Федеральное государственное автономное образовательное
учреждение высшего образования
Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина *«Системы ввода-вывода»*

Отчет по лабораторной работе №2

«Основы написания драйверов устройств с использованием операционной системы»

Вариант: 1

Студент:
Барсуков Максим Андреевич,
поток 1.3, группа P3315

Преподаватель:
Табунщик Сергей Михайлович

г. Санкт-Петербург, 2025 г.

Оглавление

Введение

Цели работы

Познакомится с основами разработки драйверов устройств с использованием операционной системы на примере создания драйверов символьных устройств под операционную систему Linux.

Задачи работы

1. Написать драйвер символьного устройства, удовлетворяющий требованиям:
    - должен создавать символьное устройство `/dev/varN`, где `N` – это номер варианта
    - должен обрабатывать операции записи и чтения в соответствии с вариантом задания
2. Оформить отчет по работе в электронном формате

Вариант

№ варианта	Описание
1	При записи текста в файл символьного устройства должен осуществляться подсчет введенных символов. Последовательность полученных результатов (количество символов) с момента загрузки модуля ядра должна выводиться при чтении файла.

Выполнение

Ссылка на репозиторий: GitHub.

Описание функций

`my_read`

```
#define BUF_SIZE 256
#define MAX_LOG_ENTRIES 50
#define LOG_BUF_SIZE (BUF_SIZE * MAX_LOG_ENTRIES)

static char log_buf[LOG_BUF_SIZE];
static int log_size = 0;
static DEFINE_MUTEX(log_mutex);

static ssize_t my_read(struct file *f, char __user *buf, size_t len, loff_t
*off)
{
  int ret = 0;
  printk(KERN_INFO "Driver: read()\n");
  mutex_lock(&log_mutex);

  if (*off >= log_size)
  {
    mutex_unlock(&log_mutex);
    return 0;
  }

  if (*off + len > log_size)
  {
    len = log_size - *off;
  }

  if (copy_to_user(buf, log_buf + *off, len) != 0)
  {
    mutex_unlock(&log_mutex);
    return -EFAULT;
  }

  *off += len;
  ret = len;
  mutex_unlock(&log_mutex);
  return ret;
}
```

Функция `my_read` обеспечивает чтение накопленных данных о длине записанных строк из буфера `log_buf`. Она проверяет корректность смещения `off` и длины запроса `len`, обрезая её при достижении конца буфера. Если смещение уже больше нуля, значит пользователь ранее читал данные. Чтобы избежать повторного чтения того же самого, возвращается 0 (конец файла). После успешного чтения обновляется `off`, чтобы не читать снова. Возвращает количество фактически прочитанных байт или код ошибки `-EFAULT` (ошибка доступа к памяти пользователя).

my_write

```
static void log_number(int count)
{
  char int_to_str[12];
  int str_len;

  if (log_size + sizeof(int_to_str) >= LOG_BUF_SIZE)
  {
    memmove(log_buf, log_buf + log_size / 2, log_size / 2);
    log_size = log_size / 2;
  }

  str_len = snprintf(int_to_str, sizeof(int_to_str), "%d ", count);
  memcpy(log_buf + log_size, int_to_str, str_len);
  log_size += str_len;
}

static ssize_t my_write(struct file *f, const char __user *buf, size_t len, loff_t
*off)
{
  char input_buf[BUF_SIZE];
  int count;

  printk(KERN_INFO "Driver: write()\n");

  if (len == 0 || len > BUF_SIZE)
  {
    return -EINVAL;
  }

  if (copy_from_user(input_buf, buf, len))
  {
    return -EFAULT;
  }

  input_buf[len] = '\0';

  mutex_lock(&log_mutex);
  count = strlen(input_buf);
  log_number(count);
  mutex_unlock(&log_mutex);

  return len;
}
```

Функция `my_write` обрабатывает запись строки в устройство: копирует данные из пользователя в буфер `input_buf`, вычисляет длину строки и сохраняет ее в `log_buf` в виде строки через `log_number`. При переполнении `log_buf` старые записи сдвигаются на половину буфера, освобождая место. Возвращает исходную длину данных или ошибки `-EINVAL` (неверная длина) и `-EFAULT` (ошибка копирования). Обе функции гарантируют потокобезопасность через мьютекс.

Демонстрация

lsmod:

```
ubuntu@ubuntu:~$ make
make -C /lib/modules/5.15.0-1018-generic/build M="/home/ubuntu" modules
make[1]: Entering directory '/usr/src/linux-headers-5.15.0-1018-generic'
warning: the compiler differs from the one used to build the kernel
  The kernel was built by: gcc-10 (Ubuntu 10.3.0-1ubuntu1~20.04) 10.3.0
  You are using:           gcc (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0
make[1]: Leaving directory '/usr/src/linux-headers-5.15.0-1018-generic'
ubuntu@ubuntu:~$
ubuntu@ubuntu:~$ sudo insmod ch_drv.ko
ubuntu@ubuntu:~$ lsmod | grep ch_drv
ch_drv                 36864  0
ubuntu@ubuntu:~$
```

dmesg:

```
ubuntu@ubuntu:~$ dmesg | tail -n 4
[   30.097876] audit: type=1400 audit(1747581698.820:11): apparmor="STATUS"
 operation="profile_load" profile="unconfined" name="nvidia_modprobe//kmod"
 pid=442 comm="apparmor_parser"
[  156.901174] ch_drv: loading out-of-tree module taints kernel.
[  156.901747] ch_drv: module verification failed: signature and/or require
d key missing - tainting kernel
[  156.903307] Hello!
ubuntu@ubuntu:~$
```

Использование:

```
ubuntu@ubuntu:~$
ubuntu@ubuntu:~$ echo "Hello" | sudo tee /dev/var1
Hello
ubuntu@ubuntu:~$ echo "it's-a" | sudo tee /dev/var1
it's-a
ubuntu@ubuntu:~$ echo "me" | sudo tee /dev/var1
me
ubuntu@ubuntu:~$ sudo cat /dev/var1
6 7 3 ubuntu@ubuntu:~$
ubuntu@ubuntu:~$ echo "Mario" | sudo tee /dev/var1
Mario
ubuntu@ubuntu:~$ echo "!" | sudo tee /dev/var1
!
ubuntu@ubuntu:~$ sudo cat /dev/var1
6 7 3 6 2 ubuntu@ubuntu:~$
ubuntu@ubuntu:~$
```

Вывод

В ходе выполнения лабораторной работы я познакомился с процессом создания драйверов символьных устройств под операционную систему Linux.