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*

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:

Драйвер для использования графического планшета Wacom Bamboo CTH-670 в качестве клавиатуры

Студент ИУ7-61Б _____ А.Д. Сушина
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ М.В. Филиппов
(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)
« _____ » _____ 2020 г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине _____Операционные системы_____

Студент группы ИУ7-61б

Сушина Анастасия Дмитриевна
(Фамилия, имя, отчество)

Тема курсового проекта Драйвер для использования графического планшета Wacom Bamboo CTH-670 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
Разработать драйвер для устройства графический планшет Wacom Bamboo CTH-670, который позволит использовать устройство в качестве клавиатуры. Драйвер разработать для операционный системы Linux.

Оформление курсового проекта:

Расчетно-пояснительная записка на 25-30 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
На защиту проекта должна быть представлена презентация, состоящая из 10-15 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, интерфейс, характеристики разработанного ПО.

Дата выдачи задания « ___ » _____ 20__ г.

Руководитель курсового проекта _____ Филиппов М.В.
(Подпись, дата) (И.О.Фамилия)

Студент _____ Сушина А.Д.
(Подпись, дата) (И.О.Фамилия)

Введение

В современном мире люди каждый день используют компьютеры и телефоны для обмена информацией. Чтобы была возможность производить поиск информации и набирать ее необходима клавиатура.

Клавиатура — это устройство, позволяющее пользователю вводить информацию, т. е. устройство ввода. Устройство представляет собой набор клавиш, расположенных в определенном порядке.

Кроме физических устройств существуют экранные клавиатуры, где клавиши — это определенные части экрана, на которые можно нажать с помощью мыши или пальца. Однако зачастую этот способ набора информации неудобен: мышь не позволяет нажимать на клавиши достаточно быстро, а на устройствах с сенсорным экраном могут быть слишком маленькие кнопки, по которым не всегда можно попасть пальцем.

В качестве клавиатуры может работать, например, графический планшет, если разделить его на зоны, каждая из которых будет отвечать за определенную кнопку. Однако, устройство не может работать так само по себе. Необходимо написать драйвер, который позволит операционной системе воспринимать сигналы устройства так, как этого хочет пользователь.

Использование устройства таким образом может быть полезно, если у пользователя нет компьютерной клавиатуры или ему неудобно использовать встроенную экранную клавиатуру планшетов. Такое использование планшета может повысить скорость набора информации: можно нажимать на клавиши сразу несколькими пальцами, а кнопки достаточно большие, чтобы не было возможности промахнуться.

Целью данной работы является разработка драйвера графического планшета, который позволит использовать планшет в качестве клавиатуры. Драйвер должен быть реализован для ОС Linux.

Для достижения этой цели были поставлены следующие задачи:

1. Изучить, каким образом реализуются драйвера ОС Linux.
2. Опередить, к какому типу устройств относится графический планшет, каким образом драйвер сможет получать информацию о нажатии на графическим планшет.
3. Изучить способы имитации работы клавиатуры.
4. Разработать драйвер для графического планшета.

1 Аналитический раздел

В аналитическом разделе будет выполнена постановка задачи, а также приведены методы решения задачи.

1.1 Постановка задачи

Необходимо разработать драйвер для графического планшета Wacom Bamboo CTH-670, который позволит использовать планшет в качестве клавиатуры. Драйвер должен быть разработан для операционной системы Linux.

Экран графического планшета должен быть разделен на зоны, каждая из которых будет отвечать за свою кнопку. При нажатии на планшет, драйвер должен определить, на какую зону было совершено нажатие и сообщить операционной системе, какая кнопка была нажата.

1.2 Графический планшет

Чтобы понимать, каким образом должен быть реализован драйвер, необходимо сначала определить, для какого устройства он будет разработан и как это устройство работает.

Графический планшет - это устройство для ввода информации, созданной от руки, непосредственно в компьютер.

Планшет Wacom Bamboo CTH-670 состоит из стилуса и плоского планшета с сенсорным экраном и несколькими кнопками в левой части. Данный планшет реагирует на нажатие на сенсорный экран пальцем или пером, а также на близость пера. К компьютеру планшет подключается с помощью usb. Устройство представлено на рисунке 1.

Рис. 1 Графический планшет Wacom Bamboo CTH-670

1.3 Драйвера в ОС Linux

Драйвер — это программное обеспечение, с помощью которого операционная система получает доступ к аппаратному обеспечению некоторого устройства. В Linux драйвера — это «черные ящики», которые заставляют специфичную часть оборудования соответствовать строго заданному программному интерфейсу; они полностью скрывают детали того, как работает устройство [1]. Этот программный интерфейс таков, что драйверы могут быть собраны отдельно от остальной части ядра и подключены в процессе работы, когда это необходимо.

Драйвера устройств необходимы в операционной системе, так как каждое отдельное устройство воспринимает только свой строго фиксированный набор специализированных команд, с помощью которых им можно управлять. Приложения обычно используют команды высокого уровня, преподставляемого ОС, а преобразовывает эти команды в управляющие последовательности для конкретного устройства драйвер этого устройства. Поэтому каждое отдельное устройство должно иметь свой программый драйвер, который выполняет роль связующего звена между аппаратной частью устройства и программными приложениями, использующими это устройство.

В Linux драйверы устройств бывают трех типов:

- Драйверы первого типа являются частью программного кода ядра. Соответствующие устройства автоматически обнаруживаются системой и становятся доступны доя приложений.
- Драйверы второго типа представлены загружаемыми модулями ядра. Они оформлены в виде отдельных файлов. Для их подключения необходимо выполнить команду подключения модуля. Если необходимость в использовании отпала, модуль можно выгрузить из памяти. Использование модулей обеспечивает большую гибкость, так как каждый драйвер может быть переконфигурирован без остановки системы.
- В драйверах третьего типа программный код драйвера поделен между ядром и специальной утилитой, предназначенной для управления данным устройством.

Во всех драйверах взаимодействие с устройством осуществляет ядро или модуль ядра, а пользовательские программы взаимодействуют через специальный файлы, расположенные в каталоге /dev и его подкаталогах.

Драйвер графического планшета может быть реализован как драйвер второго типа. Так, у пользователя будет возможность подключить и отключить драйвер в любой момент.

1.4 Загружаемые модули ядра

Загружаемые модули ядра Linux позволяют добавлять функциональность в ядро, когда система запущена и работает [1].

Часть кода, которая может быть добавлена в ядро во время работы, называется модулем. Ядро Linux предлагает поддержку довольно большого числа типов модулей, включая драйвера устройств. Каждый модуль является подготовленным объектным кодом, который может быть динамически подключен в работающее ядро программой insmod и отключен программой rmmod.

Каждый модуль обычно классифицируется как символьный модуль, блочный модуль или сетевой модуль. Однако существуют и другие пути классификации модулей-драйверов, которые по-другому подразделяют устройства. Так, можно говорить о модулях универсальной последовательной шины (USB), последовательных модулях, модулях SCSI и других.

Каждое USB-устройство управляется модулем USB, который работает с подсистемой USB, но само устройство представлено в системе или как симпольное устройство, или как блочное устройство, или как сетевой интерфейс.

1.5 Драйвера USB-устройств

Universal Serial Bus (USB, Универсальная Последовательная Шина) является соединением между компьютером и несколькими периферийными устройствами.

Ядро Linux поддерживает два основных типа драйверов USB:

- Драйверы на хост-системе. Они управляют USB-устройствами, которые к ней подключены, с точки зрения хоста (обычно хостом USB является персональный компьютер.)
- Драйверы на устройстве. Они контролируют, как одно устройство видит хост-компьютер в качестве устройства USB.

В ОС Linux USB устройства описываются структурой usb_device.

1.5.1 Основы USB устройства

Ядро Linux предоставляет подсистему, называемую ядром USB. Ядро USB предоставляет интерфейс для драйверов USB, используемый для доступа и управления USB оборудованием, без необходимости беспокоится о различных типах аппаратных контроллеров USB, которые присутствуют в системе.

USB - устройства всегда отвечают на запросы host-компьютера, но они никогда не могу посылать информацию самостоятельно. Передача данных выполняется между буфером в памяти хост компьютера и конечной точкой универсальной

последовательной шины USB устройства. Перед передачей данные организуются в пакеты.

В составе USB-функции, то есть в устройстве с интерфейсом USB, имеется периферийный контроллер USB. Этот контроллер имеет две основные функции: он взаимодействует с USB-системой и содержит в себе буферы в количестве от одного до шестнадцати, называемые конечными точками. Конечная точка USB может переносить данные только в одном направлении, либо со стороны хост-компьютера на устройство (OUT) или от устройства на хост-компьютер (IN).

Конечная точка USB может быть одной из четырёх типов, которые описывают, каким образом передаются данные:

1. Control. Передача типа control является двунаправленным и предназначен для обмена с устройством короткими пакетами типа «вопрос-ответ». Обеспечивает гарантированную доставку данных.
2. Isochronous. Изохронный канал имеет гарантированную пропускную способность (N пакетов за один период шины) и обеспечивает непрерывную передачу данных. Передача осуществляется без подтверждения приема.
3. Interrupt. Канал прерывания позволяет доставлять короткие пакеты без гарантии доставки и без подтверждений приема, но с гарантией времени доставки – пакет будет доставлен не позже, чем через N миллисекунд.
4. Bulk. Поточные оконечные точки передают большие объёмы данных.Этим передачам протокол USB не гарантирует выполнения в определённые сроки. Если на шине нет достаточного места, чтобы отправить целый пакет BULK, он распадается на несколько передач в или из устройства.

Конечные точки USB описаны в ядре структурой struct usb_host_endpoint. Эта структура содержит реальную информацию конечно точки в другой структуре struct usb_endpoint_descriptor. Из нее можно получить информацию об адресе данной

конечной точки, ее типе, максимальном размере информации, которую эта конечная точка может обработать за раз.

Конечные точки USB завернуты в интерфейсы [1]. USB-интерфейсы обрабатывают только один тип логического соединения, такого как мышь, клавиатура или аудио-поток. Некоторые usb-устройства имеют несколько интерфейсов. Каждый драйвер USB управляет интерфейсом. USB-интерфейсы описаны в ядре структурой struct usb_interface.

Драйверу USB устройства обычно требуется преобразовать данные из заданной структуры struct usb_interface в структуру struct usb_device, которая необходима ядру USB для широкого круга функциональных вызовов. Для этого предоставляется функция interface_to_usbdev.

1.5.2 Блоки запроса USB

Код USB в ядре Linux взаимодействует со всеми устройствами USB помощью так называемых urb (USB request block, блок запроса USB). Этот блок запроса описывается структурой struct urb.

URB используется для передачи или в или из заданной конечной точки USB на заданное USB устройство в асинхронном режиме. В зависимости от потребностей, драйвер USB устройства может выделить для одной конечной точки много блоков или может повторно использовать один urb для множества разных конечных точек.

Типичный жизненный цикл содержит следующие шаги:

1. Создание urb драйвером USB.
2. Назначение urb в определенную точку конечную точку.
3. Передача драйвером USB устройства в USB ядро.
4. Передача urb драйвером USB в заданный драйвер контроллера USB узла для указанного устройства

5. Обработка драйвером контроллера USB узла, который выполняет передачу в USB устройство.
6. После завершения работы с urb драйвер контроллера USB узла уведомляет драйвер USB устройства.

URB может быть отменен драйвером, который передал urb или драйвером USB. URB создается динамически и содержит счетчик ссылок, что позволяет освободить urb автоматически, когда освобождается блок последним пользователем.

Структура struct urb не должна быть создана статически в драйвере или внутри другой структуры, потому что это нарушит схему подсчёта ссылок, используемую USB ядром для urb-ов. Она должна быть создана вызовом функции usb_alloc_urb. Для того, чтобы сказать USB ядру, что драйвер закончил работу с urb-ом, драйвер должен вызвать функцию usb_free_urb.

Для правильной инициализации urb-а, посылаемого в конечную точку устройства используются функции usb_fill_int_urb, usb_fill_bulk_urb, usb_fill_control_urb, которые используются для urb-ов прерывания, поточных и управляющих urb-ов соответсвенно. Изохронные Urb не имеют функции инициализации, поэтому должны быть проинициализированны вручную.

После того, как urb был надлежащим образом создан и проинициализирован USB драйвером, он готов быть отправленным в USB ядро для передачи в USB устройство. Это делается с помощью вызова функции usb_submit_urb.

Если вызов usb_submit_urb был успешен, передавая контроль над urb в USB ядро, функция возвращает 0; иначе возвращается отрицательное число ошибки. Если функция завершается успешно, завершающий обработчик urb (задаваемый указателем на функцию complete) вызывается только один раз, когда urb завершается. Когда вызывается эта функция, ядро USB завершает работу с URB и контроль над ним теперь возвращается драйверу устройства.

1.5.3 Регистрация USB-драйвера

USB драйвер является драйвером устройства, т.е. он должен подключиться к реальному устройству в пространстве аппаратных средств. Загрузка подходящего драйвера осуществляется по комбинации PID/VID (Product ID/Vendor ID). Регистрация устройства выполняется с помощью команд usb_register, в которую передается указатель на струкруту usb_driver.

В этой структуре описываются сведения о драйвере. Она должна быть заполненна драйвером USB. В соответствующих полях должны быть указаны имя устройства - name, идентификационная таблица – id_table, используемая для автоматического обнаружения конкретного устройства, и две функции обратного вызова – probe и disconnect, которые вызываются ядром USB при горячем подключении и отключении устройства, соответственно.

Структура struct usb_device_id содержит список различных типов USB устройств, которые поддерживает этот драйвер. Этот список используется ядром USB, чтобы решить, какой драйвер предоставить устройству, или скриптами горячего подключения, чтобы решить, какой драйвер автоматически загрузить, когда устройство подключается к системе. Эта структура записывается в поле id_table структуры usb_driver.

1.6 Драйвера ввода

Для того, чтобы объединить рассеянные драйверы устройств ввода данных, была создана подсистема ввода ядра [2]. Эта подсистема дает следующие преимущества:

- Единообразную обработку функционально похожих устройств ввода.
- Удобный интерфейс для событий отправки пользовательским приложениям сообщений о вводе.
- Выделение из входных драйверов общих частей. Это упрощает драйверы.

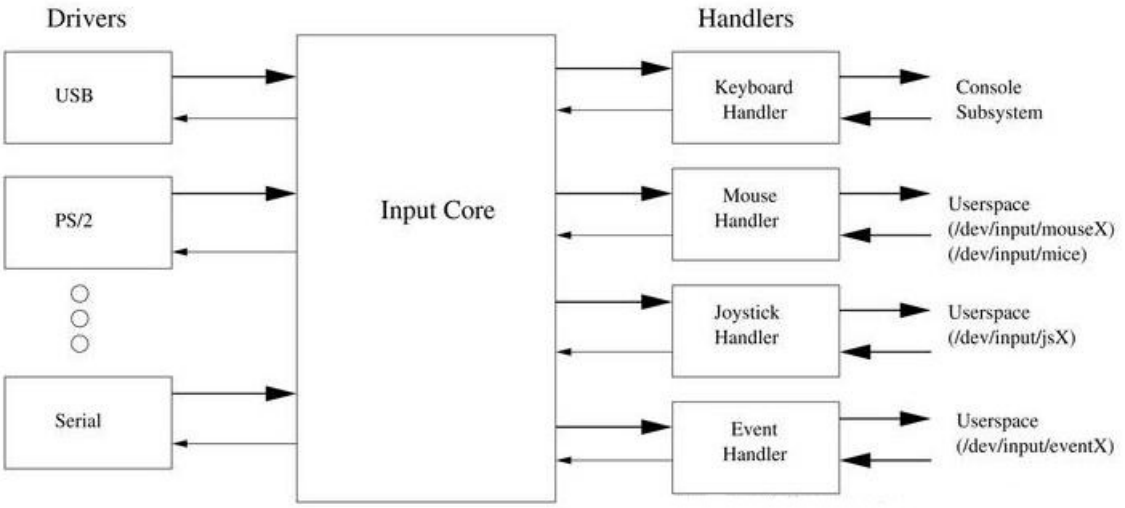

Рис 2. Подсистема ввода [5]

Спецификациями usb, связанными с устройствами взаимодействия с человеком(HID) предусмотрен протокол, по которому для взаимодействия используются usb-клавиатуры, мыши, наборы кнопок и другие устройства ввода. В Linux это осуществляется через клиентский драйвер usbhid. Он регистрирует себя в качестве драйвера устройства ввода. Таким образом, если стоит задача написать свой драйвер устройства ввода, сначала необходимо выгрузить модуль usbhid, иначе он установит стандартный драйвер для устройства.

Чтобы зарегистрировать драйвер в качестве драйвера устройста ввода, сначала необходимо выделить память для структуры input_dev с использованием функции input_allocate_device. Затем необходимо объявить, какие события будет генерировать устройство ввода с помощью функции set_bit. После этого можно непосредственно зарегистрировать устройство ввода, вызвав функцию input_register_device.

Для генерации событий используются функции input_report_key, input_report_rel и другие, в зависимости от типа события.

1.7 Обработка прерываний

Обратный вызов urb, описанный ранее, выполняется в контексте прерывания, поэтому к нему применимы те же методы, что и для обработки прерываний.

Когда операционная система получает прерывания из-за некоторого аппаратного события, обработка начинается в контексте прерывания. Обычно прерывание приводит к выполнению большого объема работы. Однако обработчику прерывания необходимо завершиться быстро и не держать прерывания надолго заблокированными. Эти две потребности конфликтуют друг с другом.

Решить эту проблему можно, разделив обработчик прерывания на верхнюю и нижнюю половину ядра. Нижняя половина является процедурой, которая планируется верхней половиной, чтобы быть выполненной позднее, в более безопасное время.

Верхняя половина сохраняет данные устройства в зависимый от устройства буфер, планирует свою нижнюю половину и выходит: эта операция очень быстрая. Затем нижняя половина выполняет все необходимые операции. Эта установка позволяет верхней половине обслужить новое прерывание, пока нижняя половина всё ещё работает [1].

Существует несколько способов реализации нижней половины обработчика: гибкое прерывание (softirq), такслет (tasklet) и очереди работ (workqueue).

1.7.1 Тасклеты

Softirq-функции изначально были созданы в виде вектора из 32 записей softirq, поддерживающих разнообразные программные прерывания. Сегодня только девять векторов используются для softirqs, один из которых TASKLET_SOFTIRQ. Хотя softirqs-функции все еще существуют в ядре, рекомендуется использовать тасклеты и рабочие очереди вместо размещения новых векторов softirq [4].

Тасклеты являются структурами отложенного исполнения, которые вы можете запланировать на запуск позже в виде зарегистрированных функций. Конкретный тасклет будет работать только на одном процессоре (для которого он запланирован), и один и тот же тасклет никогда не будет работать более чем на одном заданном процессоре одновременно. Но различные тасклеты могут одновременно работать на

разных процессорах. Тасклеты представлены в виде структуры tasklet_struct (см. рис.2), в которой содержатся все данные, необходимые для управления тасклетом и поддержания его работы.

1.7.2 Очереди работ

Вместо того, чтобы предлагать однократную схему отложенного исполнения, как в случае с тасклетами, очереди работ являются обобщенным механизмом отложенного исполнения, в котором функция обработчика, используемая для очереди работ, может "засыпать".

В очередях работ предлагается обобщенный механизм, в котором отсроченная функциональность переносится в механизмы выполнения нижних половин. В основе лежит очередь работ (структура workqueue_struct), которая является структурой, в которую помещается данные объект work. Работа (т.е. объект work) представлена структурой work_struct, в которой идентифицируется работа, исполнение которой откладывается, и функция отложенного исполнения, которая будет при этом использоваться. Потоки ядра выбирают работу (т.е. объект work) из очереди работ и активируют один из обработчиков нижней половины.

После того, как будет инициализирована структура для объекта work, следующим шагом будет помещение этой структуры в очередь работ. Вы можете сделать это несколькими способами. Можно просто добавить работу (объект work) в очередь работ с помощью функции queue_work(которая назначает работу текущему процессору). Либо с помощью функции queue_work_on можно указать процессор, на котором будет выполняться обработчик [4].

1.8 Выводы

В аналитическом разделе была выполнена постановка задачи. Были рассмотрены различные типы драйверов. Было установлено, что для графического планшета Wacom Bamboo CTH-670 необходимо будет написать драйвер USB-

устройства ввода, который позволит реализовать весь необходимый функционал.

Для обработки нижних половин прерываний будет использоваться очередь работ.

2 Конструкторский раздел

В конструкторском разделе необходимо продемонстрировать место разрабатываемого ПО в системе и в ядре, в частности. Определяется тип программного обеспечения, например, драйвер и приложение, взаимодействующее с драйвером, или драйвер и сервис, или для Linux драйвер и демон, и описывается взаимодействие модулей разработанного ПО через подсистему ввода/вывода, например, на основе событийной модели.

Описать структуру разрабатываемого драйвера, основные структуры ядра, описывающие драйвер и функции драйвера (точки входа).

Описывается реализуемая драйвером функция или функции и приводится схема алгоритма или схемы алгоритмов их работы по ГОСТу.

3 Технологический раздел

Обоснованно выбирается язык программирования и среда программирования (следует определить основные особенности языка, среды и специализированных библиотек и на этом основании сделать выбор языка программирования, средства или библиотеки).

Для драйверов, демонов или сервисов подробно описывается среда разработки; приводятся выбранные структуры данных и разработанные функции, реализующие поставленную задачу. Для патчей описать особенности реализации и перекомпиляции ядра ОС.

Обосновывается выбор интерфейса пользователя и приводится его описание; описываются действия по установке программного обеспечения. *Для Linux make файл.*

Заключение

В заключении кратко излагаются основные направления работы, особенности реализации, полученные результаты и выводы.

В выводах кратко по пунктам перечисляются основные результаты работы (например, 1. Показано .. 2. Исследовано/ны .. 3.Разработана/ны структура .. 4. Определена целесообразность .. 5. Разработано программное обеспечение .. 6. Тестирование показало .. и т. п.)

Список литратуры

1. 1. Драйверы Устройств Linux, Третья Редакция Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman

http://dmilvdv.narod.ru/Translate/LDD3/Linux_Device_Drivers_3_ru.pdf
2. 2. http://dmilvdv.narod.ru/Translate/ELDD/Essential_Linux_Device_Drivers_ru.pdf
3. 3 . https://valadoc.org/linux/Linux.Input.html
4. http://rus-linux.net/nlib.php?name=/MyLDP/kernel/api/kernelapi2.html
5. https://russianblogs.com/article/8248247507/

Приложения