УНИВЕРСИТЕТ ИТМО
Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

«Системное и прикладное программное обеспечение»

Дисциплина «Программирование в режиме ядра»

Драйвер, подсчитывающий количество обращений на чтение.

Студент:
Шехаде Даниэль

Группа:
P4215

Санкт-Петербург, 2025 г.

Отчет по лабораторной работе

Вариант 6. Драйвер “Счетчик”

Цель работы

Цель данной лабораторной работы — изучить процесс разработки и тестирования символьных драйверов в Linux, научиться обрабатывать системные вызовы `open`, `release`, `read`, `write` и реализовать собственное устройство, работающее через файловый интерфейс в `/dev`.

Задание

Реализовать драйвер, подсчитывающий количество обращений на чтение.

Требования:

1. Драйвер хранит одну целочисленную переменную-счётчик.
2. Операция write не поддерживается, драйвер должен возвращать ошибку.
3. Операция read при каждом вызове увеличивает счётчик на единицу и возвращает новое значение пользователю в виде строки.

Листинг

```c
#include <linux/module.h>
#include <linux/init.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/uaccess.h> // copy_to_user, copy_from_user

#define DEVICE_NAME "read_counter"

static int major;

static int counter = 0;  // счётчик обращений на чтение

// open
static int rc_open(struct inode *inode, struct file *file)
{
    printk(KERN_INFO "read_counter: device opened\n");
    return 0;
}

// release
static int rc_release(struct inode *inode, struct file *file)
{
    printk(KERN_INFO "read_counter: device closed\n");
    return 0;
}
```

```c
// read
static ssize_t rc_read(struct file *file, char __user *buf, size_t count,
loff_t *ppos)
{
    printk(KERN_INFO "read_counter: read file\n");

    // если пользователь уже читал этот буфер, возвращаем 0 (EOF), как
правило происходит 2 вызова rc_read
    if (*ppos > 0) {
        printk(KERN_INFO "read_counter: already read file\n");
        return 0;
    }

    char msg[32];
    int len;

    counter++; // увеличиваем счётчик
    len = snprintf(msg, sizeof(msg), "%d\n", counter);

    if (copy_to_user(buf, msg, len))
        return -EFAULT;

    *ppos = len; // сдвигаем файловую позицию
    return len;
}

// write (не поддерживается)
static ssize_t rc_write(struct file *file, const char __user *buf, size_t
count, loff_t *ppos)
{
    return -EOPNOTSUPP;
}

static struct file_operations fops = {
    .owner   = THIS_MODULE,
    .open    = rc_open,
    .release = rc_release,
    .read    = rc_read,
    .write   = rc_write,
};

// загрузка модуля
static int __init rc_init(void)
{

    // Запрашиваем major number у ядра
    major = register_chrdev(0, DEVICE_NAME, &fops);
    if (major < 0) {
        printk(KERN_ALERT "failed to register a major number\n");
        return major;
    }


    printk(KERN_INFO "read_counter: module loaded, major=%d\n", major);
```

```
    printk(KERN_INFO "Create device file: mknod /dev/%s c %d 0\n",
DEVICE_NAME, major);

    return 0;
}

// выгрузка модуля
static void __exit rc_exit(void)
{
    unregister_chrdev(major, DEVICE_NAME);
    printk(KERN_INFO "read_counter: module unloaded\n");
}

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Kernel Programming Task");
MODULE_DESCRIPTION("Char driver that counts read() calls");

module_init(rc_init);
module_exit(rc_exit);
```

Makefile

```
obj-m += read_counter.o

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

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

Протокол работы

1. Сборка модуля:

   ```
   make
   ```

2. Загрузка модуля:

   ```
   sudo insmod read_counter.ko
   ```

3. Создание файла символьного устройства:

   ```
   sudo mknod /dev/read_counter c 507 0
   ```

4. Проверка чтения из устройства:

```
cat /dev/read_counter
# Вывод: 1
cat /dev/read_counter
# Вывод: 2
cat /dev/read_counter
# Вывод: 3
```

5. Проверка записи (операция не поддерживается):

```
echo 123 > /dev/read_counter
```

6. Удаление модуля:

```
sudo rmmod read_counter
```

Выводы

В ходе работы был реализован символьный драйвер устройства для Linux, который подсчитывает количество обращений к операции `read`.

- Драйвер корректно обрабатывает системные вызовы `open`, `release`, `read` и `write`.
- При каждом чтении счётчик увеличивается и возвращается пользователю.
- Попытка записи в устройство завершается ошибкой, как и требовалось.
- Модуль успешно компилируется, загружается и тестируется через интерфейс `/dev`.

Таким образом, цель работы достигнута, задание выполнено.