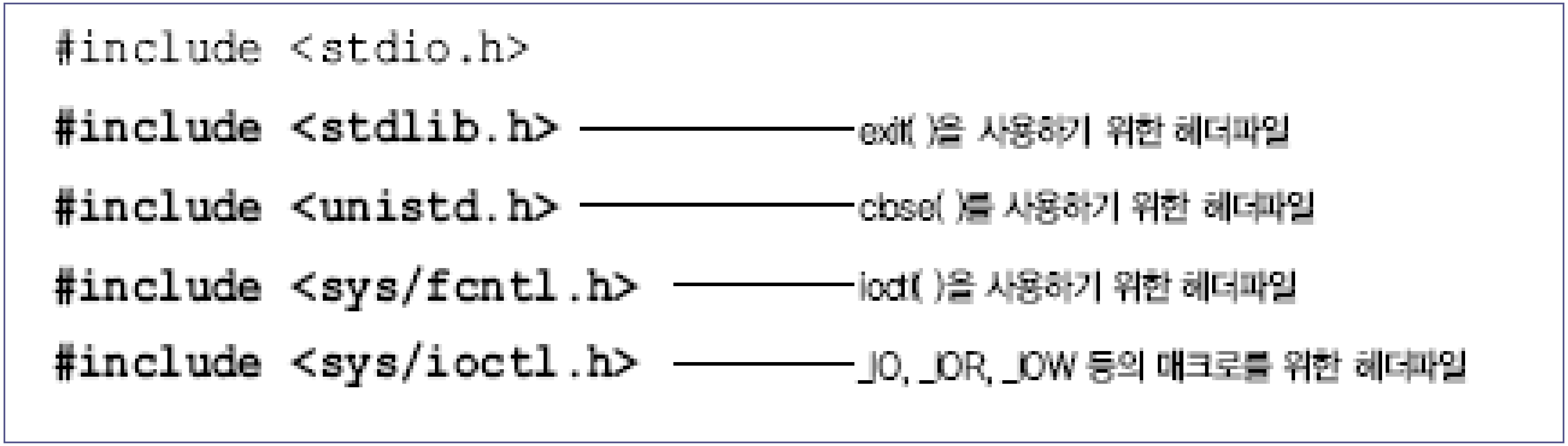


```
void __exit virtual_device_exit( void )
{
  if( unregister_chrdev( 250, DEV_NAME ) < 0 )
      printk( KERN_ALERT "driver cleanup failed\n" );
  else
      printk( KERN_ALERT "driver cleanup successful\n" );
}

module_init( virtual_device_init );
module_exit( virtual_device_exit );
MODULE_LICENSE( "GPL" );
```

문자 디바이스 제거

IOCTL application-1

- Header files



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/fcntl.h>
#include <sys/ioctl.h>
```

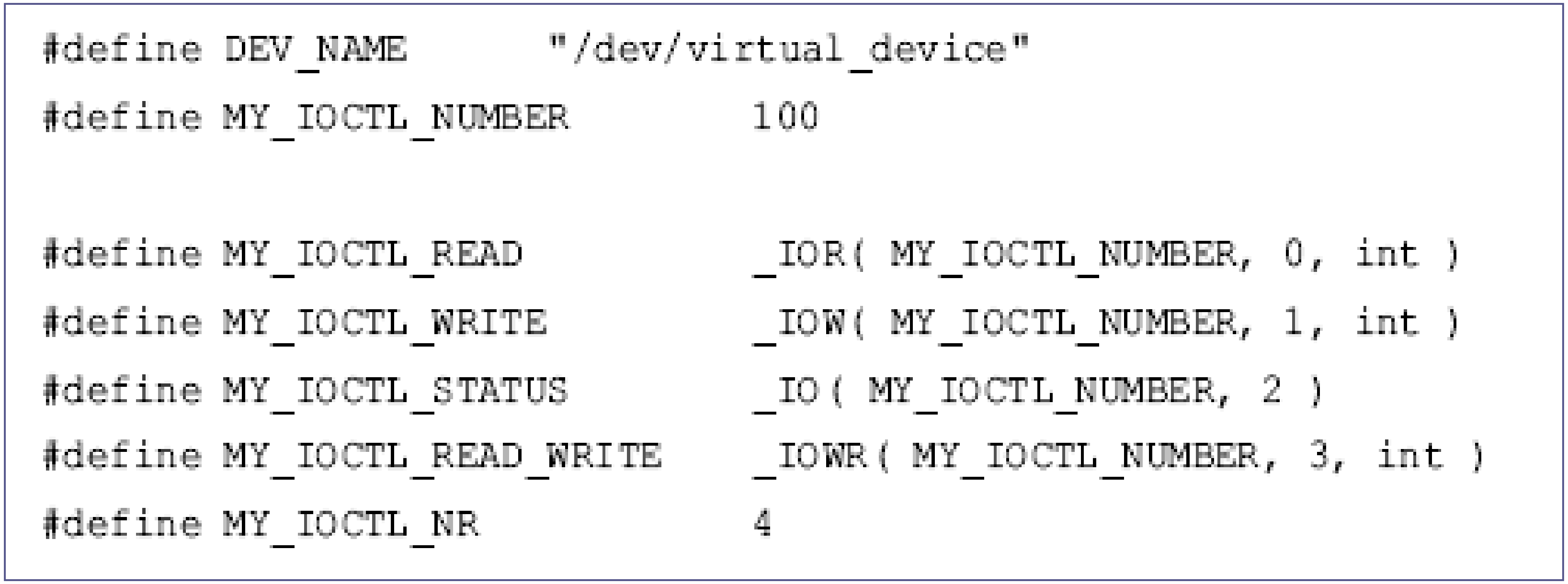
exit( )을 사용하기 위한 헤더파일

close( )를 사용하기 위한 헤더파일

ioctl( )을 사용하기 위한 헤더파일

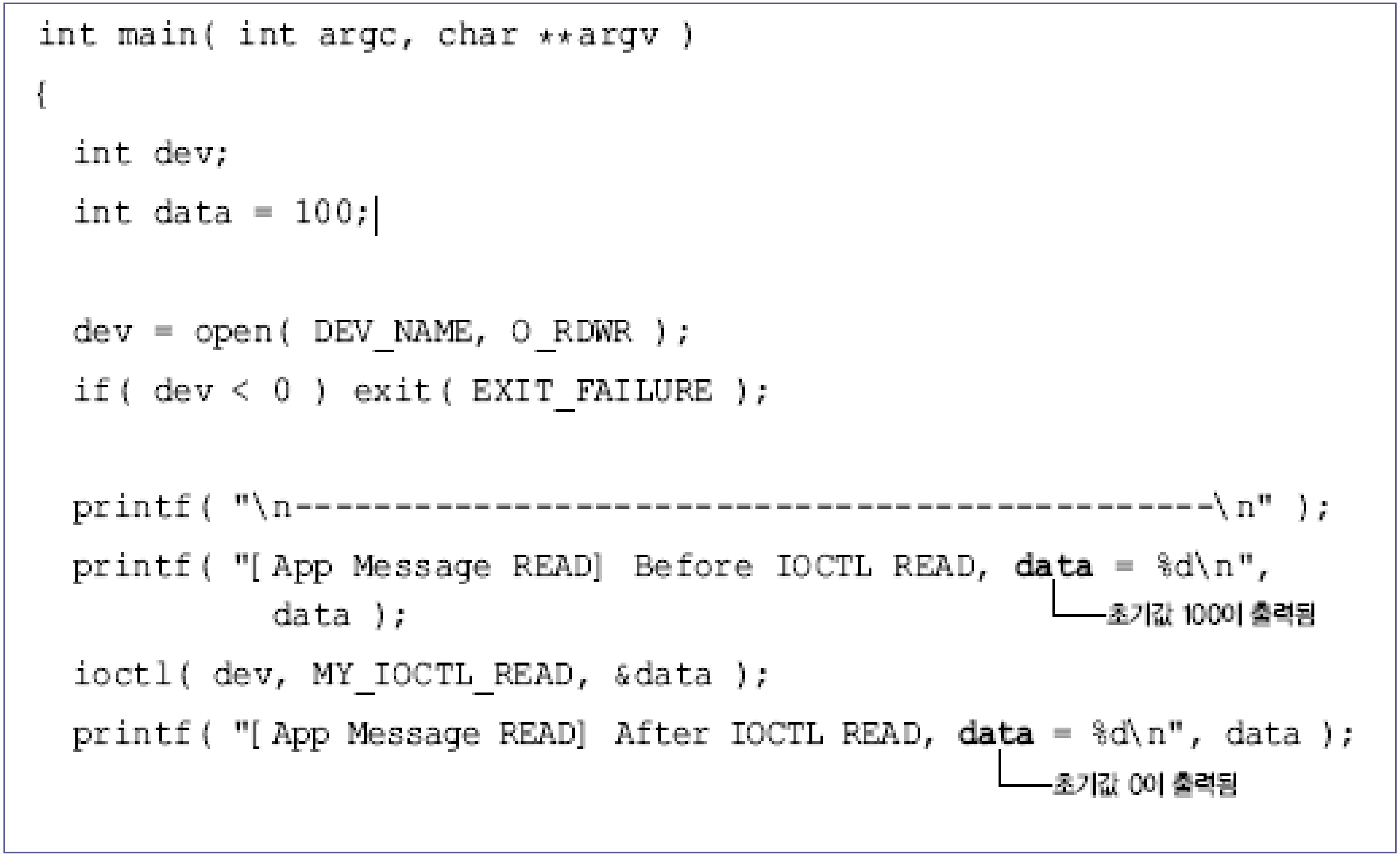
_IO, _IOR, _IOW 등의 매크로를 위한 헤더파일

IOCTL application-2



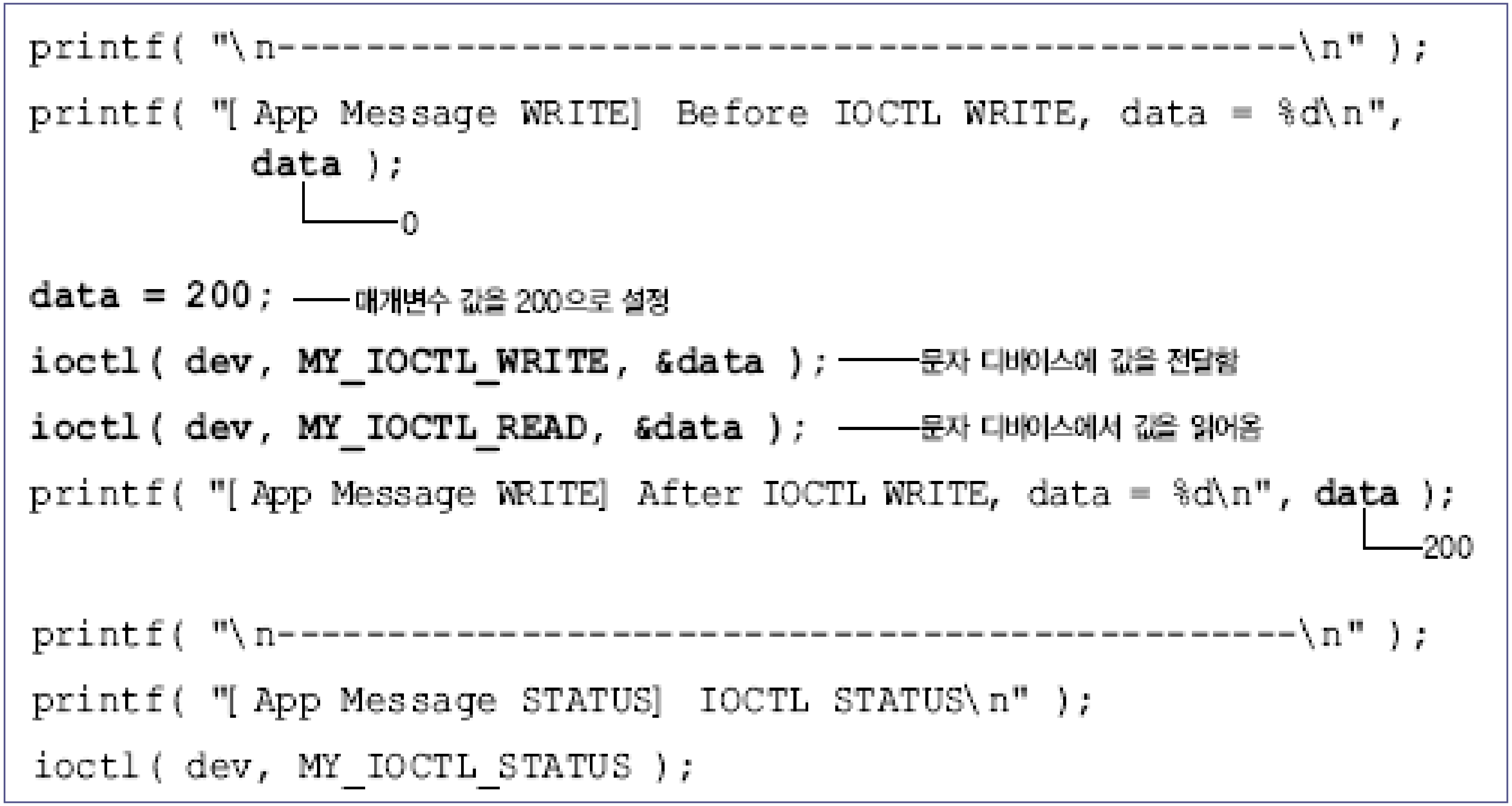
```
#define DEV_NAME      "/dev/virtual_device"
#define MY_IOCTL_NUMBER         100

#define MY_IOCTL_READ           _IOR( MY_IOCTL_NUMBER, 0, int )
#define MY_IOCTL_WRITE          _IOW( MY_IOCTL_NUMBER, 1, int )
#define MY_IOCTL_STATUS         _IO( MY_IOCTL_NUMBER, 2 )
#define MY_IOCTL_READ_WRITE     _IOWR( MY_IOCTL_NUMBER, 3, int )
#define MY_IOCTL_NR             4
```

IOCTL application-3



```
int main( int argc, char **argv )
{
  int dev;
  int data = 100;

  dev = open( DEV_NAME, O_RDWR );
  if( dev < 0 ) exit( EXIT_FAILURE );

  printf( "\n--------------------------------------------\n" );
  printf( "[App Message READ] Before IOCTL READ, data = %d\n",
          data );                                   초기값 100이 출력됨
  ioctl( dev, MY_IOCTL_READ, &data );
  printf( "[App Message READ] After IOCTL READ, data = %d\n", data );
                                                 초기값 0이 출력됨
```

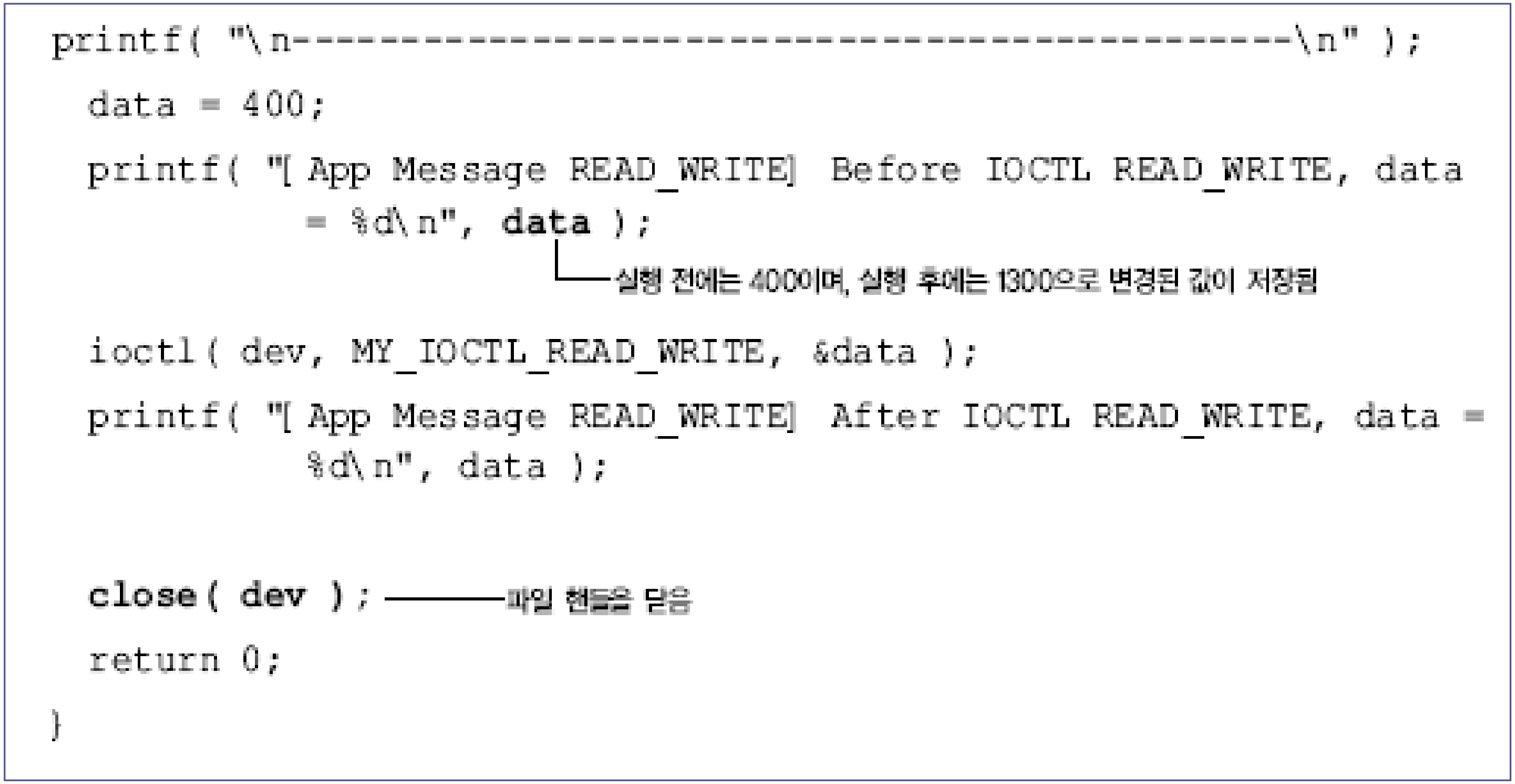
IOCTL application-4



```
printf( "\n-----------------------------------------\n" );
printf( "[ App Message WRITE]  Before IOCTL WRITE, data = %d\n",
            data );

data = 200;
ioctl( dev, MY_IOCTL_WRITE, &data );
ioctl( dev, MY_IOCTL_READ, &data );
printf( "[ App Message WRITE]  After IOCTL WRITE, data = %d\n", data );


printf( "\n-----------------------------------------\n" );
printf( "[ App Message STATUS]  IOCTL STATUS\n" );
ioctl( dev, MY_IOCTL_STATUS );
```

0
매개변수 값을 200으로 설정
문자 디바이스에 값을 전달함
문자 디바이스에서 값을 읽어옴
200

IOCTL application-5



```
printf( "\n------------------------------------------------\n" );
  data = 400;
  printf( "[ App Message READ_WRITE]  Before IOCTL READ_WRITE, data
            = %d\n", data );
```

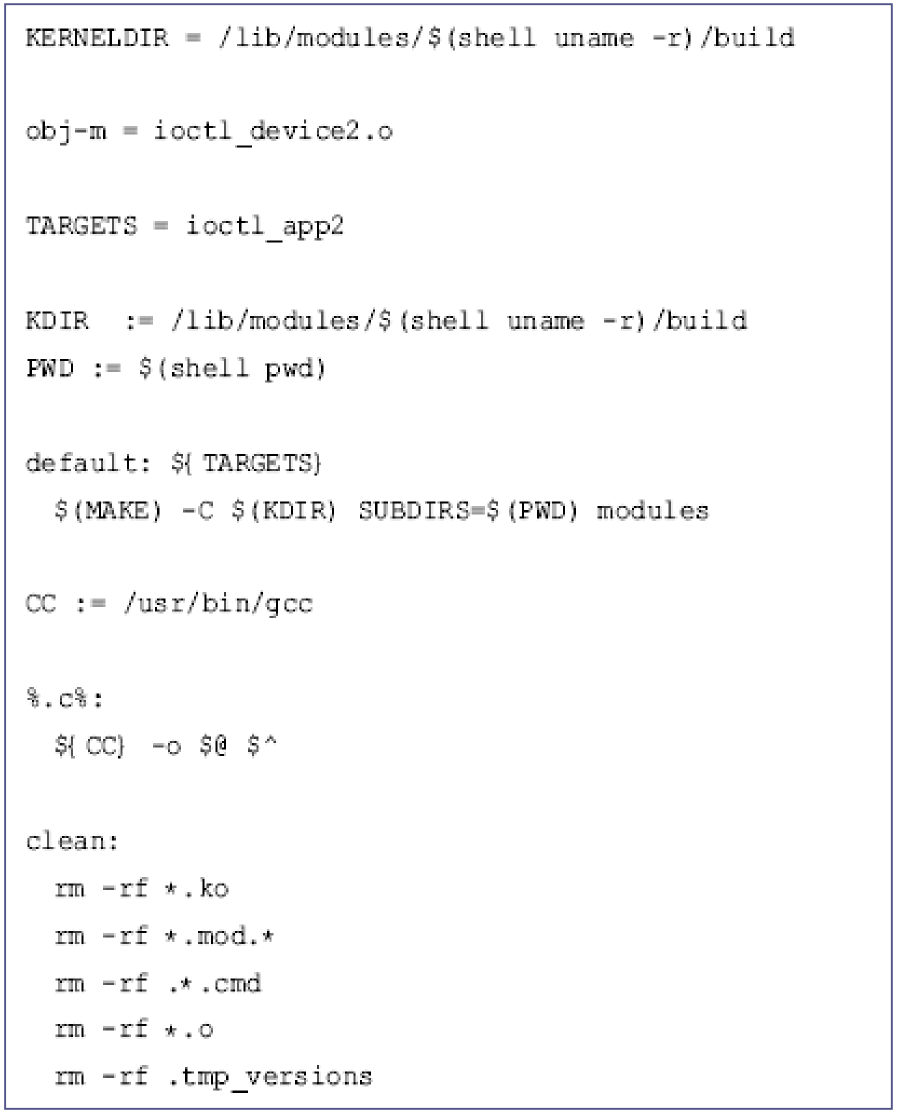
실행 전에는 400이며, 실행 후에는 1300으로 변경된 값이 저장됨

```
  ioctl( dev, MY_IOCTL_READ_WRITE, &data );
  printf( "[ App Message READ_WRITE]  After IOCTL READ_WRITE, data =
            %d\n", data );


  close( dev );
```

파일 핸들을 닫음

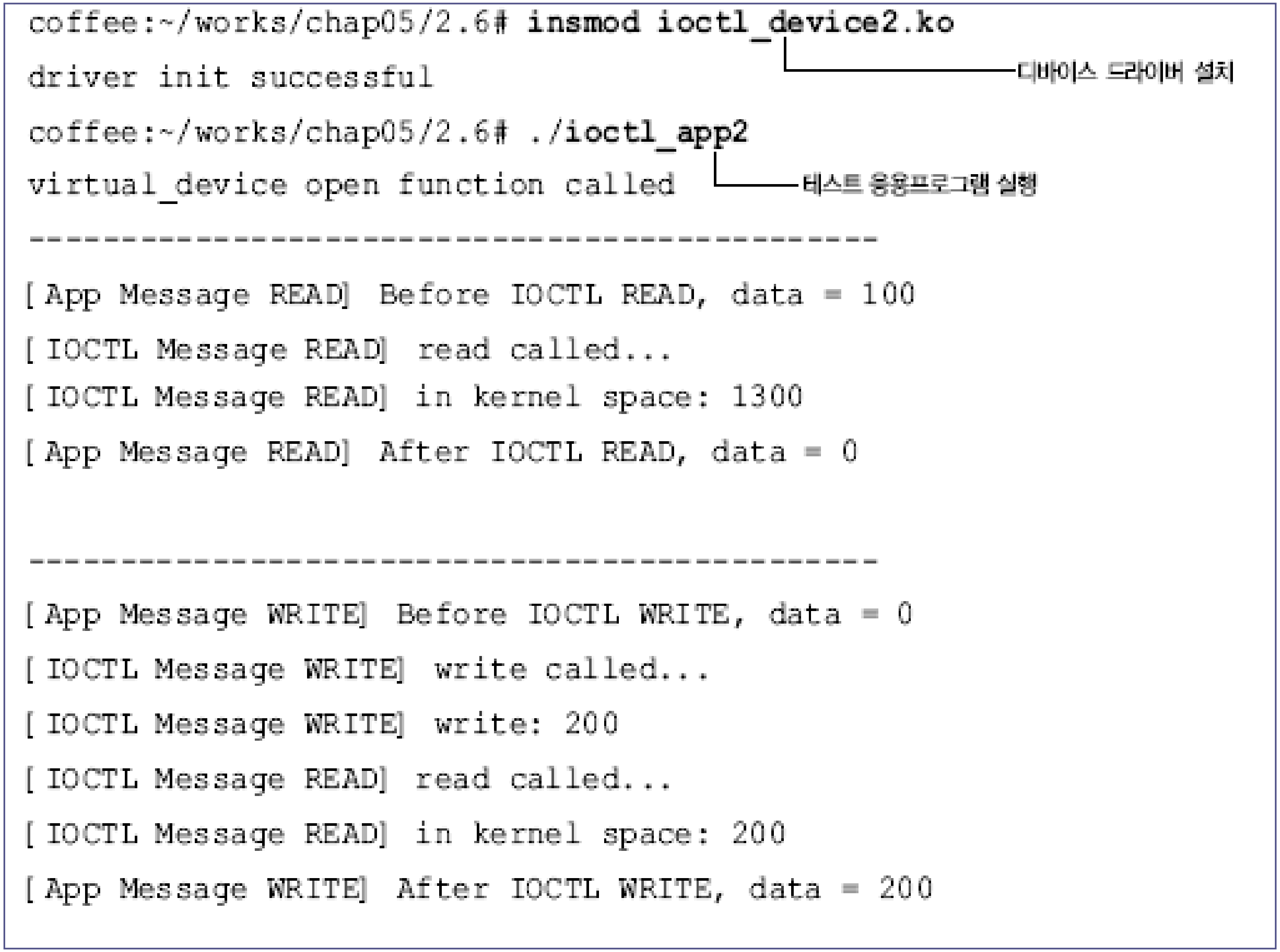
```
  return 0;
}
```

Makefile



```
KERNELDIR = /lib/modules/$(shell uname -r)/build

obj-m = ioctl_device2.o

TARGETS = ioctl_app2

KDIR  := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

default: ${ TARGETS}
  $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules

CC := /usr/bin/gcc

%.c%:
  ${ CC} -o $@ $^

clean:
  rm -rf *.ko
  rm -rf *.mod.*
  rm -rf .*.cmd
  rm -rf *.o
  rm -rf .tmp_versions
```

IOCTL test-1

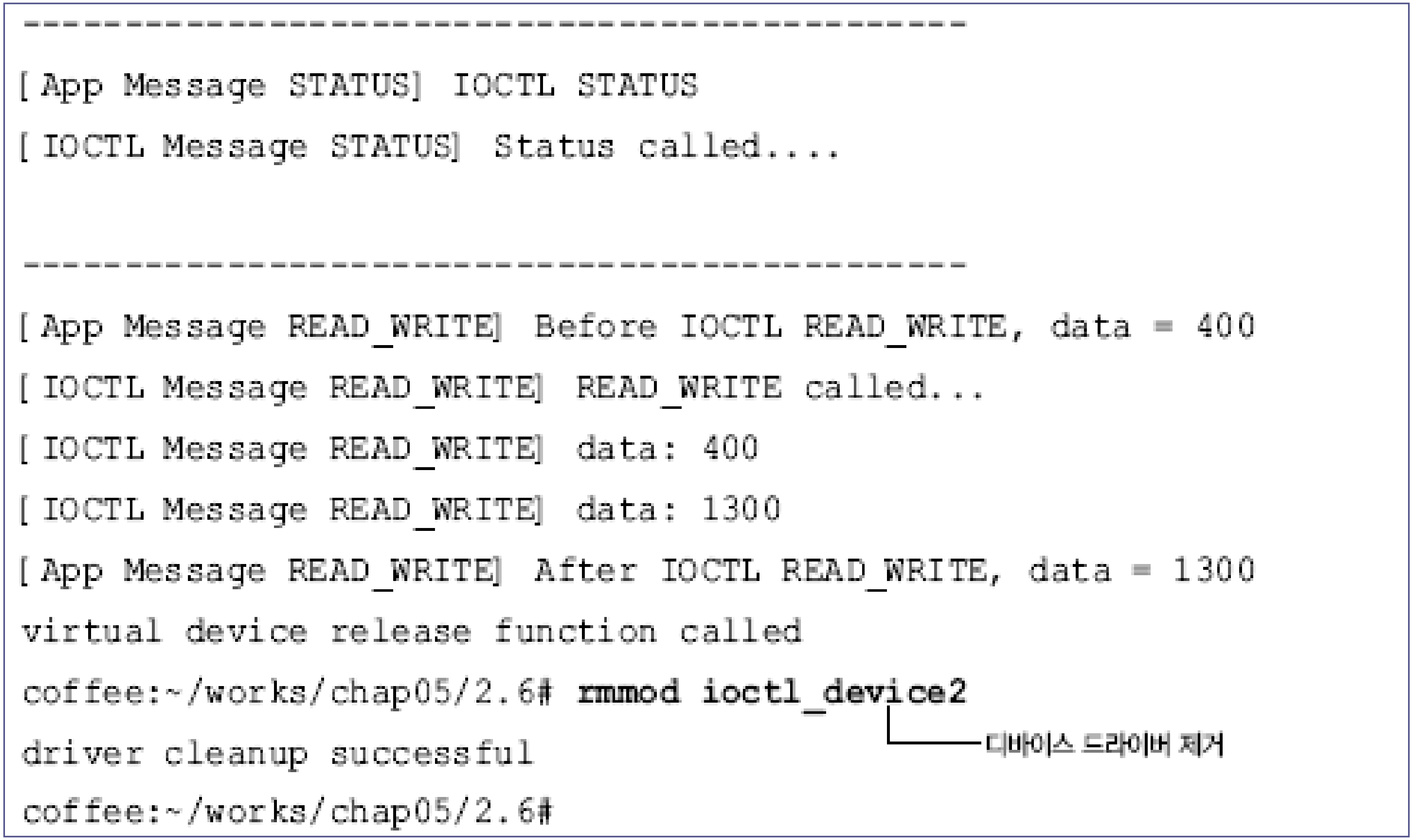


```
coffee:~/works/chap05/2.6# insmod ioctl_device2.ko
driver init successful
coffee:~/works/chap05/2.6# ./ioctl_app2
virtual_device open function called
----------------------------------------------
[App Message READ]  Before IOCTL READ, data = 100
[IOCTL Message READ]  read called...
[IOCTL Message READ]  in kernel space: 1300
[App Message READ]  After IOCTL READ, data = 0


----------------------------------------------
[App Message WRITE]  Before IOCTL WRITE, data = 0
[IOCTL Message WRITE]  write called...
[IOCTL Message WRITE]  write: 200
[IOCTL Message READ]  read called...
[IOCTL Message READ]  in kernel space: 200
[App Message WRITE]  After IOCTL WRITE, data = 200
```

디바이스 드라이버 설치

테스트 응용프로그램 실행

IOCTL test-2



```
----------------------------------------------
[App Message STATUS] IOCTL STATUS
[IOCTL Message STATUS] Status called....


----------------------------------------------
[App Message READ_WRITE] Before IOCTL READ_WRITE, data = 400
[IOCTL Message READ_WRITE] READ_WRITE called...
[IOCTL Message READ_WRITE] data: 400
[IOCTL Message READ_WRITE] data: 1300
[App Message READ_WRITE] After IOCTL READ_WRITE, data = 1300
virtual device release function called
coffee:~/works/chap05/2.6# rmmod ioctl_device2
driver cleanup successful
coffee:~/works/chap05/2.6#
```

디바이스 드라이버 제거