Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика, искусственный интеллект и системы управления»

КАФЕДРА ИУ7 «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Драйвер геймпада Logitech F310 в качестве мыши и клавиатуры»

Студент группы ИУ7-75Б _____ А. К. Клименко

(Подпись, дата)

Руководитель _____ Н. Ю. Рязанова

(Подпись, дата)

Москва, 2022 г.

РЕФЕРАТ

Курсовая работа по дисциплине «Операционные системы» на тему «Драйвер геймпада Logitech F310 в качестве мыши и клавиатуры», студента Клименко А. К.

Работа изложена на 25 страницах машинописного текста. Состоит из: введения, 4 разделов, заключения и списка литературы из 11 источников.

Ключевые слова: загружаемый модуль ядра; драйвер геймпада; ОС Linux.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель данной работы – написать драйвер геймпада Logitech F310 для использования его в качестве мыши и клавиатуры.

Для достижения поставленной цели необходимо проанализировать протокол взаимодействия геймпада с подсистемой USB, ...

1 Аналитический раздел

1.1 Драйвер устройства в ОС Linux

Linux – ”модульная” операционная система. Для расширения функционала используются загружаемые модули ядра.

Прикладные программы не могут обращаться к устройствам напрямую. Вся работа с устройствами должна происходить с использованием средств, предоставляемых операционной системой. Реализация взаимодействия операционной системы с новым внешним устройством требует написания драйвера – управляющей программы.

Драйверы устройств – частный случай модуля ядра (actually – модуль ядра в качестве своей составной части может включать в себя код драйвера устройства).

1.2 Подсистема USB

В настоящее время существует множество различных протоколов физического уровня (?), которые используются при подключении внешних устройств. Для устройств ввода, таких как мыши, клавиатуры и геймпады (Human Interaction Device (HID)) наибольшее распространение получил протокол USB.

В ядре операционной системы Linux имеется подсистема предназначенная для работы с USB-устройствами.

Так как целевое устройство – геймпад – подключается по USB, в дальнейшем будет использована именно эта подсистема. (не нужно писать драйвер с ”чистого листа”).

1.2.1 Протокол взаимодействия геймпада Logitech F310

Спецификацию работы стандартного драйвера xpad для большинства геймпадов можно прочитать здесь [https://docs.kernel.org/input/gamepad.html].

1.3 Подсистемы ввода

Существует множество типов устройств ввода. Примерами могут являться различные датчики температуры, давления, акселерометры, АЦП и ЦАП, и др.

В ядре Linux есть подсистема промышленного ввода-вывода (Industrial I/O) которая предназначена для поддержки устройств, которые в том или ином смысле производят аналого-цифровые и/или цифро-аналоговые преобразования. Данная подсистема была разработана чтобы заполнить пропасть между уже имеющимися подсистемами hwmon и input. Подсистема hwmon предназначена для датчиков с низкой частотой дискретизации, используемых для мониторинга и управления самой системой, таких как регулирование скорости вентилятора или измерение температуры. Подсистема input, как следует из ее названия, ориентирована на устройства ввода для взаимодействия с человеком (клавиатура, мышь, сенсорный экран). В некоторых случаях они значительно пересекаются с промышленным вводом-выводом.

//TODO: возможно, стоит рассмотреть Industrial I/O (iio) и процессы взаимодействия с ним.

Для реализации управления мышью через геймпад можно использовать джойстики (один для перемещения мыши, второй для управления колесиком). Кнопки геймпада A и B можно назначить на кнопки мыши (левую и правую соответственно).

1.4 Виртуальная клавиатура

Ввод символов с использованием геймпада – нетривиальная задача. Одним из возможных вариантов является использование виртуальной клавиатуры и указателя, который можно перемещать по ней с помощью D-pad секции на геймпаде. Ввод символа под указателем можно осуществлять нажатием кнопки

X на геймпаде. Таким образом можно вводить любой символ, располагающийся на виртуальной клавиатуре.

Однако для удобства пользователя нужно иметь возможность отобразить на экране виртуальную клавиатуру вместе с текущей позицией указателя.

Так как работа с дисплеем напрямую из ядра требует учета множества факторов (например таких как установленный оконный менеджер, состояние дисплея) управление отображением виртуальной клавиатуры становится трудоёмкой задачей.

Более оптимальным вариантом является написание демона, который по запросу будет выводить на экран виртуальную клавиатуру. Благодаря наличию большого числа графических библиотек уровня пользователя, отображение клавиатуры из демона является посильной задачей даже для непрофессионала.

1.5 Взаимодействие драйвера и демона

В операционной системе Linux существует множество способов передачи информации из ядра в пространство пользователя и наоборот.

– сокеты семейства AF_UNIX.

– программные каналы.

– ...

Выводы

Для управления мышью и клавиатурой с использованием геймпада необходимо написать загружаемый модуль ядра с USB-драйвером, а также демона, предоставляющего сервис – отображение виртуальной клавиатуры.

При написании драйвера будет использована подсистема ввода (input) так как она предназначена именно для ввода информации, поступающего от человека, а не от измерительных приборов и т.п.

Задача демона будет заключаться в прослушивании сокета(рили ???? может так и сделать?.. Или все же обычный файл в проце будет эффективнее, да) и отслеживании поступающих событий. По запросу он должен открывать окно с виртуальной клавиатурой и отображать перемещение виртуального указателя.

2 Конструкторский раздел

В данном разделе приведены ключевые алгоритмы использовавшиеся при написании драйвера и демона с описанием в виде схем.

2.1 Алгоритм обработки URB

На рисунке 1 приведена схема алгоритма обработки URB.

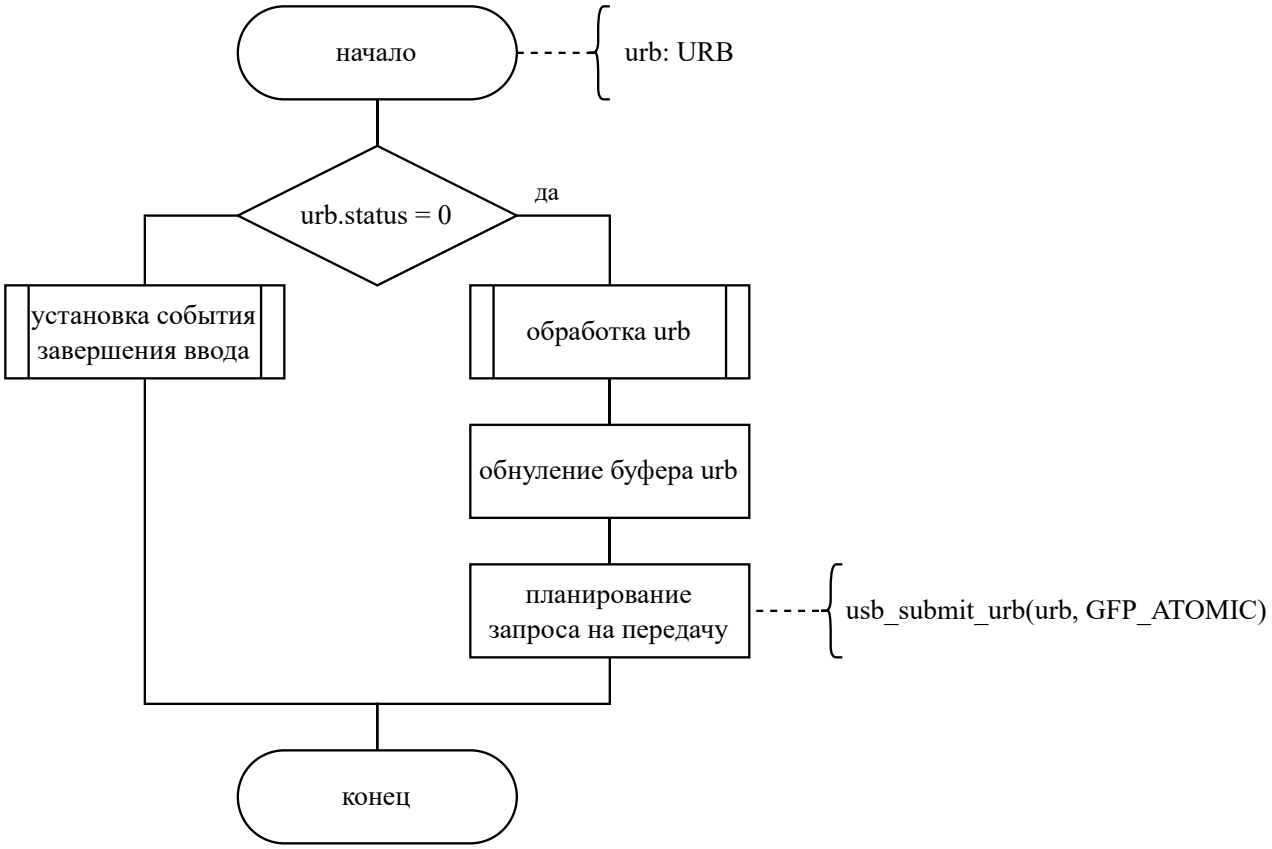

Рисунок 1 – Схема алгоритма обработки URB

На рисунке 2 приведена схема алгоритма обработки корректного URB пакета и генерации необходимых сообщений о вводе.

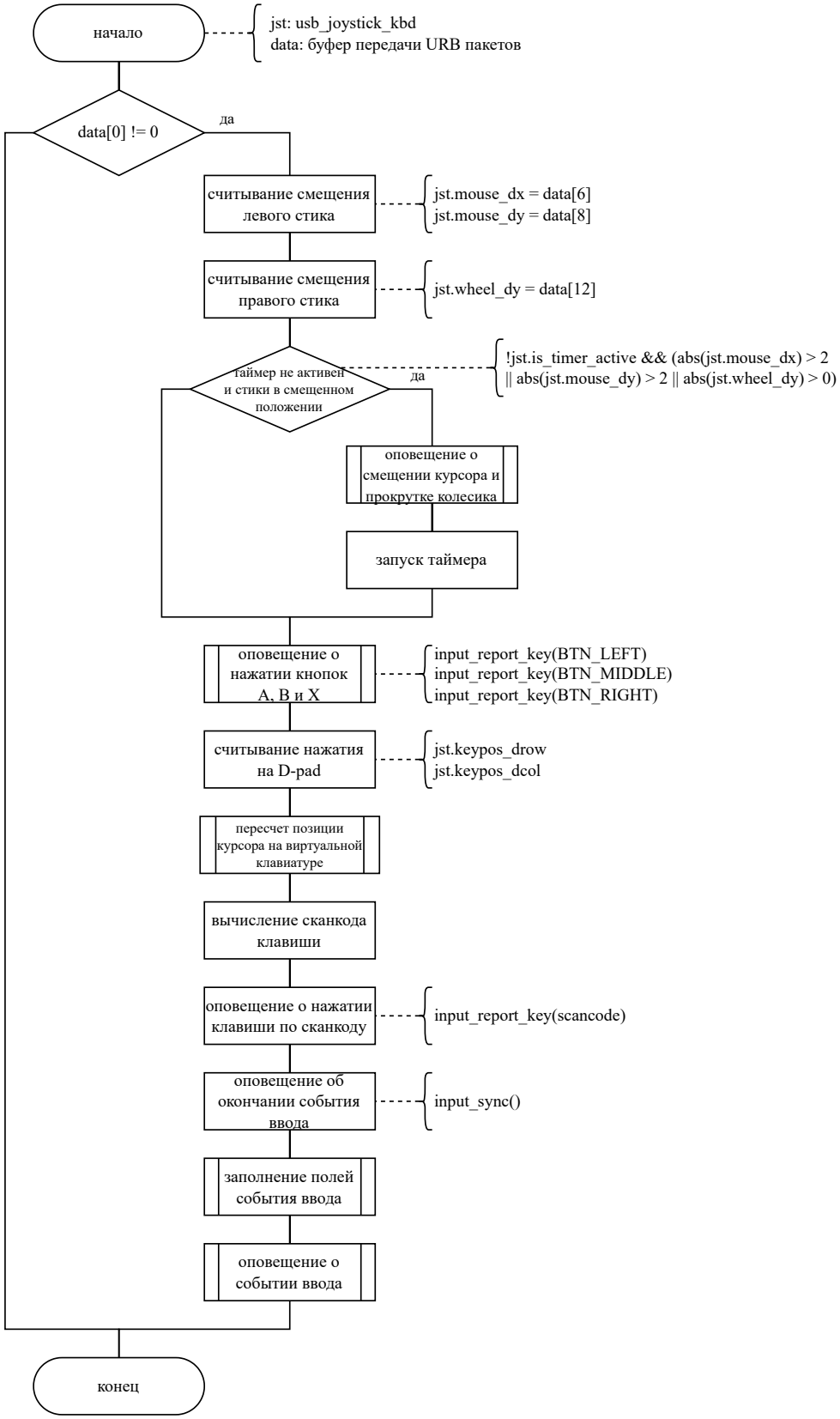

Рисунок 2 – Схема алгоритма обработки корректного URB пакета

2.2 Алгоритмы работы с событиями ввода

На рисунке 3 представлена схема алгоритма издания события с пробуждением ждущих процессов.

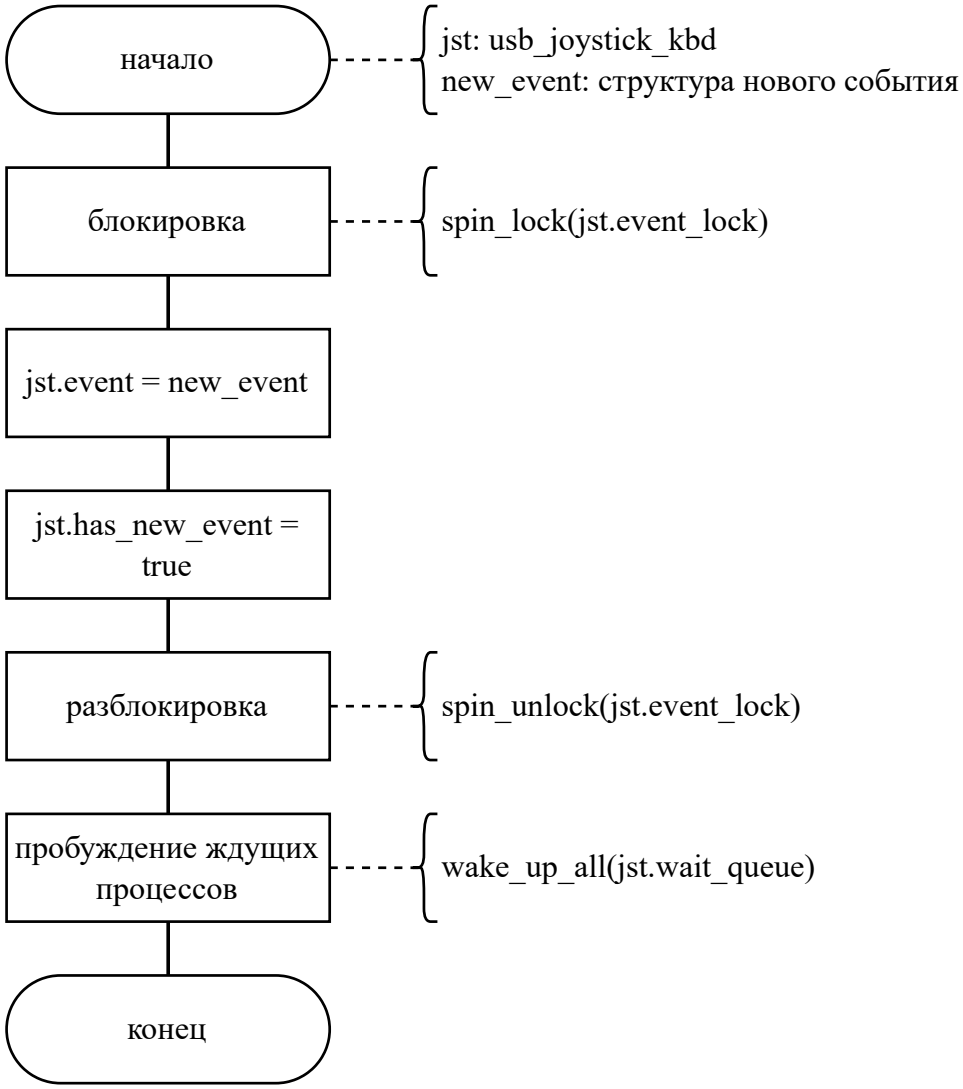

Рисунок 3 – Схема алгоритма издания события

На рисунке 4 приведена схема алгоритма чтения события с блокировкой и ожиданием.

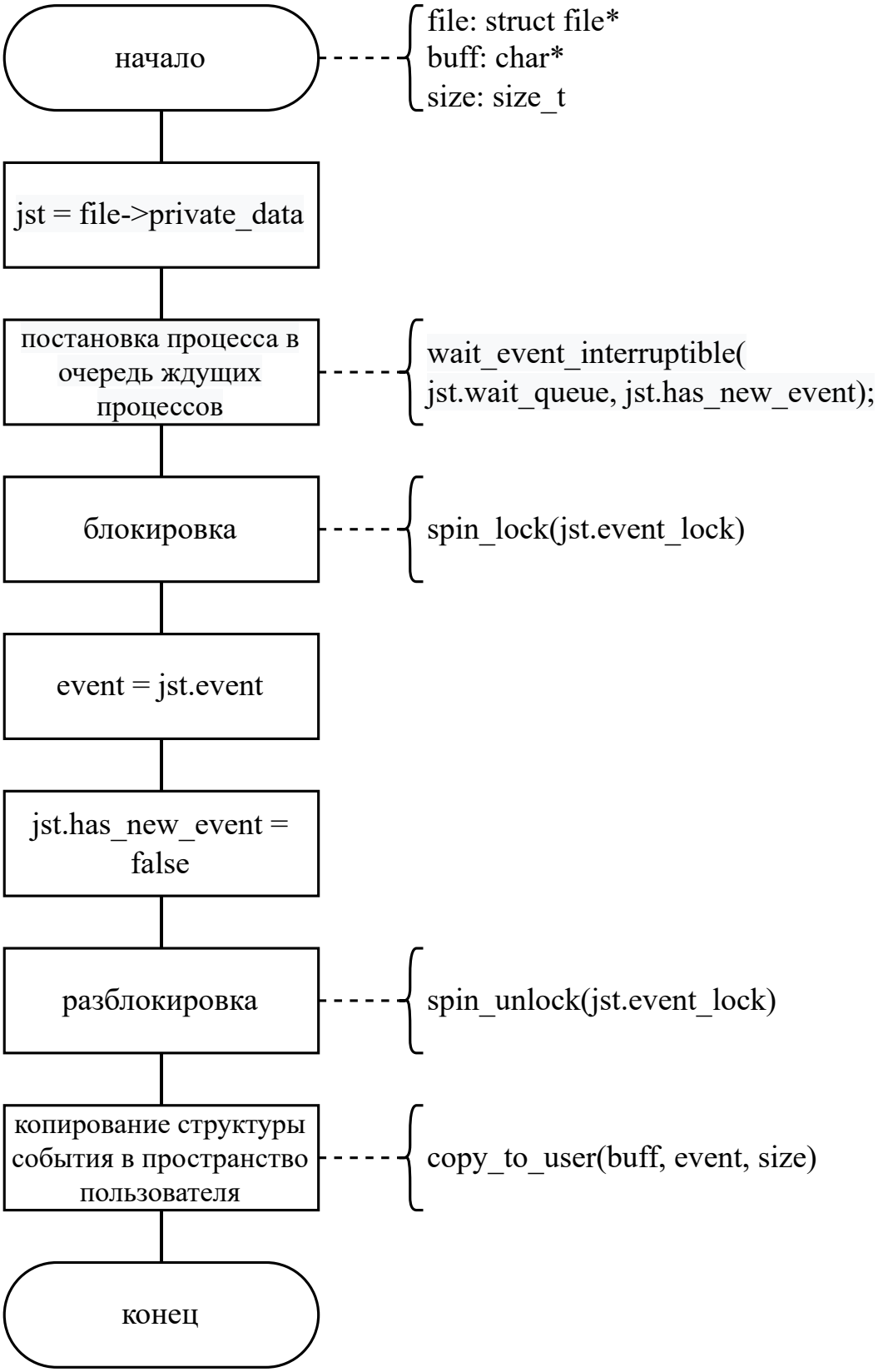

Рисунок 4 – Схема алгоритма чтения события из файла

3 Технологический раздел

На момент написания данной работы новейшей версией ярда Linux была версия 6.1. LTS версия ядра – 5.15.

Загружаемый модуль ядра будет написан для версий ядра 5.15 и 6.1. Для учета особенностей каждой из версий будет использована условная компиляция.

Демон будет написан в виде сервиса утилиты systemd.

3.1 Выбор средств реализации

Несмотря на то, что в ядро Linux с версии 6.1 добавлена поддержка языка программирования Rust, для написания драйвера был выбран язык программирования C, так как внедрение языка Rust является экспериментальной особенностью.

Для написания демона был выбран язык программирования C++. Также было решено использовать библиотеку Qt для создания и отображения виртуальной клавиатуры.

Для сборки обоих программ будет использована утилита Make.

В качестве среды разработки была выбран редактор VSCode.

3.2 Реализация алгоритмов

На листинге 1 приведена реализация обработки корректного URB пакета.

Листинг 1: Реализация алгоритма обработки корректного блока URB

```
void dispatch_joystick_input(struct usb_joystick_kbd *jst)
{
    struct joystick_event new_event;
    unsigned char *data;
    unsigned char *prev_data;
    unsigned char keycode;
    int keypos_d_row; // virtual keyboard cursor offset
    int keypos_d_col; //

    data = jst->transfer_buffer;
    prev_data = jst->prev_data;
    if (data[0] != 0x00)
        return;

    jst->mouse_dx = data[6]; // left stick
    jst->mouse_dy = data[8];
    jst->wheel_dy = data[12]; // right stick
    if (((abs(jst->mouse_dx) > 2) || (abs(jst->mouse_dy) > 2) ||
         (abs(jst->wheel_dy) > 0)) && !jst->is_timer_active)
    {
        input_report_rel(jst->input_dev, REL_X, jst->mouse_dx);
        input_report_rel(jst->input_dev, REL_Y, jst->mouse_dy);
        input_report_rel(jst->input_dev, REL_WHEEL, jst->wheel_dy);

        mod_timer(&jst->timer, jiffies + TIMER_PERIOD);
        jst->is_timer_active = true;
    }

    input_report_key(jst->input_dev, BTN_LEFT, data[3] & 0x10);
    input_report_key(jst->input_dev, BTN_RIGHT, data[3] & 0x20);
    input_report_key(jst->input_dev, BTN_MIDDLE, data[3] & 0x40);

    // read D-pad input for virtual keyboard cursor movement
    keypos_d_row = ((int)(data[2] >> 1) & 1) - ((int)(data[2] >> 0) & 1);
```

```
        keypos_d_col = ((int)(data[2] >> 3) & 1) - ((int)(data[2] >> 2) & 1);

        keycode = move_keyboard_cursor(&jst->keypos_row, &jst->keypos_col,0,0);
        input_report_key(jst->input_dev, keycode, 0); // release old key

        // get new key
        keycode = move_keyboard_cursor(&jst->keypos_row, &jst->keypos_col,
            keypos_d_row, keypos_d_col);

        input_report_key(jst->input_dev, keycode, data[3] & BIT(0));
        input_sync(jst->input_dev);

        new_event.keyboard_cursor_row = jst->keypos_row;
        new_event.keyboard_cursor_col = jst->keypos_col;

        memcpy(prev_data, data, PACKET_LEN);

        spin_lock(&jst->event_lock);
        jst->event = new_event;
        jst->has_new_event = true;
        spin_unlock(&jst->event_lock);
        wake_up_all(&jst->wq);
    }
```

На листинге 2 приведена реализация функций работы с файлом в виртуальной файловой системе /proc.

Листинг 2: Реализация функций работы с файлом событий

```
static struct proc_ops proc_ops = {
    .proc_open = proc_open,
    .proc_read = proc_read,
};

struct proc_dir_entry *create_joystick_event_entry
    (struct usb_joystick_kbd *usb_joystick_kbd)
{
    return proc_create_data("joystick_kbd", S_IRUGO, NULL, &proc_ops,
```

```
10              usb_joystick_kbd);
11      }
12
13      int proc_open(struct inode *inode, struct file *file)
14      {
15      #if LINUX_VERSION_CODE < KERNEL_VERSION(5, 17, 0)
16          file->private_data = PDE_DATA(inode);
17      #else
18          file->private_data = pde_data(inode);
19      #endif
20          return 0;
21      }
22
23      ssize_t proc_read(struct file *file, char __user *buff, size_t size,
24          loff_t *offset)
25      {
26          struct usb_joystick_kbd *joystick_kbd = file->private_data;
27          struct joystick_event event;
28
29          wait_event_interruptible(joystick_kbd->wq,joystick_kbd->has_new_event);
30
31          spin_lock(&joystick_kbd->event_lock);
32          event = joystick_kbd->event;
33          joystick_kbd->has_new_event = false;
34          spin_unlock(&joystick_kbd->event_lock);
35
36          size = min(size, sizeof(struct joystick_event));
37          if (copy_to_user(buff, &event, size))
38          {
39              printk(KERN_ERR MOD_PREFIX "failed to copy to user in read");
40              return -1;
41          }
42
43          return size;
44      }
```

На листинге 3 представлена реализация функций для работы с устройством ввода подсистемы `input`.

Листинг 3: Реализация функций для работы с устройством ввода подсистемы `input`

```
struct input_dev *allocate_joystick_input_dev(struct usb_device *usb_dev)
{
    struct input_dev *input_dev = devm_input_allocate_device(&usb_dev->dev);
    if (input_dev != NULL)
    {
        usb_to_input_id(usb_dev, &input_dev->id);
        input_dev->name = INPUT_DEV_NAME;
        input_dev->open = input_open;
        input_dev->close = input_close;
    }
    return input_dev;
}

int input_open(struct input_dev *input_dev)
{
    struct usb_joystick_kbd *jst = input_get_drvdata(input_dev);
    timer_setup(&jst->timer, input_timer_callback, 0);
    mod_timer(&jst->timer, jiffies + TIMER_PERIOD);
    if (usb_submit_urb(jst->urb, GFP_KERNEL) != 0)
        return -EIO;
    return 0;
}

void input_close(struct input_dev *input_dev)
{
    struct usb_joystick_kbd *jst = input_get_drvdata(input_dev);
    del_timer_sync(&jst->timer);
    usb_kill_urb(jst->urb);
}

void input_timer_callback(struct timer_list *timer)
{
    bool was_update = false;
    struct usb_joystick_kbd *jst = from_timer(usb_joystick_kbd,
        timer, timer);

```

```
37          if (abs(jst->mouse_dx) > 2)
38          {
39              input_report_rel(jst->input_dev, REL_X, jst->mouse_dx);
40              was_update = true;
41          }
42          if (abs(jst->mouse_dy) > 2)
43          {
44              input_report_rel(jst->input_dev, REL_Y, jst->mouse_dy);
45              was_update = true;
46          }
47          if (abs(jst->wheel_dy) > 0)
48          {
49              input_report_rel(jst->input_dev, REL_WHEEL, jst->wheel_dy);
50              was_update = true;
51          }
52
53          if (was_update)
54          {
55              input_sync(jst->input_dev);
56              mod_timer(timer, jiffies + TIMER_PERIOD);
57          }
58          else
59              jst->is_timer_active = false;
60      }
```

3.3 Файлы конфигурации сервисов

Для запуска программы в виде демона с использованием утилиты systemd необходимо создать юнит-файл с конфигурацией сервиса. На листинге 4 приведено содержимое файла `joystick_virt_kbd.service`.

Листинг 4: Конфигурационный файл `joystick_virt_kbd.service`

```
1   [Unit]
2   Description=Joystick virtual keyboard service
3
```

```
[Service]
Type=exec
Restart=always
RestartSec=1
EnvironmentFile=/var/lib/joystick_virt_kbd/.env
ExecStart=/usr/bin/env joystick_virt_kbd

[Install]
WantedBy=multi-user.target
```

На листинге 5 представлена часть файла сборки, отвечающая за установку и удаление сервиса из подсистемы `systemd`.

Листинг 5: makefile для сборки демона

```
EXECUTABLE := daemon

# install paths
APP_INSTALL_PATH := /usr/local/bin/joystick_virt_kbd
SERVICE_INSTALL_PATH := /etc/systemd/system
PRIVATE_DATA_PATH := /var/lib/joystick_virt_kbd

install: $(EXECUTABLE)
    sudo cp $(EXECUTABLE) $(APP_INSTALL_PATH)
    sudo cp joystick_virt_kbd.service $(SERVICE_INSTALL_PATH)
    sudo mkdir -p $(PRIVATE_DATA_PATH)
    sudo -E sh -c 'env > $(PRIVATE_DATA_PATH)/.env'
    sudo systemctl daemon-reload
    sudo systemctl start joystick_virt_kbd

uninstall:
    sudo systemctl stop joystick_virt_kbd
    sudo rm -rf $(PRIVATE_DATA_PATH) $(APP_INSTALL_PATH) \
        $(SERVICE_INSTALL_PATH)/joystick_virt_kbd.service
    sudo systemctl daemon-reload

# остальные правила для сборки программы ...
```

4 Исследовательский раздел

В данном разделе будет проведено исследование зависимости времени обработки URB от (???).

ИЛИ зависимость времени обработки URB от установленного оконного менеджера.

ДА НЕ, БРЕД КАКОЙ-ТО.

4.1 Технические характеристики

Исследование будет проводиться на ноутбуке Dell Vostro [можно ссылочку]. Характеристики системы приведены ниже.

- Процессор: .
- Оперативная память: 8 ГБ.
- Операционная система: Arch Linux.
- Графическая оболочка: KDE.

4.2 Описание исследования

В качестве оконного менеджера будут использованы Wayland [ссылочку] и X11 [ссылочку] как одни из наиболее распространенных [тоже можно ссылочку].

А МОЖНО ДАЖЕ И ЕЩЕ ЧТО-НИБУДЬ ПОСТАВИТЬ. (НЕТ)

Замер времени будет (не будет).

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. aaa

ПРИЛОЖЕНИЕ А