

# Лекция 3.0 Реализация ioctl

Разработали: Максимов А.Н., Крапивный А.В.

# Назначение IOCTL

Команды IOCTL используются для контроля работы устройств, модуля ядра, передача параметров устройству.

Пример применения:

- Установка скорости(делителя) для последовательного интерфейса;
- Сброс параметров работы, буферов модуля;
- Чтение/запись данных устройства;

Применение в пространстве пользователя(User space):

```
#include <sys/ioctl.h>
```

```
#include "mydriverio.h"
```

```
int ioctl(int fd, ulong cmd, ...);
```

# IOCTL

Применение в пространстве пользователя(User space):

```
#include <sys/ioctl.h>
#include "mydriverio.h"
int ioctl(int fd, ulong cmd, ...);
```

В данном случае точки это не переменное число параметров, у системного вызова должно быть постоянное количество аргументов так как пользовательские программы могут получить доступ к ним только через аппаратные "ворота" (hardware gates), точки позволяют избежать проверки типов в процессе компиляции.

Аргумент передается в форме unsigned long, что может соответствовать как целому значению, так и указателю. Если вызывающая программа не передает третий аргумент, то его значение, полученное драйвером имеет неопределенное значение.

int fd – файловый дескриптор

ulong cmd – команда, из списка команд драйвера, описанного в «mydriverio.h»

# Формат вызова IOCTL

## **Формат вызова в пространстве пользователя (User space):**

```
#include <sys/ioctl.h>
#include "mydriverio.h"
int ioctl(int fd, ulong cmd, ...);
```

## **Формат вызова в пространстве ядра (Kernel space):**

```
#include <linux/ioctl.h>
#include "mydriverio.h"
int ioctl(struct inode *, struct file *,ulong cmd, ulong arg);
```

struct inode \*inode,  
struct file \*filp как в open()  
ulong cmd команда, из списка команд драйвера.  
ulong arg дополнительные аргументы (...)

# Команды IOCTL

Различные команды имеют различные значения, которые определены в файле mydriverio.h

```
#include " mydriverio.h"
```

```
...
```

```
switch (cmd) {
```

```
case IOC_GET: ...
```

```
break;
```

```
case IOC_SET: ...
```

```
break;
```

```
int device_ioctl(struct inode *inode, struct file *filp, ulong cmd, ulong arg){
```

```
    int ret=0;
```

```
    switch (cmd) {
```

```
    case IOC_GET: ...; ret = +val;
```

```
        break;
```

```
    case IOC_SET: ...
```

```
        break;
```

```
    default: //return -EINVAL; // old style
```

```
        return -ENOTTY;
```

```
    }
```

```
    return ret;
```

```
}
```

# Определение команд

Команды являются аппаратно зависимой структурой битовых полей.

Формирование структуры обеспечивают макросы, описанные в библиотеках `<asm/ioctl.h>` и `<linux/ioctl.h>`)

Макросы:

`_IO(type,nr)`

`_IOR(type,nr,dataitem)`

`_IOW(type,nr,dataitem)`

`_IORW(type,nr,dataitem)`

Пример.

```
#define IOC_DOIT _IOR(0xF1, 17, struct rbus)
```

# Определение команд в ioctl.h

```
/* ioctl command encoding: 32 bits total, command in lower 16 bits,  
 * size of the parameter structure in the lower 14 bits of the  
 * upper 16 bits.  
 * Encoding the size of the parameter structure in the ioctl request  
 * is useful for catching programs compiled with old versions  
 * and to avoid overwriting user space outside the user buffer area.  
 * The highest 2 bits are reserved for indicating the ``access mode".  
 * NOTE: This limits the max parameter size to 16kB -1 ! */  
  
/* The following is for compatibility across the various Linux  
 * platforms. The i386 ioctl numbering scheme doesn't really enforce  
 * a type field. De facto, however, the top 8 bits of the lower 16  
 * bits are indeed used as a type field, so we might just as well make  
 * this explicit here. Please be sure to use the decoding macros  
 * below from now on.  
 */
```

# Пример заголовочного файла

Очевидно, что эти значения должны быть одинаковы для драйвера и программы пользователя. Поэтому команды описываются в файле, который подключается и к модулю ядра и программе пользователя.

"mydriverio.h":

```
#ifndef __MYDRIVERIO_H__
#define __MYDRIVERIO_H__
#include <linux/ioctl.h>
#define MAGIC_NUM 0xF1
#define IOC_GET_IOR(MAGIC_NUM, 0, int)
#define IOC_SET_IO (MAGIC_NUM, 1)
#endif // __MYDRIVERIO_H__
```



# Особенности реализации обработчика

Возвращаемое значение должно быть положительным

Отрицательное - error

```
return -ENOTTY; now: "Inappropriate ioctl for device"
```

При обмене с пространством пользователя необходимо  
использовать

```
copy_from_user ((void*)куда,(void*)откуда, (int)сколько);
```

или

```
copy_to_user ((void*)куда,(void*)откуда, (int)сколько);
```

# Пример реализации в драйвере

```
int device_ioctl(struct inode *inode, struct file *filp, ulong cmd, long arg) {  
    int ret=0;  
  
    switch (cmd) {  
        case IOC_GET:  
            ...; ret = +val;  
            break;  
        case IOC_SET: ...  
            break;  
        default: //return -EINVAL; // old style  
            return -ENOTTY;  
    }  
    return ret;  
}
```

# Пример регистрации fops

```
static struct file_operations fops = {  
    read:  device_read,  
    open:  device_open,  
    release: device_release,  
    ioctl:  device_ioctl,    // Реализация IOCTL  
    owner: THIS_MODULE  
};
```

# Пример реализации обработки

```
case IOC_GET: offs = (char *)filp->private_data - mess;
    put_user(offs, (int*)arg);    // *(int*)arg=offs;

    sprintf(me, " %s Get - Offs: %d \r\n", dev_name, offs);
    if (debu>1) print_x(me);
    ret=offs;
    break;

case IOC_SET: offs=arg;
    ml=strlen(mess);
    if (offs<0) offs=0; if (offs>ml) offs=ml;
    filp->private_data = mess + offs;
    sprintf(me, " %s Set - Offs: %d \r\n", dev_name, offs);
    if (debu>1) print_x(me);
    break;
```

# В Linux 2.6.37 ioctl убрали.

В версия 2.6.37 ioctl удален из file\_operations.

ioctl удален по следующим причинам:

- содержал Big Kernel Lock (BKL)
- при длительных операциях ioctl могла возникать значительная задержка

Вместоа ioctl следует использовать

`long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);`

`compat_ioctl` — вызов позволяющий 32 битным пользовательским программам вызывать ioctl на 64-битном ядре.

см. <http://unix.stackexchange.com/questions/4711/what-is-the-difference-between-ioctl-unlocked-ioctl-and-compat-ioctl>

# file\_operations для ядра 3.2.6

Измененная версия file\_operations для ядра 3.2.6 (ioctl удален)

```
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 (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    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 *, fl_owner_t id);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, loff_t, loff_t, int datasync);
}
```

# Пример unlocked\_ioctl

```
#include <linux/module.h>
#include <linux/ioctl.h>
#include "fooioctl.h"
```

```
static inline long foo_unlocked_ioctl( struct file *fp, unsigned int cmd, unsigned long arg ) {
    int retval;
    switch( cmd ) {
        case FOOIOC_1:
            printk( KERN_INFO "FOOIOC_1 is called \n");
            break;
        case FOOIOC_2:
            printk( KERN_INFO "FOOIOC_2 is called \n");
            break;
        default:
            printk( KERN_INFO " got invalid case, CMD=%d\n", cmd );
            return -EINVAL;
    }
    return 0;
}

static const struct file_operations fops = {
    .owner = THIS_MODULE,
    .unlocked_ioctl = foo_unlocked_ioctl,
    .open = foo_generic_open,
    .release = foo_generic_release
};
```

# Практическое задание. Вариант 1.

Реализовать функцию ioctl обрабатывающую две команды в качестве параметров которым передается структура типа

```
struct {  
    void *data;  
    int size;  
    int seek;  
    int keyDevice;  
    int keyData;  
}
```

Реализация команды MYDRIVER\_WRITE: Загрузить в статически выделяемую область памяти данные из пользовательского пространства, расположенные по адресу data, размером size

Реализация команды MYDRIVER\_READ: Загрузить из статически выделяемой области памяти данные в пользовательское пространство, расположенные по адресу data, размером size



# Практическое задание. Вариант 2.

Разработать символьны драйвер.

Предположим, что драйвер управляет дискретными выходами (8 шт). Необходимо реализовать следующие IOCTL команды:

- Включить  $i$ -й вход
- Выключить  $i$ -й вход
- Получить состояние  $i$ -го входа

Текущее состояние входов должно показываться через proc

# Вопросы