Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ: Информатика и системы управления
КАФЕДРА: Программное обеспечение ЭВМ и информационные технологии

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе на тему:

Разработка программного обеспечения для управления мышью с помощью Android-устройства для операционной системы "Linux".

Руководитель курсовой работы _____ /Оленев А.А./

Студент _____ /Кукуев С.А./

Москва, 2016.

Содержание

Введение

Сегодня сложно представить себе работу с персональным компьютером без привычного каждому из нас устройства - мыши. С момента создания этот девайс претерпел колоссальные изменения. От обычных механических мышей, с которыми было неудобно работать в виду тяжелого веса и плохого позиционирования, мир перешел к оптическим, отличавшимся легкостью, надежностью и высокой точностью. Несмотря на это, метаморфозы устройства продолжаются. В настоящее время появились гироскопические мыши[1], позволяющее распознавать движение не только на поверхности, но и в пространстве. Для тех, кто использует большие плазменные экраны или проекторы, такое новшество позволяет управлять компьютером в качестве пульта, не задействуя посторонние предметы. Но стоимость таких мышей составляет порядка 4-5 тысячи рублей, что в свою очередь является дорогим удовольствием.

Данное программное обеспечение позволяет использовать Android-устройство в качестве беспроводной мыши и выполнять соответствующую функциональность. В процессе реализации должны быть решены следующие задачи:

— Задержка обработки данных. Программное обеспечение обрабатывает данные в реальном времени. При таком подходе важно учесть скорость передачи данных и их последующую обработку. Поскольку передается небольшой объем данных, со скоростью не должно возникнуть проблем, а вот слишком большая задержка увеличит дергание курсора.

— Дрожание мыши. Для реализации используются датчики Android-устройства, погрешность которых достигает порядка 5° (в зависимости от выбора датчика). Помимо того, использование акселерометра или гироскопа добавляет внешний фактор - дрожание человеческой руки. Все это в совокупности вызывает дребезжание курсора, которое необходимо минимизировать.

— Функциональность. Под функциональностью понимается минимальный набор требуемых функций, необходимых для замены базовой мыши. В данном случае будет достаточно поддержки двух осей вращения и двух кнопок.

1 Аналитический раздел

Перед проведением анализа необходимо разбить реализацию на не зависимые части. Для каждой предстоит выбрать алгоритм, удовлетворяющий поставленным задачам. Проанализировав возможные методы реализации были выделены следующие "подсистемы":

— Способы управления курсором.
— Передача данных с Android-устройства.
— Обработка полученных данных.
— Взаимодействие с ядром.

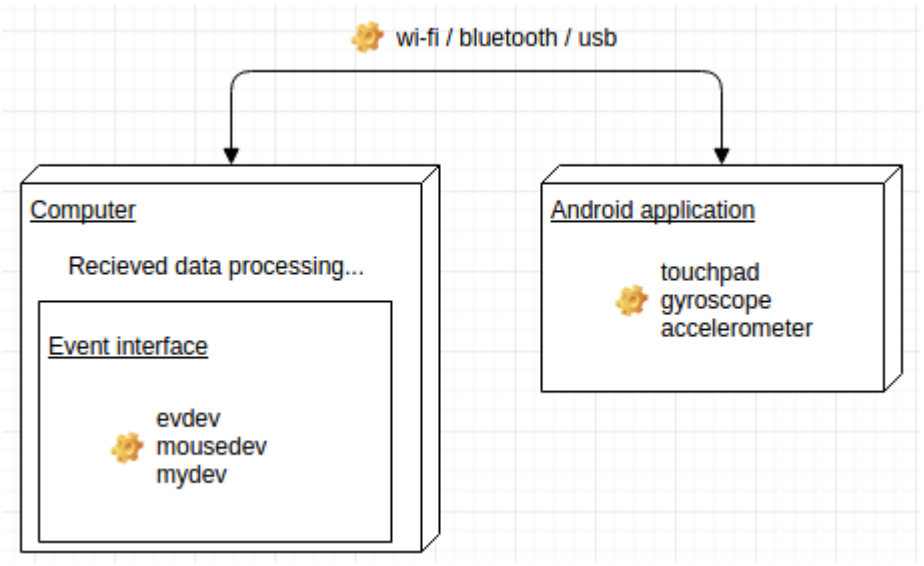

Рисунок 1.1 — Общая структура.

1.1 Выбор способа управления

Для выбора способа управления необходимо отталкиваться от поставленных задач. Самый распространенный вариант - имитация сенсорной панели, на первый взгляд выглядит очень привлекательно. При такой реализации дрожание курсора зависит лишь от задержки обработки данных. С другой стороны присутствие в Android-устройствах таких датчиков[2], как G-сенсор и гироскоп, дают дополнительную степень свободы, что дает большую функциональность.

Гироскоп (гиродатчик) - устройство, позволяющее определить ориентацию в пространстве, используя гравитацию Земли. По количеству степеней свободы различают: двухстепенные и трехстепенные. На Android устройствах на сегодняшний день установлены вибрачионные микрогироскопы, принцип действия которых основан на силе Кориолиса.

Акселерометр (G-сенсор) - устройство, предназначенное для измерения негравитационного ускорения. Различают три типа: однокомпонентные, двухкомпонентные и трехкомпонентные. Чтобы определить ориентацию в пространстве, необходимо данные измерения проинтегрировать, получая инерциальную скорость и координаты устройства.

Основным из преимуществ гироскопа является то, что акселерометр не в состоянии отличить гравитационное ускорение от любого другого. Находясь в системе отсчета, которая ускорено движется в определенном направлении, данный датчик будет показывать некорректные результаты. Гироскопы в свою очередь ориентируются лишь на гравитацию и не зависят от внешних факторов, поэтому и были выбраны в качестве способа управления.

1.2 Методы передачи данных

Взаимодействие Android-устройства и персонального компьютера является необходимой частью реализации, поэтому необходимо выбрать один из существующих методов. Основными факторами выбора будут: радиус действия, скорость передачи и сложность настройки. Вариант с USB 3.0, не смотря на высокую скорость, не подходит из-за наличия проводной передачи данных.

Bluetooth и wi-fi[3] являются беспроводными стандартами связи и используют для передачи данных определенные радиочастоты. Чтобы определиться с выбором метода, составим таблицу из самых распространенных стандартов с данными о радиусах действия и максимальных скоростях передач.

Таблица 1.1 — Сравнение стандартов

Стандарт	Дистанция(в помещении), m	Скорость, Mbps
Wi-fi		
IEEE 802.11.g	38	54
IEEE 802.11.n	70	600
Bluetooth		
Bluetooth 3.0	10	24
Bluetooth 4.0	60	30

В таблице 1.1 наглядно видно, что wi-fi обладает лучшими характеристиками. Но для передачи информации он требует тщательной настройки параметров, а также обязательного наличия третьего звена - точки-маршрутизатора. Возможен вариант соединения ad-hoc, который позволяет соединяться напрямую, но при этом станет невозможно подключить другие устройства. Bluetooth же не требует никакого дополнительного конфигурирования и позволяет подключить несколько устройств

одновременно. Для передачи координат с Android-устройства хватает скорости передачи и bluetooth, поэтому он выбран для реализации передачи информации.

1.3 Интерфейсы событий

В операционной системе Linux для решения поставленной задачи существует специальная подсистема ввода ядра[4], которая объединяет все рассеянные драйвера для обработки перифирийных устройств. Одним из ее плюсов является удобный интерфейс событий, позволяющий драйверу не создавать и не управлять узлами /dev и связанными с ними методов доступа. Вместо этого он имеет возможность вызывать API для ввода, чтобы отправить движения мыши. Такие приложения, как X Window, хорошо взаимодействуют с интерфейсами событий, экспортируемыми подсистемой ввода.

Чтобы выбрать один из возможных интерфейсов, необходимо обозначить требования, которым он должен соответствовать, и ознакомиться со всеми доступными вариантами. Исходя из задач, поставленных в рамках реализации, основным требованием к интерфейсу является необходимая функциональность. Кроме движения мыши, драйвер событий должен предоставлять возможность нажатия на кнопки мыши. Рассмотрим три возможных варианта: evdev, mousedev и собственный интерфейс.

«`Evdev`» - универсальный драйвер событий ввода. Каждый пакет события, создаваемый «`evdev`», имеет строго определенный формат и содержит в себе следующие параметры: время, тип события, код события, значение события. Одно из преимуществ интерфейса является простота использования и универсальность. Кроме того, он имеет возможность работать с акселерометром, что упростит задачу преобразования данных, и оснащен минимальным требуемым функционалом для решения задачи.

«`Mousedev`» - специальный драйвер событий ввода для мыши. В отличие от универсального evdev, этот интерфейс предназначен лишь для одного перефирийного устройства, и не умеет взаимодействовать с такими датчиками, как гироскоп и акселерометр. Правда и функций у драйвера намного больше.

Помимо готовых вариантов, имеется возможность написать собственный драйвер событий. Является более гибким вариантом, чем два, представленных выше, поскольку мы сможем постепенно наращивать функционал.

Для решения поставленной задачи целесообразней использовать универсальный драйвер событий ввода. Поскольку он оснащен требуемой функциональностью, а простота использование является неоспоримым аргументом в пользу его выбора.

2 Конструкторский раздел

2.1 Структура разрабатываемого программного обеспечения

Для создания разрабатываемого программного обеспечения необходимо реализовать:

— Загружаемый модуль ядра.
— Пользовательское приложение.

Согласно поставленной задачи, пользователь должен осуществлять управление мышью через Android устройство удаленно, поэтому пользовательское приложение необходимо реализовать в виде клиент-серверного приложения. Таким образом, пользовательское приложение будет состоять из двух частей: программы сервера и программы клиента.

— Клиент - приложение, разработанное для Android устройства, которое предоставляет ползователю осуществлять управление мышью.
— Сервер - приложение, работающее на ПК в фоновом режиме, отвечающее за получение данных от клиентского приложения и передачу их загружаемому модулю ядра.

Передача данных между смартфоном и ПК будет осуществляться с помощью простокола sdp(Service Discovery Protocol)[5]. В связи с этим структура программного обеспечения будет выглядеть, как на рисунке 2.1.

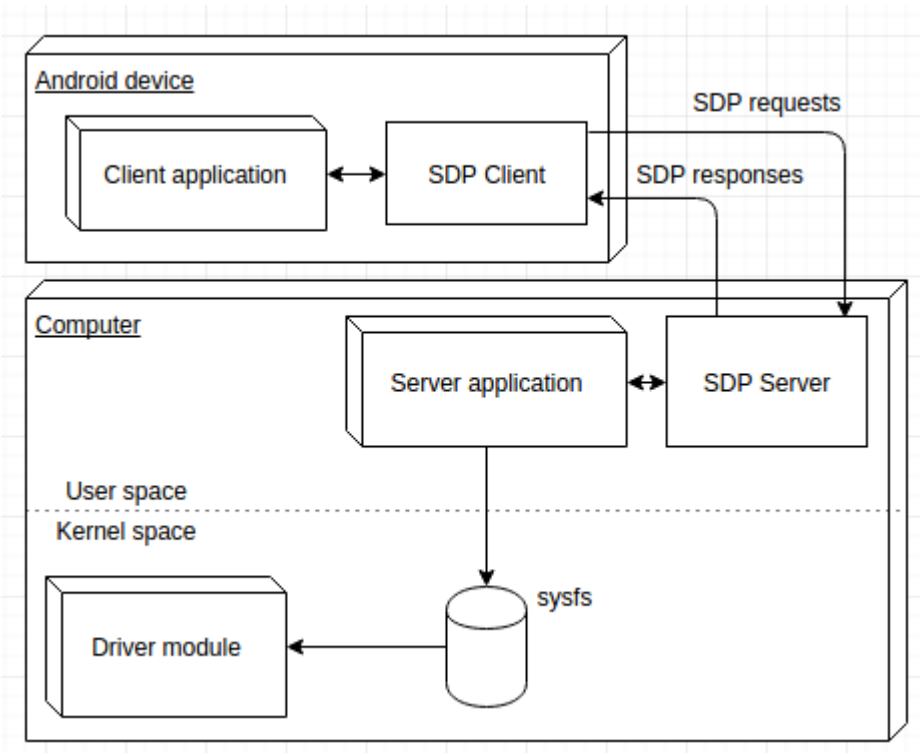

Рисунок 2.1 — Структура программного обеспечения

2.2 Загружаемый модуль ядра

Загружаемый модуль ядра должен решать следующие задачи:

— Чтение данных из пространства ядра.
— Регистрация устройства в подсистеме ввода ядра.

Обмен данными между пространством ядра и пространством пользователя происходит при помощи виртуальной файловой системы в ОС Linux - `sysfs` [4]. В `Sysfs` имеется подкаталог, где содержится вся информация о перифирийных подключаемых устройствах, присущих конкретной платформе. Все, что необходимо для обмена: создать в этом самом подкаталоге `/sys/devices/platform`, каталог с файлом, в который сервер будет записывать поступающие данные от Android приложения. Данный каталог создается командой **command_result = sysfs_create_group(&vms_dev->dev.kobj, &vms_attr_group);**.

При обмене информацией между необходимо определить содержание и формат передаваемых данных. Для корректной работы мыши, загружаемый модуль должен получать текущие координаты положения курсора и тип команды. Передаваемые данные запишем в виде строки, состоящей из 4 чисел:

— Команда
— Координата курсора мыши по оси X
— Координата курсора мыши по оси Y
— Координата курсора мыши по оси Z

Также, необходимо сразу уточнить, какие команды могут подаваться от устройства:

— Перемещение курсора
— Нажатие левой клавиши мыши
— Нажатие правой клавиши мыши
— Двойной щелчок левой клавиши мыши

Регистрация устройства происходит по следующим этапам:

а) Регистрация платформо зависимого устройства в системе.
б) Создание файла устройства в **sysfs**.
в) Выделяем памяти под устройство ввода.
г) Установка реакций на события.
д) Регистрация устройства в подсистеме ввода.

Листинг 2.1 — Алгоритм регистрации устройства в системе

```
1  def display_init(void):
2      command_result = 0;
3
4      vms_dev ← platform_device_register_simple("vms", -1, NULL, 0)
5      if (IS_ERR(vms_dev))
6          PTR_ERR(vms_dev)
7          printk("vms_init: error\n")
8          return ERROR_REGISTER_PLATFORM_DEVICE
9
10     command_result ← sysfs_create_group(vms_dev→dev.kobj,
           vms_attr_group);
11
12     if (command_result < 0)
13         printk("Error sysfs_create_group\n")
14         return ERROR_SYSFS_CREATE_GROUP
15
16     vms_input_dev ← input_allocate_device()
17     if (!vms_input_dev)
18         printk("Bad input_alloc_device()\n")
19         return ERROR_ALLOCATE_INPUT_DEVICE
20
21     set_bit(EV_REL, vms_input_dev→evbit)
22     set_bit(REL_X, vms_input_dev→relbit)
23     set_bit(REL_Y, vms_input_dev→relbit)
24     set_bit(EV_KEY, vms_input_dev→evbit)
25     set_bit(BTN_LEFT, vms_input_dev→keybit)
26     set_bit(BTN_RIGHT, vms_input_dev→keybit)
27
28     vms_input_dev→evbit[0] ← BIT_MASK(EV_KEY) | BIT_MASK(EV_REL)
29     vms_input_dev→keybit[BIT_WORD(BTN_MOUSE)] ← BIT_MASK(BTN_LEFT)
           |
30         BIT_MASK(BTN_MIDDLE) | BIT_MASK(BTN_RIGHT)
31     vms_input_dev→relbit[0] ← BIT_MASK(REL_X) | BIT_MASK(REL_Y)
32
33     vms_input_dev→name ← "Virtual BT mouse"
34     vms_input_dev→id.bustype ← BUS_VIRTUAL
35     vms_input_dev→id.vendor  ← 0x0000
36     vms_input_dev→id.product ← 0x0000
37     vms_input_dev→id.version ← 0x0000
38
```

```
39      command_result ← input_register_device(vms_input_dev)
40
41      if (command_result < 0)
42          printk("Error input_register_device\n")
43          return ERROR_REGISTER_INPUT_DEVICE
44
45      printk("Virtual BT Mouse Driver Initialized.\n")
46      return 0
```

2.3 Приложение для Android устройства

Клиентское приложение уровня пользователя должно осуществлять обработку запросов пользователя, установку соединения с серверным приложением и последующую передачу данных.

Для установки соединения с сервером необходимо использовать Bluetooth API [6]. Работа с bluetooth происходит в 4 этапа:

а) Инициализация адаптера.

б) Поиск доступного сервера.

в) Установка соединения.

г) Передача данных.

Инициализация адаптера происходит с помощью функции **BluetoothAdapter.getDefaultAdapter()**. Передача данных осуществляется при помощи сокетов. Сокет - это название программного интерфейса для обеспечения обмена данными между процессами. Процессы при таком обмене могут исполняться как на одной ЭВМ, так и на различных ЭВМ, связанных между собой сетью. Сокет — абстрактный объект, представляющий конечную точку соединения. Ниже представлен алгоритм подключения к серверу:

Листинг 2.2 — Алгоритм подключения к серверу

```
1  def connectToServer():
2      device ← adapter.getRemoteDevice(SERVER_MAC_address)
3      socket ← device.createRfcommSocketToServiceRecord(SERVER_UUID)
4      adapter.cancelDiscovery()
5      socket.connect()
6      stream ← socket.getOutputStream()
```

Для передачи данных мы используем выходной поток **stream**, созданный при подключении.

Листинг 2.3 — Алгоритм передачи данных

```
1  def sendDataToServer():
2      buffer ← msg.getBytes()
3      if (stream != null)
4          stream.write(buffer)
```

2.4 Сервер

Серверное приложение уровня пользователя должно решать следующие задачи:

— Установка соединения с клиентским приложением.
— Получение данных от клиентского приложения.
— Передача данных в пространство ядра.

Установка соединения с клиентским приложением осуществляется по алгоритму 2.1. Перед запуском сервера [7] необходимо установить уникальный UUID - 16-байтный номер, используемый для уникальной идентификации сервера. Кроме того, требуется установить значение порта.

Листинг 2.4 — Алгоритм запуска сервера

```
1  def startServer():
2      server.socket ← BluetoothSocket(RFCOMM)
3      server.socket ← bind("",server.port)
4      server.socket ← listen(server.port)
5
6      advertise_service( server.socket, server.name,
7          service_id ← server.uuid,
8          service_classes ← [ server.uuid, SERIAL_PORT_CLASS ],
9          profiles = [ SERIAL_PORT_PROFILE ]
10         )
11
12     print("Waiting a client...")
13
14     client.socket, client.info ← server.socket ← accept()
```

После запуска, сервер находится в ожидании подключения клиента. Как только один из клиентов подал правильный sdp-запрос, сервер переходит в бесконечный цикл для приема данных от клиента и последующей записи в пространство ядра.

Листинг 2.5 — Алгоритм передачи данных в пространство ядра

```
1 def sendData():
2     while True:
3         data ← client.socket ← recv(size)
4         if len(data) == 0: break
5         os.write(fd, data) # fd - file created by the driver for the
              adoption of the coordinates
6         os.fsync(fd)
7         if "</EOM>" in data: break
```

Как только клиент отправил запрос об окончании передачи данных, сервер отправляет ответное сообщение о закрытии подключения и закрывает сначала клиентский сокет, а затем и свой.

Листинг 2.6 — Закрытие подключения

```
1 def stopServer():
2     client.socket ← send("The server will be turned off soon")
3
4     client.socket ← close()
5     server.socket ← close()
```

3 Технологический раздел

3.1 Выбор языка программирования

Комплекс программ разработан для использования в операционной системе Linux Ubuntu. В рамках реализации было разработано три программы:

— Драйвер устройства.
— Bluetooth сервер.
— Пользовательское приложение для Android устройства.

Драйвер устройства для Linux разработан на языке Си. Данный выбор ограничен внутренним устройством ОС Linux и отсутствием средств разработки с использованием других языков.

Bluetooth сервер для приема данных разработан на языке Python. Выбор основан на простоте использования rfcomm сокетов с использованием протокола sdp.

Пользовательское приложение на платформе Android разработано на Java. Данный язык имеет строгую статическую типизацию, за счет чего выигрывает в производительности.

3.2 Выбор среды программирования

Для драйвера устройства и bluetooth сервера было решено использовать стандартный текстовый редактор и соответствующие компиляторы для языков: C и Python - gcc и python. В качестве средства разработки был выбран Sublime Text 2 - текстовый редактор с подцветкой синтаксиса. Приложение под Android разрабатывалось в интегрированной среде разработке Android Studio. Выбор был сделан в пользу данной IDE, поскольку Eclipse прекратила поддержку плагина Android Development Tools(ADP) в 2014 году. Из плюсов необходимо выделить:

— Способность протестирвать приложение на устройствах с разным экраном и с разной версией API.
— Множество шаблонов и макетов компонентов Android.

3.3 Установка и использование программного обеспечения

Для корректной работы драйвера и Android приложения необходимо скомпилировать модуль с помощью make файла и положить его в ядро с помощью команды `sudo insmod name_module`. Далее необходимо включить bluetooth на компьютере и запустить сервер. После проделанных операций можно использовать Android приложение. Интерфейс приложения представлен на рисунке 3.1.

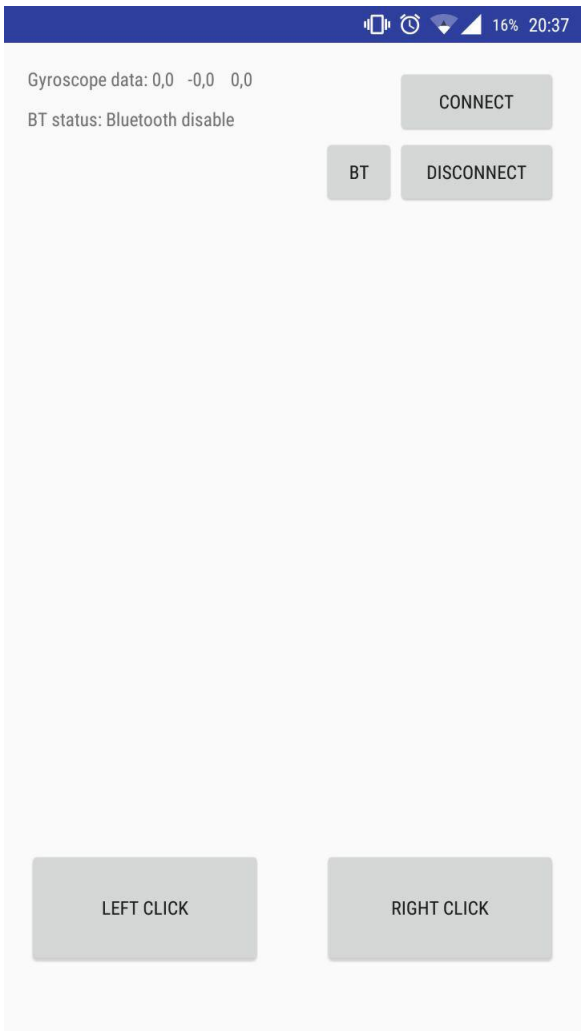

Рисунок 3.1 — Интерфейс приложения.

Функционал Android приложения

Как видно на рисунке 3.1 сверху слева показывается информация о координатах x, y и z гиродатчика. Строкой ниже выводится информация о состоянии bluetooth и подключения к серверу:

— `Bluetooth disable` - bluetooth неактивен, нет соединения с сервером.

— `Bluetooth enable` - bluetooth активен, нет соединения с сервером.

— `Device_name (device_address)` - соединение с сервером активно, где Device_name - имя сервера, device_address - адресс сервера.

Для взаимодействием с пользователем в программном обеспечении используются кнопки. В совокупности они составляют функционал программы, который представлен в таблице 3.1.

Таблица 3.1 — Функционал Android приложения

Кнопка	Действие
BT	Включение/выключение bluetooth
Connect	Подключение к серверу
Disconnect	Отключение от сервера
Left Click	Нажатие левой кнопки мыши. Предусмотрено двойное нажатие.
Right Click	Нажатие правой кнопки мыши

3.4 Технические требования

Требования для программного обеспечения минимальны. Для запуска драйвера необходима операционная система Linux. Для запуска приложения на Android устройстве требуется Android API 20 и выше.

Заключение

При написании программного обеспечения была проделана работа по изучению литературы, специализирующейся на создании драйверов устройств. Изучены способы беспроводной передачи данных, разновидности датчиков Android и их принцип работы. На базе полученных знаний были проанализированы достоинства и недостатки структур драйверов интерфейсов, датчиков смартфона, таких как акселерометр и гироскоп и передачи данных через bluetooth и wi-fi. Изучены механизмы встраивания драйвера устройства в ядро Linux, создание и дальнейшая работа bluetooth серверов на основе протоколов rfcomm и sdp.

Программное обеспечение разработано в соответствии с техническим заданием и протестировано на нескольких устройствах.

Данный комплекс программ имеет несколько направлений дальнейшего развития:

— Добавление фильтрации координат, необходимого для более плавного перемещения курсора.

— Исключение драйвера и bluetooth сервера путем представления Android устройства HID устройством.

— Добавление функциональности мыши (центральная кнопка, колесо).

Есть место для оптимизации приложения и расширения пользовательского интерфейса.

Список использованных источников

1. Gyroscopic Air Mouse Technology. — https://www.gyration.com/pages/gyration-technology. — 2016. — [Online; accessed 20-November-2016].

2. Accelerometer vs. Gyroscope: What's the Difference? — http://www.livescience.com/40103-accelerometer-vs-gyroscope.html. — 2013. — [Online; accessed 20-November-2016].

3. Bluetooth vs. Wi-Fi. — http://www.diffen.com/difference/Bluetooth_vs_Wifi. — 2013. — [Online; accessed 20-November-2016].

4. Jonathan Corbet Alessandro Rubini, Kroah-Hartman Greg. Linux Device Drivers, Third Edition. — O'Reilly, 2005.

5. Introduction to the Bluetooth Service Discovery Protocol. — http://homepages.inf.ed.ac.uk/group/sli_archive/slip0304_b/resources/com/karl/sdp/sdp_intro.html. — 2004. — [Online; accessed 14-December-2016].

6. Bluetooth in android. — https://developer.android.com/guide/topics/connectivity/bluetooth.html. — 2010. — [Online; accessed 14-December-2016].

7. An Introduction to Bluetooth Programming. — https://people.csail.mit.edu/albert/bluez-intro/index.html. — 2008. — [Online; accessed 14-December-2016].

Приложение А. Исходный код загружаемого модуля ядра.

Листинг 3.1 — Исходный код загружаемого модуля ядра (**mouse_driver.c**)

```
1  #include <linux/fs.h>
2  #include <asm/uaccess.h>
3  #include <linux/pci.h>
4  #include <linux/input.h>
5  #include <linux/platform_device.h>
6  #include <linux/module.h>
7  #include <linux/slab.h>
8  #include <linux/kernel.h>
9  #include <linux/usb.h>
10
11 #define LEFT_BUTTON_PRESSED 1
12 #define RIGHT_BUTTON_PRESSED 2
13 #define DOUBLE_CLICK 4
14 #define MOVE 3
15
16 #define KEY_WAS_PRESSED 8
17 #define KEY_WAS_RELEASED 0
18
19 #define ERROR_REGISTER_PLATFORM_DEVICE -1
20 #define ERROR_ALLOCATE_INPUT_DEVICE -2
21 #define ERROR_REGISTER_INPUT_DEVICE -3
22 #define ERROR_SYSFS_CREATE_GROUP -4
23
24 struct input_dev *vms_input_dev;
25
26 static struct platform_device *vms_dev;
27
28 static ssize_t write_vms(struct device *dev,
29                          struct device_attribute *attr,
30                          const char *buffer, size_t count)
31 {
32     int x = 0, y = 0, z = 0, command_type = 0;
33
34     sscanf(buffer, "%d%d%d%d", &command_type, &x, &y, &z);
35     printk("coordinates = %d %d %d %d\n", command_type, x, y, z);
36
37     input_report_rel(vms_input_dev, REL_X, -z);
38     input_report_rel(vms_input_dev, REL_Y, -x);
39
40     if (command_type == LEFT_BUTTON_PRESSED)
41     {
42         input_report_key(vms_input_dev, BTN_LEFT, KEY_WAS_PRESSED);
43         input_report_key(vms_input_dev, BTN_LEFT, KEY_WAS_RELEASED);
```

```
44      }
45      else if (command_type == RIGHT_BUTTON_PRESSED)
46      {
47          input_report_key(vms_input_dev, BTN_RIGHT, KEY_WAS_PRESSED);
48          input_report_key(vms_input_dev, BTN_RIGHT, KEY_WAS_RELEASED);
49      }
50      else if (command_type == DOUBLE_CLICK)
51      {
52          input_report_key(vms_input_dev, BTN_LEFT, KEY_WAS_PRESSED);
53          input_report_key(vms_input_dev, BTN_LEFT, KEY_WAS_RELEASED);
54          input_report_key(vms_input_dev, BTN_LEFT, KEY_WAS_PRESSED);
55          input_report_key(vms_input_dev, BTN_LEFT, KEY_WAS_RELEASED);
56      }
57      input_sync(vms_input_dev);
58      return count;
59  }
60
61  DEVICE_ATTR(coordinates, 0644, NULL, write_vms);
62
63  static struct attribute *vms_attrs[] =
64  {
65      &dev_attr_coordinates.attr,
66      NULL
67  };
68
69  static struct attribute_group vms_attr_group =
70  {
71      .attrs = vms_attrs,
72  };
73
74  static int __init display_init(void)
75  {
76      int command_result = 0;
77
78      vms_dev = platform_device_register_simple("vms", -1, NULL, 0);
79      if (IS_ERR(vms_dev))
80      {
81          PTR_ERR(vms_dev);
82          printk("vms_init: error\n");
83          return ERROR_REGISTER_PLATFORM_DEVICE;
84      }
85
86      command_result = sysfs_create_group(&vms_dev->dev.kobj,
            &vms_attr_group);
87      if (command_result < 0)
88      {
89          printk("Error sysfs_create_group\n");
```

```
90          return ERROR_SYSFS_CREATE_GROUP;
91      }
92
93      vms_input_dev = input_allocate_device();
94      if (!vms_input_dev)
95      {
96          printk("Bad input_alloc_device()\n");
97          return ERROR_ALLOCATE_INPUT_DEVICE;
98      }
99
100     set_bit(EV_REL, vms_input_dev->evbit);
101     set_bit(REL_X, vms_input_dev->relbit);
102     set_bit(REL_Y, vms_input_dev->relbit);
103     set_bit(EV_KEY, vms_input_dev->evbit);
104     set_bit(BTN_LEFT, vms_input_dev->keybit);
105     set_bit(BTN_RIGHT, vms_input_dev->keybit);
106
107
108     vms_input_dev->evbit[0] = BIT_MASK(EV_KEY) | BIT_MASK(EV_REL);
109     vms_input_dev->keybit[BIT_WORD(BTN_MOUSE)] = BIT_MASK(BTN_LEFT) |
110         BIT_MASK(BTN_MIDDLE) | BIT_MASK(BTN_RIGHT);
111     vms_input_dev->relbit[0] = BIT_MASK(REL_X) | BIT_MASK(REL_Y);
112
113     vms_input_dev->name = "Virtual BT mouse";
114     vms_input_dev->id.bustype = BUS_VIRTUAL;
115     vms_input_dev->id.vendor  = 0x0000;
116     vms_input_dev->id.product = 0x0000;
117     vms_input_dev->id.version = 0x0000;
118
119     command_result = input_register_device(vms_input_dev);
120     if (command_result < 0)
121     {
122         printk("Error input_register_device\n");
123         return ERROR_REGISTER_INPUT_DEVICE;
124     }
125     printk("Virtual BT Mouse Driver Initialized.\n");
126     return 0;
127 }
128
129 static void vms_cleanup(void)
130 {
131     input_unregister_device(vms_input_dev);
132     input_free_device(vms_input_dev);
133
134     sysfs_remove_group(&vms_dev->dev.kobj, &vms_attr_group);
135
136     platform_device_unregister(vms_dev);
```

```
137 }
138
139 module_init(display_init);
140 module_exit(vms_cleanup);
141
142 MODULE_LICENSE("GPL");
143 MODULE_AUTHOR("Kukuev Sergey");
144 MODULE_DESCRIPTION("Virtual BT mouse driver");
```