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)

ФАКУЛЬТЕТ ИУ «Информатика и системы управления»

КАФЕДРА ИУ7 «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

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

Студент **ИУ7-73Б** _____ **Д. О. Звягин**

(Подпись, дата) (И.О.Фамилия)

Руководитель _____ **Н. Ю. Рязанова**

(Подпись, дата) (И.О.Фамилия)

2025 г.

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

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 56 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

СОКРАЩЕНИЯ

HID — Human Interface Device

RFCOMM — Radio Frequency Communication

ВВЕДЕНИЕ

В работе разрабатывается загружаемый модуль ядра Linux, реализующий виртуальное устройство ввода типа «мышь», управление которым осуществляется с использованием сенсорного экрана смартфона 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 предметную область;

2) разработать алгоритмы и структуру загружаемого модуля ядра, обрабатывающего информацию, полученную со смартфона и преобразующего её в события подсистемы ввода;

3) реализовать загружаемый модуль ядра, выполняющий поставленную задачу;

4) протестировать работу реализованного загружаемого модуля ядра.

1.2 Анализ способов преобразования касаний сенсорного экрана в перемещение курсора 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 рассмотрены 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.

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.

Использование относительных координат основано на передаче смещений *dx, dy*, вычисляемых из последовательности касаний (или непрерывного перемещения пальца). Такой подход соответствует модели классических компьютерных мышей, где устройство генерирует относительные изменения положения независимо от абсолютной позиции курсора.

Кроме того, относительные смещения естественным образом интегрируются в архитектуру Android-приложения, где компонент обработчика касаний оперирует разностями коорди-

нат, что исключает необходимость приведения значений к масштабу удалённого экрана.

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

1.3 Анализ способов обмена данными между смартфоном и модулем ядра

Передача данных между мобильным устройством и загружаемым модулем ядра может быть реализована различными способами, включая сетевые протоколы и беспроводные стеки передачи данных.

Для решения поставленной задачи выбрано беспроводное соединение на основе подсистемы 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 реализован как подсистема `BlueZ`, включающая поддержку протокола L2CAP, протокола RFCOMM 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 [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.

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]. Ядро Linux предоставляет структуры и функции для работы с RFCOMM, объявленные в заголовках `<net/bluetooth/bluetooth.h>` и `<net/bluetooth/rfcomm.h>`. 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 выполняется через API класса `BluetoothSocket`, который инкапсулирует установление RFCOMM-соединения по указанному UUID сервиса [6; 7]. Таким образом формируется связка: серверный RFCOMM-сокет

в пространстве ядра и клиентское соединение в приложении Android.

В рамках данной разработки загружаемый модуль ядра создаёт серверный RFCOMM-сокет и принимает входящие соединения, а мобильное приложение выступает клиентом этого сервера.

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].

Протокол L2CAP

L2CAP представляет собой базовый протокол Bluetooth, обеспечивающий мультиплексирование каналов и передачу пакетов между устройствами [1; 2]. Реализация L2CAP входит в стек BlueZ ядра Linux; интерфейс для работы с L2CAP предоставляется ядром через функции и структуры, объявленные в заголовке `<net/bluetooth/l2cap.h>` [1; 8]. 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].

1.4 Анализ способов управления курсором мыши из загружаемого модуля ядра

Для генерации событий подсистемы ввода рассмотрено два подхода: регистрация собственного виртуального устройства в подсистеме input и вмешательство в существующий стек ввода (перехват/модификация событий уже зарегистрированных устройств или графической подсистемы).

Виртуальное устройство в подсистеме input

Этот способ заключается в регистрации виртуального устройства мыши через подсистему input [9; 10]. Модуль выделяет и инициализирует `struct input_dev`, настраивает поддержку событий `EV_REL` (`REL_X`, `REL_Y`) и `EV_KEY` (`BTN_LEFT`, `BTN_RIGHT`), после чего регистрирует устройство в input-core [9]. Структура `input_dev` и функции `input_report_*` объявлены в заголовочном файле `<linux/input.h>` ядра Linux. Генерация событий выполняется вызова-

ми input_report_rel, input_report_key и input_sync, что делает виртуальное устройство эквивалентным аппаратной мыши для остального стека ввода [9].

Структура struct input_dev приведена в листинге 1.

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 ffbit;
  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;
};
```

Перехват или модификация существующего стека ввода

Альтернативный вариант — вмешательство в уже зарегистрированные устройства или обработчики графической подсистемы (например, перехват операций /dev/**input**/eventX, модификация обработчиков оконной системы). Такой подход требует изменения существующих компонентов стека ввода или добавления промежуточных слоёв, усложняет сопровождение и увеличивает количество точек отказа. Кроме того, он не создаёт самостоятельного модуля ядра для эмуляции мыши, что расходится с задачей разработки отдельного загружаемого модуля.

1.5 Минимальные структуры и точки входа модуля

Загружаемый модуль ядра использует следующие точки входа [11]:

— функцию инициализации, регистрируемую через макрос module_init (объявлен в <linux/init.h>), выполняющую создание виртуального устройства ввода, настройку RFCOMM-сокета и запуск служебного потока;

— функцию завершения, регистрируемую через макрос module_exit из <linux/init.h>, 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 (объявлена в <linux/kthread.h>) и выполняющего цикл приёма команд по RFCOMM и генерацию событий ввода [12].

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 [9;

10].

Генерация событий выполняется вызовами input_report_rel, input_report_key и input_sync для зарегистрированного устройства [9].

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].

Выводы

Проанализированы способы сопоставления данных сенсорного экрана с перемещением курсора. Было рассмотрено использование абсолютных координат и относительных смещений dx/dy. Сравнительный анализ способов сопоставления данных сенсорного экрана с перемещением курсора приведён в таблице 1.

Таблица 1 — Сравнительный анализ способов сопоставления данных сенсорного экрана с перемещением курсора

Критерий	Абсолютные координаты	Относительные смещения
Зависимость от геометрии экранов	Есть	Нет
Объём передаваемых данных	Полные координаты	Только смещения dx/dy
Устойчивость к многомониторным конфигурациям	Требуется нормализация и учёт топологии	Не требует специальных настроек

Был выбран способ передачи относительных смещений, так как этот подход независим от геометрии экранов, минимизирует объём данных и достаточен для перемещения курсора мыши.

Проанализированы способы удалённого взаимодействия устройств на основе подсистемы Bluetooth. Было рассмотрено взаимодействие по протоколу L2CAP и использование RFCOMM в пространстве ядра. RFCOMM обеспечивает потоковый канал, прямо поддерживается Android API и не требует ручной сборки пакетов. Использование протокола L2CAP требует собственной обработки параметров канала и не имеет стандартной поддержки на стороне Android-приложения. Был выбран RFCOMM-сокет в пространстве ядра, так как он совместим с Android API и обеспечивает надёжный обмен без усложнения модуля.

Проанализированы способы генерации событий ввода. Была рассмотрена регистрация собственного input_dev и вмешательство в существующий стек ввода. Для input_dev доступ-

ны прямые вызовы `input_report_*`, обеспечивающие полный контроль над формируемыми событиями в пределах модуля ядра. Вмешательство в существующий стек зависит от внешних обработчиков и их форматов, что накладывает ограничения и увеличивает риск несовместимостей. Был выбран вариант с регистрацией собственного виртуального устройства ввода. Прямое формирование `EV_REL`/`EV_KEY` в модуле обеспечивает требуемый функционал без вмешательства в существующий стек ввода. Виртуальное устройство определяется системой как компьютерная мышь, что позволяет производить настраивать устройство в том числе и с помощью настроек графической оболочки системы.

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; 9; 10];

— мобильное приложение для Android, устанавливающее 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 [9; 10], 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].

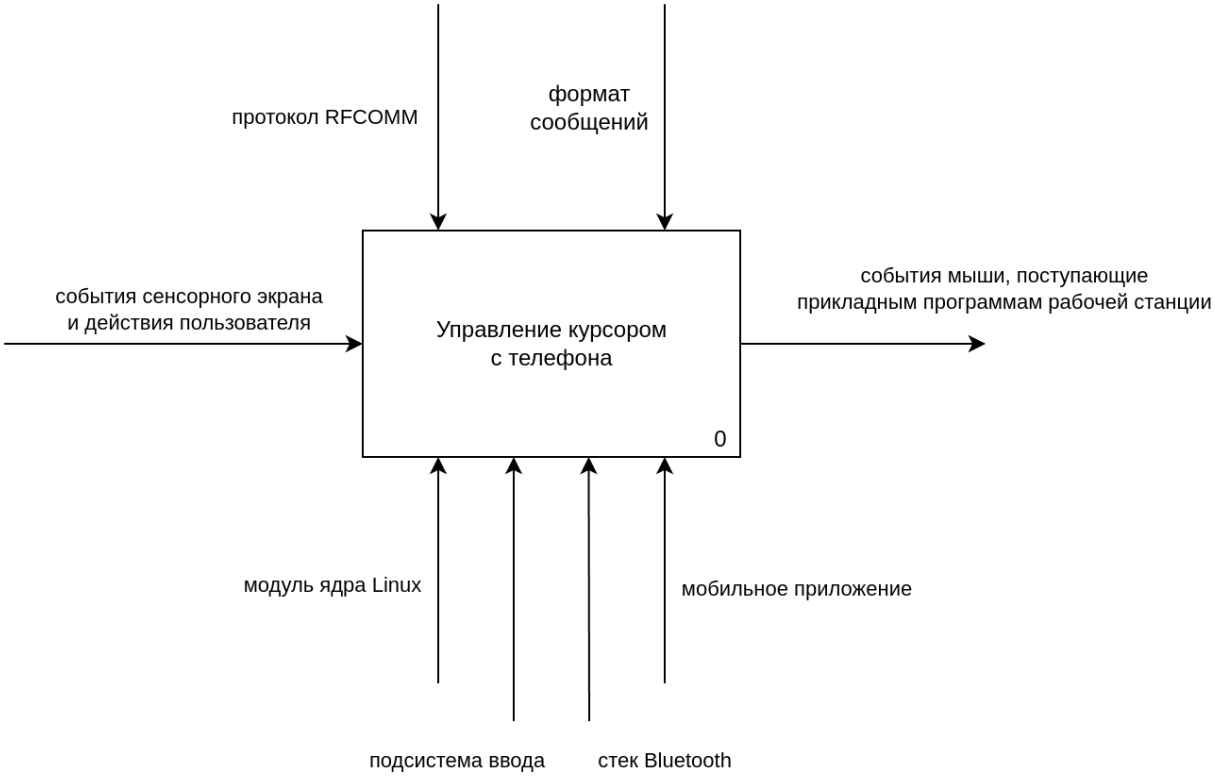

Рис. 1 — IDEF0-нулевого уровня

Части 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.

Алгоритм регистрации серверного RFCOMM-сокета представлен на рис. 2.

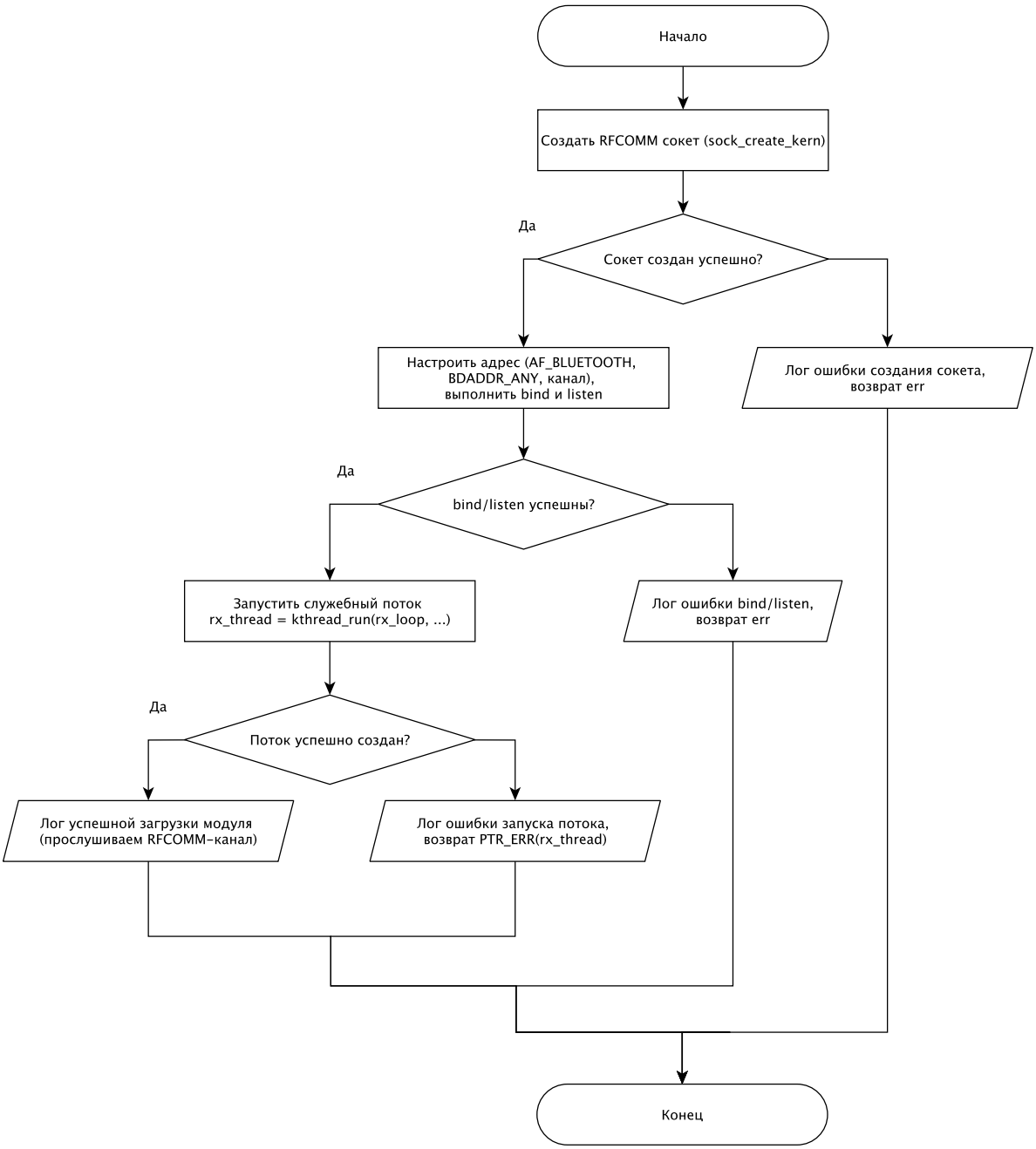

Рис. 2 — Алгоритм регистрации RFCOMM сервера

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.

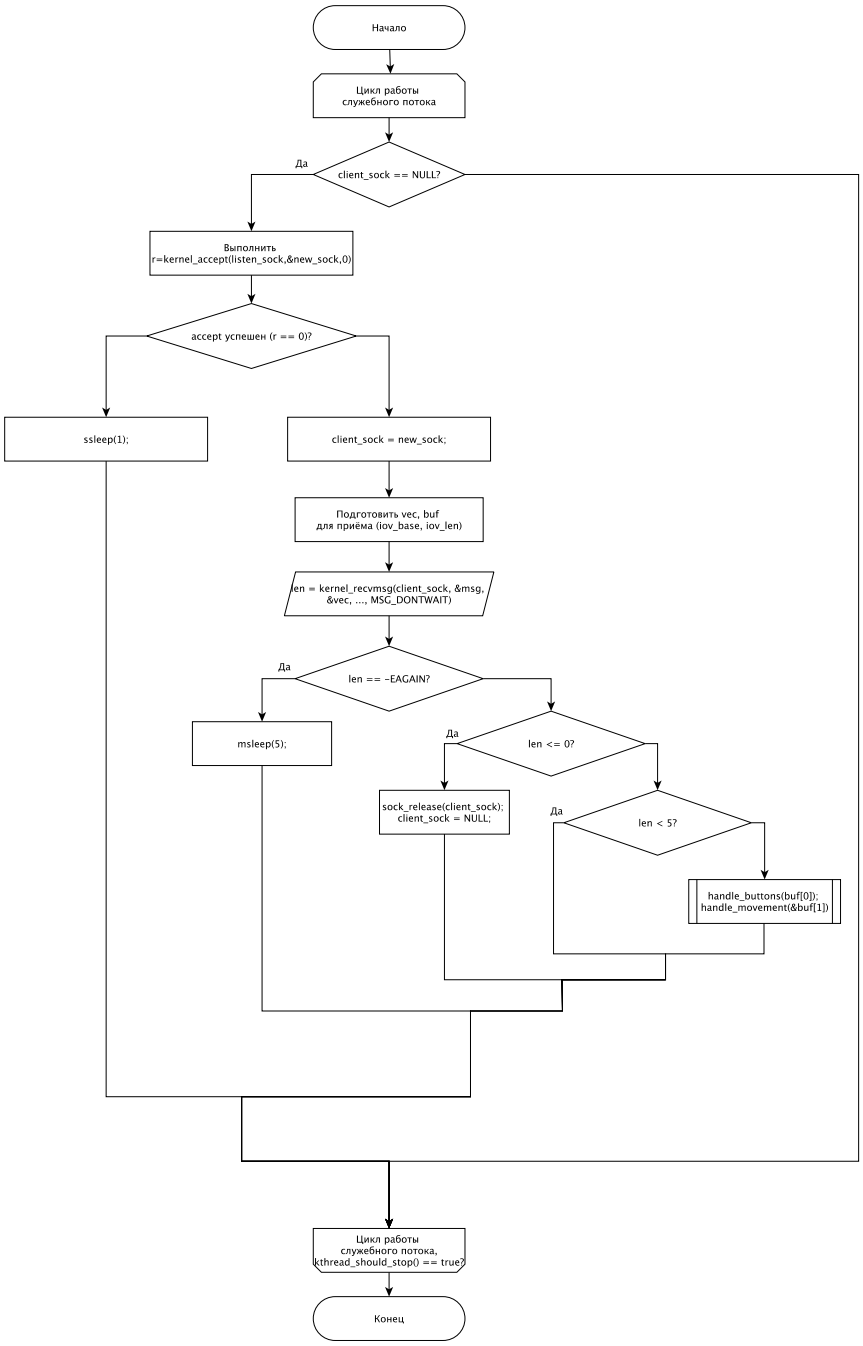

Рис. 3 — Алгоритм работы служебного потока `rx_loop`

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. Алгоритм handle_buttons приведена на рис. 4.

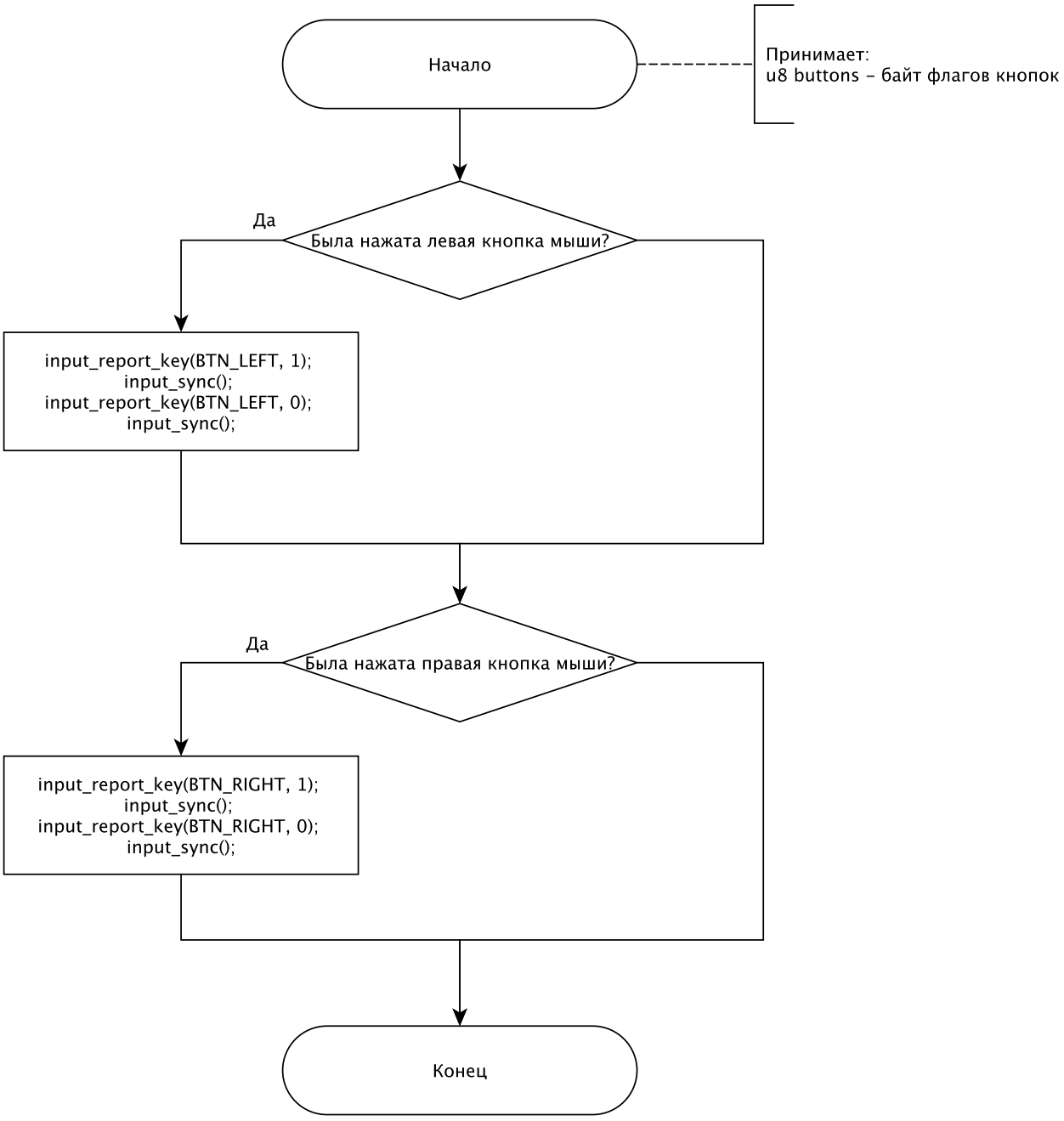

Рис. 4 — Алгоритм имитации щелчков мыши handle_buttons

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.

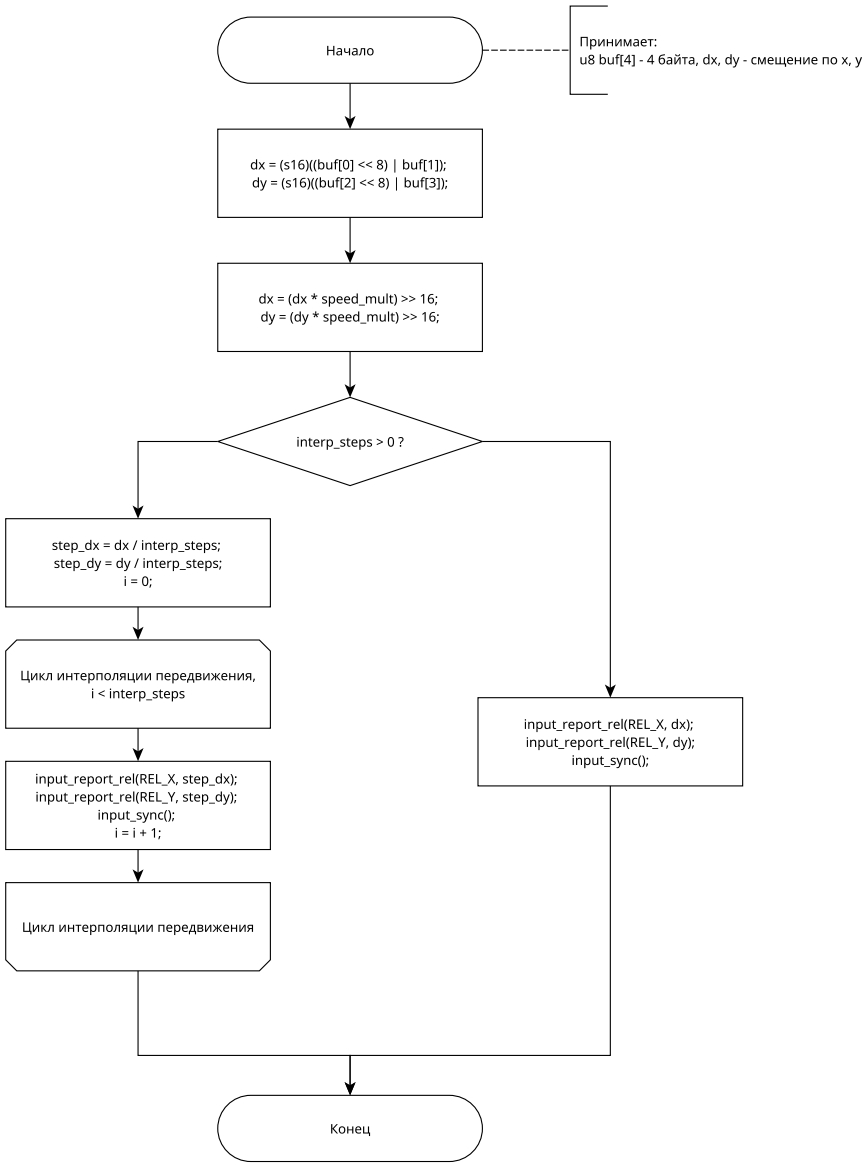

Рис. 5 — Алгоритм обработки движения курсора handle_movement

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 [9; 11].

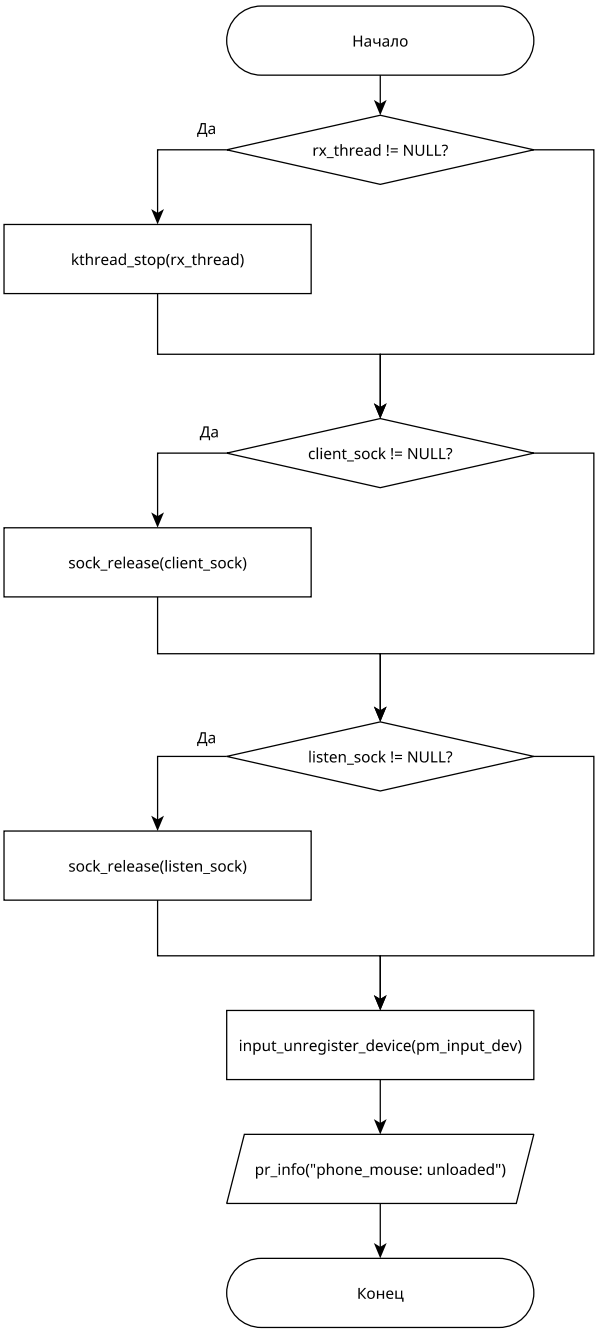

Рис. 6 — Алгоритм завершения работы модуля ядра `phone_mouse_bt`

Части 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 движений, восстановление сохранённого MAC-адреса, установка обработчиков для поля ввода MAC-адреса, кнопок мыши и области touchpad, а также, при наличии сохранённого адреса, инициируется подключение к рабочей станции.

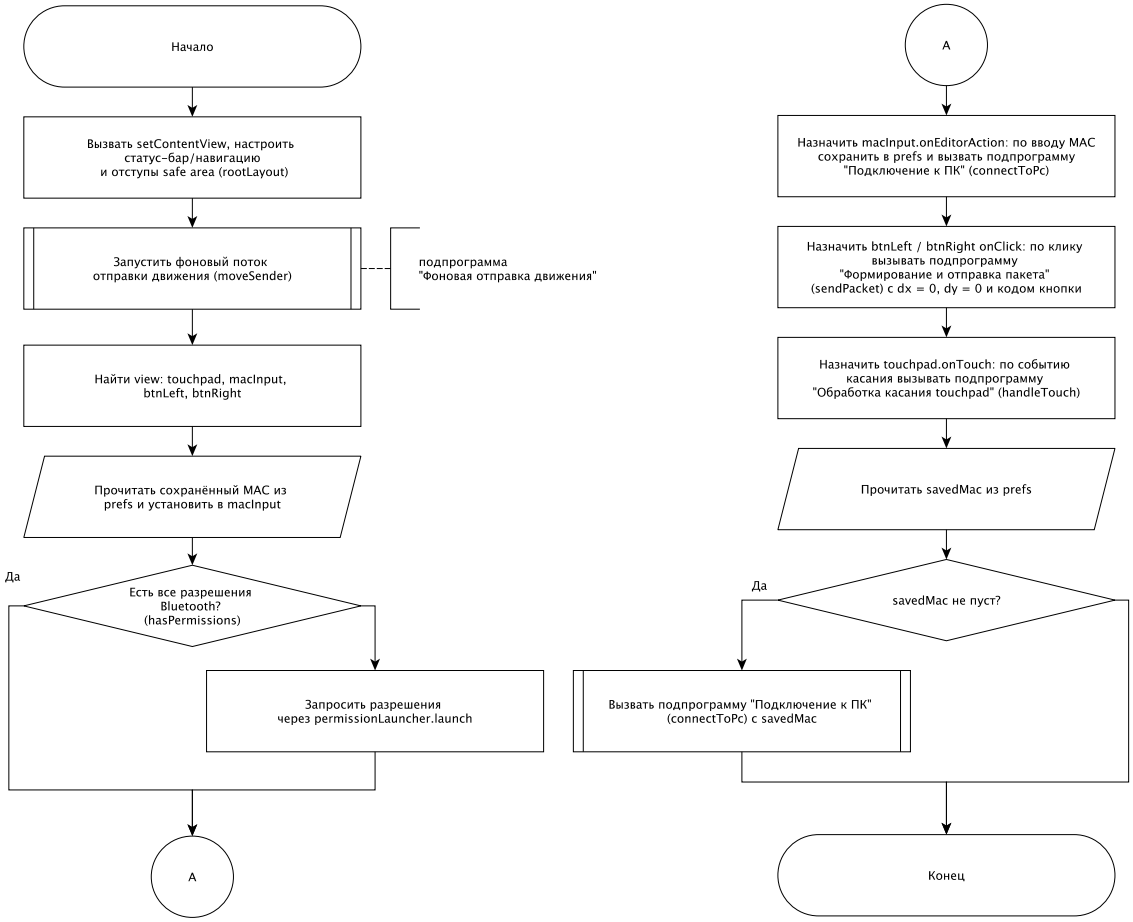

Рис. 7 — Алгоритм работы основной активности мобильного приложения

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.

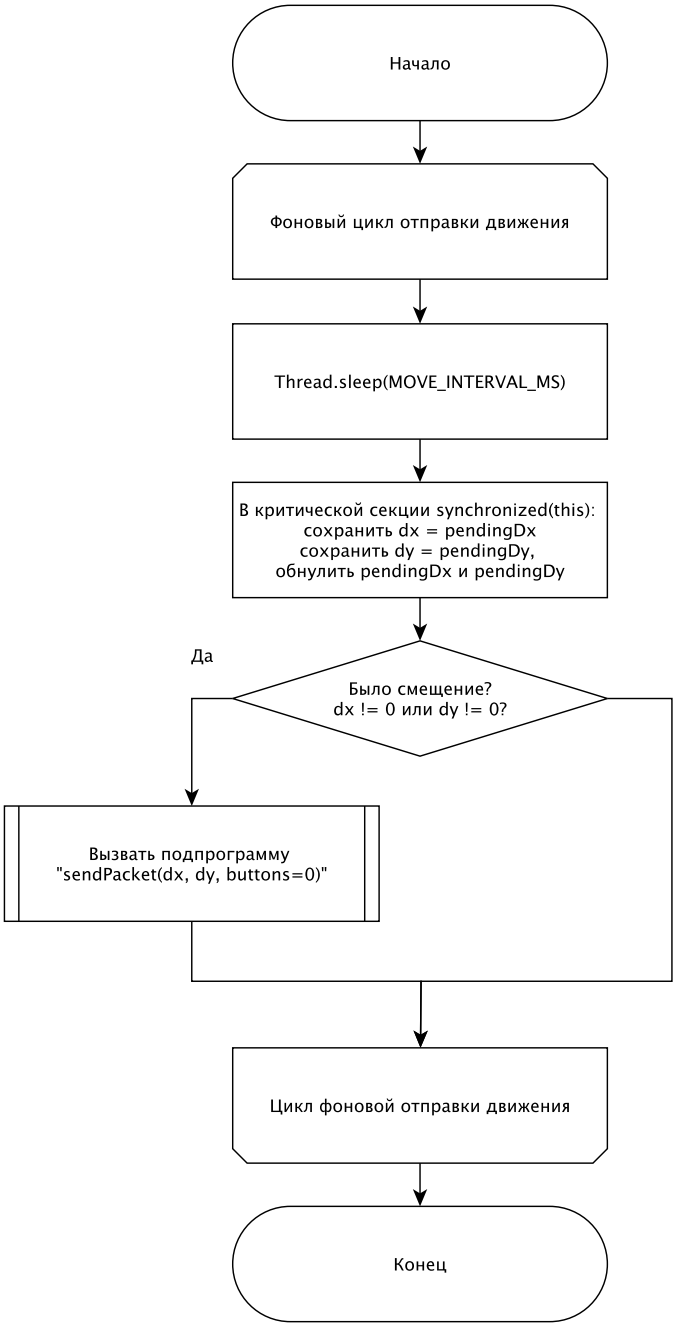

Рис. 8 — Алгоритм фонового потока отправки накопленных смещений

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.

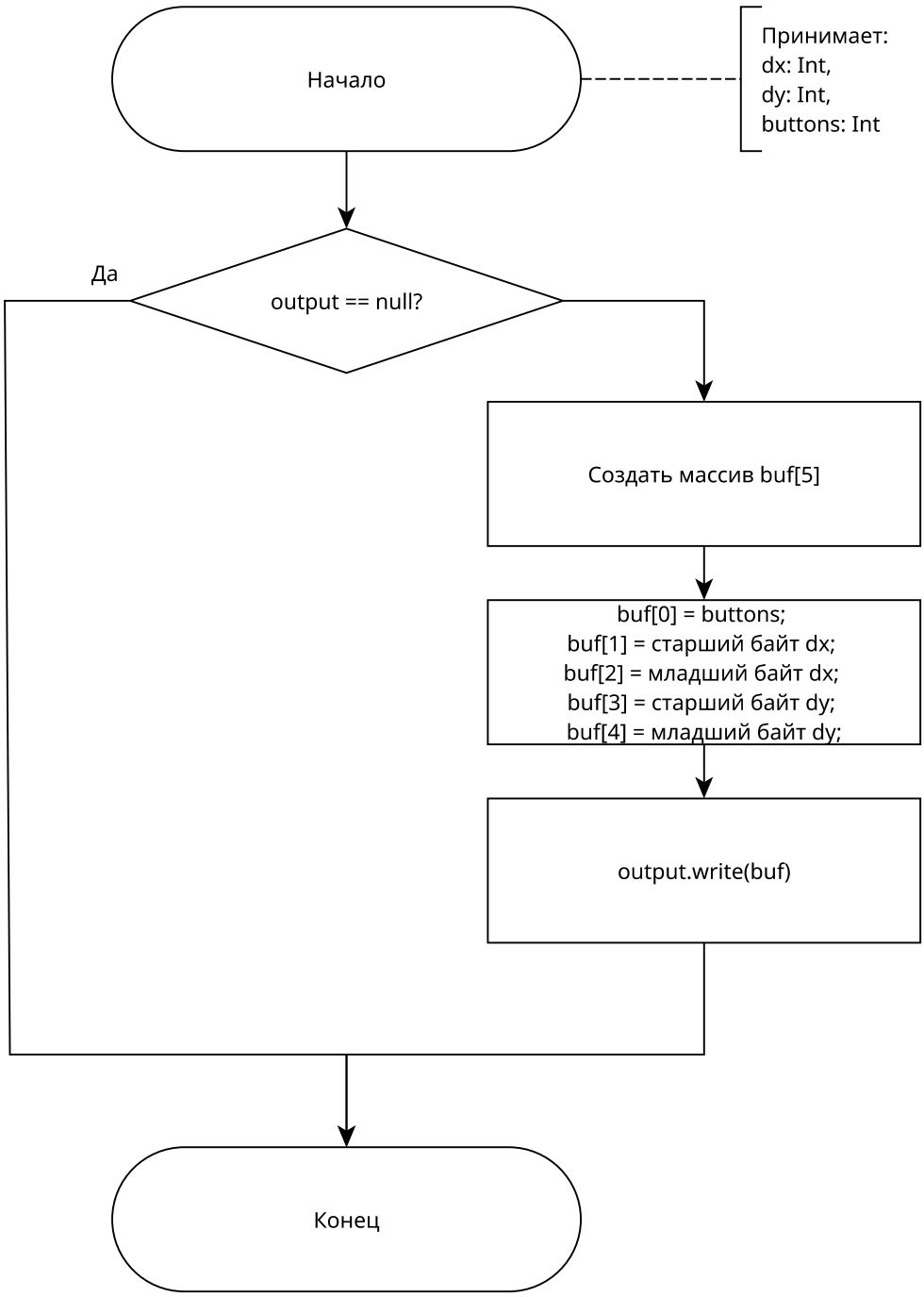

Рис. 9 — Алгоритм формирования и отправки пакета команд sendPacket

Установление соединения с целевым устройством выполняется функцией connectToPc, которая проверяет корректность введённого MAC-адреса, получает адаптер Bluetooth, инициирует фоновый поток подключения, создаёт RFCOMM-сокет 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.

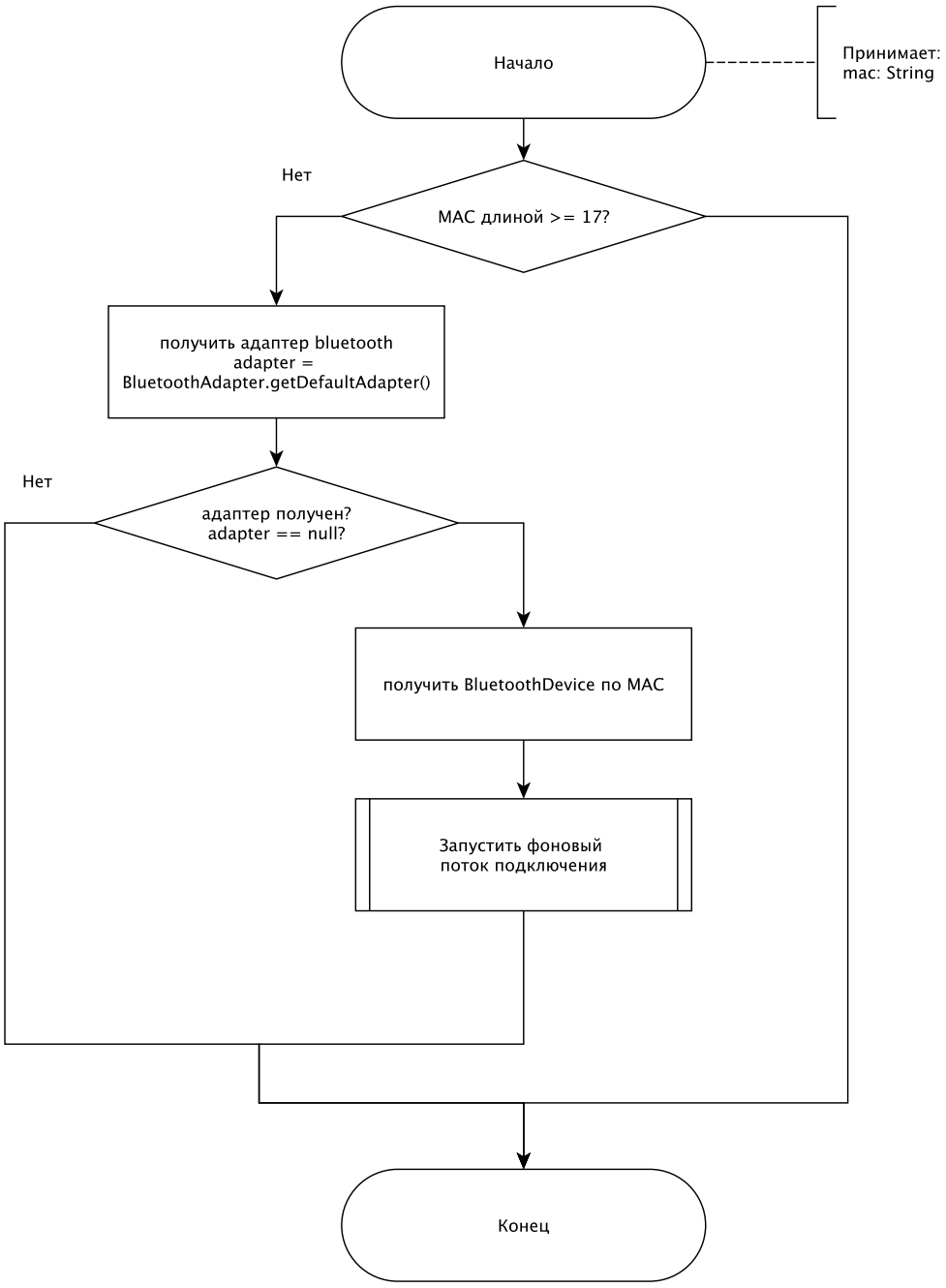

Рис. 10 — Алгоритм подключения к рабочей станции connectToPc

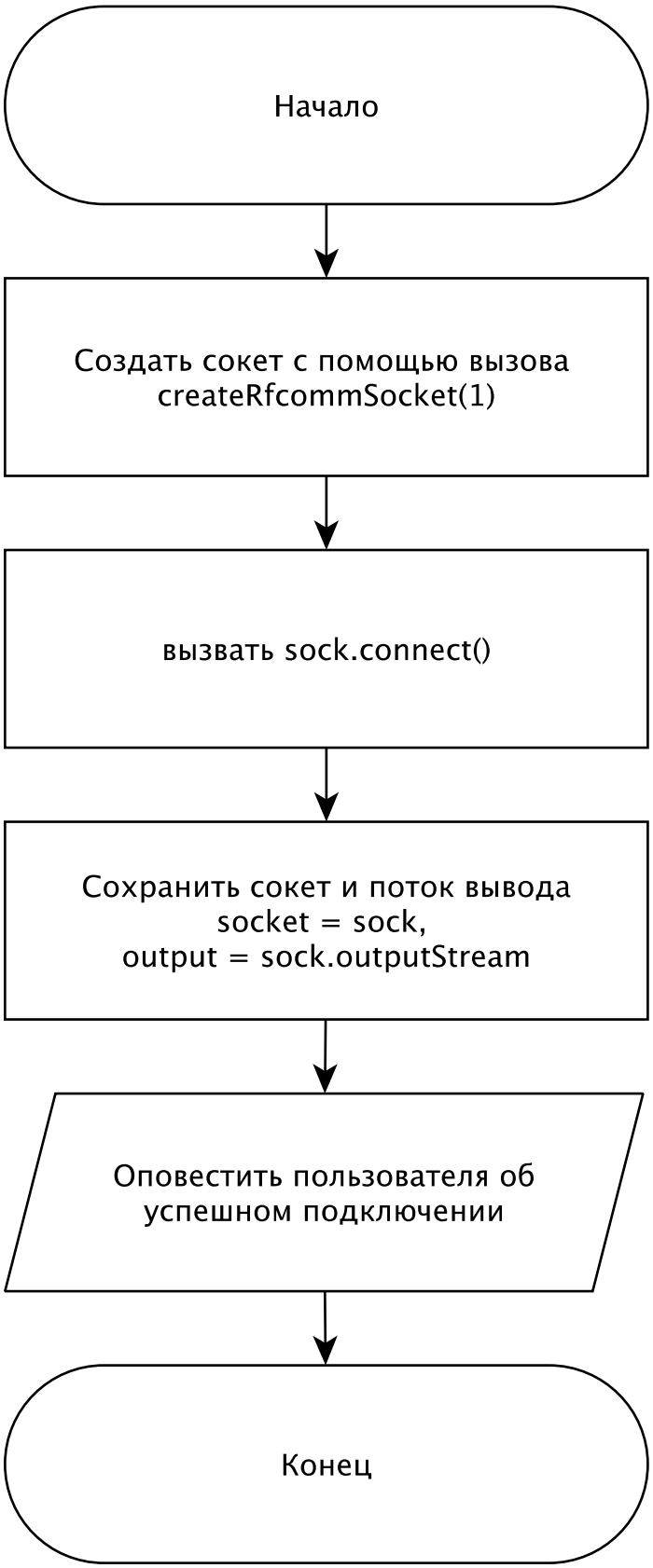

Рис. 11 — Алгоритм внутреннего потока установления RFCOMM-соединения

Обработка перемещения 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 модуль ядра.

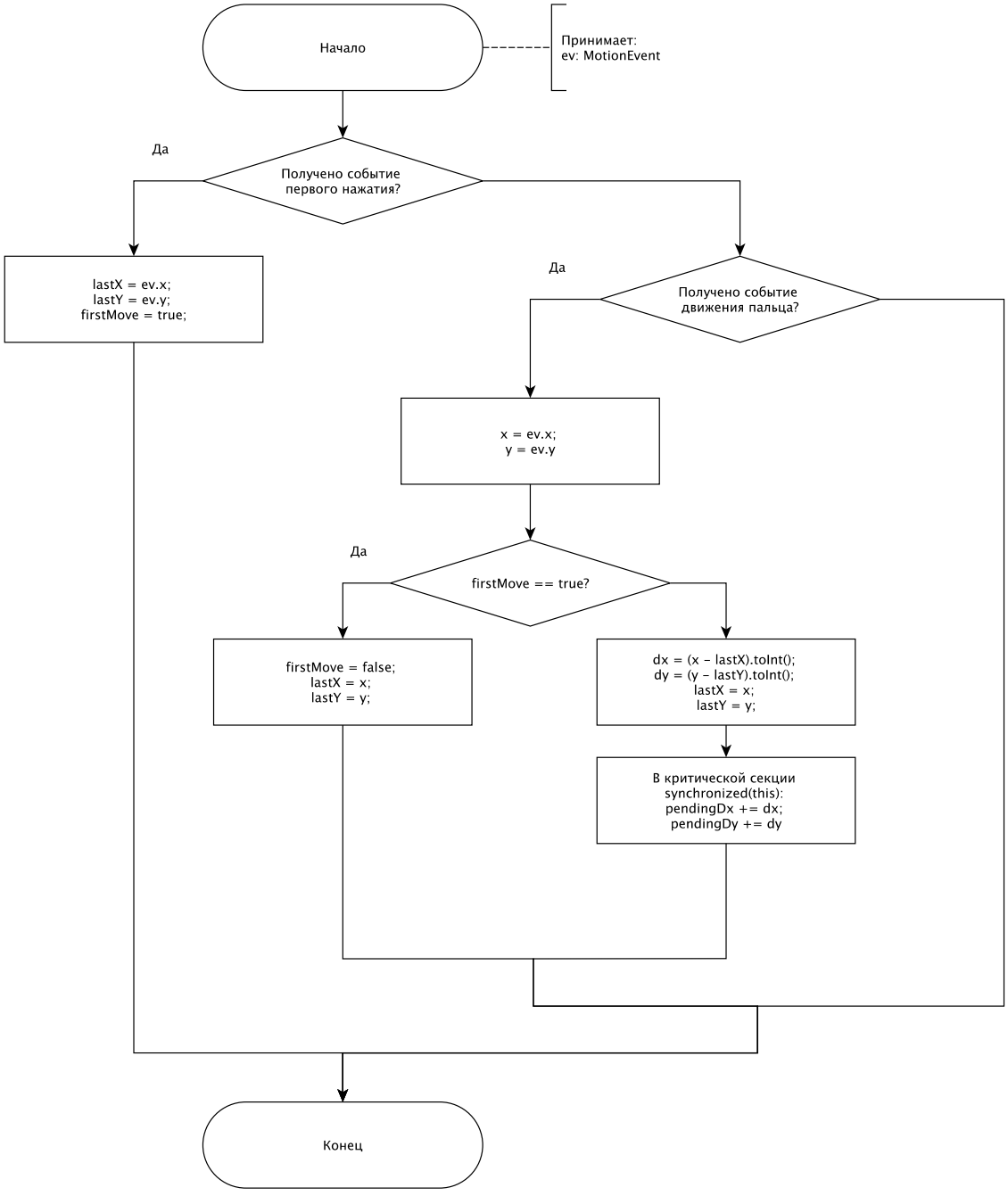

Рис. 12 — Алгоритм обработки касаний на области touchpad handleTouch

2.2 Структура ПО

Структура программного обеспечения изображена на рисунке 13

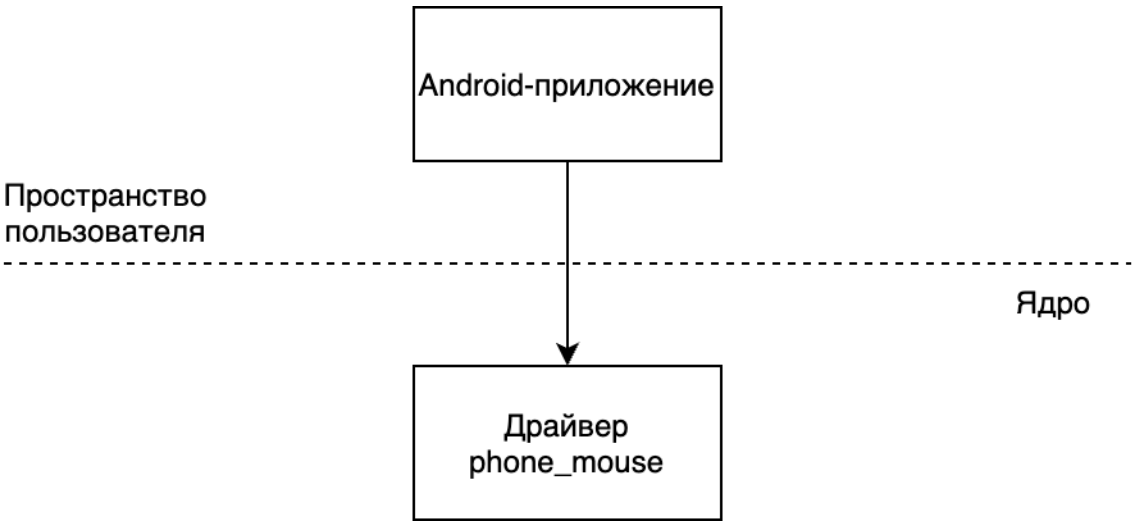

Рис. 13 — Структура ПО

3 Технологический раздел

3.1 Выбор языка и среды программирования

Загружаемый модуль ядра реализован на языке программирования C. Ядро Linux и большинство 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 [9; 11; 13]. 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 [11]. Таким образом, язык программирования C является достаточным для реализации модуля.

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 [11].

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; 14]. 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 модуля

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, содержащая идентификаторы устройства, указатели на функции обратного вызова и описания поддерживаемых типов и кодов событий [9; 10]. Дополнительно используются структуры для хранения текущих состояний кнопок и параметров RFCOMM-соединения. Фрагмент объявления структур данных модуля приведён в листинге 2.

Листинг 2 — Фрагмент объявления структур данных модуля `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 смартфона */
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 протокольном 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 [11]. В функции инициализации выполняются:

— создание и настройка объекта struct input_dev и регистрация виртуального устройства мыши в подсистеме ввода;

— инициализация структур данных состояния модуля;

— 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; 12].

Функция завершения выполняет остановку служебного потока, закрытие RFCOMM-сокета, снятие виртуального устройства с регистрации и освобождение всех выделенных ресурсов [11].

Функции инициализации и завершения 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 от-

сутствии 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; 9; 10].

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_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;
  }
```

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 [11].

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 (модуль 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.outputStream

      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 приложения 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 модуль 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 модуль ядра 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 модуль 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;

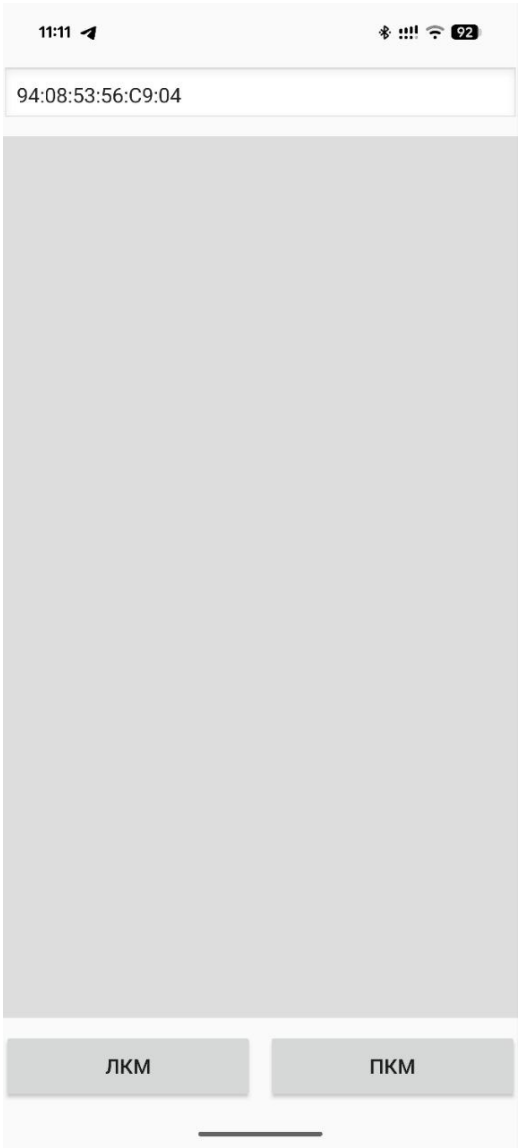

Рис. 14 — Интерфейс мобильного приложения управления курсором

— 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 со смартфона по Bluetooth посредством загружаемого модуля ядра.

ЗАКЛЮЧЕНИЕ

При выполнении курсовой работы проанализированы способы преобразования данных, полученных с сенсорного экрана смартфона в перемещение курсора мыши, способы удалённого взаимодействия ПК и смартфона и способы генерации событий ввода в ядре Linux.

Было рассмотрено использование абсолютных координат и относительных смещений `dx/dy`. Был выбран способ передачи относительных смещений, так как этот подход независим от геометрии экранов, минимизирует объём данных и достаточен для перемещения курсора мыши.

Были проанализированы способы удалённого взаимодействия устройств на основе подсистемы Bluetooth. Рассмотрено взаимодействие по протоколу L2CAP и использование RFCOMM-сокета в пространстве ядра. Был выбран RFCOMM-сокет в пространстве ядра, так как он совместим с Android API и обеспечивает надёжный обмен без усложнения модуля.

Проанализированы способы генерации событий ввода. Была рассмотрена регистрация собственного `input_dev` и вмешательство в существующий стек ввода. Был выбран вариант с регистрацией собственного виртуального устройства ввода, так как прямое формирование событий в модуле обеспечивает требуемый функционал без вмешательства в существующий стек ввода и не подвержено непредсказуемому воздействию внешних обработчиков.

При выполнении работы разработан загружаемый модуль ядра и Android-приложение. Загружаемый модуль ядра выступает в качестве RFCOMM-сервера и выполняет обработку пакетов, отправленных из приложения на смартфоне. Пакеты содержат информацию об относительных смещениях, которые переводятся в события перемещения курсора мыши и события щелчков кнопок мыши. Мобильное приложение является клиентом, подключающимся к RFCOMM-серверу модуля ядра. Оно обеспечивает считывание событие с сенсорного экрана и преобразование их в пакеты, отправляемые в загружаемый модуль ядра.

Выполнено тестирование разработанного ПО. В ходе тестирования подтверждена корректная работа модуля и приложения.

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. Input Subsystem. — 2024. — https://docs.kernel.org/input/input.html, accessed 15.02.2025. Linux Kernel Documentation.
10. Creating an input device driver. — 2024. — https://dri.freedesktop.org/docs/drm/input/input-programming.html, accessed 15.02.2025. Linux Kernel Documentation.
11. Driver Basics. — 2024. — https://www.kernel.org/doc/html/latest/driver-api/basics.html, accessed 15.02.2025. Linux Kernel Documentation.
12. kthread_run — create and wake a thread. — https://docs.huihoo.com/doxygen/linux/kernel/3.7/kthread_8h.html, accessed 15.02.2025. Linux Kernel API Reference.
13. C Language Documentation. — accessed 15.02.2025. https://learn.microsoft.com/cpp/c-language/.
14. 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.outputStream

                    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) {}
    }
}
```