

КАФЕДРА ИУ7 «Программное обеспечение ЭВМ и информационные технологии»

### ***НА ТЕМУ:***

***«Разработка загружаемого модуля ядра для удалённого управления курсором компьютерной мыши с помощью смартфона»***

\_\_\_\_\_ **Д. О. Звягин**  
(Подпись, дата) (И.О.Фамилия)

**Н. Ю. Рязанова**

---

(Подпись, дата)                      (И.О.Фамилия)

**Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«Московский государственный технический университет имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)**

---

УТВЕРЖДАЮ

Заведующий кафедрой ИУ7

И. В. Рудаков

«\_\_» сентября 2025 г.

**З А Д А Н И Е  
на выполнение курсовой работы**

по дисциплине

**Операционные системы**

Студент группы **ИУ7-73Б Звягин Даниил Олегович**

Тема курсовой работы **Мониторинг планирования процессов в Linux**

Направленность КР (учебная, исследовательская, практическая, др.): учебная.

Источник тематики (кафедра, предприятие, НИР): кафедра.

График выполнения НИР: 25% к 5 нед., 50% к 8 нед., 75% к 11 нед., 100% к 15 нед.

***Задание***

*Разработать загружаемый модуль ядра, предоставляющий пользователю возможность удалённого управления курсором мыши с помощью смартфона.*

***Оформление курсовой работы:***

Расчетно-пояснительная записка на 30-40 листах формата А4.

***Перечень графического (иллюстративного) материала:***

Презентация на 6-15 слайдах.

Дата выдачи задания «\_\_» сентября 2025 г.

**Руководитель курсовой работы**

\_\_\_\_\_ **Н. Ю. Рязанова**

**Студент**

\_\_\_\_\_ **Д. О. Звягин**

# РЕФЕРАТ

Расчетно-пояснительная записка 59 с., 14 рис., 24 ист.

ОПЕРАЦИОННЫЕ СИСТЕМЫ, ЗАГРУЖАЕМЫЙ МОДУЛЬ ЯДРА, ПОДСИСТЕМА  
ВВОДА, BLUETOOTH, RFCOMM, ВИРТУАЛЬНОЕ УСТРОЙСТВО МЫШИ

Цель работы — разработка загружаемого модуля ядра Linux, обеспечивающего управление курсором мыши с использованием сенсорного экрана мобильного устройства по каналу Bluetooth.

В работе реализовано виртуальное устройство ввода типа «мышь», зарегистрированное в подсистеме input ядра Linux. Передача управляющих данных осуществляется по протоколу RFCOMM. Мобильное приложение для операционной системы Android формирует сообщения о перемещении курсора и состояниях кнопок мыши на основе событий сенсорного экрана.

Разработанный модуль обеспечивает приём сообщений по Bluetooth, их декодирование и генерацию соответствующих событий подсистемы ввода. Реализованное программное решение позволяет использовать мобильное устройство в качестве беспроводного манипулятора для управления курсором рабочей станции.

# СОДЕРЖАНИЕ

|  |           |
|--|-----------|
| <b>РЕФЕРАТ</b> . . . . .   | <b>3</b>  |
| <b>СОКРАЩЕНИЯ</b> . . . . .  | <b>5</b>  |
| <b>ВВЕДЕНИЕ</b> . . . . .  | <b>6</b>  |
| <b>1 Аналитический раздел</b> . . . . .  | <b>7</b>  |
| 1.1 Постановка задачи . . . . .  | 7         |
| 1.2 Способы преобразования касаний сенсорного экрана в движение мыши . . . . . | 7         |
| 1.3 Способы обмена данными между телефоном и модулем ядра . . . . .            | 8         |
| 1.4 Способы управления курсором мыши из загружаемого модуля ядра . . . . .     | 11        |
| <b>2 Конструкторский раздел</b> . . . . .                                      | <b>14</b> |
| 2.1 Общая структура решения . . . . .  | 14        |
| 2.2 Базовые структуры и точки входа драйвера . . . . .                         | 15        |
| 2.3 Последовательность работы модуля ядра . . . . .                            | 15        |
| 2.4 Последовательность работы мобильного приложения . . . . .                  | 21        |
| <b>3 Технологический раздел</b> . . . . .                                      | <b>28</b> |
| 3.1 Выбор языка и среды программирования . . . . .                             | 28        |
| 3.2 Реализация загружаемого модуля ядра Linux . . . . .                        | 28        |
| 3.3 Реализация мобильного приложения для Android . . . . .                     | 36        |
| <b>4 Исследовательский раздел</b> . . . . .                                    | <b>42</b> |
| 4.1 Технические характеристики . . . . .                                       | 42        |
| 4.2 Демонстрация работы программы . . . . .                                    | 42        |
| <b>ЗАКЛЮЧЕНИЕ</b> . . . . .  | <b>45</b> |
| <b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</b> . . . . .                              | <b>46</b> |
| <b>ПРИЛОЖЕНИЕ А</b> . . . . .  | <b>48</b> |

# **СОКРАЩЕНИЯ**

HID — Human Interface Device

RFCOMM — Radio Frequency Communication

# ВВЕДЕНИЕ

Разработан загружаемый модуль ядра Linux, реализующий виртуальное устройство ввода типа «мышь», управление которым осуществляется с использованием сенсорного экрана мобильного телефона по каналу Bluetooth. Для формирования управляющих воздействий используется мобильное приложение под управлением операционной системы Android.

Целью работы является разработка загружаемого модуля ядра Linux и взаимодействующего с ним мобильного приложения, обеспечивающего передачу команд управления курсором от мобильного устройства к ПК по беспроводному каналу связи.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- проанализировать предметную область;
- разработать алгоритмы функционирования загружаемого модуля ядра и мобильного приложения;
- реализовать загружаемый модуль ядра Linux, обеспечивающий генерацию событий подсистемы ввода;
- реализовать мобильное приложение для операционной системы Android;
- выполнить тестирование разработанного программного комплекса.

# 1 Аналитический раздел

## 1.1 Постановка задачи

В соответствии с заданием на курсовую работу, необходимо разработать загружаемый модуль ядра Linux, реализующий виртуальное устройство мыши, принимающий команды от мобильного приложения на смартфоне по каналу Bluetooth и преобразующий эти команды в события подсистемы ввода.

Для выполнения этой задачи требуется:

- 1) проанализировать и выбрать способ перевода касаний сенсорного экрана телефона в движение курсора мыши;
- 2) проанализировать и выбрать метод передачи информации с телефона на компьютер;
- 3) проанализировать и выбрать способ генерации событий подсистемы ввода;
- 4) разработать алгоритмы и структуру загружаемого модуля ядра, обрабатывающего информацию, полученную с телефона и преобразующего её в события подсистемы ввода;
- 5) реализовать загружаемый модуль ядра, выполняющий поставленную задачу;
- 6) протестировать работу реализованного загружаемого модуля ядра.

## 1.2 Способы преобразования касаний сенсорного экрана в движение мыши

При интеграции мобильного приложения, формирующего события касаний, и загружаемого модуля ядра, эмулирующего поведение мыши, необходимо определить способ сопоставления данных сенсорного экрана с координатами курсора.

В соответствии с логикой построения указательных устройств возможно два подхода:

- 1) преобразование координат касаний в *абсолютные* координаты на экране ПК;
- 2) преобразование перемещения пальца в *относительные* смещения курсора.

Абсолютные координаты требуют точного соответствия геометрии сенсорного устройства и рабочего стола, а также нормализации значений, поступающих от Android-приложения, к диапазону координат подсистемы ввода ядра Linux. Такой способ приводит к ряду ограничений: несовпадение пропорций сенсорной панели и монитора, масштабирование рабочего стола и наличие нескольких экранов приводят к неоднозначности отображения координат.

Кроме того, абсолютное позиционирование предполагает передачу полного набора координат при каждом событии касания, что повышает интенсивность обмена данными по Bluetooth и увеличивает требования к стабильности канала связи.

В отличие от этого, использование относительных координат основано на передаче смещений  $dx$ ,  $dy$ , вычисляемых из последовательности касаний (или непрерывного движения). Такой подход соответствует модели классических HID-мышей, где устройство генерирует относительные изменения положения независимой от абсолютной позиции курсора.

Кроме того, относительные смещения естественным образом интегрируются в архитектуру Android-приложения, где компонент обработчика касаний оперирует разностями координат, что исключает необходимость приведения значений к масштабу удалённого экрана.

С учётом перечисленных факторов, был выбран способ передачи *относительных координат*. Он достаточен для обеспечения движения мыши, а также не требует конфигурации для сопоставления масштабов экрана.

### 1.3 Способы обмена данными между телефоном и модулем ядра

Передача данных между мобильным устройством и загружаемым модулем ядра может быть реализована различными способами, включая сетевые протоколы, каналы межпроцессного взаимодействия через пользовательское пространство и беспроводные стеки передачи данных.

Для решения поставленной задачи выбрано беспроводное соединение по технологии Bluetooth. Оно обеспечивает устойчивый двунаправленный канал связи, нативно поддерживается в ядре Linux через подсистему BlueZ и предоставляет API, совместимый с типовыми средствами разработки Android-приложений.

Стек Bluetooth в Linux реализуется в виде подсистемы BlueZ, включающей поддержку базового протокола L2CAP, протокола RFCOMM, профиля HID и вспомогательных служб [1—3]. На уровне ядра для взаимодействия с Bluetooth-устройствами используются сокеты семейства PF\_BLUETOOTH с такими протоколами, как BTPROTO\_L2CAP и BTPROTO\_RFCOMM [1; 4]. В пространстве ядра доступен интерфейс для создания и использования таких сокетов, обеспечивающий работу с Bluetooth-соединениями без участия пользовательского процесса [4].

С точки зрения обмена данными между смартфоном и модулем ядра возможны следующие варианты:

- 1) использование RFCOMM-сокетов в пространстве ядра;
- 2) работа непосредственно с L2CAP в пространстве ядра;
- 3) задействование профиля HID поверх Bluetooth;
- 4) использование устройств /dev/rfcommN и пользовательского пространства.

#### *RFCOMM-сокеты в пространстве ядра*

Протокол RFCOMM реализует поверх L2CAP байтовый поток, логически аналогичный последовательному порту, и применяется для построения сервисов, требующих надёжного двунаправленного канала [1; 3]. В ядре Linux поддержка RFCOMM интегрирована в сетевой стек, что позволяет создавать серверные и клиентские сокеты с использованием семейства PF\_BLUETOOTH и протокола BTPROTO\_RFCOMM [1; 5]. Адресация RFCOMM-сокета выполняется с использованием структуры адреса, содержащей семейство, Bluetooth-адрес удалённого устройства и номер канала.

На стороне Android-приложения подключение к такому сервису реализуется через API



класса `BluetoothSocket`, который инкапсулирует установление RFCOMM-соединения по указанному UUID сервиса [6; 7]. Таким образом реализуется связка: серверный RFCOMM-сокет в пространстве ядра и клиентское соединение в приложении Android.

При использовании RFCOMM модуль ядра получает поток байтов непосредственно от смартфона, что позволяет задать прикладной протокол управления курсором (например, фиксированный формат кадров с координатами и битовой маской кнопок) без вовлечения дополнительных уровней абстракции [1; 3]. Обработка входящего потока выполняется в обработчиках на стороне загружаемого модуля ядра, что упрощает синхронизацию с подсистемой ввода [4].

#### *Использование L2CAP в пространстве ядра*

L2CAP представляет собой базовый протокол Bluetooth, обеспечивающий мультиплексирование каналов и передачу пакетов между устройствами [1; 2]. Прямое использование L2CAP даёт доступ к более низкому уровню стека и предоставляет гибкость при реализации собственных протоколов, но требует дополнительной обработки параметров канала и управления MTU [8].

В контексте рассматриваемой задачи использование L2CAP в качестве средства передачи данных для прикладного протокола управления курсором приводит к усложнению логики модуля ядра и дублированию функциональности, уже реализованной в RFCOMM, как надстройке над L2CAP [1; 2]. Кроме того, на стороне Android типовые высокоуровневые API ориентированы на RFCOMM-сервисы, что делает прямую работу с L2CAP при реализации мобильного приложения неоправданным ограничением [6].

#### *Профиль HID поверх Bluetooth*

Для устройств ввода, Bluetooth-стек предусматривает профиль HID, обеспечивающий транспорт HID-отчётов по каналу L2CAP и интеграцию с HID-core ядра [1; 9]. Такой подход используется для аппаратных Bluetooth-мышей и клавиатур: транспортный драйвер профиля HID принимает HID-отчёты по Bluetooth и передаёт их в HID-core, где они интерпретируются и преобразуются в события подсистемы ввода [9; 10]. Поддержка профиля HIDP на стороне ядра конфигурируется параметром `CONFIG_BT_HIDP` в конфигурации ядра [11].

Применение профиля HID для эмуляции мыши на базе смартфона потребовало бы разработки или адаптации транспортного драйвера, а также формирования корректных HID-описателей и отчётов [12; 13]. В рамках курсовой работы такая интеграция выходит за пределы необходимого объёма, так как влечёт за собой разработку HID-описателя, поддержку отчётов и согласование с существующей инфраструктурой HID при отсутствии требований к использованию расширенных возможностей HID-протокола [9; 14].

#### *Использование устройств `/dev/rfcommN` и пользовательского пространства*

Подсистема BlueZ предоставляет возможность отображать RFCOMM-соединения в виде псевдотерминальных устройств `/dev/rfcommN`, которые доступны пользовательским прило-

жениям как последовательные порты [3; 15]. Пользовательские программы открывают такие устройства через стандартные системные вызовы и читают поток байтов, реализуя прикладной протокол в пользовательском пространстве [**rfcomm-missing-dev**; 15].

Для интеграции с модулем ядра в этом случае потребовалась бы дополнительная связка между пользовательским процессом, обрабатывающим `/dev/rfcommN`, и драйвером мыши, например через символьное устройство или вспомогательный интерфейс обмена. Такая архитектура приводит к разделению логики между модулем ядра и пользовательским процессом и увеличивает количество точек отказа. Кроме того, основной функциональный поток управления курсором в этом случае реализуется в пользовательском пространстве, тогда как модуль ядра выполняет вспомогательные функции маршрутизации событий [16].

#### *Обоснование выбора RFCOMM-сокета в пространстве ядра*

В результате анализа способов передачи данных в модуль ядра по протоколу Bluetooth, получены следующие выводы:

- L2CAP предоставляет универсальный транспортный уровень и используется в качестве основы для протоколов более высокого уровня, однако при прямом использовании влечёт усложнение драйвера за счёт необходимости дополнительной обработки параметров канала [1; 2; 8];
- профиль HID ориентирован на аппаратные HID-устройства и предполагает интеграцию с HID-core и существующей реализацией профиля в ядре, что не требуется для протокола обмена фиксированного формата между смартфоном и драйвером мыши [1; 9—11];
- использование `/dev/rfcommN` и пользовательских программ переводит основную часть обработки в пользовательское пространство и требует дополнительного канала взаимодействия с модулем ядра [3; 15; 16];
- RFCOMM-сокет в пространстве ядра обеспечивает потоковый канал непосредственно в драйвер, использует реализованный протокол поверх L2CAP и поддерживает соединение с высокоуровневым API Android-приложения [4—7].

На основании этих соображений в разрабатываемом драйвере выбран подход, основанный на серверном RFCOMM-сокет в пространстве ядра. Модуль ядра, фактически, становится сервером, к которому присоединяются клиенты с помощью мобильного приложения Android. Этот сервер обрабатывает сообщения фиксированного формата с данными о перемещении и состояниях кнопок, выполняет обработку входящих данных и синхронно генерирует события подсистемы ввода для виртуального устройства мыши [1; 6; 17]. Такой стек (виртуальное устройство в подсистеме input и RFCOMM-соединение в пространстве ядра) соответствует постановке задачи и является достаточным для реализации драйвера.

## 1.4 Способы управления курсором мыши из загружаемого модуля ядра

Для управления курсором из загружаемого модуля рассматриваются три класса подходов:

- 1) регистрация виртуального устройства ввода в подсистеме `input`;
- 2) генерация событий через подсистему `uinput` из пользовательского пространства;
- 3) интеграция с подсистемой HID с эмуляцией HID-манипулятора.

### *Виртуальное устройство в подсистеме `input`*

Базовый способ генерации событий мыши из загружаемого модуля ядра заключается в регистрации виртуального устройства ввода через подсистему `input` [17; 18]. Драйвер выделяет и инициализирует структуру `struct input_dev`, заполняя сведения об идентификаторе устройства и поддерживаемых типах событий, а затем регистрирует её в `input-core`, после чего в системе появляется соответствующее устройство мыши [17]. Для устройства мыши обычно указываются типы событий `EV_REL` (относительное перемещение по осям `REL_X`, `REL_Y`) и `EV_KEY` (нажатия `BTN_LEFT`, `BTN_RIGHT`) [9; 13; 17].

Генерация событий для зарегистрированного устройства выполняется специализированными функциями подсистемы ввода, такими как `input_report_rel`, `input_report_key` и `input_sync`, которые обновляют внутреннее состояние устройства и доставляют события всем подписанным обработчикам [17; 18]. В контексте разрабатываемого драйвера каждое поступившее сообщение от смартфона интерпретируется как набор элементарных действий мыши (смещение по осям и изменение состояний кнопок) и преобразуется в одну или несколько последовательностей вызовов указанных функций. Такой подход обеспечивает интеграцию с архитектурой ввода и делает виртуальное устройство неотличимым от аппаратной мыши для остального программного обеспечения [9; 17].

### *Подсистема `uinput` и генерация событий из пользовательского пространства*

Подсистема `uinput` реализует интерфейс, позволяющий пользовательским процессам создавать виртуальные устройства ввода и генерировать события от их имени через специальное символьное устройство `/dev/uinput` [17; 19]. Пользовательское приложение описывает возможности устройства, после чего отправляет структуры `struct input_event`, которые ядро интерпретирует как события подсистемы ввода [18; 19]. Этот механизм применяется для реализации эмуляторов устройств ввода и прикладных программ, инжектирующих события в пользовательском пространстве.

Для решения рассматриваемой задачи применение `uinput` означало бы перенос логики приёма данных по Bluetooth и обработки протокола с модуля ядра в пользовательское приложение, которое затем транслировало бы события мыши через `/dev/uinput`. При таком подходе загружаемый модуль ядра фактически не участвовал бы в формировании событий, а являлся

бы вспомогательным компонентом либо полностью отсутствовал. Этот подход не совпадает с утверждённым заданием на курсовую работу.

### *Интеграция с подсистемой HID и интерфейс UHID*

Подсистема HID ядра Linux обеспечивает поддержку устройств ввода, реализующих стандартный HID-протокол (клавиатуры, мыши, графические планшеты и др.), и опирается на разделение на транспортные драйверы и HID-core [9; 12; 13]. Транспортный драйвер отвечает за доставку HID-отчётов от конкретной шины (USB, Bluetooth и т.п.), а HID-core интерпретирует отчёты, формируя события подсистемы ввода [9; 14]. Интерфейс UHID предоставляет возможность создавать HID-устройства из пользовательского пространства: пользовательский процесс взаимодействует с устройством `/dev/uhid`, отправляя события, которые далее обрабатываются HID-core и преобразуются в события ввода [9; 20].

Использование HID-подсистемы для эмуляции мыши обеспечивает совместимость с существующим стеком драйверов и поддержкой HID в ядре [9; 13]. Однако для учебной задачи, в которой акцент сделан на реализации логики непосредственно в модуле ядра, такой подход требует разработки HID-дескриптора, генерации корректных HID-отчётов и учёта дополнительных особенностей HID-профиля, что усложняет структуру драйвера без необходимости использования расширенных возможностей HID-протокола [12; 14].

### *Другие способы вмешательства в стек ввода*

На практике применяются подходы, основанные на перехвате или модификации событий на поздних этапах стека ввода, например через перехват операций устройств `/dev/input/eventX` или вмешательство в обработчики графической подсистемы [17]. Такие решения опираются на модификацию существующих драйверов либо на установку промежуточных слоёв между ядром и пользовательскими приложениями. Для курсовой работы по разработке загружаемого модуля ядра эти варианты не соответствуют постановке задачи, так как не создают отдельного драйвера мыши, а изменяют поведение уже существующих компонентов.

### *Обоснование выбора подсистемы input и виртуального устройства мыши*

В результате анализа рассмотренных подходов сформулированы следующие выводы:

- регистрация виртуального устройства через подсистему `input` обеспечивает интеграцию с ядром, контролируемый набор зависимостей и представление мыши в системе как стандартного устройства ввода, управляемого загружаемым модулем ядра [17; 18];
- использование `uinput` переносит ключевую логику в пользовательское пространство и приводит к тому, что основная часть функциональности оказывается реализованной вне модуля ядра [16; 19];
- интеграция через HID и UHID требует разработки HID-дескриптора и обработки HID-отчётов, что не является необходимым для эмуляции мыши с фиксированным набором событий, но усложняет реализацию драйвера [9; 13].

В связи с этим, в разрабатываемой системе выбран подход, основанный на регистрации виртуального устройства мыши в подсистеме `input` и генерации событий с помощью функций подсистемы ввода из загружаемого модуля ядра [17; 18]. Этот способ обладает достаточным функционалом для выполнения работы и совпадает с утверждённым заданием.

## Выводы

В разделе проанализированы варианты преобразования касаний в движение, способы беспроводного обмена и варианты интеграции с подсистемой ввода. Для сопоставления касаний выбраны относительные смещения  $dx/du$ ; этот способ не требует согласования геометрии экранов и достаточен для достижения цели — движения курсора по экрану ПК.

Для передачи данных между смартфоном и модулем ядра, при требовании использовать Bluetooth, рассмотрены L2CAP, HID, устройства `/dev/rfcommN` и RFCOMM в ядре. Выбран серверный RFCOMM-сокеты в пространстве ядра с клиентским `BluetoothSocket` на Android, так как он даёт потоковый канал без пользовательских прослоек и покрывает нужный объём функций.

Для генерации событий ввода сопоставлены регистрация виртуального устройства через `struct input_dev`, использование `uinput` и интеграция через HID/UHID. Выбрана регистрация собственного устройства в подсистеме `input`: она позволяет формировать события `EV_REL` и `EV_KEY` непосредственно из модуля ядра и держать логику в драйвере.

Итоговая схема «Android-приложение — RFCOMM — загружаемый модуль ядра — подсистема `input`» отражает рассмотренные решения и достаточна для реализации драйвера в соответствии с требованиями ТЗ.

## 2 Конструкторский раздел

### 2.1 Общая структура решения

Разрабатываемое решение включает два основных компонента:

- загружаемый модуль ядра Linux, регистрирующий виртуальное устройство ввода типа «мышь» в подсистеме ввода и принимающий команды по RFCOMM-соединению Bluetooth [1; 3; 17; 18];
- мобильное приложение для Android, устанавливающеее RFCOMM-соединение с рабочей станцией и преобразующее действия пользователя на сенсорном экране в последовательность бинарных сообщений фиксированного формата [6; 7].

Модуль ядра взаимодействует с подсистемой ввода через структуру `struct input_dev` и функции генерации событий ввода [17; 18], а с подсистемой Bluetooth — через серверный сокет с семейством `PF_BLUETOOTH` и протоколом `BTPROTO_RFCOMM` [1; 4; 5]. Мобильное приложение использует стек Bluetooth Android и API `BluetoothAdapter`, `BluetoothDevice` и `BluetoothSocket` для установления соединения с модулем ядра и обмена бинарными сообщениями [6; 7].

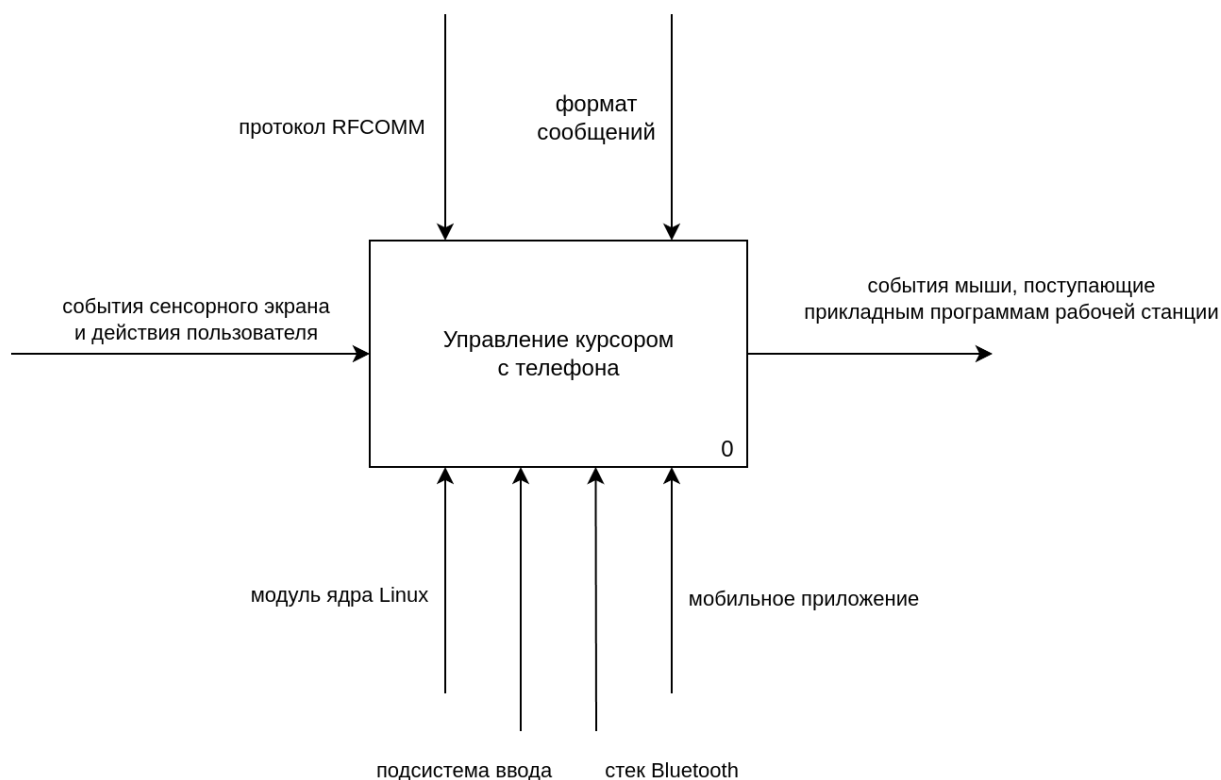


Рисунок 1 — IDEF0-нулевого уровня

## 2.2 Базовые структуры и точки входа драйвера

Загружаемый модуль ядра использует следующие точки входа [16]:

- функцию инициализации, регистрируемую через макрос `module_init`, выполняющую создание виртуального устройства ввода, настройку RFCOMM-сокета и запуск служебного потока;
- функцию завершения, регистрируемую через макрос `module_exit`, выполняющую остановку служебного потока, закрытие RFCOMM-сокета и снятие виртуального устройства с регистрации в подсистеме ввода;
- функцию служебного потока обработки соединения, создаваемого с помощью `kthread_run` и выполняющего цикл приёма команд по RFCOMM и генерацию событий ввода [21].

Для интеграции с подсистемой ввода используется структура `struct input_dev`, в которой настраиваются:

- поддерживаемые типы событий `EV_REL` и `EV_KEY`;
- коды событий `REL_X`, `REL_Y`, `BTN_LEFT`, `BTN_RIGHT`;
- идентификаторы производителя, продукта и человекочитаемое имя устройства [17; 18].

Генерация событий выполняется вызовами `input_report_rel`, `input_report_key` и `input_sync` для зарегистрированного устройства [17].

Для взаимодействия с RFCOMM создаётся серверный Bluetooth-сокет с семейством `PF_BLUETOOTH` и протоколом `BTPROTO_RFCOMM`. Адрес сокета задаётся структурой адреса с полями семейства адресов, Bluetooth-адреса устройства и номера RFCOMM-канала, после чего выполняются операции привязки и перевода сокета в режим прослушивания [1; 4; 5].

## 2.3 Последовательность работы модуля ядра

Работа модуля ядра логически разделяется на три этапа: инициализацию, обслуживание соединения и завершение работы.

На этапе инициализации выполняются проверка параметров, регистрация виртуального устройства ввода в подсистеме `input`, создание и настройка серверного RFCOMM-сокета и запуск служебного потока обработки соединения. Последовательность действий функции инициализации представлена на рис. 2.

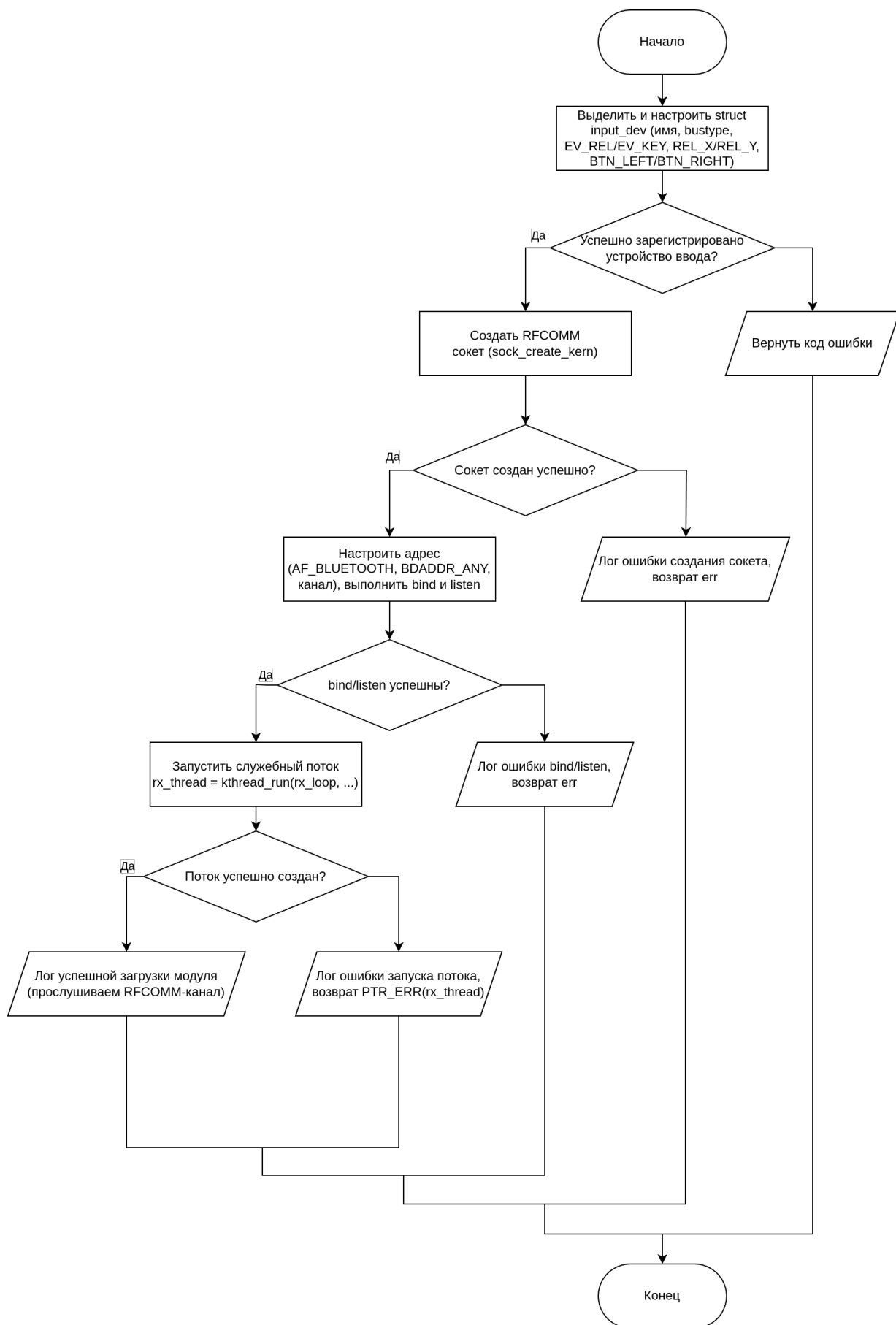


Рисунок 2 — Схема алгоритма инициализации модуля ядра phone\_mouse\_bt



Служебный поток `rx_loop` реализует цикл ожидания входящего RFCOMM-соединения, приёма и проверки сообщений фиксированной длины, обработки разрыва соединения и передачи декодированных команд во внутренние обработчики, генерирующие события ввода. Логика работы потока, включая обработку временного отсутствия данных, разрыва соединения и игнорирования некорректных пакетов, показана на рис. 3.

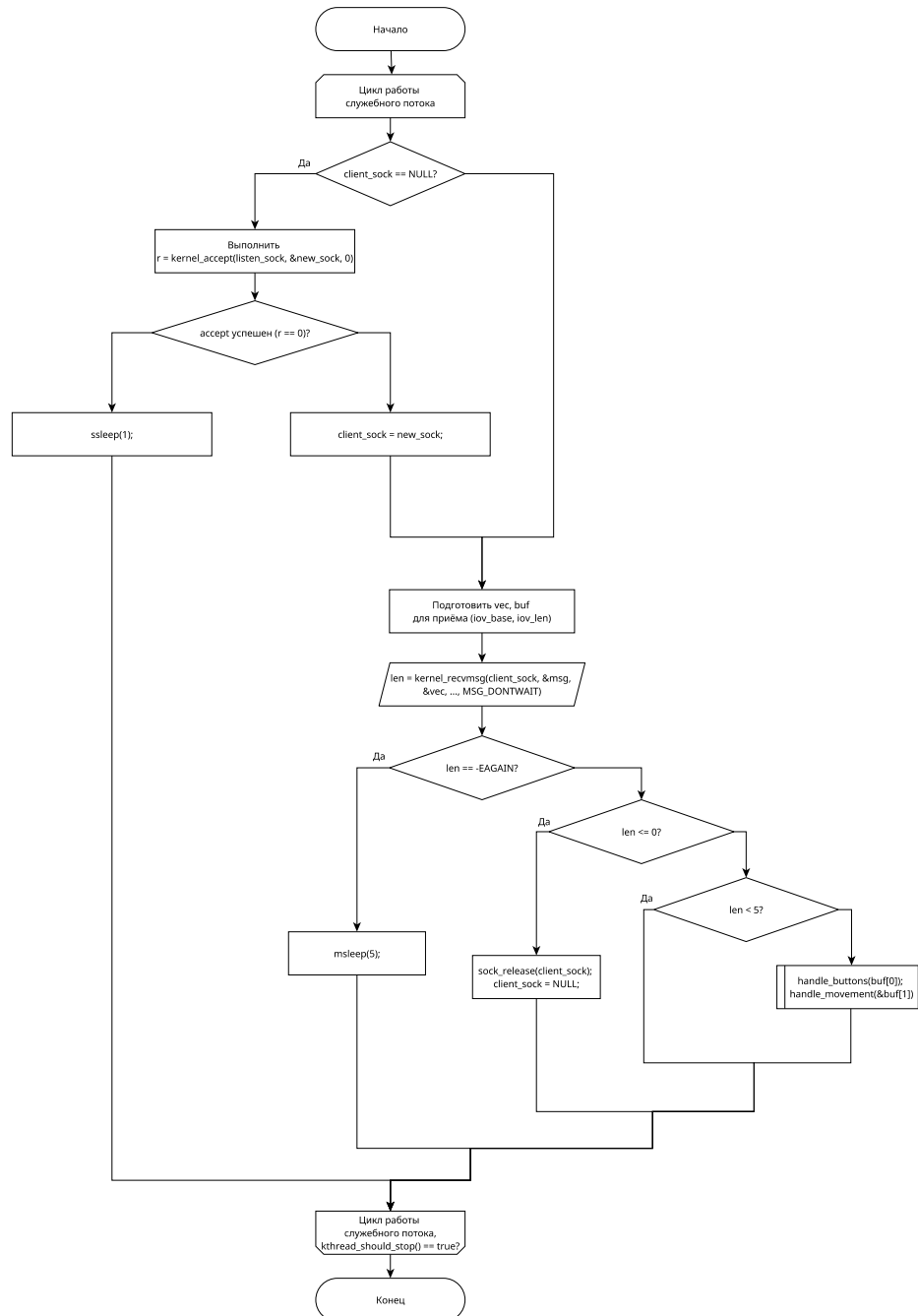


Рисунок 3 — Схема алгоритма работы служебного потока `rx_loop`

Обработка состояний кнопок мыши вынесена в отдельную функцию `handle_buttons`. Эта функция декодирует битовую маску нажатых кнопок и порождает для каждой активной кнопки последовательность событий нажатия и отпускания с вызовами `input_report_key` и `input_sync`, формируя короткие клики левой и правой кнопок мыши. Схема алгоритма `handle_buttons` приведена на рис. 4.

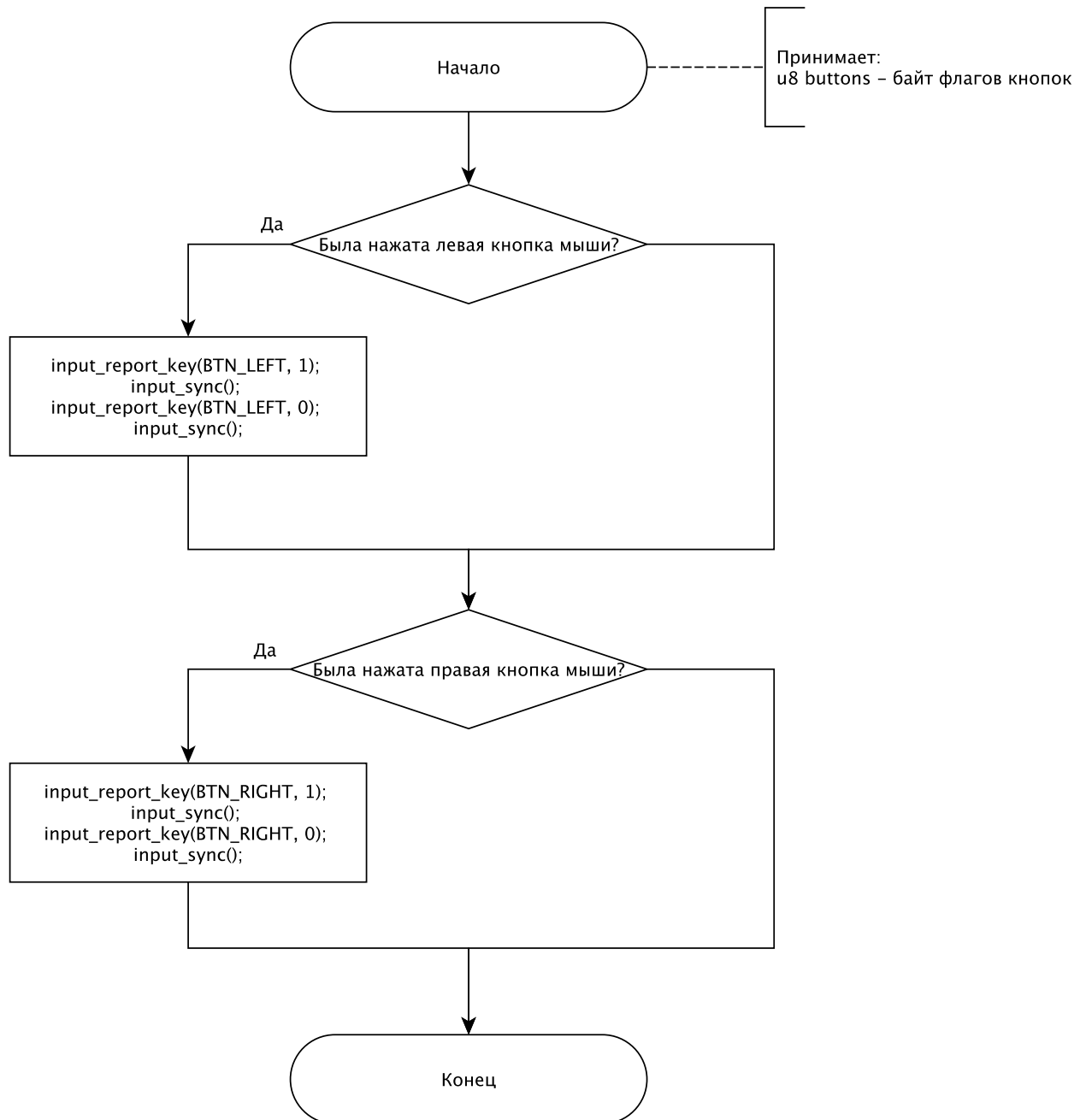


Рисунок 4 — Схема алгоритма обработки нажатий кнопок мыши `handle_buttons`

Обработка движения курсора реализована в функции `handle_movement`. Функция восстанавливает из четырёх байт 16-разрядные смещения по осям, масштабирует их с использованием коэффициента `speed_mult` в формате Q16.16 и, в зависимости от значения `interp_steps`, либо разбивает движение на несколько мелких шагов с генерацией последовательности событий `REL_X` и `REL_Y`, либо передаёт смещения единым событием. В обоих случаях каждое изменение сопровождается вызовом `input_sync` для фиксации событий подсистемой ввода. Алгоритм функции `handle_movement` представлен на рис. 5.

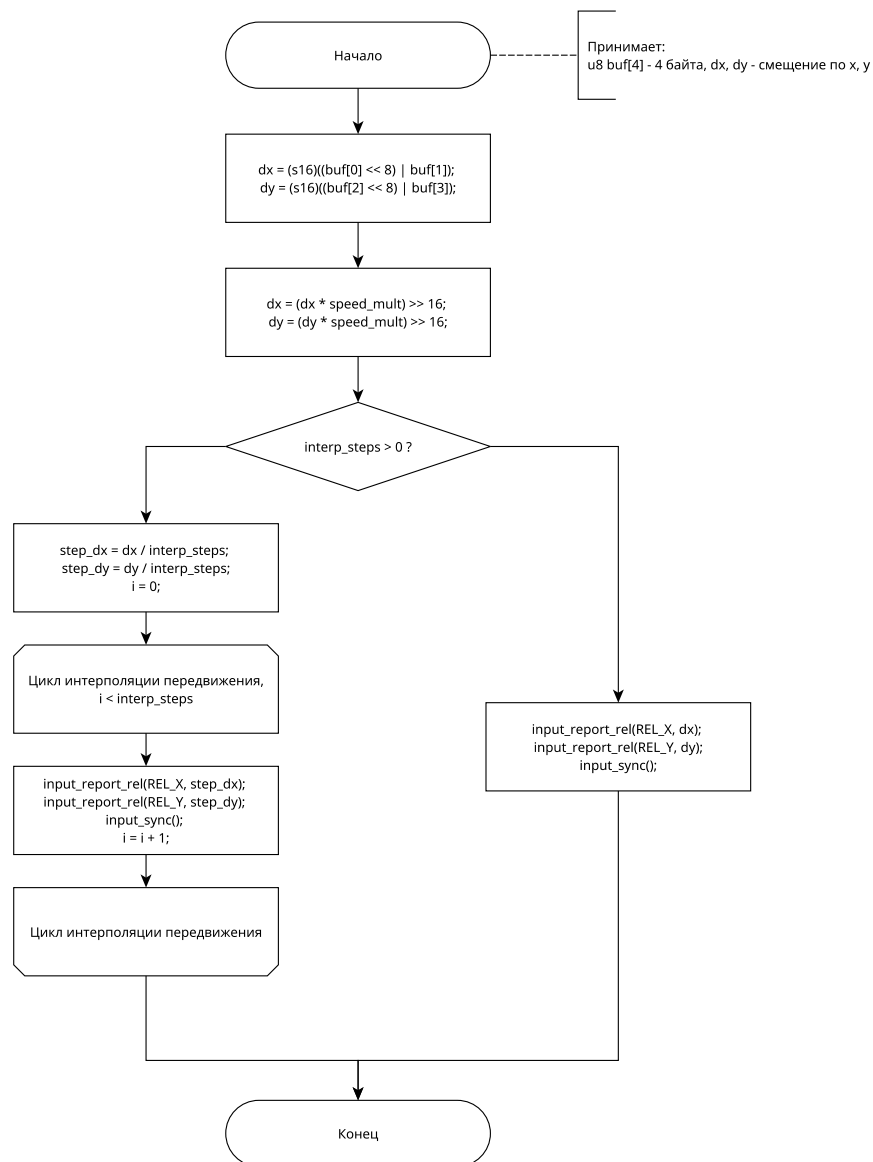


Рисунок 5 — Схема алгоритма обработки движения курсора `handle_movement`

Завершение работы модуля включает остановку служебного потока, закрытие клиентского и серверного RFCOMM-сокетов, снятие виртуального устройства с регистрации и освобождение ресурсов. Последовательность действий функции `pm_exit`, зарегистрированной через `module_exit`, приведена на рис. 6 [16; 17].

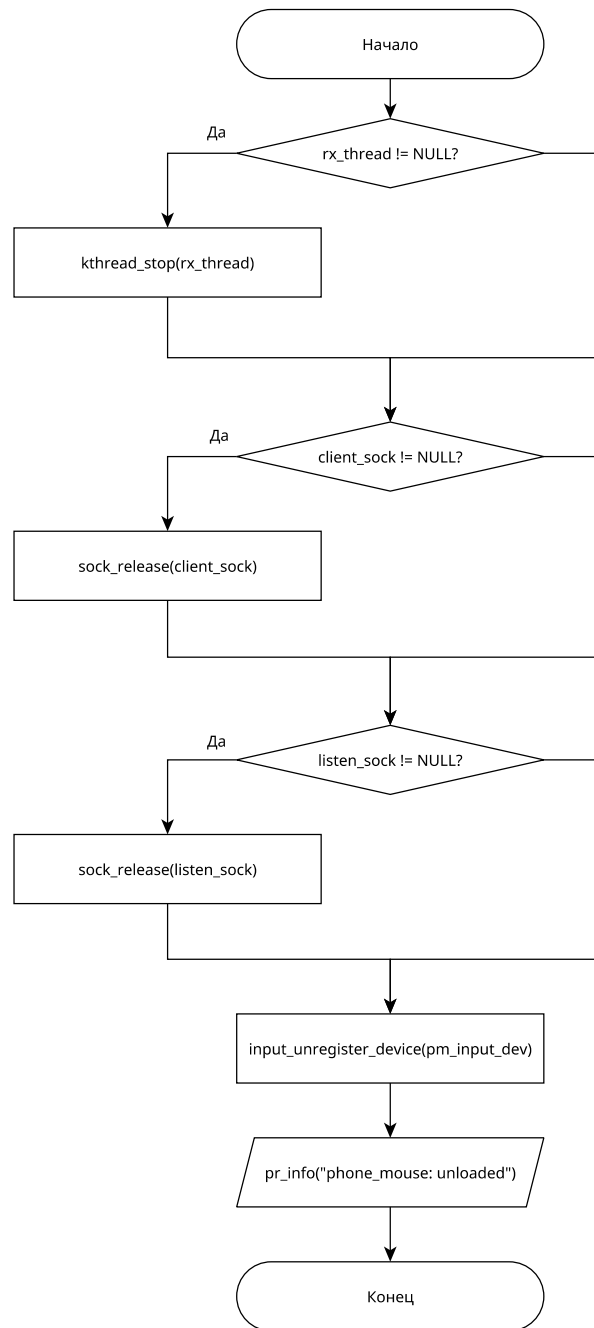


Рисунок 6 — Схема алгоритма завершения работы модуля ядра `phone_mouse_bt`

## 2.4 Последовательность работы мобильного приложения

Мобильное приложение реализовано в виде т. н. "Android-активности"(Activity), которая инициализирует пользовательский интерфейс, запускает фоновый поток отправки накопленных смещений курсора, устанавливает Bluetooth-соединение с рабочей станцией и обрабатывает события сенсорного экрана и нажатия кнопок мыши [6; 7]. Общая последовательность работы основной активности MainActivity представлена на рис. 7: при создании активности выполняется настройка интерфейса, запуск фонового потока отправки движения, восстановление сохранённого MAC-адреса, установка обработчиков для поля ввода MAC-адреса, кнопок мыши и области touchpad, а также, при наличии сохранённого адреса, иницируется подключение к рабочей станции.

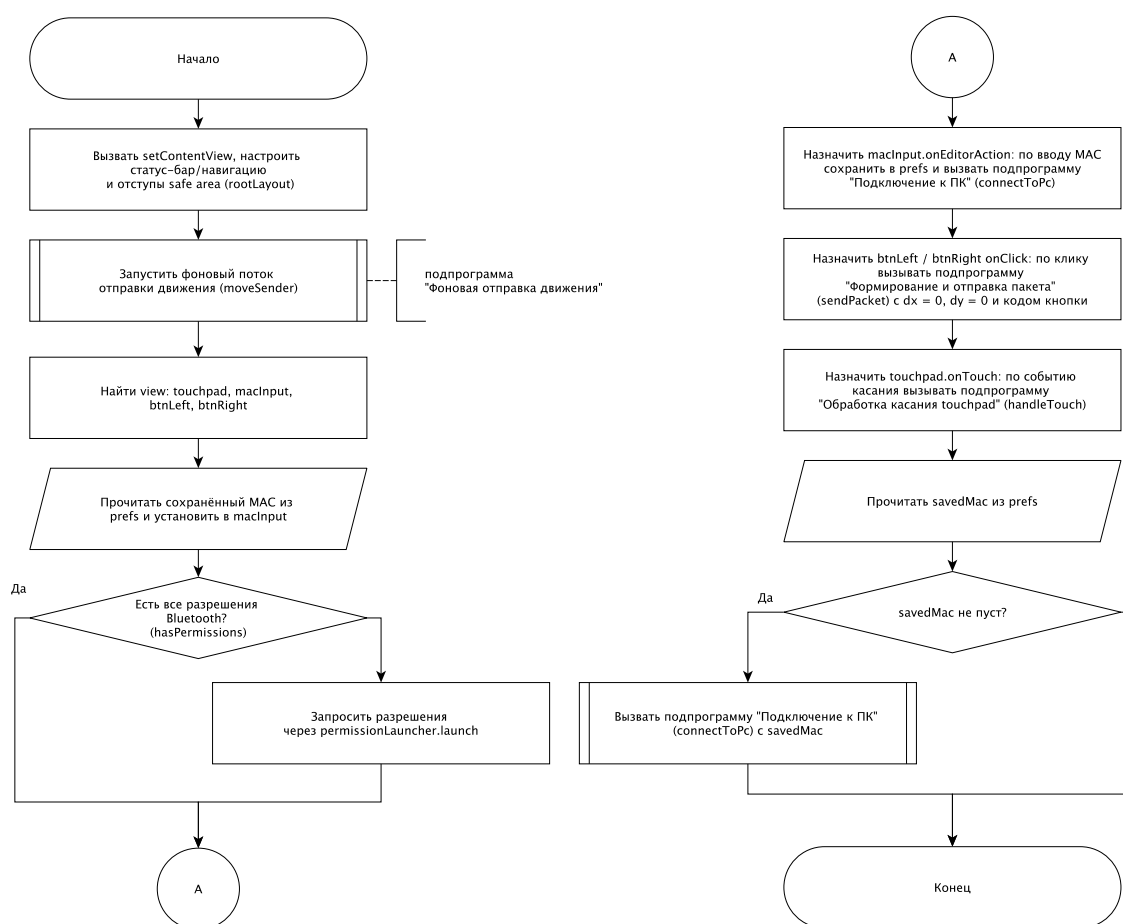


Рисунок 7 — Схема алгоритма работы основной активности мобильного приложения

Отправка накопленных смещений курсора реализована в отдельном потоке, который с фиксированным интервалом времени считывает значения переменных `pendingDx` и `pendingDy`, обнуляет накопленные значения и, при ненулевых смещениях, формирует пакет данных о смещении курсора, которая отправляется на устройство-обработчик с помощью вызова `sendPacket`. Алгоритм работы данного потока показан на рис. 8.

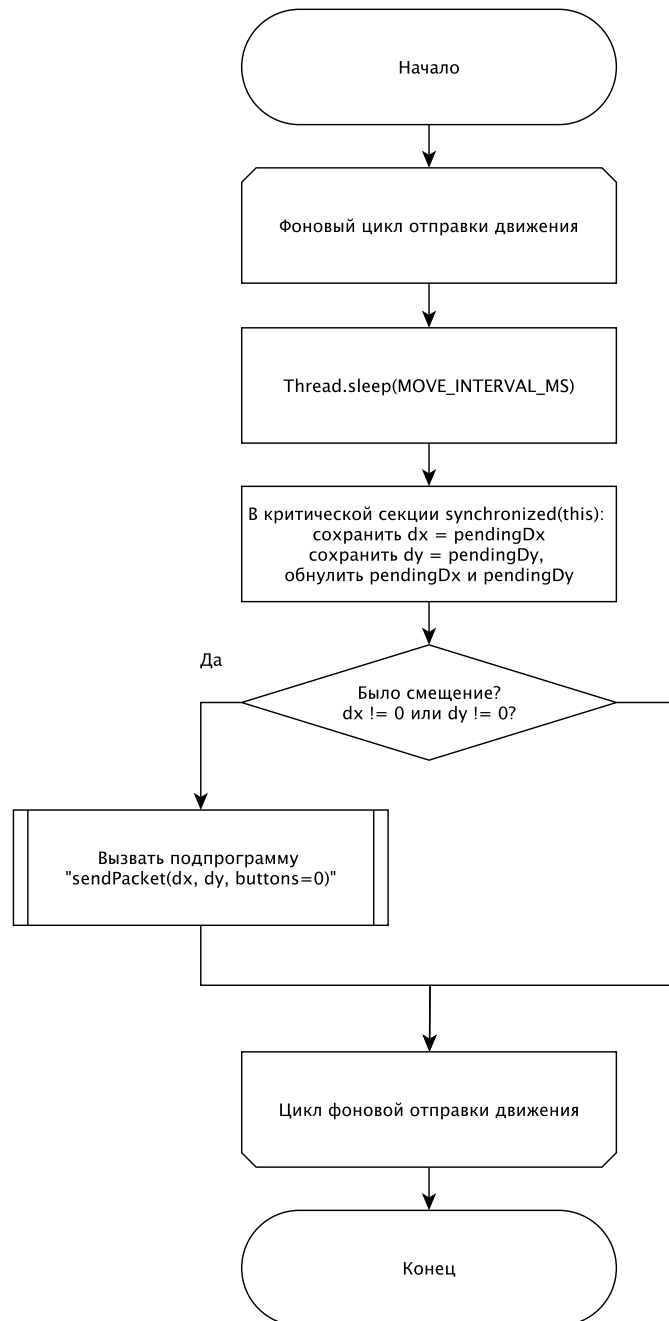


Рисунок 8 — Схема алгоритма фоновой отправки накопленных смещений

Функция `sendPacket` формирует бинарное сообщения протокола из смещений по осям и битовой маски кнопок мыши, проверяет наличие открытого выходного потока, упаковывает значения в массив из пяти байт и выполняет запись массива в `OutputStream`, игнорируя исключения ввода-вывода. Последовательность действий функции `sendPacket` представлена на рис. 9.

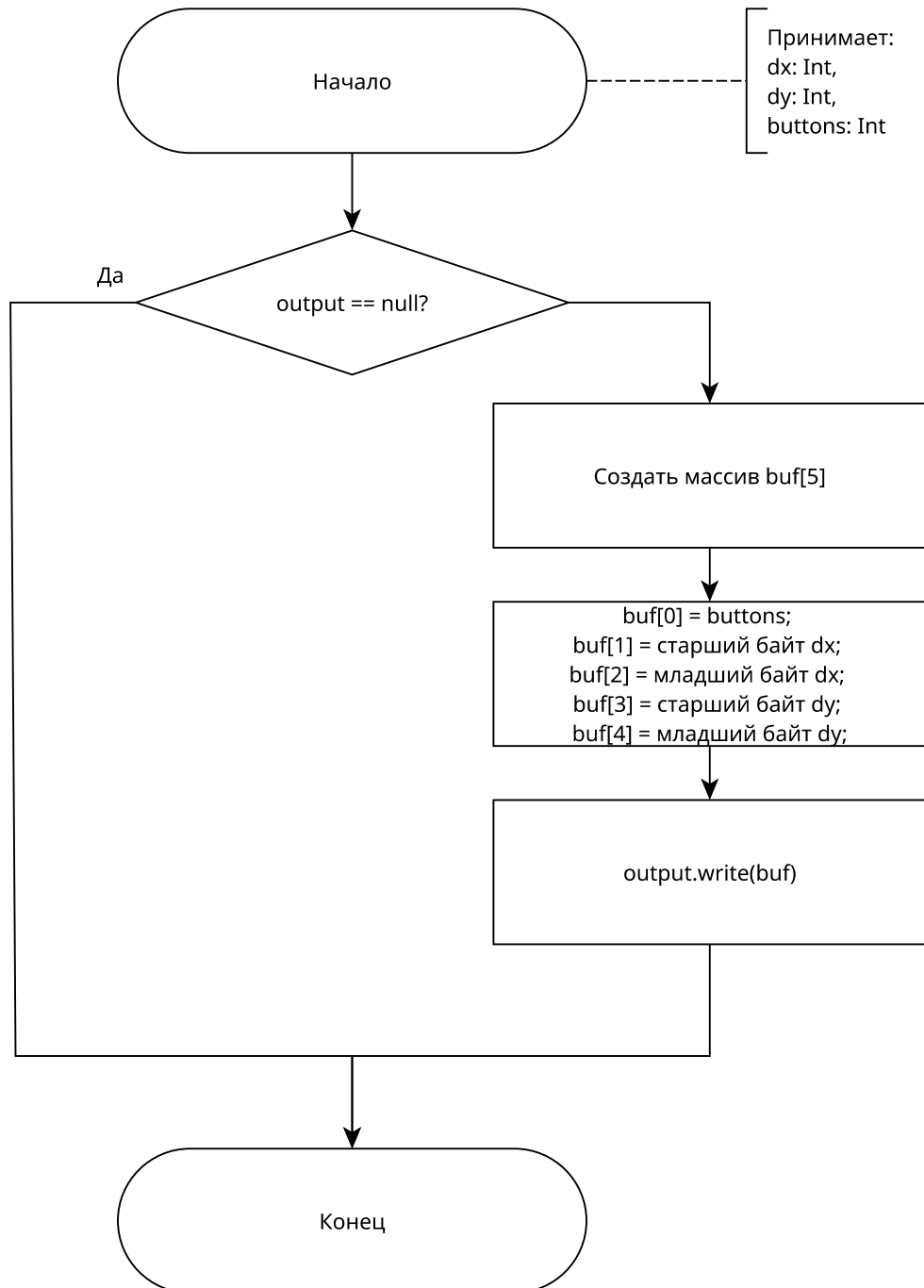


Рисунок 9 — Схема алгоритма формирования и отправки пакета команд `sendPacket`

Установление соединения с целевым устройством выполняется функцией `connectToPc`, которая проверяет корректность введённого MAC-адреса, получает адаптер Bluetooth, иницирует фоновый поток подключения, создаёт RFCOMM-сокеты к указанному каналу, выполняет соединение и сохраняет `BluetoothSocket` и поток вывода. В случае успеха и при ошибках пользователю отображаются диагностические сообщения. Общий алгоритм функции `connectToPc` показан на рис. 10, а внутренний поток подключения, использующий вызовы сокета и обработку исключений, представлен отдельно на рис. 11.

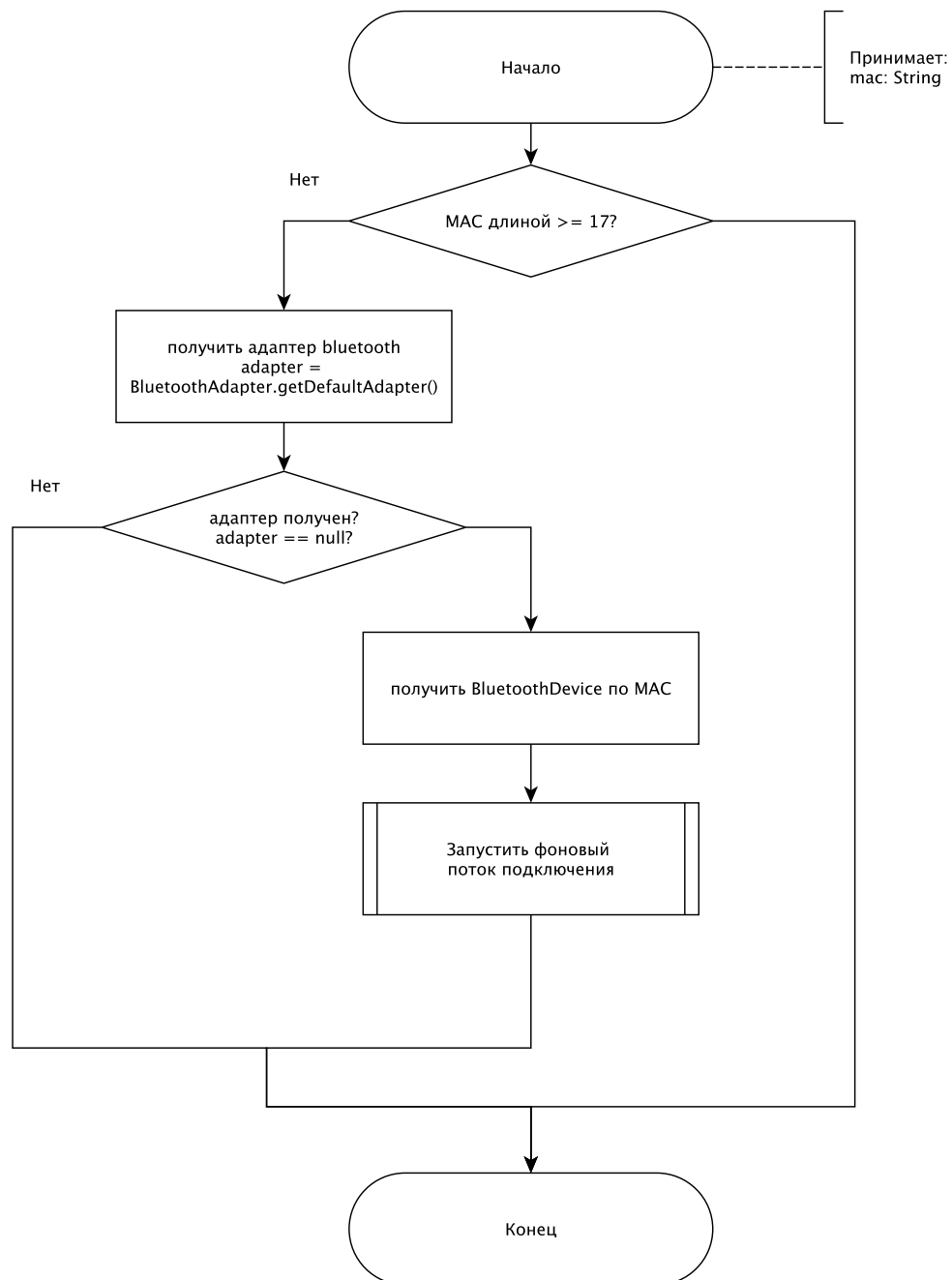


Рисунок 10 — Схема алгоритма функции подключения к рабочей станции `connectToPc`



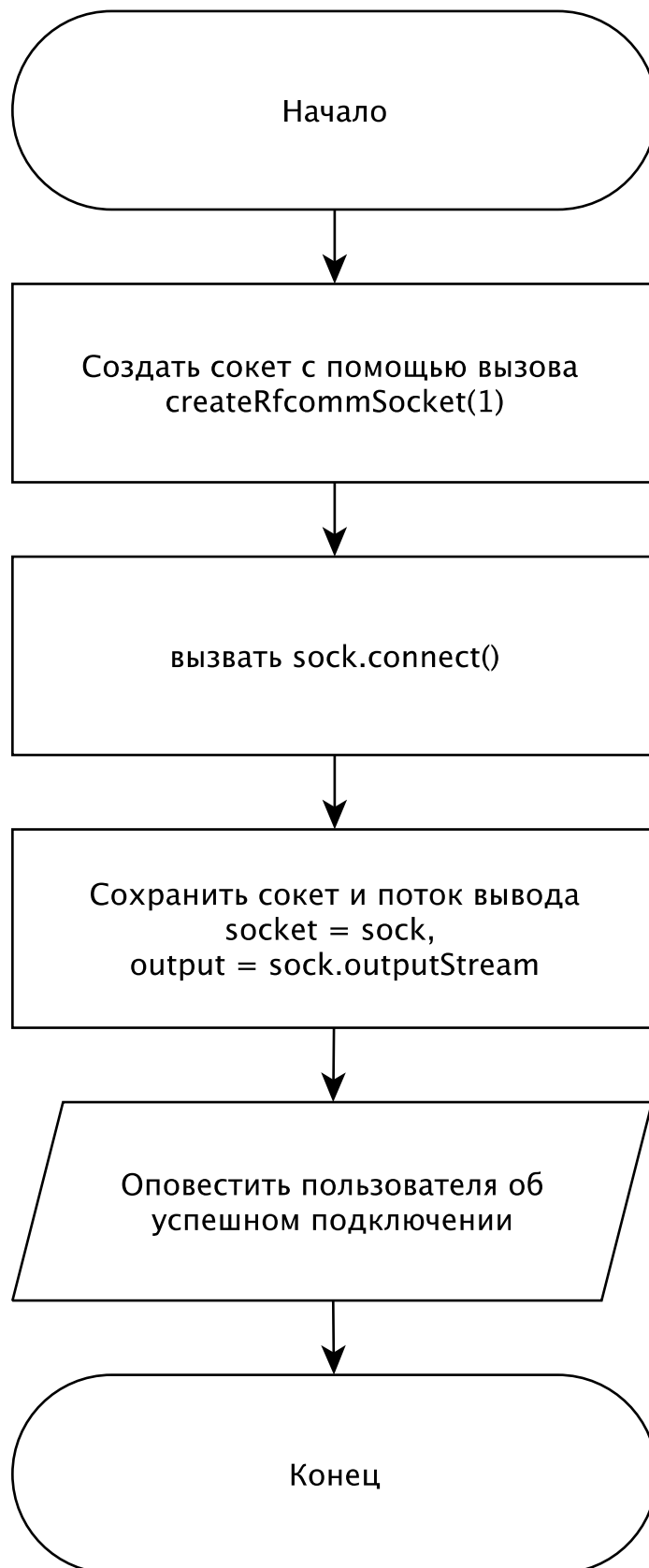


Рисунок 11 — Схема алгоритма внутреннего потока установления RFCOMM-соединения

Обработка движения пальца по сенсорной области реализована в функции `handleTouch`, которая по событию `ACTION_DOWN` запоминает исходные координаты и сбрасывает флаг первого движения, а по событиям `ACTION_MOVE` вычисляет относительные смещения по осям, обновляет координаты последнего касания и в критической секции увеличивает накопленные значения `pendingDx` и `pendingDy`. Эта логика отражена на рис. 12. Накопленные таким образом смещения затем периодически считываются фоновым потоком, изображённым на рис. 8, и передаются в драйвер.

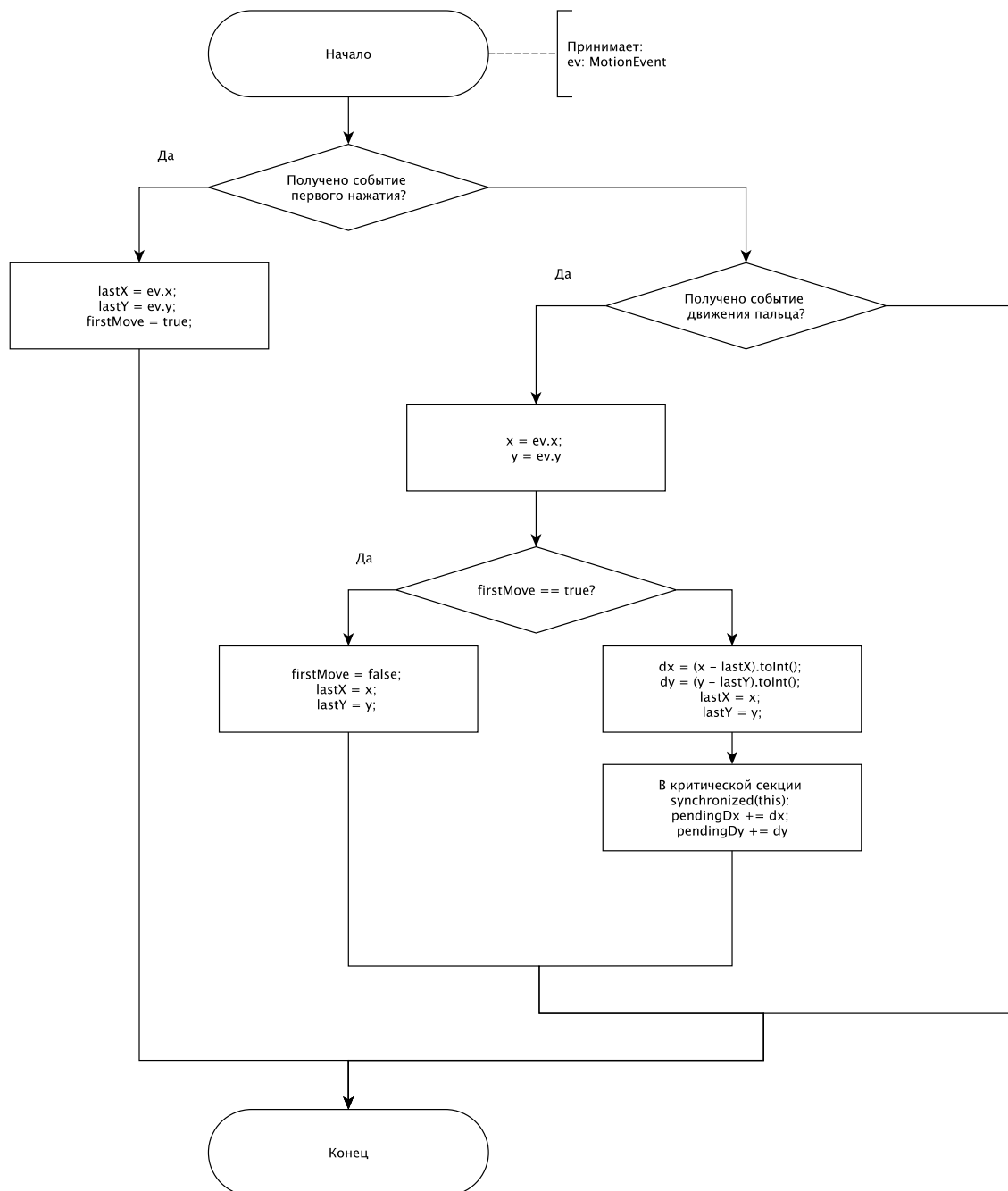


Рисунок 12 — Схема алгоритма обработки касаний на области touchpad `handleTouch`

Структура программного обеспечения изображена на рисунке 13

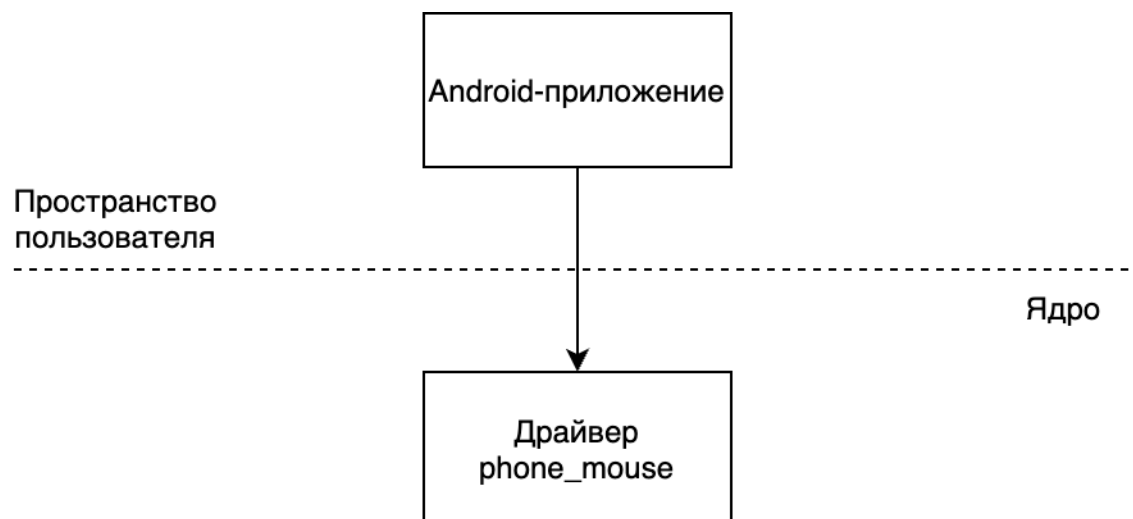


Рисунок 13 — Схема структуры ПО

## 3 Технологический раздел

### 3.1 Выбор языка и среды программирования

Загружаемый модуль ядра реализован на языке программирования C. Ядро Linux и большинство драйверов и загружаемых модулей традиционно реализуются на C, а интерфейсы ядровых API, включая подсистему ввода, работу с сокетами и потоками, определяются в терминах C-структур и функций [16; 17; 22]. Язык C предоставляет контроль над управлением памятью, представлением структур данных и взаимодействием с заголовочными файлами ядра, а также поддерживается стандартным инструментарием сборки модулей ядра на основе подсистемы Kbuild [16]. Таким образом, язык программирования C является достаточным для реализации драйвера.

Сборка модуля ядра выполняется с использованием стандартной инфраструктуры сборки ядра Linux и утилиты make. Для интеграции с подсистемой Kbuild используется конфигурационный файл Makefile, в котором описаны цели сборки, имя модуля и перечень исходных файлов [16].

Мобильное приложение разработано на языке Kotlin. Этот язык официально поддерживается в экосистеме Android и интегрирован с Android Studio и системой сборки Gradle, что обеспечивает доступ к Android SDK и библиотекам платформы, включая API Bluetooth [6; 23]. Совместимость с Java-API позволяет использовать классы BluetoothAdapter, BluetoothDevice и BluetoothSocket непосредственно из Kotlin-кода [6; 7]. Таким образом, Kotlin обладает достаточными возможностями для реализации клиентского приложения, а также поддержания связи с модулем ядра на целевом устройстве.

### 3.2 Реализация загружаемого модуля ядра Linux

#### Структуры данных и состояние драйвера

Загружаемый модуль ядра хранит состояние виртуального устройства мыши и параметры протокола обмена в собственных структурах данных. Основой для интеграции с подсистемой ввода является структура `struct input_dev` (представлена в листинге 1), которая содержит идентификаторы устройства, указатели на функции обратного вызова и описания поддерживаемых типов и кодов событий [17; 18]. Дополнительно используются структуры для хранения текущих состояний кнопок и параметров RFCOMM-соединения. Фрагмент объявления структур данных драйвера приведён в листинге 2.

Листинг 1 — Структура `input_dev`

```
struct input_dev {  
    const char * name;  
    const char * phys;  
    const char * uniq;
```

```

struct input_id id;
unsigned long propbit;
unsigned long evbit;
unsigned long keybit;
unsigned long relbit;
unsigned long absbit;
unsigned long mscbit;
unsigned long ledbit;
unsigned long sndbit;
unsigned long ffbbit;
unsigned long swbit;
unsigned int hint_events_per_packet;
unsigned int keycodemax;
unsigned int keycodesize;
void * keycode;
int (* setkeycode) (struct input_dev *dev, const struct
    input_keymap_entry *ke, unsigned int *old_keycode);
int (* getkeycode) (struct input_dev *dev, struct input_keymap_entry *
    ke);
struct ff_device * ff;
unsigned int repeat_key;
struct timer_list timer;
int rep;
struct input_mt * mt;
struct input_absinfo * absinfo;
unsigned long key;
unsigned long led;
unsigned long snd;
unsigned long sw;
int (* open) (struct input_dev *dev);
void (* close) (struct input_dev *dev);
int (* flush) (struct input_dev *dev, struct file *file);
int (* event) (struct input_dev *dev, unsigned int type, unsigned int
    code, int value);
struct input_handle __rcu * grab;
spinlock_t event_lock;
struct mutex mutex;
unsigned int users;
bool going_away;
struct device dev;
struct list_head h_list;

```

```

    struct list_head node;
    unsigned int num_vals;
    unsigned int max_vals;
    struct input_value * vals;
    bool devres_managed;
};

```

Листинг 2 — Фрагмент объявления структур данных драйвера phone\_mouse\_bt

```

/* Виртуальное устройство мыши в подсистеме input */
static struct input_dev *pm_input_dev;

/* RFCOMM-сокеты для ожидания и обслуживания соединения */
static struct socket *listen_sock;
static struct socket *client_sock;

/* Служебный поток приёма пакетов от телефона */
static struct task_struct *rx_thread;

/* Параметры обработки движения */
static int interp_steps = 0; /* число шагов интерполяции движения */
static int speed_mult;      /* коэффициент скорости в формате Q16.16 */
/*

/* Маски кнопок в протольном байте buttons */
#define LMB_MASK 0x01
#define RMB_MASK 0x02

```

## Точки входа модуля и функции обработки

Точки входа загружаемого модуля определяются функциями инициализации и завершения, регистрируемыми макросами `module_init` и `module_exit` [16]. В функции инициализации выполняются:

- создание и настройка объекта `struct input_dev` и регистрация виртуального устройства мыши в подсистеме ввода;
- инициализация структур данных состояния драйвера;
- создание серверного RFCOMM-сокета и его привязка к выбранному каналу;
- запуск служебного потока ядра с помощью `kthread_run` для обработки соединения и входящих команд [5; 21].

Функция завершения выполняет остановку служебного потока, закрытие RFCOMM-сокета, снятие виртуального устройства с регистрации и освобождение всех выделенных ресурсов [16].

Функции инициализации и завершения драйвера модуля приведены в листингах 3 и 4 соответственно.

Листинг 3 — Функция инициализации модуля

```
static int __init pm_init(void) {
    int err;
    struct sockaddr_rc addr = {0};

    if (interp_steps < 0) {
        pr_err("phone_mouse_bt: ERROR: interp_steps must be >= 0 (got %d)\n",
            ,
            interp_steps);
        return -EINVAL;
    }

    speed_mult = (speed_pct * 65536) / 100;
    pr_info("phone_mouse_bt: speed coefficient = %d (Q16.16)\n",
        speed_mult);

    // Allocate new input device
    pm_input_dev = input_allocate_device();
    if (!pm_input_dev)
        return -ENOMEM;

    pm_input_dev->name = "Bluetooth Phone Mouse";
    pm_input_dev->id.bustype = BUS_BLUETOOTH;

    __set_bit(EV_KEY, pm_input_dev->evbit);
    __set_bit(EV_REL, pm_input_dev->evbit);

    __set_bit(BTN_LEFT, pm_input_dev->keybit);
    __set_bit(BTN_RIGHT, pm_input_dev->keybit);

    __set_bit(REL_X, pm_input_dev->relbit);
    __set_bit(REL_Y, pm_input_dev->relbit);

    err = input_register_device(pm_input_dev);
    if (err)
        return err;

    // RFCOMM socket
    err = sock_create_kern(&init_net, PF_BLUETOOTH, SOCK_STREAM,
```

```

        BTPROTO_RFCOMM,
&listen_sock);
if (err < 0) {
    pr_err("phone_mouse: sock_create_kern failed\n");
    return err;
}

addr.rc_family = AF_BLUETOOTH;
bacpy(&addr.rc_bdaddr, BDADDR_ANY);
addr.rc_channel = bt_listen_channel;

err = listen_sock->ops->bind(listen_sock, (struct sockaddr *)&addr,
sizeof(addr));
if (err < 0) {
    pr_err("phone_mouse: bind failed\n");
    return err;
}

err = listen_sock->ops->listen(listen_sock, 1);
if (err < 0) {
    pr_err("phone_mouse: listen failed\n");
    return err;
}

// Main loop in kernel thread
rx_thread = kthread_run(rx_loop, NULL, "phone_mouse_rx");
if (IS_ERR(rx_thread)) {
    pr_err("phone_mouse: failed to start thread\n");
    return PTR_ERR(rx_thread);
}

pr_info("phone_mouse: module loaded, listening RFCOMM channel %d\n",
bt_listen_channel);

return 0;
}

```

Листинг 4 — Функция завершения модуля

```

static void __exit pm_exit(void) {
    if (rx_thread)
        kthread_stop(rx_thread);
}

```



```

    if (client_sock)
        sock_release(client_sock);

    if (listen_sock)
        sock_release(listen_sock);

    input_unregister_device(pm_input_dev);

    pr_info("phone_mouse: unloaded\n");
}

```

Служебный поток `rx_loop` реализует цикл приёма данных из RFCOMM-сокета: при отсутствии активного клиента поток ожидает входящего соединения, после установления соединения периодически читает из сокета фиксированный пакет длиной пять байт, обрабатывает ситуации временного отсутствия данных и разрыва соединения и, для корректных пакетов, передаёт первый байт в функцию `handle_buttons`, а четыре следующих байта — в функцию `handle_movement`. Функция `handle_buttons` на основе битовой маски кнопок порождает события `EV_KEY`, а функция `handle_movement` декодирует смещения по осям, масштабирует их и генерирует соответствующие события `EV_REL` с последующим вызовом `input_sync` [1; 3; 17; 18].

Листинг 5 — Функция служебного потока приёма пакетов по RFCOMM

```

static int rx_loop(void *data) {
    struct msghdr msg = {0};
    struct kvec vec;
    u8 buf[5];

    while (!kthread_should_stop()) {

        if (!client_sock) {
            /* Ждём подключения */
            struct socket *new_sock = NULL;
            int r = kernel_accept(listen_sock, &new_sock, 0);
            if (r == 0) {
                client_sock = new_sock;
                pr_info("phone_mouse: client connected!\n");
            } else {
                ssleep(1);
                continue;
            }
        }
    }
}

```

```

vec.iov_base = buf;
vec.iov_len = sizeof(buf);

int len =
kernel_recvmmsg(client_sock, &msg, &vec, 1, sizeof(buf), MSG_DONTWAIT
);

if (len == -EAGAIN) {
    msleep(5);
    continue;
}
if (len <= 0) {
    pr_info("phone_mouse: client disconnected\n");
    sock_release(client_sock);
    client_sock = NULL;
    continue;
}
if (len < 5)
    continue;

handle_buttons(buf[0]);

handle_movement(&buf[1]);
}

return 0;
}

```

Листинг 6 — Функция обработки нажатия кнопки

```

#define LMB_MASK 0b00000001
#define RMB_MASK 0b00000010
static void handle_buttons(u8 buttons) {
    if (buttons & LMB_MASK) {
        input_report_key(pm_input_dev, BTN_LEFT, 1);
        input_sync(pm_input_dev);
        input_report_key(pm_input_dev, BTN_LEFT, 0);
        input_sync(pm_input_dev);
    }

    if (buttons & RMB_MASK) {
        input_report_key(pm_input_dev, BTN_RIGHT, 1);
        input_sync(pm_input_dev);
    }
}

```

```

        input_report_key(pm_input_dev, BTN_RIGHT, 0);
        input_sync(pm_input_dev);
    }
}

```

Листинг 7 — Функция обработки движения курсора

```

static void handle_movement(u8 buf[4]) {
    s16 dx = (s16)((buf[0] << 8) | buf[1]);
    s16 dy = (s16)((buf[2] << 8) | buf[3]);

    dx = (dx * speed_mult) >> 16;
    dy = (dy * speed_mult) >> 16;

    if (interp_steps > 0) {
        int step_dx = dx / interp_steps;
        int step_dy = dy / interp_steps;

        int i;
        for (i = 0; i < interp_steps; i++) {
            input_report_rel(pm_input_dev, REL_X, step_dx);
            input_report_rel(pm_input_dev, REL_Y, step_dy);
            input_sync(pm_input_dev);
        }

        return;
    }

    input_report_rel(pm_input_dev, REL_X, dx);
    input_report_rel(pm_input_dev, REL_Y, dy);

    input_sync(pm_input_dev);
}

```

## Сборка модуля ядра

Сборка модуля ядра выполняется внешней по отношению к дереву исходных текстов ядра командой `make` с использованием файла `Makefile`, описывающего цель сборки и исходные файлы. В `Makefile` используются переменные и правила `Kbuild`, что позволяет компилировать модуль в соответствии с конфигурацией ядра и подключать необходимые заголовочные файлы [16].

Листинг 8 — Фрагмент файла `Makefile` для сборки модуля ядра

```

obj-m += phone_mouse_bt.o

KDIR := /lib/modules/$(shell uname -r)/build

PWD := $(shell pwd)

all:
$(MAKE) -C $(KDIR) M=$(PWD) modules

clean:
$(MAKE) -C $(KDIR) M=$(PWD) clean

```

### 3.3 Реализация мобильного приложения для Android

#### Основные компоненты приложения

Мобильное приложение реализовано в виде Android-приложения с основной активностью, отвечающей за инициализацию пользовательского интерфейса, установление Bluetooth-соединения и обработку входных событий. Логика обработки касаний и генерации команд размещается в коде активности и связанных с ней обработчиков событий, а обмен данными по Bluetooth — в отдельном компоненте, использующем объект BluetoothSocket [6; 7].

Листинг 9 — Фрагмент основной активности Android-приложения

```

class MainActivity : AppCompatActivity() {

    private val prefs by lazy { getSharedPreferences("btmouse",
        MODE_PRIVATE) }

    private var socket: BluetoothSocket? = null
    private var output: OutputStream? = null

    private var lastX = 0f
    private var lastY = 0f
    private var firstMove = true

    private var pendingDx = 0
    private var pendingDy = 0
    private val MOVE_INTERVAL_MS = 15L

    @Volatile private var moveSenderRunning = true

```

```

private val btPermissions = arrayOf(
    Manifest.permission.BLUETOOTH_CONNECT,
    Manifest.permission.BLUETOOTH_SCAN
)

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    // Фоновый поток отправки накопленных смещений курсора
    thread {
        while (moveSenderRunning) {
            Thread.sleep(MOVE_INTERVAL_MS)

            val dx: Int
            val dy: Int

            synchronized(this) {
                dx = pendingDx
                dy = pendingDy
                pendingDx = 0
                pendingDy = 0
            }

            if (dx != 0 || dy != 0) {
                sendPacket(dx, dy, 0)
            }
        }
    }

    // Поиск представлений интерфейса
    val touchpad = findViewById<View>(R.id.touchpad)
    val macInput = findViewById<EditText>(R.id.macInput)
    val btnLeft = findViewById<Button>(R.id.btnLeft)
    val btnRight = findViewById<Button>(R.id.btnRight)

    // Восстановление сохранённого MAC-адреса
    macInput.setText(prefs.getString("mac", ""))

    if (!hasPermissions()) {
        permissionLauncher.launch(btPermissions)
    }
}

```

```

    }

    // Подключение при вводе нового MAC-адреса
    macInput.setOnEditorActionListener { _, _, _ ->
        val mac = macInput.text.toString().trim()
        prefs.edit().putString("mac", mac).apply()
        connectToPc(mac)
        true
    }

    // Обработчики нажатий кнопок мыши
    btnLeft.setOnClickListener {
        sendPacket(0, 0, 1) // левый клик
    }
    btnRight.setOnClickListener {
        sendPacket(0, 0, 2) // правый клик
    }

    // Обработка движения по сенсорной области
    touchpad.setOnTouchListener { _, ev ->
        handleTouch(ev)
        true
    }

    // Автоматическое подключение при сохранённом MAC
    val savedMac = prefs.getString("mac", "")
    if (!savedMac.isNullOrEmpty()) {
        connectToPc(savedMac)
    }
}

// ...
}

```

## Работа с Bluetooth-API Android

Для установления RFCOMM-соединения приложение использует BluetoothAdapter для доступа к локальному Bluetooth-модулю, BluetoothDevice для представления удалённого устройства рабочей станции и BluetoothSocket для установления и использования сокетного соединения [6; 7]. Соединение создаётся методом createRfcommSocketToServiceRecord с использованием согласованного UUID сервиса, после чего приложение выполняет операцию подключения и

получает потоки ввода-вывода для обмена бинарными сообщениями.

#### Листинг 10 — Фрагмент работы с BluetoothAdapter и BluetoothSocket

```
private fun hasPermissions(): Boolean =
    btPermissions.all {
        ContextCompat.checkSelfPermission(this, it) ==
            PackageManager.PERMISSION_GRANTED
    }

private fun connectToPc(mac: String) {
    if (mac.length < 17) {
        Toast.makeText(this, "Invalid MAC", Toast.LENGTH_SHORT).show()
        return
    }

    val adapter = BluetoothAdapter.getDefaultAdapter() ?: return

    try {
        val device: BluetoothDevice = adapter.getRemoteDevice(mac)

        // Подключение к ПК во внутреннем потоке
        thread {
            try {
                // RFCOMM channel 1 (драйвер слушает на этом канале)
                val method = device.javaClass.getMethod(
                    "createRfcommSocket",
                    Int::class.javaPrimitiveType
                )
                val sock = method.invoke(device, 1) as BluetoothSocket

                adapter.cancelDiscovery()
                sock.connect()

                socket = sock
                output = sock.getOutputStream

                runOnUiThread {
                    Toast.makeText(
                        this,
                        "Connected to $mac",
                        Toast.LENGTH_SHORT
                    ).show()
                }
            }
        }
    }
}
```

```

    }
    } catch (e: Exception) {
        e.printStackTrace()
        runOnUiThread {
            Toast.makeText(
                this,
                "Connect err: ${e.message}",
                Toast.LENGTH_LONG
            ).show()
        }
    }
}
} catch (e: Exception) {
    e.printStackTrace()
    Toast.makeText(
        this,
        "MAC error: ${e.message}",
        Toast.LENGTH_LONG
    ).show()
}
}
}

```

Код обработки сенсорных событий в пользовательском интерфейсе преобразует координаты касаний и состояния элементов управления в смещения по осям и битовую маску состояний кнопок мыши, после чего упаковывает их в бинарный формат протокола и передаёт через BluetoothSocket в модуль ядра [6; 7].

## Конфигурация проекта и разрешения

Конфигурация Android-приложения включает файл `AndroidManifest.xml`, в котором описываются разрешения на использование Bluetooth и, при необходимости, дополнительные особенности аппаратной платформы. В манифесте объявляется основная активность приложения и настраиваются параметры, связанные с версией SDK и требованиями к окружению [6].

Листинг 11 — Фрагмент файла `AndroidManifest.xml` с разрешениями Bluetooth

```

<!-- Разрешения Bluetooth для Android 12+ и Android 15 -->
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
<uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
<uses-permission android:name="android.permission.BLUETOOTH_ADVERTISE" />

<application
    android:allowBackup="true"

```



```
    android:label="PhoneMouse"
    android:supportsRtl="true"
    android:theme="@style/Theme.AppCompat.Light.NoActionBar">

    <activity
        android:name=".MainActivity"
        android:exported="true">

        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>

    </activity>
</application>
```

## **4 Исследовательский раздел**

### **4.1 Технические характеристики**

Проверка работы разработанного программного обеспечения выполнялась на стенде, включающем рабочую станцию под управлением операционной системы семейства Linux и смартфон под управлением Android. Конфигурация стенда приведена ниже.

#### **Рабочая станция**

В качестве рабочей станции использован компьютер со следующими характеристиками:

- операционная система: Arch Linux x86\_64;
- ядро: Linux 6.17.9-zen1-1-zen;
- объём оперативной памяти: 32 Гб;
- процессор: AMD Ryzen 7 7700 (16) @ 5.39 ГГц;
- графический процессор №1: NVIDIA GeForce RTX 3080 Ti [Дискретный];
- графический процессор №2: AMD Raphael [Интегрированный];
- встроенный Bluetooth-адаптер, поддерживающий профили, реализуемые стеком BlueZ (RFCOMM поверх L2CAP);

Указанная конфигурация достаточна для загрузки загружаемого модуля ядра, регистрации виртуального устройства ввода типа «мышь» и приёма событий по RFCOMM-соединению от мобильного приложения.

#### **Мобильное устройство**

В качестве мобильного устройства использован смартфон со следующими характеристиками:

- операционная система: Android 16;
- модуль Bluetooth, поддерживающий работу в режиме классического Bluetooth и установление RFCOMM-соединений;
- сенсорный экран с поддержкой многоточечного ввода;
- объём оперативной памяти: 12 Гб;
- процессор: Google Tensor G3;

Выбранная конфигурация обеспечивает установление RFCOMM-соединения со стороны Android-приложения, обработку жестов на сенсорном экране и формирование бинарных сообщений протокола для передачи в драйвер ядра.

### **4.2 Демонстрация работы программы**

Демонстрация работы разработанного решения проводилась в виде набора экспериментальных сценариев, охватывающих установление соединения, управление курсором и генера-

цию событий нажатия кнопок мыши.

Перед запуском мобильного приложения на рабочей станции выполнялась загрузка загружаемого модуля ядра `phone_mouse_bt`. При загрузке модуль регистрировал виртуальное устройство ввода в подсистеме `input`, создавал серверный RFCOMM-сокет и запускал служебный поток `rx_loop`, ожидающий входящего соединения от смартфона. В системном журнале ядра фиксировалось сообщение о готовности модуля и номере прослушиваемого RFCOMM-канала.

На стороне мобильного устройства запускалось Android-приложение. Пользователь вводил MAC-адрес Bluetooth-адаптера рабочей станции в соответствующее текстовое поле и инициировал подключение. Приложение получало объект `BluetoothDevice` по указанному MAC-адресу, создавалось RFCOMM-соединение с указанным каналом, после успешного выполнения операции `connect` сохранялись объекты `BluetoothSocket` и `OutputStream`, и на экране отображалось уведомление об успешном подключении.

Графический интерфейс мобильного приложения содержит:

- область `touchpad`, в которой обрабатываются жесты перемещения пальца и вычисляются относительные смещения по осям;
- две кнопки, инициирующие логические события нажатия левой и правой кнопок мыши;
- поле ввода MAC-адреса рабочего устройства.

Интерфейс мобильного приложения показан на рис. 14. Сенсорная область занимает центральную часть экрана, кнопки мыши расположены в нижней части, поле ввода MAC-адреса — в верхней части интерфейса.

При перемещении пальца пользователя по области `touchpad` приложение фиксирует координаты касания, вычисляет относительные смещения по осям  $dx$  и  $dy$  и накапливает их во внутренних переменных `pendingDx` и `pendingDy`. В отдельном фоновом потоке с фиксированным интервалом времени выполняется чтение накопленных смещений под защитой синхронизации, после чего при ненулевом значении хотя бы одной из компонент формируется бинарный пакет из пяти байт (код кнопок и смещения по осям) и передаётся в драйвер через `BluetoothSocket`.

При нажатии на кнопку, соответствующую левой или правой кнопке мыши, приложение вызывает функцию формирования пакета с нулевыми смещениями и установленным кодом нужной кнопки. В результате в драйвер ядра поступает пакет с соответствующей маской кнопок, а функция `handle_buttons` генерирует последовательность событий `BTN_LEFT` или `BTN_RIGHT`. На стороне рабочей станции это проявляется в виде стандартных действий графической среды: выделения объектов, вызова контекстного меню и других операций, связанных с нажатием соответствующих кнопок мыши.

В ходе демонстрации проверялись следующие сценарии:

- установление и закрытие RFCOMM-соединения при корректном и некорректном MAC-адресе;
- устойчивое перемещение курсора по экрану рабочей станции при различных траекториях движения пальца по сенсорной области;

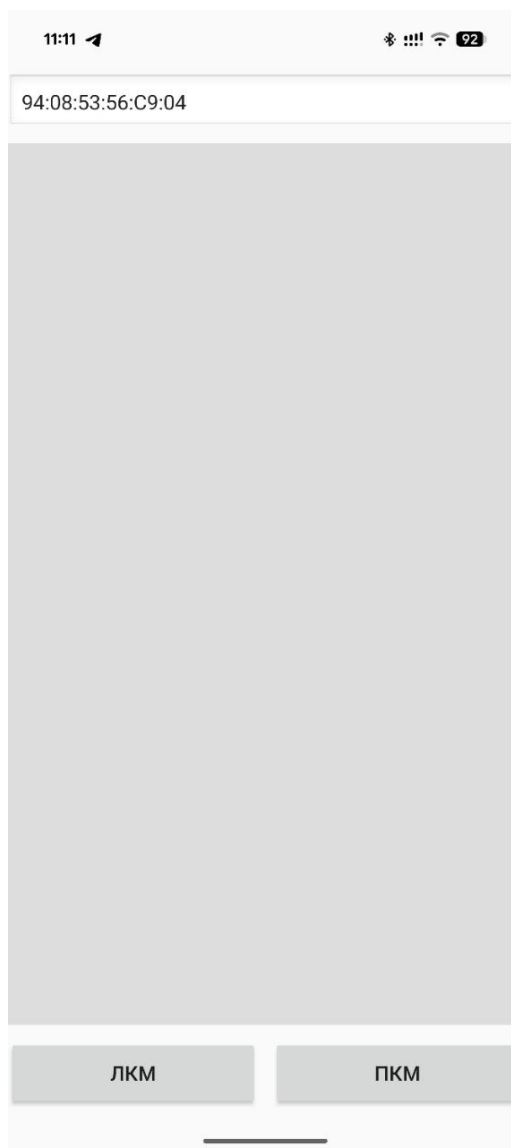


Рисунок 14 — Интерфейс мобильного приложения управления курсором

- корректная генерация событий нажатия левой и правой кнопок мыши и их обработка оконной системой;
- работа системы при изменении параметров скорости и числа шагов интерполяции движения, задаваемых параметрами модуля.

Во всех указанных сценариях виртуальное устройство мыши, регистрируемое модулем ядра, корректно обрабатывало события, поступающие от мобильного приложения, а взаимодействие с графической средой рабочей станции происходило через стандартный стек ввода операционной системы. Набор сценариев демонстрирует соответствие ключевым требованиям ТЗ: управление курсором и кнопками с телефона по Bluetooth посредством загружаемого модуля ядра.

## ЗАКЛЮЧЕНИЕ

В аналитическом разделе выполнен анализ способов генерации событий ввода в ядре Linux и вариантов организации обмена данными между мобильным устройством и рабочей станцией. Рассмотрены подходы к регистрации виртуального устройства в подсистеме input, использованию подсистемы uinput и интеграции через подсистему HID. В результате анализа установлено, что регистрация виртуального устройства типа «мышь» в подсистеме input обеспечивает формирование событий EV\_REL и EV\_KEY непосредственно в модуле ядра, не требует переноса логики в пользовательское пространство и не влечёт усложнения реализации за счёт поддержки HID-протокола. Данный способ выбран в качестве базового для реализации драйвера.

Также проанализированы основные варианты беспроводного обмена данными по каналу Bluetooth: прямое использование протокола L2CAP, применение профиля HID, обработка RFCOMM-соединений через устройства /dev/rfcommN в пользовательском пространстве и организация серверного RFCOMM-сокета в пространстве ядра. Показано, что использование RFCOMM-сокета непосредственно в модуле ядра обеспечивает потоковый двунаправленный канал передачи данных, позволяет передавать управляющие команды без промежуточных пользовательских прослоек и естественно сочетается с API Bluetooth операционной системы Android. На основе проведённого анализа выбран механизм обмена данными с использованием протокола RFCOMM.

На основании результатов аналитического раздела сформирована целевая архитектура программного комплекса, включающая связку «Android-приложение — RFCOMM — загружаемый модуль ядра — подсистема input». Данная архитектура положена в основу дальнейшей разработки и реализации программного решения.

В рамках работы разработаны алгоритмы функционирования загружаемого модуля ядра и мобильного приложения, реализованы приём и обработка управляющих сообщений, а также генерация событий подсистемы ввода для виртуального устройства мыши. В результате исследования подтверждена работоспособность разработанного ПО и возможность устойчивого управления курсором рабочей станции с использованием мобильного устройства по беспроводному каналу связи. Разработанное ПО полностью соответствует техническому заданию.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Bluetooth Core Specification. — accessed 15.02.2025. <https://www.bluetooth.com/specifications/bluetooth-core-specification/>.
2. L2CAP. — 2024. — <https://github.com/bluez/bluez/wiki/L2CAP>, accessed 15.02.2025. BlueZ Wiki.
3. BlueZ rfcomm Wiki. — accessed 15.02.2025. <https://github.com/bluez/bluez/wiki/rfcomm>.
4. Linux Networking and Network Devices APIs. — 2024. — <https://www.kernel.org/doc/html/latest/networking/kapi.html>, accessed 15.02.2025. Linux Kernel Documentation.
5. RFCOMM implementation for Linux Bluetooth stack (net/bluetooth/rfcomm/core.c). — 2024. — <https://github.com/torvalds/linux/blob/master/net/bluetooth/rfcomm/core.c>, accessed 15.02.2025. Linux kernel source code.
6. Connect Bluetooth Devices. — 2024. — <https://developer.android.com/develop/connectivity/bluetooth/connect-bluetooth-devices>, accessed 15.02.2025. Android Developers.
7. BluetoothSocket. — 2024. — <https://developer.android.com/reference/android/bluetooth/BluetoothSocket>, accessed 15.02.2025. Android Developers.
8. l2cap(7): Bluetooth L2CAP protocol. — 2024. — <https://man.archlinux.org/man/extra/bluez-utils/l2cap.7.en>, accessed 15.02.2025. Linux man pages (bluez-utils).
9. HID Subsystem. — <https://docs.kernel.org/hid/index.html>, accessed 15.02.2025. Linux Kernel Documentation.
10. HIDP core implementation for the Linux Bluetooth stack. — 2024. — <https://github.com/torvalds/linux/blob/master/net/bluetooth/hidp/core.c>, accessed 15.02.2025. Linux kernel source code, net/bluetooth/hidp/core.c.
11. CONFIG\_BT\_HIDP — HIDP (Human Interface Device Protocol) support. — 2024. — [https://www.kernelconfig.io/config\\_bt\\_hidp](https://www.kernelconfig.io/config_bt_hidp), accessed 15.02.2025. Linux kernel configuration reference.
12. HID I/O Transport Drivers. — 2024. — <https://docs.kernel.org/hid/hid-transport.html>, accessed 15.02.2025. Linux Kernel Documentation.
13. Introduction to HID report descriptors. — 2024. — <https://docs.kernel.org/hid/hidintro.html>, accessed 15.02.2025. Linux Kernel Documentation.
14. Care and feeding of your Human Interface Devices (hiddev interface). — 2018. — <https://mjmwired.net/kernel/Documentation/hid/hiddev.txt>, accessed 15.02.2025. Linux Kernel Documentation.

15. Connecting Nokia 6600 and Linux over Bluetooth HOWTO. — 2005. — <https://www.unix-ag.uni-kl.de/~leonard/linux-n6600-howto.html>, accessed 15.02.2025. Online HOWTO.
16. Driver Basics. — 2024. — <https://www.kernel.org/doc/html/latest/driver-api/basics.html>, accessed 15.02.2025. Linux Kernel Documentation.
17. Input Subsystem. — 2024. — <https://docs.kernel.org/input/input.html>, accessed 15.02.2025. Linux Kernel Documentation.
18. Creating an input device driver. — 2024. — <https://dri.freedesktop.org/docs/drm/input/input-programming.html>, accessed 15.02.2025. Linux Kernel Documentation.
19. uinput module. — 2017. — <https://www.kernel.org/doc/html/v4.12/input/uinput.html>, accessed 15.02.2025. Linux Kernel Documentation.
20. UHID — User-space I/O driver support for HID subsystem. — 2024. — <https://docs.kernel.org/hid/uhid.html>, accessed 15.02.2025. Linux Kernel Documentation.
21. kthread\_run — create and wake a thread. — [https://docs.huihoo.com/doxygen/linux/kernel/3.7/kthread\\_8h.html](https://docs.huihoo.com/doxygen/linux/kernel/3.7/kthread_8h.html), accessed 15.02.2025. Linux Kernel API Reference.
22. C Language Documentation. — accessed 15.02.2025. <https://learn.microsoft.com/cpp/c-language/>.
23. Kotlin Programming Language. — accessed 15.02.2025. <https://kotlinlang.org/>.

# ПРИЛОЖЕНИЕ А

Листинг 12 — Исходный код загружаемого модуля ядра

```
#include <linux/delay.h>
#include <linux/init.h>
#include <linux/input.h>
#include <linux/kernel.h>
#include <linux/kthread.h>
#include <linux/module.h>
#include <linux/net.h>
#include <net/bluetooth/bluetooth.h>
#include <net/bluetooth/rfcomm.h>

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Bluetooth-controlled virtual mouse\n"
                   "Works with android phones and an app\n"
                   "Tested on kernel 6.17.8-zen1-1-zen");
MODULE_AUTHOR("Zvyagin Daniil");

static int speed_pct = 100;
module_param(speed_pct, int, 0644);
MODULE_PARM_DESC(speed_pct, "Mouse speed in percent (100 = normal, 50 =
    half, "
                                "200 = double, negatives invert)");

static int interp_steps = 0;
module_param(interp_steps, int, 0644);
MODULE_PARM_DESC(interp_steps, "Interpolation steps (0=off)");

static struct input_dev *pm_input_dev;

static struct socket *listen_sock = NULL;
static struct socket *client_sock = NULL;

static struct task_struct *rx_thread = NULL;

static int bt_listen_channel = 1;
static int speed_mult = 65536; // 1.0 in Q16.16

#define LMB_MASK 0b00000001
#define RMB_MASK 0b00000010
static void handle_buttons(u8 buttons) {
```



```

    if (buttons & LMB_MASK) {
        input_report_key(pm_input_dev, BTN_LEFT, 1);
        input_sync(pm_input_dev);
        input_report_key(pm_input_dev, BTN_LEFT, 0);
        input_sync(pm_input_dev);
    }

    if (buttons & RMB_MASK) {
        input_report_key(pm_input_dev, BTN_RIGHT, 1);
        input_sync(pm_input_dev);
        input_report_key(pm_input_dev, BTN_RIGHT, 0);
        input_sync(pm_input_dev);
    }
}

static void handle_movement(u8 buf[4]) {
    s16 dx = (s16)((buf[0] << 8) | buf[1]);
    s16 dy = (s16)((buf[2] << 8) | buf[3]);

    dx = (dx * speed_mult) >> 16;
    dy = (dy * speed_mult) >> 16;

    if (interp_steps > 0) {
        int step_dx = dx / interp_steps;
        int step_dy = dy / interp_steps;

        int i;
        for (i = 0; i < interp_steps; i++) {
            input_report_rel(pm_input_dev, REL_X, step_dx);
            input_report_rel(pm_input_dev, REL_Y, step_dy);
            input_sync(pm_input_dev);
        }

        return;
    }

    input_report_rel(pm_input_dev, REL_X, dx);
    input_report_rel(pm_input_dev, REL_Y, dy);

    input_sync(pm_input_dev);
}

```

```

static int rx_loop(void *data) {
    struct msghdr msg = {0};
    struct kvec vec;
    u8 buf[5];

    while (!kthread_should_stop()) {

        if (!client_sock) {
            // wait for connection
            struct socket *new_sock = NULL;
            int r = kernel_accept(listen_sock, &new_sock, 0);
            if (r == 0) {
                client_sock = new_sock;
                pr_info("phone_mouse: client connected!\n");
            } else {
                ssleep(1);
                continue;
            }
        }

        vec.iov_base = buf;
        vec.iov_len = sizeof(buf);

        int len =
            kernel_recvmsg(client_sock, &msg, &vec, 1, sizeof(buf),
                MSG_DONTWAIT);

        if (len == -EAGAIN) {
            msleep(5);
            continue;
        }

        if (len <= 0) {
            pr_info("phone_mouse: client disconnected\n");
            sock_release(client_sock);
            client_sock = NULL;
            continue;
        }

        if (len < 5)
            continue;
    }
}

```

```

        handle_buttons(buf[0]);

        handle_movement(&buf[1]);
    }

    return 0;
}

static int __init pm_init(void) {
    int err;
    struct sockaddr_rc addr = {0};

    if (interp_steps < 0) {
        pr_err("phone_mouse_bt: ERROR: interp_steps must be >= 0 (got %d)\n",
            interp_steps);
        return -EINVAL;
    }

    speed_mult = (speed_pct * 65536) / 100;
    pr_info("phone_mouse_bt: speed coefficient = %d (Q16.16)\n", speed_mult);
    ;

    // Allocate new input device
    pm_input_dev = input_allocate_device();
    if (!pm_input_dev)
        return -ENOMEM;

    pm_input_dev->name = "Bluetooth Phone Mouse";
    pm_input_dev->id.bustype = BUS_BLUETOOTH;

    __set_bit(EV_KEY, pm_input_dev->evbit);
    __set_bit(EV_REL, pm_input_dev->evbit);

    __set_bit(BTN_LEFT, pm_input_dev->keybit);
    __set_bit(BTN_RIGHT, pm_input_dev->keybit);

    __set_bit(REL_X, pm_input_dev->relbit);
    __set_bit(REL_Y, pm_input_dev->relbit);

    err = input_register_device(pm_input_dev);
    if (err)

```

```

    return err;

// RFCOMM socket
err = sock_create_kern(&init_net, PF_BLUETOOTH, SOCK_STREAM,
    BTPROTO_RFCOMM,
    &listen_sock);

if (err < 0) {
    pr_err("phone_mouse: sock_create_kern failed\n");
    return err;
}

addr.rc_family = AF_BLUETOOTH;
bacpy(&addr.rc_bdaddr, BDADDR_ANY);
addr.rc_channel = bt_listen_channel;

err = listen_sock->ops->bind(listen_sock, (struct sockaddr *)&addr,
    sizeof(addr));

if (err < 0) {
    pr_err("phone_mouse: bind failed\n");
    return err;
}

err = listen_sock->ops->listen(listen_sock, 1);
if (err < 0) {
    pr_err("phone_mouse: listen failed\n");
    return err;
}

// Main loop in kernel thread
rx_thread = kthread_run(rx_loop, NULL, "phone_mouse_rx");
if (IS_ERR(rx_thread)) {
    pr_err("phone_mouse: failed to start thread\n");
    return PTR_ERR(rx_thread);
}

pr_info("phone_mouse: module loaded, listening RFCOMM channel %d\n",
    bt_listen_channel);

return 0;
}

```

```

static void __exit pm_exit(void) {
    if (rx_thread)
        kthread_stop(rx_thread);

    if (client_sock)
        sock_release(client_sock);

    if (listen_sock)
        sock_release(listen_sock);

    input_unregister_device(pm_input_dev);

    pr_info("phone_mouse: unloaded\n");
}

module_init(pm_init);
module_exit(pm_exit);

```

Листинг 13 — Исходный код основной активности Android приложения

```

package com.example.bt_sender

import android.Manifest
import android.bluetooth.BluetoothAdapter
import android.bluetooth.BluetoothDevice
import android.bluetooth.BluetoothSocket
import android.content.pm.PackageManager
import android.os.Bundle
import android.view.MotionEvent
import android.widget.Button
import android.widget.EditText
import android.widget.Toast
import android.view.View
import androidx.appcompat.app.AppCompatActivity
import androidx.activity.result.contract.ActivityResultContracts
import androidx.core.content.ContextCompat
import java.io.OutputStream
import java.util.UUID
import kotlin.concurrent.thread
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import androidx.core.view.WindowInsetsControllerCompat

```

```

class MainActivity : AppCompatActivity() {

    private val prefs by lazy { getSharedPreferences("btmouse",
        MODE_PRIVATE) }

    private var socket: BluetoothSocket? = null
    private var output: OutputStream? = null

    private var lastX = 0f
    private var lastY = 0f
    private var firstMove = true

    private var pendingDx = 0
    private var pendingDy = 0

    private val MOVE_INTERVAL_MS = 15L

    @Volatile private var moveSenderRunning = true

    private val btPermissions = arrayOf(
        Manifest.permission.BLUETOOTH_CONNECT,
        Manifest.permission.BLUETOOTH_SCAN
    )

    private val permissionLauncher =
        registerForActivityResult(
            ActivityResultContracts.RequestMultiplePermissions()
        ) { result ->
            val ok = result.values.all { it }
            if (!ok) {
                Toast.makeText(this, "No BT permission", Toast.LENGTH_LONG
                ).show()
            }
        }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        WindowInsetsControllerCompat(window, window.decorView).

```

```

        isAppearanceLightStatusBars = true
WindowInsetsControllerCompat(window, window.decorView).
    isAppearanceLightNavigationBars = true

// Apply safe area insets to the layout
val root = findViewById<View>(R.id.rootLayout)
ViewCompat.setOnApplyWindowInsetsListener(root) { view, insets ->
    val systemBars = insets.getInsets(WindowInsetsCompat.Type.
        systemBars())

    view.setPadding(
        view.paddingLeft,
        systemBars.top + 16,    // TOP padding = safe area + dp
        view.paddingRight,
        systemBars.bottom + 16 // BOTTOM padding = safe area + dp
    )

    WindowInsetsCompat.CONSUMED
}

thread {
    while (moveSenderRunning) {
        Thread.sleep(MOVE_INTERVAL_MS)

        val dx: Int
        val dy: Int

        synchronized(this) {
            dx = pendingDx
            dy = pendingDy
            pendingDx = 0
            pendingDy = 0
        }

        if (dx != 0 || dy != 0) {
            sendPacket(dx, dy, 0)
        }
    }
}

// views

```

```

val touchpad = findViewById<View>(R.id.touchpad)
val macInput = findViewById<EditText>(R.id.macInput)
val btnLeft = findViewById<Button>(R.id.btnLeft)
val btnRight = findViewById<Button>(R.id.btnRight)

// restore saved MAC
macInput.setText(prefs.getString("mac", ""))

if (!hasPermissions()) {
    permissionLauncher.launch(btPermissions)
}

// connect when MAC changed (lossless and simple)
macInput.setOnEditorActionListener { view, _, _ ->
    val mac = macInput.text.toString().trim()
    prefs.edit().putString("mac", mac).apply()
    connectToPc(mac)
    true
}

// mouse buttons
btnLeft.setOnClickListener {
    sendPacket(0, 0, 1) // left click
}
btnRight.setOnClickListener {
    sendPacket(0, 0, 2) // right click
}

// touchpad movement
touchpad.setOnTouchListener { _, ev ->
    handleTouch(ev)
    true
}

// auto-connect if MAC saved
val savedMac = prefs.getString("mac", "")
if (!savedMac.isNullOrEmpty()) connectToPc(savedMac)
}

private fun hasPermissions(): Boolean =
    btPermissions.all {

```



```

        ContextCompat.checkSelfPermission(this, it) == PackageManager.
            PERMISSION_GRANTED
    }

private fun connectToPc(mac: String) {
    if (mac.length < 17) {
        Toast.makeText(this, "Invalid MAC", Toast.LENGTH_SHORT).show()
        return
    }

    val adapter = BluetoothAdapter.getDefaultAdapter() ?: return

    try {
        val device: BluetoothDevice = adapter.getRemoteDevice(mac)

        thread {
            try {
                // RFCOMM channel 1 hack (driver listens there)
                val method = device.javaClass.getMethod(
                    "createRfcommSocket",
                    Int::class.javaPrimitiveType
                )
                val sock = method.invoke(device, 1) as BluetoothSocket

                adapter.cancelDiscovery()
                sock.connect()

                socket = sock
                output = sock.getOutputStream

                runOnUiThread {
                    Toast.makeText(this, "Connected to $mac", Toast.
                        LENGTH_SHORT).show()
                }
            } catch (e: Exception) {
                e.printStackTrace()
                runOnUiThread {
                    Toast.makeText(this, "Connect err: ${e.message}",
                        Toast.LENGTH_LONG).show()
                }
            }
        }
    }
}

```

```

        }
    }

    } catch (e: Exception) {
        e.printStackTrace()
        Toast.makeText(this, "MAC error: ${e.message}", Toast.
            LENGTH_LONG).show()
    }
}

private fun handleTouch(ev: MotionEvent) {
    when (ev.actionMasked) {
        MotionEvent.ACTION_DOWN -> {
            lastX = ev.x
            lastY = ev.y
            firstMove = true
        }

        MotionEvent.ACTION_MOVE -> {
            val x = ev.x
            val y = ev.y

            if (firstMove) {
                firstMove = false
                lastX = x
                lastY = y
                return
            }

            val dx = (x - lastX).toInt()
            val dy = (y - lastY).toInt()

            lastX = x
            lastY = y

            synchronized(this) {
                pendingDx += dx
                pendingDy += dy
            }
        }
    }
}

```

```

    }

    private fun sendPacket(dx: Int, dy: Int, buttons: Int) {
        try {
            val out = output ?: return

            val buf = ByteArray(5)
            buf[0] = buttons.toByte()
            buf[1] = (dx shr 8).toByte()
            buf[2] = dx.toByte()
            buf[3] = (dy shr 8).toByte()
            buf[4] = dy.toByte()

            out.write(buf)
        } catch (_: Exception) {}
    }

    override fun onDestroy() {
        super.onDestroy()
        moveSenderRunning = false

        try {
            output?.close()
            socket?.close()
        } catch (_: Exception) {}
    }
}

```