РЕФЕРАТ

Расчетно-пояснительная 34 с., 15 рис., 4 ист.

Ключевые слова: драйвер, USB-мышь, яркость, цветовая температура.

В рамках данной курсовой работы был разработан драйвер для изменения яркости и цветовой температуры дисплея с использованием USB-мыши.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Одной из важнейших задач операционной системы является организация работы с устройствами ввода-вывода. Наиболее распространенными представителями таких устройств являются стандартные компьютерные мыши, клавиатуры, джойстики, мониторы; они позволяют пользователю интерактивно взаимодействовать с компьютерной системой.

Управление внешними устройствами драйверами, изменение их поведения осуществляется написанием драйверов. Ключевыми характеристиками дисплея компьютера являются яркость и цветовая температура (теплота). Чтобы задать необходимые пользователю (в частности, безопасные для его зрения) значения яркости и теплоты, целесообразно использовать устройство, позволяющее одновременно нажимать на клавиши и осуществлять скроллинг. Одним из таких устройств является USB-мышь.

1 Аналитический раздел

1.1 Постановка задачи

В соответствии с заданием на курсовую работу необходимо разработать драйвер для изменения яркости и цветовой температуры дисплея с использованием USB-мыши.

Для достижения поставленной задачи необходимо:

1) провести анализ USB-подсистемы Linux;
2) провести анализ особенностей USB-устройства и формата передаваемых данных;
3) разработать алгоритм работы драйвера;
4) реализовать программный код драйвера;
5) провести исследование результатов работы реализованного драйвера.

1.2 Особенности шины USB

Устройства USB могут являться хабами, функциями или их комбинацией.

Хаб (hub) — сетевой концентратор; обеспечивает дополнительные точки подключения устройств к шине.

Функции — устройства, способные передавать или принимать данные или управляющую информацию по шине. Функции USB предоставляют системе дополнительные возможности, например подключение акустической системы, мыши и т. п.

С точки зрения топологии, USB подсистема является не шиной, а деревом с одним корнем — хостом (компьютером) с хост-контроллером, в который встроен корневой хаб (root hub) [1]. Хост-контроллер формирует запросы, а устройства посылают ответы. Устройства никогда не отправляют информацию самостоятельно. Запросы хост-контроллера имеют направление: IN — хост отправляет запрос на прием данных, OUT — хост отсылает данные устройству. Схема USB топологии представлена на рисунке 1.

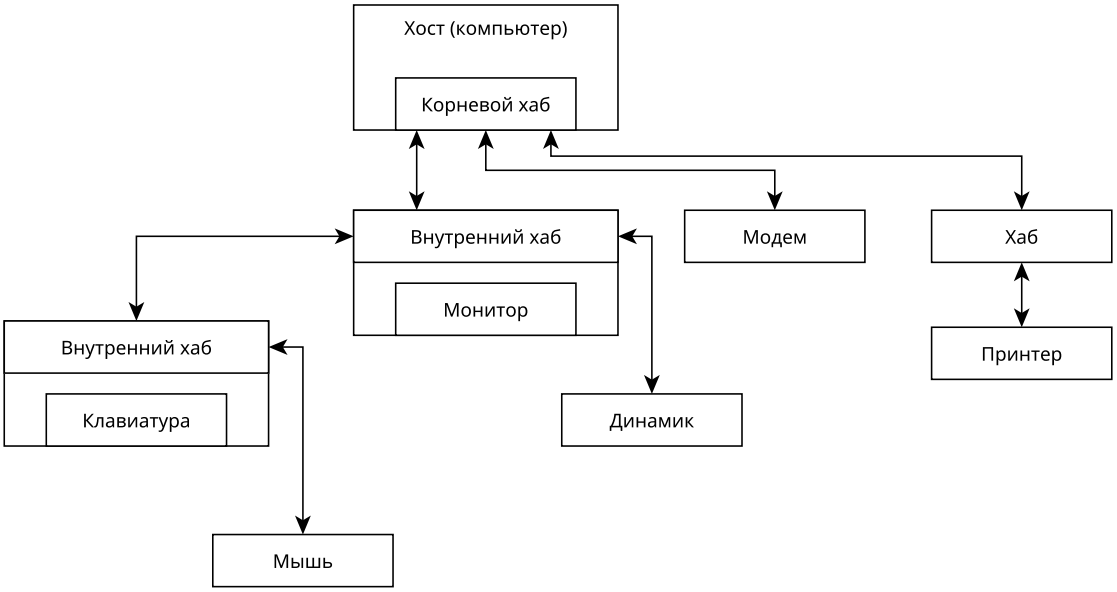

Рисунок 1 — USB топология

Конечные точки (endpoints) — базовые объекты связи интерфейсов USB. Устройство может иметь до 16 конечных точек, нумерация начинается с 0 и заканчивается 15. Каждая конечная точка может включать в себя два буфера (адреса): входной и выходной, то есть устройство может обладать 32 адресами конечных точек. Каждая USB-функция должна содержать как минимум одну (нулевую) конечную точку с входным и выходным буфером.

Каналы (pipes) определяются хостом, они связаны с конечными точками функции. В отличие от конечной точки, которая имеет физическую сущность, канал является всего лишь логической концепцией, правилом. После установки канала, становится определенным и тип передачи данных, который он поддерживает.

Схема связи хоста и USB-интерфейсов представлена на рисунке 2.

Архитектура (структура) USB-драйвера хоста представлена на рисунке 3.

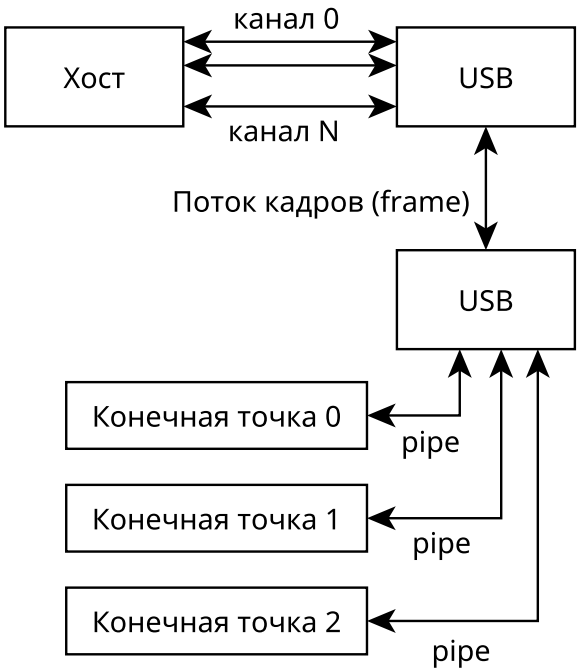

Рисунок 2 — Связь хоста и USB-интерфейсов (конечные точки, каналы)

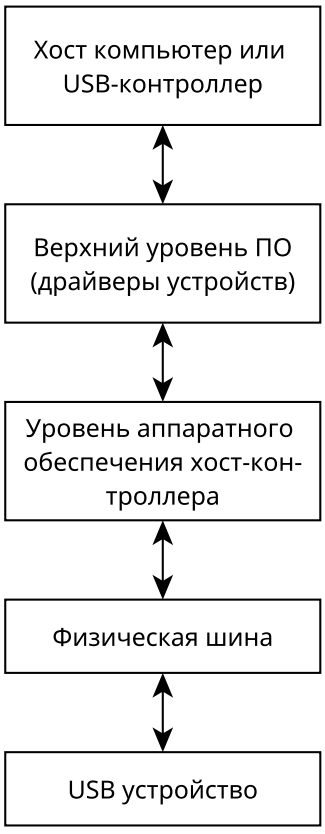

Рисунок 3 — Архитектура (структура) USB-драйвера хоста

Задачей хоста является контроль данных, передающихся от устройства или на него.

Физическая шина — «USB кабель», который соединяет контроллер и периферию; состоит из четырех медных проводников (два отвечают за питание, другие два — витая пара), то есть передача данных выполняется последовательно, а не параллельно.

В процессе передачи данных выполняются следующие действия:

1) USB-устройство инициализирует передачу, используя функции интерфейса USB-драйвера, выдавая запросы драйверу;
2) USB-драйвер отсылает запросы к модулю драйвера контроллера хоста (HCDM);
3) HCDM делит запросы на отдельные транзакции, учитывая особенности шины и USB-устройства, и планирует транзакции по шине;
4) хост-контроллер выполняет или завершает транзакцию (транзакция определяется шиной, устройство полностью зависимо).

Передача осуществляется между буфером хоста и конечной точкой на USB-устройстве. Шина является хост-ориентированной. Хост-контроллер — активная сторона шины.

Существует 4 типа передач:

1) control — передача является двунаправленной, предназначена для обмена с устройством короткими пакетами типа «вопрос-ответ», используется для отправки определенных общих команд на USB-устройство и позволяет программному обеспечению операционной системы прочитать информацию об устройстве (например, коды производителя и модели) (обычно осуществляется конечной точкой 0);
2) isochronous — изохронный канал имеет гарантированную пропускную способность (N пакетов за один период шины) и обеспечивает непрерывную передачу данных, подтверждение приема не требуется, для приложений реального времени;
3) interrupt — канал прерывания позволяет доставлять короткие пакеты без гарантии доставки и без подтверждений приема, но с гарантией времени доставки — пакет будет доставлен не позже, чем через N миллисекунд (до 64 байт на полной скорости, до 8 байт на низкой скорости);

4) bulk — поточная или сплошная передача, используется устройствами, отправляющими и принимающими большое количество данных, но не имеющих определенную пропускную способность, имеется гарантия доставки каждого пакета; bulk пакеты имеют самый низкий приоритет, заполняют всю полосу пропускания шины.

Для взаимодействия с USB-устройствами в ОС Linux предусмотрена структура URB. Эта структура содержит необходимые для описания USB-запроса параметры, включая тип передачи данных, адрес назначения, буфер для данных, размер передачи и обратный вызов для обработки завершения запроса. URB используется для всех типов USB-передач: control, bulk, interrupt и isochronous, что делает URB универсальным инструментом для взаимодействия с различными USB-устройствами. Эта структура представлена на листинге 1.

Листинг 1 — Структура urb

```
struct urb {
  /* private: usb core and host controller only fields in the urb */
  struct kref kref;    /* reference count of the URB */
  int unlinked;      /* unlink error code */
  void *hcpriv;      /* private data for host controller */
  atomic_t use_count;    /* concurrent submissions counter */
  atomic_t reject;    /* submissions will fail */

  /* public: documented fields in the urb that can be used by drivers */
  struct list_head urb_list;  /* list head for use by the urb's
  * current owner */
  struct list_head anchor_list;  /* the URB may be anchored */
  struct usb_anchor *anchor;
  struct usb_device *dev;    /* (in) pointer to associated device */
  struct usb_host_endpoint *ep;  /* (internal) pointer to endpoint */
  unsigned int pipe;    /* (in) pipe information */
  unsigned int stream_id;    /* (in) stream ID */
  int status;      /* (return) non-ISO status */
  unsigned int transfer_flags;  /* (in) URB_SHORT_NOT_OK | ...*/
  void *transfer_buffer;    /* (in) associated data buffer */
  dma_addr_t transfer_dma;  /* (in) dma addr for transfer_buffer */
  struct scatterlist *sg;    /* (in) scatter gather buffer list */
  int num_mapped_sgs;    /* (internal) mapped sg entries */
  int num_sgs;      /* (in) number of entries in the sg list */
  u32 transfer_buffer_length;  /* (in) data buffer length */
  u32 actual_length;    /* (return) actual transfer length */
  unsigned char *setup_packet;  /* (in) setup packet (control only) */
  dma_addr_t setup_dma;    /* (in) dma addr for setup_packet */
  int start_frame;    /* (modify) start frame (ISO) */
  int number_of_packets;    /* (in) number of ISO packets */
```

```
  int interval;      /* (modify) transfer interval
  * (INT/ISO) */
  int error_count;    /* (return) number of ISO errors */
  void *context;      /* (in) context for completion */
  usb_complete_t complete;  /* (in) completion routine */
  struct usb_iso_packet_descriptor iso_frame_desc[];
  /* (in) ISO ONLY */
};
```

1.3 USB драйвер

В Linux имеются драйверы трех типов:

1) низкого уровня — пишутся разработчиками устройств;
2) верхнего уровня — реализованы как загружаемые модули ядра;
3) смешанного уровня — код поделен между ядром и специальной утилитой, управляющей устройством.

Схема взаимодействия прикладных программ с аппаратной частью компьютера: устройство ⇔ ядро ⇔ специальный файл устройства ⇔ программа пользователя.

Драйвер ОС Linux представляет собой загружаемый модуль ядра, который определяет точки входа для работы с устройством. Инициализируются и заполняются следующие структуры:

— структура `usb_driver`, представленная на листинге 2 и описывающая тип устройства, имя, точки входа, коды производителя и модели;

— структура `usb_class_driver`, представленная на листинге 3 и описывающая создаваемый специальный файл (блочный) в директории `/dev/`;

— структура, содержащая данные для работы драйвера с конкретным USB-устройством.

Листинг 2 — Структура usb_driver

```
struct usb_driver {
  const char *name;
  int (*probe) (struct usb_interface *intf,
                const struct usb_device_id *id);
  void (*disconnect) (struct usb_interface *intf);
  int (*unlocked_ioctl) (struct usb_interface *intf, unsigned int code,
                         void *buf);
  int (*suspend) (struct usb_interface *intf, pm_message_t message);
  int (*resume) (struct usb_interface *intf);
```

```
  int (*reset_resume)(struct usb_interface *intf);
  int (*pre_reset)(struct usb_interface *intf);
  int (*post_reset)(struct usb_interface *intf);
  void (*shutdown)(struct usb_interface *intf);
  const struct usb_device_id *id_table;
  const struct attribute_group **dev_groups;
  struct usb_dynids dynids;
  struct device_driver driver;
  unsigned int no_dynamic_id:1;
  unsigned int supports_autosuspend:1;
  unsigned int disable_hub_initiated_lpm:1;
  unsigned int soft_unbind:1;
};
```

Функции, являющиеся основными точками входа драйвера, описаны далее.

— probe() — функция, вызываемая при подключении устройства; заполняет те поля структуры, описывающей конкретное USB-устройство, которые можно получить непосредственно при его подключении (USB-интерфейс устройства, адрес его конечной точки, размер буфера для передачи данных); далее создается URB с соответствующими параметрами и файл в директории /dev/; URB подтверждается, после чего хост начинает отправлять периодические запросы на получение данных с устройства (тип передачи interrupt transfer).

— disconnect() — функция, вызываемая при отключении устройства; удаляет соответствующий файл из директории /dev/.

— read() — переопределенная функция чтения из файла устройства; упаковывает данные ввода из 64-байтового пакета данных в 4-байтовый массив и копирует его в пространство пользователя.

— init() — функция инициализации модуля; регистрирует данный тип устройства в системе.

— exit() — функция выгрузки модуля; выгружает данный тип устройства из системы.

Листинг 3 — Структура usb_class_driver

```
struct usb_class_driver {
  char *name;
  char *(*devnode)(const struct device *dev, umode_t *mode);
  const struct file_operations *fops;
  int minor_base;
};
```

1.4 Особенности USB-мыши и разрабатываемого ПО

Для определения формата передаваемых устройством данных, а также некоторых его параметров была использована программа Wireshark. Данное ПО позволяет перехватить трафик между уже существующим HID-драйвером и устройством.

В результате анализа результатов работы программы Wireshark выяснено, что

— необработанные данные устройства представляют из себя пакет размером 64 байта, из них 4 байта используются для передачи пользовательского ввода;

— 0-й байт — направление движения колесика (0 — не двигается, 1 — вниз, 255 — вверх);

— 1-й байт — смещение мыши по координатам оси X;

— 2-й байт — смещение мыши по координатам оси Y;

— 3-й байт — состояние клавиш мыши (левой — нулевой бит, правой — первый бит, средней — второй бит);

— USB-мышь имеет одну конечную точку с номером 0 по адресу 80h.

Схема поля data пакета, передаваемого с устройства, представлена на рисунке 4.

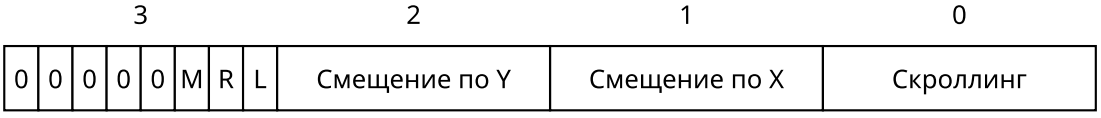

Рисунок 4 — Схема поля data пакета, передаваемого с устройства

Изменение яркости дисплея осуществляется двумя способами:

1) с использованием функции ядра `intel_backlight_set_acpi`;
2) с использованием файловой системы sysfs, осуществляя чтение из и запись в файл `/sys/class/backlight/intel_backlight/brightness`.

Изменение цветовой температуры дисплея осуществляется в пространстве пользователя посредством изменения значения переменной в файле, содержащем настройки GNOME. Это выполняется при помощи утилиты gsettings, пример использования представлен на листинге 4.

Листинг 4 — Использование утилиты gsettings

```
gsettings get org.gnome.settings-daemon.plugins.color night-light-temperature
```

Поскольку был найден только такой способ изменения цветовой температуры дисплея, было принято решение изменять яркость дисплея с использованием файловой системы sysfs.

Таким образом, в качестве типа разрабатываемого программного обеспечения выбраны драйвер, реализованный в виде загружаемого модуля ядра, и демон, который осуществляет чтение данных из специального файла устройства и изменение яркости, теплоты дисплея.

Выводы

В результате проведенного анализа были сделаны следующие выводы:

— для изменения поведения USB-мыши необходимо разработать USB-драйвер, осуществляющий запись данных устройства в специальный файл устройства, и демон, который считывает эти данные и выполняет дальнейшие изменения в системе;

— для изменения яркости необходимо взаимодействовать с файловой системой sysfs;

— для изменения цветовой температуры необходимо использовать утилиту gsettings;

— для определения вида взаимодействия устройства и системы будет использована структура URB.

2 Конструкторский раздел

На рисунках 5, 6 представлены диаграммы IDEF0 нулевого и первого уровней соответственно.

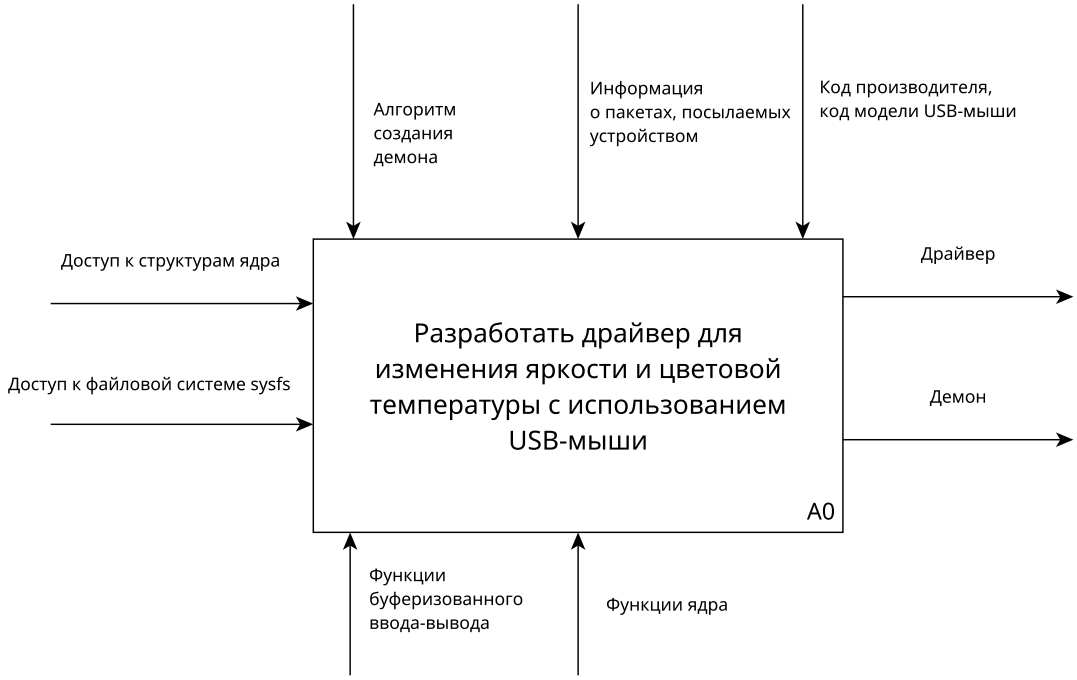

Рисунок 5 — Диаграмма IDEF0 нулевого уровня

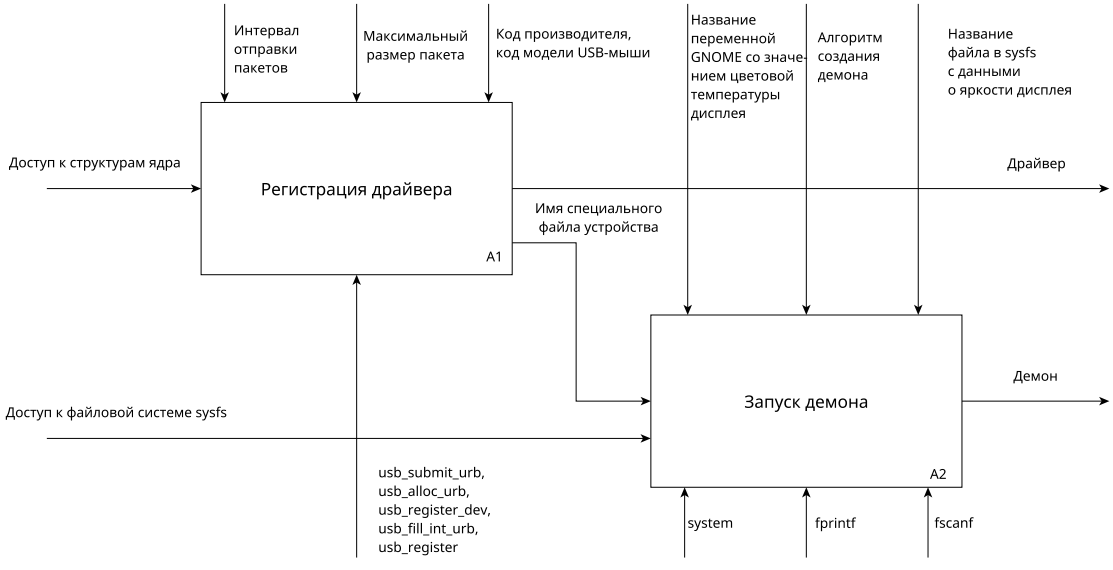

Рисунок 6 — Диаграмма IDEF0 первого уровня

Структура, содержащая данные для работы драйвера с конкретным USB-устройством, представлена на листинге 5.

Листинг 5 — Структура usb_aceline

```
struct usb_aceline {
  struct usb_device *udev;
  struct usb_interface *interface;
  struct urb *intf_in_urb;
  unsigned char *intf_in_buffer;
  unsigned char *file_buffer;
  size_t intf_in_size;
  size_t time;
  __u8 intf_in_endpoint_addr;
  short connected;
};
```

На рисунке 7 представлена диаграмма компонентов.

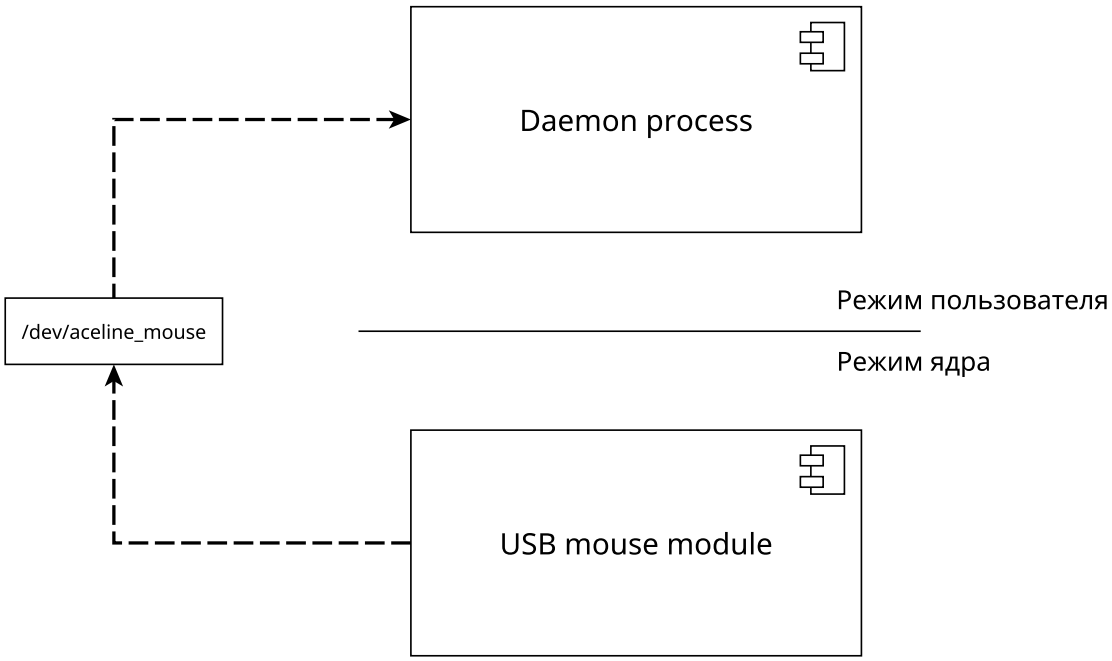

Рисунок 7 — Диаграмма компонентов

На рисунке 8 представлена схема алгоритма изменения яркости дисплея.

На рисунке 9 представлена схема алгоритма изменения цветовой температуры дисплея.

На рисунке 10 представлена схема алгоритма работы демона.

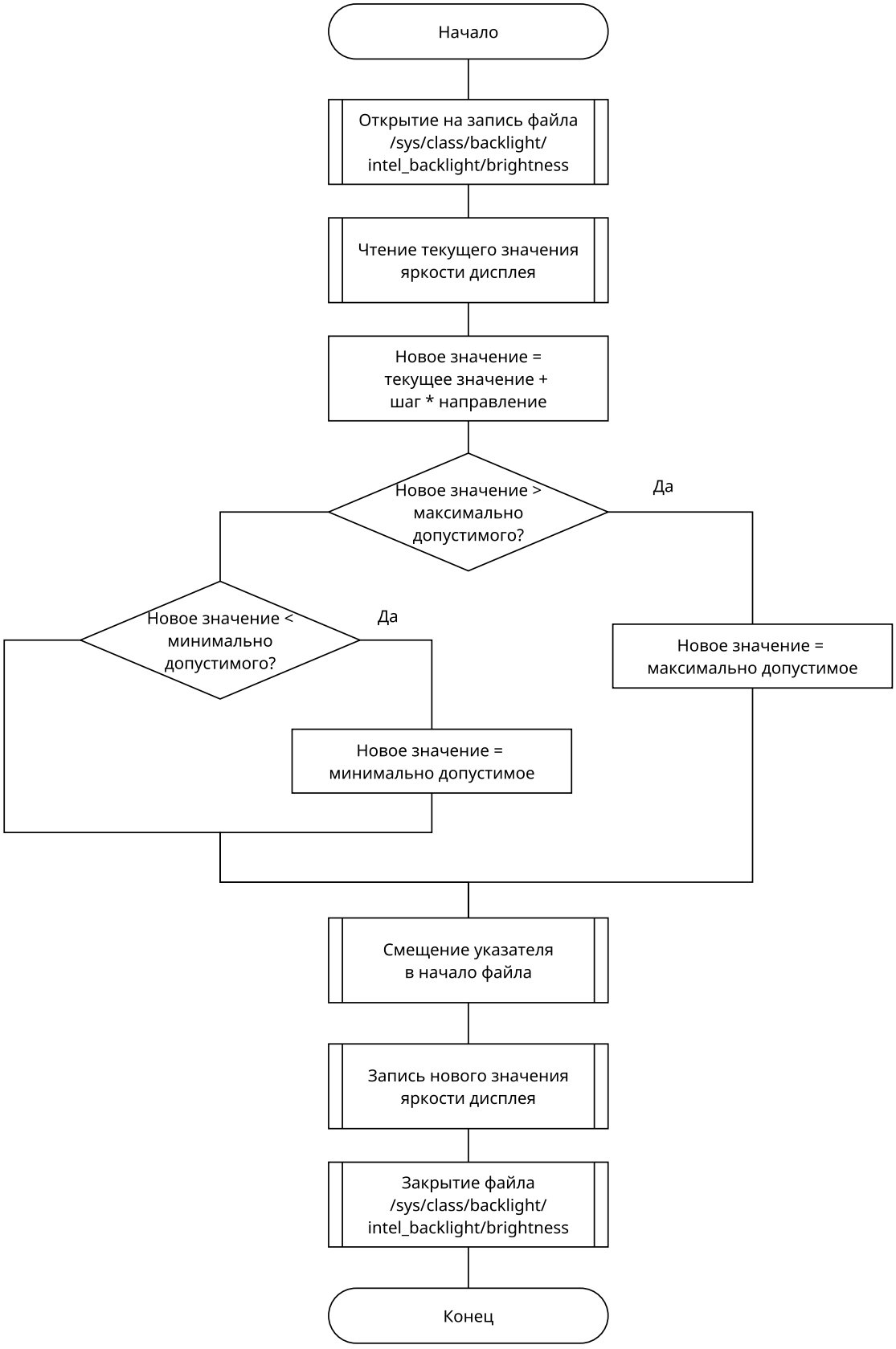

Рисунок 8 — Алгоритм изменения яркости дисплея

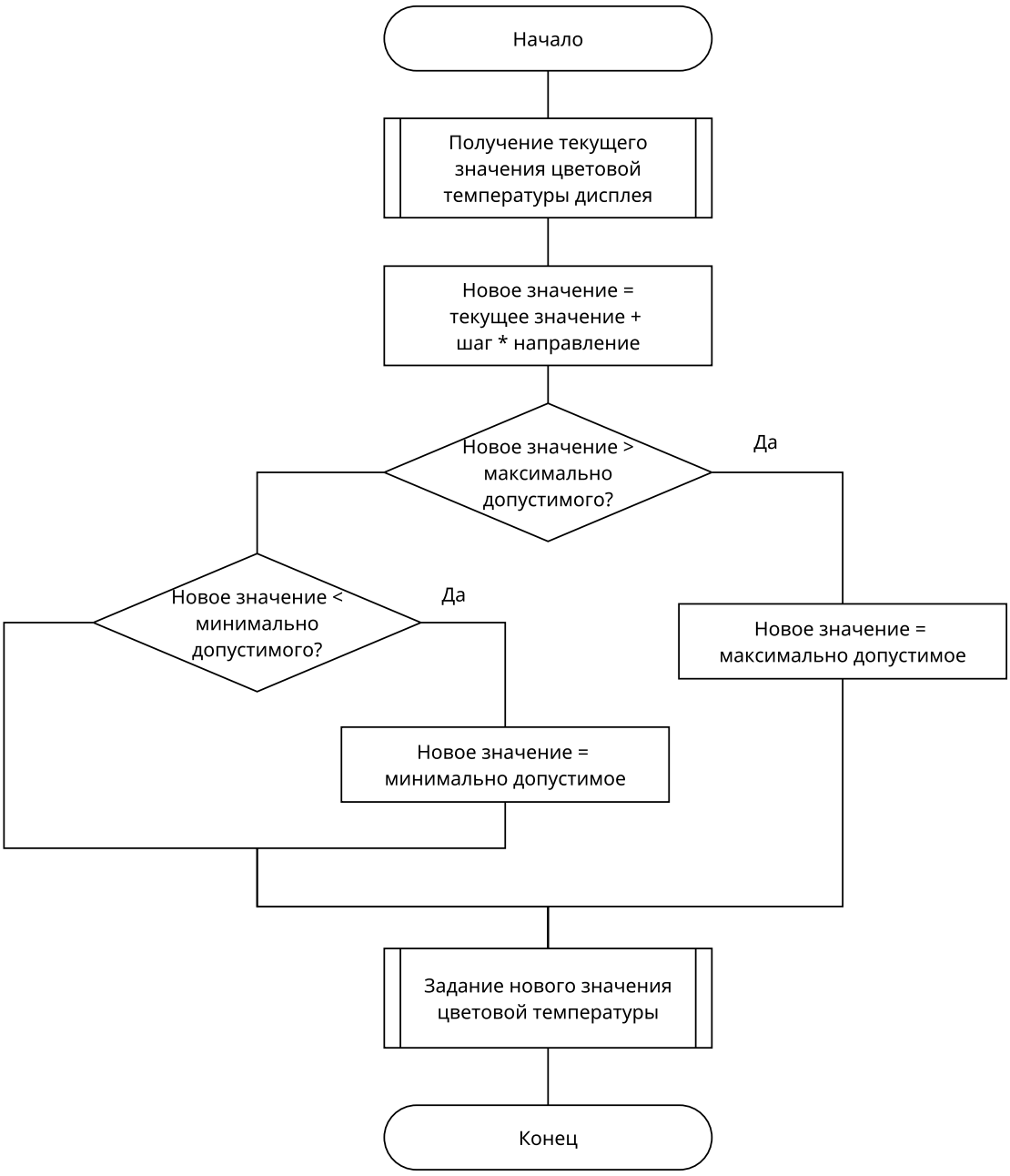

Рисунок 9 — Алгоритм изменения цветовой температуры дисплея

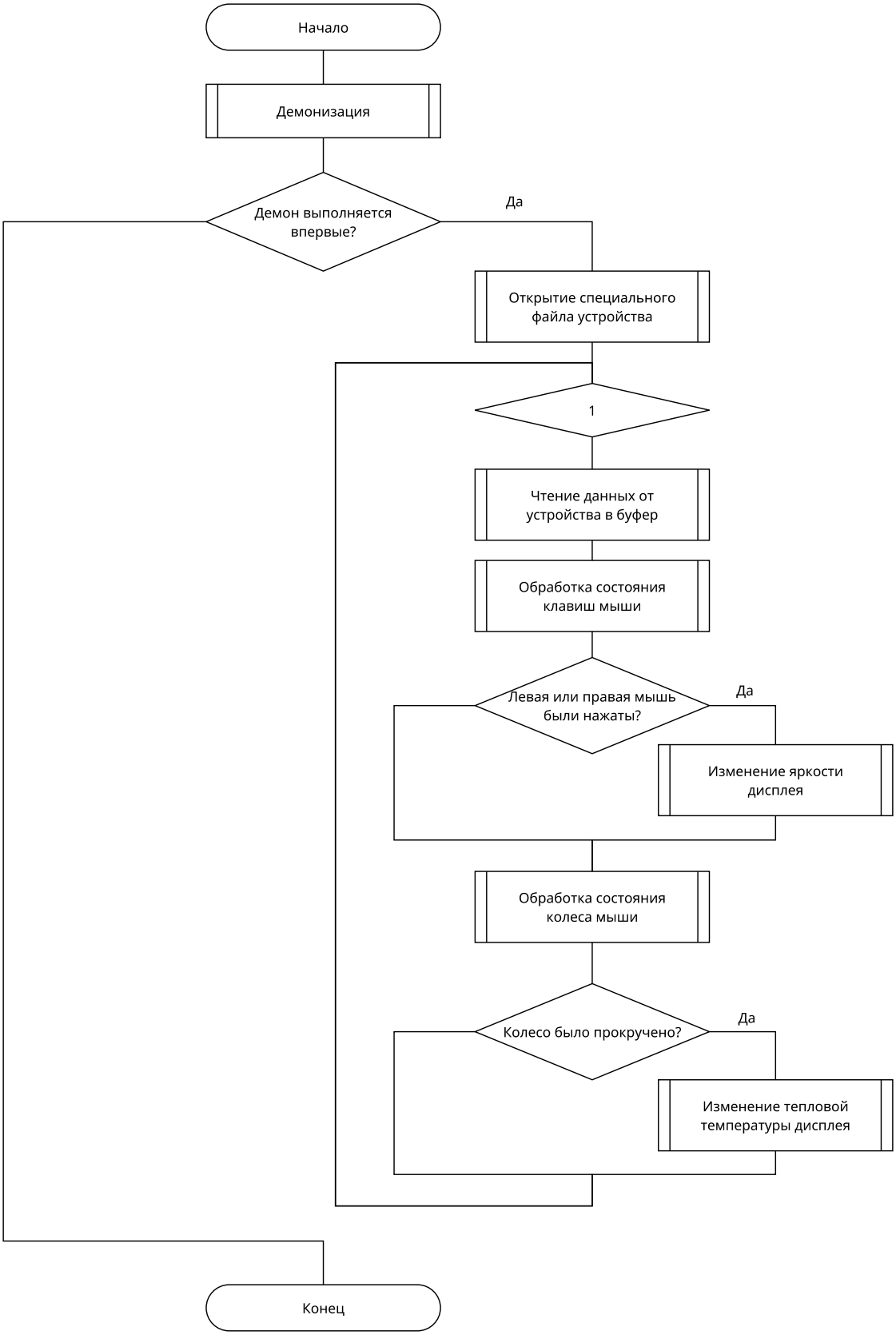

Рисунок 10 — Диаграмма IDEF0 нулевого уровня

3 Технологический раздел

3.1 Выбор языка и среды программирования

В качестве языка программирования выбран язык C [2] стандарта C99, так как на этом языке написан код ядра операционной системы Linux.

Для компиляции модуля был использован GNU Compiler Collection (GCC) [3] — стандартный компилятор UNIX-подобных операционных систем.

Сборка модуля драйвера производится автоматически с помощью утилиты make [4].

3.2 Реализация драйвера

На листинге 6 представлен файл сборки модуля драйвера Makefile, используемый утилитой make.

Листинг 6 — Файл сборки модуля драйвера Makefile

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

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

clean:
  @rm -f *.o .*.cmd .*.flags *.mod.c *.order
  @rm -f .*.*.cmd *~ *.*~ TODO.*
  @rm -fR .tmp*
  @rm -rf .tmp_versions

disclean: clean
  @rm *.ko *.symvers

endif
```

Реализация драйвера представлена на листинге 7.

Листинг 7 — Реализация драйвера

```c
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/input.h>
#include <linux/usb.h>
#include <linux/slab.h>
#include <linux/fs.h>
#include <linux/buffer_head.h>
#include <asm/segment.h>
#include <asm/uaccess.h>
#include <linux/hid.h>

#define MIN(a,b) (((a) <= (b)) ? (a) : (b))

#define DRIVER_NAME "aceline_mouse"
#define MINOR_BASE  111
#define VENDOR_ID   0x1a2c
#define PRODUCT_ID  0x0044
#define INT_MS      10
#define DATA_SIZE_BYTES 4

struct usb_aceline {
    struct usb_device *udev;
    struct usb_interface *interface;
    struct urb *intf_in_urb;
    unsigned char *intf_in_buffer;
    unsigned char *file_buffer;
    size_t intf_in_size;
    size_t time;
    __u8 intf_in_endpoint_addr;
    short connected;
};

static struct usb_aceline device;

static void save_data(unsigned char *buffer)
{
    device.file_buffer[0] = buffer[0];
    device.file_buffer[1] = buffer[1];
    device.file_buffer[2] = buffer[2];
    device.file_buffer[3] = buffer[3];
}

static void aread_intf_callback(struct urb *urb)
{
    int res;

    if (!device.connected)
        return;

```

```
50      res = usb_submit_urb(device.intf_in_urb, GFP_KERNEL);
51
52      if (res < 0) {
53          printk(KERN_INFO "@ ERROR: usb_submit_urb error\n");
54          return;
55      }
56
57      save_data(device.intf_in_buffer);
58  }
59
60  static ssize_t aread(struct file *file, char *buffer, size_t count,
        loff_t *ppos)
61  {
62      printk(KERN_INFO "@ INFO : device call aread\n");
63      int ret;
64      unsigned char *devbuf;
65      devbuf = device.file_buffer;
66      ret = copy_to_user(buffer, devbuf, DATA_SIZE_BYTES);
67      printk(KERN_INFO "@ INFO : device aread success\n");
68      return DATA_SIZE_BYTES;
69  }
70
71  static int aopen(struct inode *inode, struct file *file)
72  {
73      printk(KERN_INFO "@ INFO : device open\n");
74      return 0;
75  }
76
77  static struct file_operations afops = {
78      .read = aread,
79      .open = aopen,
80  };
81
82  static struct usb_class_driver adriver_class = {
83      .name = DRIVER_NAME,
84      .fops = &afops,
85      .minor_base = MINOR_BASE,
86  };
87
88  static int mouse_probe(struct usb_interface *interface, const struct
        usb_device_id *id)
89  {
90      printk(KERN_INFO "@ INFO : mouse_probe\n");
91      int res, ret;
92
93      if (!device.connected) {
94          struct usb_endpoint_descriptor *intf_in;
95          device.udev = usb_get_dev(interface_to_usbdev(interface));
96          device.interface = usb_get_intf(interface);
97
98          res = usb_find_common_endpoints(interface->cur_altsetting, NULL,
                NULL, &intf_in, NULL);
```

```
99          if (res < 0) {
100             printk(KERN_INFO "@ ERROR: no endpoints\n");
101             return -1;
102         }
103         device.intf_in_size = usb_endpoint_maxp(intf_in);
104         device.intf_in_endpoint_addr = intf_in->bEndpointAddress;
105
106         device.intf_in_buffer = kmalloc(device.intf_in_size, GFP_KERNEL);
107         if (device.intf_in_buffer == NULL) {
108             printk(KERN_INFO "@ ERROR: kmalloc intf_in_buffer\n");
109             return -1;
110         }
111
112         device.file_buffer = kmalloc(DATA_SIZE_BYTES, GFP_KERNEL);
113         if (device.file_buffer == NULL) {
114             printk(KERN_INFO "@ ERROR: kmalloc file_buffer\n");
115             return -1;
116
117         }
118
119         device.intf_in_urb = usb_alloc_urb(0, GFP_KERNEL);
120         if (device.intf_in_urb == NULL) {
121             printk(KERN_INFO "@ ERROR: usb_alloc_urb\n");
122             return -1;
123         }
124
125         device.time = INT_MS;
126
127         ret = 0;
128         printk(KERN_INFO "@ INFO : device connect success\n");
129         printk(KERN_INFO "@ INFO : intf_in_endpoint_addr = %d\n",
                device.intf_in_endpoint_addr);
130
131         ret = usb_register_dev(interface, &adriver_class);
132         if (ret != 0) {
133             printk(KERN_INFO "@ ERROR: usb_register_dev error\n");
134             return -1;
135         }
136
137         printk(KERN_INFO "@ INFO : device driver register success\n");
138
139         usb_fill_int_urb(
140             device.intf_in_urb,
141             device.udev,
142             usb_rcvintpipe(device.udev, device.intf_in_endpoint_addr),
143             device.intf_in_buffer,
144             device.intf_in_size,
145             aread_intf_callback,
146             NULL,
147             device.time
148         );
149
```

```
150         printk(KERN_INFO "@ INFO : usb_fill_int_urb success\n");
151
152         res = usb_submit_urb(device.intf_in_urb, GFP_KERNEL);
153         if (res < 0) {
154             printk(KERN_INFO "@ ERROR: usb_submit_urb error\n");
155             return -1;
156         }
157
158         device.connected = 1;
159         printk(KERN_INFO "@ INFO : device connection success\n");
160
161         return 0;
162     }
163
164     printk(KERN_INFO "@ ERROR: device in use\n");
165     return -1;
166 }
167
168 static void mouse_disconnect(struct usb_interface *interface)
169 {
170     printk(KERN_INFO "@ INFO : call mouse_disconnect\n");
171     usb_set_intfdata(device.interface, NULL);
172     device.connected = 0;
173     usb_deregister_dev(interface, &adriver_class);
174     usb_kill_urb(device.intf_in_urb);
175     printk(KERN_INFO "@ INFO : mouse disconnect\n");
176 }
177
178 static struct usb_device_id mouse_usb_id_table[] = {
179     {USB_DEVICE(VENDOR_ID, PRODUCT_ID)},
180     {}
181 };
182 MODULE_DEVICE_TABLE(usb, mouse_usb_id_table);
183
184 static struct usb_driver adriver = {
185     .name       = DRIVER_NAME,
186     .probe      = mouse_probe,
187     .disconnect = mouse_disconnect,
188     .id_table   = mouse_usb_id_table,
189 };
190
191 static int __init my_driver_init(void)
192 {
193     printk(KERN_INFO "@ INFO : call my_driver_init\n");
194     int r;
195     device.connected = 0;
196     r = usb_register(&adriver);
197
198     if (r < 0) {
199         printk(KERN_ERR "@ ERROR: usb_register error, return code %d\n",
            r);
200         return -1;
```

```
201     }
202
203     printk(KERN_INFO "@ INFO : driver load\n");
204     return 0;
205 }
206
207 static void __exit my_driver_exit(void)
208 {
209     printk(KERN_INFO "@ INFO : call my_driver_exit\n");
210     usb_deregister(&adriver);
211 }
212
213 module_init(my_driver_init);
214 module_exit(my_driver_exit);
215
216 MODULE_LICENSE("GPL");
217 MODULE_AUTHOR("Valeria Avdeykina");
```

3.3 Реализация демона

Реализация демона представлена на листинге 8.

Листинг 8 — Реализация демона

```
1  #include <syslog.h>
2  #include <fcntl.h>
3  #include <sys/resource.h>
4  #include <sys/stat.h>
5  #include <sys/time.h>
6  #include <unistd.h>
7  #include <stdio.h>
8  #include <stdint.h>
9  #include <signal.h>
10 #include <string.h>
11 #include <linux/input.h>
12 #include <errno.h>
13 #include <stdlib.h>
14 #include <math.h>
15 #include <gio/gio.h>
16
17 #define LOCKFILE "/var/run/daemon.pid"
18 #define LOCKMODE (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)
19
20 #define DATA_SIZE_BYTES 4
21
22 #define MIN_BRIGHTNESS  193
23 #define MAX_BRIGHTNESS  19393
```

```
24  #define STP_BRIGHTNESS  1
25
26  #define MIN_TEMPERATURE 1700
27  #define MAX_TEMPERATURE 4700
28  #define STP_TEMPERATURE 20
29
30  #define ACELINE_BUTTON_LEFT 0b00000001
31  #define ACELINE_BUTTON_RGHT 0b00000010
32  #define ACELINE_BUTTON_MDDL 0b00000100
33  #define ACELINE_SCROLL_UP   0b00000001
34  #define ACELINE_SCROLL_DOWN '\xff'
35
36  #define GET_NLTEMP_COMMAND  "sudo -H -u sheglar DISPLAY=:0
        DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus gsettings get
        org.gnome.settings-daemon.plugins.color night-light-temperature >
        /home/sheglar/.aceline.tmp"
37  #define GET_NLTEMP_VALUE    "sudo tail -c 5 /home/sheglar/.aceline.tmp >
        /home/sheglar/.aceline.tmp1"
38  #define SET_NLTEMP_COMMAND  "sudo -H -u sheglar DISPLAY=:0
        DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus gsettings set
        org.gnome.settings-daemon.plugins.color night-light-temperature %d"
39  #define ACELINE_FILENAME    "/dev/aceline_mouse"
40  #define NLTEMP_FILENAME     "/home/sheglar/.aceline.tmp1"
41  #define BRIGHTNESS_FILENAME
        "/sys/class/backlight/intel_backlight/brightness"
42
43  int lockfile(int fd)
44  {
45      struct flock fl;
46      fl.l_type = F_WRLCK;
47      fl.l_start = 0;
48      fl.l_whence = SEEK_SET;
49      fl.l_len = 0;
50      return(fcntl(fd, F_SETLK, &fl));
51  }
52
53  int already_running(void)
54  {
55      int fd;
56      char buf[16];
57
58      fd = open(LOCKFILE, O_RDWR|O_CREAT, LOCKMODE);
59      if (fd < 0)
60      {
61          syslog(LOG_ERR, "невозможно открыть %s: %s", LOCKFILE,
                strerror(errno));
62          exit(1);
63      }
64      if (lockfile(fd) < 0)
65      {
66          if (errno == EACCES || errno == EAGAIN)
67          {
```

```
68              close(fd);
69              return(1);
70          }
71          syslog(LOG_ERR, "невозможно установить блокировку на %s: %s",
                LOCKFILE, strerror(errno));
72          exit(1);
73      }
74      ftruncate(fd, 0);
75
76      return 0;
77  }
78
79  void daemonize(const char *cmd)
80  {
81      int i, fd0, fd1, fd2;
82      pid_t pid;
83      struct rlimit r1;
84      struct sigaction sa;
85
86      umask(0);
87
88      if (getrlimit(RLIMIT_NOFILE, &r1) < 0)
89      {
90          perror("Невозможно получить макс номер дескриптора");
91          exit(1);
92      }
93
94      if ((pid = fork()) < 0)
95      {
96          perror("Fork error");
97          exit(1);
98      }
99      else if (pid != 0)
100         exit(0);
101     setsid();
102
103     sa.sa_handler = SIG_IGN;
104     sigemptyset(&sa.sa_mask);
105     sa.sa_flags = 0;
106     if (sigaction(SIGHUP, &sa, NULL) < 0)
107     {
108         perror("Cant ignore sighup");
109         exit(1);
110     }
111
112     if ((pid = fork()) < 0)
113     {
114         perror("Fork error");
115         exit(1);
116     }
117     else if (pid != 0)
118         exit(0);
```

```
119
120     if (chdir("/") < 0)
121     {
122         perror("Error");
123         exit(1);
124     }
125
126     if (r1.rlim_max == RLIM_INFINITY)
127         r1.rlim_max = 1024;
128     for (i=0; i<r1.rlim_max; i++)
129         close(i);
130
131     fd0 = open("/dev/null", O_RDWR);
132     fd1 = dup(0);
133     fd2 = dup(0);
134
135     openlog(cmd, LOG_CONS, LOG_DAEMON);
136     if (fd0 != 0 || fd1 != 1 || fd2 != 2)
137     {
138         syslog(LOG_ERR, "ошибочные файловые дескрипторы %d %d %d", fd0,
                fd1, fd2);
139         exit(1);
140     }
141 }
142
143 void change_brightness(int direction)
144 {
145     if (!direction)
146         return;
147
148     FILE *brightness_fd = fopen(BRIGHTNESS_FILENAME, "r");
149
150     int cur_value = MIN_BRIGHTNESS;
151     fscanf(brightness_fd, "%d", &cur_value);
152     int new_value = cur_value + STP_BRIGHTNESS * direction;
153
154     if (new_value < MIN_BRIGHTNESS)
155         new_value = MIN_BRIGHTNESS;
156     else if (new_value > MAX_BRIGHTNESS)
157         new_value = MAX_BRIGHTNESS;
158
159
160     brightness_fd = fopen(BRIGHTNESS_FILENAME, "w");
161     fseek(brightness_fd, 0, SEEK_SET);
162     fprintf(brightness_fd, "%d\n", new_value);
163     fclose(brightness_fd);
164 }
165
166 void change_nltemp(int direction)
167 {
168     if (!direction)
169         return;
```

```
170
171     system(GET_NLTEMP_COMMAND);
172     system(GET_NLTEMP_VALUE);
173
174     FILE *fd = fopen(NLTEMP_FILENAME, "r");
175     int cur_value = MIN_TEMPERATURE;
176     fscanf(fd, "%d", &cur_value);
177     fclose(fd);
178
179     int new_value = cur_value + STP_TEMPERATURE * direction;
180
181     if (new_value < MIN_TEMPERATURE)
182         new_value = MIN_TEMPERATURE;
183     else if (new_value > MAX_TEMPERATURE)
184         new_value = MAX_TEMPERATURE;
185
186     char command[200];
187     snprintf(command, 200, SET_NLTEMP_COMMAND, new_value);
188     system(command);
189 }
190
191 int get_brightness_direction(char *buffer)
192 {
193     if (!(buffer[0] ^ ACELINE_BUTTON_LEFT))
194         return 1;
195
196     if (!(buffer[0] ^ ACELINE_BUTTON_RGHT))
197         return -1;
198
199     return 0;
200 }
201
202 int get_nltemp_direction(char *buffer)
203 {
204     if (!(buffer[3] ^ ACELINE_SCROLL_UP))
205         return -1;
206
207     if (buffer[3] == ACELINE_SCROLL_DOWN)
208         return 1;
209
210     return 0;
211 }
212
213 int main()
214 {
215     daemonize("ddaceline");
216     syslog(LOG_WARNING, "@ DAEMON daemonize success\n");
217
218     if (already_running() != 0) {
219         syslog(LOG_ERR, "@ DAEMON already_running\n");
220         exit(1);
221     }
```

```
222
223     syslog(LOG_WARNING, "@ DAEMON already_running success\n");
224     syslog(LOG_WARNING, "@ DAEMON start process\n");
225     syslog(LOG_WARNING, "@ DAEMON pid=%d\n", getpid());
226
227     int aceline_fd = open(ACELINE_FILENAME, O_RDONLY);
228
229     if (aceline_fd < 0) {
230         syslog(LOG_ERR, "@ DAEMON open /dev/aceline_mouse error: %s\n",
                strerror(errno));
231         return -1;
232     }
233
234     syslog(LOG_WARNING, "@ DAEMON open aceline_mouse success\n");
235
236     while (1) {
237         unsigned char buffer[DATA_SIZE_BYTES];
238         ssize_t ret = read(aceline_fd, buffer, DATA_SIZE_BYTES);
239         int d_brightness = get_brightness_direction(buffer);
240
241         if (d_brightness != 0) {
242             change_brightness(d_brightness);
243         }
244
245         int d_nltemp = get_nltemp_direction(buffer);
246
247         if (d_nltemp != 0) {
248             change_nltemp(d_nltemp);
249         }
250     }
251 }
```

4 Исследовательский раздел

Программное обеспечение реализовано на дистрибутиве Ubuntu 24.04 LTS Linux, версия ядра 6.8.0-52-generic.

На рисунке 11 представлены записи из системного журнала в результате инициализации разработанного драйвера.

```
[  +0,993817] + INFO : call my_driver_init
[  +0,000086] usbcore: registered new interface driver aceline_mouse
[  +0,000002] + INFO : driver load
[ +27,230262] + INFO : mouse_probe
[  +0,000011] + INFO : Interrupt In Endpoint:
[  +0,000001] + INFO :   Endpoint Address: 81
[  +0,000003] + INFO :   Max Packet Size: 4
[  +0,000001] + INFO :   Endpoint Type: 3
[  +0,000001] + INFO :   Interval: 10
[  +0,000004] + INFO : device connect success
[  +0,000001] + INFO : intf_in_endpoint_addr = 129
[  +0,000155] + INFO : device driver register success
[  +0,000002] + INFO : usb_fill_int_urb success
[  +0,000043] + INFO : device connection success
```

Рисунок 11 — Инициализация разработанного драйвера

На рисунке 12 представлены записи из системного журнала в результате запуска разработанного демона.

```
:ac:1e:4f:08:00 SRC=192.168.1.56 DST=224.0.0.251 LEN=32 TOS=0x00 PREC=0x00 TTL=1 ID=47458 PROTO=2
2025-02-04T22:06:34.255140+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON daemonize success
2025-02-04T22:06:34.255368+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON already_running success
2025-02-04T22:06:34.255430+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON start process
2025-02-04T22:06:34.255466+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON pid=129252
2025-02-04T22:06:34.255502+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON open aceline_mouse success
2025-02-04T22:06:34.255543+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON open brightness success
2025-02-04T22:06:34.255586+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON scroll: 00000000
2025-02-04T22:06:34.255754+03:00 sheglar-ThinkPad-X13-Gen-4 kernel: @ INFO : device open
```

Рисунок 12 — Запуск разработанного демона

На рисунках 13, 14 представлены записи из системного журнала в результате прокрутки колесика мыши и нажатия кнопок соответственно.

```
2025-02-04T22:07:10.075651+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON nltemp direction=1, new_value=3455, cur_value=3435
2025-02-04T22:07:10.075682+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON scroll: 11111111
2025-02-04T22:07:10.152028+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON nltemp direction=1, new_value=3475, cur_value=3455
2025-02-04T22:07:10.152051+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON scroll: 11111111
2025-02-04T22:07:10.221615+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON nltemp direction=1, new_value=3495, cur_value=3475
2025-02-04T22:07:10.221638+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON scroll: 11111111
2025-02-04T22:07:10.286759+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON nltemp direction=1, new_value=3515, cur_value=3495
2025-02-04T22:07:10.286791+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON scroll: 11111111
2025-02-04T22:07:10.354009+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON nltemp direction=1, new_value=3535, cur_value=3515
2025-02-04T22:07:10.354030+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON scroll: 11111111
2025-02-04T22:07:10.426680+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON nltemp direction=1, new_value=3555, cur_value=3535
2025-02-04T22:07:10.426723+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON scroll: 11111111
2025-02-04T22:07:10.498233+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON nltemp direction=1, new_value=3575, cur_value=3555
2025-02-04T22:07:10.498274+03:00 sheglar-ThinkPad-X13-Gen-4 ddaceline: @ DAEMON scroll: 11111111
```

Рисунок 13 — Прокрутка колесика мыши

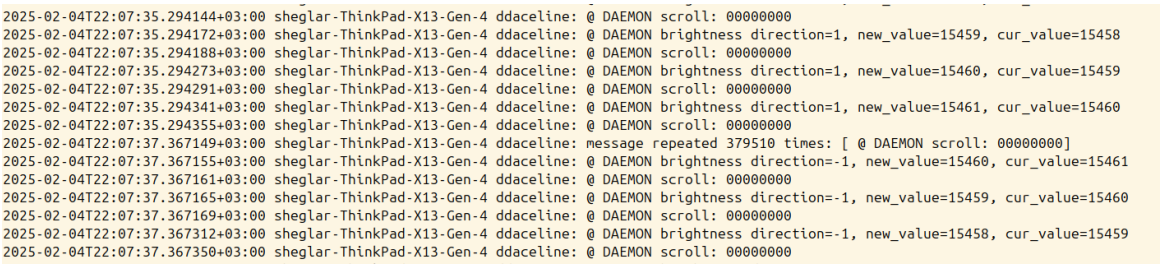

Рисунок 14 — Нажатие кнопок мыши

На рисунке 15 представлены записи из системного журнала в результате завершения разработанного драйвера.

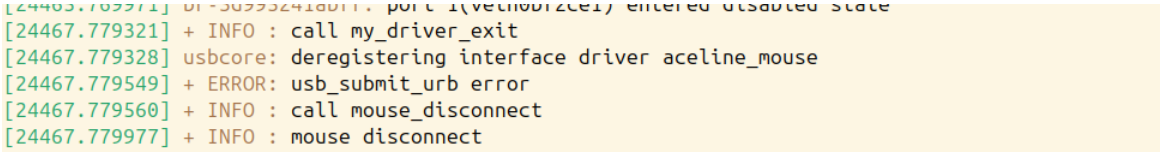

Рисунок 15 — Завершение разработанного драйвера

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были проанализированы особенности USB-подсистемы Linux, типы драйверов в ОС Linux, описаны структуры и функции ядра, используемые для реализации USB-драйвера.

Выполнен анализ способов изменения яркости и цветовой температуры дисплея.

Спроектированы и реализованы драйвер (как загружаемый модуль ядра) и демон, анализирующий данные, передаваемые драйвером, и меняющий яркость и цветовую температуру дисплея при нажатии кнопок и прокрутке колесика.

Проведено исследование результатов работы реализованного драйвера.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Corbet J., Rubini A., Kroah-Hartman G. Linux device drivers. — O'Reilly Media, Inc., 2005.
2. The GNU C Reference Manual [Электронный ресурс]. — Режим доступа: https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html (дата обращения: 04.02.2025).
3. GCC, the GNU Compiler Collection [Электронный ресурс]. — Режим доступа: https://gcc.gnu.org (дата обращения: 04.02.2025).
4. GNU Make [Электронный ресурс]. — Режим доступа: https://www.gnu.org/software/make/ (дата обращения: 04.02.2025).