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 3](#_4wv7arx2x4xz)

[Цели работы 3](#_rymypilpit2s)

[Задачи работы 3](#_rfqtgunsc9pf)

[Вариант 3](#_7djy6o5r8um8)

[Выполнение 4](#_z5ouzacohqk7)

[Описание функций 4](#_2ag8cldjolhe)

[my\_read 4](#_g0geehv59b1r)

[my\_write 5](#_n6pwxarm0ayd)

[Демонстрация 6](#_ur5208a6hjk5)

[Вывод 7](#_sbim1ukgwjm0)

# **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](https://github.com/maxbarsukov/itmo/tree/master/6%20%D1%81%D0%B2%D0%B2/%D0%BB%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D1%8B%D0%B5/lab2).

## **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: |
| --- |
| dmesg: |
| Использование: |

# **Вывод**

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