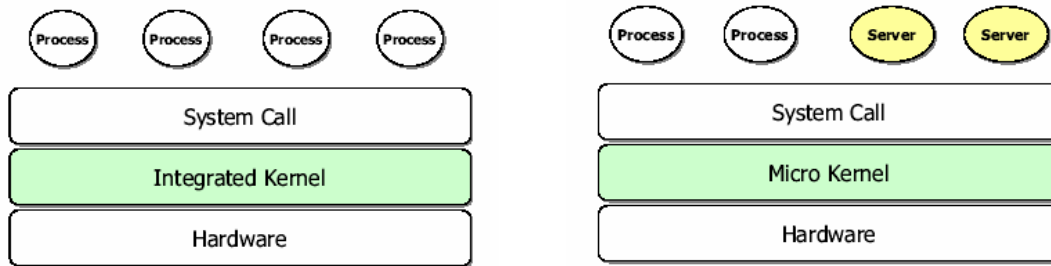


디바이스 드라이버 프로그래밍 소개

- 모듈 프로그래밍
- 디바이스 드라이버
- 가상 문자 디바이스 드라이버 예

모듈 프로그래밍

커널의 종류



□ Monolithic Kernel

- 커널의 모든 기능이 하나의 실행 파일로 구성됨
- 유연하지 못함
- Linux, 대부분의 사용 UNIX

□ Micro Kernel

- 커널은 극히 제한 적인 작업만을 수행
 - 서버와 프로세스 간의 통신, H/W 제어
- 파일시스템 디바이스 드라이버 등의 기능은 서버로 구현
- 유연하지만 통신 오버헤드가 심함
- Windows

모듈(module) 소개 (1)

□ 리눅스의 특징 중 하나는 커널이 동작 중에 기능을 추가

- 수행 중에 커널에 부가되는 코드의 부분을 **모듈**
 - 실행중인 커널에 동적으로 적재되거나 제거
- 하나의 오브젝트 파일(*.o)
- 이벤트 처리(Event handling) 형태의 프로그램 방식
 - main() 함수가 없다
 - startup, cleanup 함수 존재
- 주로 파일 시스템과 디바이스 드라이버 구현 시 모듈 프로그래밍
- 각 모듈은 동적으로 링크될 수 있는 목적코드로 **insmod** 로 커널에 링크되고 **rmmmod** 로 링크가 해제

모듈 소개 (2)

□ 모듈 프로그램의 이점

- 효과적인 메모리 사용
- 커널 전체를 다시 컴파일하지 않고 커널의 일부분 또는 디바이스 드라이버를 교체할 수 있음

□ 모듈 프로그램의 단점

- namespace pollution
- 커널에 삽입되어 확장된 커널이 됨
- 기존 커널의 광역변수 혹은 함수명이 충돌 가능성

모듈 프로그래밍의 예 - hello.c

```

/* Module example -hello.c */

#include <linux/kernel.h> /* 커널에서 수행될 때 필요한 헤더 파일 */
#include <linux/module.h> /* 모듈에 필요한 헤더 파일 */
#include <linux/init.h> /* module_init(), module_exit() 매크로 정의 */

int module_begin(void) /* 모듈이 설치될 때 초기화 수행,
                       insmod 할 때 수행되는 함수 */
{
    printk("<1> Hello, world\n");
    return 0;
}

void module_end(void) /* 모듈이 제거될 때 반환 작업 수행,
                      rmmod 할 때 수행되는 함수 */
{
    printk("<1> Goodbye, world\n");
}

module_init(module_begin);
module_exit(module_end);

```

□ 모듈 등록 및 해제

- 등록 시 `module_begin()` 실행
- 해제 시 `module_end()` 실행

□ 예제 코드

- `insmod`로 모듈을 로드하고 커널에 링크된 후 `printk()`를 호출
- `rmmod`로 모듈을 제거할 때도 `printk()`를 호출

□ `printk()` 함수

- 리눅스 커널은 C 라이브러리 없이 자체 고유의 함수 지원
- 따라서 `printf()`는 지원되지 않고 비슷한 기능의 `printk()` 함수 지원
- `<1>`은 메시지의 우선순위를 나타내며 번호가 낮을수록 우선순위가 높음
- 콘솔에서 실행된 경우 화면에 출력되고 `xterm` 등에서는 `/var/log/messages` 등과 같은 시스템 로그 파일에 출력

모듈 프로그래밍의 예 - Makefile

```
# Makefile for a basic kernel module
```

```
CC := gcc      # arm-linux-gcc, 크로스컴파일러
```

```
# Modify this statement to your kernel directory
INCLUDEDIR := /usr/src/linux-2.4.20-8/include/
```

```
MODCFLAGS := -Wall -O2 -DMODULE -D__KERNEL__ -DLINUX -I$(INCLUDEDIR)
```

```
hello.o : hello.c
        $(CC) $(MODCFLAGS) -c hello.c
```

```
# make
# /sbin/insmod hello.o
# /sbin/lsmod
# /sbin/rmmod hello
# tail /var/log/messages
```

```
<- 모듈 컴파일
<- 모듈을 커널에 적재
<- 적재된 모듈 보기
<- 모듈 삭제하기
<- 로그 파일 출력
```

```

slee@localhost: /home/slee/test
[root@localhost test]# make
gcc -Wall -O2 -DMODULE -D__KERNEL__ -DLINUX -I/usr/src/linux-2.4.20-8/include/ -c hello.c
[root@localhost test]# /sbin/insmod hello.o
Warning: loading hello.o will taint the kernel: no license
See http://www.tux.org/ikml/#export-tainted for information about tainted modules
Module hello loaded, with warnings
[root@localhost test]# /sbin/lsmod
Module                Size  Used by    Tainted: PF
hello                  792    0 (unused)
parport_pc            19076    1 (autoclean)
lp                    8996    0 (autoclean)
parport               37056    1 (autoclean) [parport_pc lp]
autofs                13268    0 (autoclean) (unused)
vmhgfs                38380    4
vmxnet                 8236    1
ipt_REJECT             3928    6 (autoclean)
iptables_filter        2412    1 (autoclean)
ip_tables             15096    2 [ipt_REJECT iptable_filter]
sg                     36524    0 (autoclean)
sr_mod                18136    0 (autoclean)
ide-scsi              12208    0
ide-cd                35708    0
cdrom                 33728    0 [sr_mod ide-cd]
keybdev               2944    0 (unused)
mousedev              5492    1
hid                   22148    0 (unused)
input                 5856    0 [keybdev mousedev hid]
usb-uhci              26348    0 (unused)
usbcore               78784    1 [hid usb-uhci]
ext3                  70784    2
jbd                   51892    2 [ext3]
BusLogic              100796    3
sd_mod                13452    6
scsi_mod              107128    5 [sg sr_mod ide-scsi BusLogic sd_mod]
[root@localhost test]# /sbin/rmmod hello
[root@localhost test]# tail /var/log/messages
Oct 24 23:39:55 localhost syslogd 1.4.1: restart.
Oct 25 08:51:45 localhost vsftpd: warning: can't get client address: Bad file descriptor
Oct 25 09:06:06 localhost kernel: Hello, world
Oct 25 09:06:27 localhost kernel: Goodbye, world
Oct 25 09:18:07 localhost kernel: Hello, world
Oct 25 09:18:39 localhost kernel: Goodbye, world
[root@localhost test]#

```

버 프로그래밍 소개

명령어	용도
insmod	모듈을 커널에 적재하여 실행
rmmod	실행중인 모듈을 커널에서 제거
lsmod	현재 커널에 적재된 모듈들 정보를 표시
depmod	커널 내부에 적재된 모듈간의 의존성을 검사
modprobe	insmod와 유사하나, 모듈간 의존성을 검사하여 그 결과 누락된 다른 모듈을 찾아서 적재한다는 점에서 차이
modinfo	목적화일을 검사해서 관련된 정보를 표시

디바이스 드라이버

디바이스 드라이버란?

- ❑ 응용프로그램이 물리적인 하드웨어 장치를 제어 관리하도록 인터페이스를 제공하는 코드
- ❑ 종류
 - 문자 디바이스 드라이버 (Character Device Driver)
 - 블록 디바이스 드라이버 (Block Device Driver)
 - 네트워크 디바이스 드라이버 (Network Device Driver)

디바이스 종류 (1)

□ 문자 디바이스(character device)

- 파일과 같이 바이트 단위로 입출력하고 `open`, `close`, `read`, `write` 시스템 콜로 구현
- 텍스트 콘솔(/dev/console), 시리얼포트(/dev/ttyS0) 등
- 문자 디바이스는 /dev/tty1, /dev/lp0 등과 같은 파일 시스템 노드를 통하여 접근
- 파일 시스템과 다른 점은 문자 디바이스는 데이터 채널로써 순차적으로만(sequential) 접근 가능

디바이스 종류 (2)

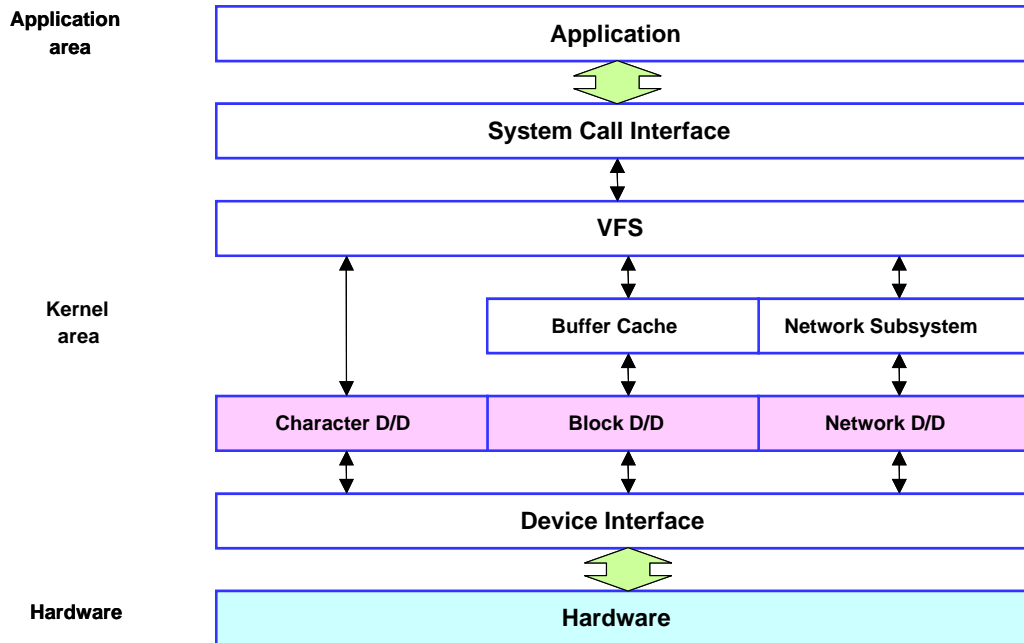
□ 블록 디바이스(block device)

- 블록 디바이스는 2의 멍승의 Kbyte 크기로 표시되는 블록 단위로 접근되는 장치로 파일 시스템이 놓이는 디스크와 같은 장치
- 문자 디바이스와 같은 커널 인터페이스를 갖는다.

□ 네트워크 인터페이스(network interface)

- 네트워크 인터페이스는 데이터 패킷의 송수신을 담당
- 개개의 패킷의 응용을 구분하지는 못함
- 네트워크 인터페이스는 파일 시스템의 노드로 표현되지 않고 `eth0` 등과 같은 고유의 이름을 가짐
- 커널과 네트워크 인터페이스 간의 통신은 문자나 블록 디바이스와는 완전히 다름
 - `read`, `write` 대신에 패킷전송과 관련된 함수들을 호출하여 사용

리눅스 계층 구조



디바이스 파일(노드) (1)

- 디바이스들은 파일 시스템 트리 상에서 특수 파일 또는 **디바이스 파일** 또는 단순히 **노드**라 불리는 이름을 통하여 접근

```

$ ls -l /dev
crw-rw-rw- 1 root    root      1, 3      Feb 23 1999 null
crw----- 1 root    root     10, 1     Feb 23 1999 psaux
crw----- 1 rubini  tty       4, 1     Aug 16 22:22 ttyl
crw-rw-rw- 1 root    dialout   4, 64     Jun 30 11:19 ttyS0
crw-rw-rw- 1 root    dialout   4, 65     Aug 16 00:00 ttyS1
crw----- 1 root    sys       7, 1     Feb 23 1999 vcs1
crw----- 1 root    sys       7, 129   Feb 23 1999 vcsa1
crw-rw-rw- 1 root    root      1, 5     Feb 23 1999 zero
  
```

- 위에서 날짜 전에 표시된 숫자는 디바이스의 **주(major)**, **부(minor)** 번호
 - 주 번호는 디바이스와 관련된 드라이버를 표시, 디바이스 유형
 - 부 번호는 주 번호에 의해 기술된 드라이버에서만 사용, 디바이스 단위
 - 드라이버 여러 개의 디바이스를 제어하는 경우 이들을 구분하기 위해 부 번호가 사용

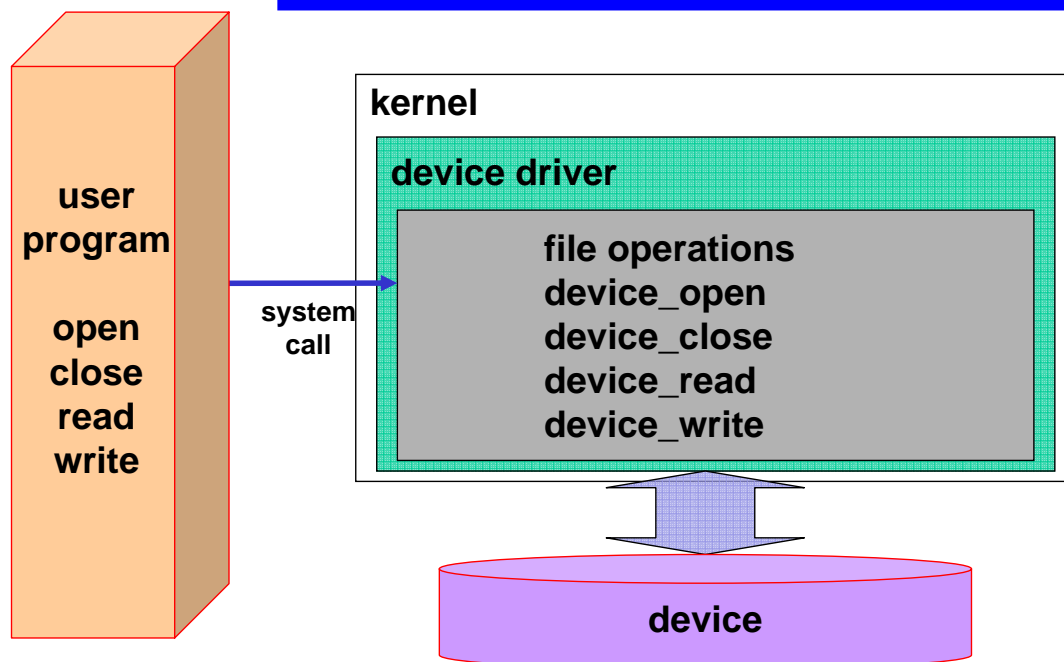
디바이스 파일(노드) 생성

- ❑ # mknod “/dev/파일이름” {b,c} 주번호 부번호
 - 예) # mknod /dev/led_dd c 124 0
 - 생성 후 속성변경 : #chmod ug+w /dev/led_dd

디바이스 드라이버의 개발 프로세스



디바이스 드라이버 시스템 콜

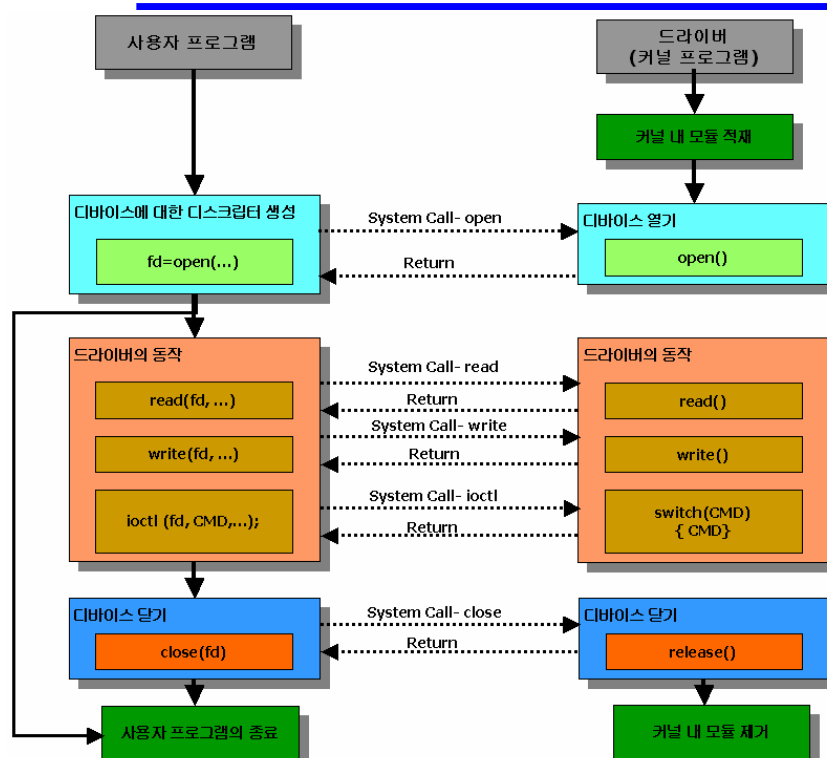


가상 문자 디바이스 드라이버 예

vtest 가상 문자 디바이스 드라이버

- ❑ 응용 프로그램과 디바이스 드라이버 간의 동작 관계를 보여주는 예제
 - 디바이스 드라이버: vtest_dev.c
 - 응용 프로그램: vtest_app.c
 - 디바이스 파일
 - /dev/vtest
 - 주번호 240, 부번호 0
- ❑ 드라이버 동작 예
 - 드라이버 등록하여 주번호 할당
 - open, read, ioctl, close 파일 오퍼레이션
 - 응용 프로그램, 디바이스 파일, 드라이버의 인터페이스 관계

디바이스 드라이버 처리과정 개념도



디바이스 등록 - 주 번호 할당 (1)

- ❑ 새로운 드라이버를 추가하는 것은 주 번호 할당을 의미
- ❑ `<linux/fs.h>`에 정의된 다음 함수가 드라이버(모듈) 초기화 시 주 번호를 할당


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

 - 할당 성공 시 0 또는 양수를, 실패 시 음수를 리턴
 - `major`는 할당하고자 하는 주 번호
 - `name`은 `/proc/devices`에 표시되는 디바이스의 이름
 - `fops`는 드라이버의 엔트리 포인트를 실행하는데 사용되는 함수의 포인터 배열의 포인터
- ❑ 주 번호는 char 드라이버의 정적배열의 인덱스로 사용
 - 8비트로 표현되며 2.0에서는 128개, 2.2와 2.4에서는 256개(0~255)까지 할당
 - 부 번호 역시 8비트로 표현

디바이스 등록 - 주 번호 할당 (2)

- ❑ 드라이버가 커널 테이블에 등록되면 등록된 주 번호와 관련하여 연산이 수행
 - 주 번호와 관련된 문자 디바이스 파일 상에 연산이 수행될 때마다 커널은 `file_operations` 구조체로부터 해당 함수를 찾아서 실행
- ❑ 파일 시스템에서 디바이스 노드를 생성하기 위해서는 `mknod` 명령을 사용
 - 이는 관리자 권한으로 수행되며 다음과 같이 3개의 인수를 기술


```
mknod /dev/vtest c 240 0
```

 - 주 번호 240, 부 번호 0, `vtest`의 이름을 갖는 char 디바이스 파일을 생성
 - 제거 시에는 `rm /dev/vtest` 명령을 사용

vtest_dev.c 예 - 디바이스 등록 및 해제

```
#define VTEST_DEV_NAME      "vtest"
#define VTEST_DEV_MAJOR    240

int vtest_init(void)
{
    int result;

    result = register_chrdev( VTEST_DEV_MAJOR, VTEST_DEV_NAME,&vtest_fops);
    if (result < 0) return result;

    return 0;
}
void vtest_end(void)
{
    unregister_chrdev(VTEST_DEV_MAJOR, VTEST_DEV_NAME );
}
module_init(vtest_init);
module_exit(vtest_end);
```

파일 오퍼레이션 구조체

- ❑ 디바이스는 내부적으로 파일 구조체로 표현
- ❑ 커널은 **file_operation 구조체**를 사용하여 드라이버의 함수(메소드)에 접근
 - 이 구조체는 <linux/fs.h>에 함수(메소드)의 포인터 배열로 정의
 - 각 메소드는 응용 프로그램의 시스템 콜의 엔트리 포인트
 - 구조체의 각 필드는 드라이버의 특정 기능을 구현한 함수의 포인터
 - 이들 필드는 태그화된 초기화(tagged initialization)을 통하여 이름이 기술되어 초기화
 - 태그화된 초기화는 표준 C는 아니고 GNU 컴파일러에서 확장

메소드 (1)

□ 다음은 `struct file_operations` 의 각 필드에서 표현되는 함수(메소드)의 포인터

- 성공인 경우에 0, 함수가 실패한 경우에는 음수를 리턴
- `loff_t (*llseek) (struct file *, loff_t, int);`
llseek 함수는 파일의 현재 읽기/쓰기 위치를 변경하여 새로운 위치를 리턴
loff_t 는 “long offset”을 의미하며 적어도 64 비트로 표현
- `ssize_t (*read) (struct file *, char *, size_t, loff_t *);`
디바이스로부터 데이터를 읽음.
실패 시 -EINVAL, 성공하면 읽어들이는 바이트 수를 리턴
- `ssize_t (*write) (struct file *, const char *, size_t, loff_t *);`
디바이스로 데이터 쓰기를 수행.
실패 시 -EINVAL, 성공하면 데이터 쓰기의 바이트 수를 리턴
- `int (*readdir) (struct file *, void *, filldir_t);`
디바이스 파일인 경우 이 필드는 NULL이 됨.
파일 시스템에서 디렉토리를 읽기 위해 사용
- `unsigned int (*poll) (struct file *, struct poll_table_struct *);`
디바이스 상태를 조사하는 함수로 디바이스가 읽기 가능, 쓰기 가능 또는 특수한 상태에 있는지를 조사.
디바이스의 상태를 기술하는 비트 마스크를 리턴

메소드 (2)

- `int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);`
디바이스 관련 특정 명령을 이슈(예를들어 플로피 디스크 포맷 등과 같이 읽기, 쓰기가 아닌 명령들).
실패 시 -ENOTTY, 성공 시 양수를 리턴
- `int (*mmap) (struct file *, struct vm_area_struct *);`
디바이스 메모리를 프로세스의 주소 공간으로 매핑.
실패 시 -ENODEV를 리턴
- `int (*open) (struct inode *, struct file *);`
디바이스 파일 상에 수행되는 첫번째 연산으로 디바이스 오픈
- `int (*flush) (struct file *);`
프로세스가 디바이스의 파일 디스크립터를 종결(close)할 때 호출.
현재는 네트워크 파일 시스템(NFS)에서만 사용
- `int (*release) (struct inode *, struct file *);`
파일 구조체가 해제될 때 호출된다.
- `int (*fsync) (struct inode *, struct dentry *, int);`
현재 보류된 데이터를 배출(flush)하고자 할 때 사용

메소드 (3)

- `int (*lock) (struct file *, int, struct file_lock *)`;
파일의 잠금 시 사용되는데 디바이스 드라이버에서는 거의 구현되지 않음
- `ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *)`;
`ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *)`;
분산된 여러 메모리 영역들에 대해 한 번의 읽기/쓰기를 수행
- `struct module *owner`;
이 필드는 이 `file_operation` 구조체를 소유하는 모듈의 포인터. 커널이 모듈의 사용횟수를 관리하기 위해 사용

vtest_dev.c 예 - 파일 오퍼레이션 초기화

□ 메소드들은 태그화된 초기화가 되어 선언

```
struct file_operations vtest_fops =
{
    open    : vtest_open,
    release : vtest_release,
    read    : vtest_read,
    write   : vtest_write,
    ioctl   : vtest_ioctl
};
```

파일 구조체 (File Structure)

□ struct file 구조체

- <linux/fs.h>에서 정의된 디바이스 드라이버에서 두번째로 중요한 자료구조
 - C 사용자 프로그램의 FILE 자료구조와는 별개의 것
- file 구조체는 **open file**을 표현
 - open file은 struct inode 로 표현되는 디스크 파일을 다룸
- open file은 커널 상에서 open 메소드(함수)로 생성되어 해당 open file 상에서 동작하는 함수로 전달되고 close로 해제
- 커널 소스에서 struct file의 포인터를 **flip("file pointer")**로 표시

struct file의 주요 필드 (1)

□ mode_t f_mode;

- 파일의 모드는 파일이 읽기 가능 또는 쓰기 가능인지를 비트 FMODE_READ, FMODE_WRITE로 표현

□ loff_t f_pos;

- 현재 읽기와 쓰기의 위치를 표시
- loff_t는 64비트 값이다(gcc 용어로는 long long)

□ unsigned int f_flags;

- 비블록킹(nonblocking) 여부 등을 체크하는 파일 플래그로써 O_RDONLY, O_NONBLOCK, O_SYNC 등으로 표시
- 모든 플래그는 <linux/fcntl.h>에서 정의

□ struct file_operations *f_op;

- 파일과 관련된 오퍼레이션들의 포인터
- 커널은 open 구현의 결과로 이 포인터를 지정
- 같은 주 번호를 갖는 드라이버의 open 시 flip->f_op 는 **오픈되는 부 번호**에 따라 지정

struct file의 주요 필드 (2)

❑ void *private_data;

- 드라이버는 자유롭게 이 필드를 사용하여 데이터를 할당하여 시스템 콜 간에 상태 정보 등을 저장하는 공간을 확보
- file 구조체가 커널에 의해 해제되기 전에 이 포인터가 가리키는 메모리 영역을 해제해야 함

❑ struct dentry *f_dentry;

- 파일과 관련된 디렉토리 엔트리(dentry)의 포인터
- 드라이버가 flip->f_dentry->d_inode 로 inode 구조체에 접근하기 위해 사용

open, release 메소드

❑ 드라이버는 open 메소드를 사용하여 다음과 같은 초기화를 수행

- 사용횟수(usage count)를 증가
- 디바이스 준비여부(device-not-ready), 하드웨어 문제 등과 같은 디바이스 관련 에러를 체크
- 처음 오픈된 경우 디바이스를 초기화
- 부 번호를 인지하고 필요하면 f_op 포인터를 갱신
- 특정 자료구조를 위해 메모리를 할당하고 flip->private_data 에 저장

❑ release 메소드는 다음 기능을 수행

- open 시 flip->private_data 에 할당된 메모리를 해제
- 마지막 release인 경우 디바이스를 종료(shut down)
- 모듈의 사용횟수를 감소

vtest_dev.c 예 – open, release 메소드

```

int vtest_open (struct inode *inode, struct file *filp)
{
    MOD_INC_USE_COUNT;
    return 0;
}

int vtest_release (struct inode *inode, struct file *filp)
{
    MOD_DEC_USE_COUNT;
    return 0;
}

```

read, write 메소드 프로토타입

- ❑ `ssize_t read(struct file *filp, char *buff, size_t count, loff_t *offp);`
`ssize_t write(struct file *filp, const char *buff, size_t count, loff_t *offp);`
 - filp는 파일 포인터
 - count는 전송되는 데이터의 크기
 - buff는 송수신되는 사용자 버퍼
 - offp는 사용자가 액세스하는 파일의 위치를 표시
 - “signed size type”을 리턴
- ❑ 이들 메소드는 **커널 주소공간**과 **사용자 주소공간** 간에 데이터를 송수신
 - 사용자 주소공간은 **스와핑(swapped out)**이 된다는 점에서 커널 주소공간과 차이
 - 따라서 단순 포인터나 memcpy를 통해 송수신할 수는 없음
 - 커널이 사용자 영역 포인터를 액세스할 때 해당 페이지가 메모리에 없는 경우 페이지 폴트가 발생

사용자 공간과 커널 공간과의 데이터 전송

- ❑ 리눅스에서는 사용자 공간과 커널 공간과의 전송은 `<asm/uaccess.h>`에 정의된 특수 함수들에 의해 수행
 - `unsigned long copy_to_user(void *to, const void *from, unsigned long count);`
 - `unsigned long copy_from_user(void *to, const void *from, unsigned long count);`
- ❑ **read 메소드**는 디바이스에서 사용자 공간으로 데이터를 복사(`copy_to_user` 사용)
- ❑ **write 메소드**는 사용자 공간에서 디바이스로 데이터를 복사(`copy_from_user` 사용).

read 메소드 인수의 사용

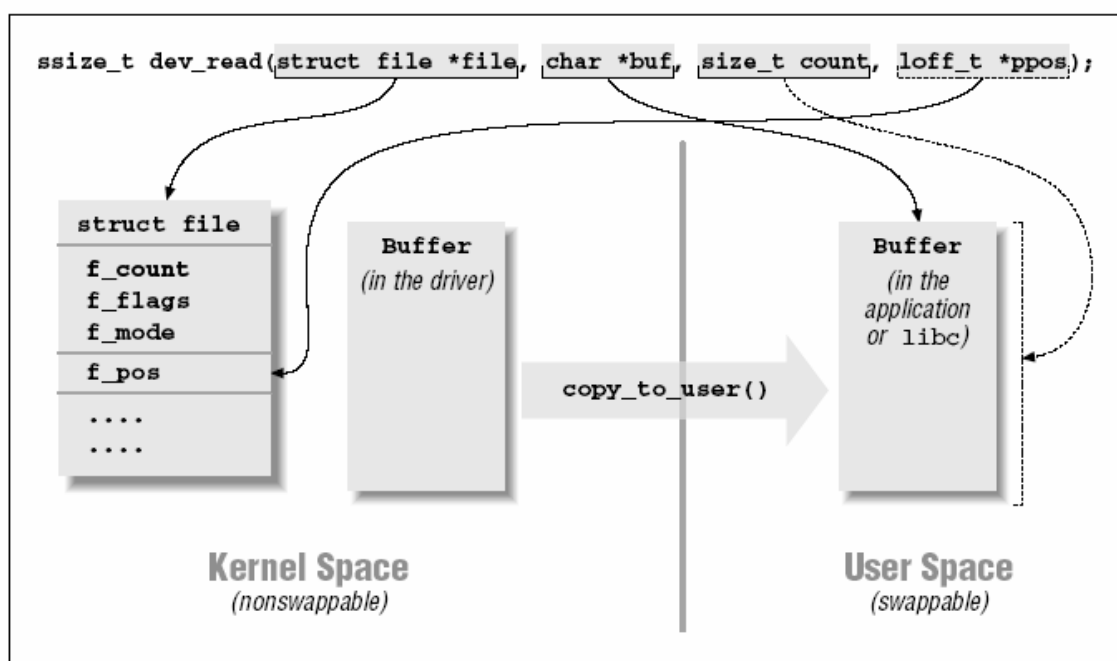


Figure 3-2. The arguments to read

vtest_dev.c 예 – read,write 메소드

```

ssize_t vtest_read(struct file *filp, char *buf, size_t count, loff_t *f_pos)
{
    char s[20] = "Computer Eng."; // 13+1 characters

    if (count > 14)
        count = 14;
    copy_to_user(buf, s, count);
    printk("<1> vtest_read: %s, %dWn", buf, count);
    return count;
}

ssize_t vtest_write (struct file *filp, const char *buf, size_t count, loff_t *f_pos)
{
    char s[20];

    copy_from_user(s, buf, count);
    printk("<1> vtest_write: %s, %dWn", s, count);

    return count;
}

```

ioctl 메소드

- ❑ 디바이스 드라이버는 읽기 쓰기 외에 **하드웨어 제어**도 수행
 - **ioctl 메소드**를 사용하여 하드웨어의 제어를 수행
- ❑ 커널의 ioctl 메소드
 - **int (*ioctl) (struct inode *inode, struct file *filp, unsigned int cmd, unsigned long arg);**
 - inode와 filp 포인터는 응용 프로그램의 파일 디스크립터 fd와 일치하는 인수
 - **cmd 인수**는 명령을 나타내는 응용 프로그램의 인수 전달
 - **arg 인수**는 명령 실행의 결과 데이터가 전달되는 unsigned long 형의 정수 또는 포인터
- ❑ 대부분의 ioctl 메소드 구현은 cmd 인수 값에 따라 올바른 동작을 선택하는 switch 문으로 구성

vtest_dev.c 예 – read,write 메소드

```
int vtest_ioctl(struct inode *inode, struct file *filp, unsigned int cmd,
                unsigned long arg)
{
    printk("<1> ioctl cmd: %d\n", cmd);
}
```

- 응용 프로그램에서는 open(), close(), read(), write(),
ioctl() 시스템 콜을 호출하여 드라이버의 각 메소드 실행

vtest_dev.c 코드 (1)

```
/* virtual test character device */
#include <linux/kernel.h> /* 커널에서 수행될 때 필요한 헤더 파일 */
#include <linux/module.h> /* 모듈에 필요한 헤더 파일 */
#include <linux/init.h> /* module_init(), module_exit() 매크로 정의 */
#include <linux/fs.h> /* file_operations, struct file 구조체 */
#include <asm/uaccess.h> /* copy_to_user(), copy_from_user() */
#define VTEST_DEV_NAME "vtest"
#define VTEST_DEV_MAJOR 240

int vtest_open (struct inode *inode, struct file *filp)
{
    MOD_INC_USE_COUNT;
    return 0;
}

int vtest_release (struct inode *inode, struct file *filp)
{
    MOD_DEC_USE_COUNT;
    return 0;
}

ssize_t vtest_read(struct file *filp, char *buf, size_t count, loff_t *f_pos)
{
    char s[20] = "Computer Eng."; // 13+1 characters

    if (count > 14)
        count = 14;
    copy_to_user(buf, s, count);
    printk("<1> vtest_read: %s, %d\n", buf, count);
    return count;
}
```

vtest_dev.c 코드 (2)

```

ssize_t vtest_write (struct file *filp, const char *buf, size_t count, loff_t *f_pos)
{
    char s[20];

    copy_from_user(s, buf, count);
    printk("<1> vtest_write: %s, %dWn", s, count);

    return count;
}

int vtest_ioctl(struct inode *inode, struct file *filp, unsigned int cmd, unsigned long arg)
{
    printk("<1> ioctl cmd: %dWn", cmd);
}

struct file_operations vtest_fops =
{
    open      : vtest_open,
    release   : vtest_release,
    read      : vtest_read,
    write     : vtest_write,
    ioctl     : vtest_ioctl
};

```

vtest_dev.c 코드 (3)

```

int vtest_init(void)
{
    int result;

    result = register_chrdev( VTEST_DEV_MAJOR, VTEST_DEV_NAME, &vtest_fops);
    if (result < 0) return result;

    return 0;
}

void vtest_end(void)
{
    unregister_chrdev(VTEST_DEV_MAJOR, VTEST_DEV_NAME );
}

module_init(vtest_init);
module_exit(vtest_end);

```

vtest_app.c 코드

```
#include <stdio.h>
#include <fcntl.h>      // O_RDWR
#include <unistd.h>     // open(), read(), write()
#include <sys/ioctl.h>  // ioctl()
#include <string.h>
int main()
{
    int dev, count;
    char buff[20];

    dev = open("/dev/vtest", O_RDWR);
    if (dev == -1) {
        printf("device file open error!!\n");
        return 0;
    }
    count = read(dev, buff, sizeof(buff));
    printf("vtest read: %s, %d\n", buff, count);
    printf("vtest ioctl -> 77\n");
    ioctl(dev, 77);
    strcpy(buff, "Soonchunhyang");
    count = strlen(buff)+1;
    write(dev, buff, count);
    printf("vtest write -> %s, %d\n", buff, count);
    close(dev);
    return 0;
}
```

Makefile

```
# Makefile for a basic kernel module

CC := gcc    # arm-linux-gcc, 크로스컴파일러

INCLUDEDIR := /usr/src/linux-2.4.20-8/include/ # 커널소스디렉토리/include
MODCFLAGS := -Wall -O2 -DMODULE -D__KERNEL__ -DLINUX -I$(INCLUDEDIR)

all: vtest_dev vtest_app

vtest_dev: vtest_dev.c
    $(CC) $(MODCFLAGS) -c vtest_dev.c

vtest_app: vtest_app.c
    $(CC) $(MODCFLAGS) -o vtest_app vtest_app.c

clean:
    rm -rf vtest_dev.o vtest_app.o vtest_app
```

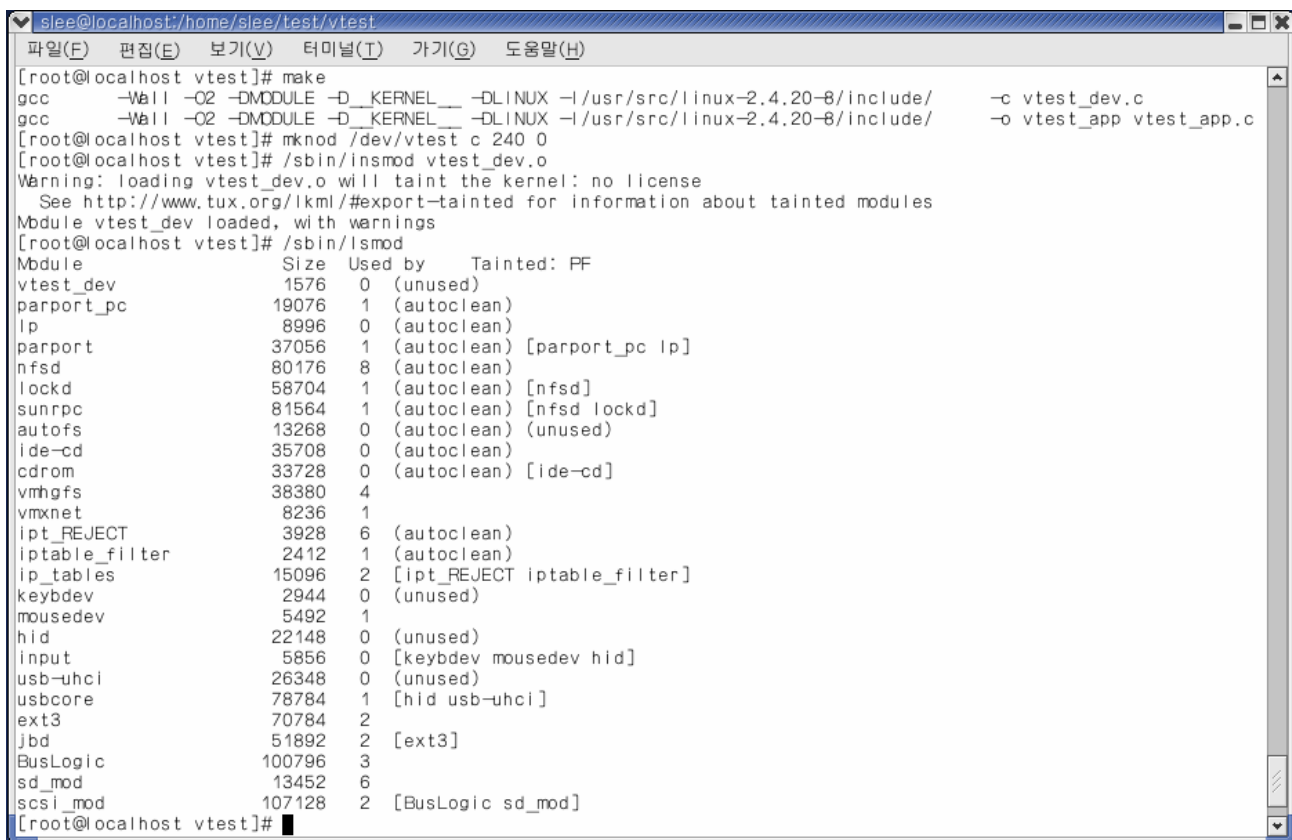
실행 예 (1)

```
# make                                <- 컴파일
# mknod /dev/vtest c 240 0           <- 디바이스 파일 생성
# /sbin/insmod vtest_dev.o           <- 드라이버 모듈 적재
# /sbin/lsmod                        <- 모듈 적재 확인
# ./vtest_app                        <- 응용 프로그램 실행

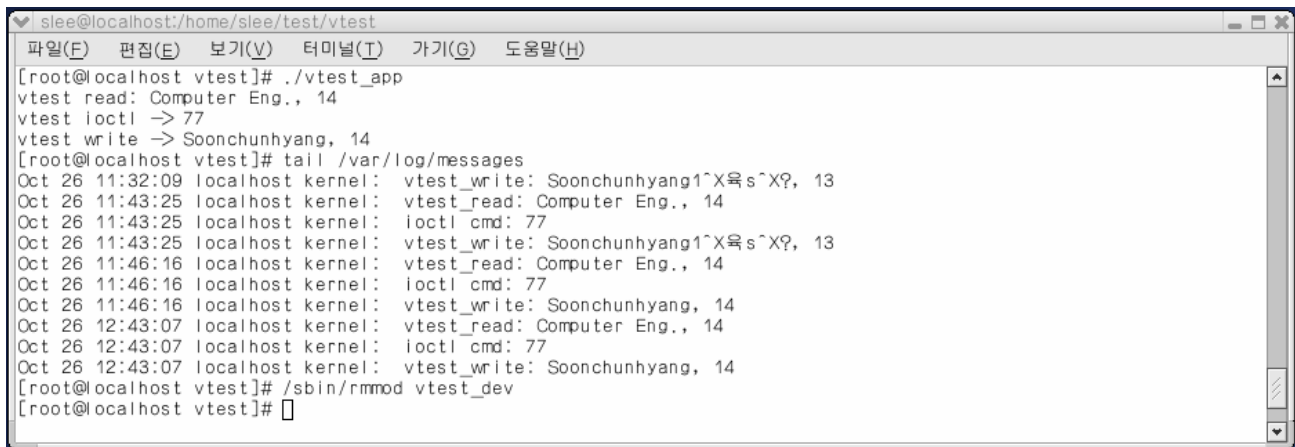
# cat /dev/vtest
# echo "test..." > /dev/vtest

# tail /var/log/messages             <- 드라이버 동작확인
# /sbin/rmmod vtest_dev              <- 드라이버 모듈 제거
```

실행 예 (2)



```
slee@localhost:~/home/slee/test/vtest
[root@localhost vtest]# make
gcc -Wall -O2 -DMODULE -D__KERNEL__ -DLINUX -I/usr/src/linux-2.4.20-8/include/ -c vtest_dev.c
gcc -Wall -O2 -DMODULE -D__KERNEL__ -DLINUX -I/usr/src/linux-2.4.20-8/include/ -o vtest_app vtest_app.c
[root@localhost vtest]# mknod /dev/vtest c 240 0
[root@localhost vtest]# /sbin/insmod vtest_dev.o
Warning: loading vtest_dev.o will taint the kernel: no license
See http://www.tux.org/lkml/#export-tainted for information about tainted modules
Module vtest_dev loaded, with warnings
[root@localhost vtest]# /sbin/lsmod
Module                Size  Used by    Tainted: PF
vtest_dev              1576    0 (unused)
parport_pc             19076    1 (autoclean)
lp                     8996    0 (autoclean)
parport                37056    1 (autoclean) [parport_pc lp]
nfsd                   80176    8 (autoclean)
lockd                  58704    1 (autoclean) [nfsd]
sunrpc                 81564    1 (autoclean) [nfsd lockd]
autofs                 13268    0 (autoclean) (unused)
ide-cd                 35708    0 (autoclean)
cdrom                  33728    0 (autoclean) [ide-cd]
vmhgfs                 38380    4
vmxnet                 8236    1
ipt_REJECT             3928    6 (autoclean)
iptables_filter        2412    1 (autoclean)
ip_tables              15096    2 [ipt_REJECT iptable_filter]
keybdev                2944    0 (unused)
mousedev               5492    1
hid                    22148    0 (unused)
input                  5856    0 [keybdev mousedev hid]
usb-uhci               26348    0 (unused)
usbcore                78784    1 [hid usb-uhci]
ext3                   70784    2
jbd                     51892    2 [ext3]
BusLogic               100796    3
sd_mod                 13452    6
scsi_mod               107128    2 [BusLogic sd_mod]
[root@localhost vtest]#
```

```
slee@localhost:/home/slee/test/vtest
[root@localhost vtest]# ./vtest_app
vtest read: Computer Eng., 14
vtest ioctl -> 77
vtest write -> Soonchunhyang, 14
[root@localhost vtest]# tail /var/log/messages
Oct 26 11:32:09 localhost kernel: vtest_write: Soonchunhyang1^X욕s^X?, 13
Oct 26 11:43:25 localhost kernel: vtest_read: Computer Eng., 14
Oct 26 11:43:25 localhost kernel: ioctl cmd: 77
Oct 26 11:43:25 localhost kernel: vtest_write: Soonchunhyang1^X욕s^X?, 13
Oct 26 11:46:16 localhost kernel: vtest_read: Computer Eng., 14
Oct 26 11:46:16 localhost kernel: ioctl cmd: 77
Oct 26 11:46:16 localhost kernel: vtest_write: Soonchunhyang, 14
Oct 26 12:43:07 localhost kernel: vtest_read: Computer Eng., 14
Oct 26 12:43:07 localhost kernel: ioctl cmd: 77
Oct 26 12:43:07 localhost kernel: vtest_write: Soonchunhyang, 14
[root@localhost vtest]# /sbin/rmmod vtest_dev
[root@localhost vtest]#
```