Драйвер символьного устройства в Linux

Драйвер устройства — Это низкоуровневая программа, содержащая специфический код для работы с устройством, которая позволяет пользовательским программам (и самой ОС) управлять им стандартным образом. Устройства можно разделить на:

- Символьные - чтение и запись устройства идет посимвольно. Примеры таких устройств: клавиатура, последовательные порты.
- Блочные - чтение и запись устройства возможны только блоками, обычно по 512 или 1024 байта. Пример - жесткий диск.
- Сетевые интерфейсы - отличаются тем, что не отображаются на файловую систему, т.е. не имеют соответствующих файлов в директории /dev, поскольку из-за специфики этих устройств работа с сетевыми устройствами как с файлами неэффективна. Пример - сетевая карта (*eth0*).

Установка драйвера может быть выполнена в режиме суперпользователя при помощи команды *insmod*. Для просмотра установленных модулей предусмотрена команда *lsmod*. Удаление модуля - *rmmod*. Для просмотра информации о модуле можно воспользоваться утилитой *modinfo*. Вызов утилиты с ключом *-d* возвращает информацию об устройстве, вызов с ключом *-a* возвращает информацию об авторах модуля.

Для символьных и блочных устройств - взаимодействие с драйвером реализуется через специальные файлы, расположенные в директории /dev. Каждый файл устройства имеет два номера - старший, определяющий тип устройства, и младший, определяющий конкретный номер устройства (в системе может быть несколько устройств одного типа - например, несколько жестких дисков). Многие из старших номеров устройств зарезервированы. Посмотреть номера можно, например командой

*ls -l /dev/sda**

В результатах этой команды будут, например, два числа разделённых запятой между словом «*disk*» и датой. Первое число называют старшим номером устройства. Старший номер указывает на то, какой драйвер используется для обслуживания данного устройства. Каждый драйвер имеет свой уникальный старший номер.

В современных версиях ядра *Linux* установка и работа с драйверами происходит при помощи механизма динамически загружаемых модулей. Механизм дает возможность устанавливать драйвера новых устройств "на лету" - без перекомпиляции ядра и без перезагрузки системы. Однако устанавливать можно драйвера только для устройств, реально присутствующих в системе. Динамические модули, в отличие от обычных программ, представляют собой объектные файлы, скомпилированные по определенным

правилам. В первых версиях *Linux* драйвера устройств были "зашиты" в ядро. Недостатки такого решения очевидны. Во-первых, драйвера, включенные в ядро, загружаются даже при отсутствии устройства в системе - и потребляют системные ресурсы. Во-вторых, при подключении нового устройства (или новой версии драйвера) требуется перекомпиляция ядра.

Для создания модулей предусмотрен ряд библиотек *linux/* среди которых обязательными являются <*linux/init.h*> и <*linux/module.h*>. В библиотеке <*linux/init.h*> содержатся описания таких функций как *init_module*(…) и *cleanup_module*(…) а также *int module_init (void * callback)* и *void module_exit (void * callback)* – они обязательны и - вызывается при загрузке модуля ядром. При нормальной работе *module_init* должен возвращать "0". Библиотека <*linux/module.h*> содержит ряд функций, макросов и констант ядра *Linux*. В том числе функция *printk* для вывода сообщений в консоль (при отладке), а также системные константы *__KERNEL__* и *MODULE*, которые заставляют компилятор генерировать код динамически загружаемого модуля. Также в файле <*module.h*> определены ряд системных макросов, среди которых выделим такие как *MODULE_DESCRIPTION, MODULE_AUTHOR* и *MODULE_LICENSE* – позволяющие предоставить информацию об авторе и описание модуля.

Отметим, что в модуле ядра нет доступа к стандартным потокам ввода-вывода, поэтому печать может осуществляться только в системный лог-файл с помощью функции *printk*. Результат этого вывода можно увидеть с помощью команды *dmesg*. Кроме того, в коде модуля нет доступа к функциям стандартной библиотеки С, такими как, например, *printf*. Вместо них в ядре реализована собственная "стандартная" библиотека — *Linux Kernel API*. Также важно отметить, что модули функционируют на уровне ядра – поэтому надо быть особенно внимательным к тому, что в них содержится.

Задание 1.

Следует отметить, что изучение драйверов рекомендуется с использованием виртуальной машины.

1. Создайте проект.
2. В проекте создайте файл hello.c, который должен содержать следующий код

```
#include<linux/module.h>
#include<linux/init.h>
#include<linux/kernel.h>
#ifndef MODULE
    #define MODULE
#endif
#ifndef __KERNEL__
    #define __KERNEL__
#endif
```

```
static int hello_init(void){
    printk(KERN_ALERT "HELLO WORLD!\n");
    return 0;
}
static void hello_exit(void){
    printk(KERN_INFO "HELLO MODULE HAS BEEN STOPED\n");
}
module_init(hello_init);
module_exit(hello_exit);

MODULE_AUTHOR("STUDENT NAME");
MODULE_DESCRIPTION("THE HELLO MODULE.");
MODULE_LICENSE("GPL");
```

3. Создайте Makefile со следующим скриптом

```
obj-m +=hello.o
PWD:= $(shell pwd)
KDIR:= /usr/src/$(shell uname -r)
all:
    make -C $( KDIR) M=$( PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$( PWD) clean
```

Важно отметить, что сборка драйвера осуществляется предпочтительно с использованием *make* файла. Также следует отметить, что в некоторых версиях *Linux* следует использовать путь вида */lib/modules/$(shell uname -r)/build.*

В случае отсутствия необходимых заголовочных файлов их можно установить с помощью менеджера пакетов ОС. Например, в Debian-подобных Linux системах это можно сделать следующим образом:

sudo apt-get install build-essential linux-headers- $(uname -r)

4. Скомпилируйте файл драйвера.
5. Проверьте папку проекта.
6. Проверьте информацию о модуле при помощи утилиты *modinfo* (для файла с расширением «*.ko*»).
7. Прейдите в режим *root* (суперпользователя, *su*).

   Если у вас возникает ошибка при переходе в режим попробуйте проверить наличие пароля для *su* и установите его в случае отсутствия командой *sudo passwd root*.
8. Выполните установку модуля при помощи утилиты *insmod*.
9. Проверьте наличие модуля в списке при помощи утилиты *lsmod* и утилиты *grep*
10. Проверьте логи (*dmesg | tail);*
11. Зарегистрируйте модуль при помощи утилиты *mknod*

    mknod /dev/hello c 249 0
12. Проверьте наличие модуля в *dev/*.
13. Удалите модуль при помощи утилиты *rmmod* и проверьте логи.

14. Напишите комментарии к коду написанного модуля.
15. Создайте в *Makefile* цель тест, выполняющую действия по проверки модуля (описанные выше пункты). Также создайте цель установка модуля и цель его удаления. Проверьте работоспособность целей.

Описанные функции *hello_init* и *hello_exit* регистрируются как *callback* (с помощью *module_init* и *module_exit*) и будут вызываться, соответственно, при загрузке выгрузке модуля ядра. Часто рекомендуется помечать эти функции *callback* специальными макросами *__init* и *__exit*. Функции работы с модулями ядра должны быть максимально быстрыми. То есть основная часть кода модуля должна быть вне этих функций. Также следует отметить, что при разработке модулей не должно быть слишком долгих функций (например ожидания ресурсов в бесконечном цикле) так как из-за этого может повиснуть вся система – ведь модуль работает в *zero-ring*.

Модуль работает не линейно, в нем отсутствует функция *main* таким образом большая часть функционала реализуется по заданному протоколу.

Если модуль должен быть не отдельным файлом, подключаемым к ядру, а непосредственно включен в образ ядра, то вместо *obj-m* следует использовать *obj-y*. Также важно отметить, что если модуль имеет лицензию, несовместимую с *GPL*, то при его загрузке, ему не будут доступно множество функционала ядра.

В предыдущем задании была проведена ручная регистрация устройства и отведение к нему ресурсов Регистрация устройства важна чтобы сделать устройство доступным для системы. Для полноценной работы модуля в *Linux* – его необходимо зарегистрировать и указать используемые ресурсы. Такими ресурсами является таблица доступных функций. Драйвер в *Linux* хранит таблицу доступных функций, а система вызывает эти функции, когда кто-либо пытается выполнить некоторые действия (открытие, запись, чтение) с файлом устройства. Для символьного устройства - таблица функций драйвера хранится в структуре типа *file_operations*.

Для автоматической регистрации различных типов устройств предусмотрена библиотека <*fs.h*>, в частности семейство функций с префиксом *register*. Для регистрации описанного выше драйвера символьного устройства возможно использовать функцию

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

где первый параметр - старший номер файла устройства (тип устройства, если «0», то функция возвращает свободный старший номер; второй параметр - имя устройства (имя, под которым устройство будет отображаться в списке); третий параметр - структура с указателями на функции драйвера. Структура *file_operations* может быть определена

пользователем, например, структура может содержать такие поля, как *open*, *read*, *write*, *release*, и т.д. Каждое из этих полей должно включать указатель на соответствующую функцию, имеющую заданный прототип.

Задание 2.

1. Добавьте в проект инициализацию автоматической регистрации, например включающую следующие части.

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

#ifndef MODULE
        #define MODULE
#endif
#ifndef __KERNEL__
        #define __KERNEL__
#endif

#define DEVICE_NAME "TEST"

static int major_number = 0;

static int device_open(struct inode *inode, struct file *file);
static int device_release(struct inode *inode, struct file *file);
static ssize_t dev_write(struct file*, const char*, size_t, loff_t *);

static struct file_operations fops = {
        .open = device_open,
        .release = device_release,
        .write = dev_write
};
static int __init hello_init(void){
   printk(KERN_ALERT "Hello WORLD \n");
   major_number = register_chrdev(0,DEVICE_NAME, &fops);
   if (major_number<0){
           printk("register failed %d\n", major_number);
           return major_number;
   }
   printk("register sucess");
   printk("please create dev file with 'sudo mknode /dev/hello c %d 0'\n",major
_number);
   return 0;
}


static void __exit hello_exit(void){
   unregister_chrdev(major_number, DEVICE_NAME);
   printk(KERN_INFO "Goodbye \n");
}



static int device_open(struct inode *n, struct file *f)
{ Terminal
        printk("device open");
        return 0;
}
static int device_release(struct inode *n, struct file *f){
        printk("release 1");
        return 0;
} Ubuntu Software
static ssize_t dev_write(struct file *flip, const char* buff, size_t len, loff_
t * off){
        printk("NOT IMLPEMENTED\n");
        return -EINVAL;
}
module_init(hello_init);
module_exit(hello_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("MV");
MODULE_DESCRIPTION("TEST DEVICE");
```

2. Перейдите в режим суперпользователя и попробуйте собрать проект и установить модуль.
3. Проверьте логи.
4. Зарегистрируйте устройство в соответствии с рекомендациями.
5. Попробуйте выполнить команду

   echo 1>/dev/hello

Объясните результат.

6. Удалите модуль и зарегистрированный файл.
7. Напишите комментарии к коду в пункте 2.

Процедура установки драйвера может быть выпалена с указанием аргументов. Такие параметры должны быть проинициализированы при помощи функции *module_parameter*. Данная функция находится в библиотеке *linux/modulepararm*.

Задание 3.

1. Дополните проект задания 2, включением следующих операций
   a. Инициализация *linux/modulepararm.h*
   b. Объявление глобальной переменной *myint* (тип *int*).
   c. Проведите инициализацию аргумента при помощи вызова
      module_param(myint, int, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
   d. Добавьте описание аргумента при помощи макроса
      MODULE_PARM_DESC(myint, "An integer");
2. Создайте проект с указанием одного параметра.
3. Проверьте работоспособность проекта.
4. Пересоздайте проект, включив два аргумента со значениями по умолчению, например первый *int*, а второй аргумент может быть строкой,
   *static char *mystring = "test string";*
   которая должна быть проинициализирована как
   module_param(mystring, charp, 0000);
   MODULE_PARM_DESC(mystring, "A character string");
5. Проверьте работоспособность проекта при создании без параметров, с одним и с двумя параметрами.