

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 . . . . .	3
1 Аналитический раздел . . . . .	4
1.1 Выбор способа управления . . . . .	4
1.2 Методы передачи данных . . . . .	5
1.3 Интерфейсы событий . . . . .	6
1.3.1 Интерфейс evdev . . . . .	6
1.3.2 Интерфейс mousetdev . . . . .	7
1.3.3 Интерфейс mydev . . . . .	7
2 Конструкторский раздел . . . . .	8
2.1 Структура разрабатываемого программного обеспечения . . . . .	8
2.2 Формат передаваемых данных между приложениями . . . . .	9
2.3 Приложение для Android устройства . . . . .	10
2.3.1 Работа с датчиками Android устройства . . . . .	10
2.3.2 Работа с Bluetooth . . . . .	10
2.4 Серверное приложение . . . . .	11
2.5 Загружаемый модуль ядра . . . . .	13
2.5.1 Чтение данных из пространства ядра . . . . .	13
2.5.2 Регистрация устройства в подсистеме ввода ядра . . . . .	13
3 Технологический раздел . . . . .	15
3.1 Выбор языка программирования . . . . .	15
3.2 Выбор среды программирования . . . . .	15
3.3 Установка и использование программного обеспечения . . . . .	15
3.4 Технические требования . . . . .	17
Заключение . . . . .	18
Список использованных источников . . . . .	19
Приложение А. Исходный код загружаемого модуля ядра. . . . .	20

## 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 продолжают. В настоящее время появились гироскопические мыши[1], позволяющие 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  $5^\circ$  (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 независимые 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.

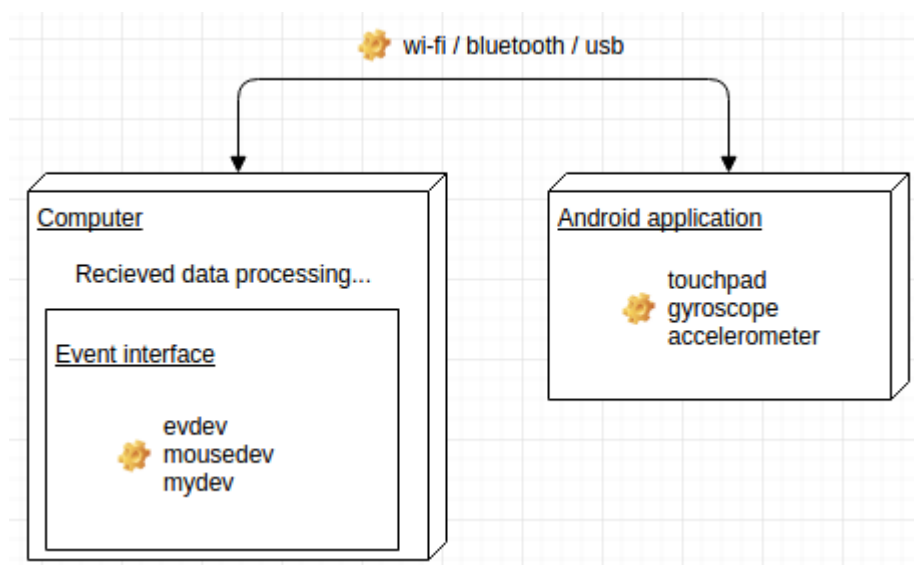


Рисунок 1.1 — Общая структура.

### 1.1 Выбор способа управления

Для выбора способа управления необходимо отталкиваться от поставленных задач. Самый распространенный вариант - имитация сенсорной панели. 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 позволяет получить 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 вибрационные микрогироскопы, принцип действия которых основан на силе Кориолиса[3]. Выходные данные - скорости вращения для каждой оси трехмерного пространства (X, Y и Z), которые можно использовать как скорость

перемещения курсора мыши. В таком случае, для вычисления перемещения курсора необходимо полученные данные умножить на время передвижения устройства в пространстве.

Акселерометр (G-сенсор) - устройство, предназначенное для измерения негравитационного ускорения. Различают три типа: однокомпонентные, двухкомпонентные и трехкомпонентные. Используя этот датчик, можно обнаруживать движение и изменение его скорости. Акселерометр не измеряет скорость как таковую, поэтому, основываясь на единичном замере, получить скорость движения невозможно. Вместо этого необходимо учитывать изменения ускорения на протяжении какого-то отрезка времени. Путем интегрирования полученных от датчика измерений, можно посчитать инерциальную скорость и координаты устройства. Изменение координат устройства можно использовать в качестве перемещения курсора мыши.

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[4] являются беспроводными стандартами связи и используются 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[5], которая объединяет все рассеянные драйвера для обработки периферийных устройств. Одним из ее плюсов является удобный интерфейс событий, позволяющий драйверу не создавать и не управлять узлами /dev и связанными с ними методами доступа. Вместо этого он имеет возможность вызывать API для ввода, чтобы отправить движения мыши. Такие системы, 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.

#### 1.3.1 Интерфейс evdev

«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 - тесное взаимодействие с драйвером устройством для X Window, который используется во всех UNIX-подобных ОС. Благодаря этому факту, данный драйвер событий является простым в использовании и универсальным. Набор функционала интерфейса сопоставим минимально требуемому для решения поставленной задачи.

### 1.3.2 Интерфейс mousedev

«Mousedev» - специальный драйвер событий ввода для мыши. Имеет стандартный набор точек входа: подключение устройства, отсоединение устройства, открытие, закрытие, чтение и запись файла устройства. В отличие от универсального evdev, этот интерфейс предназначен лишь для одного периферийного устройства. В драйвере реализуются события следующих типов:

- EV\_ABS - код события, связанного с абсолютным изменением значений координат по осям.
- EV\_REL - код события, связанного с относительным изменением значений координат по осям.
- EV\_KEY - код события, связанного с нажатием кнопки.
- EV\_SYN - описывает разделенные во времени или пространстве события (для multitouch сенсоров).

Отличительной особенностью является событие EV\_SYN, позволяющая писать драйвера устройства для сенсорных панелей.

### 1.3.3 Интерфейс mydev

Помимо готовых вариантов, имеется возможность написать собственный драйвер событий. Чтобы его написать, требуется экспортировать его в пользовательское пространство с помощью `/dev/input/mydev`, создать структуру, которая хранит в себе следующие данные:

- Обработчик сообщений о событиях, посылаемых драйверами устройств ввода.
- Указатель на методы управления, находящиеся в `/dev/input/mydev`.
- Младший номер `/dev/input/mydev`.
- Имя драйвера событий.
- Функцию регистрации драйвера.
- Функцию отмены регистрации драйвера.

Является более гибким вариантом, чем два, представленных выше, поскольку мы сможем постепенно наращивать функционал драйвера событий.

### Итоговый выбор

Для решения поставленной задачи целесообразней использовать универсальный драйвер событий ввода. Поскольку он оснащен требуемой функциональностью.

## 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 пользователю 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 протокола RFCOMM[6]. Используется как транспортный протокол протоколом L2CAP[6] (базовый протокол передачи данных для Bluetooth). Основной принцип действия заключается в эмулировании соединения point-to-point по последовательному порту. В связи с этим, структура программного обеспечения будет выглядеть как на рисунке 2.1.



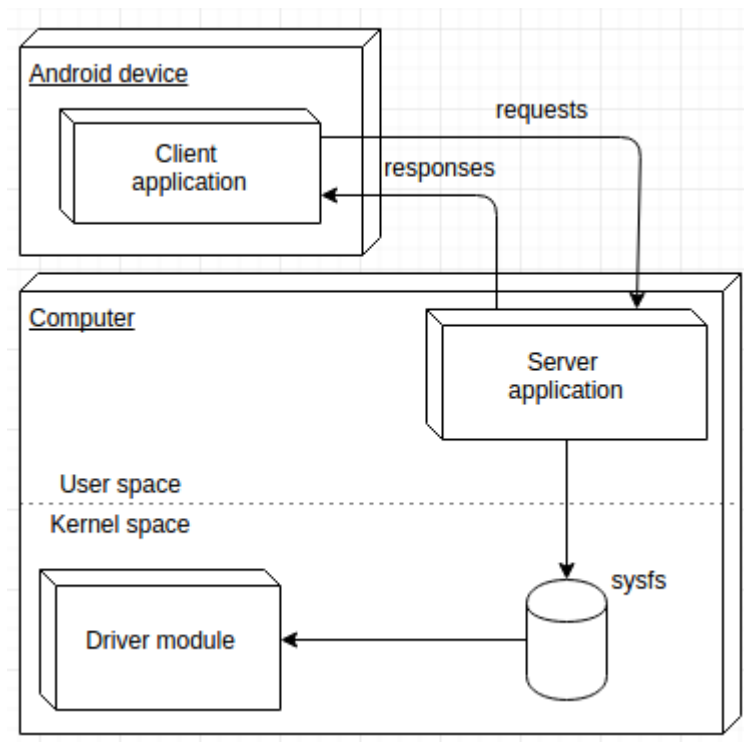


Рисунок 2.1 — Структура программного обеспечения

## 2.2 Формат передаваемых данных между приложениями

При обмене информацией между всеми приложениями 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, представленной на рисунке 2.2, состоящей из 4 чисел. Данные числа будут записаны в строку, которая в дальнейшем будет обрабатываться в ядре, и разделены пробелом.

Тип команды	Координата X	Координата Y	Координата Z
-------------	--------------	--------------	--------------

Рисунок 2.2 — Формат передаваемых данных

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

## 2.3 Приложение для Android устройства

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

### 2.3.1 Работа с датчиками Android устройства

В качестве способа управления был выбран гироскоп. В Android устройстве он представлен двумя структурами[7]: **TYPE\_GYROSCOPE** и **TYPE\_GYROSCOPE\_UNCALIBRATED**. Отличия структур в том, что **TYPE\_GYROSCOPE\_UNCALIBRATED** хранит скорости вращения вокруг осей X, Y и Z без дрейфа датчика, тем самым данные скорости являются неоткалиброванными. При этом дрейф содержится в отдельных переменных. Для данного программного обеспечения была выбрана структура откалиброванного гироскопа (**TYPE\_GYROSCOPE**).

Чтобы начать работу с датчиком, необходимо его проинициализировать:

Листинг 2.1 — Инициализация датчика

```
1   SensorManager ← (SensorManager)
      getSystemService (Context.SENSOR_SERVICE) ;
2   Sensor ← SensorManager.getDefaultSensor (Sensor.TYPE_GYROSCOPE) ;
```

После инициализации выставляем интервал получения данных от гироскопа и сохраняем информацию, полученную от датчика:

Листинг 2.2 — Установка времени и функция сохранения данных

```
1   sensorManager.registerListener (listener , sensorGyro ,
      SensorManager.SENSOR_DELAY_NORMAL)
2   ...
3   dev getData (SensorEvent event) :
4       for (int i=0; i < 3; i++)
5           coordinates[i] = event.values[i] * time
```

### 2.3.2 Работа с Bluetooth

Для установки соединения с сервером необходимо использовать Bluetooth API [8]. 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[9]. Процессы при таком обмене могут выполняться 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 2.3 — 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 2.4 — Алгоритм передачи данных

```
1 def sendDataToServer(command, coordinates):
2     msg ← Integer.toString(command) + " " +
3           Integer.toString(coordinates[0]) + " " +
4           Integer.toString(coordinates[1]) + " " +
5           Integer.toString(coordinates[2]) + "\n"
6     buffer ← msg.getBytes()
7     if (stream != null)
8         stream.write(buffer)
```

## 2.4 Серверное приложение

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 [10] необходимо установить уникальный UUID - 16-байтный номер, используемый для уникальной идентификации сервера. Кроме того, требуется установить значение порта.

### Листинг 2.5 — 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 запрос, сервер переходит в бесконечный цикл для приема данных от клиента и последующей записи в пространство ядра.

### Листинг 2.6 — 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
6                             adoption of the coordinates
7         os.fsync(fd)
8         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 2.7 — 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()

```

## 2.5 Загружаемый модуль ядра

Загружаемый модуль ядра должен решать следующие задачи:

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

### 2.5.1 Чтение данных из пространства ядра

Обмен данными между пространством ядра и пространством пользователя происходит при помощи виртуальной файловой системы в ОС Linux - `sysfs`[5]. В `sysfs` имеется подкаталог, где содержится вся информация о периферийных 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);`.

### 2.5.2 Регистрация устройства в подсистеме ввода ядра

Регистрация устройства происходит по следующим этапам:

- Регистрация платформо зависимого устройства в системе.
- Создание файла устройства в `sysfs`.
- Выделение памяти под устройство ввода.
- Установка обработчика на события.
- Регистрация устройства в подсистеме ввода.

Листинг 2.8 — 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 ,  
11                                       vms_attr_group);  
12  
13    if (command_result < 0)  
14        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     |
31     BIT_MASK(BTN_MIDDLE) | BIT_MASK(BTN_RIGHT)
32
33     vms_input_dev→relbit[0] ← BIT_MASK(REL_X) | BIT_MASK(REL_Y)
34
35     vms_input_dev→name ← "Virtual BT mouse"
36     vms_input_dev→id.bustype ← BUS_VIRTUAL
37     vms_input_dev→id.vendor ← 0x0000
38     vms_input_dev→id.product ← 0x0000
39     vms_input_dev→id.version ← 0x0000
40
41     command_result ← input_register_device(vms_input_dev)
42
43     if (command_result < 0)
44         printk("Error input_register_device\n")
45         return ERROR_REGISTER_INPUT_DEVICE
46
47     printk("Virtual BT Mouse Driver Initialized.\n")
48     return 0

```

## **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 протестировать 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.

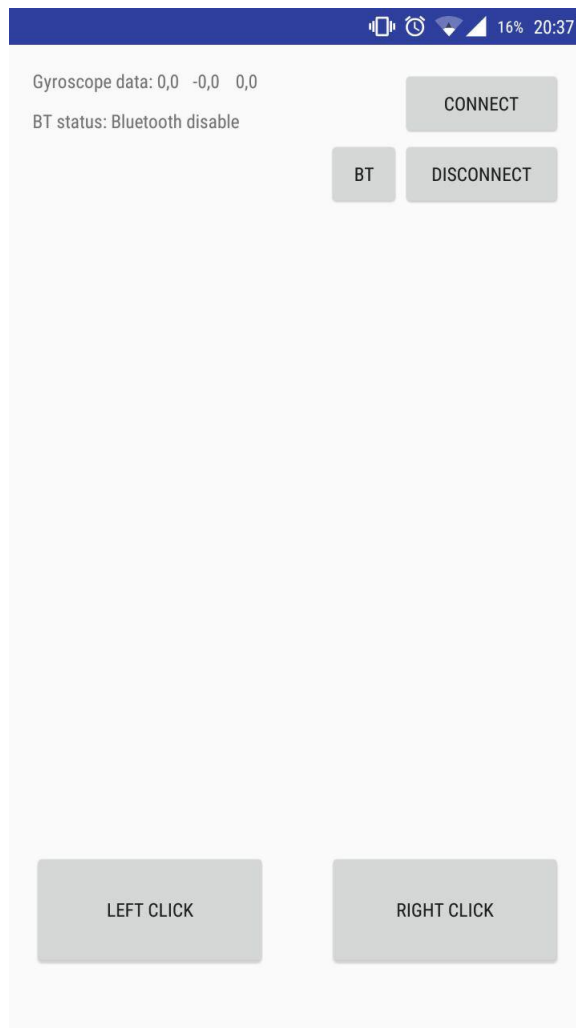


Рисунок 3.1 — Интерфейс приложения.

### 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** - адрес сервера.

Для взаимодействия 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. V. Saveliev I. Course of general physics, Second Edition. — Science. Home edition of Physical and Mathematical Literature., 1982.
4. Bluetooth vs. Wi-Fi. — [http://www.diffen.com/difference/Bluetooth\\_vs\\_Wifi](http://www.diffen.com/difference/Bluetooth_vs_Wifi). — 2013. — [Online; accessed 20-November-2016].
5. Jonathan Corbet Alessandro Rubini, Kroah-Hartman Greg. Linux Device Drivers, Third Edition. — O'Reilly, 2005.
6. Bluetooth Protocol. — <http://www.gaw.ru/html.cgi/txt/doc/Wireless/bluetooth/7.htm>. — 2004. — [Online; accessed 16-December-2016].
7. Motion Sensors. — [https://developer.android.com/guide/topics/sensors/sensors\\_motion.html](https://developer.android.com/guide/topics/sensors/sensors_motion.html). — 2015. — [Online; accessed 16-December-2016].
8. Bluetooth in android. — <https://developer.android.com/guide/topics/connectivity/bluetooth.html>. — 2015. — [Online; accessed 14-December-2016].
9. Borovskiy Andrew. Advanced Programming in the Linux. — Linux Format, 2007.
10. 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", &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,
87                                         &vms_attr_group);
88     if (command_result < 0)
89     {
90         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");
```