МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
“Национальный исследовательский университет ИТМО”

**ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ
И КОМПЬЮТЕРНОЙ ТЕХНИКИ**

ЛАБОРАТОРНАЯ РАБОТА №2

“Основы написания драйверов устройств с использованием операционной системы”

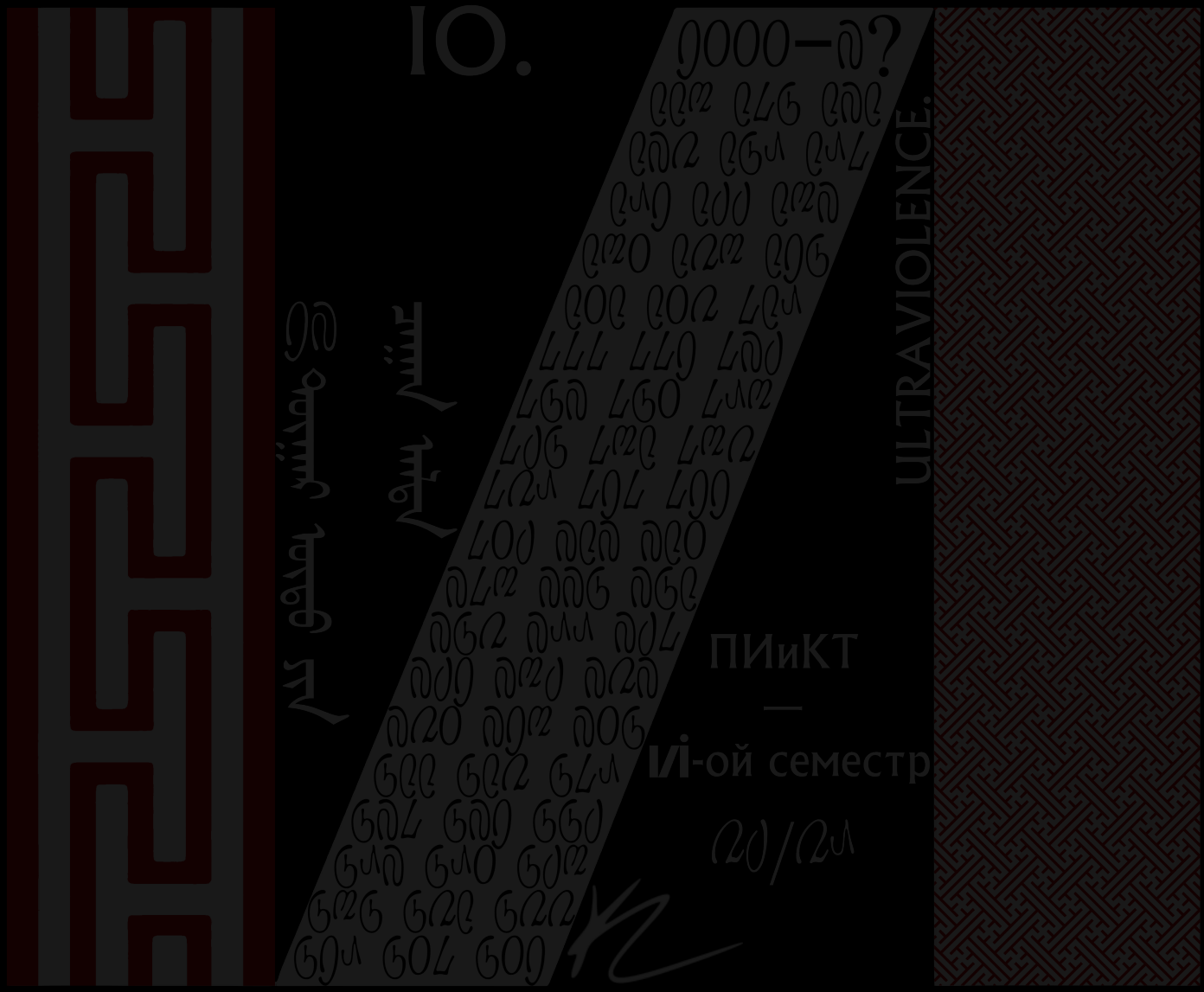
по дисциплине
“Системы ввода-вывода”

Вариант: 1.

Работу выполнил:
Студент потока 1.1, группы P3311
Болорболд Аригуун
Лектор:
Быковский Сергей Вячеславович
Практик:
Табунщик Сергей Михайлович

г. Санкт-Петербург
2025 г.

Содержимое



1 Текст задания

1.1 Цель задания

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

1.2 Задачи

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.

№ 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.

2 Выполнение

Ссылка на репозиторию: GitHub

2.1 Описания функции

1. `my_read()`:

```
static ssize_t my_read(struct file *f, char __user *buf, size_t len, loff_t *offf) {
    int count = strlen(log_buf);
    printk(KERN_INFO "Driver: read()\n");
    if (*off > 0 || count > len) {
        return 0;
    }

    if (copy_to_user(buf, log_buf, count) != 0) {
        return -EFAULT;
    }

    *off = count;

    return count;
}
```

   - `*off > 0` — если оффсет (смещение) уже больше нуля, значит пользователь ранее читал данные. Чтобы избежать повторного чтения того же самого, возвращается 0 (конец файла).
   - `len < count` — если буфер пользователя меньше длины `log_buf`, то ничего не читаем (ограничение безопасности).
   - Если операция не удалась (вернула не 0), то возвращается `-EFAULT` (ошибка доступа к памяти пользователя).
   - После успешного чтения обновляется `*off`, чтобы не читать снова.

2. `count_sym()`:

```
static int count_sym(int len) {
    int i;
    int sym_count;

    for(i = 0; i < len; ++i) {
        ++sym_count;
    }

    return sym_count - 1;
}
```

Обычный подсчёт символов, всё тривиально. Только в конце отнимается 1, чтобы не учесть нуль-терминатор.

3. `log_number()`:

```
static void log_number(int count) {
    char int_to_str[5];
    sprintf(int_to_str, "%d ", count);
    strncat(log_buf, int_to_str, strlen(int_to_str));
}
```

- Почему 5? Максимальное 4-значное число (например, 9999) + завершающий `\0`.
- `sprintf()` записывает `count` в `int_to_str` в формате `"%d "` (число + пробел).
- `strncat()` добавляет `int_to_str` в `log_buf`.
- `strlen(int_to_str)` гарантирует, что `strncat()` не запишет больше, чем нужно.

4. `my_write()`:

```
static ssize_t my_write(struct file *f, char __user *buf, size_t len, loff_t *off) {
    printk(KERN_INFO "Driver: write()\n");

    if(len > BUF_SIZE) {
        return 0;
    }

    if (copy_from_user(input_buf, buf, len) != 0) {
      return -EFAULT;
    }

    if(log_off >= BUF_SIZE - 4) {
      return 0;
    }

    int count = count_sym(len);
    log_number(count);
    return len;
}
```

- Если пользователь пытается записать слишком длинную строку, ничего не делаем (`return 0;`).
- Если копирование данных из пользовательского пространства в `input_buf` не удалось, возвращается `-EFAULT` (ошибка памяти).

- `log_off` должен отслеживать, сколько данных записано в `log_buf`.
- `BUF_SIZE - 4` оставляет немного места для записи числа, чтобы не выйти за границы массива.
- Если буфер заполнен, запись не выполняется.
- `count_sym(len)` считает символы.
- `log_number(count)` записывает это число в `log_buf`.
- В итоге возвращается количество записанных байтов.

2.2 Демонстрация

- `lsmod`:

```
  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:~/ch_drv$ make load
sudo insmod ch_drv.ko
ubuntu@ubuntu:~/ch_drv$ lsmod
Module                  Size  Used by
ch_drv                 24576  0
nls_iso8859_1          16384  1
dm_multipath           61440  0
scsi_dh_rdac           20480  0
scsi_dh_emc            20480  0
scsi_dh_alua           32768  0
```

- `dmesg`:

```
[   38.611201] into device "20000000.flash"
[   38.953528] Searching for RedBoot partition table in 20000000.flash at offset 0
[   38.954355] No RedBoot partition table detected in 20000000.flash
[   43.049338] alua: device handler registered
[   43.065138] emc: device handler registered
[   43.077753] rdac: device handler registered
[   46.089520] audit: type=1400 audit(1741902769.904:2): apparmor="STATUS" operati
 name="lsb_release" pid=435 comm="apparmor_parser"
[   46.320688] audit: type=1400 audit(1741902770.136:3): apparmor="STATUS" operati
 name="/usr/lib/NetworkManager/nm-dhcp-client.action" pid=437 comm="apparmor_parse
[   46.320754] audit: type=1400 audit(1741902770.136:4): apparmor="STATUS" operati
 name="/usr/lib/NetworkManager/nm-dhcp-helper" pid=437 comm="apparmor_parser"
[   46.320776] audit: type=1400 audit(1741902770.136:5): apparmor="STATUS" operati
 name="/usr/lib/connman/scripts/dhclient-script" pid=437 comm="apparmor_parser"
[   46.320792] audit: type=1400 audit(1741902770.136:6): apparmor="STATUS" operati
 name="/{,usr/}sbin/dhclient" pid=437 comm="apparmor_parser"
[   46.416405] audit: type=1400 audit(1741902770.232:7): apparmor="STATUS" operati
 name="/usr/bin/man" pid=438 comm="apparmor_parser"
[   46.416462] audit: type=1400 audit(1741902770.232:8): apparmor="STATUS" operati
 name="man_filter" pid=438 comm="apparmor_parser"
[   46.416482] audit: type=1400 audit(1741902770.232:9): apparmor="STATUS" operati
 name="man_groff" pid=438 comm="apparmor_parser"
[   46.524679] audit: type=1400 audit(1741902770.340:10): apparmor="STATUS" operat
" name="nvidia_modprobe" pid=439 comm="apparmor_parser"
[   46.524737] audit: type=1400 audit(1741902770.340:11): apparmor="STATUS" operat
" name="nvidia_modprobe//kmod" pid=439 comm="apparmor_parser"
[  271.691429] ch_drv: loading out-of-tree module taints kernel.
[  271.694087] ch_drv: module verification failed: signature and/or required key m
[  271.708905] Hello!
```

- Использование:

```
ubuntu@ubuntu:~/ch_drv$ echo "something" > /dev/var1
ubuntu@ubuntu:~/ch_drv$ cat /dev/var1
9 ubuntu@ubuntu:~/ch_drvecho "{1}d" > /dev/var1
ubuntu@ubuntu:~/ch_drv$ cat /dev/var1
9 4 ubuntu@ubuntu:~/ch_drv$
```

3 Вывод

В рамке этой лабораторной работы я ознакомился с процессом создания драйверов устройств с использованием ОС Linux.