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 для использования его 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 для использования его в качестве мыши и клавиатуры», студента Клименко А. К.

Работа изложена на 41 страницах машинописного текста. Состоит из: введения, 4 разделов, заключения и списка литературы из 10 источников. Содержит 10 рисунков.

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

СОДЕРЖАНИЕ

РЕФЕРАТ **3**

ВВЕДЕНИЕ **6**

1 Аналитический раздел **7**

1.1 Постановка задачи 7

1.2 Анализ особенностей работы геймпада 7

1.3 Подсистема USB 8

1.3.1 Способы передачи URB пакетов 9

1.4 Подсистема ввода для интерактивных устройств 10

1.4.1 Использование геймпада в качестве мыши 10

1.4.2 Использование геймпада в качестве клавиатуры 11

1.5 Взаимодействие драйвера и демона 12

2 Конструкторский раздел **14**

2.1 Структура ПО 14

2.2 Последовательность преобразований 15

2.3 Алгоритм обработки URB 15

2.4 Алгоритм генерации события перемещения указателя 17

2.5 Алгоритм чтения события перемещения указателя 17

3 Технологический раздел **19**

3.1 Выбор языка и среды разработки 19

3.2 Структуры драйвера 19

3.3 Точки входа драйвера 21

3.4 Функция обработки URB пакета 24

3.5 Функция чтения файла событий 26

3.6 Точки входа подсистемы ввода для интерактивных устройств 27

3.7 Файлы конфигурации сервисов 29

3.8 Сборка и запуск 30

4 Исследовательский раздел **31**

4.1 Исследуемые параметры 31

4.2 Результаты исследования 31

ЗАКЛЮЧЕНИЕ **33**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **34**

ПРИЛОЖЕНИЕ А **36**

ПРИЛОЖЕНИЕ Б **38**

ПРИЛОЖЕНИЕ В **39**

ПРИЛОЖЕНИЕ Г **40**

ВВЕДЕНИЕ

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

1 Аналитический раздел

1.1 Постановка задачи

В соответствии с техническим заданием, требуется написать драйвер геймпада Logitech F310 для использования его в качестве мыши и клавиатуры.

Для решения поставленной задачи необходимо:

— проанализировать особенности работы геймпада,

— рассмотреть способы получения данных от устройства с использованием подсистемы USB,

— исследовать подсистему ввода для интерактивных устройств,

— разработать драйвер в виде загружаемого модуля ядра,

— провести исследование разработанного драйвера.

1.2 Анализ особенностей работы геймпада

Устройство геймпада Logitech F310 приведено на рисунке 1.

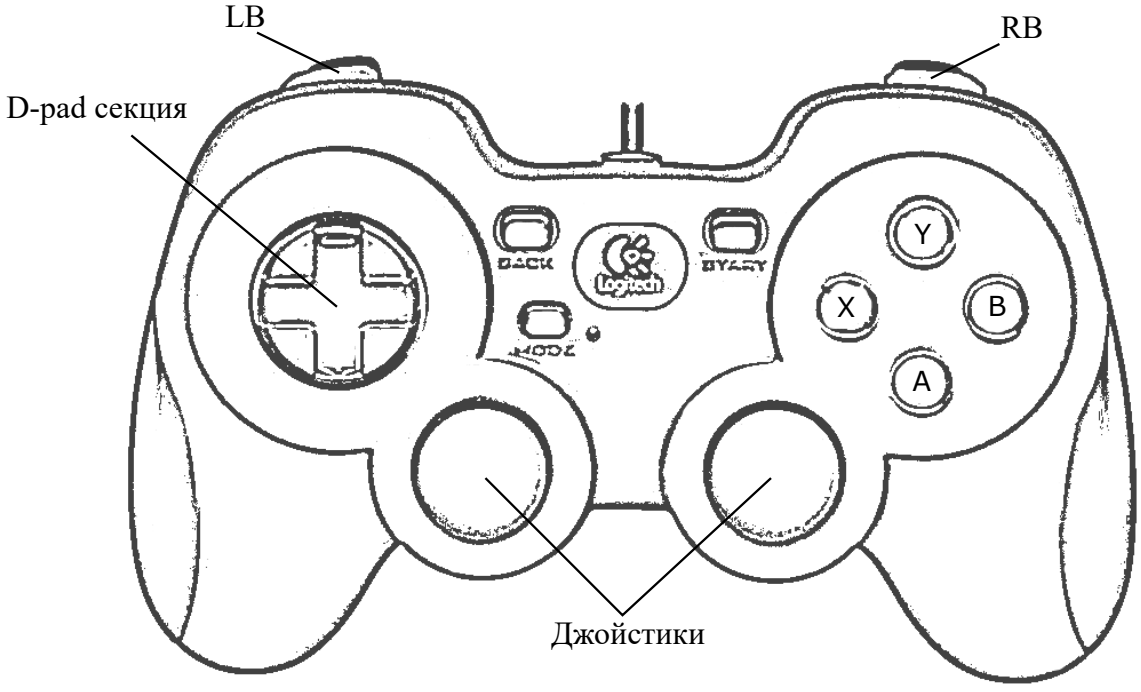

Рисунок 1 – Геймпад Logitech F310

В составе геймпада имеются два джойстика, положение которых кодируется двумя независимыми парами величин – смещениями вдоль горизонтальной и вертикальной осей соответственно (рисунок 2). Смещение может принимать целочисленные значения в интервале от -127 до 128.

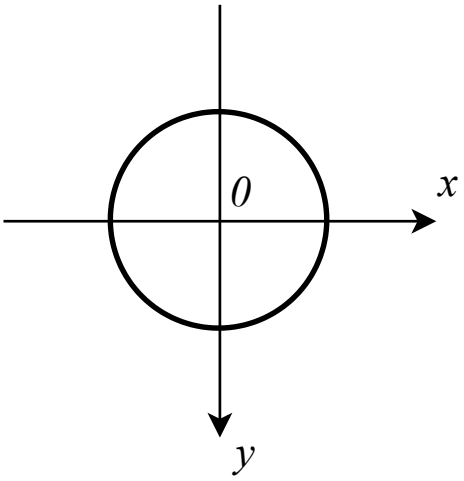

Рисунок 2 – Система координат джойстика

Так как геймпад Logitech F310 подключается по USB, при написании драйвера будет использована подсистема USB.

1.3 Подсистема USB

USB драйвер имеет две точки входа – `probe` и `disconnect`, которые вызываются подсистемой USB при подключении и отключении устройства, соответствующего данному драйверу.

Обмен информацией между подключенным устройством и драйвером осуществляется пакетами по запросу. Блок запроса USB (USB Request Block – URB) представляется структурой ядра `urb`:

```
struct urb {
    struct list_head urb_list;
    struct usb_device *dev; /* (in) pointer to associated device */
    unsigned int pipe; /* (in) pipe information */
```

```
    int status; /* (return) non-ISO status */
    void *transfer_buffer; /* (in) associated data buffer */
    dma_addr_t transfer_dma; /* (in) dma addr for transfer_buffer */
    u32 transfer_buffer_length; /* (in) data buffer length */
    u32 actual_length; /* (return) actual transfer length */
    int interval; /* (modify) transfer interval */
    void *context; /* (in) context for completion */
    usb_complete_t complete; /* (in) completion routine */
    /* ... */
};
```

1.3.1 Способы передачи URB пакетов

Передача URB пакетов по шине USB является дуплексной и может происходить в одной из четырех форм, в зависимости от подключенного устройства:

- `Control` – используется для передачи управляющих команд для настройки устройства или получения его статуса.
- `Bulk` – используется для обмена большими пакетами данных. Зачастую именно эта форма передачи используется устройствами на подобие сканеров и SCSI адаптеров.
- `Interrupt` – используется для запроса передачи небольших пакетов в режиме опроса устройства. Если был запрошен пакет в режиме прерывания, то драйвер хост-контроллера автоматически будет повторять этот запрос с определенным интервалом (1–255 мс).
- `Isochronous` – используется для передачи данных в реальном времени с гарантированной пропускной способностью шины, но ненадежно. В общем случае изохронный тип используется для передачи аудио и видео информации.

Так как рассматриваемое устройство – геймпад – является HID устройством, и поддерживает формат передачи `Interrupt`, то при реализации драйвера будет использован именно этот формат.

1.4 Подсистема ввода для интерактивных устройств

Подсистема ввода для интерактивных устройств представляет уровень абстракции между устройствами ввода и обработчиками ввода. Устройства ввода фиксируют входные данные от действий пользователя и генерируют входные события, которые, проходя через подсистему ввода, отправляются соответствующим обработчикам. Ядро ввода обеспечивает сопоставление ”многие ко многим” между устройствами ввода и обработчиками событий.

Список наиболее распространенных типов событий, генерируемых с использованием данной подсистемы:

— `EV_KEY` – используется для описания изменений состояния клавиатур, кнопок или других устройств, похожих на клавиши;

— `EV_REL` – используется для описания изменения значения относительной оси, например, перемещения мыши на 5 единиц влево;

— `EV_ABS` – используется для описания изменений значений абсолютной оси, например, описания координат касания на сенсорном экране.

1.4.1 Использование геймпада в качестве мыши

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

Изменение положения курсора на экране происходит посредством генерирования события типа `EV_REL`. Согласно спецификации геймпада [1], движение джойстиков генерирует события типа `EV_ABS`. В связи с этим, реализация движения курсора будет некорректной при простой замене одного типа события на другой – фиксация джойстика в смещенном положении не будет приводить к поступлению новых URB пакетов и генерированию новых событий перемещения мыши, и как следствие не будет происходить изменение положения курсора.

Для решения изложенной проблемы необходимо использовать таймер, который должен с постоянной периодичностью генерировать события при возникновении описанной ситуации. Для работы с таймером в ядре существует структура `timer_list`.

```
struct timer_list {
    struct hlist_node entry;
    unsigned long expires;
    void (*function)(struct timer_list *);
    u32 flags;
};

#define timer_setup(timer, callback, flags) /* ... */
int del_timer(struct timer_list *timer);
```

Запуск таймера должен происходить только тогда, когда джойстики находятся в смещенном положении. Планирование следующего выполнения осуществляется вызовом функции `mod_timer`.

```
#define TIMER_PERIOD (HZ / 100) // 10 ms
mod_timer(&timer, jiffies + TIMER_PERIOD);
```

1.4.2 Использование геймпада в качестве клавиатуры

Ввод символов является задачей не свойственной геймпаду – количество имеющихся кнопок на устройстве не позволяет установить их однозначного соответствия символам, вводимым с клавиатуры.

В качестве решения данной задачи предложено использование виртуальной клавиатуры (рисунок 3) и указателя, перемещение которого осуществляется с использованием D-pad секции на геймпаде (рисунок 1). Ввод символа под указателем осуществляется нажатием кнопки LB на геймпаде. Таким образом можно вводить любой символ, располагающийся на виртуальной клавиатуре.

Однако для отслеживания позиции указателя нужно иметь возможность отобразить на экране виртуальную клавиатуру вместе с текущей позицией ука-

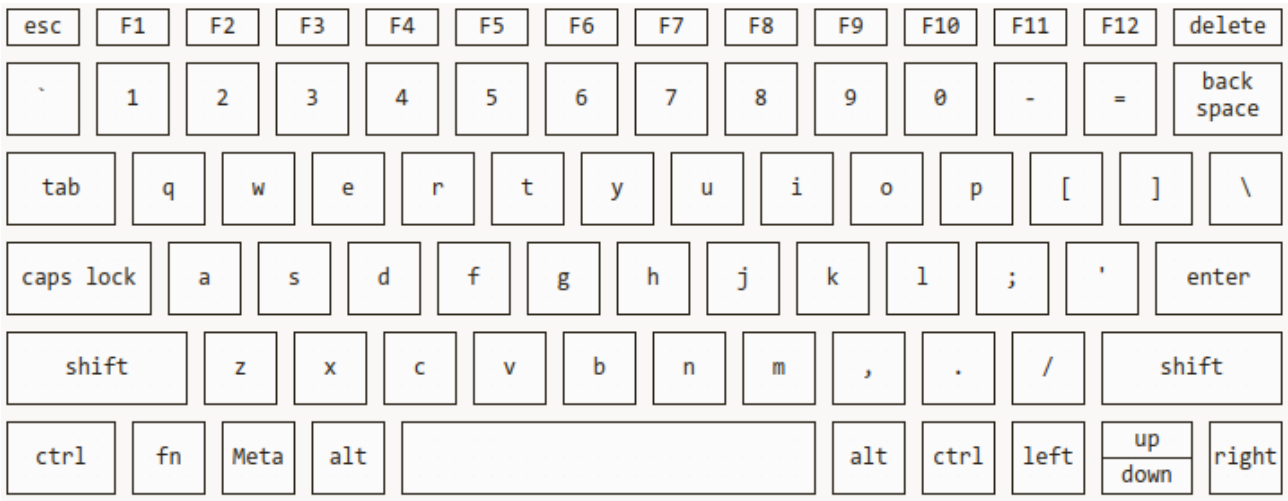

Рисунок 3 – Виртуальная клавиатура

зателя. Так как работа с дисплеем напрямую из ядра требует учета множества факторов, управление отображением виртуальной клавиатуры напрямую из ядра становится трудоемкой задачей.

Более оптимальным вариантом является написание демона, который по запросу будет выводить на экран виртуальную клавиатуру.

1.5 Взаимодействие драйвера и демона

Одним из возможных способов передачи информации из пространства ядра в пространство пользователя является использование виртуальной файловой системы proc [7].

Операции, которые могут быть осуществлены с файлом определяются структурой `proc_ops`. В случае реализации передачи событий из пространства ядра в пространство пользователя, необходимо и достаточно реализовать две операции: `open` и `read`.

При этом, если события не возникают, процесс должен быть заблокирован при попытке чтения. Необходимость блокировки процесса и ожидания появления нового события приводит к использованию очередей ожидания, которые представляются в ядре структурой `wait_queue_head`:

```
struct wait_queue_entry {
```

```
    unsigned int flags;
    void *private;
    wait_queue_func_t func;
    struct list_head entry;
};

struct wait_queue_head {
    spinlock_t lock;
    struct list_head head;
};
```

Блокировка процесса и ожидание возникновения события осуществляется вызовом макроса

`wait_event_interruptible(wq_head, condition).`

Все ждущие в данной очереди процессы пробуждаются посредством вызова макроса `wake_up_all(wq_head)`, после чего будет анализировано выражение `condition` переданное при блокировке. Если оно ложно, то процесс вновь блокируется.

Выводы

Использование геймпада в качестве мыши и клавиатуры имеет ряд особенностей:

1) тип событий, генерируемых геймпадом по спецификации не совместим с типом события перемещения мыши – реализация перемещения мыши требует использования таймеров;
2) непосредственное сопоставление скан-кодов клавиш на клавиатуре с кнопками геймпада невозможно – необходима реализация поддержки виртуальной клавиатуры.

2 Конструкторский раздел

В данном разделе приведены ключевые алгоритмы использовавшиеся при написании драйвера и демона.

2.1 Структура ПО

Разрабатываемое ПО состоит из драйвера геймпада в виде загружаемого модуля ядра, и демона, запускаемого с использованием утилиты `systemd` [8]. Структура ПО представлена на рисунке 4.

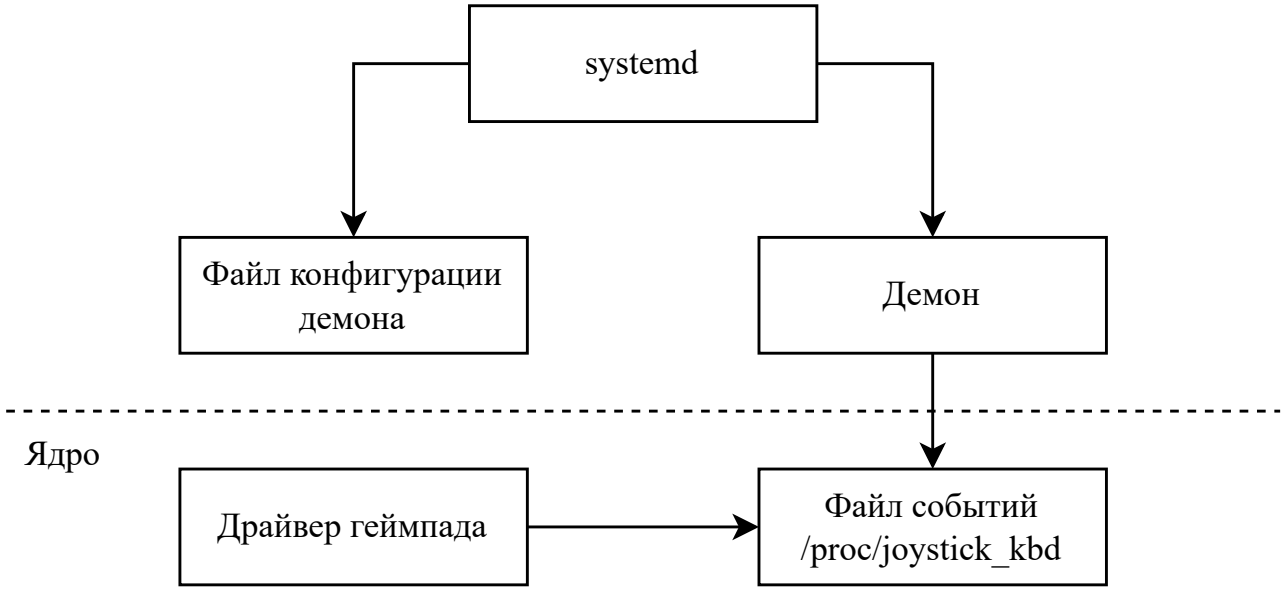

Рисунок 4 – Структура ПО

2.2 Последовательность преобразований

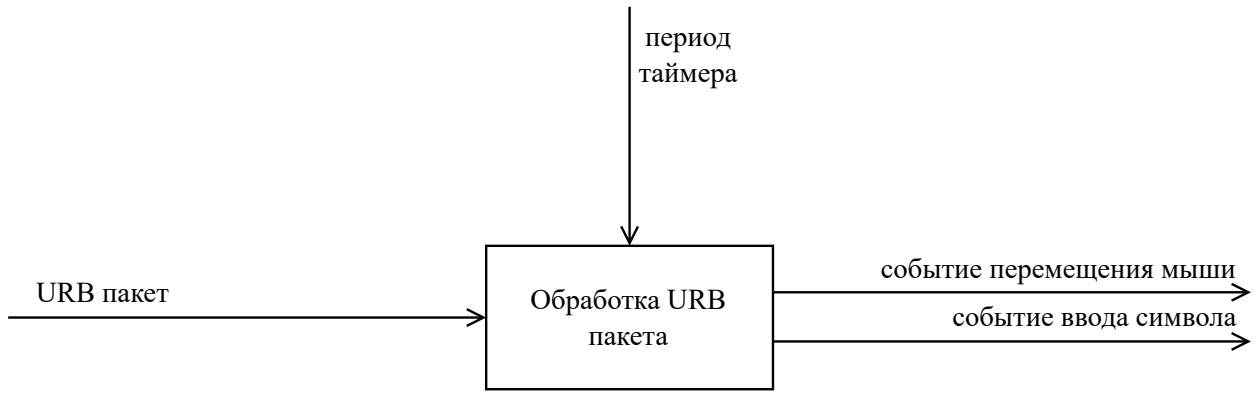

Рисунок 5 – IDEF0-диаграмма процесса обработки URB пакета. (Часть 1)

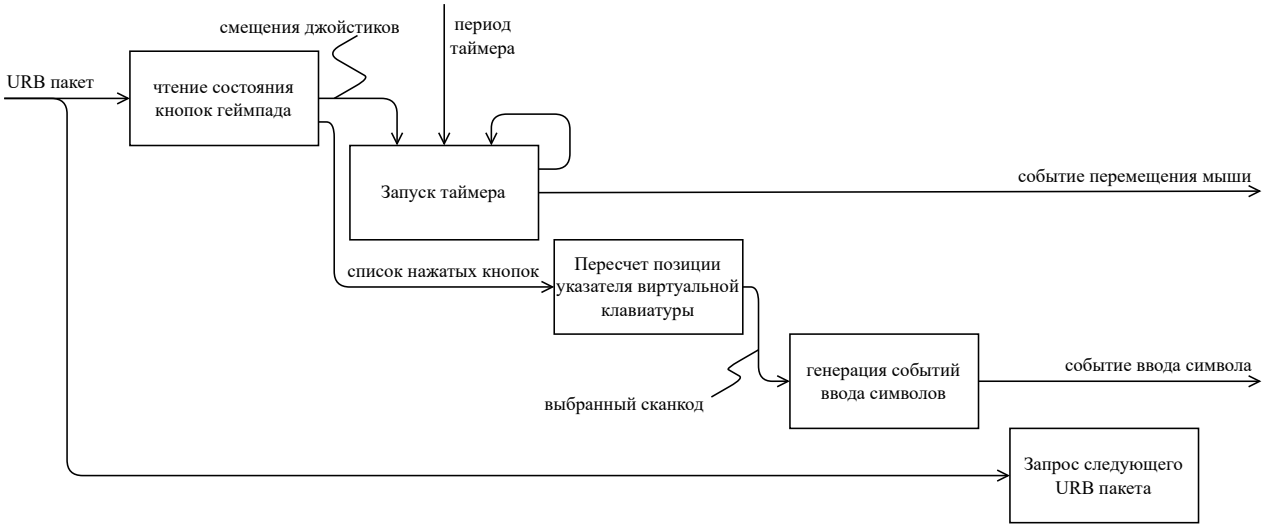

Рисунок 6 – IDEF0-диаграмма процесса обработки URB пакета. (Часть 2)

2.3 Алгоритм обработки URB

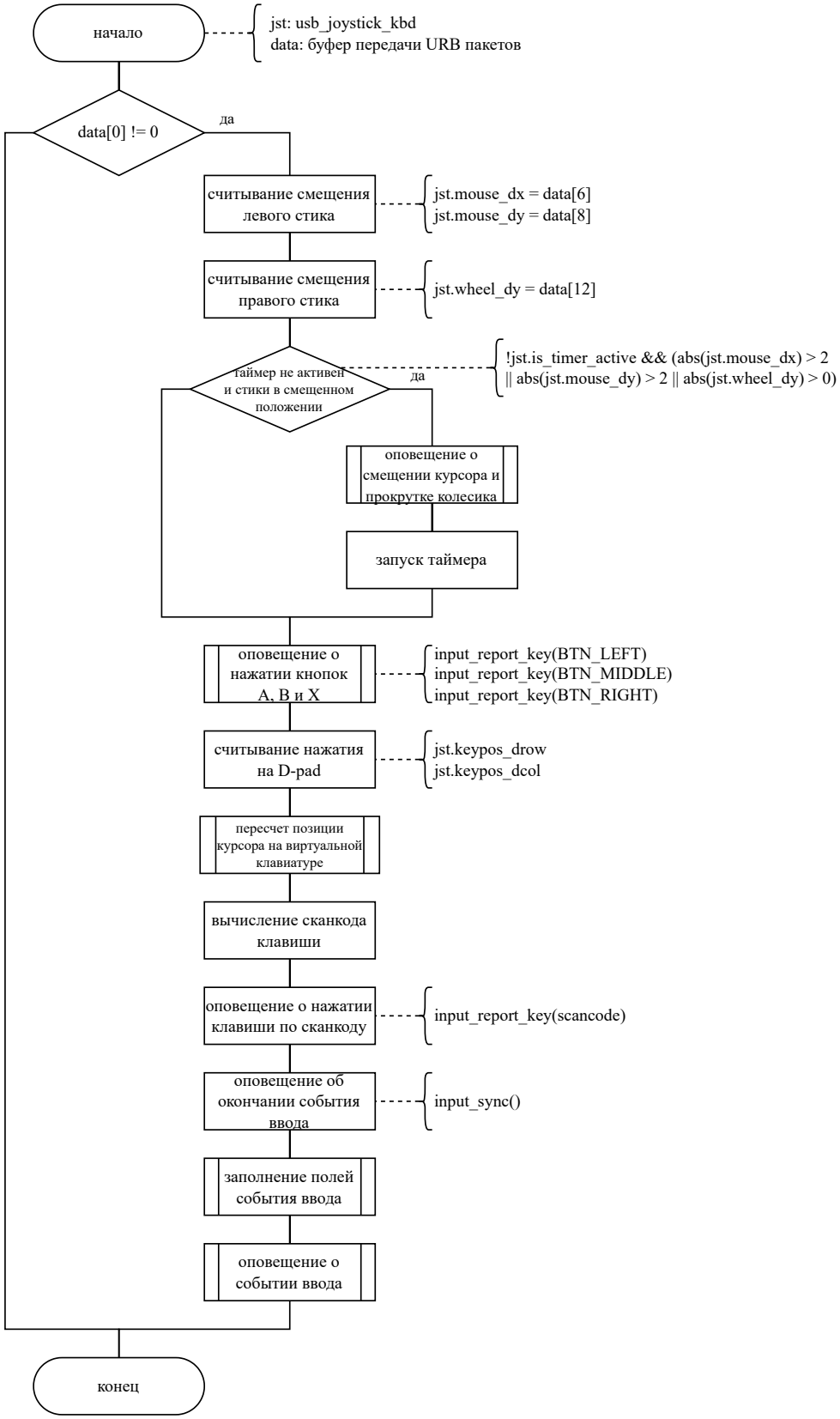

Рисунок 7 – Алгоритм обработки URB пакета

2.4 Алгоритм генерации события перемещения указателя

На рисунке 8 представлен алгоритм генерации события с пробуждением ждущих процессов.

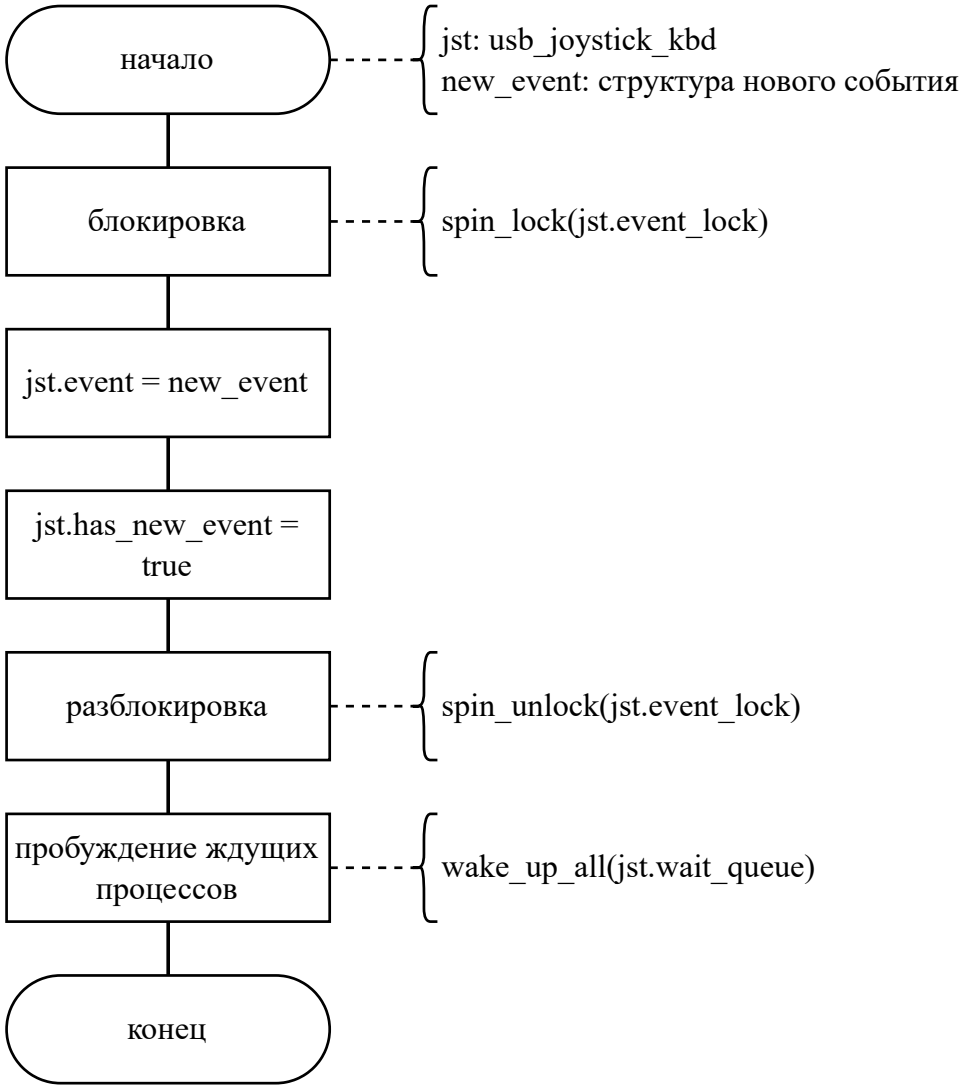

Рисунок 8 – Алгоритм генерации события перемещения указателя виртуальной клавиатуры

2.5 Алгоритм чтения события перемещения указателя

На рисунке 9 приведен алгоритм чтения события с блокировкой и ожиданием.

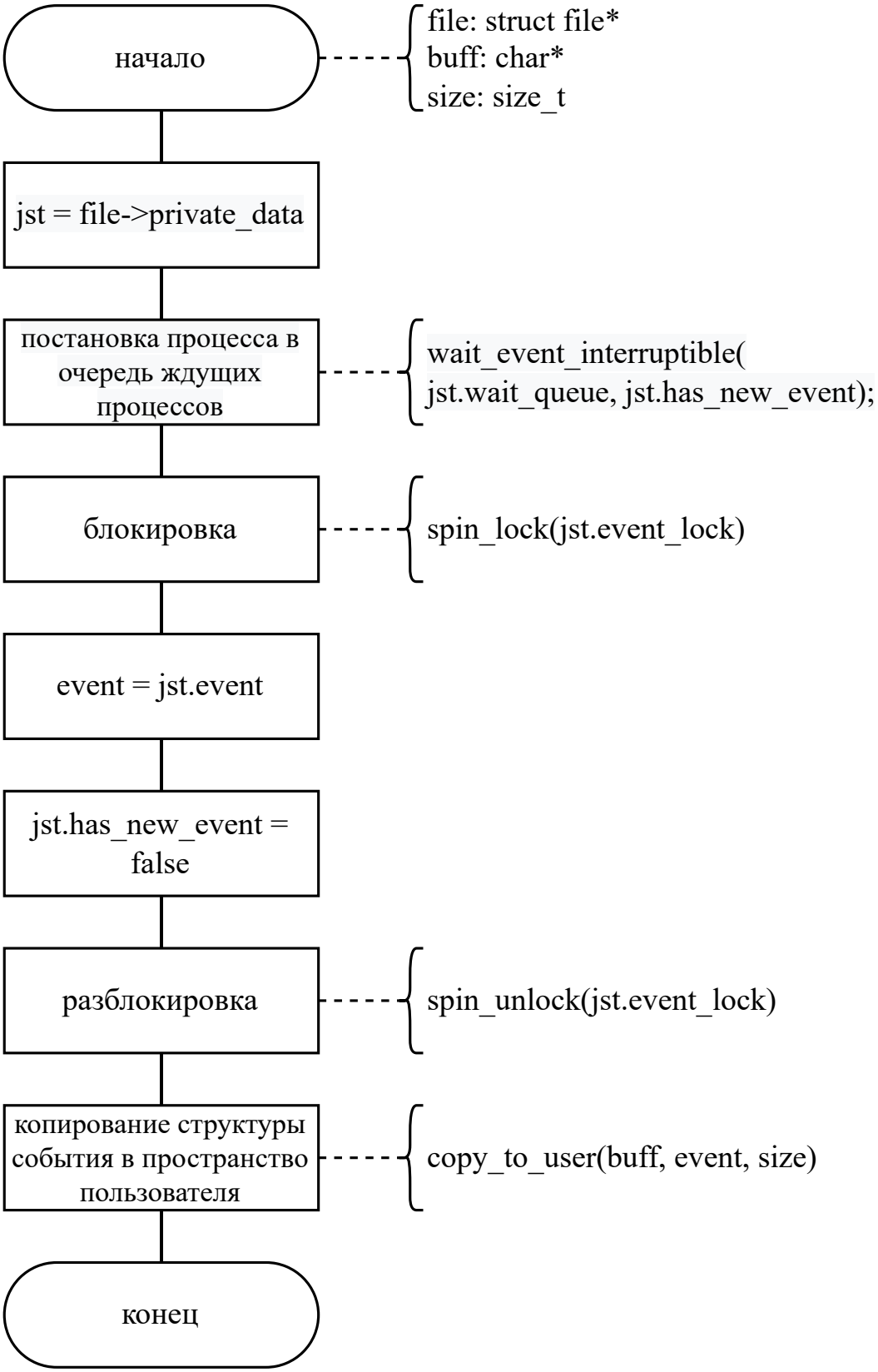

Рисунок 9 – Алгоритм чтения события из файла

3 Технологический раздел

3.1 Выбор языка и среды разработки

Несмотря на то, что в ядро Linux с версии 6.1 добавлена поддержка языка программирования Rust [9], для написания драйвера был выбран язык программирования C, так как язык Rust внедряется в качестве эксперимента.

Для написания демона был выбран язык программирования C++ и библиотека Qt для создания и отображения виртуальной клавиатуры.

Для сборки обоих программ будет использована утилита Make. В качестве среды разработки был выбран редактор VSCode.

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.

3.2 Структуры драйвера

Для хранения данных, связанных с подключенным устройством была объявлена структура, показанная на листинге 1.

Листинг 1: Структура `usb_joystick_kbd`

```
struct usb_joystick_kbd
{
    /* device related section */
    struct usb_device *usbdev;
    struct urb *urb;
    unsigned char *transfer_buffer;
    dma_addr_t dma_addr;

```

```
9         /* mouse input handling stuff */
10        struct input_dev *input_dev;
11        int mouse_dx;
12        int mouse_dy;
13        int wheel_dy;
14        struct timer_list timer;
15        bool is_timer_active;
16
17        /* keyboard input handling */
18        unsigned char *keycodes;
19        unsigned int keypos_row;
20        unsigned int keypos_col;
21
22        // proc entry for events passing between kernel and user space
23        struct proc_dir_entry *proc_entry;
24        struct wait_queue_head wq;
25        bool has_new_event;
26        struct joystick_event event;
27        struct spinlock event_lock;
28        char prev_data[PACKET_LEN];
29    };
```

Для передачи события изменения позиции указателя на виртуальной клавиатуре объявлена структура, показанная на листинге 2.

Листинг 2: Структура `joystick_event`

```
1   struct joystick_event
2   {
3       unsigned int keyboard_cursor_row;
4       unsigned int keyboard_cursor_col;
5   };
```

3.3 Точки входа драйвера

Листинг 3: функция `probe` USB драйвера

```
int probe(struct usb_interface *intf, const struct usb_device_id *id)
{
    int status;
    struct usb_joystick_kbd *jst;
    struct usb_endpoint_descriptor *int_in, *int_out;
    int pipe;

    // allocating memory for usb_joystick_kbd struct
    jst = kzalloc(sizeof(struct usb_joystick_kbd), GFP_KERNEL);
    if (jst == NULL)
    {
        printk(KERN_ERR MOD_PREFIX "failed to allocate memory for struct
            usb_joystick_kbd\n");
        return -ENOMEM;
    }

    /* set up the endpoint. Use only the first int-in endpoint */
    status = usb_find_common_endpoints(intf->cur_altsetting, NULL, NULL,
        &int_in, &int_out);
    if (status != 0)
    {
        printk(KERN_ERR MOD_PREFIX "failed to find common endpoints: %d\n",
            status);
        goto ret1;
    }

    // allocating buffer data for data transfer
    jst->usbdev = interface_to_usbdev(intf);
    jst->transfer_buffer = usb_alloc_coherent(jst->usbdev, PACKET_LEN,
        GFP_KERNEL, &jst->dma_addr);
    if (jst->transfer_buffer == NULL)
    {
        printk(KERN_ERR MOD_PREFIX "failed to allocate transfer buffer\n");
        status = -ENOMEM;
        goto ret1;
```

```
36          }
37
38          // allocating USB request block
39          jst->urb = usb_alloc_urb(0, GFP_KERNEL);
40          if (jst->urb == NULL)
41          {
42              printk(KERN_ERR MOD_PREFIX "failed to allocate urb\n");
43              status = -ENOMEM;
44              goto ret2;
45          }
46
47          // allocate input device
48          jst->input_dev = allocate_joystick_input_dev(jst->usbdev);
49          if (jst->input_dev == NULL)
50          {
51              printk(KERN_ERR MOD_PREFIX "failed to allocate input device\n");
52              status = -ENOMEM;
53              goto ret3;
54          }
55
56          input_set_drvdata(jst->input_dev, jst);
57
58          // link keycodes
59          jst->keycodes = get_keycode_map(&jst->keypos_row, &jst->keypos_col);
60
61          // register created input device
62          status = input_register_device(jst->input_dev);
63          if (status != 0)
64          {
65              printk(KERN_ERR MOD_PREFIX "failed to register input device.
66                  status: %d\n", status);
67              goto ret4;
68          }
69
70          pipe = usb_rcvintpipe(jst->usbdev, int_in->bEndpointAddress);
71
72          // fill urb for interrupt type
73          usb_fill_int_urb(jst->urb, jst->usbdev, pipe, jst->transfer_buffer,
74              PACKET_LEN, jskbd_complete, jst, int_in->bInterval);
75          jst->urb->transfer_dma = jst->dma_addr;
```

```
76          jst->urb->transfer_flags |= URB_NO_TRANSFER_DMA_MAP;
77
78          usb_set_intfdata(intf, jst);
79
80          // creating entry in proc vfs
81          jst->proc_entry = create_joystick_event_entry(jst);
82          if (jst->proc_entry == NULL)
83          {
84              printk(KERN_ERR MOD_PREFIX "could not create proc entry");
85              status = -EINVAL;
86              goto ret5;
87          }
88
89          init_waitqueue_head(&jst->wq);
90          __module_get(THIS_MODULE);
91          return 0;
92
93      ret5:
94          input_unregister_device(jst->input_dev);
95      ret4:
96          input_free_device(jst->input_dev);
97      ret3:
98          usb_free_urb(jst->urb);
99      ret2:
100         usb_free_coherent(jst->usbdev, PACKET_LEN, jst->transfer_buffer,
101             jst->dma_addr);
102     ret1:
103         kfree(jst);
104         return status;
105     }
```

Листинг 4: функция `disconnect` USB драйвера

```
1   void disconnect(struct usb_interface *intf)
2   {
3       struct usb_joystick_kbd *jst = usb_get_intfdata(intf);
4
5       del_timer_sync(&jst->timer);
```

```
6
7         proc_remove(jst->proc_entry);
8
9         // removing custom input device
10        input_unregister_device(jst->input_dev);
11        input_free_device(jst->input_dev);
12
13        // no need to kill. Killed in input close already
14        usb_free_urb(jst->urb);
15        usb_free_coherent(jst->usbdev, PACKET_LEN, jst->transfer_buffer,
16            jst->dma_addr);
17        kfree(jst);
18
19        module_put(THIS_MODULE);
20        printk(KERN_INFO MOD_PREFIX "disconnect\n");
21    }
```

3.4 Функция обработки URB пакета

Листинг 5: Функция обработки URB пакета

```
1   void dispatch_joystick_input(struct usb_joystick_kbd *jst)
2   {
3         struct joystick_event new_event;
4         unsigned char *data;
5         unsigned char *prev_data;
6         unsigned char keycode;
7         int keypos_d_row; // virtual keyboard cursor offset
8         int keypos_d_col; //
9
10        data = jst->transfer_buffer;
11        prev_data = jst->prev_data;
12        if (data[0] != 0x00)
13            return;
14
15        jst->mouse_dx = data[6]; // left stick
16        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);
```

```
57  }
```

3.5 Функция чтения файла событий

Листинг 6: Функция чтения файла событий

```
1   static struct proc_ops proc_ops = {
2       .proc_open = proc_open,
3       .proc_read = proc_read,
4   };
5
6   struct proc_dir_entry *create_joystick_event_entry
7       (struct usb_joystick_kbd *usb_joystick_kbd)
8   {
9       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      }
```

3.6 Точки входа подсистемы ввода для интерактивных устройств

Листинг 7: Функции для работы с устройством ввода подсистемы `input`

```
1   struct input_dev *allocate_joystick_input_dev(struct usb_device *usb_dev)
2   {
3       struct input_dev *input_dev = devm_input_allocate_device(&usb_dev->dev);
4       if (input_dev != NULL)
5       {
6           usb_to_input_id(usb_dev, &input_dev->id);
7           input_dev->name = INPUT_DEV_NAME;
8           input_dev->open = input_open;
9           input_dev->close = input_close;
10      }
11      return input_dev;
12  }
13
14  int input_open(struct input_dev *input_dev)
15  {
16      struct usb_joystick_kbd *jst = input_get_drvdata(input_dev);
17      timer_setup(&jst->timer, input_timer_callback, 0);
18      mod_timer(&jst->timer, jiffies + TIMER_PERIOD);
19      if (usb_submit_urb(jst->urb, GFP_KERNEL) != 0)
```

```
20              return -EIO;
21          return 0;
22      }
23
24      void input_close(struct input_dev *input_dev)
25      {
26          struct usb_joystick_kbd *jst = input_get_drvdata(input_dev);
27          del_timer_sync(&jst->timer);
28          usb_kill_urb(jst->urb);
29      }
30
31      void input_timer_callback(struct timer_list *timer)
32      {
33          bool was_update = false;
34          struct usb_joystick_kbd *jst = from_timer(usb_joystick_kbd,
35              timer, timer);
36
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    }
```

3.7 Файлы конфигурации сервисов

Для запуска программы в виде демона с использованием утилиты systemd необходимо создать юнит-файл с конфигурацией сервиса. На листинге 8 приведено содержимое файла `joystick_virt_kbd.service`.

Листинг 8: Конфигурационный файл `joystick_virt_kbd.service`

```
1   [Unit]
2   Description=Joystick virtual keyboard service
3
4   [Service]
5   Restart=always
6   RestartSec=1
7   EnvironmentFile=/var/lib/joystick_virt_kbd/.env
8   ExecStart=/usr/bin/env joystick_virt_kbd
9
10  [Install]
11  WantedBy=multi-user.target
```

На листинге 9 представлена часть файла сборки, отвечающая за установку и удаление сервиса из подсистемы `systemd`.

Листинг 9: makefile для сборки демона

```
1   EXECUTABLE := daemon
2
3   # install paths
4   APP_INSTALL_PATH := /usr/local/bin/joystick_virt_kbd
```

```
5   SERVICE_INSTALL_PATH := /etc/systemd/system
6   PRIVATE_DATA_PATH := /var/lib/joystick_virt_kbd
7
8   install: $(EXECUTABLE)
9       sudo cp $(EXECUTABLE) $(APP_INSTALL_PATH)
10      sudo cp joystick_virt_kbd.service $(SERVICE_INSTALL_PATH)
11      sudo mkdir -p $(PRIVATE_DATA_PATH)
12      sudo -E sh -c 'env > $(PRIVATE_DATA_PATH)/.env'
13      sudo systemctl daemon-reload
14      sudo systemctl start joystick_virt_kbd
15
16  uninstall:
17      sudo systemctl stop joystick_virt_kbd
18      sudo rm -rf $(PRIVATE_DATA_PATH) $(APP_INSTALL_PATH) \
19          $(SERVICE_INSTALL_PATH)/joystick_virt_kbd.service
20      sudo systemctl daemon-reload
21
22  # остальные правила для сборки программы ...
```

3.8 Сборка и запуск

Для сборки загружаемого модуля ядра достаточно выполнить команду `make` в папке с кодом модуля, после чего загрузка модуля в ядро осуществляется командой `insmod joystick_kbd.ko`.

Сборка и запуск демона выполняется аналогичным образом. Выполнение команды `make` без параметров приведет к сборке исполняемого файла. Чтобы установить его в систему, необходимо выполнить команду `make install`. При этом сервис будет сразу же запущен. Для остановки и удаления программы из системы необходимо использовать команду `make uninstall`.

4 Исследовательский раздел

4.1 Исследуемые параметры

Исследуется зависимость времени обработки URB пакета от того, запущен демон или нет.

Замер времени будет происходить для функции драйвера `jskbd_complete` с использованием функции ядра `ktime_get` [10], для двух сценариев:

- при запущенном демоне;
- при остановленном демоне.

При сравнении будут учитываться только первые 1000 обработанных URB пакетов с момента запуска драйвера и подключения геймпада. В результате анализа полученных данных будут рассчитаны и сопоставлены средние значения измерений.

4.2 Результаты исследования

На рисунке 10 показана гистограмма полученных измерений.

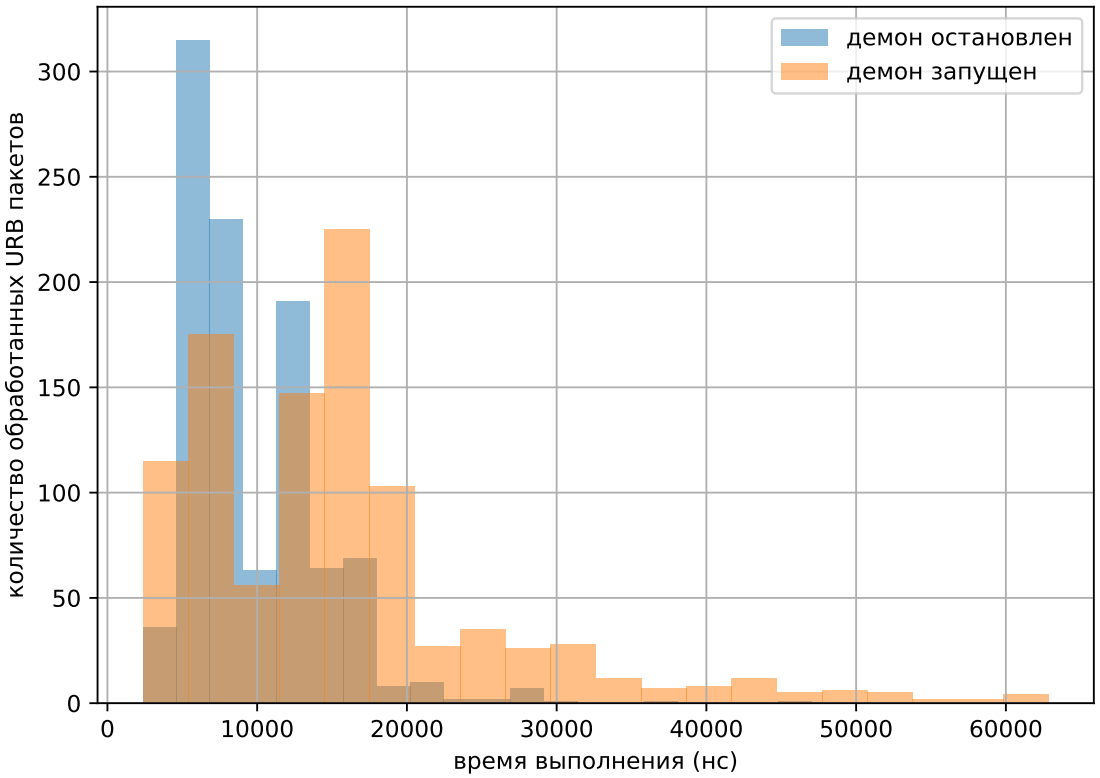

Рисунок 10 – Результаты замеров времени для двух сценариев

По результатам исследования установлены следующие средние значения обработки URB пакетов:

- при запущенном демоне: 15649 нс;
- при остановленном демоне: 9679 нс.

Данный результат можно объяснить тем, что запущенный демон требует больше времени на поддержание очереди ждущих процессов. Данное время в среднем можно считать равным 5970 нс.

ЗАКЛЮЧЕНИЕ

В ходе работы была изучена подсистема USB и подсистема ввода для интерактивных устройств.

Рассмотрены и решены проблемы связанные с использованием геймпада в качестве мыши и клавиатуры.

Был разработан драйвер для геймпада, а также написана программа, запускаемая в режиме демона с использованием утилиты `systemd`.

Работоспособность и корректность выполнения были протестированы на реальном устройстве Logitech F310 для двух версий ядра 5.15 и 6.1.

Также было проведено исследование зависимости времени обработки URB от работы демона.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Linux Gamepad Specification. [Электронный ресурс]. URL: https://docs.kernel.org/input/gamepad.html (Дата обращения: 16.11.2022)

2. Linux USB Basics. [Электронный ресурс]. URL: https://www.kernel.org/doc/htmldocs/writing_usb_driver/basics.html (Дата обращения: 21.11.2022)

3. Human Interface Devices (HID) Specifications and Tools. [Электронный ресурс]. URL: https://www.usb.org/hid (Дата обращения: 16.11.2022)

4. Jonathan Corbet. Linux Device Drivers, 3rd Edition. [Электронный ресурс]. O'REILLY. URL: https://www.oreilly.com/library/view/linux-device-drivers/0596005903 (Дата обращения: 16.11.2022)

5. Linux source code (v6.1) – Bootlin. [Электронный ресурс]. URL: https://elixir.bootlin.com/linux/v6.1/source (Дата обращения: 20.11.2022)

6. Devres – Managed Device Resource – The Linux Kernel documentation. [Электронный ресурс]. URL: https://docs.kernel.org/driver-api/driver-model/devres.html (Дата обращения: 19.11.2022)

7. The /proc Filesystem – The Linux Kernel documentation. [Электронный ресурс]. URL: https://docs.kernel.org/filesystems/proc.html (Дата обращения: 21.11.2022)

8. System and Service Manager. [Электронный ресурс]. URL: https://systemd.io (Дата обращения: 26.11.2022)

9. Rust – The Linux Kernel documentation. [Электронный ресурс]. URL: https://www.kernel.org/doc/html/next/rust/index.html (Дата обращения: 27.11.2022)

10. ktime accessors – The Linux Kernel documentation. [Электронный ресурс]. URL: https://docs.kernel.org/core-api/timekeeping.html (Дата обращения: 29.11.2022)

ПРИЛОЖЕНИЕ А

Листинг 10: Функция пересчета позиции указателя на виртуальной клавиатуре

```
static unsigned char keycode_map[] = {
    KEY_ESC, KEY_F1, KEY_F2, KEY_F3, KEY_F4, KEY_F5, KEY_F6, KEY_F7, KEY_F8,
        KEY_F9, KEY_F10, KEY_F11, KEY_F12, KEY_DELETE,
    KEY_GRAVE, KEY_1, KEY_2, KEY_3, KEY_4, KEY_5, KEY_6, KEY_7, KEY_8,
        KEY_9, KEY_0, KEY_MINUS, KEY_EQUAL, KEY_BACKSPACE,
    KEY_TAB, KEY_Q, KEY_W, KEY_E, KEY_R, KEY_T, KEY_Y, KEY_U, KEY_I, KEY_O,
        KEY_P, KEY_LEFTBRACE, KEY_RIGHTBRACE, KEY_BACKSLASH,
    KEY_CAPSLOCK, KEY_A, KEY_S, KEY_D, KEY_F, KEY_G, KEY_H, KEY_J, KEY_K,
        KEY_L, KEY_SEMICOLON, KEY_APOSTROPHE, KEY_ENTER,
    KEY_LEFTSHIFT, KEY_Z, KEY_X, KEY_C, KEY_V, KEY_B, KEY_N, KEY_M,
        KEY_COMMA, KEY_DOT, KEY_SLASH, KEY_RIGHTSHIFT,
    KEY_LEFTCTRL, 0, KEY_META, KEY_LEFTALT, KEY_SPACE, KEY_RIGHTALT,
        KEY_RIGHTCTRL, KEY_LEFT, KEY_UP, KEY_DOWN, KEY_RIGHT,
};

static int keycode_row_len[] = { 14, 14, 14, 13, 12, 11 };
static int keycode_row_len_acc[] = { 14, 28, 42, 55, 67, 78 };

#define KEYCODE_ROWS (sizeof(keycode_row_len) / sizeof(keycode_row_len[0]))
#define KEYCODE_COLS(row) (keycode_row_len[row])

unsigned char move_keyboard_cursor(unsigned int *row, unsigned int *col,
    int d_row, int d_col)
{
    unsigned int new_row;
    unsigned int new_col;
    unsigned int index;

    if (d_row == 0 && d_col == 0)
    {
        index = *row == 0 ? *col : keycode_row_len_acc[*row - 1] + *col;
        return keycode_map[index];
    }

    new_row = ((int)*row + d_row + KEYCODE_ROWS) % KEYCODE_ROWS;
```

```
36          *row = new_row;
37
38          new_col = ((int)*col + d_col + KEYCODE_COLS(*row)) % KEYCODE_COLS(*row);
39          *col = new_col;
40
41          index = *row == 0 ? *col : keycode_row_len_acc[*row - 1] + *col;
42          if (index > ARRAY_SIZE(keycode_map))
43          {
44              printk(MOD_PREFIX "invalid keycode index: %u\n", index);
45              printk(MOD_PREFIX "given row=%u col=%u\n", *row, *col);
46              return keycode_map[0];
47          }
48
49          return keycode_map[index];
50      }
```

ПРИЛОЖЕНИЕ Б

Листинг 11: Файл с функциями `module_init` и `module_exit`

```
#include <linux/module.h>
#include "config.h"
#include "driver.h"

static int md_init(void)
{
    int status = register_driver();
    if (status != 0)
        printk(KERN_ERR MOD_PREFIX "failed to register usb driver\n");
    else
        printk(KERN_INFO MOD_PREFIX "loaded\n");

    return status;
}

static void md_exit(void)
{
    deregister_driver();
    printk(KERN_INFO MOD_PREFIX "unloaded\n");
}

module_init(md_init);
module_exit(md_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Klimenko Alexey");
MODULE_DESCRIPTION("Курсовая работа по операционным системам");
```

ПРИЛОЖЕНИЕ В

Листинг 12: `Makefile` для сборки драйвера

```
.PHONY: clean

obj-m += joystick_kbd.o
joystick_kbd-objs := src/entry.o src/proc_event.o src/input_dev.o
    src/driver.o

all:
    $(MAKE) -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    $(MAKE) -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

ПРИЛОЖЕНИЕ Г

Листинг 13: `Makefile` для сборки демона

```
CPP := g++
MOC := moc
EXECUTABLE := daemon
CFLAGS := -std=c++23 -Wall -pedantic -Wno-deprecated-enum-enum-conversion
INCDIRS := $(shell pkg-config --cflags Qt5UiTools Qt6Widgets)
LIBS := $(shell pkg-config --libs Qt5UiTools Qt6Widgets)
MOC_HEADERS := src/main_window.h src/worker.h
SOURCES := src/main.cpp src/main_window.cpp src/worker.cpp

# Change postfixes
MOC_SOURCES := $(MOC_HEADERS:.h=.moc.cpp)
OBJECTS := $(SOURCES:.cpp=.o) $(MOC_SOURCES:.cpp=.o)

# 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

.PHONY: clean install uninstall

all: $(EXECUTABLE)

install: $(EXECUTABLE)
    sudo cp $(EXECUTABLE) $(APP_INSTALL_PATH)
    sudo cp systemd/joystick_virt_kbd.service
        $(SERVICE_INSTALL_PATH)/joystick_virt_kbd.service
    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 $(SERVICE_INSTALL_PATH)/joystick_virt_kbd.service
    sudo rm -rf $(PRIVATE_DATA_PATH) $(APP_INSTALL_PATH)
```

```
36          sudo systemctl daemon-reload
37
38      $(EXECUTABLE): $(OBJECTS)
39          $(CPP) $^ -o $@ $(LIBS)
40
41      %.o: %.cpp
42          $(CPP) $(CFLAGS) $(INCDIRS) -c $< -o $@
43
44
45      %.moc.cpp: %.h
46          $(MOC) $(INCDIRS) $< -o $@
47
48      clean:
49          rm -rf $(EXECUTABLE) src/*.o
```