Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное 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 ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

«Загружаемый модуль ядра Linux для задания опредёленных действий системы движением нескольких пальцев по тачпаду»

Студент ИУ7-75Б _____ **Богаченко А. Е.**

Руководитель курсового проекта _____ **Рязанова Н. Ю.**

2023 г.

РЕФЕРАТ

Расчетно-пояснительная записка 31 с., 10 рис., 2 ист., 2 прил.

В работе представлена реализация ПО для задания определённых действий системы движением нескольких пальцев.

Рассмотрены требования к реализации драйвера устройства ввода. Выбран способ реализации, приведены листинги кода. Представлена демонстрация работы программы.

КЛЮЧЕВЫЕ СЛОВА

Linux, Kernel, драйвер, тачпад

Содержание

ВВЕДЕНИЕ

Возможность удобно задавать часто-используемые действия системы напрямую влияет на работу человека за портативным компьютером. Далеко не все портативные компьютеры оснащены возможностью задавать определённые действия системы, особенно с помощью движения нескольких пальцев по тачпаду.

Тачпад – указательное (координатное) устройство ввода, предназначенное для управления курсором и отдачи различных команд компьютеру. Оно является обязательным для каждого портативного компьютера, поэтому имеет смысл добавить возможность поддержки движений, использующих несколько пальцев.

Тема курсовой работы – разработать ПО, позволяющее задавать определённые действия системы движением нескольких пальцев по тачпаду.

1 Аналитический раздел

1.1 Постановка задачи

В соответствии с заданием на курсовую работу по дисциплине «Операционные системы» требуется разработать ПО, позволяющее задавать определённые действия системы движением нескольких пальцев по тачпаду.

Для выполнения задания требуется решить следующие задачи:

1) провести анализ существующих подходов к обработке устройств ввода;
2) провести анализ существующих способов задания действий движением нескольких пальцев по тачпаду;
3) провести анализ возможности исполнения системой заданных действий;
4) разработать алгоритмы, необходимые для реализации ПО;
5) разработать ПО, предоставляющее требуемую функциональность;
6) провести исследование разработанного ПО.

Для разработки и тестирования данной работы используется портативный компьютер `Huawei Matebook X Pro` и операционная система `Linux` с дистрибутивом `Kali Linux`.

1.2 Загружаемый модуль ядра

Ядро Linux относится к классу монолитных. В архитектуре данного класса прикладные приложения выполняются посредством создания отдельной ветки кода в пространстве ядра. Поскольку в ранних версиях расширение функциональности требовало перекомпиляции ядра, что было недопустимо для систем промышленного уровня, позднее была добавлена технология модуля ядра. Модуль ядра может реализовывать драйвер устройства, файловую систему или сетевой протокол

К преимуществам такого подхода следует отнести сокращение неиспользуемого кода в базовом ядре и, как следствие, уменьшение занимаемой памяти. Недостатком является фрагментация ядра, после загрузки модулей, несмотря на то, что код базового ядра не фрагментируем. Фрагментация ведет к незначительному снижению производительности из-за увеличения пропусков записей TLB. Также использование загружаемого модуля снижает безопасность системы, поскольку злоумышленники, имея доступ в пространство ядра, могут скрывать процессы и файлы.

1.2.1 Компоненты модуля ядра

Загрузка модуля осуществляется с помощью команд `insmod` или `modprobe`. Отличие заключается в том, что `modprobe` разрешает зависимости модулей. При загрузке вызывается функция входа, определенная в модуле. Для указания ее назначения используется макрос `module_init`. При корректной загрузке функция входа возвращает 0. В ином случае, модуль не будет загружен.

Для просмотра списка загруженных модулей и информации об отдельном модуле используются команды `lsmod` и `modinfo` соответственно. Командой `rmmod` осуществляется выгрузка модуля, при которой вызывается функция выхода. Для неё используется макрос `module_exit`.

1.2.2 Драйвера

Одной из разновидностей модуля ядра являются драйверы устройств. Драйверы полностью скрывают детали, касающиеся работы устройства и предоставляют чёткий программный интерфейс для работы с аппаратурой. В Linux драйверы устройств бывают трёх типов:

1) драйверы первого типа являются частью программного кода ядра. Соответствующие устройства автоматически обнаруживаются системой и становятся доступны доя приложений;
2) драйверы второго типа представлены загружаемыми модулями ядра. Они оформлены в виде отдельных файлов. Для их подключения необходимо выполнить команду подключения модуля. Если необходимости в использовании нет, модуль можно выгрузить из памяти. Использование модулей обеспечивает большую гибкость, так как каждый драйвер может быть переконфигурирован без остановки системы;
3) в драйверах третьего типа программный код драйвера поделён между ядром и специальной утилитой, предназначенной для управления данным устройством.

Во всех драйверах взаимодействие с устройством осуществляет ядро или модуль ядра, а пользовательские программы взаимодействуют через специальные файлы, расположенные в каталоге `/dev` и его подкаталогах.

1.2.3 Драйвер устройства ввода

Драйвер устройства ввода – это модуль, обеспечивающий возможность взаимодействия с интерактивным устройством через события.

Алгоритм регистрации драйвера устройства ввода [1]:

1) заполнение структуры `struct input_handler`;
2) регистрация обработчика функцией `input_register_handler`.

Для обработки устройств ввода можно воспользоваться подсистемой ввода/вывода. Данная подсистема выполняет запросы файловой подсистемы и подсистемы управления процессами для доступа к периферийным устройствам (дискам, магнитным лентам, терминалам и т.д.). Она обеспечивает необходимую буферизацию данных и взаимодействует с драйверами устройств – специальными модулями ядра, непосредственно обслуживающими внешние устройства.

1.3 Передача информации из пространства ядра в пространство пользователя

Файловая система (ФС) `proc` создана для того, чтобы в режиме пользователя получать информацию о процессах и ресурсах, которые используют эти процессы. ФС `proc` формирует интерфейс, позволяющий обращаться к структурам ядра.

В ФС `proc` можно создавать свои файлы, ссылки, директории. Для этого в ядре предоставляется структура `proc_dir_entry`, определенная в файле `<fs/proc/internal.h>`. Для определения обратных вызовов чтения и записи предоставляется структура `proc_ops`. Данная структура определена в файле `<include/linux/proc_fs.h>`, наиболее важные поля которой представлены в листинге 1.

Листинг 1: Структура proc_ops

```
struct proc_ops {
  unsigned int proc_flags;
  int (*proc_open)(struct inode *, struct file *);
  ssize_t (*proc_read)(struct file *, char __user *, size_t, loff_t
      *);
  ...
  ssize_t (*proc_write)(struct file *, const char __user *, size_t,
      loff_t *);
  ...
  int (*proc_release)(struct inode *, struct file *);
  ...
} __randomize_layout;
```

Использование функций `proc_create` и `remove_proc_entry`, определённых в файле `<include/linux/proc_fs.h>`, позволяет регистрировать и отменять регистрацию файла в `proc`. Для взаимодействия ядра с приложениями ис-

пользуется функция `copy_to_user`, определение которой представлено в файле `<include/linux/uaccess.h>`. Данная функция копирует блоки данных из ядра в пространство пользователя. Для передачи информации в пространство ядра используется функция `copy_from_user`.

1.4 Прерывания

Прерывания делятся на:

- исключения (деление на ноль, переполнение стека), синхронные;
- системные вызовы (программные) - вызываются с помощью соответствующей команды из программы (`int 21h`), синхронные;
- аппаратные прерывания (прерывания от системного таймера, клавиатуры), асинхронные.

Прерывания делятся на 2 группы:

- быстрые;
- медленные.

Для того чтобы сократить время обработки медленных прерываний, они делятся на 2 части:

- `top half`, верхняя половина, запускается в результате получения процессором сигнала прерывания;
- `bottom half`, нижняя половина, отложенные вызовы.

Существует несколько способов реализации “нижней половины“ обработчика:

- `softirq`;
- тасклет (`tasklet`);
- очереди работ (`workqueue`).

1.4.1 Обработчики аппаратных прерываний

Обработчик прерывания должен выполнять минимальный объем действий и завершаться как можно быстрее. Обычно такой обработчик прерывания сохраняет данные, поступившие от внешнего устройства, в буфере ядра. Но для того чтобы обработать прерывания полностью, обработчик аппаратного прерывания должен инициализировать постановку в очередь на выполнение отложенное действие.

1.4.2 Очереди работ

Очереди работ являются обобщённым механизмом отложенного выполнения, в котором функция обработчика, реализующая соответствующие дей-

ствия, может блокироваться.

`struct workqueue_struct` - описывает очередь работ.

Листинг 2: Структура workqueue_struct

```
1  struct workqueue_struct {
2    struct list_head pwqs; /* WR: all pwqs of this wq */
3    struct list_head list; /* PR: list of all workqueues */
4    ...
5    struct pool_workqueue  *dfl_pwq; /* PW: only for unbound wqs */
6    ...
7    struct pool_workqueue __percpu *cpu_pwqs; /* I: per-cpu pwqs */
8    ...
9  };
```

`struct work_struct` - описывает работу (обработчик нижней половины).

Листинг 3: Структура workqueue_struct

```
1  struct work_struct {
2    atomic_long_t data;
3    struct list_head entry;
4    work_func_t func;
5    ...
6  };
```

Работа может инициализироваться 2-мя способами:

- статически;
- динамически.

При статической инициализации используется макрос:

Листинг 4: статическая инициализация

```
1  DECLARE_WORK(name, void (*func)(void *));
```

где: `name` – имя структуры `work_struct`, `func` – функция, которая вызывается из `workqueue` – обработчик нижней половины.

При динамической инициализации используются макросы:

Листинг 5: динамическая инициализация

```
1  INIT_WORK(sruct work_struct *work, void (*func)(void),void *data)
       ;
```

После того, как будет инициализирована структура для объекта `work`, следующим шагом будет помещение этой структуры в очередь работ. Это можно сделать несколькими способами. Во-первых, можно добавить работу (объект `work`) в очередь работ с помощью функции `queue_work` (которая назначает работу текущему процессору). Во-вторых, можно с помощью функции `queue_work_on` указать процессор, на котором будет выполняться обработчик.

1.5 Поток ядра

Потоки или нити ядра работают внутри пространства ядра и не имеют своего собственного адресного пространства. Они могут независимо назначаться на выполнение. Потоки используют стандартные механизмы синхронизации ядра, такие как `sleep` или `wakeup`.

Потоки ядра используются для выполнения асинхронного ввода-вывода. Ядро создаёт поток для обработки запросов каждой такой операции. Потоки также могут быть использованы для обработки прерываний.

1.5.1 Переключение потоков

Переключение потоков является основной задачей планировщика. В ОС Linux механизм планирования основывается на приоритетах. Когда процесс просыпается, ядро устанавливает значение текущего приоритета, равное значению приоритета сна события или ресурса, на котором он был заблокирован. Такой процесс будет назначен на выполнение раньше, чем другие процессы в режиме задачи.

Когда процесс завершил выполнение системного вызова и находится в состоянии возврата в режим задачи, его приоритет сбрасывается обратно в значение текущего приоритета в режиме задачи.

1.6 Запуск процессов пользователя из пространства ядра

Использование функции `call_usermodhelper` [2] позволяет выполнять процесс пользователя из процесса ядра. Работа этой функции аналогична работе вызова `exec()` в пространстве пользователя. Особенность данного метода заключается в том, что процесс запускается без управляющего терминала и с нестандартным окружением. Внутренняя реализация функции представлена на рисунке 1.

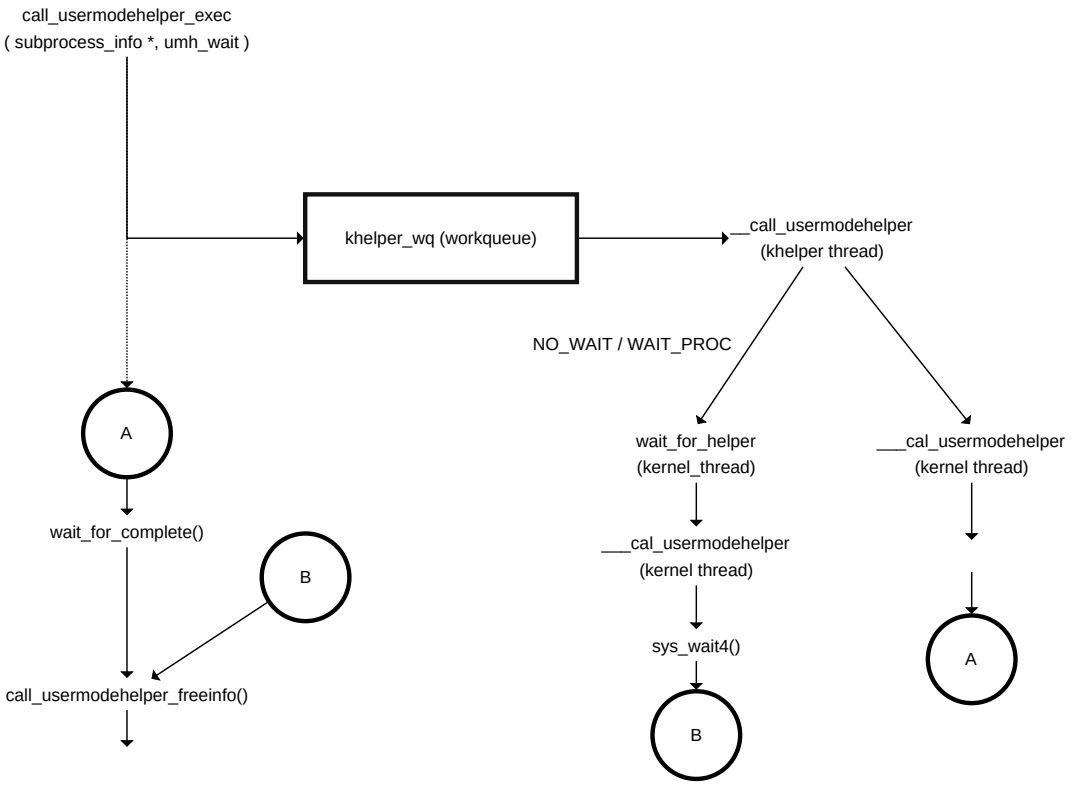

Рисунок 1 – Внутренняя реализация функции `call_usermodehelper`

1.7 Определение нескольких пальцев на тачпаде

Для определения нескольких пальцев на тачпаде можно воспользоваться `Multi Touch` протоколом. Для каждого пальца отправляется набор событий, содержащий:

- `ABS_MT_SLOT` – порядковый номер;
- `ABS_MT_TRACKING_ID` – идентификатор;
- `ABS_MT_POSITION_X` – координата X;
- `ABS_MT_POSITION_Y` – координата Y.

Каждый набор событий завершается событием `SYN_REPORT`.

При удалении пальца с тачпада идентификатор `ABS_MT_TRACKING_ID` для соответствующего пальца будет равен -1. Пример потока событий для двух пальцев показан в листинге 6.

Листинг 6: Поток событий для двух пальцев

```
ABS_MT_SLOT 0
ABS_MT_TRACKING_ID 45
ABS_MT_POSITION_X x[0]
ABS_MT_POSITION_Y y[0]
ABS_MT_SLOT 1
ABS_MT_TRACKING_ID 46
ABS_MT_POSITION_X x[1]
ABS_MT_POSITION_Y y[1]
SYN_REPORT
```

1.8 Выводы

В результате проведённого анализа было решено:

1) для выполнения задания разработать драйвер устройства ввода, который должен быть реализован как загружаемый модуль ядра;
2) для выполнения заданных действий использовать очередь работ;
3) для задания действий системы воспользоваться скриптовым языком `bash`;
4) для передачи инструкций в драйвер использовать виртуальную систему `/proc/`.

2 Конструкторский раздел

2.1 Требование к ПО

Необходимо реализовать загружаемый модуль ядра, считывать перемещение пальцев по тачпаду и выполнять заданные действия системы. Действия необходимо поместить в файл ФС `proc`.

2.2 IDEF0

На рисунке 2 изображена диаграмма IDEF0 требуемой задачи.

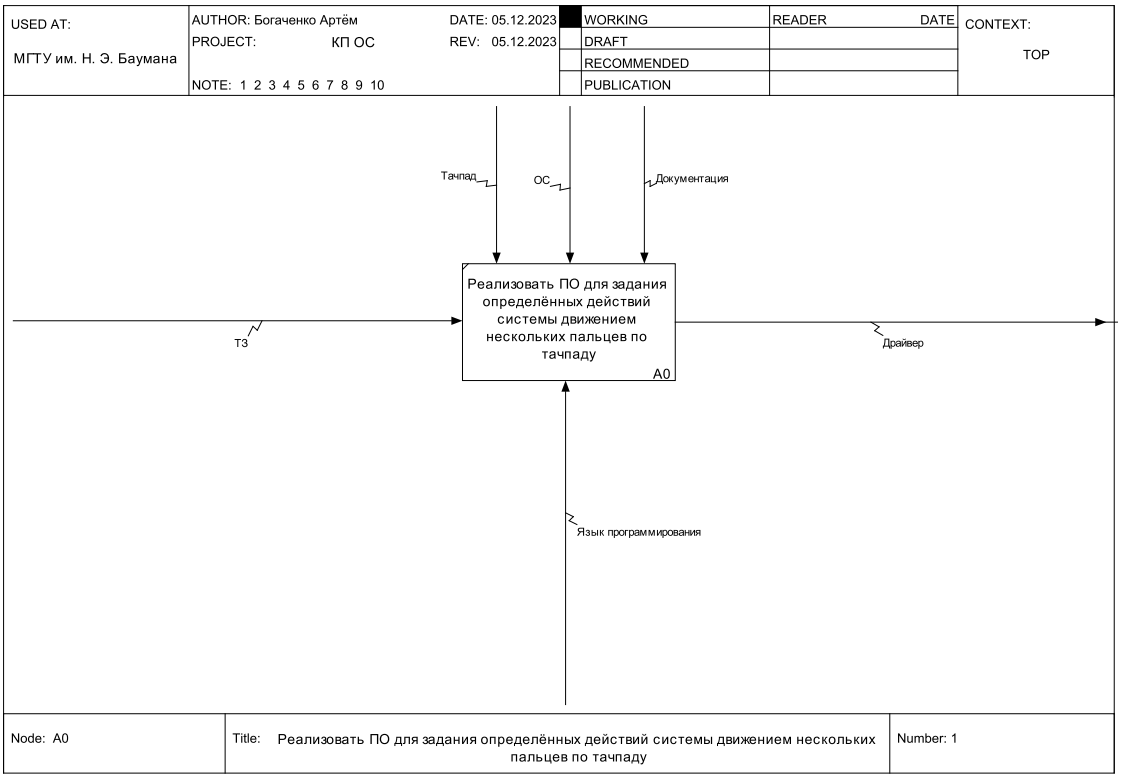

Рисунок 2 – Диаграмма IDEF0

2.3 Алгоритм работы обработчика

На рисунке 3 представлен алгоритм работы обработчика.

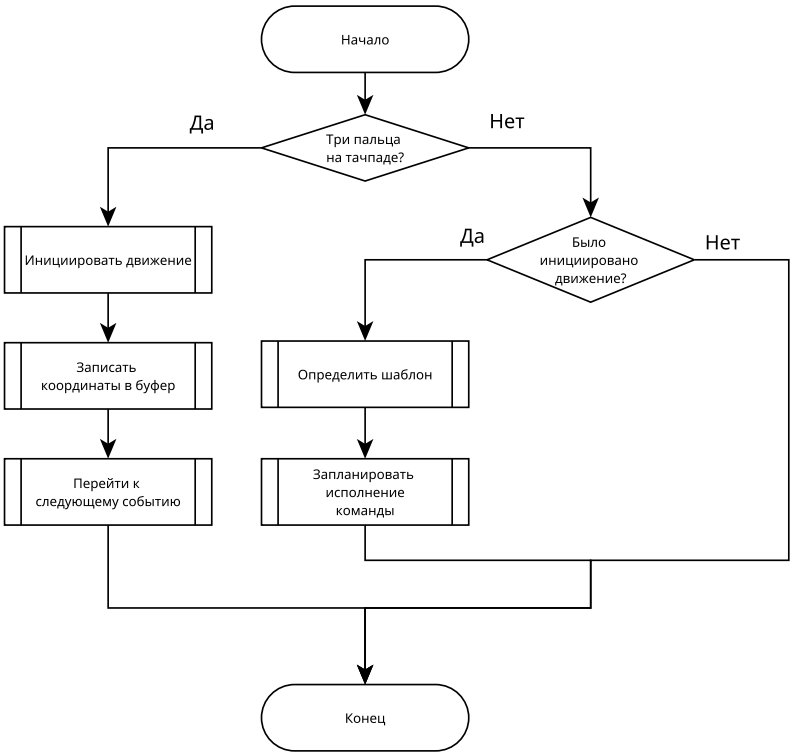

Рисунок 3 – Алгоритм работы обработчика

2.4 Структура ПО

На рисунке 4 представлена структура конечного ПО.

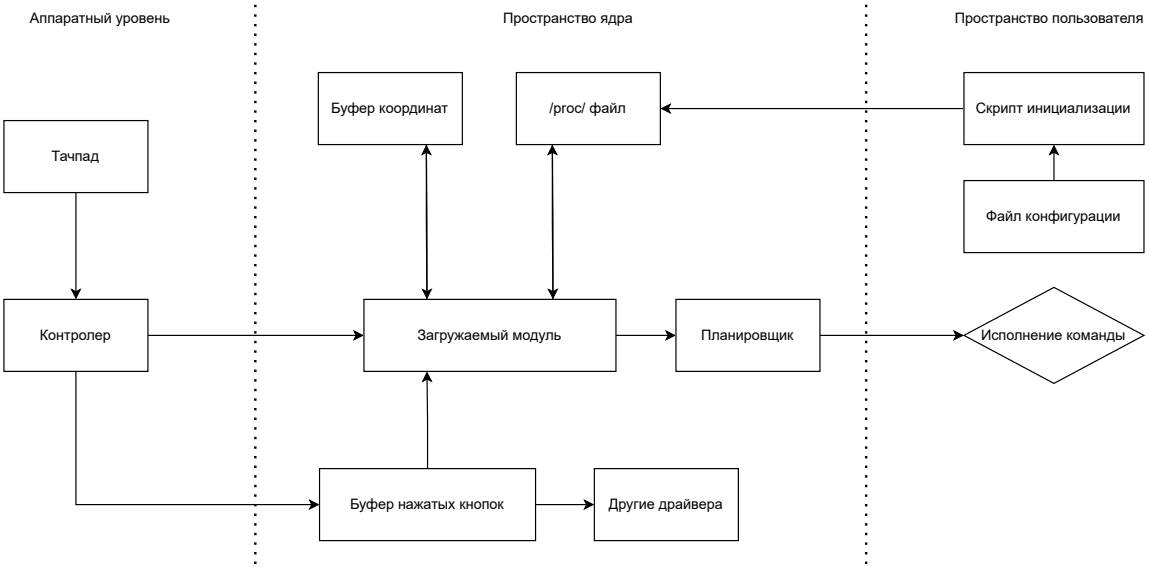

Рисунок 4 – Структура ПО

3 Технологический раздел

3.1 Выбор языка и среды программирования

В качестве языка программирования был выбран язык C. На этом языке реализованы все модули ядра и драйверы операционной системы Linux.

В качестве среды разработки был выбран текстовый редактор Vim.

3.2 Функциональность разработанного ПО

Одновременное касанием несколькими пальцами по тачпаду инициирует запись движения. После удаления пальцев с тачпада, записанное движение анализируется и сопоставляется с одним из шести шаблонов:

1) вертикальное движение вверх;
2) вертикальное движение вниз;
3) горизонтальное движение (влево/вправо);
4) движение в форме латинской буквы V;
5) движение по диагонали с правого верха, влево вниз;
6) движение по диагонали с левого низа, вправо вверх.

Функциональность тачпада сохраняется, ПО не влияет на действия с одним пальцем (перемещение курсора) и двумя пальцами (прокрутка).

3.3 Инициализация структуры драйвера устройства ввода

Листинг 7: Заполнение структур драйвера

```
1  static const struct input_device_id touchpad_gest_ids[] = {
2    {.driver_info = 1},
3    {},
4  };
5
6  static struct input_handler touchpad_gest_handler = {
7    .filter = touchpad_gest_filter,
8    .connect = touchpad_gest_connect,
9    .disconnect = touchpad_gest_disconnect,
10   .name = DEVICE_NAME,
11   .id_table = touchpad_gest_ids,
12 };
```

Поля структуры:

- filter – обработчик событий;
- connect – функция, вызываемая для подключении обработчика к устройству ввода;

- disconnect – функция, вызываемая для отключения обработчика от устройства ввода;
- name – имя обработчика (отображается в /proc/bus/input/handlers);
- id_table – указатель на таблицу идентификаторов устройств ввода, поддерживаемых драйвером.

3.4 Инициализация структуры для создания файла в /proc/

Листинг 8: Заполнение структуры proc

```
1  static struct proc_ops touchpad_gest_fops = {
2    .proc_write = proc_write,
3    .proc_read = proc_read};
```

Поля структуры:

- proc_write – функция записи;
- proc_read – функция чтения;

3.5 Листинг загружаемого модуля ядра

Исходный код загружаемого модуля и файла сборки представлены в приложениях А и Б.

4 Исследовательский раздел

На рисунке 5 показан лог загрузки модуля.

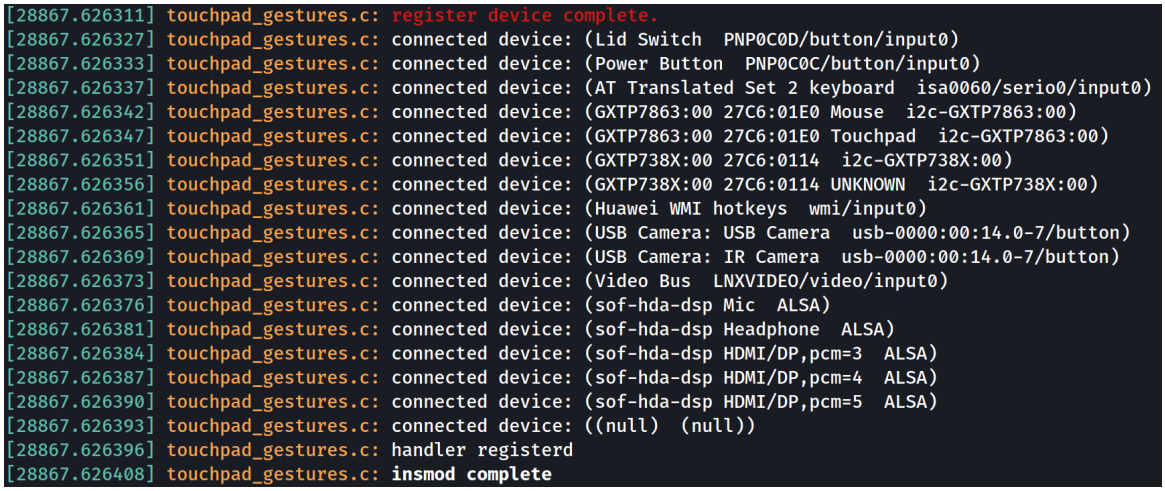

Рисунок 5 – Загрузка модуля

На рисунке 6 показан лог инициализации команд.

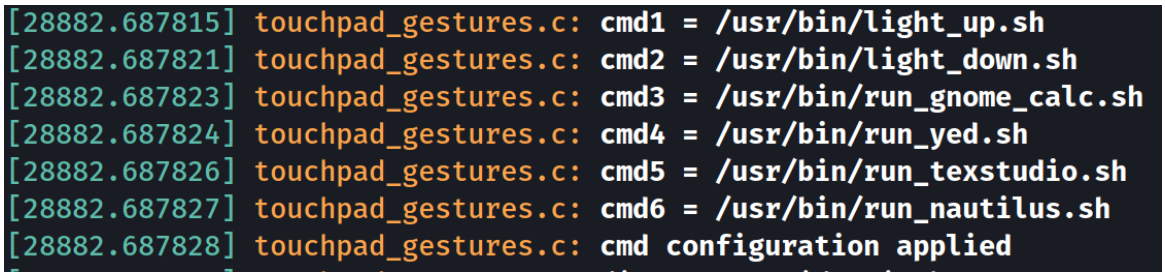

Рисунок 6 – Инициализация команд

На рисунке 7 показана проверка файла в /proc/.

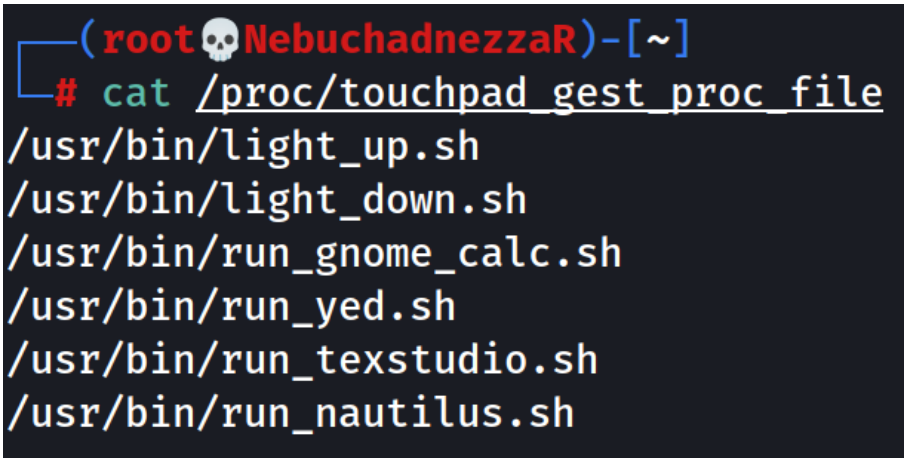

Рисунок 7 – Проверка файла в /proc/

На рисунке 8 показан пример успешной работы.

```
[28545.016829] touchpad_gestures.c: started pattern recording
[28545.331597] touchpad_gestures.c: pattern recording complete
[28545.331732] touchpad_gestures.c: Verical Line, cmd = /usr/bin/light_down.sh
[28545.347444] touchpad_gestures.c: execute status = 0
[28547.300344] touchpad_gestures.c: started pattern recording
[28547.517562] touchpad_gestures.c: pattern recording complete
[28547.517696] touchpad_gestures.c: Verical Line, cmd = /usr/bin/light_up.sh
[28547.534288] touchpad_gestures.c: execute status = 0
[28550.124388] touchpad_gestures.c: started pattern recording
[28550.660217] touchpad_gestures.c: pattern recording complete
[28550.660351] touchpad_gestures.c: V shape /usr/bin/run_yed.sh
[28564.114898] touchpad_gestures.c: execute status = 0
[28570.343442] touchpad_gestures.c: started pattern recording
[28570.641621] touchpad_gestures.c: pattern recording complete
[28570.641754] touchpad_gestures.c: Left Diagonal /usr/bin/run_nautilus.sh
[28570.886935] touchpad_gestures.c: execute status = 0
```

Рисунок 8 – Пример работы

На рисунке 9 показана обработка неизвестного шаблона движения.

```
[28575.350364] touchpad_gestures.c: started pattern recording
[28576.063747] touchpad_gestures.c: pattern recording complete
[28576.063873] touchpad_gestures.c: unknown gesture
[28576.063876] touchpad_gestures.c: execute status = 0
```

Рисунок 9 – Неизвестный шаблон

На рисунке 10 показана выгрузка модуля.

```
[28895.577141] touchpad_gestures.c: disconnect Lid Switch
[28895.631130] touchpad_gestures.c: disconnect Power Button
[28895.683214] touchpad_gestures.c: disconnect AT Translated Set 2 keyboard
[28895.747085] touchpad_gestures.c: disconnect GXTP7863:00 27C6:01E0 Mouse
[28895.791134] touchpad_gestures.c: disconnect GXTP7863:00 27C6:01E0 Touchpad
[28895.859046] touchpad_gestures.c: disconnect GXTP738X:00 27C6:0114
[28895.919021] touchpad_gestures.c: disconnect GXTP738X:00 27C6:0114 UNKNOWN
[28895.995093] touchpad_gestures.c: disconnect Huawei WMI hotkeys
[28896.062996] touchpad_gestures.c: disconnect USB Camera: USB Camera
[28896.127060] touchpad_gestures.c: disconnect USB Camera: IR Camera
[28896.183042] touchpad_gestures.c: disconnect Video Bus
[28896.259120] touchpad_gestures.c: disconnect sof-hda-dsp Mic
[28896.315015] touchpad_gestures.c: disconnect sof-hda-dsp Headphone
[28896.367067] touchpad_gestures.c: disconnect sof-hda-dsp HDMI/DP,pcm=3
[28896.427071] touchpad_gestures.c: disconnect sof-hda-dsp HDMI/DP,pcm=4
[28896.487004] touchpad_gestures.c: disconnect sof-hda-dsp HDMI/DP,pcm=5
[28896.563024] touchpad_gestures.c: disconnect (null)
[28896.639187] touchpad_gestures.c: rmmod complete
```

Рисунок 10 – Выгрузка модуля

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы было разработано ПО, позволяющее задавать определённые действия системы движением нескольких пальцев по тачпаду. Были достигнуты следующие задачи:

1) проведён анализ существующих подходов к обработке устройств ввода;
2) проведён анализ существующих способов задания действий движением нескольких пальцев по тачпаду;
3) проведён анализ возможности выполнения системой заданных действий;
4) разработаны алгоритмы, необходимые для реализации ПО;
5) разработано ПО, предоставляющее требуемую функциональность;
6) проведено исследование разработанного ПО.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. The Linux Kernel Documentation [Электронный ресурс]. Режим доступа: https://www.kernel.org/doc/html/latest/, свободный (дата обращения: 21 декабря 2023 г.).

2. Linux source code [Электронный ресурс]. Режим доступа: https://elixir.bootlin.com/linux/latest/source, свободный (дата обращения: 21 декабря 2023 г.).

ПРИЛОЖЕНИЕ А

Листинг 9: Реализация загружаемого модуля ядра

```
#include <linux/input.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/timekeeping.h>
#include <linux/uaccess.h>

#define DEVICE_NAME "touchpad_gestures"
#define PROCFS_MAX_SIZE 1000
#define MAX_GEST_CNT 6
#define MAX_BUFF_SIZE 350

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Artyom Bogachenko");
MODULE_DESCRIPTION("Touchpad gestures");
MODULE_VERSION("1.0.0");

static int count_x = 0;
static int count_y = 0;
static int position_x[MAX_BUFF_SIZE] = {0};
static int position_y[MAX_BUFF_SIZE] = {0};
static int y_end, x_end;
char proc_buf[PROCFS_MAX_SIZE] = {0};

char cmd1[300] = {0};
char cmd2[300] = {0};
char cmd3[300] = {0};
char cmd4[300] = {0};
char cmd5[300] = {0};
char cmd6[300] = {0};

static char *argv[3];
static char *envp[] = {"HOME=/root",
  "TERM=linux",
  "USER=root",
  "DISPLAY=:1",
  "DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/0/bus",
  "XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg",
  "XDG_RUNTIME_DIR=/run/user/0",
```

```
39      "XDG_SESSION_TYPE=x11",
40      "PATH=/sbin:/usr/sbin:/bin:/usr/bin",
41      NULL};
42
43    static struct state {
44      char execute[300];
45    } cmd_state[MAX_GEST_CNT];
46
47    struct work_arg_struct {
48      struct work_struct work;
49      int data;
50    };
51
52    static struct work_arg_struct my_work;
53
54    static void parse_pattern(struct work_struct *work) {
55      int result = 0;
56
57      argv[0] = "/usr/bin/zsh";
58      argv[2] = NULL;
59      if (abs(position_x[0] - x_end) <= 200 && abs(y_end - position_y
          [0]) > 700) {
60        int up = position_y[0] > y_end ? 1 : 0;
61        printk("touchpad_gestures.c: Verical Line, cmd = %s", up ?
            cmd1 : cmd2);
62        argv[1] = up ? cmd1 : cmd2;
63        result = call_usermodehelper(argv[0], argv, envp,
            UMH_WAIT_PROC);
64      } else if (abs(y_end - position_y[0]) <= 200 &&
65      abs(position_y[count_y / 2] - position_y[0]) <= 100 &&
66      abs(position_x[0] - x_end) > 700) {
67        printk("touchpad_gestures.c: Horizonal Line %s", cmd3);
68        argv[1] = cmd3;
69        result = call_usermodehelper(argv[0], argv, envp,
            UMH_WAIT_PROC);
70      } else if (abs(position_x[0] - x_end) > 700 &&
71      (position_y[count_y / 2] > position_y[0]) &&
72      abs(y_end - position_y[0]) <= 200) {
73        printk("touchpad_gestures.c: V shape %s", cmd4);
74        argv[1] = cmd4;
```

```
75        result = call_usermodehelper(argv[0], argv, envp,
            UMH_WAIT_PROC);
76      } else if ((position_x[0] > x_end) && (position_y[0] < y_end))
            {
77        printk("touchpad_gestures.c: Right Diagonal %s", cmd5);
78        argv[1] = cmd5;
79        result = call_usermodehelper(argv[0], argv, envp,
            UMH_WAIT_PROC);
80      } else if ((position_x[0] < x_end) && (position_y[0] > y_end))
            {
81        printk("touchpad_gestures.c: Left Diagonal %s", cmd6);
82        argv[1] = cmd6;
83        result = call_usermodehelper(argv[0], argv, envp,
            UMH_WAIT_PROC);
84      } else {
85        printk("touchpad_gestures.c: unknown gesture");
86      }
87      printk("touchpad_gestures.c: execute status = %d", result);
88
89      count_x = 0;
90      count_y = 0;
91      for (int i = 0; i < MAX_BUFF_SIZE; ++i) position_x[i] = 0;
92      for (int i = 0; i < MAX_BUFF_SIZE; ++i) position_y[i] = 0;
93    }
94
95    static struct proc_dir_entry *proc_file;
96
97    bool connected = false;
98    bool start_record = false;
99    int tracking_id = -1;
100
101   static bool touchpad_gest_filter(struct input_handle *handle,
          unsigned int type,
102   unsigned int code, int value) {
103     if (code == SYN_REPORT) return 0;
104
105     if (code == ABS_MT_SLOT) {
106       bool slot_matched = (value == 2);
107
108       if (!connected) {
109         connected = slot_matched;
```

```
110         }
111
112         if (connected) {
113           start_record = slot_matched;
114         }
115
116         return 0;
117       }
118
119       if (start_record && code == ABS_MT_TRACKING_ID) {
120         if (tracking_id == -1) {
121           printk("touchpad_gestures.c: started pattern recording");
122           tracking_id = value;
123           return 0;
124         }
125
126         if (value == -1) {
127           tracking_id = value;
128           connected = false;
129           start_record = false;
130           printk("touchpad_gestures.c: pattern recording complete");
131           schedule_work(&my_work.work);
132           return 0;
133         }
134
135         return 0;
136       }
137
138       if (start_record && code == ABS_MT_POSITION_X) {
139         if (count_x < MAX_BUFF_SIZE) {
140           position_x[count_x++] = value;
141         }
142         x_end = value;
143       }
144
145       if (start_record && code == ABS_MT_POSITION_Y) {
146         if (count_y < MAX_BUFF_SIZE) {
147           position_y[count_y++] = value;
148         }
149         y_end = value;
150       }
```

```
151
152     return 0;
153   }
154
155   static int touchpad_gest_connect(struct input_handler *handler,
156   struct input_dev *dev,
157   const struct input_device_id *id) {
158     struct input_handle *handle;
159     int error;
160
161     handle = kzalloc(sizeof(struct input_handle), GFP_KERNEL);
162     if (!handle) return -ENOMEM;
163
164     handle->dev = dev;
165     handle->handler = handler;
166     handle->name = "touchpad_gestures_handle";
167
168     error = input_register_handle(handle);
169     if (error) goto err_free_handle;
170
171     error = input_open_device(handle);
172     if (error) goto err_unregister_handle;
173
174     printk(KERN_INFO "touchpad_gestures.c: connected device: (%s %
          s)\n",
175     dev->name, dev->phys);
176
177     return 0;
178
179     err_unregister_handle:
180     input_unregister_handle(handle);
181     err_free_handle:
182     kfree(handle);
183     return error;
184   }
185
186   static void touchpad_gest_disconnect(struct input_handle *handle)
        {
187     printk(KERN_INFO "touchpad_gestures.c: disconnect %s\n", handle
          ->dev->name);
188
```

```
189     input_close_device(handle);
190     input_unregister_handle(handle);
191     kfree(handle);
192   }
193
194   static ssize_t proc_read(struct file *filp, char __user *buf,
        size_t size,
195   loff_t *off) {
196     loff_t offset = *off;
197     size_t remaining;
198
199     if (offset < 0) return -EINVAL;
200
201     if (offset >= MAX_BUFF_SIZE || size == 0) return 0;
202
203     if (size > MAX_BUFF_SIZE - offset) size = MAX_BUFF_SIZE -
          offset;
204
205     remaining = copy_to_user(buf, proc_buf + offset, size);
206     if (remaining == size) {
207       printk(KERN_ERR "touchpad_gestures.c: copy_to_user failed\n")
            ;
208       return -EFAULT;
209     }
210
211     size -= remaining;
212     *off = offset + size;
213     return size;
214   }
215
216   static ssize_t proc_write(struct file *fp, const char *buf,
        size_t len,
217   loff_t *off) {
218     int i, index = 0, idx = 0;
219     if (len > PROCFS_MAX_SIZE) {
220       return -EFAULT;
221     }
222
223     if (copy_from_user(proc_buf, buf, len)) {
224       return -EFAULT;
225     }
```

```
226     for (i = 0; i < len; i++) {
227       if (proc_buf[i] == '\n') {
228         cmd_state[index].execute[idx++] = '\0';
229         idx = 0;
230         index++;
231       } else {
232         cmd_state[index].execute[idx++] = proc_buf[i];
233       }
234     }
235     for (i = 0; i < MAX_GEST_CNT; i++) {
236       if (i == 0) {
237         strcpy(cmd1, cmd_state[i].execute);
238         printk("touchpad_gestures.c: cmd1 = %s", cmd1);
239       } else if (i == 1) {
240         strcpy(cmd2, cmd_state[i].execute);
241         printk("touchpad_gestures.c: cmd2 = %s", cmd2);
242       } else if (i == 2) {
243         strcpy(cmd3, cmd_state[i].execute);
244         printk("touchpad_gestures.c: cmd3 = %s", cmd3);
245       } else if (i == 3) {
246         strcpy(cmd4, cmd_state[i].execute);
247         printk("touchpad_gestures.c: cmd4 = %s", cmd4);
248       } else if (i == 4) {
249         strcpy(cmd5, cmd_state[i].execute);
250         printk("touchpad_gestures.c: cmd5 = %s", cmd5);
251       } else if (i == 5) {
252         strcpy(cmd6, cmd_state[i].execute);
253         printk("touchpad_gestures.c: cmd6 = %s", cmd6);
254       }
255     }
256     printk("touchpad_gestures.c: cmd configuration applied");
257 
258     return len;
259   }
260 
261   static const struct input_device_id touchpad_gest_ids[] = {
262     {.driver_info = 1}, /* Matches all devices */
263     {},
264   };
265 
266   static struct input_handler touchpad_gest_handler = {
```

```
267     .filter = touchpad_gest_filter,
268     .connect = touchpad_gest_connect,
269     .disconnect = touchpad_gest_disconnect,
270     .name = DEVICE_NAME,
271     .id_table = touchpad_gest_ids,
272   };
273
274   static struct proc_ops touchpad_gest_fops = {.proc_write =
        proc_write,
275     .proc_read = proc_read};
276
277   static int __init touchpad_gest_init(void) {
278     int error;
279
280     error = input_register_handler(&touchpad_gest_handler);
281
282     if (error) {
283       printk(KERN_ERR
284       "touchpad_gestures.c: registering input handler failed with
            (%d)\n",
285       error);
286     } else {
287       printk(KERN_INFO "touchpad_gestures.c: handler registerd");
288     }
289
290     proc_file =
291     proc_create("touchpad_gest_proc_file", 0666, NULL, &
          touchpad_gest_fops);
292     if (!proc_file) {
293       printk("touchpad_gestures.c: couldn't create proc_file");
294       return -ENOMEM;
295     }
296
297     INIT_WORK(&my_work.work, parse_pattern);
298     printk("touchpad_gestures.c: insmod complete");
299     return 0;
300
301     err_exit:
302     return error;
303   }
304
```

```
305  static void __exit touchpad_gest_exit(void) {
306    input_unregister_handler(&touchpad_gest_handler);
307    remove_proc_entry("touchpad_gest_proc_file", NULL);
308    printk("touchpad_gestures.c: rmmod complete");
309  }
310
311  module_init(touchpad_gest_init);
312  module_exit(touchpad_gest_exit);
```

ПРИЛОЖЕНИЕ Б

Листинг 10: Makefile

```
VERSION = $(shell uname -r)
KDIR = /lib/modules/$(VERSION)/build
PWD = $(shell pwd)
TARGET = touchpad_gestures
obj-m := $(TARGET).o
default:
$(MAKE) -C $(KDIR) M=$(PWD) modules
clean:
@rm -f *.o .*.cmd .*.flags *.mod.c *.order .md.o.d
@rm -f .*.*.cmd *~ *.*~ TODO.*
@rm -fR .tmp*
@rm -rf .tmp_versions
disclean: clean
@rm -f *.ko *.symvers *.mod
```