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 и системы управления_____

КАФЕДРА ____Программное обеспечение ЭВМ и информационные технологии» (ИУ7)_____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

Драйвер нулевого уровня для использования графического планшета в качестве клавиатуры

Студент __ИУ7-73Б_____ _____ ____**Степанов А. О.**___
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ _____**Тассов К. Л.**____
(Подпись, дата) (И.О.Фамилия)

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

2020 г.

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)

УТВЕРЖДАЮ
Заведующий кафедрой ___ИУ7___
(Индекс)
_____ __И.В.Рудаков___
(И.О.Фамилия)
« _____ » _____ 20_ __ г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине _Операционные системы_____

Студент группы __ИУ7-73Б____

_____Степанов Александр Олегович_____
(Фамилия, имя, отчество)

Тема курсового проекта _Драйвер нулевого уровня для использования графического планшета в качестве клавиатуры._____
_____
_____

Направленность КП (учебный, исследовательский, практический, производственный, др.)
_учебный_____
Источник тематики (кафедра, предприятие, НИР) ___кафедра_____

График выполнения проекта: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

Задание __Реализовать загружаемый модуль ядра для использования графического планшета в качестве клавиатуры._____
_____

Оформление курсового проекта:

Расчетно-пояснительная записка на _20-25__ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
К защите должны быть подготовлены презентация и доклад, отражающие суть выполненной работы, содержание и методы решения основных задач, а также полученные результаты.
_____
_____
_____
–

Дата выдачи задания « ____ » _____ 20__ г.

Руководитель курсового проекта _____ _____ ___К.Л. Тассов_____
(Подпись, дата) (И.О.Фамилия)

Студент _____ _____ ___А.О. Степанов___
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

РЕФЕРАТ

Отчет содержит 35 стр., 7 рис., 12 источн..

Ключевые слова: linux, драйвер, графический планшет, загружаемый модуль ядра, клавиатура, прерывания, пространство ядра.

Курсовой проект представляет собой загружаемый модуль ядра для использования графического планшета в качестве клавиатуры. Используется язык программирования C.

СОДЕРЖАНИЕ

Реферат 3

Введение 6

1 Аналитический раздел 7

1.1 Графический планшет 7

1.2 Загружаемый модуль ядра 7

1.2.1 Драйвер устройства 8

1.3 Подключение графического планшета 8

1.3.1 Прерывания 8

1.3.2 Драйвер usb устройства 9

1.3.2.1 Структура usb_driver 9

1.3.2.2 Работа драйвера usb устройства 9

1.3.2.3 Подключение графического планшета 9

1.3.3 Многозадачность для прерываний 10

1.3.3.1 Тасклеты 10

1.3.3.2 Очереди работ 11

1.4 Работа клавиатуры 12

1.4.1 Подсистема ввода/вывода 13

1.4.1.1 Установка событий 13

1.4.1.2 Вызов событий 13

1.5 Вывод 14

2 Конструкторский раздел 15

2.1 Состав программного обеспечения 15

2.1.1 Загрузка модуля ядра 15

2.2 Создание usb драйвера 16

2.3 Структура для хранения ифнормации о графическом планшете ... 16

2.4 Структура для передачи данных о прерывании 17
2.5 Обработка прерываний 17
2.6 Разделение поверхности планшета на клавиши 18
2.7 Вывод 19
3 Технологический раздел 20
3.1 Выбор языка программирования 20
3.2 Работа программы 20
3.2.1 Начальная настройка 20
3.2.2 Драйвер для планшета 21
3.2.3 Нажатие клавиш клавиатуры 21
3.3 Пример работы программы 21
3.4 Вывод 23
Заключение 24
Список использованных источников 25
Реализация 27

ВВЕДЕНИЕ

В настоящее время невозможно представить работу за компьютером без полноценной клавиатуры, но не всегда имеется возможность взять с собой такое большое устройство. Одним из самых мобильных и небольших устройств, которое похоже на клавиатуру, является графический планшет. Графический планшет имеет плоскую форму, которую удобно взять с собой, в отличии от клавиатуры. Поэтому существует потребность создания драйвера для использования графического планшета в качестве клавиатуры.

Целью данной работы является разработка загружаемого модуля ядра для эмитации работы клавиатуры на графическом планшете.

Для достижения этой цели ставятся следующие задачи:

а) изучение подходов для реализации драйвера устройства linux;

б) изучение подходов для эмитации работы клавиатуры;

в) реализация требуемого драйвера нулевого уровня.

1 Аналитический раздел

Требуется разработать драйвер для графического планшета, позволяющий работать устройству в качестве клавиатуры. В данном разделе рассматриваются методы решения поставленной задачи.

1.1 Графический планшет

Графический планшет – это устройство для ввода информации, созданной от руки, непосредственно в компьютер. Состоит из пера (стилуса) и плоского планшета, чувствительного к нажатию или близости пера.

Для разработки и тестирования данной работы используется планшет Wacom CTL-671, его изоражение представлено на рисунке 1.1.

Рисунок 1.1 — Графический планшет Wacom CTL-671

1.2 Загружаемый модуль ядра

Загружаемый модуль ядра – объектный файл, содержащий код, расширяющий возможности ядра операционной системы. Модули используются, чтобы добавить поддержку нового оборудования или файловых систем или для добавления новых системных вызовов. Когда функциональность, предоставляемая модулем, больше не требуется, он может быть выгружен, чтобы освободить память и другие ресурсы.

1.2.1 Драйвер устройства

Драйверы устройств являются одной из разновидностей модулей ядра. Они играют особую роль. Драйверы полностью скрывают детали, касающиеся рабоыт устройства и предоставляют четкий программный интерфейс для работы с аппаратурой. В Unix каждое аппаратное устройство представлено псевдофайлом (файлом устройства) в каталоге /dev. Этот файл обеспечивает средства взаимодействия с аппаратурой.

1.3 Подключение графического планшета

Первой задачей стоит подключение графического планшета для обработки прерываний при нажатии на устройство.

1.3.1 Прерывания

Прерывание – сигнал к процессору, испускаемый аппаратными средствами или программным обеспечением, и указывающий на событие, которое требует немедленного внимания. Прерывание предупреждает процессор о высокоприоритетном состоянии, требующем прерывания текущего кода, выполняемого процессором. Процессор отвечает, приостанавливая свои текущие действия, сохраняя свое состояние и выполняя функцию, называемую обработчиком прерываний (или подпрограммой обработки прерываний, ISR) для обработки события. Это прерывание является временным, и после завершения обработки обработчика прерывания процессор возобновляет обычную работу. Существует два типа прерываний: аппаратные прерывания и программные прерывания [1].

Каждое прерывание имеет свой собственный обработчик прерываний. Количество аппаратных прерываний ограничено числом строк запроса прерывания (IRQ) для процессора, но могут быть сотни различных программных прерываний. Прерывания — это широко используемая техника многозадачности компьютеров, в первую очередь в реальном времени. Такая система называется управляемой прерываниями.

1.3.2 Драйвер usb устройства

Для того, чтобы перехватывать прерывания графического планшета необходимо создать драйвер usb устройства и подключить планшет к нему.

1.3.2.1 Структура usb_driver

Для создания драйвера usb устройства необходимо использовать структуру usb_driver [2]. 4 главных поля, которые необходимо использовать это: name (имя загружаемого драйвера), probe (указатель на функцию, вызывающуюся при подключении устройства), disconnect (указатель на функцию, вызывающуюся при отключении устройства) и id_table (список устройств, которые надо автоматически подключать к драйверу, для идентификации устройства используются id поставщика устройства и id самого устройства).

1.3.2.2 Работа драйвера usb устройства

После создания экземляра структуры, представляющей из себя usb драйвер, его необходимо зарегестрировать в системе с помощью системного вызова usb_register. Если usb драйвер будет успешно зарегестрирован, то он попытается подключить все подходящие устройства, подключенные к системе и незанятые никаким драйвером. Выбор устройств для попытки подключения делается с помощью поля id_table. Если функция подключения вернет код успешного завершения, то устройство будет подключено к драйверу [2].

1.3.2.3 Подключение графического планшета

Для подключения планшета в функции probe необходимо проделать следующую последовательность действий:

— выделить память для экземпляра структуры планшета;

— выделить память для устройства ввода;

— выделить память для URB (USB Request Block);

— получить свободный путь в файловой системе для устройства ввода;

— связать прерывание устройства с функцией;

— зарегистрировать устройство.

1.3.3 Многозадачность для прерываний

Чтобы сократить время выполнения обработчиков прерываний обработчики медленных аппаратных прерываний делятся на две части, которые традиционно называются верхняя (top) и нижняя (bottom) половины (half). Верхними половинами остаются обработчики, устанавливаемы функцией request_irq() на определенных IRQ. Выполнение нижних половин инициируется верхними половинами, т.е. обработчиками прерываний [3].

В ОС Linux имеется три типа нижних половин (bottom half) [3]:

— softirq – отложенные прерывания;

— tascklet – тасклеты;

— workqueue – очереди работ.

Для обработки прерываний испольуют тасклеты и очереди работ. Рассмотрим данные методы.

1.3.3.1 Тасклеты

Тасклеты – это механизм обработки нижних половин, построенный на основе механизма отложенных прерываний [3].

Тасклеты описываются следующей структурой, описанной на листинге 1.1 [4].

Листинг 1.1 — Структура tasklet

```
1 struct tasklet_struct
2 {
3     struct tasklet_struct *next;
4     unsigned long state;
5     atomic_t count;
6     void (*func)(unsigned long);
```

```
7       unsigned long data;
8   };
```

В данной структуре имеются поля [4]:

— next – указатель на следюущий тасклет в списке;

— state – состояние тасклета;

— func – функция обработчик тасклета;

— data – аргумент функции обработчика тасклета.

Когда tasklet запланирован, он добавляется в очередь. Пока он находится в этом состоянии, запланировать его еще раз не получится – в этом случае просто ничего не произойдет. Tasklet не может находиться сразу в нескольких местах в очереди на планирование, которая организуется через поле next структуры tasklet_struct. После того, как тасклет был запланирован, он выполнится один раз [4].

1.3.3.2 Очереди работ

Очередь работ является еще одной концепцией для обработки отложенных функций. Функции рабочих очередей выполняются в контексте процесса ядра. Это означает, что функции очереди задач не должны быть атомарными, как функции тасклета. Подсистема рабочей очереди представляет собой интерфейс для создания потоков ядра для обработки работы (work), которая ставится в очередь. Такие потоки ядра называются рабочими потоками. Рабочая очередь поддерживается типом struct work_struct, описанным на листинге 1.2 [5].

Листинг 1.2 — Структура work_struct

```
1  struct work_struct
2  {
3       atomic_long_t data;
4       struct list_head entry;
5       work_func_t func;
6  # ifdef CONFIG_LOCKDEP
7       struct lockdep_map lockdep_map;
8  # endif
```

```
9 };
```

Очередь работ создается функцией:

```
1 int create_workqueue(char *name, unsigned int flags, int max_active);
```

— name – имя очереди, но в отличие от старых реализаций потоков с этим именем не создается;

— flags – флаги определяют как очередь работ будет выполняться;

— max_active – ограничивает число задач из данной очереди, которые могут одновременно выполняться на одном CPU.

Для того, чтобы поместить задачу в очередь работ надо заполнить (инициализировать) структуру work_struct.

После того, как работа была создана, следующим шагом будет помещение этой структуры в очередь работ. Это можно сделать несколькими способами. Во-первых, просто добавить работу (объект work) в очередь работ с помощью функции queue_work (которая назначает работу текущему процессору). Можно с помощью функции queue_work_on указать процессор, на котором будет выполняться обработчик. Две дополнительные функции обеспечивают те же функции для отложенной работы (в которой инкапсулирована структура work_struct и таймер, определяющий задержку): queue_delayed_work, queue_delayed_work_on [5].

Кроме того, можно использовать глобальное ядро – глобальную очередь работ с четырьмя функциями, которые работают с этой очередью работ. Эти функции имитируют предыдущие функции, за исключением лишь того, что не нужно определять структуру очереди работ [5].

1.4 Работа клавиатуры

Для эмуляции работы клавиатуры необходимо вызывать нажатия клавиш после обработки прерываний графического планшета.

1.4.1 Подсистема ввода/вывода

Подсистема ввода/вывода выполняет запросы файловой подсистемы и подсистемы управления процессами для доступа к периферийным устройствам (дискам, магнитным лентам, терминалам и т.д.). Она обеспечивает необходимую буферизацию данных и взаимодействует с драйверами устройств — специальными модулями ядра, непосредственно обслуживающими внешние устройства [6].

Подсистемой ввода/вывода поддерживаются три вида устройств:

— символьные устройства для поддержки последовательных устройств;

— блочные устройства для поддержки устройств с произвольным доступом, блочные устройства имеют важное значение для реализации файловых систем;

— сетевые устройства, которые поддерживают широкий спектр устройств на канальном уровне.

Для использования подсистемы ввода/вывода используется структура input_dev [7]. Чтобы инициализировать эту структуру используется функция set_bit, которая принимает два аргумента: бит, который устанавливается, и адрес, куда этот бит устанавливать [8].

1.4.1.1 Установка событий

Для установки типа события, которое будет вызываться необходимо установить бит EV_KEY [9] в поле evbit в структуре input_dev [7]. После установки бита события устанавливаются биты клавиш, которые будут вызываться, например, KEY_0 (клавиша с цифрой 0), KEY_Z (клавиша с буквой z) и KEY_CAPSLOCK (клавиша CapsLock).

1.4.1.2 Вызов событий

Для вызова событий, связанных с клавишами используется системный вызов input_report_key [10], который принимает устройство ввода (структура

input_dev), клавишу, на которую вызывается событие, и код события. В данной работе необходимы два кода событий: 1 – кнопка зажата, 0 – кнопка отжата.

1.5 Вывод

Таким образом были рассмотрены методы решения задачи разработки драйвера нулевого уровня для использования графического планшета в качестве клавиатуры.

2 Конструкторский раздел

В данном разделе рассматривается структура программного обеспечения.

2.1 Состав программного обеспечения

Программное обеспечение состоит из загружаемого модуля ядра.

2.1.1 Загрузка модуля ядра

Для компиляции модуля используется специальный makefile. На листинге 2.1 представлен makefile, с помощью которого компилировался модуль ядра из данной работы.

Листинг 2.1 — Makefile для компиляции модуля ядра

```
1  ifneq ($(KERNELRELEASE),)
2      obj-m := keyboard_tablet.o
3  else
4      CURRENT = $(shell uname -r)
5      KDIR = /lib/modules/$(CURRENT)/build
6      PWD = $(shell pwd)
7
8  all:
9      $(MAKE) -C $(KDIR) M=$(PWD) modules
10
11 clean:
12     @rm -f *.o .*.cmd .*.flags *.mod.c *.order
13     @rm -f .*.*.cmd *~ *.*~ TODO.*
14     @rm -fR .tmp*
15     @rm -rf .tmp_versions
16
17 disclean: clean
18     @rm *.ko *.symvers
19
20 endif
```

После компиляции получается объектный файл с расширением .ko, который можно загрузить в ядро с помощью команды insmod под sudo.

2.2 Создание usb драйвера

Для создания usb драйвера создается экземпляр структуры usb_driver [2]. Создание экземпляра приведено на листинге 2.2.

Листинг 2.2 — Создание экземпляра usb драйвера

```
1 static struct usb_driver tablet_driver = {
2     .name       = DRIVER_NAME,
3     .probe      = tablet_probe,
4     .disconnect = tablet_disconnect,
5     .id_table   = tablet_table,
6 };
```

Для ругистрации usb драйвера используется системный вызов usb_register [11].

2.3 Структура для хранения ифнормации о графическом планшете

Для передачи данных, связанных с графическим планшетом была создана структура tablet, приведенная на листинге 2.3.

Листинг 2.3 — Структура планшета

```
1 struct tablet {
2     unsigned char     *data;
3     struct input_dev  *input_dev;
4     struct usb_device *usb_dev;
5     struct urb        *irq;
6 };
7 typedef struct tablet tablet_t;
```

В данной структуре созданы следующие поля:

— data – данные, передаваемые планшетом при прерывании;

— input_dev – подключенный планшет;

— usb_dev – представление usb устройства;

— irq – обработчик прерываний.

2.4 Структура для передачи данных о прерывании

Для того, чтобы передавать в работу из очереди данные о прерывании была создана структура container_urb, представленная на листинге 2.4.

Листинг 2.4 — Структура для передачи данных о прерывании

```
1 struct container_urb {
2     struct urb *urb;
3     struct work_struct work;
4 };
5
6 typedef struct container_urb container_urb_t;
```

В данной структуре созданы следующие поля:

— urb – обработанное прерывание;

— work – текущая работа в очереди.

2.5 Обработка прерываний

На рисунке 2.1 представлена схема обработки прерываний графического планшета и отправка событий нажатия на клавиши клавиатуры.

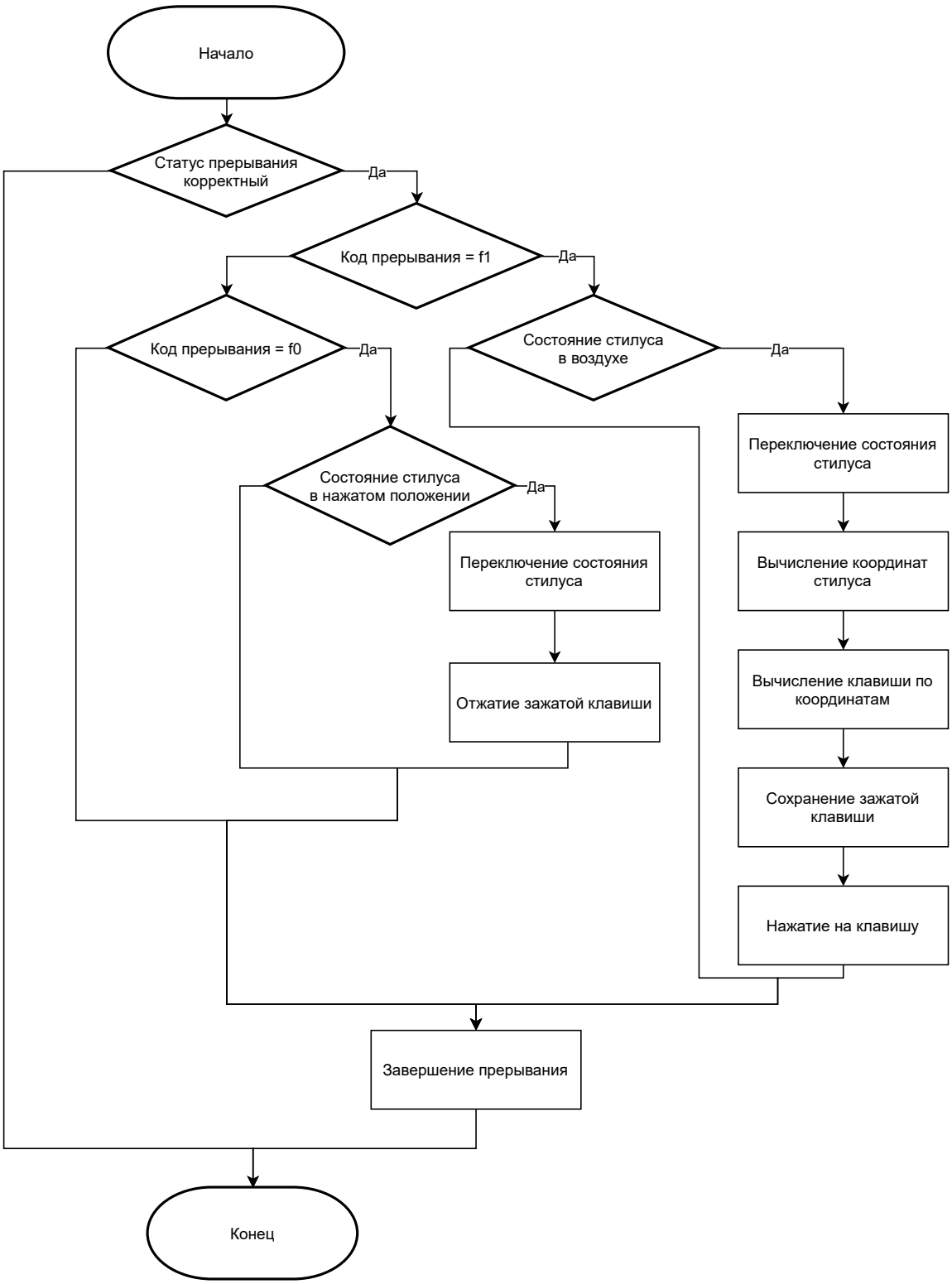

Рисунок 2.1 — Обработка прерываний

2.6 Разделение поверхности планшета на клавиши

Для комфортного использования модуля клавиши были расставлены на основе классической qwerty раскладки [12].

Координаты стилуса находятся в диапазоне от 0 до 1344 по оси x и от 0 до 468 по y. Поскольку координаты стилуса принимают только значения кратные 16 для оси x и кратные 9 для y, то поверхость была поделена на области размером 16 на 9. Таких областей получилось 84 на 52. Именно координаты этих областей и используются для вычисления клавиши. На рисунке 2.2 представлена схема разделения поверхности планшета на клавиши клавиатуры.

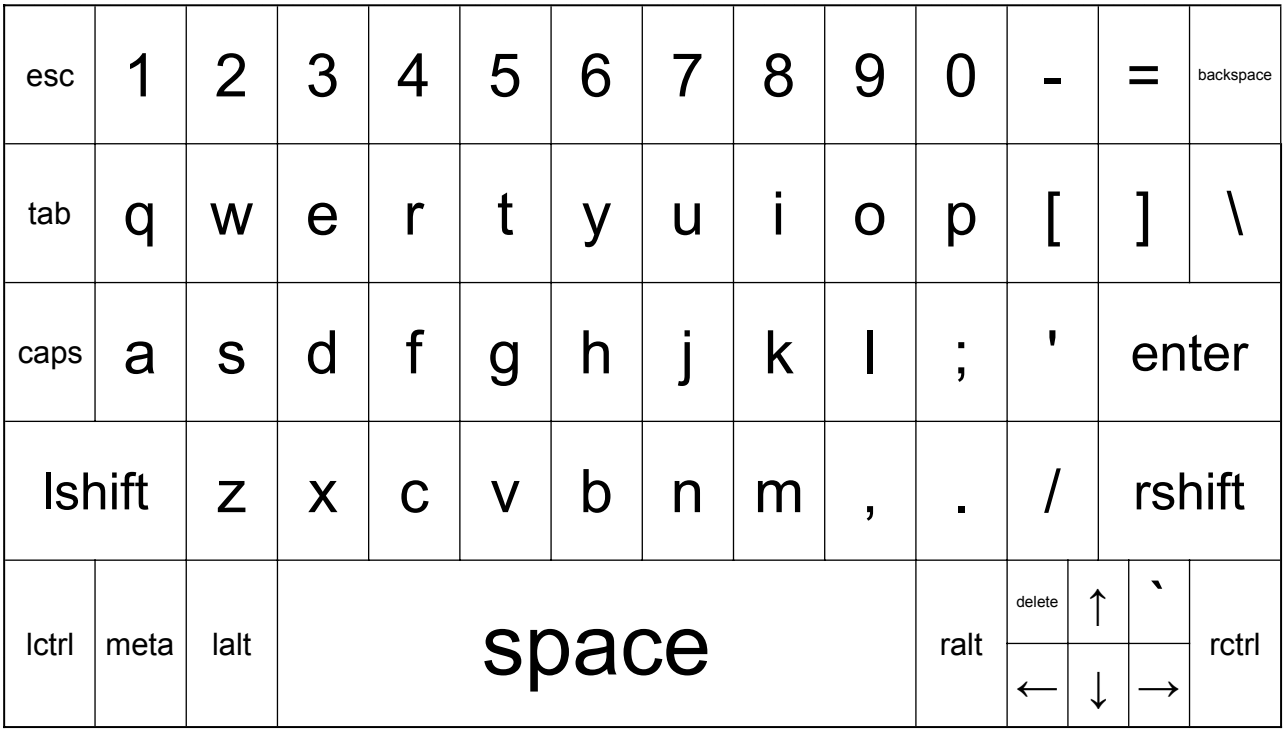

Рисунок 2.2 — Разделение поверхности планшета на клавиши

2.7 Вывод

В данном разделе был рассмотрен процесс проектирования структуры программного обеспечения.

3 Технологический раздел

В данном разделе производится выбор средств для разработки и рассматривается реализация программного обеспечения.

3.1 Выбор языка программирования

В качестве языка программирования был выбран язык C. На этом языке реализованы все модули ядра и драйверы операционной системы Linux. Компилятор – gcc.

3.2 Работа программы

Рассмотрим работу модуля с листингами.

3.2.1 Начальная настройка

На листинге 3.1 представлено объявление всех необходимых макросов. На листинге 3.2 представлено объявление всех глобальных пременных, а именно:

— pen_enter – положение стилуса (на планшете или в воздухе);

— pressed_key – нажатая в текущий момент клавиша;

— workq – очередь работ;

— keyboard – виртуальное устрйоство для вывода событий клавиш.

Также там объявляются структуры tablet, которая нужна для хранения данных о состоянии планшета, и container_urb, которая нужна для передачи данных о текущем прерывании в работу.

На листинге 3.3 представлена функция инициализации модуля, где происходит регистрация драйвера, инициализацяи очереди работ, настройка виртуального устройства клавиатуры и ее регистрация.

На листинге 3.4 представлена функция выгрузки модуля, где происходит очистка памяти, выключение драйвера и удаление вртуального устройства.

3.2.2 Драйвер для планшета

На листинге 3.6 представлена функция подключения планшета, в которой производится все необходимое выделение памяти и настрйока.

На листинге 3.7 представлены функции открытия и закрытия устройства ввода.

На листинге 3.8 представлена функция отключения планшета, в которой освобождается память и удаляются обработчики.

На листинге 3.9 представлена таблица устройств, которые необходимо подключать. Первый элемент этой таблицы это планшет Wacom Ltd CTL-671, который использовался для тестирования драйвера, а второй – пустой элемент, это означает, что если нет устройства, у которого совпадают идентификаторы из других элементов таблицы, то драйвер будет пытаться подключить каждое свободное устройство.

На листинге 3.10 представлена функция, которая перехватывает прерывание. В ней создается работа и отправляется в очередь обработки.

На листинге 3.11 представлена функция, обрабатывающая прерывание. Ее алгоритм представлен на рисунке 2.1.

3.2.3 Нажатие клавиш клавиатуры

На листинге 3.12 представлены функции нажатия и отжатия текущей клавиши.

На листинге 3.13 представлена часть функции выбора клавиши в зависимости от координат. Разделение поверхности планшета на клавиши представлено на рисунке 2.2.

3.3 Пример работы программы

На рисунке 3.1 изображены логи при подключении планшета в систему. В этот момент планшет также подключается и к драйверу.

```
[  +0.055038] keyboard_tablet: loading out-of-tree module taints kernel.
[  +0.000036] keyboard_tablet: module verification failed: signature and/or required key missing - tainting kernel
[  +0.000262] usbcore: registered new interface driver keyboard_tablet
[  +0.001184] input: virtual keyboard as /devices/virtual/input/input6
[  +0.000078] keyboard_tablet: module loaded
[ +15.253828] usb 2-2.1: new full-speed USB device number 4 using uhci_hcd
[  +0.479183] usb 2-2.1: New USB device found, idVendor=056a, idProduct=0301, bcdDevice= 1.00
[  +0.000003] usb 2-2.1: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[  +0.000001] usb 2-2.1: Product: CTL-671
[  +0.000000] usb 2-2.1: Manufacturer: Wacom Co.,Ltd.
[  +0.004877] keyboard_tablet: probe checking tablet
[  +0.000047] input: keyboard_tablet as /devices/pci0000:00/0000:00:11.0/0000:02:00.0/usb2/2-2/2-2.1/2-2.1:1.0/input/input7
[  +0.000034] keyboard_tablet: device is conected
[  +0.000026] keyboard_tablet: probe checking tablet
[  +0.000019] input: keyboard_tablet as /devices/pci0000:00/0000:00:11.0/0000:02:00.0/usb2/2-2/2-2.1/2-2.1:1.1/input/input8
[  +0.000018] keyboard_tablet: device is conected
```

Рисунок 3.1 — Подключение планшета к операционной системе

На рисунке 3.2 изображены логи нажатий на планшет в разных областях.

```
[  +0.763070] keyboard_tablet: work_irq - urb status is -75
[  +0.278020] keyboard_tablet: pressed f
[  +0.000024] keyboard_tablet: pen enters 448 234
[  +0.089983] keyboard_tablet: pen leaves
[  +0.912105] keyboard_tablet: pressed o
[  +0.000028] keyboard_tablet: pen enters 912 135
[  +0.067980] keyboard_tablet: pen leaves
[  +0.261979] keyboard_tablet: pressed p
[  +0.000022] keyboard_tablet: pen enters 1008 126
[  +0.052023] keyboard_tablet: pen leaves
[  +0.618039] keyboard_tablet: pressed backspace
[  +0.000023] keyboard_tablet: pen enters 1280 45
[  +0.068031] keyboard_tablet: pen leaves
[  +0.179966] keyboard_tablet: pressed backspace
[  +0.000024] keyboard_tablet: pen enters 1296 36
[  +0.060014] keyboard_tablet: pen leaves
[  +0.171983] keyboard_tablet: pressed backspace
[  +0.000023] keyboard_tablet: pen enters 1296 36
[  +0.054227] keyboard_tablet: pen leaves
[  +0.975844] keyboard_tablet: pressed w
[  +0.000027] keyboard_tablet: pen enters 240 135
[  +0.059993] keyboard_tablet: pen leaves
[  +0.210005] keyboard_tablet: pressed q
[  +0.000015] keyboard_tablet: pen enters 160 126
[  +0.066033] keyboard_tablet: pen leaves
[  +0.646037] keyboard_tablet: pressed tab
[  +0.000016] keyboard_tablet: pen enters 16 144
[  +0.059988] keyboard_tablet: pen leaves
[  +1.188096] keyboard_tablet: pressed 2
[  +0.000016] keyboard_tablet: pen enters 240 36
[  +0.044009] keyboard_tablet: pen leaves
[  +0.187999] keyboard_tablet: pressed 2
[  +0.000016] keyboard_tablet: pen enters 240 36
[  +0.060004] keyboard_tablet: pen leaves
[  +1.874169] keyboard_tablet: pressed enter
[  +0.000016] keyboard_tablet: pen enters 1248 234
[  +0.046004] keyboard_tablet: pen leaves
```

Рисунок 3.2 — Логи нажатий на планшет

На рисунке 3.3 изображены логи нажатий на планшет, когда открыта консоль с выводом логов. Здесь можно заметить перед сообщением о печати буквы сами эти буквы, которые были напечатаны с помощью планшета.

Рисунок 3.3 — Логи нажатий на планшет в консоли с логами

На рисунке 3.4 изображены логи отключения планшета.

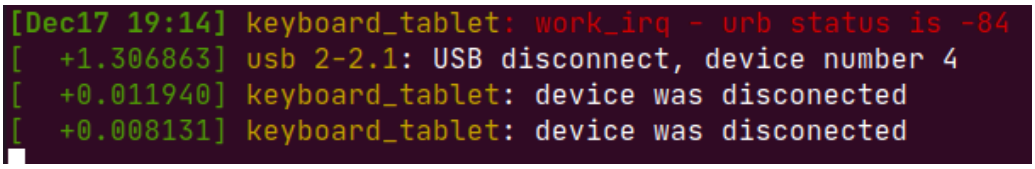

Рисунок 3.4 — Отключение планшета от оперционной системы

3.4 Вывод

В данном разделе был выбран язык программирования C, а также рассмотрена реализация программного обеспечения.

ЗАКЛЮЧЕНИЕ

В ходе данной работы были выполнены следующие задачи:

а) изучены подходы для реализации драйвера устройства linux;

б) изучены подходы для эмитации работы клавиатуры;

в) реализован требуемый драйвер нулевого уровня.

Таким образом, достигнута цель реализации драйвера нулевого уровня для использования графического планшета в качестве клавиатуры на языке программирования C.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Jonathan Corbet Alessandro Rubini Greg Kroah-Hartman. Linux Device Drivers. — 3 edition. — O'Reilly Media, 2005.

2. Writing USB Device Drivers. — Access mode: `https://kernel.readthedocs.io/en/sphinx-samples/writing_usb_driver.html` (online; accessed: 18.12.2020).

3. Rothberg Valentin. Interrupt Handling in Linux. — Dept. of Computer Science, University of Erlangen, Germany, 2015.

4. tasklet_struct. — Access mode: `https://elixir.bootlin.com/linux/latest/source/include/linux/interrupt.h#L609` (online; accessed: 18.12.2020).

5. workqueue.h. — Access mode: `https://elixir.bootlin.com/linux/latest/source/include/linux/workqueue.h#L102` (online; accessed: 18.12.2020).

6. Liangfeng Fu Linbo Xie Zhigang Zhou. The design of touch screen driver based on Linux input subsystem and S3C6410 platform. — International Conference on Information Science and Technology Application (ICISTA-13), 2013.

7. input_dev. — Access mode: `https://elixir.bootlin.com/linux/v5.10.1/source/include/linux/input.h#L131` (online; accessed: 18.12.2020).

8. set_bit. — Access mode: `https://www.kernel.org/doc/htmldocs/kernel-api/API-set-bit.html` (online; accessed: 18.12.2020).

9. EV_KEY. — Access mode: `https://elixir.bootlin.com/linux/latest/source/include/uapi/linux/input-event-codes.h#L39` (online; accessed: 18.12.2020).

10. input_report_key. — Access mode: `https://elixir.bootlin.com/linux/latest/source/include/linux/input.h#L415` (online; accessed: 18.12.2020).

11. usb_register. — Access mode: https://elixir.bootlin.com/linux/latest/source/include/linux/usb.h#L1289 (online; accessed: 18.12.2020).

12. Behdad Esfahbod Hamed Hatami. Keyboard Layouts:From QWERTY to Dvorak. — Computer Engineering DepartmentSharif University of Technology Tehran, Iran, 2003.

Листинг 3.1 — Объявление начальных макросов

```
1  #include <linux/module.h>
2  #include <linux/init.h>
3  #include <linux/kernel.h>
4  #include <linux/usb/input.h>
5  #include <linux/slab.h>
6  #include <linux/workqueue.h>
7
8  #define DRIVER_NAME     "keyboard_tablet"
9  #define DRIVER_AUTHOR   "Alexander Stepanov"
10 #define DRIVER_DESC     "Simulate table like a keyboard."
11 #define DRIVER_LICENSE "GPL"
12
13 MODULE_AUTHOR(DRIVER_AUTHOR);
14 MODULE_DESCRIPTION(DRIVER_DESC);
15 MODULE_LICENSE(DRIVER_LICENSE);
16
17 #define ID_VENDOR_TABLET  0x056a /* Wacom Co. */
18 #define ID_PRODUCT_TABLET 0x0301 /* Ltd CTL-671 */
19
20 #define USB_PACKET_LEN  10
21 #define WHEEL_THRESHOLD 4
22
23 #define MAX_X 1920
24 #define MAX_Y 1080
25
26 #define MAX_VALUE 0x7F
27
28 #define X_FACTOR (MAX_X / MAX_VALUE + 1)
29 #define Y_FACTOR (MAX_Y / MAX_VALUE + 1)
```

Листинг 3.2 — Объявление структур и глобальных переменных

```
1  struct tablet {
2      unsigned char      *data;
3      dma_addr_t          data_dma;
4      struct input_dev   *input_dev;
5      struct usb_device  *usb_dev;
6      struct urb         *irq;
7      int                 old_wheel_pos;
8      char                phys[32];
9  };
10
11 typedef struct tablet tablet_t;
```

```
12
13 struct container_urb {
14     struct urb *urb;
15     struct work_struct work;
16 };
17
18 typedef struct container_urb container_urb_t;
19
20 static bool pen_enter;
21 static int pressed_key;
22
23 static struct workqueue_struct *workq;
24
25 static struct input_dev *keyboard;
26
27 static int keys[5][14] = {
28     { KEY_ESC, 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 },
29     { 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 },
30     { 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 },
31     { KEY_LEFTSHIFT, 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_RIGHTSHIFT },
32     { KEY_LEFTCTRL, KEY_LEFTMETA, KEY_LEFTALT, KEY_SPACE, KEY_SPACE,
          KEY_SPACE, KEY_SPACE, KEY_SPACE, KEY_SPACE, KEY_SPACE, KEY_RIGHTALT,
          0, 0, KEY_RIGHTCTRL },
33 };
34
35 static int extra_keys[2][3] = {
36     { KEY_DELETE, KEY_UP, KEY_GRAVE },
37     { KEY_LEFT, KEY_DOWN, KEY_RIGHT },
38 };
```

Листинг 3.3 — Инициализация модуля

```
1  static int __init keyboard_tablet_init(void) {
2      int result = usb_register(&tablet_driver);
3
4      if (result < 0) {
5          printk(KERN_ERR "%s: usb register error\n", DRIVER_NAME);
6          return result;
7      }
8
9      workq = create_workqueue("workqueue");
10     if (workq == NULL) {
```

```
11          printk(KERN_ERR "%s: allocation workqueue error\n", DRIVER_NAME);
12          return -1;
13      }
14
15      keyboard = input_allocate_device();
16      if (keyboard == NULL) {
17          printk(KERN_ERR "%s: allocation device error\n", DRIVER_NAME);
18          return -1;
19      }
20
21      keyboard->name = "virtual keyboard";
22
23      set_bit(EV_KEY, keyboard->evbit);
24
25      for (i = 0; i < 5; ++i) {
26          for (j = 0; j < 14; ++j) {
27              if (keys[i][j] != 0) {
28                  set_bit(keys[i][j], keyboard->keybit);
29              }
30          }
31      }
32
33      for (i = 0; i < 2; ++i) {
34          for (j = 0; j < 3; ++j) {
35              set_bit(extra_keys[i][j], keyboard->keybit);
36          }
37      }
38
39      result = input_register_device(keyboard);
40      if (result != 0) {
41          printk(KERN_ERR "%s: registration device error\n", DRIVER_NAME);
42          return result;
43      }
44
45      printk(KERN_INFO "%s: module loaded\n", DRIVER_NAME);
46      return 0;
47  }
```

Листинг 3.4 — Выгруза модуля и установка функций init и exit

```
1  static void __exit keyboard_tablet_exit(void) {
2      flush_workqueue(workq);
3      destroy_workqueue(workq);
4      input_unregister_device(keyboard);
5      usb_deregister(&tablet_driver);
6      printk(KERN_INFO "%s: module unloaded\n", DRIVER_NAME);
7  }
```

```
 8
 9  module_init(keyboard_tablet_init);
10  module_exit(keyboard_tablet_exit);
```

Листинг 3.5 — Объявление экземпляра usb драйвера

```
1  static struct usb_driver tablet_driver = {
2      .name       = DRIVER_NAME,
3      .probe      = tablet_probe,
4      .disconnect = tablet_disconnect,
5      .id_table   = tablet_table,
6  };
```

Листинг 3.6 — Функция подключения планшета

```
1  static int tablet_probe(struct usb_interface *interface, const struct
       usb_device_id *id) {
2      struct usb_device *usb_device = interface_to_usbdev(interface);
3      tablet_t *tablet;
4      struct input_dev *input_dev;
5      struct usb_endpoint_descriptor *endpoint;
6      int error = -ENOMEM;
7
8      printk(KERN_INFO "%s: probe checking tablet\n", DRIVER_NAME);
9
10     tablet = kzalloc(sizeof(tablet_t), GFP_KERNEL);
11     input_dev = input_allocate_device();
12     if (!tablet || !input_dev) {
13         input_free_device(input_dev);
14         kfree(tablet);
15
16         printk(KERN_ERR "%s: error when allocate device\n", DRIVER_NAME);
17         return error;
18     }
19
20     tablet->data = (unsigned char *)usb_alloc_coherent(usb_device,
           USB_PACKET_LEN, GFP_KERNEL, &tablet->data_dma);
21     if (!tablet->data) {
22         input_free_device(input_dev);
23         kfree(tablet);
24
25         printk(KERN_ERR "%s: error when allocate coherent\n", DRIVER_NAME);
26         return error;
27     }
28
29     tablet->irq = usb_alloc_urb(0, GFP_KERNEL);
30     if (!tablet->irq) {
```

```
31          usb_free_coherent(usb_device, USB_PACKET_LEN, tablet->data,
                tablet->data_dma);
32          input_free_device(input_dev);
33          kfree(tablet);
34
35          printk(KERN_ERR "%s: error when allocate urb\n", DRIVER_NAME);
36          return error;
37      }
38
39      tablet->usb_dev = usb_device;
40      tablet->input_dev = input_dev;
41
42      usb_make_path(usb_device, tablet->phys, sizeof(tablet->phys));
43      strlcat(tablet->phys, "/input0", sizeof(tablet->phys));
44
45      input_dev->name = DRIVER_NAME;
46      input_dev->phys = tablet->phys;
47      usb_to_input_id(usb_device, &input_dev->id);
48      input_dev->dev.parent = &interface->dev;
49
50      input_set_drvdata(input_dev, tablet);
51
52      input_dev->open = tablet_open;
53      input_dev->close = tablet_close;
54
55      endpoint = &interface->cur_altsetting->endpoint[0].desc;
56
57      usb_fill_int_urb(
58          tablet->irq, usb_device,
59          usb_rcvintpipe(usb_device, endpoint->bEndpointAddress),
60          tablet->data, USB_PACKET_LEN,
61          tablet_irq, tablet, endpoint->bInterval
62      );
63
64      usb_submit_urb(tablet->irq, GFP_ATOMIC);
65
66      tablet->irq->transfer_dma = tablet->data_dma;
67      tablet->irq->transfer_flags |= URB_NO_TRANSFER_DMA_MAP;
68
69      error = input_register_device(tablet->input_dev);
70      if (error) {
71          usb_free_urb(tablet->irq);
72          usb_free_coherent(usb_device, USB_PACKET_LEN, tablet->data,
                tablet->data_dma);
73          input_free_device(input_dev);
74          kfree(tablet);
```

```
75
76          printk(KERN_ERR "%s: error when register device\n", DRIVER_NAME);
77          return error;
78      }
79
80      usb_set_intfdata(interface, tablet);
81
82      pen_enter = false;
83      printk(KERN_INFO "%s: device is conected\n", DRIVER_NAME);
84
85      return 0;
86  }
```

Листинг 3.7 — Функции открытия и закрытия устройства ввода

```
1  static int tablet_open(struct input_dev *dev) {
2      tablet_t *tablet = input_get_drvdata(dev);
3
4      tablet->old_wheel_pos = -WHEEL_THRESHOLD - 1;
5      tablet->irq->dev = tablet->usb_dev;
6      if (usb_submit_urb(tablet->irq, GFP_KERNEL))
7          return -EIO;
8
9      return 0;
10 }
11
12 static void tablet_close(struct input_dev *dev) {
13     tablet_t *tablet = input_get_drvdata(dev);
14     usb_kill_urb(tablet->irq);
15 }
```

Листинг 3.8 — Функция отключения планшета

```
1  static void tablet_disconnect(struct usb_interface *interface) {
2      tablet_t *tablet = usb_get_intfdata(interface);
3      usb_set_intfdata(interface, NULL);
4
5      if (tablet) {
6          usb_kill_urb(tablet->irq);
7          input_unregister_device(tablet->input_dev);
8          usb_free_urb(tablet->irq);
9          usb_free_coherent(interface_to_usbdev(interface), USB_PACKET_LEN,
               tablet->data, tablet->data_dma);
10         kfree(tablet);
11
12         printk(KERN_INFO "%s: device was disconected\n", DRIVER_NAME);
13     }
```

```
14 }
```

Листинг 3.9 — Таблицы подключаемых устройств

```
1  static struct usb_device_id tablet_table [] = {
2      { USB_DEVICE(ID_VENDOR_TABLET, ID_PRODUCT_TABLET) },
3      { },
4  };
5
6  MODULE_DEVICE_TABLE(usb, tablet_table);
```

Листинг 3.10 — Функция перехвата прерывания

```
1  static void tablet_irq(struct urb *urb) {
2      container_urb_t *container = kzalloc(sizeof(container_urb_t),
           GFP_KERNEL);
3      container->urb = urb;
4      INIT_WORK(&container->work, work_irq);
5      queue_work(workq, &container->work);
6  }
```

Листинг 3.11 — Функция обработки прерывания

```
1  static void work_irq(struct work_struct *work) {
2      container_urb_t *container = container_of(work, container_urb_t, work);
3      struct urb *urb;
4      int retval;
5      u16 x, y;
6      tablet_t *tablet;
7      unsigned char *data;
8
9      if (container == NULL) {
10         printk(KERN_ERR "%s: %s - container is NULL\n", DRIVER_NAME,
               __func__);
11         return;
12     }
13
14     urb = container->urb;
15     tablet = urb->context;
16     data = tablet->data;
17
18     if (urb->status != 0) {
19         printk(KERN_ERR "%s: %s - urb status is %d\n", DRIVER_NAME,
               __func__, urb->status);
20         kfree(container);
21         return;
22     }
```

```
23
24      switch(data[1]) {
25          case 0xf1:
26              if (!pen_enter) {
27                  x = data[3] * X_FACTOR;
28                  y = data[5] * Y_FACTOR;
29
30                  down_keyboard(x / 16, y / 9);
31
32                  printk(KERN_INFO "%s: pen enters %d %d\n", DRIVER_NAME, x,
                        y);
33                  pen_enter = true;
34              }
35              break;
36          case 0xf0:
37              if (pen_enter) {
38                  up_keyboard();
39
40                  printk(KERN_INFO "%s: pen leaves\n", DRIVER_NAME);
41                  pen_enter = false;
42              }
43              break;
44          default:
45              break;
46      }
47
48      retval = usb_submit_urb (urb, GFP_ATOMIC);
49      if (retval)
50          printk(KERN_ERR "%s: %s - usb_submit_urb failed with result %d\n",
               DRIVER_NAME, __func__, retval);
51
52      kfree(container);
53  }
```

Листинг 3.12 — Функции нажатия и отжатия клавиши

```
1  static void down_keyboard(u16 x, u16 y) {
2      press_key(x, y);
3      input_report_key(keyboard, pressed_key, 1);
4      input_sync(keyboard);
5  }
6
7  static void up_keyboard(void) {
8      input_report_key(keyboard, pressed_key, 0);
9      input_sync(keyboard);
10 }
```

Листинг 3.13 — Функция перевода координат в клавишу

```
1 static void press_key(u16 x, u16 y) {
2     y = y >= 50 ? 49 : y;
3     pressed_key = keys[(y / 10) % 5][x / 6];
4     if (pressed_key == 0) {
5         pressed_key = extra_keys[((y - 40) / 6) % 2][((x - 66) / 4) % 3];
6     }
7     printk(KERN_INFO "%s: pressed %x\n", DRIVER_NAME, pressed_key);
8 }
```