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 16 стр., 9 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 5

1 Аналитический раздел 6

1.1 Загружаемый модуль ядра 6

1.1.1 Драйвер устройства 6

1.2 Подключение графического планшета 6

1.2.1 Прерывания 6

1.2.2 Драйвер usb устройства 7

1.2.2.1 Структура usb_driver 7

1.2.2.2 Работа драйвера usb устройства 8

1.2.2.3 Подключение графического планшета 8

1.2.3 Многозадачность для прерываний 8

1.2.3.1 Тасклеты 9

1.2.3.2 Очереди работ 10

1.3 Работа клавиатуры 11

1.3.1 Подсистема ввода/вывода 11

1.3.1.1 Установка событий 12

1.3.1.2 Вызов событий 12

1.4 Вывод 12

2 Конструкторский раздел 13

3 Технологический раздел 14

Заключение 15

Список использованных источников 16

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 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.

1.1.1 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.

1.2 Подключение графического планшета

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

1.2.1 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.

1.2.2 Драйвер usb устройства

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

1.2.2.1 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).

1.2.2.2 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].

1.2.2.3 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.

1.2.3 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.

С современных ОС Linux имеется три типа нижних половин (bottom half):

— softirq – отложенные прерывания;

— tascklet – тасклеты;

— workqueue – очереди работ.

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

1.2.3.1 Тасклеты

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

Тасклеты описываются следующей структурой, описанной на листинге 1.1 [3].

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 [3]:

— 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 [3].

1.2.3.2 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 [4].

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 [4].

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 [4].

1.3 Работа клавиатуры

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

1.3.1 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 [5].

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 [6]. Чтобы инициализировать эту структуру используется функция set_bit, которая принимает два аргумента: бит, который устанавливается, и адрес, куда этот бит устанавливать [7].

1.3.1.1 Установка событий

Для установки типа события, которое будет вызываться необходимо установить бит EV_KEY [8] в поле evbit в структуре input_dev [6]. 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).

1.3.1.2 Вызов событий

Для вызова событий, связанных с клавишами используется системный вызов input_report_key [9], 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.

1.4 Вывод

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

2 Конструкторский раздел

3 Технологический раздел

ЗАКЛЮЧЕНИЕ

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

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

5. 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.

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

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

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

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