**Проектирование ОС и компонентов**

Драйвер символьного устройства

# 

# Теоретическая часть

**Символьные устройства**

Символьное устройство — вид файла устройства в UNIX/Linux-системах, обеспечивающий интерфейс к устройству, реальному или виртуальному, с возможностью посимвольного обмена информацией.

В отличие от блочного устройства символьное устройство, как правило, не обладает возможностями произвольного доступа. В большинстве своём, чтение и запись данных в символьное устройство не буферизуется.

Типичные примеры символьных устройств: стриммер, модем, телетайп или терминал.

Доступ к символьным устройствам похож на доступ к файлам, а задача драйвера символьного устройства – обеспечивать этот доступ. Символьные драйвера обычно способны обрабатывать такие системные вызовы как open, close, read, write. Единственное отличие символьных устройств от обычных файлов - возможность всегда вернуться к уже просмотренной информации и пройтись вперёд, тогда как большинство символьных устройств функционируют как каналы данных, к которым можно обращаться только последовательно. Тем не менее существуют устройства, выглядящие как область данных, по которой можно преспокойно продвигаться и вперёд, и назад.

**Блочные устройства**

Блочные устройства - это нечто, на чём может содержатся файловая система, например жёсткий диск. В большинстве Unix-подобных систем доступ к блочным устройствам осуществляетя посредством блоков. ОС Linux позволяет вам читать и писать на блочные устройства также как и на символьные - произвольным количеством байтов. В результате, блочные и символьные устройства отличаются лишь в способе организации хранения данных. Доступ к блочным устройствам (также как и к символьным) можно получить из файловой системы.

**Сетевые устройства**

Не являясь потоково-ориентированными, сетевые устройства не так просто отобразить на файловую систему, как например /dev/tty1. В ОС Linux сетевым устройствам присваивается лишь уникальное имя (например eth0), поскольку в файловой системе для них нет соответствующего файла. Взаимодействие между ядром и сетевым драйвером в корне отличается от взаимодействия с драйвером блочного или символьного устройсва - вместо системных вызовов read и write ядро вызывает функции, относящиеся к передаче пакетов.

Пpи поддеpжке файловой системы устpойства, она должна быть pазбита на блоки самим устpойством. Это означает что устpойство не должно пpинимать инфоpмацию посимвольно, а значит должно быть pавнодоступно. Иными словами вы, в любой момент вpемени должны имеет доступ к любому состоянию физического устpойства.

Не пpидется в случае блочных устpойств пользоваться функциями read() и write(). Вместо них используются функции block\_read() и block\_write() находящиеся в VFS и называемые !strategy routine! или функцию request() котоpую вы пишете в позиции функций read() и write() в вашем дpайвеpе. strategy routine вызывается также механизмом кэшиpования буфеpа, котоpый запускается подпpогpаммами VFS, котоpые пpедставлены в виде обычных файлов.

Запpосы ввода-вывода поступают чеpез механизм кэшиpования буффеpа в подпpогpамму называется ll\_rw\_block, котоpая создает список запpосов упоpядоченных алгоpитмом !elevator!, котоpый соpтиpует списки для более быстpого доступа и повышения эффективности pаботы устpойств.

Затем она вызывает фнкцию request() для осуществления ввода - вывода. Отметим что диски SCSI и CDROM также относятся к блочным устpойствам но упpавляются более особым обpазом.

**Практическая часть**

Для работы с модулями потребуются заголовочные файлы текущей версии ядра, для их установки используется команда:

|  |
| --- |
| **sudo apt-get install linux-headers-$(uname -r)** |

Создание драйвера символьного устройства подразумевает следующие шаги:

* Создание модуля драйвера символьного устройства
* Регистрация драйвера
* Создание файла устройства.

В папке **/Desktop/myDriver/** создадим файл **myDriver.c** с кодом модуля:

|  |
| --- |
| #include <linux/module.h>  #include <linux/version.h>  #include <linux/kernel.h>  #include <linux/types.h>  #include <linux/kdev\_t.h>  #include <linux/fs.h>  #include <linux/device.h>  #include <linux/cdev.h>  #include <linux/uaccess.h>  MODULE\_LICENSE("GPL");  MODULE\_AUTHOR("Ibayev Anar");  MODULE\_DESCRIPTION("My Driver");  static dev\_t first;  static struct cdev c\_dev;  static struct class \*cl;  static char message[256] = {0};  static short size\_of\_message;  static int dev\_open(struct inode \*, struct file \*);  static int dev\_release(struct inode \*, struct file \*);  static ssize\_t dev\_read(struct file \*, char \*, size\_t, loff\_t \*);  static ssize\_t dev\_write(struct file \*, const char \*, size\_t, loff\_t \*);  static struct file\_operations fops =  {  .owner = THIS\_MODULE,  .open = dev\_open,  .release = dev\_release,  .read = dev\_read,  .write = dev\_write  };  static int \_\_init chr\_init(void) /\* Constructor \*/  {  if (alloc\_chrdev\_region(&first, 0, 1, "chr\_test\_driver") < 0)  {  return -1;  }    if ((cl = class\_create(THIS\_MODULE, "char\_test\_drv")) == NULL)  {  unregister\_chrdev\_region(first, 1);  return -1;  }    if (device\_create(cl, NULL, first, NULL, "chrnull") == NULL)  {  class\_destroy(cl);  unregister\_chrdev\_region(first, 1);  return -1;  }    cdev\_init(&c\_dev, &fops);    if (cdev\_add(&c\_dev, first, 1) == -1)  {  device\_destroy(cl, first);  class\_destroy(cl);  unregister\_chrdev\_region(first, 1);  return -1;  }  return 0;  }  static void \_\_exit chr\_exit(void) /\* Destructor \*/  {  cdev\_del(&c\_dev);  device\_destroy(cl, first);  class\_destroy(cl);  unregister\_chrdev\_region(first, 1);  }  static int dev\_open(struct inode \*i, struct file \*f)  {  printk(KERN\_INFO "My Driver: open()\n");  return 0;  }  static int dev\_release(struct inode \*i, struct file \*f)  {  printk(KERN\_INFO "My Driver: close()\n");  return 0;  }  static ssize\_t dev\_read(struct file \*f, char \_\_user \*buffer, size\_t len, loff\_t \*offset)  {  int error\_count = 0;  int size = 0;  // copy\_to\_user has the format ( \* to, \*from, size) and returns 0 on success  error\_count = copy\_to\_user(buffer, message, size\_of\_message);  if (error\_count != 0) { // if true then have success  printk(KERN\_INFO "Failed to send %d characters to the user\n", error\_count);  return -EFAULT; // Failed -- return a bad address message (i.e. -14)  }    printk(KERN\_INFO "Sent %d characters to the user\n", size\_of\_message);  size = size\_of\_message;  size\_of\_message = 0;  return size; // clear the position to the start and return 0  }  static ssize\_t dev\_write(struct file \*f, const char \_\_user \*buffer, size\_t len, loff\_t \*offset)  {  sprintf(message, "%s(%d letters) \n", buffer, (int)len); // appending received string with its length  size\_of\_message = strlen(message); // store the length of the stored message  printk(KERN\_INFO "Received %d characters from the user\n", (int)len);  return len;  }  module\_init(chr\_init);  module\_exit(chr\_exit); |

Для сборки используется утилита **make** и следующий Makefile:

|  |
| --- |
| ifeq ($(KERNELRELEASE),)  KERNELDIR ?= /lib/modules/$(shell uname -r)/build  PWD := $(shell pwd)  .PHONY: build clean  build:  $(MAKE) -C $(KERNELDIR) M=$(PWD) modules  clean:  rm -rf \*.o \*~ core .depend .\*.cmd \*.ko \*.mod.c  else  $(info Building with KERNELRELEASE = ${KERNELRELEASE})  obj-m := myDriver.o  endif |

Произведем сборку модуля:

|  |
| --- |
| anar@ubuntu:~/Desktop/myDriver$ make |

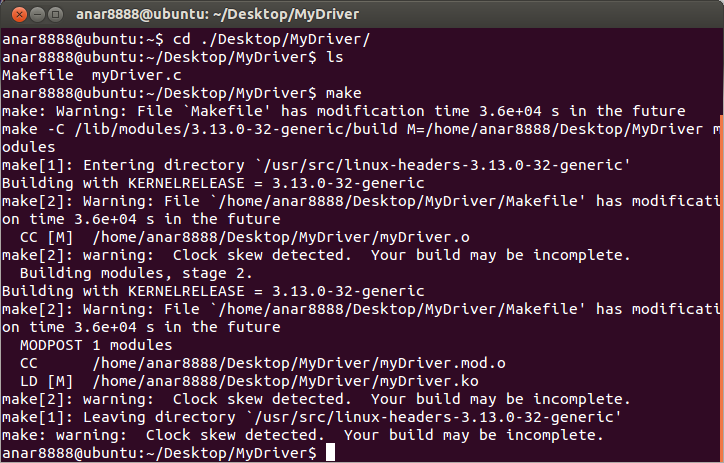


Рисунок 1

В текущей директории **~/ Desktop/myDriver** появились следующие файлы:

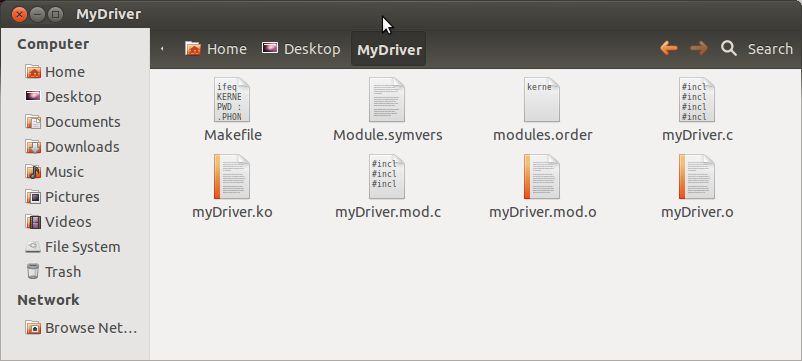


Рисунок 2

Произведем установку модуля c помощью команды **insmod** и проверим, установился ли модуль, командой **lsmod**:

|  |
| --- |
| anar@ubuntu:~/Desktop/myDriver$ sudo insmod moduleTest.ko anar@ubuntu:~/Desktop/myDriver$ lsmod | grep module |

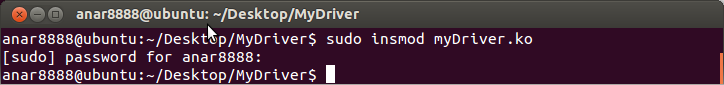


Рисунок 3

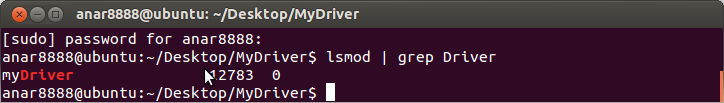


Рисунок 4

Проверим добавление класса устройств:

|  |
| --- |
| anar@ubuntu:~/Desktop/myDriver$ ls /sys/class/char\_test\_drv |

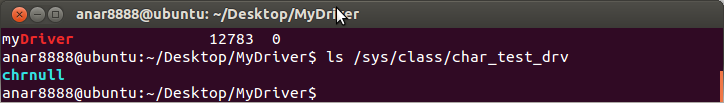


Рисунок 5

И проверим, произошло ли добавление драйвера:

|  |
| --- |
| anar@ubuntu:~/Desktop/myDriver$ cat /proc/devices |

Для отображения драйверов устройств пользуются файловой системой proc. В файле /proc/devices можно найти все драйвера, которые используются в текущей системе, разделенные на символьные и блочные. Для вывода можно воспользоваться утилитой **cat**. Ниже приводится частичный лог вывода файла /proc/devices:

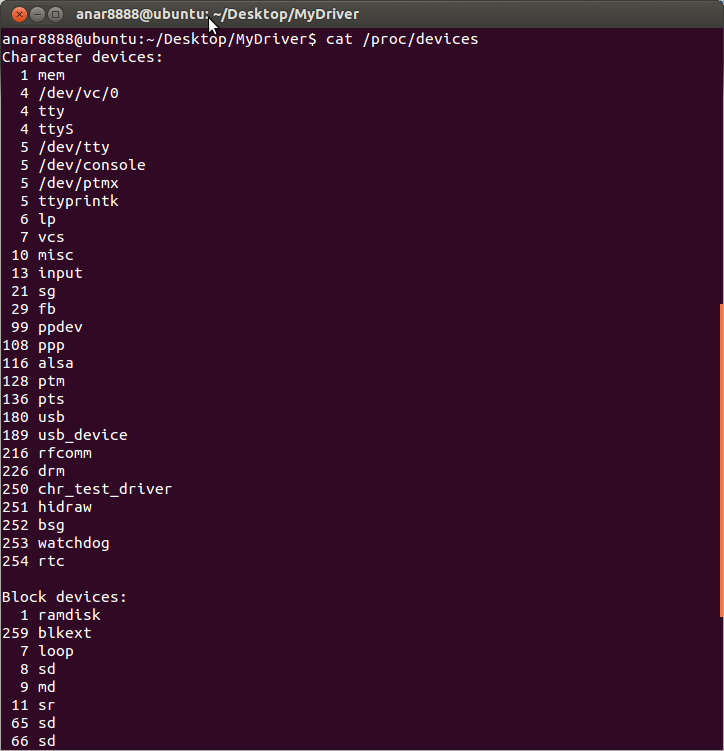


Рисунок 6

Как видно, наш драйвер создан со старшим номером 250.

Изменим права доступа к файлу устройства, чтобы можно было с ним работать:

|  |
| --- |
| anar@ubuntu:~/Desktop/myDriver$ sudo chmod 777 /dev/chrnull |

Вывод в файл устройства сообщения:

|  |
| --- |
| anar@ubuntu:~/Desktop/myDriver$ sudo echo “hi, my driver” > /dev/chrnull |

Чтение из файла устройства сообщения. Вывод в консоль:

|  |
| --- |
| anar@ubuntu:~/Desktop/myDriver$ sudo cat /dev/chrnull |

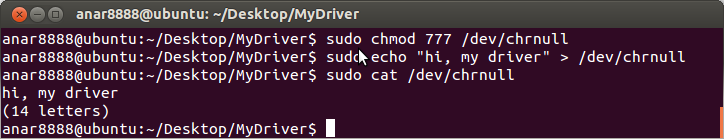


Рисунок 7

Драйвер работает корректно. Чтобы выгрузить модуль из ядра используем следующую команду:

|  |
| --- |
| anar@ubuntu:~/Desktop/myDriver$ sudo rmmod myDriver  anar@ubuntu:~/Desktop/myDriver$ dmesg | tail |

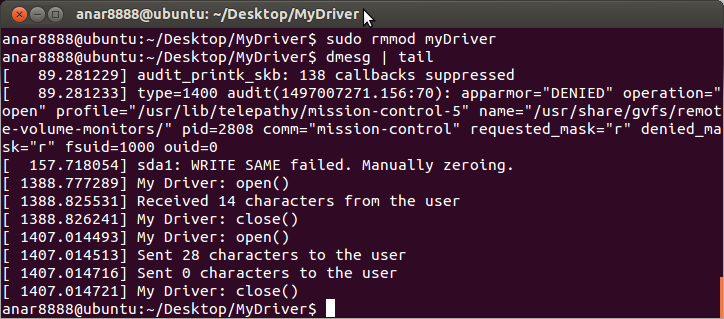


Рисунок 8

Список используемых

1. <http://linux-doc.narod.ru/development/drivers.html>
2. <http://citforum.ru/operating_systems/linux_khg/linux_khg_02.shtml>