Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

К ЗАЩИТЕ ДОПУСТИТЬ

_____ *Л. П. Поденок*

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему
ДРАЙВЕР USB КЛАВИАТУРЫ
БГУИР КП 1-40 02 01 310 ПЗ

Студент гр. 150503 А. В. Кутняк

Руководитель Д. В. Басак

Минск 2023

Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой
_____
(подпись)
_____ 2023 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту *Кутняку Алексею Викторовичу*
1. Тема проекта Драйвер USB клавиатуры
2. Срок сдачи студентом законченного проекта *15 мая 2023 г.*
3. Исходные данные к проекту *Язык программирования С*
4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке) *Введение. 1. Обзор литературы. 2. Системное проектирование. 3. Функциональное проектирование. 4. Разработка программных модулей. 5. Руководство пользователя. Заключение. Список использованных источников.*

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков) *1. Схема структурная 2. Диаграмма классов 3. Схема алгоритма.*
6. Консультант по проекту *Басак Д.В.*

7. Дата выдачи задания *18 февраля 2023 г.*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):
раздел 1 к 1 марта 2023 г. – 15 %;
разделы 2,3 к 1 апреля 2023 г. – 50 %;
разделы 4,5 к 1 мая 2023 г. – 80 %;
оформление пояснительной записки и графического материала к 15 мая 2023 г. – 100 %
Защита курсового проекта с 29 мая 2023 г. по 9 июня 2023 г.

РУКОВОДИТЕЛЬ _____ *Д.В. Басак*
(подпись)

Задание принял к исполнению _____ *А.В. Кутняк*
(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время наблюдается стремительный рост в области периферийных устройств: клавиатуры, компьютерные мыши, джойстики, вебкамеры, динамики, наушники, микрофоны, флеш-накопители, принтеры, сканеры, внешние диски и видеокарты – лишь немногая часть из того, что на сегодняшний день используется человеком в повседневной жизни. Каждое из этих устройтв имеет собственную реализацию, определяемую как назначением, так и производителем; порой даже те устройства, что имеют общее назначение, могут различаться по своей технической реализации. Однако большинство данных устройств имеют одну общую деталь, что позволяет использовать их повсеместно, и этой деталью является интерфейс последовательной шины USB.

С появлением последовательного интерфейса USB, а также его внедрения в вычислительную технику, была решена проблема существования множества интерфейсов для периферии, что существенно облегчило жизнь как производителям устройств и вычислительной техники, так и обычным пользователям.

Данное изменение не просто породило множество устройств, использующих общую технологию, но и потребовало написание не меньшего количества системного кода, поддерживающего их работу.

Одним из видов такого системного программного обеспечения является драйвер – посредник между ядром операционной системы и аппаратной реализацией устройства, предоставляющий пользователю доступ к пользованию устройством.

Тема данного курсового проекта – разработка драйвера устройства USB клавиатуры. Актуальность данного проекта обуславливается не только большой востребованностью программного обеспечения, поддерживающего работу USB-устройств, но и огромной теоретической и практической ценностью самой задачи, предоставляющей возможность изучить область разработки, не рассматриваемую в рамках основного курса дисциплины по системному программированию.

Ввиду того, что разработка драйвера во многом зависит не столько от самого устройства устройства, столько от целевой платформы, для которой он пишется, следует заранее обозначить, что данный драйвер будет разрабатываться под операционные системы на базе ядра Linux.

1 ОБЗОР ЛИТЕРАТУРЫ

Перед тем, как приступить к реализации собственного драйвера USB устройства, необходимо разобраться в том, что же такое USB, драйверы, как они работают, как ядро Linux взаимодействует с устройствами и какие инструменты предоставляет сама система для разработки драйверов устройств.

1.1 Интерфейс USB

USB (Universal Serial Bus) – это последовательный интерфейс, используемый для подключения периферийных устройств к вычислительному устройству. На сегодняшний день является основным интерфейсом подключения различной периферии к бытовой цифровой технике.

С технической точки зрения наиболее отличительной особенностью интерфейса является одновременная возможность передачи данных на высокой скорости и обеспечение питания подключенной периферии.

Название Universal Serial Bus, или универсальная последовательная шина, означает, что устройства, физически подключенные в виде древовидной структуре, воспронимаются на логическом уровне как элементы единой шины данных, обладающей единственным хост-контроллером, управляющим передачей информации между собой и устройствами.

Технология USB поддерживает «горячее» (plug'n'play) соединение устройства с динамически загружаемыми и выгружаемыми драйверами, что позволяет пользоваться устройством сразу после его подключения без необходимости самостоятельного запуска компонентов операционной системы или перезагрузки хост-устройства.

Спецификация интерфейса охватывает широкий круг вопросов, связанных с подключением и взаимодействием периферийных устройств с вычислительной системой:

- стандартизация разъемов и кабелей;
- нормирование электропотребления устройств;
- протоколы обмена данными;
- стандартизация функциональности и драйверов устройств.

Подробные технические сведения, а также информацию об USB, описанную в последующих разделах, можно найти на страницах спецификации интерфейса [1].

1.1.1 Протоколы USB

В отличие от RS-232 и схожих с ним последовательных интерфейсов, где формат передаваемых данных не задан, стек интерфейса состоит из нескольких слоев протоколов, однако, большинство контроллеров USB самостоятельно поддерживают нижние уровни стека протоколов, поэтому они остаются незаметными для конечного разработчика.

Как уже было сказано в разделе выше, USB является последовательной шиной, которым управляет хост, начинающий все транзакции передачи данных.

Каждая транзакция состоит из следующих частей:

– Token Packet – определяет, что передается далее;

– Data Packet – необязательная, содержит полезную нагрузку;

– Status Packet – возвращает статус транзакции.

Сообщение производится между хостом и определенной оконечной точкой устройства, называемой endpoint.

1.1.2 Типы пакетов

Спецификация определяет четыре различных типа пакета:

– Token - определяют тип последующей транзакции;

– Data - содержат полезную нагрузку;

– Handshake - подтверждают данные или ошибки;

– Start Of Frame - для указания начала фрейма.

Каждый из этих типов пакетов подразделяется на собственные подтипы, однако для нас интересны лишь подтипы, определяющие тип последующей транзакции:

– In - хост хочет прочитать информацию;

– Out - хост хочет отправить информацию;

– Setup - используется для указания начала управляющих передач.

1.1.3 Конечные точки

Конечные точки, или endpoints, – источники или приемники данных, относящиеся к тому или иному устройству.

Так как шина является хост-ориентированной, конечные точки всегда находятся на конце канала связи, поэтому сообщение с ней всегда проходит в одностороннем порядке.

Приведем пример сообщения между хостом и конечной точкой. Предположим, что хост отправляет некоторые данные на конечную точку. Данные приходят, после чего помещаются в выходной буфер. Когда аппаратное обеспечение будет готово, оно прочитает эти данные и обработает в соответствии со своей функцией. Если же устройство хочет вернуть данные, оно помещает их во входной буфер. Эти данные находятся там до тех пор, пока хост не отправит пакет IN, которым он запрашивает данные этой конечной точки.

1.1.4 Потоки сообщения

Для описания передачи данных со стороны клиента спецификация вводит понятие каналов (потоков) передачи данных.

Поток – это логическое соединение между хостом и конечной точкой, выполняющее передачу данных.

Потоки имеют набор параметров: тип передачи, направление потока данных, а также максимальные размеры пакета и буфера.

Простейшим примером потока является *поток по умолчанию*, являющийся двунаправленным потоком, составленным из нулевой входной и выходной точки с типом передачи control.

Спецификация определяет два типа потоков:

– Stream Pipes – не имеют предопределенного формата, являются последовательными и имеют предопределенное направление (in или out); поддерживают тип передачи bulk, isochronous и interrupt;

– Message Pipes – имеют предопределенный формат, пересылают данные в нужном направлении, указанном в запросе, и таким образом позволяют передавать данные в двух направлениях; поддерживают только тип передачи control.

1.1.5 Типы конечных точек

Спецификация определяет следующие четыре типа различных конечных точек:

– Control Endpoint – используются для передач контроля;

– Interrupt Endpoint – используются для передач прерываний;

– Isochronous Endpoint – используются для изохронных передач;

– Bulk Endpoint – используются для больших передач;

Передачи Control используются для отправки команд и операций. Именно эти передачи для основной настройки устройства USB перед началом его обслуживания хостом.

Передачи Interrupt используются в ситуациях, где требуется поведение, аналогичное прерыванию процессора от устройства. Поскольку все запросы исходят со стороны хоста, данные передачи работают на механизмах опроса: устройство ставит запрос на «прерывание» в очередь, пока хост не опросит устройство с целью получения этих данных.

Передачи Isochronous происходят переодически и продолжительное время. Обычно они содержат информацию, чувствительную к времени доставки, например аудио- или видеопотоки. Гарантируют определение ошибок с помощью контрольных сумм, но не гарантируют доставку, поскольку задержки и повторные передачи искажают изохронные данные сильнее, чем потеря их части.

Передачи Bulk используются для передачи больших объемов данных на высокой скорости [1)]. Если полезная нагрузка меньше максимального размера пакета, то нулями она не дополняется.

1.2 Дескрипторы USB

Каждое USB устройство имеет сложную иерархию дескрипторов, которые описывают некоторую информацию для хоста. К данной информации относится название устройства, его производитель, какими способами его можно сконфигурировать, число конечных точек, их типы и др.

Существуют следующие типы дескрипторов:

– Дескрипторы устройства – описывают сведения, относящиемся к общим сведениям об устройстве;

– Дескрипторы конфигурации – описывают сведения об определенной конфигурации устройства;

– Дескрипторы интерфейса – описывают сведения об определенном интерфейсе устройства;

– Дескрипторы конечной точки – описывают сведения об определенной конечной точке устройства;

– Строковые декскрипторы – содержат строковые данные.

[1)] Последний стандарт интерфейса определяет шесть спецификаций скорости передачи данных: USB 1.0/Low-Speed (1.5 Mbps), USB 1.1/Full-Speed (14 Mbps), USB 2.0/Hi-Speed (480 Mbps), USB 3.0/SuperSpeed (5 Gbps), USB 3.1/SuperSpeed+ (10 Gbps), USB 3.2/SuperSpeed++ (20 Gbps).

Все перечисленные дескрипторы, за исключением строковых, ссылаются на другие дескрипторы, в результате чего образуется дерево дескрипторов устройства, пример которого изображен на рисунке 1.1.

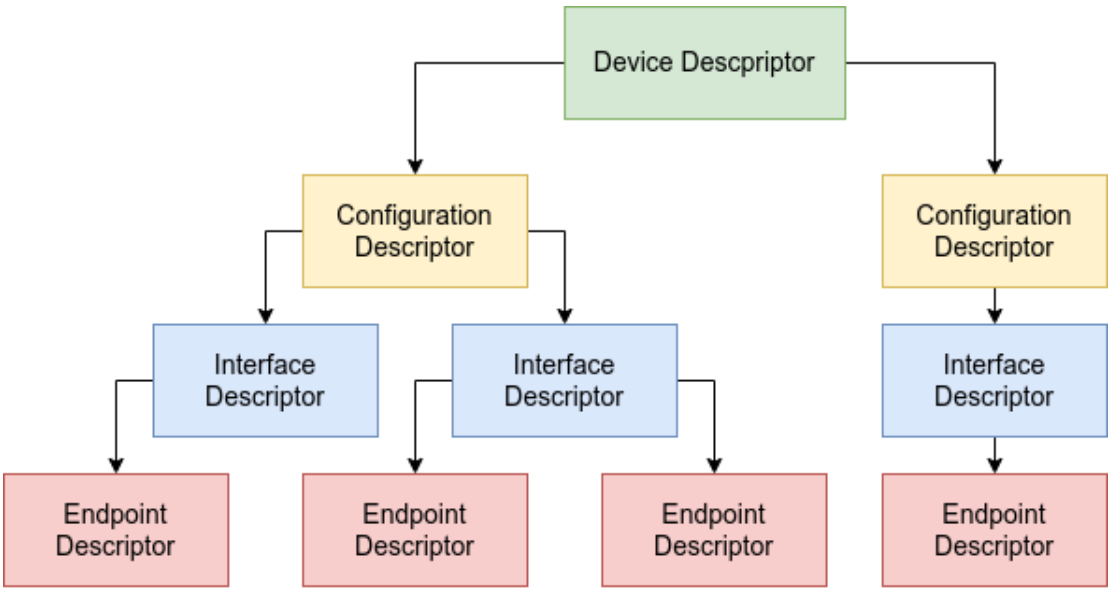


Рисунок 1.1 – Дескрипторы устройства USB

Дескриптор устройства представлен в единственном числе и включает в себя такую информацию, как ProductID, VendorID, а также количество возможных конфигураций.

Дескриптор конфигурации указывает величину мощности, потребляемой от последовательной шины, питается ли устройство от собственного источника, а также количество интерфейсов, которые есть в данной конфигурации. Решение о выборе конфигурации принимается хостом после прочтения всех конфигураций. Например, несколько конфигураций можно использовать для определения типа питания портативного устройства: одна конфигурация будет устанавливать высокое энергопотребление от шины, другая – питание от аккумулятора устройства.

Дескриптор интерфейса можно рассматривать как группу конечных точек, выполняющую отдельную функцию устройства.

Отличительной особенностью дескриптора интерфейса является наличие у него двух индексных полей – bInterfaceNumber, означающий номер интерфейса, и bAlternateSetting, которое позволяет интерфейсу выбрать альтернативную настройку. По умолчанию устройство будет использовать интерфейс с наименьшим bAlternateSetting, но хост может изменить это, отправив запрос SetInterface на этот интерфейс с другой bAlternateSetting.

Дескриптор конечной точки используется для указания типа точки, ее направления, интервала опроса и максимального размера пакета. Существует *нулевая конечная точка*, которая используется с упомянутым ранее потоком передачи по умолчанию, а потому не имеет дескриптора.

В данном разделе отсутствует описание полной структуры дескрипторов, а также их полей, так как типовое взаимодействие с данными из них возложено на инструменты, предоставляемые операционной системой.

1.3 Класс устройств USB HID

USB HID (Human Interface Device) – класс устройств, предназначенных для взаидойствия с человеком: клавиатура, мышь, джойстик и др.

Существует документ, описывающий дескрипторы устройств, в частности, списки кодов функций устройств данного класса [2].

Поскольку класс содержит множество предопределенных функций для каждого типа устройств, это позволяет ожидать, что устройство, разработанное согласно спецификации класса, будет работать с программным обеспечением, которое эту спецификацию поддерживает.

Устройства данного класса основываются на двух протоколах:

– Report protocol (протокол отчета);

– Boot protocol (загрузочный протокол).

Протокол отчета основан на отправке запросов и последующих отчетов со стороны устройства об изменении статуса, например, о нажатии клавиши или движении контроллера. Запрос со стороны хоста может содержать команду, направленную на изменение статуса устройства, например, установка диодов на клавиатуре.

Загрузочный протокол является более простым и определен для мыши и клавиатуры.

Для предотвращения сложностей в простых взаимодействиях, таких как в BIOS, загрузчиках операционных систем, а также драйверах одиночных клавиш, стандарт USB HID указывает, что клавиатуры и мыши могут объявлять себя «загрузочными» (boot-capable). Это значит, что такие клавиатуры и мыши могут быть переключены на более простой Boot protocol.

Переключение протокола можно осуществить с помощью специального запроса SetProtocol, который поддерживается только «загрузочными» устройствами [2].

1.4 Обработка нажатий клавиатуры

Так как подавляющее большинство HID устройств является boot-capable, во избежание чрезмерного усложнения задачи, в рамках данной работы будет рассмотрена реализация драйвера для USB HID Boot Protocol клавиатуры.

Клавиатуры, работающие на Boot Protocol, возвращают отчет о состоянии нажатия клавиш в упрощенном формате.

Для получения отчета достаточно отправлять запрос на входную точку клавиатуры посредством передачи прерывания.

Размер отчета, получаемого в ответ на запрос, составляет восемь байт, хотя некоторые клавиатуры могут предоставлять отчет меньшей длины, указанный в поле максимального размера пакета опрашиваемой конечной точки.

Таблица 1.1 – Структура отчета клавиатуры

Позиция	Тип	Описание
0	Байт	Статус клавиш-модификаторов
1	Байт	Зарезервирован
2	Байт	Нажатие 1
3	Байт	Нажатие 2
4	Байт	Нажатие 3
5	Байт	Нажатие 4
6	Байт	Нажатие 5
7	Байт	Нажатие 6

Из таблицы 1.1 видно, что структура представляет собой набор байт, где первый байт указывает статус клавиш-модификаторов, второй байт зарезервирован и может использоваться устройством для реализации собственной функциональности, а оставшиеся байты представляют собой очередь нажатых клавиш.

В ряде случаев, при использовании USB клавиатуры можно достичь состояния, называемое «фантомным состоянием». Данное состояние достигается при одновременном нажатии клавиш в количестве, превышающем размер очереди нажатий в отчете клавиатуры. В таком случае буфер нажатий клавиатуры переполняется, в отчет помещается сканкод 0x1, и драйвер клавиатуры не вопроизводит ни одной клавиши.

Как можно заметить из таблицы 1.1, в отличие от регулярных клавиш, статус нажатия клавиш-модификаторов представлен отдельным байтом, представление которого отражено в таблице 1.2. Данный факт можно объяснить тем, что клавиши-модификаторы как правило используются для *модификации* нажатий, и могут длительное время находится в активном состоянии.

Таблица 1.2 – Структура битового поля клавиш-модификаторов

Бит	Описание
0	Левый Ctrl
1	Левый Shift
2	Левый Alt
3	Левый Super
4	Правый Ctrl
5	Правый Shift
6	Правый Alt
7	Правый Super

По предыдущим таблицам несложно заметить, что отчет содержит в себе лишь события, означающие нажатие клавиш. Для того, чтобы зарегистрировать их отпускание, драйвер должен хранить предыдущий отчет и посредством сравнения очередей определить, какая клавиша перестала быть зажатой.

1.5 Управление светодиодами клавиатуры

Помимо регистрации нажатий клавиш и передачи их подсистеме ввода операционной системы, любой полноценный драйвер клавиатуры должен уметь регистрировать изменение светодиодов, привязанных к состоянию нажатия различных клавиш фиксации.

Если в аппаратном обеспечении клавиатуры определена поддержка дополнительной выходной конечной точки, состояние LED клавиатуры может быть установлено посредством передачи прерывания с полезной нагрузкой размером в один байт, определяющий состояние светодиодов клавиатуры.

Однако чаще всего такая конечная точка отсутствует, и для изменения состояния светодоидов необходимо использовать запрос SetReport

используя стандартную транзакцию Setup с однобайтовым каскадом данных. Тип Setup-запроса пакета должен быть равен 0x21, код запроса – 0x09. Младший байт wValue запроса должен равняться нулю, а старший байт – значению 0x02 для индикации направления запроса [3].

В таблице 1.3 представлена структура байта, передаваемого для установки значения светодиодов клавиатуры.

Таблица 1.3 – Структура битового поля светодиодов клавиатуры

Бит	Описание
0	Num Lock
1	Caps Lock
2	Scroll Lock
3	Compose
4	Kana
5 - 7	Зарезервированы, равны нулю

Если значение бита соответствующего светодиода установлено в единицу, лампочка будет переведена в активное состояние.

Так как многие стандартные USB клавиатуры не имеют специальной выходной конечной точки для изменения состояния LED клавиатуры, реализуемый драйвер будет поддерживать лишь клавиатуры с интерфейсом, содержащим единственную входную конечную точку, и использовать Setup-транзакцию для изменения состояния светодиодов.

1.6 Драйвер клавиатуры

В предыдущих разделах неоднократно упоминалось слово «драйвер». Несмотря на распространенность данного термина, а также общее определение, данное во введении, следует уточнить, что подразумевается под этим термином в данной работе.

Под *драйвером устройства* понимается отдельно загружаемый компонент ядра, называемый *модулем* [1)], предоставляющий операционной системе доступ к аппаратному обеспечению данного устройства.

Важно заметить, что под драйвером ядра понимается именно компонент ядра, т.е. системный код, исполняемый в пространстве ядра, так

[1)] В экосистеме Linux под модулем ядра понимают как независимо загружаемую подсистему ядра, так и всякий драйвер устройства, работающий в пространстве ядра.

как существуют так называемые драйверы-фильтры, которые могут работать с устройством из пользовательского пространства, используя для этого функционал аппаратного драйвера.

При разработке модуля ядра необходимы знания об устройстве операционной системы, под которую разрабатывается этот модуль, а также общие принципы и механизмы, используемые во всех операционных системах.

В рамках теоретических сведений будут описаны только те подсистемы Linux, которые необходимы для реализации функционала устройства клавиатуры.

Сведения, относящиеся к общим положениям организации аппаратного обеспечения и операционных систем будут либо упомянуты в разделах, непосредственно связанных с описанием реализации драйвера, либо опущены как не требующие подробного представления для понимания принципов работы кода.

Подробное изложение всех теоретических сведений, используемых при разработке модулей USB устройств, может быть найдено как в определенных разделах документации ядра Linux, поддерживаемой сообществом, так и в соответствующей узкоспециализированной литературе, посвященной данной тематике [4].

1.7 Подсистема USB Linux

USB Core – это подсистема ядра Linux, созданная для поддержки USB-устройств и контроллеров USB-шины. Данная подсистема определяет набор структур, макросов и функций, позволяя разработчикам системного кода абстрагироваться от аппаратной реализации USB и относящихся к ней аппаратно-зависимых функций.

Помимо реализации через различные компоненты ядра таких механизмов как «горячее подключение» устройств с последующим запуском подходящего модуля, подсистема предоставляет разработчикам единообразный программный интерфейс для разработки переносимых драйверов устройств.

Определение группы устройств, которую может обслуживать драйвер, а также функций, используемых в качестве обработчиков стандартных событий (например, подключение и отключение устройства), осуществляются через определение специальной структуры.

Сообщение между хостом и устройством осуществляется при помощи *URB (USB Request Block)*, блоков запросов, регистрируемых в подсистеме с возможностью повторного использования.

Многие аппаратно-зависимые функции, начиная с настройки устройства, заканчивая созданием когерентной памяти для совместного доступа со стороны процессора и устройства, также реализованы в виде специальных функций [5].

1.8 Подсистема ввода Linux

В то время как подсистема универсальной последовательной шины предоставляет возможность взаимодействия между драйвером и устройством, подсистема ввода Linux позволяет драйверу устройства ввода реализовать передачу сведений в пользовательское пространство в виде манипуляций ввода.

На рисунке 1.2 изображена диаграмма взаимодействия подсистемы ввода Linux с другими модулями ядра.

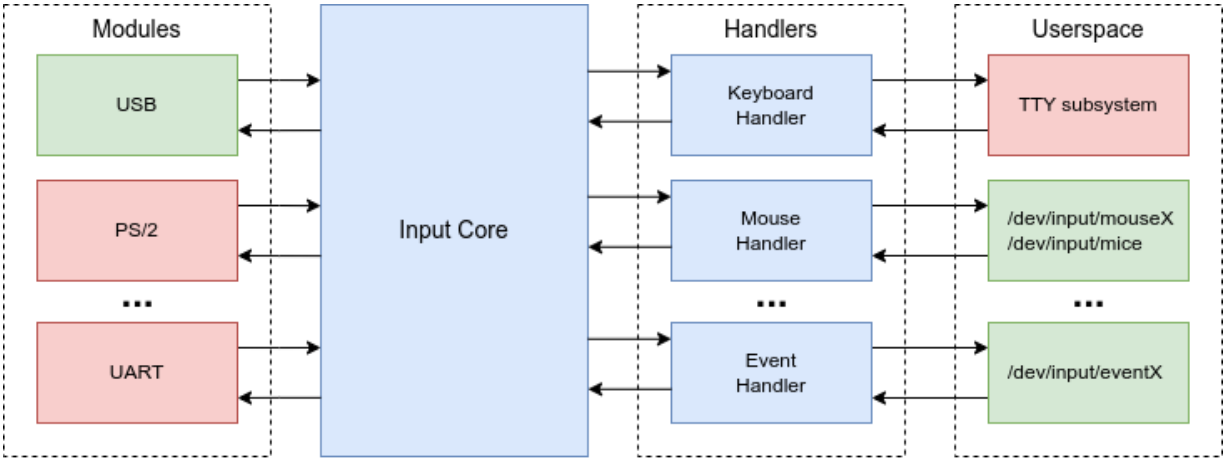


Рисунок 1.2 – Подсистема ввода Linux

Из представленной выше диаграммы видно, что Input Core является таким же посредником, как и подсистема USB: для того, чтобы модуль устройства мог функционировать как устройство ввода, достаточно зарегистрировать специальную структуру, описывающую данное устройство ввода, и использовать функции сообщения от имени этого устройства к подсистеме ввода. Последующее взаимодействие с подсистемой сводится к сообщению о возникновении нового события ввода, работа по регистрации и обработке этих событий лежит на самой подсистеме ввода и разнообразных обработчиках событий [6].

Как и в случае с подсистемой универсальной последовательной шины, подсистема ввода предоставляет специальную структуру, определяющую устройство ввода, набор функций для его регистрации, а также функции для сообщения подсистеме ввода новых событий ввода.

Стоит отметить, что несмотря на большое количество функционала, делигированного подсистеме ввода, модуль клавиатуры все еще должен заниматься некоторыми преобразованиями данных, получаемых от аппаратной части устройства, для последующей передачи в подсистему ввода. Это связано с тем, что Linux и Windows на уровне ядра работают с так называемыми raw-сканкодами, в то время как Boot Protocol клавиатуры возвращает USB-коды клавиш [7]. Для преобразования кодов в «сырые» сканкоды может быть использована специальная таблица [8].

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

С точки зрения архитектуры ядра Linux, модуль ядра, драйвер, является специальным объектным файлом, динамически загружаемым в ядро для расширение его функциональности.

При написании драйверов используется подход размещения кода в пределах одной единицы трансляции. Данный факт объясняется тем, что, как правило, модули ядра разрабатываются под одну конкретную часть системы, а потому предоставляемый разработчику программный интерфейс позволяет написать понятный, имеющий типизированную структуру код. Кроме того, подобный подход во многом упрощает сборку модуля, выполняемую с помощью системы Kbuild [9].

Несмотря на вышесказанное, функции, из которых состоит модуль ядра, всегда можно разделить на различные функциональные группы, выполняющие условно изолированные друг от друга задачи.

Драйвер клавиатуры состоит из следующих групп функций:

– группа вспомогательных функций;
– группа контрольных событий;
– группа управления нажатиями клавиш;
– группа управления состоянием светодиодов.

Разграничение на группы, представленное выше, не основывается на взаимодействии с той или иной подсистемой, а осуществляется посредством разделения функций драйвера как абстракции над реализацией: обратный вызов подсистемы ввода окажется в одной группе с обратным вызовом подсистемы шины, если оба обратных вызова отвечают за одну и ту же часть, например, за состояние светодиодов клавиатуры, и наоборот, два обратных вызова из одной подсистемы окажутся в различных группах, если они обслуживают разные части.

2.1 Группа вспомогательных функций

Данная группа функций отвечает за инициализацию и деинициализацию полей структуры драйвера `usbkeyboard`.

2.2 Группа контрольных событий

Данная группа функций содержит в себе функции, отвечающие за обработку различных событий, влияющих на состояние драйвера.

К этой группе относятся:

– обратные вызовы подсистемы ввода: открытие и закрытие устройства ввода;

– обратные вызовы «горячего» подключения: подключение и отключение устройства.

2.3 Группа управления нажатиями клавиш

Данная группа функций выполняет обработку обратного вызова для запроса состояния нажатия клавиш с последующей передачей сведений об изменении состояния в подсистему ввода.

2.4 Группа управления состоянием светодиодов

Данная группа функций отвечает за установку состояния светодиодов на клавиатуре, зависящего от подсистемы ввода.

Подход к изменению состояния светодиодов на клавиатуре как отдельном физическом устройстве через обработку событий подсистемы ввода имеет одну очень интересную особенность: поскольку события об изменении состояния светодиодов поступают не от устройства ввода, «связанного» с конкретным физическим устройством в контексте драйвера, а вообще от любого устройства ввода, зарегистрированного в подсистеме, то и нажатие Lock-клавиши влияет на состояние всех устройств, что используют данный подход, даже если это это событие было вызвано программно, например, со стороны пользовательского пространства [10].

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Ниже будут представлены результаты проектирования каждой функциональной группы, описанной в разделе 2.

3.1 Группа вспомогательных функций

Ниже представлено объявление функции инициализации памяти для структуры данных `usbkeyboard`:

```
int usbkeyboard_alloc_memory(struct usb_device* dev,
                             struct usbkeyboard* kbd);
```

Данная функция инициализирует все динамические поля структуры, связанные с подсистемой USB (память для URB и DMA).

Ниже представлено объявление функции деинициализации памяти для структуры данных `usbkeyboard`:

```
void usbkeyboard_free_memory(struct usb_device* dev,
                             struct usbkeyboard* kbd);
```

Данная функция выполняет работу, противоположную работе функции инициализации `usbkeyboard_alloc_memory`.

3.2 Группа контрольных событий

Ниже представлено объявление функции инициализации драйвера при подключении устройства:

```
int usbkeyboard_probe(struct usb_interface* intf,
                      const struct usb_device_id* id);
```

Данная функция выполняет все действия, необходимые для регистрации устройства клавиатуры в системе и его последующего обслуживания драйвером.

Ниже представлено объявление функции деинициализации драйвера при отключении устройства:

```
void usbkeyboard_disconnect(struct usb_interface *intf);
```

Данная функция выполняет работу, противоположную работе функции инициализации драйвера `usbkeyboard_probe`.

Ниже представлено объявление функции запуска обработки нажатий клавиатуры при открытии устройства ввода:

```
int usbkeyboard_open(struct input_dev* dev);
```

Данная функция регистрирует обработку нажатия клавиш клавиатуры при открытии соответствующего ей устройства ввода.

Ниже представлено объявление функции остановки обработки нажатий клавиатуры при закрытии устройства ввода:

```
void usbkeyboard_close(struct input_dev* dev);
```

Данная функция выполняет работу, противоположную работе функции запуска обработки нажатий клавиатуры `usbkeyboard_open`.

3.3 Группа управления нажатиями клавиш

Ниже представлено объявление функции обработки отчетов о нажатии клавиш клавиатуры:

```
void usbkeyboard_irq(struct urb* urb);
```

Данная функция сравнивает предыдущее состояние контекста `urb`, и, на основании этого сравнения, сообщает подсистеме ввода о новых нажатиях и отпусканиях клавиш, если они произошли.

3.4 Группа управления состоянием светодиодов

Ниже представлено объявление функции управления состоянием светодиодов на клавиатуре:

```
void usbkeyboard_led(struct urb* urb);
```

Данная функция отправляет устройству клавиатуры упрвавляющий запрос на изменение состояния светодиодов.

Ниже представлено объявление функции обработки события изменения состояния светодиодов в подсистеме ввода:

```
int usbkeyboard_event(struct input_dev* dev,
                      unsigned int type,
                      unsigned int code, int value);
```

Данная функция преобразует сведения о событии в управляющий запрос на изменение состояния светодиодов клавиатуры и регистрирует его отправление.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

Как уже было отмечено в разделе 2, архитектура драйвера, реализованного в рамках данной работы, не предполагает модульной архитектуры, а потому в качестве модулей рассматриваются группы функций, выполняющих схожие задачи.

4.1 Разработка группы вспомогательных функций

В подразделе 3.1 упоминалось, что функции из данной группы отвечают за выделение и освобождение памяти полей структуры `usbkeyboard`, поэтому, для дальнейшего обсуждения их реализации, необходимо привести ее определение:

```
struct usbkeyboard {
    struct input_dev* dev;
    struct usb_device* usbdev;

    struct urb* irq;
    struct urb* led;

    struct usb_ctrlrequest* creq;

    char name[KBD_NAME_MAX];
    char phys[KBD_PHYS_MAX];

    unsigned char prev_presses[8];
    unsigned char* presses;

    unsigned char curr_leds;
    unsigned char* leds;

    dma_addr_t presses_dma;
    dma_addr_t leds_dma;

    bool leds_urb_submitted;
    spinlock_t leds_lock;
};
```

Поля `dev` и `usbdev` выполняют указание на системные структуры устройства ввода и устройства последовательной шины.

В свою очередь `irq` и `led` указывают на URB, первый из которых отвечает за получение состояния нажатия клавиш, а второй – за установку состояния светодиодов.

Символические массивы `name` и `phys` хранят название устройства и его физический путь (в представлении топологии шины).

Массив `prev_presses` хранит предыдущее состояние нажатий, а указатель `presses` ссылается на память, используемую для хранения получаемого от устройства состояния нажатий.

Поля `curr_leds` и `leds` имеют назначение, аналогичное предыдущим, за тем лишь исключением, что размер переменной предыдущего состояния и размер памяти текущего состояния составляет один байт, что соответствует сведениям, представленным в таблице 1.3.

Самыми интересными с точки зрения взаимодействия с памятью являются поля `presses_dma` и `leds_dma`. Данные поля хранят адрес прямого доступа к памяти (DMA), механизм которого используется для совместного доступа к памяти как со стороны процессора, так и со стороны самого устройства. Выделение памяти для `presses` и `leds` осуществляется при помощи функции `usb_alloc_coherent` [1)], одним из аргументов которой является `dma_addr_t*`. В результате вызова функции возвращается указатель на буфер в CPU-пространства, согласованный с DMA, а по адресу `dma_addr_t*` возвращается DMA-адрес буфера. Чаще всего данный подход используется для того, чтобы устройство на шине могло записывать данные в оперативную память напрямую, не прерывая вычисления процессора. Подробнее о технологии прямого доступа к памяти, а также о предоставляемых операционной системой Linux возможностях, можно прочитать в источниках [4, 11].

Поле `creq` содержит в себе структуру управляющего запроса, используемого для отправки запроса на изменение состояния светодиодов устройства клавиатуры.

Оставшиеся поля `leds_usb_submitted` и `leds_lock` отвечают за синхронизацию группы управления состоянием светодиодов, и будут подробно рассмотрены в подразделе 4.4.

[1)] coherent, когерентный, в данном контексте означает «согласованный», предоставляющий совместный доступ.

4.2 Разработка группы контрольных событий

Функции данной группы отвечают за начальную инициализацию данных драйвера для нового интерфейса устройства, и инициализируют некоторые поля структуры `usbkeyboard`, не относящиеся к чистому выделению памяти, например, поля `dev`, `usbdev`, получаемые путем приведения системной структуры интерфейса и регистрации устройства ввода, а также `name`, `phys`, `leds_urb_submitted` и `leds_lock`, требующих инициализации и не требующих выделения памяти для поля структуры.

Как уже было упомянуто ранее, данные для каждого устройства не хранятся в памяти самого драйвера в виде массива структур `usbkeyboard`, а передаются в виде контекста `urb` и данных `input_dev`, поэтому предыдущая группа функций называется *вспомогательной*, а функции из данной группы можно считать функциями инициализации и деинициализации данных драйвера для устройства.

Функции `usbkeyboard_open` и `usbkeyboard_close` обрабатывают события открытия и закрытия устройства ввода, и отвечают за регистрацию и прекращение обработки нажатий клавиш клавиатуры, используя функции `usb_submit_urb` и `usb_kill_urb` соответственно.

4.3 Разработка группы управления нажатиями клавиш

Данная группа содержит единственную функцию обратного вызова `usbkeyboard_irq`, которая вызывается при возвращении устройством состояния нажатий клавиш. Как уже было сказано в теоретической части, для определения нажатых и отпущенных клавиш очередь текущих нажатий сравнивается с очередью предыдущих нажатий посредством поиска вхождения, а байт состояния клавиш-модификаторов сравнивается простыми побитовыми операциями.

Подсистема ввода уведомляется об изменении статуса клавиши посредством вызова системной функции `input_report_key`.

Для приведения usb-кодов в raw-коды используется таблица преобразования, представленная массивом `usbkeyboard_keycodes`.

В случае возникновения ошибки, не связанной с состоянием передачи, функция выполняет повторную регистрацию `irq`.

Блок-схема алгоритма функции обработчика статуса нажатий клавиатуры представлена в приложении А.

4.4 Разработка группы управления состоянием светодиодов

Данная группа представлена функциями двумя функциями и представляет особый интерес с точки зрения технической реализации.

Дело в том, что согласно спецификации Linux USB API [5], статус выполнения URB является действительным лишь тогда, когда был совершен обратный вызов, т.е. тогда, когда блок запроса был выполнен.

Ожидаемое от статуса URB поведение не выполняется в ситуации согласования обратных вызовов `usbkeyboard_led` и `usbkeyboard_event`, необходимого для совместного использования данных о светодиодах, хранящихся в структуре `usbkeyboard` [12].

Решением данной проблемы является введение логической переменной `leds_urb_submitted`, положительное значение которой означает наличие активного управляющего запроса в данный момент, а также спинлоком `leds_lock`, используемого для входа в критические секции обратных вызовов с отключением обработки прерываний на время обслуживания критической секции кода.

Для входа в критическую секцию используется синхронизация на спинлоке с помощью макроса `spin_lock_irqsave`, для выхода из критической секции используется макром `spin_lock_irqrestore`. Данные макросы используются в паре для отключения обработки прерываний на время нахождения в критической секции с возможностью последующего восстановления.

Алгоритм синхронизации для функции `usbkeyboard_event`:

1. Вход в критическую секцию;
2. Если условие `leds_urb_submitted` истинно, выход из критической секции с завершением выполнения;
3. Если новое состояние совпадает с предыдущим, выход из критической секции с завершением выполнения;
4. Сохраняется текущее состояние;
5. Регистрируется запрос с новым состоянием;
6. Если не удалось зарегистрировать запрос, выход из критической секции с завершением выполнения;
7. Устанавливается истинное значение для `leds_urb_submitted`;
8. Выход из критической секции;
9. Конец алгоритма.

Алгоритм синхронизации для функции `usbkeyboard_led`:

1. Вход в критическую секцию;
2. Если новое состояние совпадает с предыдущим, устанавливается ложное значение для `leds_urb_submitted` и совершается выход из критической секции с завершением выполнения;
3. Сохраняется текущее состояние;
4. Регистрируется запрос с новым состоянием;
5. Если не удалось зарегистрировать запрос, устанавливается ложное значение для `leds_urb_submitted` и совершается выход из критической секции с завершением выполнения;
6. Выход из критической секции;
7. Конец алгоритма.

Как видно из представленных выше алгоритмов, оба обратных вызова используют спинлок для согласования доступа к данным о состоянии светодиодов клавиатуры, а логическая переменная используется для индикации состояния активного запроса на изменение этого состояния. Помимо этого, в функции `usbkeyboard_led` присутствует повторная отправка запроса в случае, когда данные, хранящиеся в буфере текущего состояния светодиодов, не совпадают с данными, восстановленными из контекста обработанного блока.

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для тестирования разработанного в рамках проекта драйвера клавиатуры необходимо:

1. Склонировать git-репозиторий с исходным кодом драйвера, расположенным по адресу: https://github.com/Qurcaivel/usbkeyboard.git.

2. Собрать исходный код проекта согласно инструкции, приведенной в файле описания проекта readme.md.

3. Протестировать полученный модуль с помощью одного из скриптов, расположенных в репозитории.

ЗАКЛЮЧЕНИЕ

В данном разделе будут подведены итоги по проделанной в рамках данного курсового проекта работы по написанию драйвера USB клавиатуры для ядра Linux.

В ходе выполнения проекты были получены множественные теоретические сведения, связанные с общими положениями системного программирования, некоторых аппаратных решений в современных вычислительных системах.

Помимо общих сведений, например, о системе прямого доступа к памяти или интерфейсе универсальной последовательной шины, были приобретены специфические теоретические знания, относящиеся к экосистеме ядра Linux, а также к непосредственной разработке модулей ядра для обслуживания периферийных устройств: принципы работы модулей ядра в системе, устройство подсистем ввода и USB, а также использование систем сборки Make и Kbuild.

Получены практические навыки в написании модулей ядра Linux, необходимых для работы подсоединяемых устройств. Также к приобретенным навыкам можно отнести событийное программирование для некоторых подсистем ядра, а также способы синхронизации памяти и устранение гонок данных при разработке многопоточного и асинхронного программного обеспечения.

Опыт разработки модулей ядра Linux во многом отличается от системного программирования, изучаемого в рамках теоретического курса предмета, и открывает множество возможностей по дальнейшему изучению тем, связанных с операционными системами и системным программированием, например, разработка системного программного обеспечения для серверов, или программирование микроконтроллеров и систем реального времени.

Опыт в разработке драйверов устройств, связанный с изучением технической документации, стандартов и соглашений, полезен для дальнейшей самостоятельной работы в узконаправленных предметных областях, возможности по изучению которых ограничиваются документацией и ресурсами соответствующего сообщества.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Universal Serial Bus 3.1 Specification, Revision 1.0. — Hewlett-Packard Company, Intel Corporation, Microsoft Corporation, Renesas Corporation, ST- Ericsson, and Texas Instruments, 2013.

[2] Device Class Definition for Human Interface Devices (HID): Firmware Specification — 5/27/01, Version 1.11. — USB Implementer's Forum, 2001.

[3] USB Human Interface Devices [Электронный ресурс]. — Дата доступа : 16.03.2023. — Режим доступа : https://wiki.osdev.org/USB_Human_Interface_Devices.

[4] Corbet, Jonathan. Linux Device Drivers, 3rd Edition / Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman. — O'Reilly Media, Inc., 2005.

[5] Linux USB API [Электронный ресурс]. — Дата доступа : 01.04.2023. — Режим доступа : https://www.kernel.org/doc/html/v4.15/driver-api/usb/index.html.

[6] Input Subsystem [Электронный ресурс]. — Дата доступа : 01.04.2023. — Режим доступа : https://www.kernel.org/doc/html/v4.17/driver-api/input.html.

[7] Scancode - Deskthority wiki [Электронный ресурс]. — Дата доступа : 25.04.2023. — Режим доступа : https://deskthority.net/wiki/Scancode.

[8] Table: keyboard-internal scancodes [Электронный ресурс]. — Дата доступа : 16.03.2023. — Режим доступа : https://www.win.tue.nl/~aeb/linux/kbd/scancodes-10.html#ss10.6.

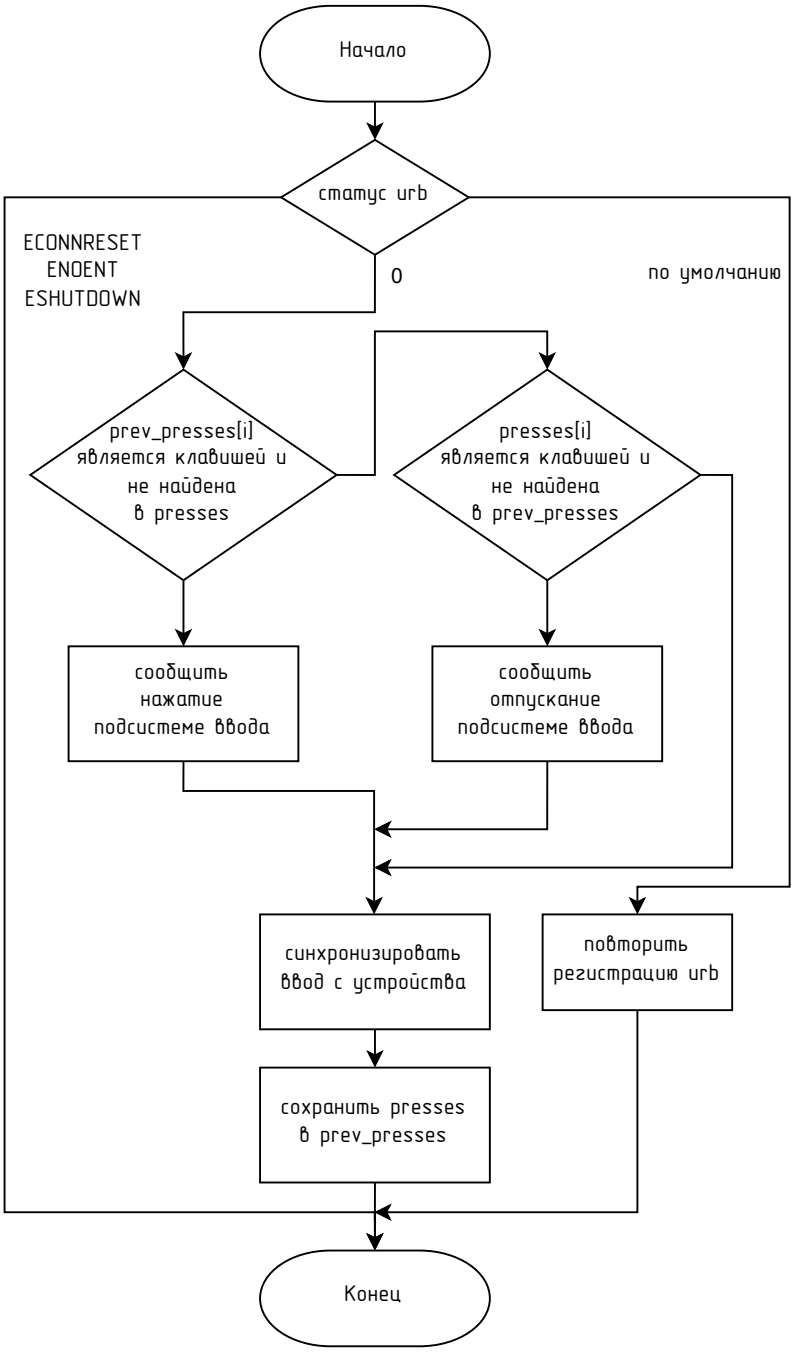
[9] Building External Modules [Электронный ресурс]. — Дата доступа : 02.03.2023. — Режим доступа : https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/Documentation/kbuild/modules.rst.

[10] Rubini, Alessandro. Usb Device Drivers [Электронный ресурс]. — Дата доступа : 12.04.2023. — Режим доступа : https://www.linux.it/~rubini/docs/usb/usb.html.

[11] Dynamic DMA Mapping Guide [Электронный ресурс]. — Дата доступа : 02.05.2023. — Режим доступа : https://docs.kernel.org/core-api/dma-api-howto.html.

[12] Sound Formal Verification of Linux's USB BP Keyboard Driver / Willem Penninckx [et al.] // Proceedings of the 4th International Conference on NASA Formal Methods. — NFM'12. — Berlin, Heidelberg: Springer-Verlag, 2012. — 210–215 P.

ПРИЛОЖЕНИЕ А
(обязательное)
Блок-схема алгоритма



Изм	Лист	№ документа	Подпись	Дата
Разраб		Кутняк		
Пров		Басак		

ГУИР.400201.001 ПД

Драйвер USB клавиатуры
Блок-схема функции
usbkeyboard_irq

Лит.	Масса	Масштаб
Т		

Лист 1	Листов 1

ЭВМ, гр. 150503

ПРИЛОЖЕНИЕ Б
(обязательное)
Ведомость документов

Обозначение	Наименование	Примечание
	Графические документы	
ГУИР.400201.001 ПД	Блок-схема алгоритма	Формат А4
	Текстовые документы	
БГУИР КП 1-40 02 01 310 ПЗ	Пояснительная записка	32 с.

Изм	Лист	№ документа	Подпись	Дата	ГУИР КП 1-40 02 01 310 Д1		
Разраб.		Кутняк			Драйвер USB клавиатуры Ведомость документов	Лит.	Лист	Листов
Пров.		Басак				Т		
						Кафедра ЭВМ гр. 150503		