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 «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Программное обеспечение для ввода символов с помощью компьютерной мыши»

Студент ИУ7-75Б _____ Е.Б.Гришин
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель _____ Н.Ю.Рязанова
(Подпись, дата) (И.О.Фамилия)

2021 г.

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)

УТВЕРЖДАЮ
Заведующий кафедрой ___ИУ7____
(Индекс)
_____ _И.В. Рудаков_____
(И.О.Фамилия)
« _____ » _____ 2021 г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине _____Операционные системы_____

Студент группы ____ИУ7-75Б____

_____Гришин Егор Борисович_____
(Фамилия, имя, отчество)

Тема курсового проекта _____
_____Программное обеспечение для ввода символов с помощью компьютерной мыши_____
_____

Направленность КП (учебный, исследовательский, практический, производственный, др.)
_____учебный_____
Источник тематики (кафедра, предприятие, НИР) _____кафедра_____

График выполнения проекта: 25% к _4_ нед., 50% к _7_ нед., 75% к 11 нед., 100% к _14 нед.

Задание *Разработать программное обеспечение, позволяющее с помощью компьютерной мыши вводить символы.*

Оформление курсового проекта:

Расчетно-пояснительная записка на 25-30 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
На защиту проекта должна быть предоставлена презентация, состоящая из 15-20 слайдов.
На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, интерфейс, результаты проведенных исследований.

Дата выдачи задания « 11 » октября 2021 г.

Руководитель курсового проекта _____ _____Н.Ю. Рязанова___
(Подпись, дата) (И.О.Фамилия)

Студент _____ _____Е.Б. Гришин_____
(Подпись, дата) (И.О.Фамилия)

Оглавление

Введение

Азбука Морзе – способ знакового кодирования, представление букв алфавита, цифр, знаков препинания и других символов последовательностью сигналов: длинных (тире) и коротких (точек). Передача кодов Морзе производится при помощи телеграфного ключа различных конструкций. Компьютерная мышь может быть использована в качестве телеграфного ключа при наличии соответствующего драйвера. Такой режим ввода может быть резервным на случай выхода из строя клавиатуры.

Данная работа посвящена разработке ПО для ввода символов с помощью компьютерной мыши.

1. Аналитический раздел

1.1 Постановка задачи

В соответствии с заданием на курсовую работу необходимо разработать ПО, позволяющее вводить символы с помощью компьютерной мыши.

Для решения поставленной задачи необходимо:

- проанализировать структуру драйвера мыши;
- проанализировать способы измерения времени в ядре;
- проанализировать способы установки задержки в ядре;
- проанализировать способы эмуляции нажатия клавиш клавиатуры;
- спроектировать и реализовать драйвер.

Разрабатываемое ПО должно обрабатывать нажатия только левой клавиши мыши.

1.2 Азбука Морзе

Азбука Моорзе — способ знакового кодирования, представление букв алфавита, цифр, знаков препинания и других символов последовательностью сигналов: длинных (тире) и коротких (точек). За единицу времени принимается длительность одной точки. Длительность тире равна трём точкам. Пауза между элементами одного знака — одна точка, между знаками в слове — 3 точки, между словами — 7 точек. Назван в честь американского изобретателя и художника Сэмюэля Морзе.

Передача кодов Морзе производится при помощи телеграфного ключа различных конструкций: классического ключа Морзе, электронного ключа, механических полуавтоматов типа «виброплекс», а также при помощи клавиатурных датчиков кода Морзе (например, Р-010, Р-020) и электронных устройств, автоматически формирующих телеграфное сообщение.

Компьютерная мышь может быть использована в качестве телеграфного ключа при наличии соответствующего драйвера.

1.3 USB шина

Universal Serial Bus (USB, Универсальная Последовательная Шина) является соединением между компьютером и несколькими периферийными устройствами. Первоначально она была создана для замены широкого круга медленных и различных шин, параллельной, последовательной и клавиатурного соединений, на один тип шины, чтобы к ней могли подключаться все устройства. USB развилась позже этих медленных соединений и в настоящее время поддерживает практически любой тип устройства, который может быть подключен к ПК.

Топологически, подсистема USB не выходит наружу как шина; скорее, это дерево, построенное из нескольких соединений "точка-точка". Контроллер узла (хост) USB отвечает на запросы каждого USB устройства, если оно имеет какие-либо данные для передачи. Из-за такой топологии USB устройство никогда не может начать передачу данных, не будучи сначала запрошенным хост-контроллером.

Шина очень проста на технологическом уровне, поскольку это реализация с одним мастером, в которой хост-компьютер опрашивает разнообразные периферийные устройства. Спецификации протокола USB определяют набор стандартов, которым может следовать любое устройство определённого типа. Если устройство следует такому стандарту, специальный драйвер для этого устройства не требуется. Эти разные типы называются классами и состоят из таких вещей, как устройства хранения данных, клавиатуры, мыши, джойстики, сетевые устройства и модемы. Другие типы устройств, которые не вписываются в эти классы, требуют от поставщика собственного драйвера, написанного для данного специфического устройства.

1.4 USB ядро

В ядре Linux реализована подсистема, которая называется USB ядром, созданная для поддержки USB устройств и контроллеров шины USB. Драйверы основного ядра обращаются к прикладным интерфейсам USB ядра. В тоже время принято выделять два основных публичных прикладных интерфейса: один — реализует взаимодействие с драйверами общего назначения (символьное устройство), другой — взаимодействие с драйверами, являющимися частью ядра (драйвер хаба). Второй тип драйверов участвует в управлении USB шиной. На рис. 1 представлена описанная выше конфигурация.

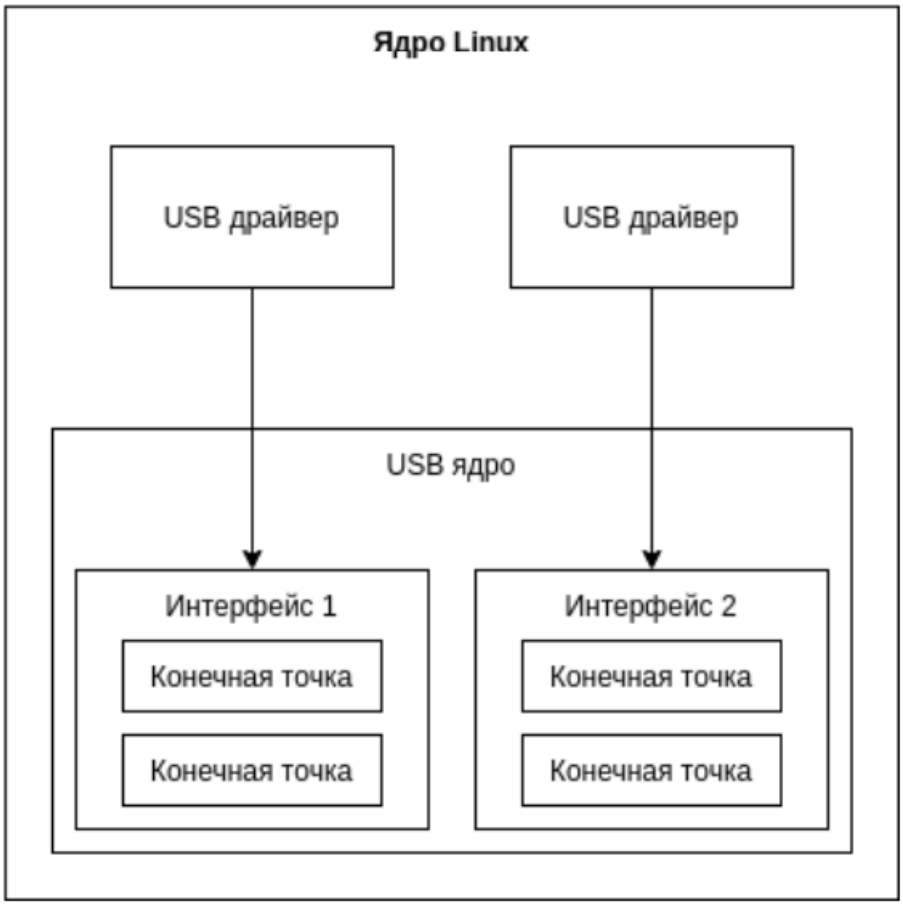

Рис. 1 – USB ядро

1.5 Конечная точка

Как видно из рисунка, интерфейс USB ядра состоит из так называемых конечных точек. Конечная точка является формой USB взаимодействия,

способной переносить данные только в одном направлении (аналогия с однонаправленным каналом). Соответственно, точка IN переносит данные от устройства на хост-систему, OUT – с хост-системы на устройство.

Конечная точка USB может быть одной из четырёх типов, которые описывают, каким образом передаются данные:

- Управляющие конечные точки используются для обеспечения доступа к различным частям устройства USB. Они широко используются для настройки устройства, получения информации об устройстве, посылке команд в устройство, или получения статусных сообщений устройства. Эти оконечные точки, как правило, малы по размеру. Каждое устройство имеет управляющую конечную точку, называемую "endpoint 0", которая используется ядром USB для настройки устройства во время подключения.
- Конечные точки прерывания передают небольшие объёмы данных с фиксированной частотой каждый раз, когда USB хост запрашивает устройство для передачи данных. Эти конечные точки являются основным транспортным методом для USB клавиатур и мышей. Они также часто используются для передачи данных на USB устройства для управления устройством, но обычно не используются для передачи больших объёмов данных.
- Поточные конечные точки передают большие объёмы данных. Они являются обычными для устройств, которые должны передавать любые данные, которые должны пройти через шину, без потери данных. Этим передачам протокол USB не гарантирует выполнения в определённые сроки. Если на шине нет достаточного места, чтобы отправить целый пакет BULK, он распадается на несколько передач в или из устройства. Эти конечные точки общеприняты на принтерах, устройствах хранения и сетевых устройствах.

- Изохронные конечные точки также передают большие объёмы данных, но этим данным не всегда гарантирована доставка. Эти конечные точки используются в устройствах, которые могут обрабатывать потери данных, и больше полагаются на сохранение постоянного потока поступающих данных. При сборе данных в реальном времени, таком, как аудио- и видео-устройства, почти всегда используются такие конечные точки.

Драйвер мыши имеет 1 конечную точку типа прерывания.

1.6 Блоки запроса USB

Для обмена данными между заданной конечной точкой USB и USB устройством в асинхронном режиме используется URB. URB содержит всю необходимую информация для выполнения USB-транзакции и доставки данных и статуса. Каждая конечная точка в устройстве может обрабатывать очередь из URB. Жизненный цикл URB можно разделить на два этапа:

1. Создание драйвером USB.
2. Назначение в определенную конечную точку заданного USB устройства.
3. Передача драйвером USB устройства в USB ядро.
4. Передача USB ядром в заданный драйвер контроллера USB узла для указанного устройства.
5. Обработка драйвером контроллера USB узла, который выполняет передачу по USB в устройство.
6. После завершения работы с URB драйвер контроллера USB узла уведомляет драйвер USB устройства.

Типы URB:

- URB прерывания;

- Поточный URB;
- Управляющий URB;
- Изохронный URB;

1.7 /dev/input/event

Основными компонентами подсистемы ввода-вывода являются драйверы, управляющие внешними устройствами, и файловая система.

Драйвер взаимодействует, с одной стороны, с модулями ядра ОС, а с другой стороны — с контроллерами внешних устройств. Драйверы таких устройств, как мышь, клавиатура, и др. генерируют события, которые в виде структур *input_event* записываются в соответствующий файл в директории “/dev/input”.

Листинг 1 – структура input_event

```
struct input_event {
        struct timeval time; // время возникновения события
        unsigned short type; // тип события (Например: EV_REL, EV_KEY)
        unsigned short code; // код события (Например: REL_X, KEY_BACKSPACE)
        unsigned int value; // статус события (Например для EV_KEY: 0-release, 1-
keypress, 2-autorepeat)
};
```

Для работы с файлами в пространстве ядра нужно использовать специальные функции: *filp_open()*, *filp_close()*, *kernel_read()*, *kernel_write()*.

С помощью /dev/input/event можно регистрировать нажатия клавиши мыши и эмулировать нажатия клавиш клавиатуры. Но в таком случае разрабатываемое ПО может работать в пространстве пользователя, что не подходит для курсовой работы по операционным системам. Поэтому данным способом будет эмулироваться только нажатие клавиш клавиатуры.

1.8 Измерение временных промежутков

Ядро следит за течением времени с помощью таймера прерываний. Прерывания таймера генерируются через постоянные интервалы системным аппаратным таймером; этот интервал программируется во время загрузки ядром в соответствии со значением *HZ*, которое является архитектурно-зависимой величиной, определённой в или файле подплатформы, подключаемом им. Оно определяет количество тиков в секунду.

Значение внутреннего счётчика ядра увеличивается каждый раз, когда происходит прерывание от таймера. Счётчик инициализируется 0 при загрузке системы, поэтому он представляет собой число тиков системных часов после последней загрузки. Счётчик является 64-х разрядной переменной (даже на 32-х разрядных архитектурах) и называется *jiffies_64*. Однако авторы драйверов обычно используют переменную *jiffies* типа *unsigned long*, которая то же самое, что и *jiffies_64* или её младшие биты.

Всякий раз, когда код должен запомнить текущее значение *jiffies*, он может просто обратится к переменной *unsigned long*, которая объявлена как volatile (нестабильная), чтобы компилятор не оптимизировал чтения памяти. Используя HZ, можно перевести промежуток времени в тиках в секунды.

1.8 Таймеры ядра

Коды Морзе разных символов имеют разную длину: от 1 (например “t” – “-”) до 5 (например “3” – “…--”). Причём границы символов в потоке “точек” и “тире” однозначно установить нельзя. Например “e” – “.”, а “s” – “…”. В таком случае возникает проблема отделения одного символа от другого при вводе. Если введена последовательность максимальной длины – 5, то её можно сразу проверять на соответствие какому-либо символу. В противном случае, нужно ставить выполнение этой проверки на задержку, например в 1 секунду. Если 1 секунду не поступало новых “точек” и “тире”, то драйвер проверяет последовательность уже введённых. Если же поступил новый сигнал, то

выставленная задержка должна обновиться, 1 секунда должна заново начать отсчитываться.

Инструментом для выполнения действия с фиксированной задержкой без блокирования текущего процесса на время ожидания являются таймеры ядра. Такие таймеры используются для планирования выполнения функции в определённое время в будущем, основываясь на тактовых тиках. Таймер ядра описывается структурой *timer_list*, которая информирует ядро о выполнении заданной пользователем функции с заданным пользователем аргументом в заданное пользователем время.

Листинг 2 – структура timer_list

```
struct timer_list {
    /* ... */
    unsigned long expires;
    void (*function)(unsigned long);
    unsigned long data;
};
```

Структура данных включает в себя больше полей, чем показано, но эти три являются основными с точки зрения использования:

- *expires* – количество тиков в *jiffies*, при котором будет запущена функция таймера. Т.е. если нужно, чтобы функция запустилась, условно, через 500 тиков, то *expires* нужно присвоить значение *jiffies + 500*;
- *function* – функция таймера, которая будет вызвана по истечении задержки;
- *data* – единственный аргумент, передаваемый при вызове функции *function*. Если необходимо передать много объектов в аргументе, можно собрать их в единую структуру данных и передать указатель, приведя к *unsigned long*, это безопасная практика на всех

поддерживаемых архитектурах и довольно распространена в управлении памятью.

1.10 Выводы

В результате проведённого анализа было решено использовать:

- структуру *input_event* для эмуляции нажатия клавиш клавиатуры;
- структуру *timer_list* для установки задержки выполнения;
- переменную *jiffies* и макрос *HZ* для измерения временных промежутков.

2. Конструкторский раздел

2.1 IDEF0

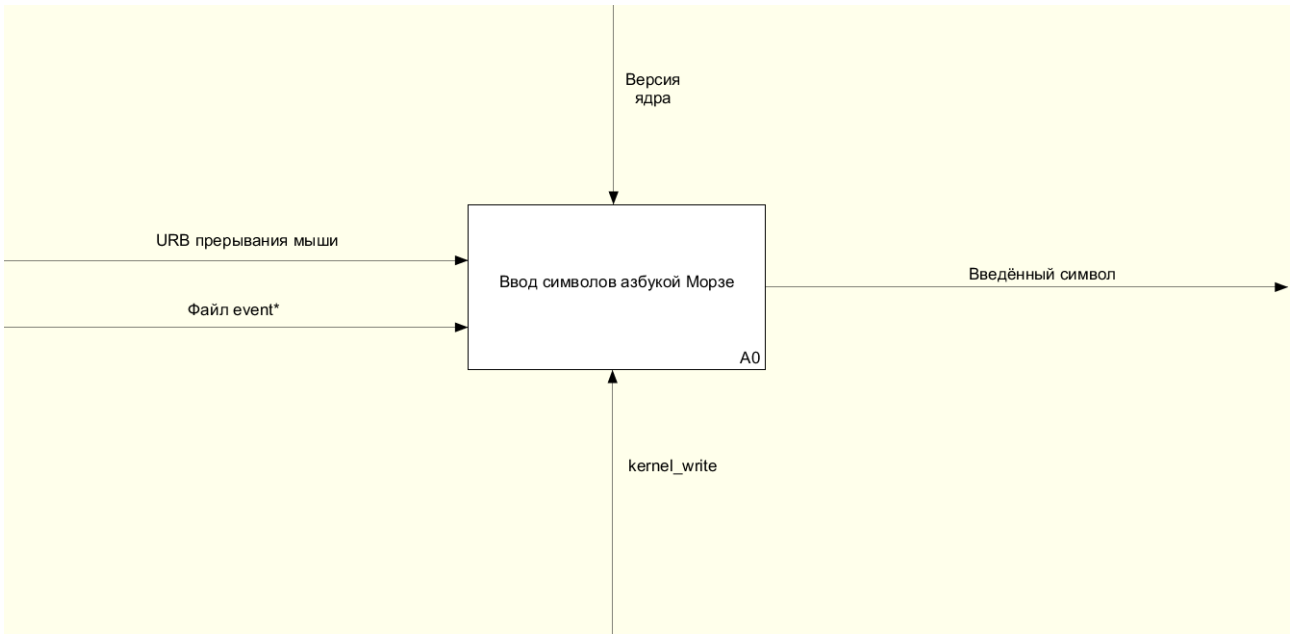

Рис. 2 – IDEF0 нулевого уровня

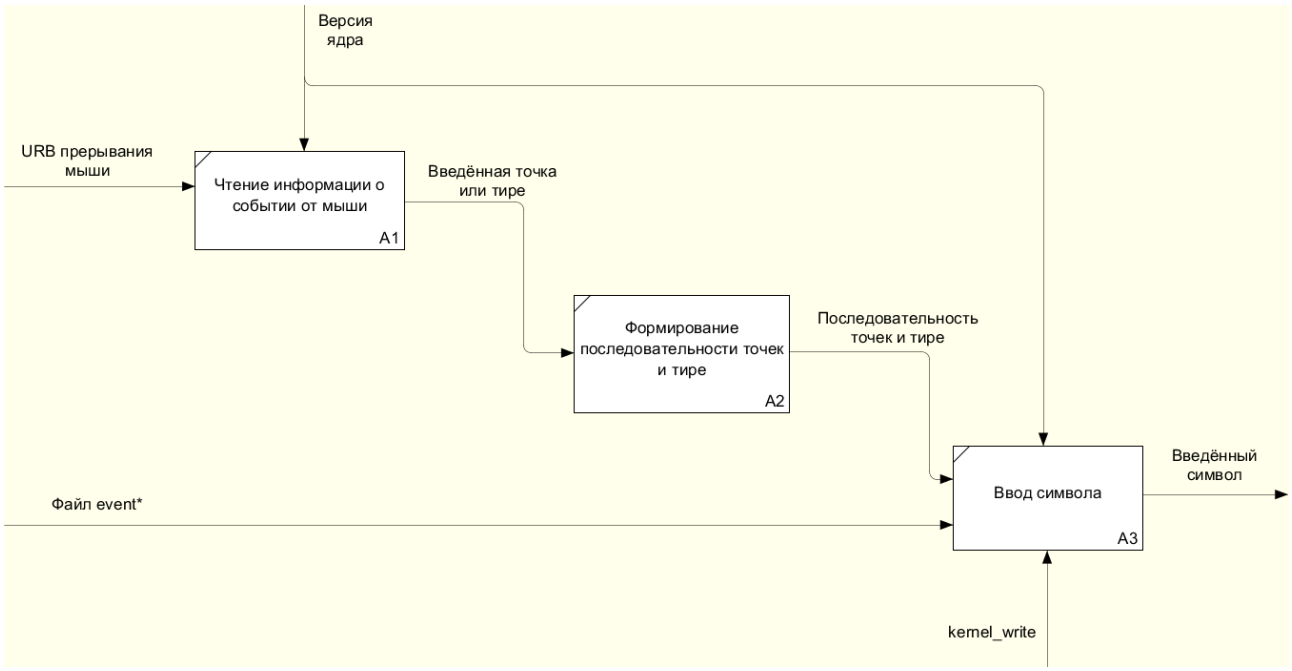

Рис. 3 – IDEF0 первого уровня

2.2 Точки входа драйвера

Прерывания от мыши поступают при нажатии и отпускании кнопок, движении мыши, вращении колёсика. Из всех них нужно выделить только

нажатие и отпускание левой кнопки мыши. Алгоритм, выполняющий это, изображен на Рис. 4.

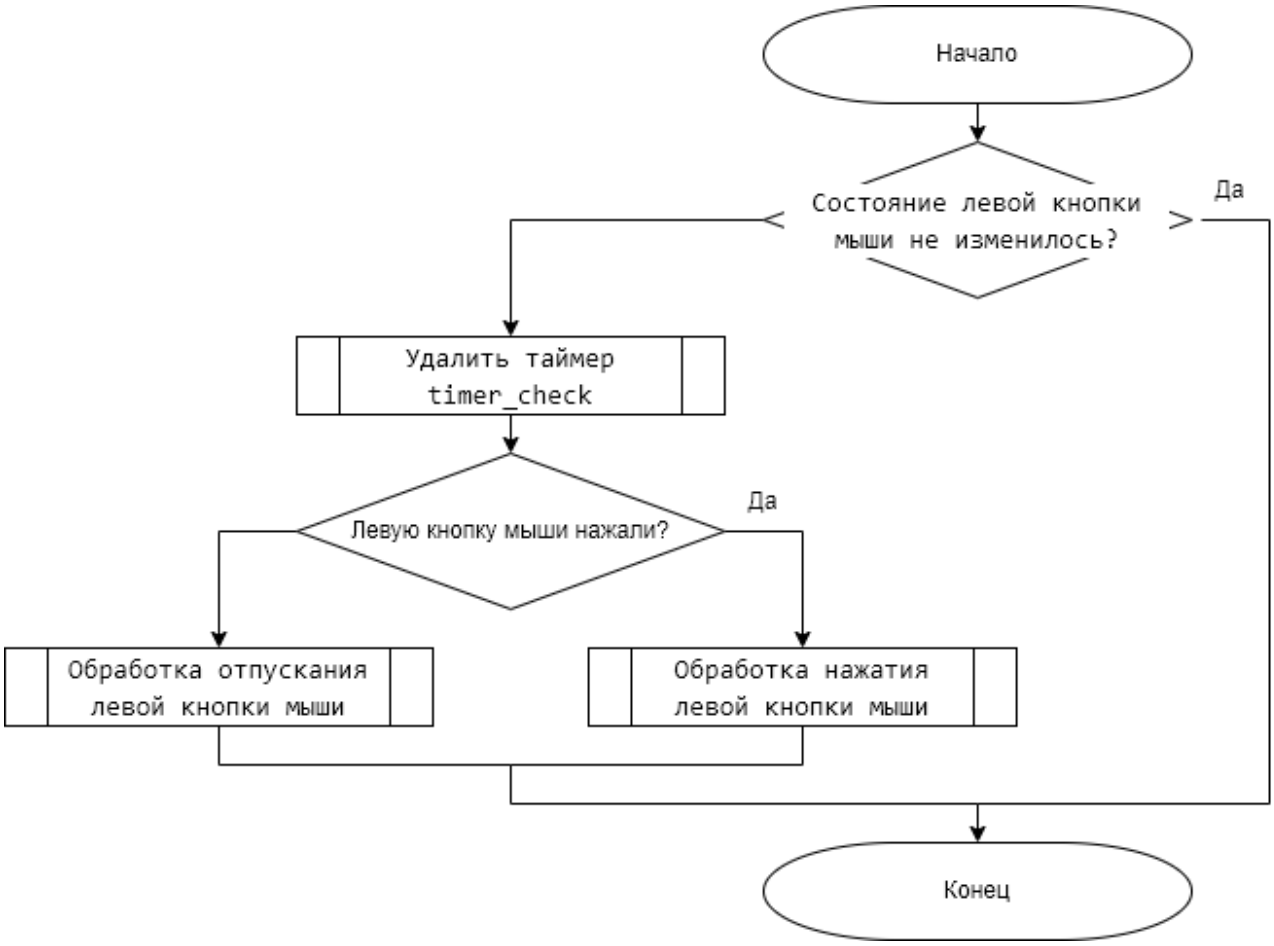

Рис. 4 – алгоритм обработки состояния левой кнопки мыши

Обработчик нажатий может находиться в двух состояниях: ввода и ожидания. В состояние ввода он переводится при первом нажатии левой кнопки мыши. В состояние ожидания он переводится при вводе последовательности точек и тире максимальной длины, или по истечении таймера *timer_check*. Таймер ставится заново при каждом изменении состояния левой кнопки мыши, когда обработчик нажатий в режиме ввода. Это отражено на Рис. 5 - 7.

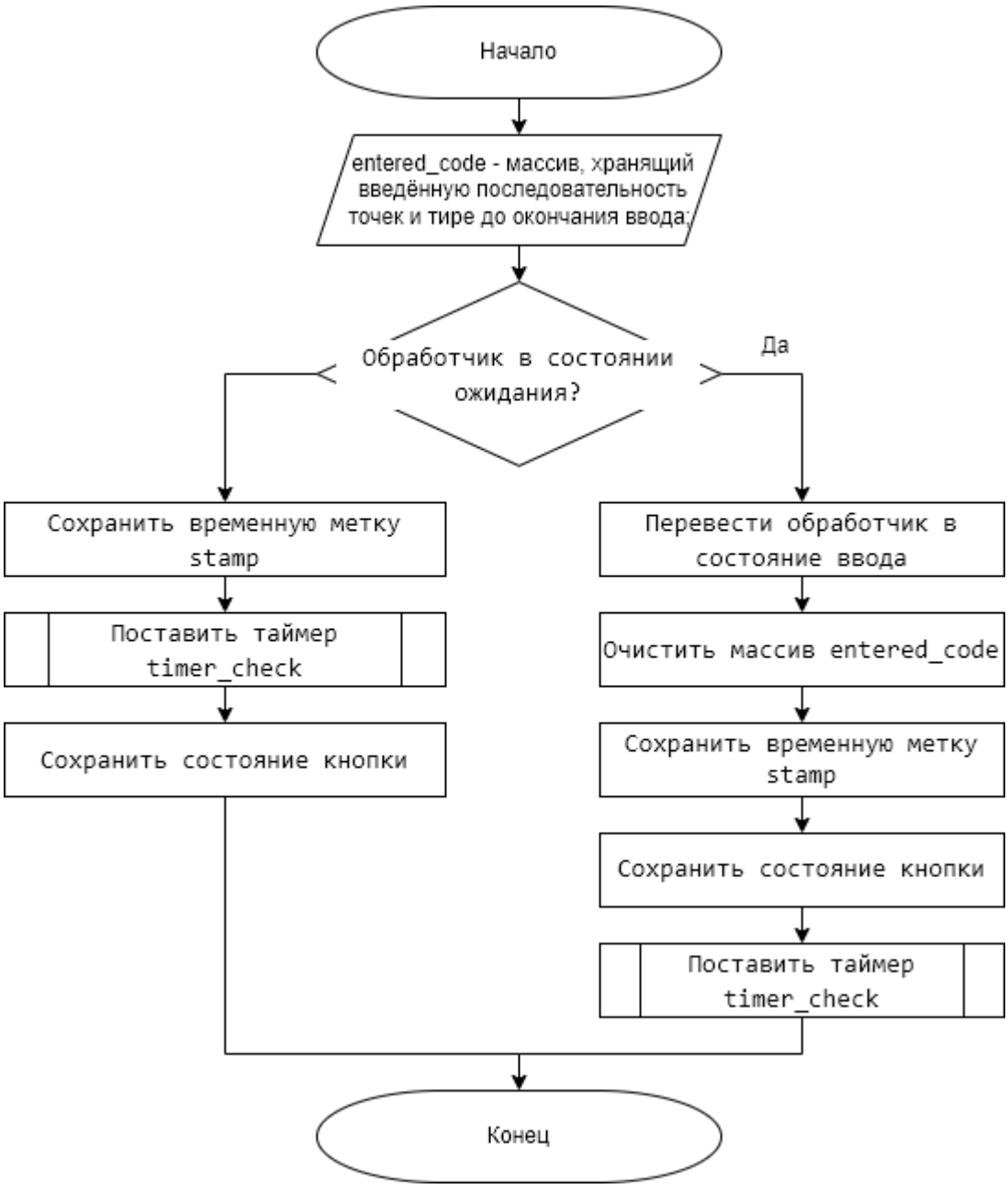

Рис. 5 – алгоритм обработки нажатия левой кнопки мыши

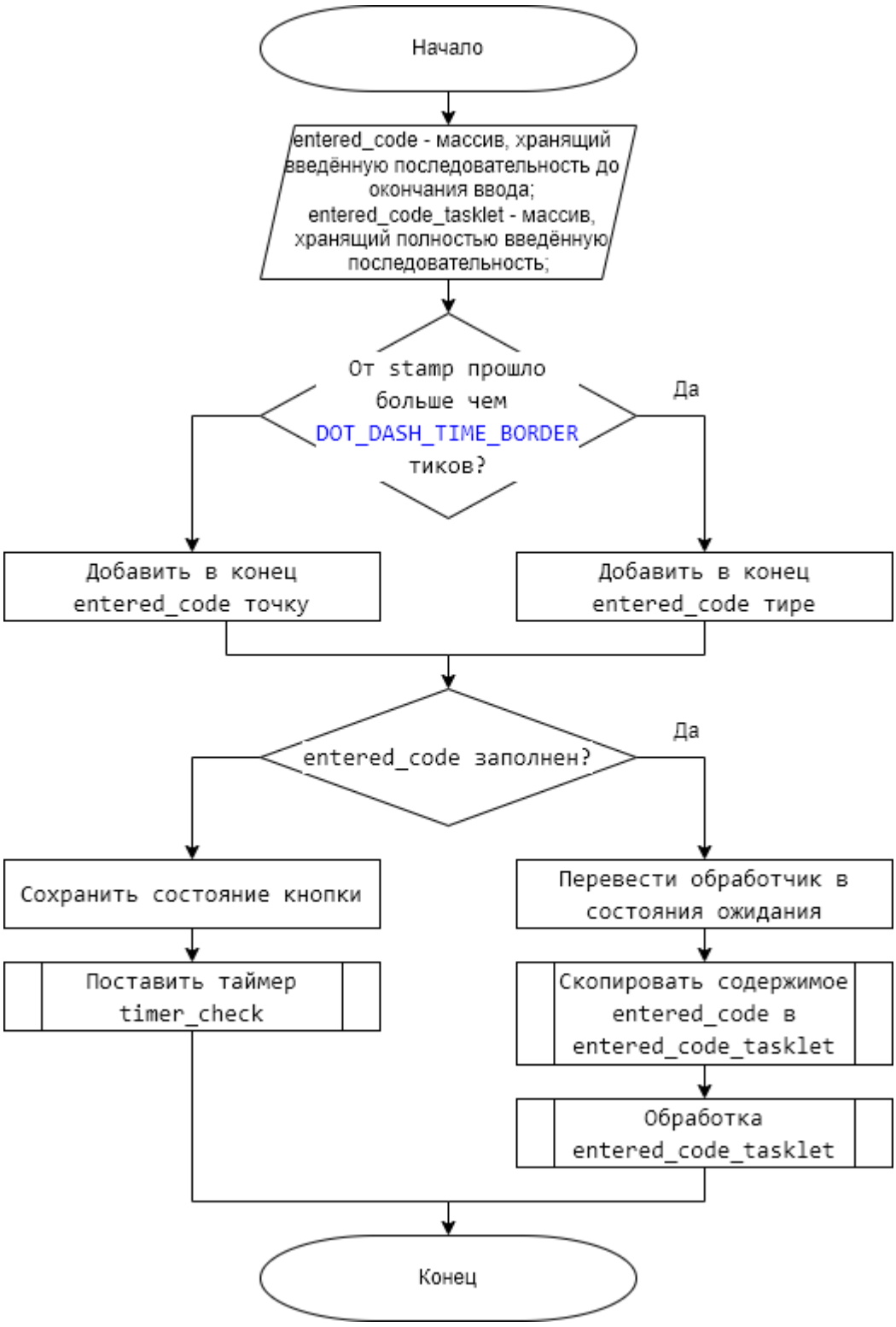

Рис. 6 – алгоритм отпускания левой кнопки мыши

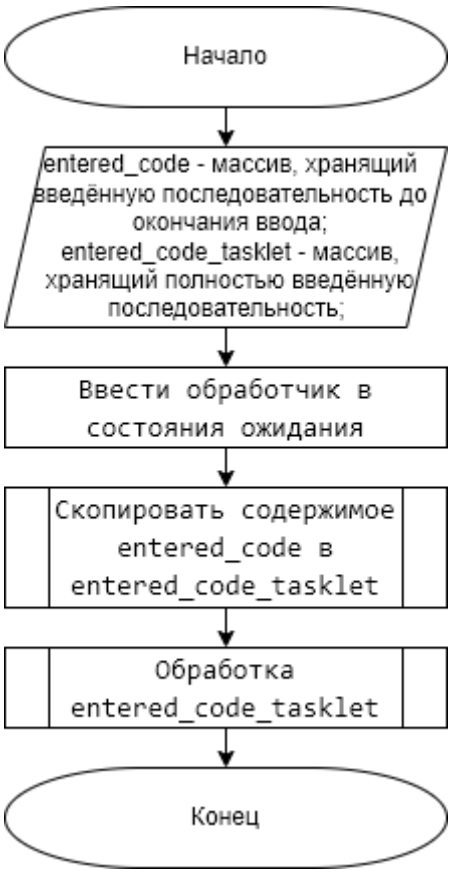

Рис. 7 – алгоритм таймера *timer_check*

При завершении ввода введённая последовательность точек и тире сохраняется в массиве *entered_code_tasklet*. Его обработка отражена на Рис. 8.

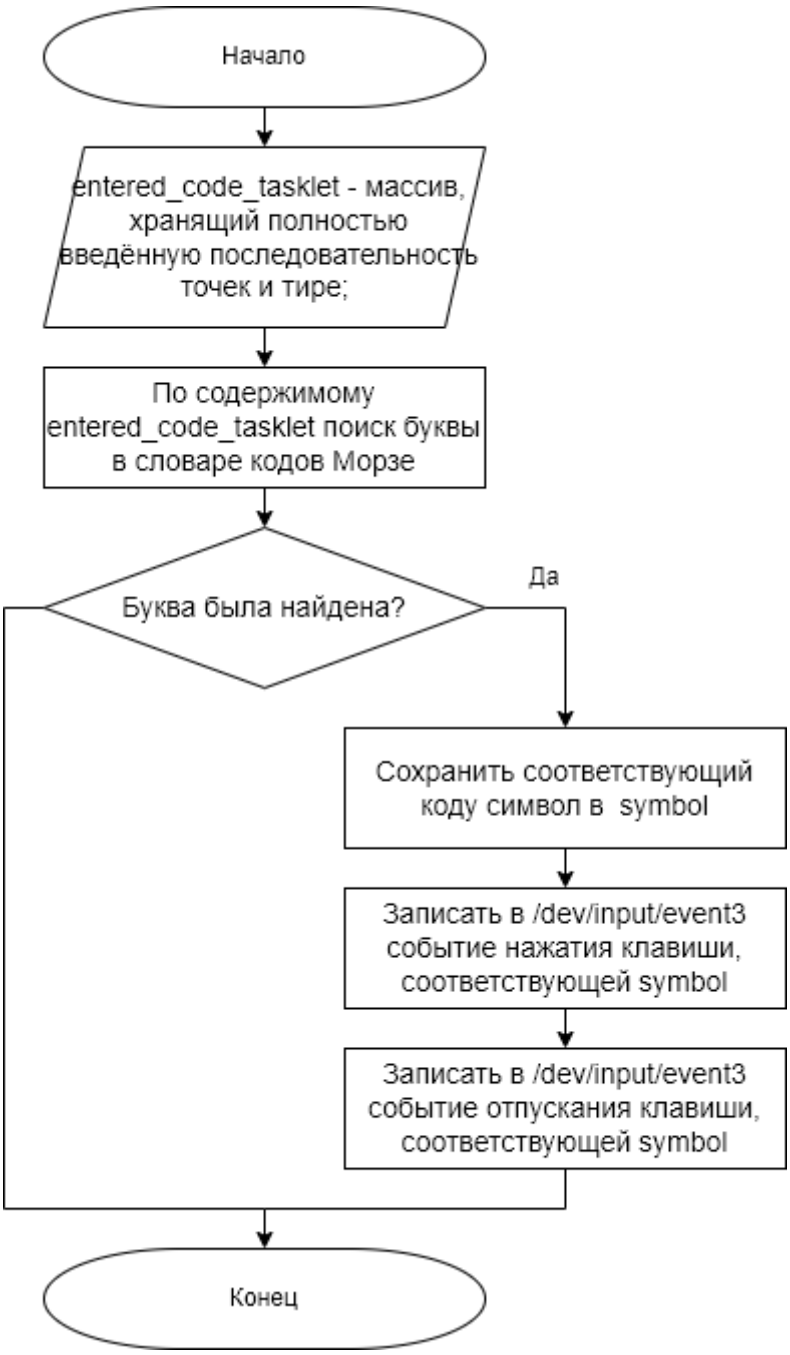

Рис. 8 – алгоритм обработки entered_code_tasklet

3. Технологический раздел

3.1 Выбор языка программирования и среды разработки

В качестве языка программирования был выбран C.

В качестве среды разработки была выбрана «Code::Blocks» т.к. она бесплатная и кроссплатформенная.

3.2 Модуль morze_enter

Функция *my_mouse_handler* при запуске проверяет изменение состояния левой кнопки мыши, т.к. запускается она при каждом прерывании от мыши, в том числе при её движении. Если состояние изменилось, т.е. кнопку нажали или отпустили, то таймер, если он был поставлен, удаляется.

Для хранения введённого кода выделяется массив символов *entered_code*. Функция *my_mouse_handler* после отпускания клавиши мыши записывает в него точку или тире и, если *entered_code* ещё не заполнен, т.е. не введён код максимально возможной длины, ставит таймер. Если же *entered_code* заполнен, то его содержимое копируется в дополнительный массив *entered_code_tasklet* и ставится на выполнение тасклет.

Если клавиша находилась в одном состоянии достаточно долго (1 секунду), то успевает начать выполнение функция-обработчик таймера – *timer_check*. Она производит те же самые действия, что и *my_mouse_handler* при заполнении *entered_code*. Т.е. если нужно ввести код не максимальной длины, пользователь должен подождать 1 секунду.

В функции тасклета *my_tasklet_function* содержимое *entered_code_tasklet* сверяется со словарём кодов Морзе и, если совпадение найдено, эмулируется нажатие соответствующей клавиши – в /dev/input/event записывается набор структур, сообщающий о её нажатии и отпускании. В данном случае *entered_code_tasklet* является разделяемой переменной. В неё пишут *timer_check* и *my_mouse_handler*, а читает функция тасклета. Но между операциями записи в неё проходит не меньше 1 секунды; вероятность, что функция тасклета не

успеет прочесть её за это время, пренебрежимо мала. Поэтому, использовать средства синхронизации для её чтения, смысла нет.

Листинг 3 – функция *my_mouse_handler*

```
extern void my_mouse_handler(char new_mouse_status)
{
    if (last_mouse_status == MOUSE_WAIT && new_mouse_status == MOUSE_RELEASE ||
        last_mouse_status == new_mouse_status ||
        new_mouse_status != MOUSE_RELEASE && new_mouse_status != MOUSE_PRESS)
        return;

    del_timer_sync(&my_timer);
    if (last_mouse_status == MOUSE_WAIT && new_mouse_status == MOUSE_PRESS)
    {
        entered_code_len = 0;
        memset(entered_code, '\0', MAX_MORZE_CODE_LEN);
        stamp = jiffies;
        last_mouse_status = MOUSE_PRESS;
        mod_timer(&my_timer, jiffies + WAIT_TIME);
    }
    else if (last_mouse_status == MOUSE_PRESS && new_mouse_status == MOUSE_RELEASE)
    {
        if ((long)jiffies - (long)stamp < DOT_DASH_TIME_BORDER)
            entered_code[entered_code_len] = '.';
        else
            entered_code[entered_code_len] = '-';
        entered_code_len++;
        if (entered_code_len == MAX_MORZE_CODE_LEN)
        {
            last_mouse_status = MOUSE_WAIT;
            strcpy(entered_code_tasklet, entered_code);
            tasklet_schedule(&my_tasklet);
        }
        else
        {
            last_mouse_status = MOUSE_RELEASE;
            mod_timer(&my_timer, jiffies + WAIT_TIME);
        }
    }
    else
    {
        stamp = jiffies;
        mod_timer(&my_timer, jiffies + WAIT_TIME);
        last_mouse_status = MOUSE_PRESS;
    }
}
```

Листинг 4 – функция *timer_check*

```
void timer_check(struct timer_list *timer)
{
    last_mouse_status = MOUSE_WAIT;
    strcpy(entered_code_tasklet, entered_code);
    tasklet_schedule(&my_tasklet);
}
```

Листинг 5 – инициализация модуля morze_enter

```
void my_tasklet_function(unsigned long data)
{
    int i = 0;
    for (; i < MORZE_DICT_LEN && strcmp(morze_codes[i], entered_code_tasklet);
i++);
    if (i < MORZE_DICT_LEN)
    {
        printk(KERN_INFO "[morze] entered code: %s; entered symbol: %c\n",
entered_code_tasklet, morze_symbols[i]);

        event_kbd.type = EV_KEY;
        event_kbd.code = morze_keys[i];
        event_kbd.value = 1;
        kernel_write(f_input_event, (char*)&event_kbd, sizeof(event_kbd),
&f_input_event->f_pos);
        event_kbd.type = EV_KEY;
        event_kbd.code = morze_keys[i];;
        event_kbd.value = 0;
        kernel_write(f_input_event, (char*)&event_kbd, sizeof(event_kbd),
&f_input_event->f_pos);
        event_kbd.type = EV_SYN;
        event_kbd.code = 0;
        event_kbd.value = 0;
        kernel_write(f_input_event, (char*)&event_kbd, sizeof(event_kbd),
&f_input_event->f_pos);
    }
}
```

3.3 Makefile

В листинге 3.3.1 приведено содержимое Makefile, содержащего набор инструкций, используемых утилитой make в инструментарии автоматизации сборки.

Листинг 6 – Makefile

```
KBUILD_EXTRA_SYMBOLS = $(shell pwd)/Module.symverscd
ifneq ($(KERNELRELEASE),)
    obj-m := driver.o morze_enter.o
else
    CURRENT = $(shell uname -r)
    KDIR = /lib/modules/$(CURRENT)/build
    PWD = $(shell pwd)

default:
    $(MAKE) -C $(KDIR) M=$(PWD) modules
    make cleanHalf

cleanHalf:
    rm -rf *.o *~ *.mod *.mod.c Module.* *.order  .tmp_versions

clean:
    make cleanHalf
    rm -rf *.ko

endif
```

4. Исследовательский раздел

ПО протестировано на ядре Linux v5.4.50.

На рис. 9 продемонстрирована последовательность загрузки драйвера мыши. На рис. 10 продемонстрирована последовательность выгрузки драйвера мыши. На рис 11 - 12 продемонстрирован ввод символов азбукой Морзе.

```
egor@egor-Lenovo-IdeaPad-L340-15API:~/CourseProjectOS/code$ sudo insmod morze_enter.ko
egor@egor-Lenovo-IdeaPad-L340-15API:~/CourseProjectOS/code$ sudo rmmod usbhid
egor@egor-Lenovo-IdeaPad-L340-15API:~/CourseProjectOS/code$ sudo insmod driver.ko
```

Рис. 9 – загрузка драйвера

```
egor@egor-Lenovo-IdeaPad-L340-15API:~/CourseProjectOS/code$ sudo rmmod driver
egor@egor-Lenovo-IdeaPad-L340-15API:~/CourseProjectOS/code$ sudo modprobe usbhid
egor@egor-Lenovo-IdeaPad-L340-15API:~/CourseProjectOS/code$ sudo rmmod morze_enter
```

Рис 10 – выгрузка драйвера

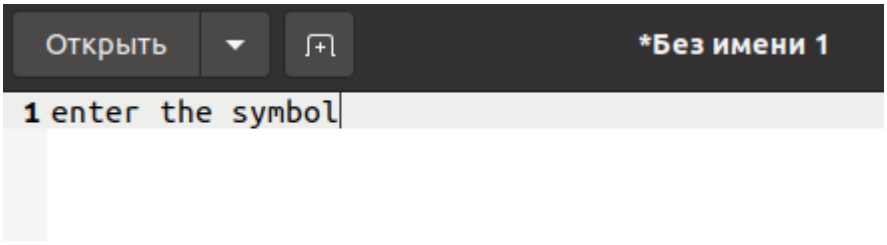

Рис 11 – введённые символы

```
[ 6346.896311] [morze] entered code: .; entered symbol: e
[ 6357.680518] [morze] entered code: -.; entered symbol: n
[ 6360.624361] [morze] entered code: -; entered symbol: t
[ 6362.704445] [morze] entered code: .; entered symbol: e
[ 6374.224706] [morze] entered code: .-.; entered symbol: r
[ 6380.505918] [morze] entered code: ..-..; entered symbol:
[ 6387.824362] [morze] entered code: -; entered symbol: t
[ 6392.976843] [morze] entered code: ....; entered symbol: h
[ 6395.632342] [morze] entered code: .; entered symbol: e
[ 6401.698919] [morze] entered code: ..-..; entered symbol:
[ 6408.432333] [morze] entered code: ...; entered symbol: s
[ 6415.792771] [morze] entered code: -.--; entered symbol: y
[ 6422.160309] [morze] entered code: --; entered symbol: m
[ 6435.856361] [morze] entered code: -...; entered symbol: b
[ 6442.768378] [morze] entered code: ---; entered symbol: o
[ 6448.948319] [morze] entered code: .-..; entered symbol: l
```

Рис 12 – сообщения ядра при вводе

Заключение

В процессе выполнения курсовой работы были:

- проанализирована структура драйвера мыши;
- проанализированы способы измерения времени в ядре;
- проанализированы способы установки задержки в ядре;
- проанализированы способы эмуляции нажатия клавиатуры;
- спроектирован и реализован драйвер.

В ходе выполнения поставленных задач были изучены возможности языка C, получены знания в области написания загружаемых модулей ядра, драйверов.

Список использованной литературы

1. Exploring /dev/input. Keerthi Vasan G.C, Suresh. B, 21.04.2017. The hacker Diary. [Электронный ресурс]. – Режим доступа: https://thehackerdiary.wordpress.com/2017/04/21/exploring-devinput-1/
2. Linux Input drivers v1.0. Vojtech Pavlik. [Электронный ресурс]. – Режим доступа: https://www.kernel.org/doc/Documentation/input/input.txt
3. А.Н. Васюнин, Н.Ю. Рязанова, канд. техн. наук, доц., Е.В. Тарасенко, С.В. Тарасенко. Анализ методов изменения функциональности внешних устройств в ОС Linux. В сб. «Автоматизация. Современные технологии», 2016 №10. С. 3-8.
4. Исходный код файла usb.h [Электронный ресурс]. – Режим доступа: https://github.com/torvalds/linux/blob/master/include/linux/usb.h
5. Corbet J., Rubini A., Kroan-Hartman G. Linux device drivers. O’Reilly Media, 2005.

Приложение А
Листинги модуля

Листинг А.1 – morze_enter.h

```
#define MAX_MORZE_CODE_LEN 5
#define MORZE_DICT_LEN 37
static const char morze_codes[MORZE_DICT_LEN][MAX_MORZE_CODE_LEN + 1] = {
    ".-", "-...", "-.-.", "-..", ".", "..-.",
    "--.", "....", "..", ".---", "-.-", ".-..",
    "--", "-.", "---", ".--.", "--.-", ".-.",
    "...", "-", "..-", "...-", ".--", "-..-",
    "-.--", "--..", ".----", "..---", "...--", "....-",
    ".....", "-....", "--...", "---..", "----.", "-----", "..-.."
};

static const char morze_keys[MORZE_DICT_LEN] = {
    KEY_A, KEY_B, KEY_C, KEY_D, KEY_E, KEY_F,
    KEY_G, KEY_H, KEY_I, KEY_J, KEY_K, KEY_L,
    KEY_M, KEY_N, KEY_O, KEY_P, KEY_Q, KEY_R,
    KEY_S, KEY_T, KEY_U, KEY_V, KEY_W, KEY_X,
    KEY_Y, KEY_Z, KEY_1, KEY_2, KEY_3, KEY_4,
    KEY_5, KEY_6, KEY_7, KEY_8, KEY_9, KEY_0, KEY_SPACE
};

static const char morze_symbols[MORZE_DICT_LEN] = {
    'a', 'b', 'c', 'd', 'e', 'f',
    'g', 'h', 'i', 'j', 'k', 'l',
    'm', 'n', 'o', 'p', 'q', 'r',
    's', 't', 'u', 'v', 'w', 'x',
    'y', 'z', '1', '2', '3', '4',
    '5', '6', '7', '8', '9', '0', ' '
};
```

Листинг А.2 – morze_enter.c

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/jiffies.h>
#include <linux/timer.h>
#include <linux/interrupt.h>
#include <linux/spinlock.h>
#include <linux/input.h>
#include "morze_enter.h"

#define MOUSE_WAIT -1
#define MOUSE_PRESS 1
#define MOUSE_RELEASE 0
```

```
#define DOT_DASH_TIME_BORDER HZ / 3
#define WAIT_TIME HZ

static char last_mouse_status = MOUSE_WAIT;
static unsigned long stamp = 0;
static char entered_code[MAX_MORZE_CODE_LEN + 1] = "\0\0\0\0\0\0";
static int entered_code_len = 0;
static char entered_code_tasklet[MAX_MORZE_CODE_LEN + 1];

static struct tasklet_struct my_tasklet;
static struct timer_list my_timer;
DEFINE_SPINLOCK(my_lock);

static struct file *f_input_event;
struct input_event event_kbd;

void my_tasklet_function(unsigned long data);
void timer_check(struct timer_list *timer);


MODULE_LICENSE("GPL");
MODULE_AUTHOR("Grishin E.B");
MODULE_DESCRIPTION("Entering by mouse morze");


static int __init morze_init( void )
{
    printk(KERN_INFO "[morze] in init\n");
    tasklet_init(&my_tasklet, my_tasklet_function, 0);
    timer_setup(&my_timer, timer_check, 0);
    f_input_event = filp_open("/dev/input/event3", O_WRONLY, 0);

    return 0;
}

static void __exit morze_exit( void )
{
    printk(KERN_INFO "[morze] in exit\n");
    tasklet_kill(&my_tasklet);
    del_timer(&my_timer);
    filp_close(f_input_event, current->files);
}


void my_tasklet_function(unsigned long data)
{
    int i = 0;
    for (; i < MORZE_DICT_LEN && strcmp(morze_codes[i], entered_code_tasklet);
i++);
    if (i < MORZE_DICT_LEN)
```

```
    {
        printk(KERN_INFO "[morze] entered code: %s; entered symbol: %c\n",
entered_code_tasklet, morze_symbols[i]);

        event_kbd.type = EV_KEY;
        event_kbd.code = morze_keys[i];
        event_kbd.value = 1;
        kernel_write(f_input_event, (char*)&event_kbd, sizeof(event_kbd),
&f_input_event->f_pos);
        event_kbd.type = EV_KEY;
        event_kbd.code = morze_keys[i];;
        event_kbd.value = 0;
        kernel_write(f_input_event, (char*)&event_kbd, sizeof(event_kbd),
&f_input_event->f_pos);
        event_kbd.type = EV_SYN;
        event_kbd.code = 0;
        event_kbd.value = 0;
        kernel_write(f_input_event, (char*)&event_kbd, sizeof(event_kbd),
&f_input_event->f_pos);
    }
}

void timer_check(struct timer_list *timer)
{
    spin_lock(&my_lock);
    if (last_mouse_status != MOUSE_WAIT)
    {
        last_mouse_status = MOUSE_WAIT;
        strcpy(entered_code_tasklet, entered_code);
        tasklet_schedule(&my_tasklet);
        //printk(KERN_INFO "[morze] sended tasklet from timer; code: %s\n",
entered_code_tasklet);
    }
    spin_unlock(&my_lock);
}

extern void my_mouse_handler(char new_mouse_status)
{
    if (last_mouse_status == MOUSE_WAIT && new_mouse_status == MOUSE_RELEASE ||
        last_mouse_status == new_mouse_status ||
        new_mouse_status != MOUSE_RELEASE && new_mouse_status != MOUSE_PRESS)
        return;

    spin_lock(&my_lock);
    del_timer(&my_timer);
    //printk(KERN_INFO "[morze] mouse handler. last: %d; new: %d\n",
last_mouse_status, new_mouse_status);
    if (last_mouse_status == MOUSE_WAIT && new_mouse_status == MOUSE_PRESS)
    {
        entered_code_len = 0;
```

```
        memset(entered_code, '\0', MAX_MORZE_CODE_LEN);
        stamp = jiffies;
        last_mouse_status = MOUSE_PRESS;
        mod_timer(&my_timer, jiffies + WAIT_TIME);
    }
    else if (last_mouse_status == MOUSE_PRESS && new_mouse_status == MOUSE_RELEASE)
    {
        if ((long)jiffies - (long)stamp < DOT_DASH_TIME_BORDER)
        {
            entered_code[entered_code_len] = '.';
            //printk(KERN_INFO "[morze] entered code: .\n");
        }
        else
        {
            entered_code[entered_code_len] = '-';
            //printk(KERN_INFO "[morze] entered code: -\n");
        }
        //printk(KERN_INFO "[morze] time interval: %ld\n", jiffies - stamp);
        entered_code_len++;
        if (entered_code_len == MAX_MORZE_CODE_LEN)
        {
            last_mouse_status = MOUSE_WAIT;
            strcpy(entered_code_tasklet, entered_code);
            tasklet_schedule(&my_tasklet);
            //printk(KERN_INFO "[morze] sended tasklet from handler; code: %s\n",
entered_code_tasklet);
        }
        else
        {
            last_mouse_status = MOUSE_RELEASE;
            mod_timer(&my_timer, jiffies + WAIT_TIME);
        }
    }
    else // if (last_mouse_status == MOUSE_RELEASE && new_mouse_status ==
MOUSE_PRESS)
    {
        stamp = jiffies;
        mod_timer(&my_timer, jiffies + WAIT_TIME);
        last_mouse_status = MOUSE_PRESS;
    }
    spin_unlock(&my_lock);
}


EXPORT_SYMBOL(my_mouse_handler);

module_init(morze_init);
module_exit(morze_exit);
```

Листинг А.3 – driver.c

```
#include <linux/kernel.h>
#include <linux/slab.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/usb/input.h>
#include <linux/hid.h>

/*
 * Version Information
 */
#define DRIVER_VERSION "v1.6"
#define DRIVER_AUTHOR "Grishin E.B."
#define DRIVER_DESC "USB HID Boot Protocol mouse driver"

extern void my_mouse_handler(char new_mouse_status);

MODULE_AUTHOR(DRIVER_AUTHOR);
MODULE_DESCRIPTION(DRIVER_DESC);
MODULE_LICENSE("GPL");

struct usb_mouse {
    char name[128];
    char phys[64];
    struct usb_device *usbdev;
    struct input_dev *dev;
    struct urb *irq;

    signed char *data;
    dma_addr_t data_dma;
};

static void usb_mouse_irq(struct urb *urb)
{
    struct usb_mouse *mouse = urb->context;
    signed char *data = mouse->data;
    struct input_dev *dev = mouse->dev;
    int status;

    my_mouse_handler(data[0]  & 0x01);

    char resubmit = 1;
    switch (urb->status) {
    case 0:         /* success */
        break;
    case -ECONNRESET:   /* unlink */
    case -ENOENT:
    case -ESHUTDOWN:
        return;
```

```
    /* -EPIPE:  should clear the halt */
    default:        /* error */
        resubmit = 0;
    }

    if (resubmit)
    {
        input_report_key(dev, BTN_LEFT,   data[0] & 0x01);
        input_report_key(dev, BTN_RIGHT,  data[0] & 0x02);
        input_report_key(dev, BTN_MIDDLE, data[0] & 0x04);
        input_report_key(dev, BTN_SIDE,   data[0] & 0x08);
        input_report_key(dev, BTN_EXTRA,  data[0] & 0x10);

        input_report_rel(dev, REL_X,     data[1]);
        input_report_rel(dev, REL_Y,     data[2]);
        input_report_rel(dev, REL_WHEEL, data[3]);

        input_sync(dev);
    }
resubmit:
    status = usb_submit_urb (urb, GFP_ATOMIC);
    if (status)
        dev_err(&mouse->usbdev->dev,
            "can't resubmit intr, %s-%s/input0, status %d\n",
            mouse->usbdev->bus->bus_name,
            mouse->usbdev->devpath, status);
}

static int usb_mouse_open(struct input_dev *dev)
{
    struct usb_mouse *mouse = input_get_drvdata(dev);
    printk(KERN_INFO "[usbmouse] in usb_mouse_open\n");

    mouse->irq->dev = mouse->usbdev;
    if (usb_submit_urb(mouse->irq, GFP_KERNEL))
        return -EIO;

    return 0;
}

static void usb_mouse_close(struct input_dev *dev)
{
    struct usb_mouse *mouse = input_get_drvdata(dev);
    printk(KERN_INFO "[usbmouse] in usb_mouse_close\n");

    usb_kill_urb(mouse->irq);
}

static int usb_mouse_probe(struct usb_interface *intf, const struct usb_device_id
*id)
```

```
{
    printk(KERN_INFO "[usbmouse] in usb_mouse_probe\n");
    struct usb_device *dev = interface_to_usbdev(intf);
    struct usb_host_interface *interface;
    struct usb_endpoint_descriptor *endpoint;
    struct usb_mouse *mouse;
    struct input_dev *input_dev;
    int pipe, maxp;
    int error = -ENOMEM;

    char fail1 = 1, fail2 = 1, fail3 = 1;

    interface = intf->cur_altsetting;

    if (interface->desc.bNumEndpoints != 1)
        return -ENODEV;

    endpoint = &interface->endpoint[0].desc;
    if (!usb_endpoint_is_int_in(endpoint))
        return -ENODEV;

    pipe = usb_rcvintpipe(dev, endpoint->bEndpointAddress);
    maxp = usb_maxpacket(dev, pipe, usb_pipeout(pipe));

    mouse = kzalloc(sizeof(struct usb_mouse), GFP_KERNEL);
    input_dev = input_allocate_device();
    if (!mouse || !input_dev)
        fail1 = 0;

    if (fail1)
    {
        mouse->data = usb_alloc_coherent(dev, 8, GFP_ATOMIC, &mouse->data_dma);
        if (!mouse->data)
            fail1 = 0;

        if (fail1)
        {
            mouse->irq = usb_alloc_urb(0, GFP_KERNEL);
            if (!mouse->irq)
                fail2 = 0;

            if (fail2)
            {
                mouse->usbdev = dev;
                mouse->dev = input_dev;

                if (dev->manufacturer)
                    strlcpy(mouse->name, dev->manufacturer, sizeof(mouse->name));

                if (dev->product) {
```

```
                if (dev->manufacturer)
                    strlcat(mouse->name, " ", sizeof(mouse->name));
                strlcat(mouse->name, dev->product, sizeof(mouse->name));
            }

            if (!strlen(mouse->name))
                snprintf(mouse->name, sizeof(mouse->name),
                     "USB HIDBP Mouse %04x:%04x",
                     le16_to_cpu(dev->descriptor.idVendor),
                     le16_to_cpu(dev->descriptor.idProduct));

            usb_make_path(dev, mouse->phys, sizeof(mouse->phys));
            strlcat(mouse->phys, "/input0", sizeof(mouse->phys));

            input_dev->name = mouse->name;
            input_dev->phys = mouse->phys;
            usb_to_input_id(dev, &input_dev->id);
            input_dev->dev.parent = &intf->dev;

            input_dev->evbit[0] = BIT_MASK(EV_KEY) | BIT_MASK(EV_REL);
            input_dev->keybit[BIT_WORD(BTN_MOUSE)] = BIT_MASK(BTN_LEFT) |
                BIT_MASK(BTN_RIGHT) | BIT_MASK(BTN_MIDDLE);
            input_dev->relbit[0] = BIT_MASK(REL_X) | BIT_MASK(REL_Y);
            input_dev->keybit[BIT_WORD(BTN_MOUSE)] |= BIT_MASK(BTN_SIDE) |
                BIT_MASK(BTN_EXTRA);
            input_dev->relbit[0] |= BIT_MASK(REL_WHEEL);

            input_set_drvdata(input_dev, mouse);

            input_dev->open = usb_mouse_open;
            input_dev->close = usb_mouse_close;

            usb_fill_int_urb(mouse->irq, dev, pipe, mouse->data,
                     (maxp > 8 ? 8 : maxp),
                     usb_mouse_irq, mouse, endpoint->bInterval);
            mouse->irq->transfer_dma = mouse->data_dma;
            mouse->irq->transfer_flags |= URB_NO_TRANSFER_DMA_MAP;

            error = input_register_device(mouse->dev);
            if (error)
                fail3 = 0;

            if (fail3)
            {
                usb_set_intfdata(intf, mouse);
                return 0;
            }


            usb_free_urb(mouse->irq);
        }
```

```
            usb_free_coherent(dev, 8, mouse->data, mouse->data_dma);
        }
    }

    input_free_device(input_dev);
    kfree(mouse);
    return error;
}

static void usb_mouse_disconnect(struct usb_interface *intf)
{
    printk(KERN_INFO "[usbmouse] in usb_mouse_disconnect\n");
    struct usb_mouse *mouse = usb_get_intfdata (intf);

    usb_set_intfdata(intf, NULL);
    if (mouse) {
        usb_kill_urb(mouse->irq);
        input_unregister_device(mouse->dev);
        usb_free_urb(mouse->irq);
        usb_free_coherent(interface_to_usbdev(intf), 8, mouse->data, mouse-
>data_dma);
        kfree(mouse);
    }
}

static const struct usb_device_id usb_mouse_id_table[] = {
    { USB_INTERFACE_INFO(USB_INTERFACE_CLASS_HID, USB_INTERFACE_SUBCLASS_BOOT,
        USB_INTERFACE_PROTOCOL_MOUSE) },
    { } /* Terminating entry */
};

MODULE_DEVICE_TABLE (usb, usb_mouse_id_table);

static struct usb_driver usb_mouse_driver = {
    .name       = "my_usb_mouse_driver",
    .probe      = usb_mouse_probe,
    .disconnect = usb_mouse_disconnect,
    .id_table   = usb_mouse_id_table,
};

module_usb_driver(usb_mouse_driver);
```

Листинг А.4 – Makefile

```
KBUILD_EXTRA_SYMBOLS = $(shell pwd)/Module.symverscd
ifneq ($(KERNELRELEASE),)
    obj-m := driver.o morze_enter.o
else
```

```
    CURRENT = $(shell uname -r)
    KDIR = /lib/modules/$(CURRENT)/build
    PWD = $(shell pwd)

default:
    $(MAKE) -C $(KDIR) M=$(PWD) modules
    make cleanHalf

cleanHalf:
    rm -rf *.o *~ *.mod *.mod.c Module.* *.order  .tmp_versions

clean:
    make cleanHalf
    rm -rf *.ko

endif
```