

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

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

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

2023 г.

РЕФЕРАТ

Расчетно-пояснительная записка 30 с., 11 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	5
1 Аналитический раздел	6
1.1 Постановка задачи	6
1.2 Загружаемый модуль ядра	6
1.2.1 Драйвера	7
1.2.2 Обработчик устройства ввода	7
1.3 Передача информации из пространства ядра в пространство поль- зователя	7
1.4 Прерывания	8
1.5 Запуск процессов пользователя из пространства ядра	9
1.5.1 call_usermodehelper	9
1.5.2 Планировщик ядра	9
1.5.3 Переключение потоков	10
1.6 Multi Touch протокол	10
1.7 Выводы	11
2 Конструкторский раздел	12
2.1 Алгоритм работы обработчика устройства ввода	12
2.2 Алгоритм обработки шаблона	13
2.3 Загружаемый модуль ядра	14
3 Технологический раздел	15
3.1 Выбор языка и среды программирования	15
3.2 Структура ПО	15
3.3 Функциональность разработанного ПО	15
3.4 Инициализация структуры обработчика устройства ввода	15
3.5 Инициализация структуры для создания файла в /proc/	16
3.6 Листинг загружаемого модуля ядра	16
4 Исследовательский раздел	17
ЗАКЛЮЧЕНИЕ	19

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	20
ПРИЛОЖЕНИЕ А	21
ПРИЛОЖЕНИЕ Б	30

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 Драйвера

В современных ядрах Linux изменение функциональности внешних устройств возможно с помощью драйверов верхнего уровня, представляющих собой загружаемые модули ядра. Драйверы полностью скрывают детали, касающиеся работы устройства и предоставляют чёткий программный интерфейс для работы с аппаратурой. В Linux имеется три типа драйверов:

- 1) драйверы первого типа являются частью программного кода ядра. Соответствующие устройства автоматически обнаруживаются системой и становятся доступны для 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.

1.2.2 Обработчик устройства ввода

Драйвер интерактивного устройства ввода — это модуль, обеспечивающий возможность взаимодействия с устройством через прерывания.

Алгоритм регистрации обработчика прерывания устройства ввода [1] может быть представлен в виде следующей последовательности действий:

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

1.3 Передача информации из пространства ядра в пространство пользователя

В файловой системе (ФС) `proc` можно создавать свои файлы, ссылки, директории. Структура `proc_dir_entry`, определённая 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`

```
1 struct proc_ops {  
2     unsigned int proc_flags;  
3     int (*proc_open)(struct inode *, struct file *);  
4     ssize_t (*proc_read)(struct file *, char __user *, size_t, loff_t  
5         *);  
6     ...  
7     ssize_t (*proc_write)(struct file *, const char __user *, size_t,  
8         loff_t *);  
9     ...  
10    int (*proc_release)(struct inode *, struct file *);  
11    ...  
12    } __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 используется 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 0x21h`), 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.

1.5 Запуск процессов пользователя из пространства ядра

1.5.1 call_usermodehelper

Использование функции `call_usermodehelper` [2] позволяет выполнять процесс пользователя из пространства 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.

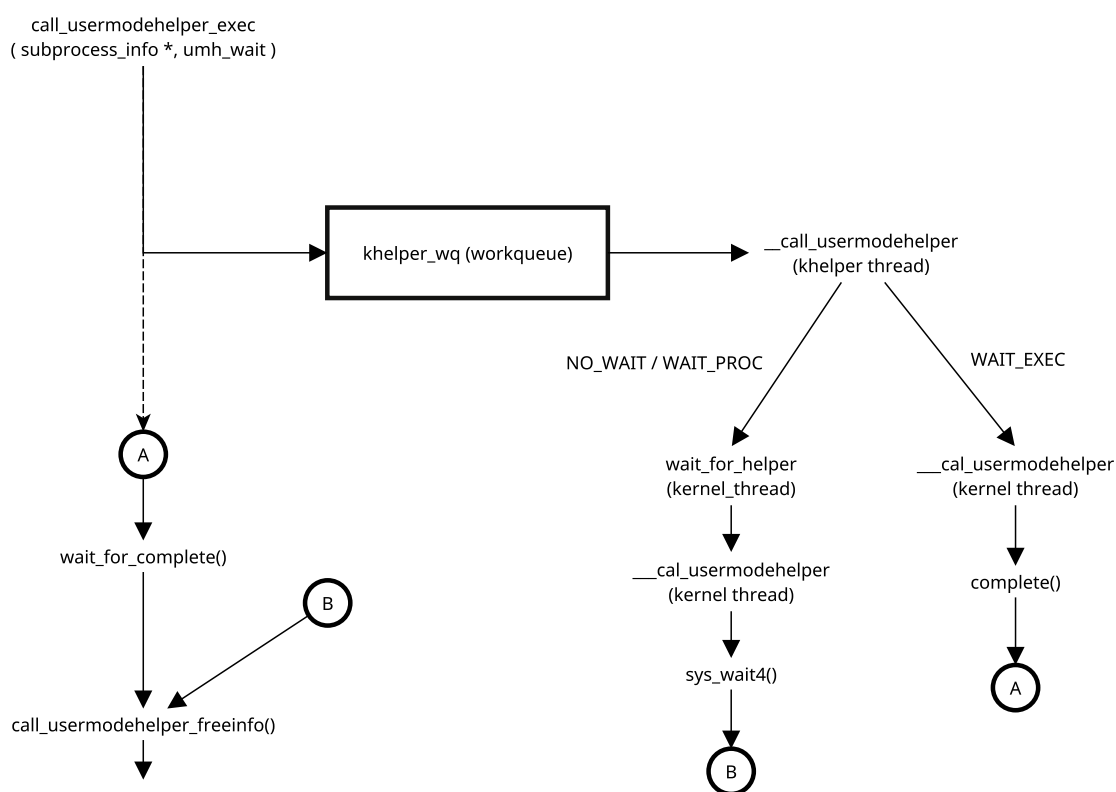


Рисунок 1 – Внутренняя реализация функции `call_usermodehelper`

1.5.2 Планировщик ядра

Для выполнения процесса в пространстве пользователя необходимо вызвать функцию `call_usermodehelper` с флагом ожидания `UMH_WAIT_PROC`. В таком случае мы будем ожидать завершения процесса. В контексте обработчика устройства ввода это невозможно, обработчик находится в неделимом контексте, который не допускает ожидание.

Для это можно воспользоваться общей очередью работ ядра. При планировании работы общим планировщиком ядра, системой создаётся отдельный

поток, в котором будет выполнена работа. В таком случае ожидание возможно и процесс будет выполнен.

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.

1.5.3 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 Multi Touch протокол

Для определения движения нескольких пальцев по тачпаду следует 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 2.

Листинг 2: Поток событий для двух пальцев

```
1 ABS_MT_SLOT 0
2 ABS_MT_TRACKING_ID 45
3 ABS_MT_POSITION_X x[0]
4 ABS_MT_POSITION_Y y[0]
5 ABS_MT_SLOT 1
6 ABS_MT_TRACKING_ID 46
7 ABS_MT_POSITION_X x[1]
```

8	ABS_MT_POSITION_Y y[1]
9	SYN_REPORT

1.7 Выводы

В результате проведённого анализа было заключено следующее:

- 1) реализовать алгоритм определения шаблона;
- 2) для выполнения заданных действий использовать общую очередь работ;
- 3) реализовать обработчик устройства ввода в виде загружаемого модуля ядра;
- 4) для задания действий системы воспользоваться скриптовым языком `bash`;
- 5) для передачи конфигурации в загружаемый модуль использовать виртуальную файловую систему `/proc/`.

2 Конструкторский раздел

2.1 Алгоритм работы обработчика устройства ввода

На рисунке 2 представлен алгоритм работы обработчика устройства ввода.

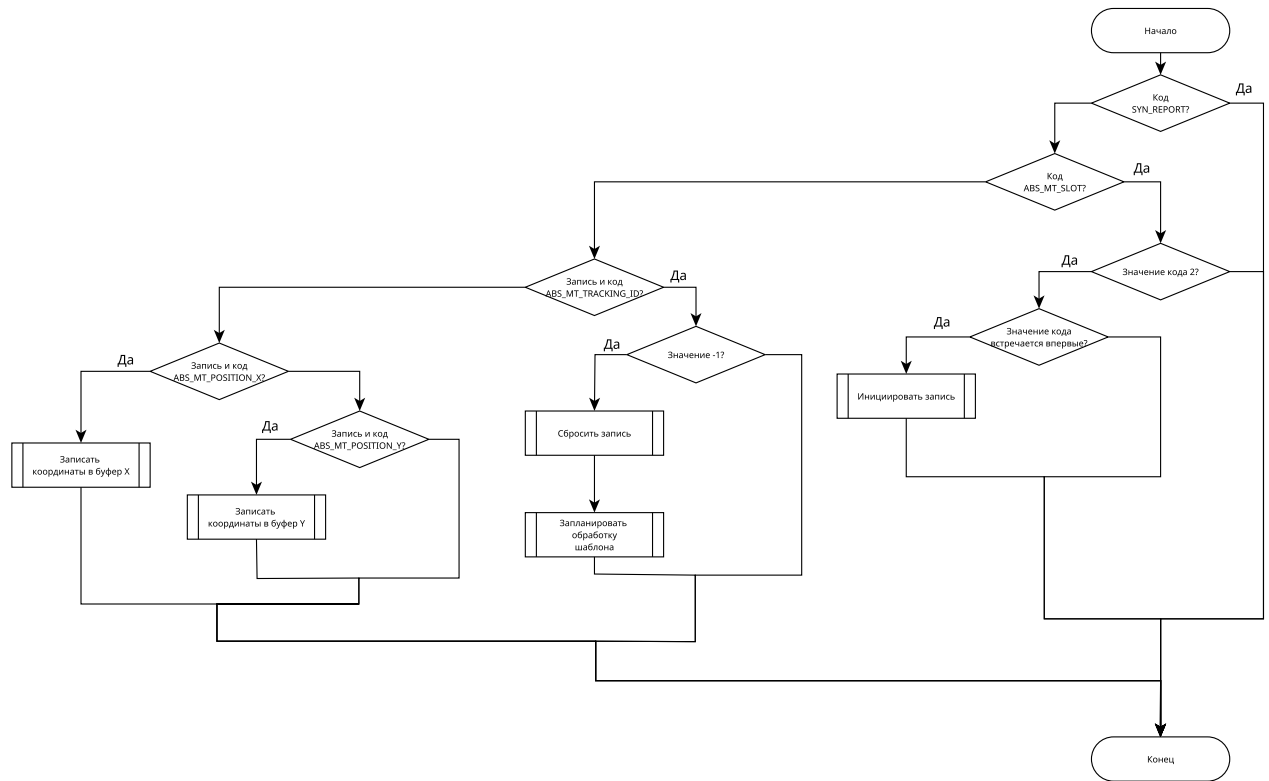


Рисунок 2 – Алгоритм работы обработчика

2.2 Алгоритм обработки шаблона

На рисунке 3 представлен алгоритм обработки шаблона.

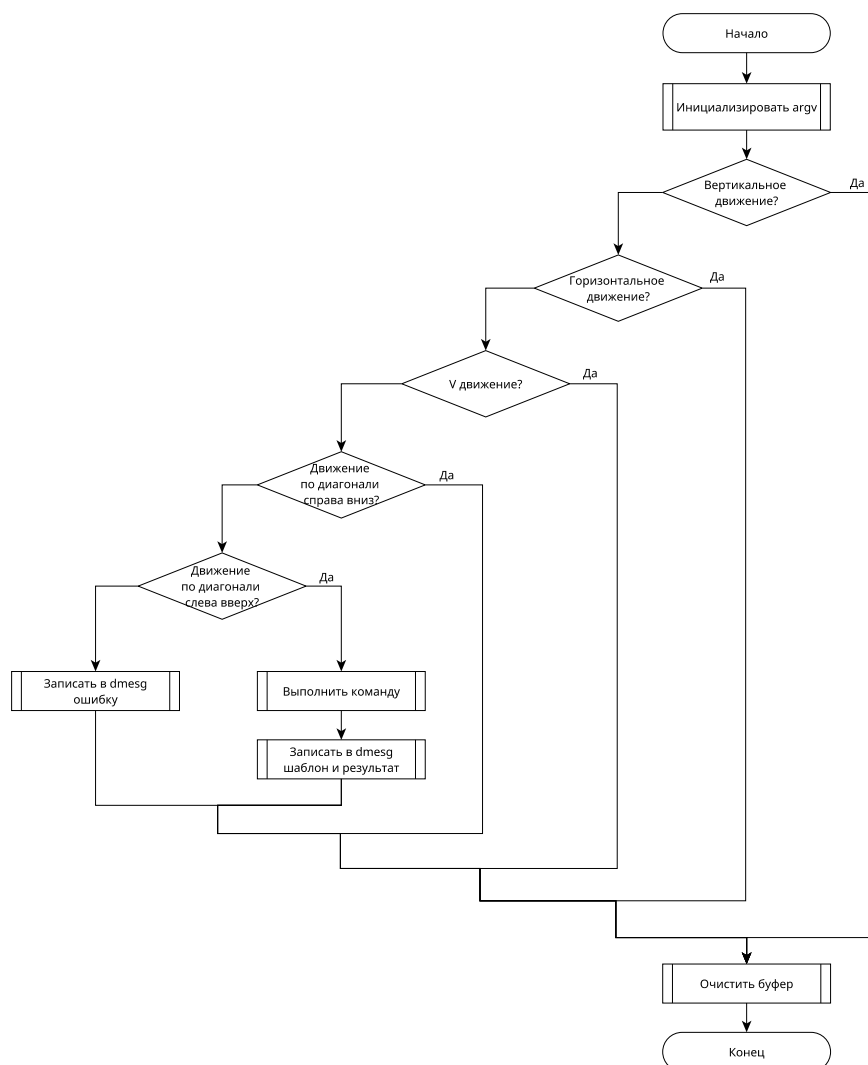


Рисунок 3 – Алгоритм обработки шаблона

2.3 Загружаемый модуль ядра

На рисунке 4 представлена схема инициализации загружаемого модуля ядра.

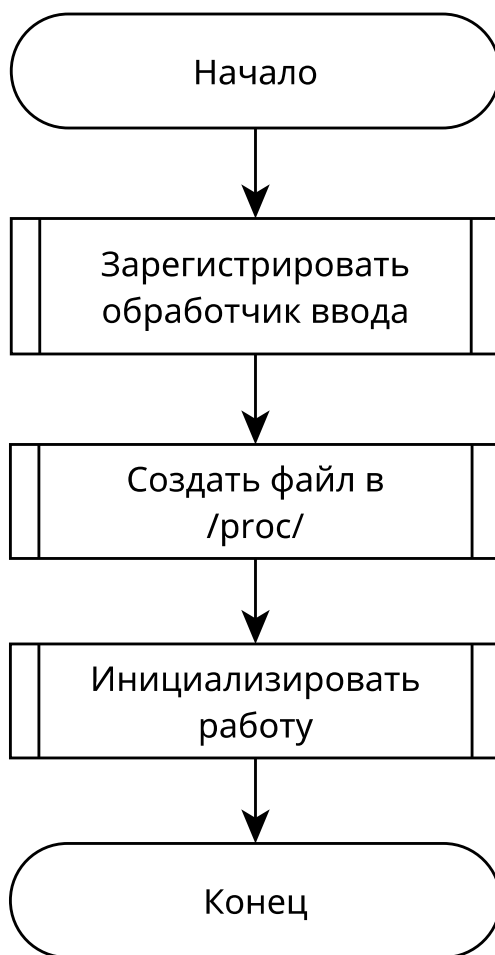


Рисунок 4 – Схема инициализации загружаемого модуля ядра

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 Структура ПО

На рисунке 5 представлена структура конечного ПО.

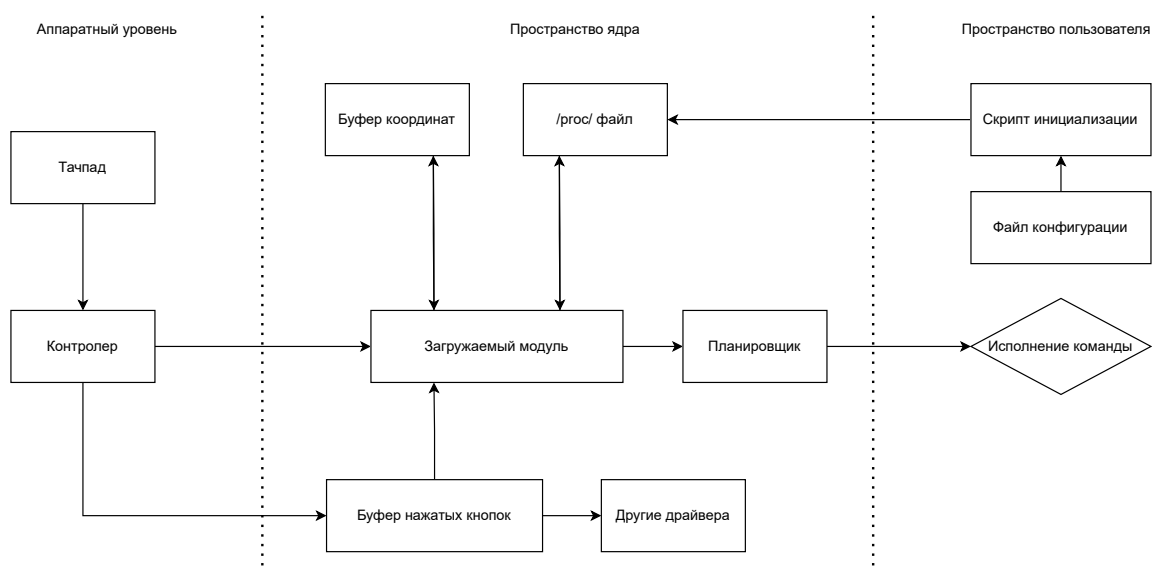


Рисунок 5 – Структура ПО

3.3 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 верх, 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).

3.4 Инициализация структуры обработчика устройства ввода

Листинг 3: Заполнение структур обработчика ввода

```
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 подключения 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.

3.5 Инициализация структуры для создания файла в /proc/

Листинг 4: 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;

3.6 Листинг загружаемого модуля ядра

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

4 Исследовательский раздел

На рисунке 6 показан лог загрузки модуля.

```
[28867.626311] touchpad_gestures.c: register device complete.
[28867.626327] touchpad_gestures.c: connected device: (Lid Switch PNP0C0D/button/input0)
[28867.626333] touchpad_gestures.c: connected device: (Power Button PNP0C0C/button/input0)
[28867.626337] touchpad_gestures.c: connected device: (AT Translated Set 2 keyboard isa0060/serio0/input0)
[28867.626342] touchpad_gestures.c: connected device: (GXTP7863:00 27C6:01E0 Mouse i2c-GXTP7863:00)
[28867.626347] touchpad_gestures.c: connected device: (GXTP7863:00 27C6:01E0 Touchpad i2c-GXTP7863:00)
[28867.626351] touchpad_gestures.c: connected device: (GXTP738X:00 27C6:0114 i2c-GXTP738X:00)
[28867.626356] touchpad_gestures.c: connected device: (GXTP738X:00 27C6:0114 UNKNOWN i2c-GXTP738X:00)
[28867.626361] touchpad_gestures.c: connected device: (Huawei WMI hotkeys wmi/input0)
[28867.626365] touchpad_gestures.c: connected device: (USB Camera: USB Camera usb-0000:00:14.0-7/button)
[28867.626369] touchpad_gestures.c: connected device: (USB Camera: IR Camera usb-0000:00:14.0-7/button)
[28867.626373] touchpad_gestures.c: connected device: (Video Bus LNXVIDEO/video/input0)
[28867.626376] touchpad_gestures.c: connected device: (sof-hda-dsp Mic ALSA)
[28867.626381] touchpad_gestures.c: connected device: (sof-hda-dsp Headphone ALSA)
[28867.626384] touchpad_gestures.c: connected device: (sof-hda-dsp HDMI/DP,pcm=3 ALSA)
[28867.626387] touchpad_gestures.c: connected device: (sof-hda-dsp HDMI/DP,pcm=4 ALSA)
[28867.626390] touchpad_gestures.c: connected device: (sof-hda-dsp HDMI/DP,pcm=5 ALSA)
[28867.626393] touchpad_gestures.c: connected device: ((null) (null))
[28867.626396] touchpad_gestures.c: handler registerd
[28867.626408] touchpad_gestures.c: insmod complete
```

Рисунок 6 – Загрузка модуля

На рисунке 7 показан лог инициализации команд.

```
[28882.687815] touchpad_gestures.c: cmd1 = /usr/bin/light_up.sh
[28882.687821] touchpad_gestures.c: cmd2 = /usr/bin/light_down.sh
[28882.687823] touchpad_gestures.c: cmd3 = /usr/bin/run_gnome_calc.sh
[28882.687824] touchpad_gestures.c: cmd4 = /usr/bin/run_yed.sh
[28882.687826] touchpad_gestures.c: cmd5 = /usr/bin/run_texstudio.sh
[28882.687827] touchpad_gestures.c: cmd6 = /usr/bin/run_nautilus.sh
[28882.687828] touchpad_gestures.c: cmd configuration applied
```

Рисунок 7 – Инициализация команд

На рисунке 8 показана проверка файла в /proc/.

```
(root👁NebuchadnezzaR)-[~]
# cat /proc/touchpad_gest_proc_file
/usr/bin/light_up.sh
/usr/bin/light_down.sh
/usr/bin/run_gnome_calc.sh
/usr/bin/run_yed.sh
/usr/bin/run_texstudio.sh
/usr/bin/run_nautilus.sh
```

Рисунок 8 – Проверка файла в /proc/

На рисунке 9 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 9 – Пример работы

На рисунке 10 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 10 – Неизвестный шаблон

На рисунке 11 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: rmmmod complete
```

Рисунок 11 – 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: 27 декабря 2023 г.).
2. Linux source code [Электронный ресурс]. Режим доступа: <https://elixir.bootlin.com/linux/latest/source>, свободный (дата обращения: 27 декабря 2023 г.).

ПРИЛОЖЕНИЕ А

Листинг 5: Реализация загружаемого модуля ядра

```
1  #include <linux/input.h>
2  #include <linux/module.h>
3  #include <linux/proc_fs.h>
4  #include <linux/timekeeping.h>
5  #include <linux/uaccess.h>
6
7  #define DEVICE_NAME "touchpad_gestures"
8  #define PROCFS_MAX_SIZE 1000
9  #define MAX_GEST_CNT 6
10 #define MAX_BUFF_SIZE 350
11
12 MODULE_LICENSE("GPL");
13 MODULE_AUTHOR("Artyom Bogachenko");
14 MODULE_DESCRIPTION("Touchpad gestures");
15 MODULE_VERSION("1.0.0");
16
17 static int count_x = 0;
18 static int count_y = 0;
19 static int position_x[MAX_BUFF_SIZE] = {0};
20 static int position_y[MAX_BUFF_SIZE] = {0};
21 static int y_end, x_end;
22 char proc_buf[PROCFS_MAX_SIZE] = {0};
23
24 char cmd1[300] = {0};
25 char cmd2[300] = {0};
26 char cmd3[300] = {0};
27 char cmd4[300] = {0};
28 char cmd5[300] = {0};
29 char cmd6[300] = {0};
30
31 static char *argv[3];
32 static char *envp[] = {"HOME=/root",
33     "TERM=linux",
34     "USER=root",
35     "DISPLAY=:1",
36     "DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/0/bus",
37     "XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg",
38     "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
60             [0]) > 700) {
61             int up = position_y[0] > y_end ? 1 : 0;
62             printk("touchpad_gestures.c: Vertical Line, cmd = %s", up ?
63                 cmd1 : cmd2);
64             argv[1] = up ? cmd1 : cmd2;
65             result = call_usermodehelper(argv[0], argv, envp,
66                 UMH_WAIT_PROC);
67         } else if (abs(y_end - position_y[0]) <= 200 &&
68             abs(position_y[count_y / 2] - position_y[0]) <= 100 &&
69             abs(position_x[0] - x_end) > 700) {
70             printk("touchpad_gestures.c: Horizontal Line %s", cmd3);
71             argv[1] = cmd3;
72             result = call_usermodehelper(argv[0], argv, envp,
73                 UMH_WAIT_PROC);
74         } else if (abs(position_x[0] - x_end) > 700 &&
75             (position_y[count_y / 2] > position_y[0]) &&
76             abs(y_end - position_y[0]) <= 200) {
77             printk("touchpad_gestures.c: V shape %s", cmd4);
78             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 %
175         s)\n",
176     dev->name, dev->phys);
177
178     return 0;
179
180     err_unregister_handle:
181     input_unregister_handle(handle);
182     err_free_handle:
183     kfree(handle);
184     return error;
185 }
186
187 static void touchpad_gest_disconnect(struct input_handle *handle)
188     {
189     printk(KERN_INFO "touchpad_gestures.c: disconnect %s\n", handle
190         ->dev->name);

```

```

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,
195     size_t size,
196     loff_t *off) {
197     loff_t offset = *off;
198     size_t remaining;
199
200     if (offset < 0) return -EINVAL;
201
202     if (offset >= MAX_BUFF_SIZE || size == 0) return 0;
203
204     if (size > MAX_BUFF_SIZE - offset) size = MAX_BUFF_SIZE -
205         offset;
206
207     remaining = copy_to_user(buf, proc_buf + offset, size);
208     if (remaining == size) {
209         printk(KERN_ERR "touchpad_gestures.c: copy_to_user failed\n");
210         ;
211         return -EFAULT;
212     }
213
214     size -= remaining;
215     *off = offset + size;
216     return size;
217 }
218
219 static ssize_t proc_write(struct file *fp, const char *buf,
220     size_t len,
221     loff_t *off) {
222     int i, index = 0, idx = 0;
223     if (len > PROCFS_MAX_SIZE) {
224         return -EFAULT;
225     }
226
227     if (copy_from_user(proc_buf, buf, len)) {
228         return -EFAULT;
229     }

```

```

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 =
275     proc_write ,
276     .proc_read = proc_read};
277
278 static int __init touchpad_gest_init(void) {
279     int error;
280
281     error = input_register_handler(&touchpad_gest_handler);
282
283     if (error) {
284         printk(KERN_ERR
285             "touchpad_gestures.c: registering input handler failed with
286             (%d)\n",
287             error);
288     } else {
289         printk(KERN_INFO "touchpad_gestures.c: handler registerd");
290     }
291
292     proc_file =
293     proc_create("touchpad_gest_proc_file", 0666, NULL, &
294         touchpad_gest_fops);
295     if (!proc_file) {
296         printk("touchpad_gestures.c: couldn't create proc_file");
297         return -ENOMEM;
298     }
299
300     INIT_WORK(&my_work.work, parse_pattern);
301     printk("touchpad_gestures.c: insmod complete");
302     return 0;
303
304     err_exit:
305     return error;
306 }

```

```
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 6: Makefile

```
1  VERSION = $(shell uname -r)
2  KDIR = /lib/modules/$(VERSION)/build
3  PWD = $(shell pwd)
4  TARGET = touchpad_gestures
5  obj-m := $(TARGET).o
6  default:
7  $(MAKE) -C $(KDIR) M=$(PWD) modules
8  clean:
9  @rm -f *.o *.cmd *.flags *.mod.c *.order .md.o.d
10 @rm -f *.*.cmd *~ *.~ TODO.*
11 @rm -fR .tmp*
12 @rm -rf .tmp_versions
13 disclean: clean
14 @rm -f *.ko *.symvers *.mod
```