Оглавление

Введение

В настоящее время вычислительные мощности компьютера это один из самых ценных ресурсов любой компании. Графические ускорители необходимы для реализации любых высокотехнологичных проектов.

В связи с высокой ценой графических вычислительных мощностей компьютера, вероятность взлома и использования в целях злоумышленников компьютеров, оснащёнными такими мощностями повышена.

Существует много способов для проведения кибератаки, одним из которых является атака при помощи USB–устройства. При проведении такой атаки при подключении устройства к компьютеру можно запустить вредоносный программный код на выполнение, запустить ПО, паразитирующее графические ускорители в целях злоумышленников [1]. Самыми часто встречающмися ПО такого предназначения является майнинг программы.

Для того, чтобы предотвратить кибератаку, проводимую посредством подключенного USB–устройства, следует строго отслеживать активные устройства в системе. Возможности запретить подключать новые устройства к компьютеру нет, так как большинство устройств ввода–вывода подключаются к USB порту, поэтому мониторинг активных устройств, их анализ и последующее принятие решений являются хорошим способом для избежания кибератаки.

Чтобы обезопасить свои вычислительные мощности, при подключении устройства можно отключать графический ускоритель. Таким образом исключается возможность использования вычислительных мощностей в целях злоумышленников.

Цель работы — разработать загружаемый модуля ядра Linux для отключения графического ускорителя при подключении USB–устройства.

1 Аналитический раздел

1.1 Постановка задачи

В соответствии с заданием на курсовую работу необходимо разработать загружаемый модуля ядра Linux для отключения графического ускорителя при подключении USB–устройства. Для решения данной задачи необходимо:

- проанализировать методы обработки событий, возникающих при взаимодействии с USB–устройствами;
- проанализировать структуры и функции ядра, предоставляющие информацию о USB–устройствах;
- проанализировать способы взаимодействия с графического ускорителя;
- разработать алгоритмы и структуру программного обеспечения;
- реализовать программное обеспечение;
- проанализировать разработанное программное обеспечение.

1.2 Обработка событий от USB–устройств

Для обработки событий, возникающих при подключении USB–устройств, необходимо перехватить событие подключения USB–устройства и выполнить необходимую обработку.

Ниже будут рассмотрены существующие подходы к определению возникновения событий подключения USB–устройств и выбран наиболее подходящий для реализации.

1.2.1 Уведомители ядра

Ядро Linux содержит механизм, называемый «уведомителями» (`notifiers`) или «цепочками уведомлений» (`notifiers chains`), который позволяет различным подсистемам подписываться на асинхронные события от других подсистем. Цепочки уведомлений в настоящее время активно используется в ядре; существуют цепочки для событий `hotplug` памяти, изменения политики частоты процессора, события `USB hotplug`, загрузки и выгрузки модулей [2].

В листинге 1.1 представлена структура `notifier_block` [3].

Листинг 1.1 – Структура `notifier_block`

```
1 struct notifier_block {
2     notifier_fn_t notifier_call;
3     struct notifier_block __rcu *next;
4     int priority;
5 };
```

Данная структура описана в `/include/linux/notifier.h`. Она содержит указатель на функцию–обработчик уведомления (`notifier_call`), указатель на следующий уведомитель (`next`) и приоритет уведомителя (`priority`). Уведомители с более высоким значением приоритета выполняются первее.

В листинге 1.2 представлен прототип функии `notifier_call`.

Листинг 1.2 – Тип `notifier_fn_t`

```
1 typedef int (*notifier_fn_t)(struct notifier_block *nb, unsigned long action, void
      *data);
```

Прототип содержит указатель на уведомитель (`nb`), действие, при котором вызывается функция (`action`) и данные, которые передаются от действия в обработчик (`data`).

Для регистрации уведомителя для USB–портов используются функции регистрации и удаления уведомителя, представленные в листинге 1.3.

Листинг 1.3 – Уведомители на USB–портах

```
/* Events from the usb core */
#define USB_DEVICE_ADD  0x0001
#define USB_DEVICE_REMOVE 0x0002
#define USB_BUS_ADD  0x0003
#define USB_BUS_REMOVE  0x0004
extern void usb_register_notify(struct notifier_block *nb);
extern void usb_unregister_notify(struct notifier_block *nb);
```

Прототипы и константы для действий описаны в файле `/include/linux/notifier.h`, а реализации функций — в файле `/drivers/usb/core/notify.c`. Соответственно действие `USB_DEVICE_ADD` означает подключение нового устройства, а `USB_DEVICE_REMOVE` — отключения.

В листинге 1.4 представлены функции уведомителя из файла `/drivers/usb/core/notify.c` для подлючения и отключения USB–устройства.

Листинг 1.4 – Функции уведомители для подлючения и отключения USB–устройства

```
void usb_notify_add_device(struct usb_device *udev)
{
    blocking_notifier_call_chain(&usb_notifier_list, USB_DEVICE_ADD, udev);
}

void usb_notify_remove_device(struct usb_device *udev)
{
    blocking_notifier_call_chain(&usb_notifier_list, USB_DEVICE_REMOVE, udev);
}
```

Особенности `уведомителей`:

- возможность привязки своего обработчика к событию;
- возможность добавления более чем одного обработчика событий;
- возможность использования интерфейса в загружаемом модуле ядра;

1.2.2 usbmon

`usbmon` [4] — средство ядра Linux, предназначенное для сбора информации о событиях, связанных с устройствами ввода–вывода, подключенных к USB порту.

`usbmon` предоставляет информацию о запросах, сделанных драйверами устройств к драйверам хост–контроллера (HCD). Если драйвера хост–контроллера неисправны, то данные, предоставленные `usbmon`, могут не соответствовать действительным переданным данным.

В настоящее время реализованы два программных интерфейса для взаимодействия с `usbmon`: текстовый и двоичный. Двоичный интерфейс доступен с помощью символьного устройства в пространстве имен `/dev`. Текстовый интерфейс устарел, но сохраняется для совместимости.

В листинге 1.5 представлена структура ответа, полученного после события, случившегося на USB–устройстве (например, подключение к компьютеру).

Листинг 1.5 – Структура `usbmon_packet`

```
struct usbmon_packet {
    u64 id;   /* 0: URB ID - from submission to callback */
    unsigned char type; /* 8: Same as text; extensible. */
    unsigned char xfer_type; /* ISO (0), Intr, Control, Bulk (3) */
    unsigned char epnum; /* Endpoint number and transfer direction */
    unsigned char devnum; /* Device address */
    u16 busnum;  /* 12: Bus number */
    char flag_setup; /* 14: Same as text */
    char flag_data;  /* 15: Same as text; Binary zero is OK. */
    s64 ts_sec;  /* 16: gettimeofday */
    s32 ts_usec;  /* 24: gettimeofday */
    int status;  /* 28: */
    unsigned int length; /* 32: Length of data (submitted or actual) */
    unsigned int len_cap; /* 36: Delivered length */
    union {   /* 40: */
        unsigned char setup[SETUP_LEN]; /* Only for Control S-type */
        struct iso_rec {  /* Only for ISO */
            int error_count;
            int numdesc;
        } iso;
    } s;
    int interval;  /* 48: Only for Interrupt and ISO */
    int start_frame; /* 52: For ISO */
```

```
    unsigned int xfer_flags; /* 56: copy of URB's transfer_flags */
    unsigned int ndesc; /* 60: Actual number of ISO descriptors */
};    /* 64 total length */
```

Особенности `usbmon`:

- возможность просматривать собранную информацию с помощью специального ПО (например, `Wireshark`);
- возможность отслеживать события на одном порте USB или на всех сразу;
- отсутствие возможности вызова обработчика при возникновении определенного события.

`usbmon` позволяет отслеживать события, но не позволяет реагировать на них без программной доработки для реализации обработчика.

1.2.3 udevadm

`udevadm` [5] — инструмент для управления устройствами `udev`. Структура `udev` описана в библиотеке `libudev` [6], которая не является системной библиотекой Linux. В данной библиотеке представлен программный интерфейс для мониторинга и взаимодействия с локальными устройствами.

При помощи `udevadm` можно получить полную информацию об устройстве, полученную из его представления в `sysfs`, чтобы создать корректные правила и обработчики событий для устройства. Кроме того можно получить список событий для устройства, установить наблюдение за ним.

В листинге 1.6 представлен пример правила обработки событий, задаваемого с помощью `udevadm`.

Листинг 1.6 – Правила `udevadm`

```
/* rules file */
SUBSYSTEM=="usb", ACTION=="add", ENV{DEVTYPE}=="usb_device", RUN+="/bin/device_added.sh"
SUBSYSTEM=="usb", ACTION=="remove", ENV{DEVTYPE}=="usb_device",
    RUN+="/bin/device_removed.sh"

/* device_added.sh */
#!/bin/bash
```

```
7 echo "USB␣device␣added␣at␣$(date)" >>/tmp/scripts.log
8
9 /* device_removed.sh */
10 #!/bin/bash
11 echo "USB␣device␣removed␣␣at␣$(date)" >>/tmp/scripts.log
```

Особенности `udevadm`:

- возможность привязки своего обработчика к событию;
- невозможность использования интерфейса в ядре Linux;

1.3 USB–устройства в ядре Linux

1.3.1 Структура `usb_device`

Для хранения информации о USB–устройстве в ядре используется структура `usb_device`, описанная в `/inlclude/linux/usb.h` [7].

Структура `usb_device` предствалена в листинге 1.7.

Листинг 1.7 – Структура `usb_device`

```
1  struct usb_device {
2      int  devnum;
3      char  devpath[16];
4      u32  route;
5      enum usb_device_state state;
6      enum usb_device_speed speed;
7      unsigned int  rx_lanes;
8      unsigned int  tx_lanes;
9      enum usb_ssp_rate ssp_rate;
10
11     struct usb_tt *tt;
12     int  ttport;
13
14     unsigned int toggle[2];
15
16     struct usb_device *parent;
17     struct usb_bus *bus;
18     struct usb_host_endpoint ep0;
19
20     struct device dev;
21
```

```
22      struct usb_device_descriptor descriptor;
23      struct usb_host_bos *bos;
24      struct usb_host_config *config;
25
26      struct usb_host_config *actconfig;
27      struct usb_host_endpoint *ep_in[16];
28      struct usb_host_endpoint *ep_out[16];
29
30      char **rawdescriptors;
31
32      unsigned short bus_mA;
33      u8 portnum;
34      u8 level;
35      u8 devaddr;
36
37      unsigned can_submit:1;
38      unsigned persist_enabled:1;
39      unsigned have_langid:1;
40      unsigned authorized:1;
41      unsigned authenticated:1;
42      unsigned wusb:1;
43      unsigned lpm_capable:1;
44      unsigned usb2_hw_lpm_capable:1;
45      unsigned usb2_hw_lpm_besl_capable:1;
46      unsigned usb2_hw_lpm_enabled:1;
47      unsigned usb2_hw_lpm_allowed:1;
48      unsigned usb3_lpm_u1_enabled:1;
49      unsigned usb3_lpm_u2_enabled:1;
50      int string_langid;
51
52      /* static strings from the device */
53      char *product;
54      char *manufacturer;
55      char *serial;
56
57      struct list_head filelist;
58
59      int maxchild;
60
61      u32 quirks;
62      atomic_t urbnum;
63
64      unsigned long active_duration;
65
66  #ifdef CONFIG_PM
67      unsigned long connect_time;
68
69      unsigned do_remote_wakeup:1;
```

```
70      unsigned reset_resume:1;
71      unsigned port_is_suspended:1;
72  #endif
73      struct wusb_dev *wusb_dev;
74      int slot_id;
75      enum usb_device_removable removable;
76      struct usb2_lpm_parameters l1_params;
77      struct usb3_lpm_parameters u1_params;
78      struct usb3_lpm_parameters u2_params;
79      unsigned lpm_disable_count;
80
81      u16 hub_delay;
82      unsigned use_generic_driver:1;
83  };
```

Каждое USB–устройство должно соответствовать спецификации USB–IF [8], одним из требований которой является наличие идентификатора поставщика (`Vendor ID (VID)`) и идентификатор продукта (`Product ID (PID)`). Эти данные присутствуют в поле `descriptor` структуры `usb_device`. В листинге 1.8 представлена структура дескриптора `usb_device_descriptor`, описанная в `/include/uapi/linux/usb/ch9.h`.

Листинг 1.8 – Структура `usb_device_descriptor`

```
1  /* USB_DT_DEVICE: Device descriptor */
2  struct usb_device_descriptor {
3      __u8 bLength;
4      __u8 bDescriptorType;
5
6      __le16 bcdUSB;
7      __u8 bDeviceClass;
8      __u8 bDeviceSubClass;
9      __u8 bDeviceProtocol;
10     __u8 bMaxPacketSize0;
11     __le16 idVendor;
12     __le16 idProduct;
13     __le16 bcdDevice;
14     __u8 iManufacturer;
15     __u8 iProduct;
16     __u8 iSerialNumber;
17     __u8 bNumConfigurations;
18 } __attribute__ ((packed));
19
20 #define USB_DT_DEVICE_SIZE  18
```

1.3.2 Структура `usb_device_id`

При подключении USB–устройства к компьютеру, оно идентифицируется и идентификационная информация записывается в структуру `usb_device_id` [9].

Структура `usb_device_id` представлена в листинге 1.9.

Листинг 1.9 – Структура `usb_device_id`

```
struct usb_device_id {
    /* which fields to match against? */
    __u16  match_flags;

    /* Used for product specific matches; range is inclusive */
    __u16  idVendor;
    __u16  idProduct;
    __u16  bcdDevice_lo;
    __u16  bcdDevice_hi;

    /* Used for device class matches */
    __u8  bDeviceClass;
    __u8  bDeviceSubClass;
    __u8  bDeviceProtocol;

    /* Used for interface class matches */
    __u8  bInterfaceClass;
    __u8  bInterfaceSubClass;
    __u8  bInterfaceProtocol;

    /* Used for vendor-specific interface matches */
    __u8  bInterfaceNumber;

    /* not matched against */
    kernel_ulong_t driver_info
    __attribute__((aligned(sizeof(kernel_ulong_t))));
};
```

1.4 Взаимодействие с графическим ускорителем

Для взаимодействия с графическим ускорителем используется NVML [10]. NVML – это API для мониторинга и управления состояниями устройств NVIDIA [11]. NVML обеспечивает прямой доступ к запросам и командам утилиты nvidia-smi [12].

API NVML поставляется вместе с драйвером дисплея NVIDIA. SDK предоставляет соответствующие заголовки, библиотеки и примеры приложений.

Все версии NVML имеют обратную совместимость и предназначены для создания приложений сторонних разработчиков.

Для работы с графическим ускорителем, устройство идентифицируется при помощью структуры nvmlDevice_t. Эти данные можно получить путём вызова функции nvmlDeviceGetHandleByIndex, которая в свою очередь принимает аргумент index, являющийся индексом устройства в списке подключенных GPU устройств. По умолчанию индекс графического ускорителя равен нулю.

В листинге 1.10 представлен прототип функции nvmlDeviceGetHandleByIndex.

Листинг 1.10 – Прототип функции nvmlDeviceGetHandleByIndex

```
1 nvmlReturn_t nvmlDeviceGetHandleByIndex(unsigned int index, nvmlDevice_t* device)
```

Для получения PCI графического ускорителя при помощи полученной структуры nvmlDevice_t используется функция nvmlDeviceGetPciInfo.

В листинге 1.11 представлен прототип функции nvmlDeviceGetPciInfo.

Листинг 1.11 – Прототип функции nvmlDeviceGetPciInfo

```
1 nvmlReturn_t nvmlDeviceGetPciInfo(nvmlDevice_t device, nvmlPciInfo_t* pci)
```

Для управления состоянием графического ускорителя используется функция nvmlDeviceModifyDrainState. С помощью этой функции графический ускоритель переводится в режим ожидания. В режиме ожидания устройство не принимает никаких новых входящих запросов. С помощью этой же функции графический ускоритель переводится в активное состояние.

В листинге 1.12 представлен прототип функции `nvmlDeviceModifyDrainState`.

Листинг 1.12 – Прототип функции nvmlDeviceModifyDrainState

```
nvmlReturn_t nvmlDeviceModifyDrainState(nvmlPciInfo_t* pciInfo, nvmlEnableState_t
    newState)
```

У всех вышеупомянутых функций возвращаемый тип `nvmlReturn_t`. В листинге 1.13 перечислены возможные значения переменной этого типа.

Листинг 1.13 – Возможные значения переменной типа nvmlReturn_t

```
typedef enum nvmlReturn_enum
{
    // cppcheck-suppress *
    NVML_SUCCESS = 0, //!< The operation was successful
    NVML_ERROR_UNINITIALIZED = 1, //!< NVML was not first initialized with nvmlInit()
    NVML_ERROR_INVALID_ARGUMENT = 2, //!< A supplied argument is invalid
    NVML_ERROR_NOT_SUPPORTED = 3, //!< The requested operation is not available on
        target device
    NVML_ERROR_NO_PERMISSION = 4, //!< The current user does not have permission for
        operation
    NVML_ERROR_ALREADY_INITIALIZED = 5, //!< Deprecated: Multiple initializations are
        now allowed through ref counting
    NVML_ERROR_NOT_FOUND = 6, //!< A query to find an object was unsuccessful
    NVML_ERROR_INSUFFICIENT_SIZE = 7, //!< An input argument is not large enough
    NVML_ERROR_INSUFFICIENT_POWER = 8, //!< A device's external power cables are not
        properly attached
    NVML_ERROR_DRIVER_NOT_LOADED = 9, //!< NVIDIA driver is not loaded
    NVML_ERROR_TIMEOUT = 10, //!< User provided timeout passed
    NVML_ERROR_IRQ_ISSUE = 11, //!< NVIDIA Kernel detected an interrupt issue with a GPU
    NVML_ERROR_LIBRARY_NOT_FOUND = 12, //!< NVML Shared Library couldn't be found or
        loaded
    NVML_ERROR_FUNCTION_NOT_FOUND = 13, //!< Local version of NVML doesn't implement
        this function
    NVML_ERROR_CORRUPTED_INFOROM = 14, //!< infoROM is corrupted
    NVML_ERROR_GPU_IS_LOST = 15, //!< The GPU has fallen off the bus or has otherwise
        become inaccessible
    NVML_ERROR_RESET_REQUIRED = 16, //!< The GPU requires a reset before it can be used
        again
    NVML_ERROR_OPERATING_SYSTEM = 17, //!< The GPU control device has been blocked by
        the operating system/cgroups
    NVML_ERROR_LIB_RM_VERSION_MISMATCH = 18, //!< RM detects a driver/library version
        mismatch
    NVML_ERROR_IN_USE = 19, //!< An operation cannot be performed because the GPU is
        currently in use
    NVML_ERROR_MEMORY = 20, //!< Insufficient memory
    NVML_ERROR_NO_DATA = 21, //!< No data
```

```
26     NVML_ERROR_VGPU_ECC_NOT_SUPPORTED = 22, //!< The requested vgpu operation is not
           available on target device, because ECC is enabled
27     NVML_ERROR_INSUFFICIENT_RESOURCES = 23, //!< Ran out of critical resources, other
           than memory
28     NVML_ERROR_FREQ_NOT_SUPPORTED = 24, //!< Ran out of critical resources, other than
           memory
29     NVML_ERROR_UNKNOWN = 999 //!< An internal driver error occurred
30 } nvmlReturn_t;
```

1.5 Виртуальная файловая система /proc

Для организации доступа к разнообразным файловым системам в Unix используется промежуточный слой абстракции – виртуальная файловая система. Виртуальная файловая система организована как специальный интерфейс. Виртуальная файловая система объявляет API доступа к ней. Это API реализовывается драйверами конкретных файловых систем.

Виртуальная файловая система `/proc` – специальный интерфейс, с помощью которого можно мгновенно получить информацию о ядре в пространство пользователя.

В основном дереве каталога `/proc` в Linux, каждый каталог имеет числовое имя и соответствует процессу, с соответствующим PID. Файлы в этих каталогах соответствуют структуре `task_struct`. С помощью команды `cat /proc/1/cmdline`, можно узнать аргументы запуска процесса с идентификатором равным единице. В дереве `/proc/sys` отображаются внутренние переменные ядра.

Ядро предоставляет возможность добавить своё дерево в каталог `/proc`. Внутри ядра объявлена структура `struct proc_ops`. Эта структура содержит такие указатели, как указатель на функции чтения и записи в файл.

В листинге 1.14 представлено объявление данной структуры в ядре.

Листинг 1.14 – `Структура struct 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     ssize_t (*proc_read_iter)(struct kiocb *, struct iov_iter *);
6     ssize_t (*proc_write)(struct file *, const char __user *, size_t, loff_t *);
7     loff_t (*proc_lseek)(struct file *, loff_t, int);
```

```
8     int (*proc_release)(struct inode *, struct file *);
9     __poll_t (*proc_poll)(struct file *, struct poll_table_struct *);
10    long (*proc_ioctl)(struct file *, unsigned int, unsigned long);
11    #ifdef CONFIG_COMPAT
12    long (*proc_compat_ioctl)(struct file *, unsigned int, unsigned long);
13    #endif
14    int (*proc_mmap)(struct file *, struct vm_area_struct *);
15    unsigned long (*proc_get_unmapped_area)(struct file *, unsigned long, unsigned long,
          unsigned long, unsigned long);
16 } __randomize_layout;
```

С помощью вызова функций `proc_mkdir()` и `proc_create()` в модуле ядра можно зарегистрировать свои каталоги и файлы в `/proc`. Функции `copy_to_user()` и `copy_from_user()` реализуют передачу данных из пространства ядра в пространство пользователя и наоборот.

Таким образом, с помощью виртуальной файловой системы `/proc` можно получать и передавать данные между пространством ядра и пространствм пользовтеля .

Выводы

Были рассмотрены методы обработки событий, возникающих при взаимодействии с USB–устройствами. Среди рассмотренных методов был выбран механизм уведомителей, так как он позволяет привязать свой обработчик события, а также реализован на уровне ядра Linux. Были рассмотрены структуры и функции ядра для работы с уведомителями, способ взаимодействия с графическим ускорителем, а также особенности разработки загружаемых модулей ядра. Для передачи данных из пространства ядра в пространство пользователя будет использоваться файловая система `proc`.

2 Конструкторский раздел

2.1 IDEF0 последовательность преобразований

На рисунках 2.1 - 2.3 представлена IDEF0 последовательность преобразований.

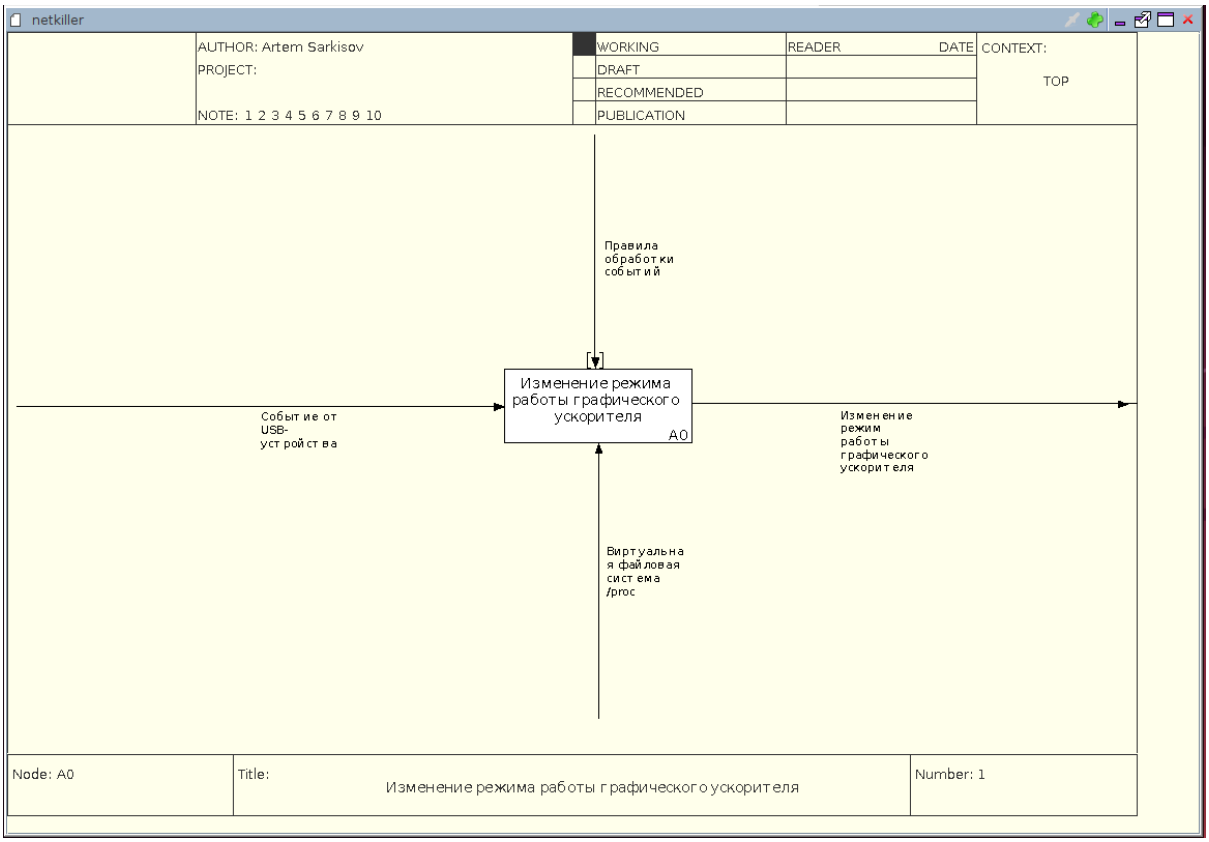

Рисунок 2.1 – Нулевой уровень преобразований

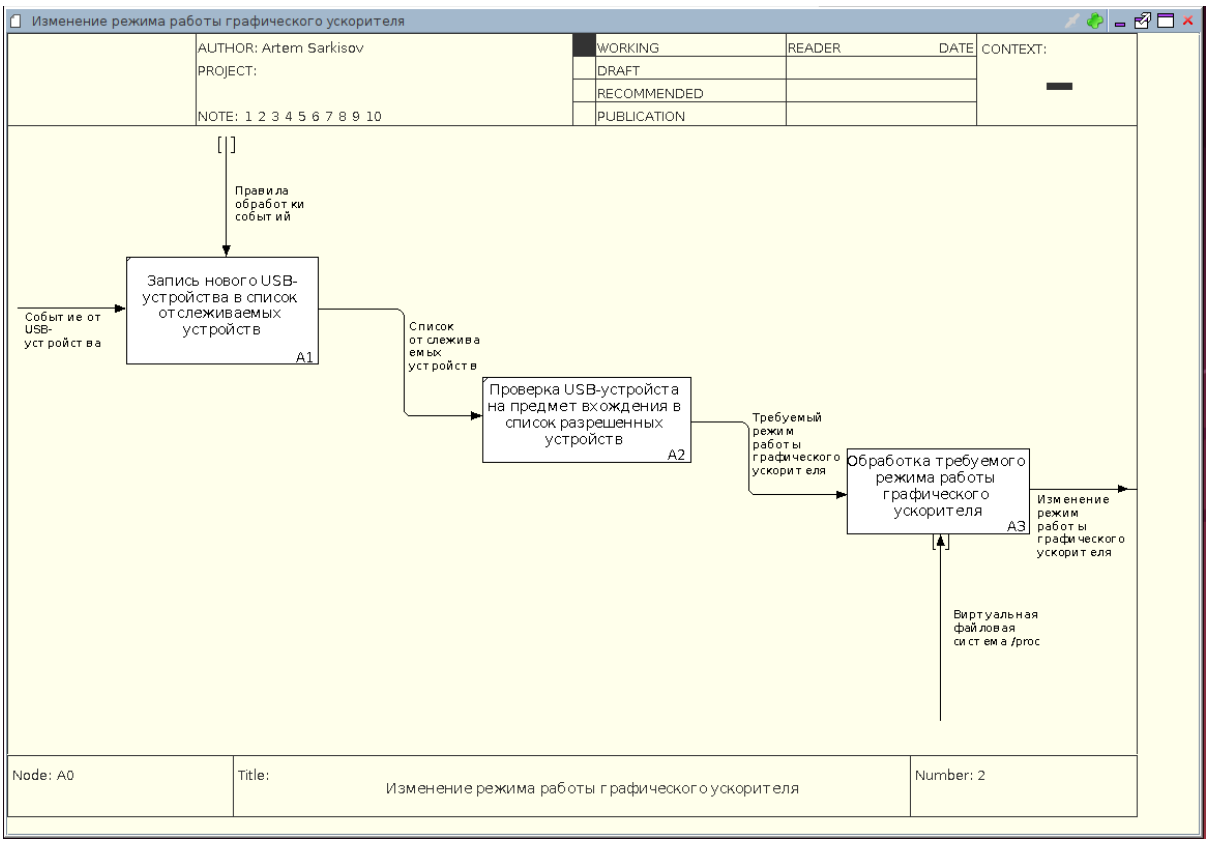

Рисунок 2.2 – Первый уровень преобразований

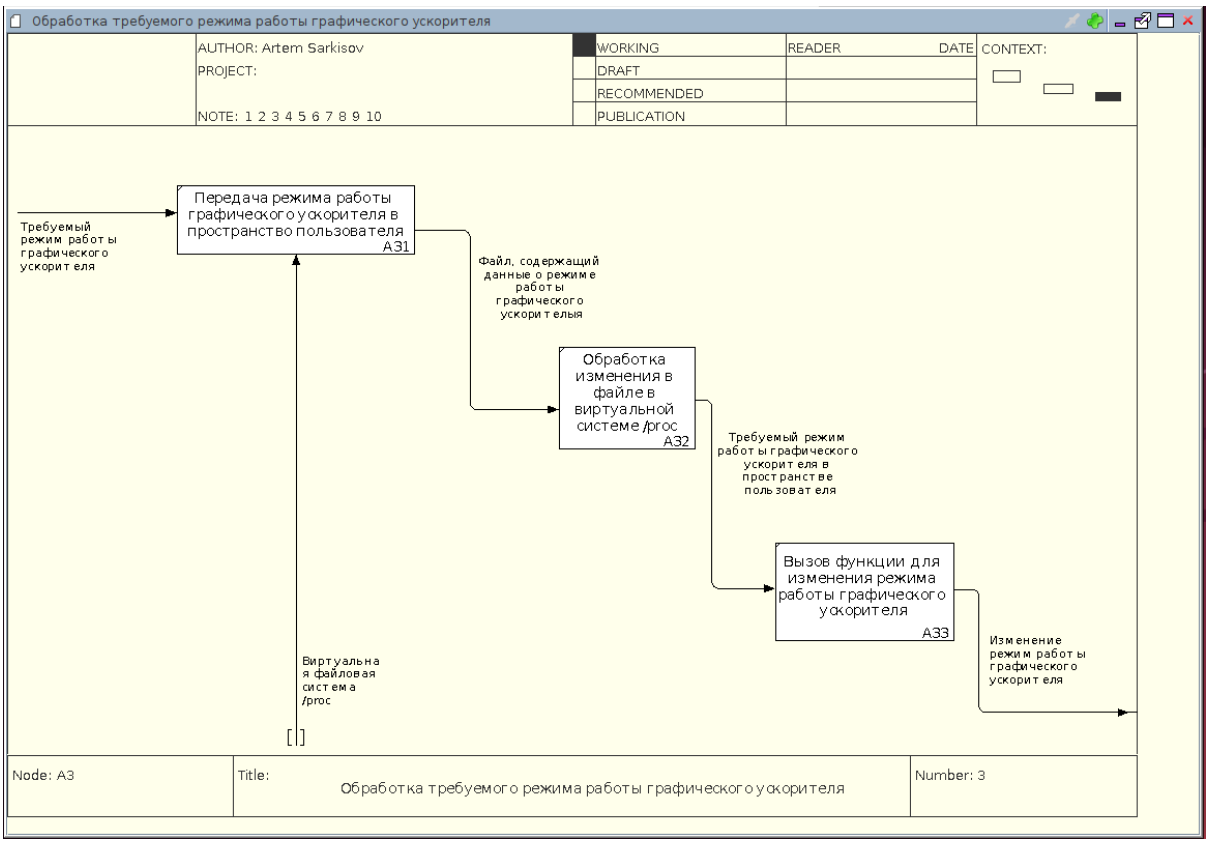

Рисунок 2.3 – Второй уровень преобразований

2.2 Алгоритм отслеживания событий

Для отслеживания событий подключения и отключения устройств в модуле ядра размещен соответствующий уведомитель, который будет зарегистрирован при загрузке модуля и удалён при его удалении.

В листинге 2.1 представлена функция обработки событий.

Листинг 2.1 – Обработка событий

```
// Handler for event's notifier.
static int notify(struct notifier_block *self, unsigned long action, void *dev)
{
    // Events, which our notifier react.
    switch (action)
    {
    case USB_DEVICE_ADD:
        usb_dev_insert(dev);
        break;
    case USB_DEVICE_REMOVE:
        usb_dev_remove(dev);
        break;
    default:
        break;
    }

    return 0;
}

// React on different notifies.
static struct notifier_block usb_notify = {
    .notifier_call = notify,
};
```

Для каждого события есть отдельный обработчик.

2.3 Алгоритм работы обработчика событий

На рисунке 2.4 представлен алгоритм работы обработчика событий .

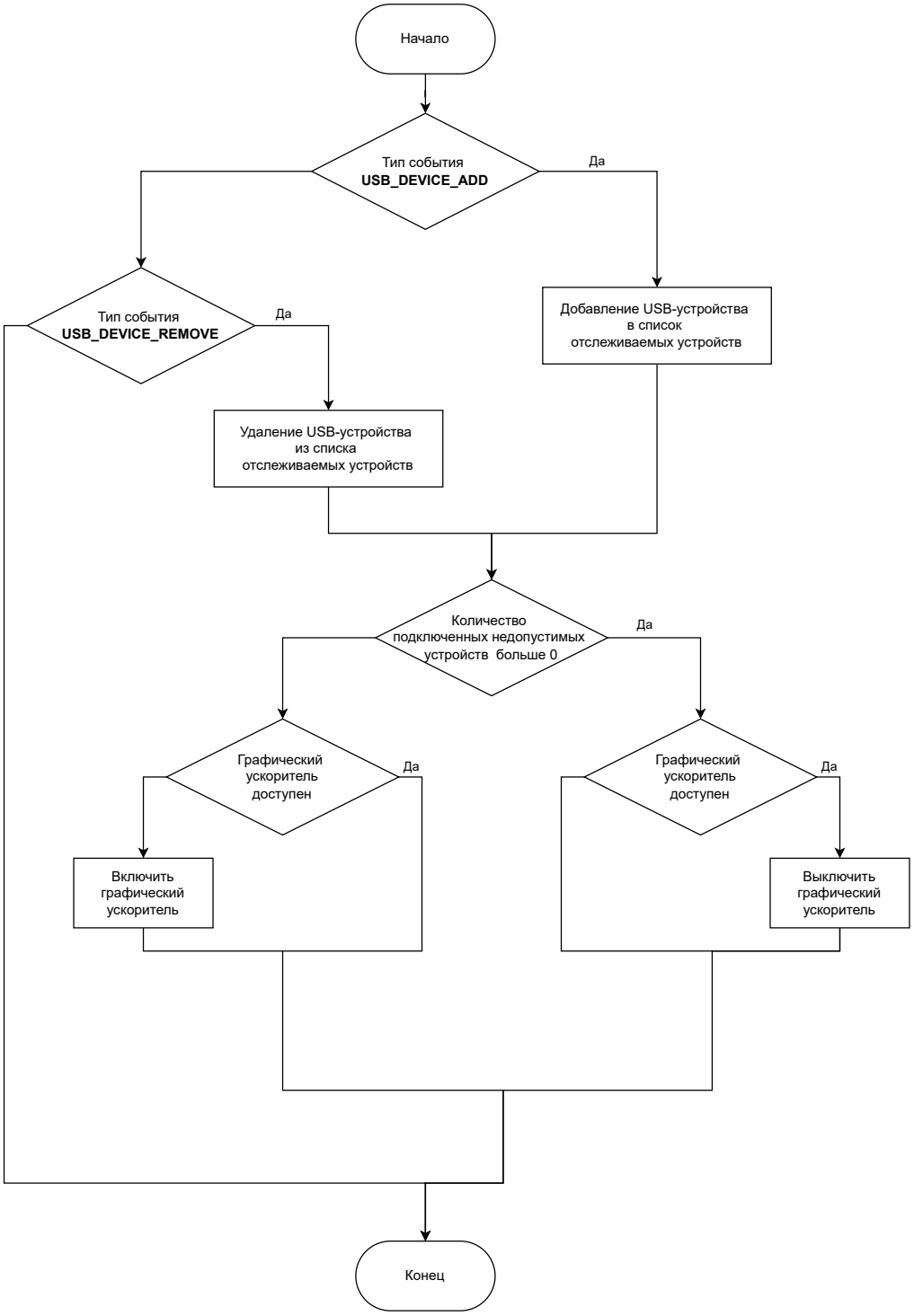

Рисунок 2.4 – Алгоритм обработчика событий

2.4 Структура программного обеспечения

В состав разрабатываемого программного обеспечения входит один загружаемый модуль ядра, который отслеживает подключенные USB–устройства и программно отключает сетевые устройства при наличии неразрешённых устройства. Неразрешённым устройством считается устройство, которое не идентифицируется в соответствии со списком допустимых устройств модуля. Список допустимых устройств задается в исходном коде модуля.

3 Технологический раздел

3.1 Выбор языка и среды программирования

Разработанный модуль ядра написан на языке программирования `C` [15]. Выбор языка программирования `C` основан на том, что исходный код ядра Linux, все его модули и драйверы написаны на данном языке.

В качестве компилятора выбран `gcc` [16].

В качестве среды разработки выбрана среда `Visual Studio Code` [17].

3.2 Хранение информации об отслеживаемых устройствах

Для хранения информации об отслеживаемых устройствах объявлена структура `int_usb_device`, которая хранит в себе идентификационные данные устройства (`PID, VID`) и указатель на элемент списка.

Структура `int_usb_device` представлена в листинге 3.1.

Листинг 3.1 – Структура `int_usb_device`

```
// Wrapper for usb_device_id with added list_head field to track devices.
typedef struct int_usb_device
{
    struct usb_device_id dev_id;
    struct list_head list_node;
} int_usb_device_t;
```

При подключении или удалении устройства, создается экземпляр данной структуры и помещается в список отслеживаемых устройств.

В листинге 3.2 представлены функции для работы со списком отслеживаемых устройств.

Листинг 3.2 – Функции для работы со списком отслеживаемых устройств

```
// Add connected device to list of tracked devices.
static void add_int_usb_dev(struct usb_device *dev)
{
    int_usb_device_t *new_usb_device = (int_usb_device_t
        *)kmalloc(sizeof(int_usb_device_t), GFP_KERNEL);
    struct usb_device_id new_id = {USB_DEVICE(dev->descriptor.idVendor,
        dev->descriptor.idProduct)};
    new_usb_device->dev_id = new_id;
    list_add_tail(&new_usb_device->list_node, &connected_devices);
}

// Delete device from list of tracked devices.
static void delete_int_usb_dev(struct usb_device *dev)
{
    int_usb_device_t *device, *temp;
    list_for_each_entry_safe(device, temp, &connected_devices, list_node)
    {
        if (is_dev_matched(dev, &device->dev_id))
        {
            list_del(&device->list_node);
            kfree(device);
        }
    }
}
```

3.3 Идентификация устройства как разрешённого

Для проверки устройства необходимо проверить его идентификационные данные с данными разрешённых устройств. В листинге 3.3 представлены объявление списка разрешённых устройств и функции для идентификации устройства.

Листинг 3.3 – Функции для идентификации устройств

```
struct usb_device_id allowed_devs[] = {
    {USB_DEVICE(0x13fe, 0x4301)}, // 0x13fe, 0x4300
};
// Match device with device id.
static bool is_dev_matched(struct usb_device *dev, const struct usb_device_id *dev_id)
{
    // Check idVendor and idProduct, which are used.
    if (dev_id->idVendor != dev->descriptor.idVendor || dev_id->idProduct !=
        dev->descriptor.idProduct)
    {
        return false;
    }

    return true;
}

// Match device id with device id.
static bool is_dev_id_matched(struct usb_device_id *new_dev_id, const struct
    usb_device_id *dev_id)
{
    // Check idVendor and idProduct, which are used.
    if (dev_id->idVendor != new_dev_id->idVendor || dev_id->idProduct !=
        new_dev_id->idProduct)
    {
        return false;
    }

    return true;
}

// Check if device is in allowed devices list.
static bool *is_dev_allowed(struct usb_device_id *dev)
{
    unsigned long allowed_devs_len = sizeof(allowed_devs) / sizeof(struct usb_device_id);

    int i;
    for (i = 0; i < allowed_devs_len; i++)
    {
        if (is_dev_id_matched(dev, &allowed_devs[i]))
        {
            return true;
        }
    }

    return false;
}

```

```
45 // Check if changed device is acknowledged.
46 static int count_not_acked_devs(void)
47 {
48     int_usb_device_t *temp;
49     int count = 0;
50 
51     list_for_each_entry(temp, &connected_devices, list_node)
52     {
53         if (!is_dev_allowed(&temp->dev_id))
54         {
55             count++;
56         }
57     }
58 
59     return count;
60 }
```

3.4 Обработка событий USB–устройства

При подключении устройство добавляется в список отслеживаемых устройств. После этого происходит проверка на наличие среди отслеживаемых устройств неразрешённых, и, в случае если такие были найдены, происходит изменение режима работы графического ускорителя.

В листинге 3.4 представлен обработчик подключения USB–устройства.

Листинг 3.4 – Обработчик подключения USB–устройства

```
1  // Handler for USB insertion.
2  static void usb_dev_insert(struct usb_device *dev)
3  {
4      printk(KERN_INFO "gpufreezer: device connected with PID '%d' and VID '%d'\n",
5             dev->descriptor.idProduct, dev->descriptor.idVendor);
6      add_int_usb_dev(dev);
7      int not_acked_devs = count_not_acked_devs();
8  
9      if (!not_acked_devs)
10     {
11         printk(KERN_INFO "gpufreezer: there are no any forbidden devices connected, 
               skipping GPU freezing\n");
12     }
13     else
14     {
15         printk(KERN_INFO "gpufreezer: there are %d not allowed devices connected, 
               freezing GPU\n", not_acked_devs);
```

```
16          if (gpu_state)
17          {
18              gpu_state = false;
19          }
20      }
21  }
```

При отключении устройство удаляется из списка отслеживаемых устройств. После этого происходит проверка на наличие среди отслеживаемых устройств неразрешённых, и, в случае если такие не были найдены, происходит происходит изменение режима работы графического ускорителя.

В листинге 3.5 представлен обработчик отключения USB–устройства.

Листинг 3.5 – Обработчик отключения USB–устройства

```
1  // Handler for USB removal.
2  static void usb_dev_remove(struct usb_device *dev)
3  {
4      printk(KERN_INFO "gpufreezer: device disconnected with PID '%d' and VID '%d'\n",
5             dev->descriptor.idProduct, dev->descriptor.idVendor);
6      delete_int_usb_dev(dev);
7      int not_acked_devs = count_not_acked_devs();
8
9      if (not_acked_devs)
10     {
11         printk(KERN_INFO "gpufreezer: there are %d not allowed devices connected, nothing
               to do\n", not_acked_devs);
12     }
13     else
14     {
15         if (!gpu_state)
16         {
17             gpu_state = true;
18         }
19     }
20 }
```

3.5 Регистрация уведомителя для USB–устройств

В листинге 3.6 представлено объявление уведомителя и его функции–обработчика.

Листинг 3.6 – Уведомитель для USB–устройств

```
// Handler for event's notifier.
static int notify(struct notifier_block *self, unsigned long action, void *dev)
{
    // Events, which our notifier react.
    switch (action)
    {
    case USB_DEVICE_ADD:
        usb_dev_insert(dev);
        break;
    case USB_DEVICE_REMOVE:
        usb_dev_remove(dev);
        break;
    default:
        break;
    }

    return 0;
}

// React on different notifies.
static struct notifier_block usb_notify = {
    .notifier_call = notify,
};
```

В листинге 3.7 представлены регистрация и дерегистрация уведомителя при загрузке и удалении модуля ядра соответственно.

Листинг 3.7 – Регистрация и дерегистрация уведомителя

```
// Module init function.
static int __init gpufreezer_init(void)
{
    if ((proc_root = proc_mkdir("gpufreezer_log", NULL)) == NULL)
    {
        return -1;
    }

    if ((proc_gpu_status_file = proc_create("gpu_state", 066, proc_root, &usb_ops)) ==
        NULL)
```

```
    {
        return -1;
    }

    usb_register_notify(&usb_notify);
    printk(KERN_INFO "gpufreezer: module loaded\n");
    return 0;
}

// Module exit function.
static void __exit gpufreezer_exit(void)
{
    if (proc_gpu_status_file != NULL)
    {
        remove_proc_entry("gpu_state", proc_root);
    }

    if (proc_root != NULL)
    {
        remove_proc_entry("gpufreezer_log", NULL);
    }

    usb_unregister_notify(&usb_notify);
    printk(KERN_INFO "gpufreezer: module unloaded\n");
}
```

3.6 Запись данных в виртуальную файловую систему /proc

В листинге 3.8 представлена запись данных в виртуальную файловую систему /proc. Данные представляют собой состояние работы, в которое нужно перевести графический ускоритель

Листинг 3.8 – Запись данных в виртуальную файловую систему /proc

```
void show_int_message(struct seq_file *m, const char *const f, const long num)
{
    char tmp[256];
    int len;

    len = snprintf(tmp, 256, f, num);
    seq_write(m, tmp, len);
}

```

```
void print_gpu_state(struct seq_file *m)
{
    show_int_message(m, "%d", gpu_state);
}

static int show_gpu_state(struct seq_file *m, void *v)
{
    print_gpu_state(m);
    return 0;
}

static int proc_gpu_state_open(struct inode *sp_inode, struct file *sp_file)
{
    return single_open(sp_file, show_gpu_state, NULL);
}

static int proc_release(struct inode *sp_node, struct file *sp_file)
{
    return 0;
}

// Override open and close file functions.
static const struct proc_ops usb_ops = {
    proc_read : seq_read,
    proc_open : proc_gpu_state_open,
    proc_release : proc_release,
};
```

3.7 Модификация режима работы графического ускорителя

В листинге 3.9 представлено получение данных устройства для дальнейшего взаимодействия с ним.

Листинг 3.9 – Получение данных подключенного графического ускорителя

```
   // Initialize NVML
   result = nvmlInit();
   if (result != NVML_SUCCESS)
   {
      return 1;
   }

```

```
   // Retrieve information about the device by GPU index
   result = nvmlDeviceGetHandleByIndex_v2(0, &device);
   if (result != NVML_SUCCESS)
   {
      nvmlShutdown();
      return 1;
   }

   // Retrieve information about the first GPU on the system
   result = nvmlDeviceGetPciInfo(device, &pciInfo);
   if (result != NVML_SUCCESS)
   {
      nvmlShutdown();
      return 1;
   }
```

В листинге 3.10 представлен код, предназначенный для обработки изменения в файле `gpu_state` в виртуальной файловой системе `/proc`.

Листинг 3.10 – Обработка изменения в файле `gpu_state` в виртуальной системе `/proc`

```
while (1)
{
   char *gpu_status_filename = "/proc/gpufreezer_log/gpu_state";
   FILE *gpu_status_fp = fopen(gpu_status_filename, "r");
   if (gpu_status_fp == NULL)
   {
      return 1;
   }

   char gpu_status_cur;
   gpu_status_cur = fgetc(gpu_status_fp);

   if (gpu_status_cur != gpu_status)
   {
      switch (gpu_status_cur)
      {
      case GPU_TO_ACTIVATE:
         nvmlDeviceModifyDrainState(&pciInfo, NVML_FEATURE_DISABLED);
         break;
      case GPU_TO_DEACTIVATE:
         nvmlDeviceModifyDrainState(&pciInfo, NVML_FEATURE_ENABLED);
         break;
      }
      gpu_status = gpu_status_cur;
   }

   fclose(gpu_status_fp);
```

```
28      sleep(2);
29    }
```

3.8 Примеры работы разработанного ПО

На рисунках 3.1 — 3.5 представлены примеры работы разработанного модуля ядра.

```
tema@tema-GS65-Stealth-Thin-8RE:~$ nvidia-smi
Mon Jan 30 16:28:41 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 510.108.03   Driver Version: 510.108.03   CUDA Version: 11.6     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  Off  | 00000000:01:00.0 Off |                  N/A |
| N/A   37C    P0    23W /  N/A |    517MiB /  6144MiB |      2%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A      1206      G   /usr/lib/xorg/Xorg                174MiB |
|    0   N/A  N/A      1559      G   /usr/bin/gnome-shell              199MiB |
|    0   N/A  N/A      2725      G   ...5/usr/lib/firefox/firefox       91MiB |
|    0   N/A  N/A      5360      G   ...RendererForSitePerProcess       48MiB |
+-----------------------------------------------------------------------------+
```

Рисунок 3.1 – Работа графического ускорителя до подключения USB–устройства

```
[ 1961.382268] gpufreezer: module loaded
[ 1998.502777] usb 1-4: new high-speed USB device number 6 using xhci_hcd
[ 1998.821587] usb 1-4: New USB device found, idVendor=13fe, idProduct=4300, bcdDevice= 1.00
[ 1998.821601] usb 1-4: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 1998.821608] usb 1-4: Product: USB DISK 2.0
[ 1998.821613] usb 1-4: Manufacturer:
[ 1998.821618] usb 1-4: SerialNumber: 07211A0F9D155513
[ 1998.823886] gpufreezer: device connected with PID '17152' and VID '5118'
[ 1998.823894] gpufreezer: there are 1 not allowed devices connected, freezing GPU
[ 1998.824684] gpufreezer: GPU is frozen
```

Рисунок 3.2 – Пример подключения неразрешённого устройства

```
tema@tema-GS65-Stealth-Thin-8RE:~$ nvidia-smi
No devices were found
```

Рисунок 3.3 – Проверка работы графического ускорителя после подключения неразрешённого устройства

```
[ 2277.228648] usb 1-4: USB disconnect, device number 6
[ 2277.305257] gpufreezer: device disconnected with PID '17152' and VID '5118'
[ 2277.305260] gpufreezer: every not allowed devices are disconnected, bringing GPU back
[ 2277.305412] gpufreezer: GPU is available now
```

Рисунок 3.4 – Пример отключения неразрешённого устройства

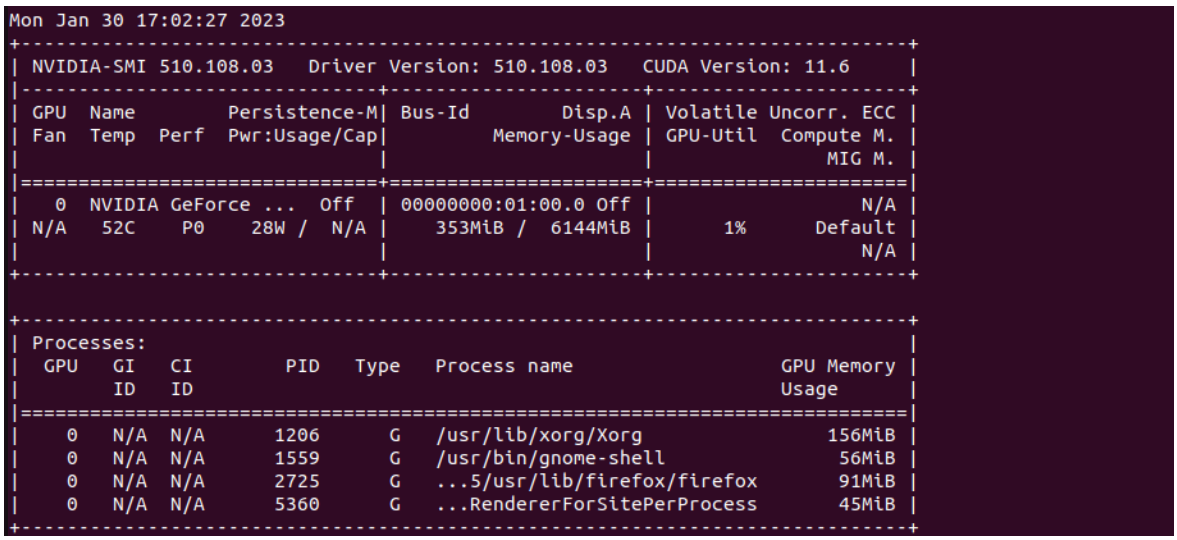

Рисунок 3.5 – Проверка работы графического ускорителя после отключения неразрешённого устройства

Заключение

В процессе выполнения курсовой работы по операционный системам был разработан загружаемый модуль ядра Linux для отключения графического ускорителя при подключении USB–устройства.

Проанализированы методы обработки событий, возникающих при взаимодействии с USB–устройствами.

Изучены структуры и функции ядра, которые предоставляют информацию о USB–устройствах, механизмы для обработки событий USB–устройств.

Разработан алгоритм отключения графического ускорителя при подключении USB–устройства.

Реализовано и протестировано программное обеспечение, реализующее разработанный алгоритм. Программное обеспечение полностью удовлетворяет техническому заданию.

Литература

[1] Juice Jacking: Security Issues and Improvements in USB Technology / Debabrata Singh, Anil Kumar Biswal, Debabrata Samanta [и др.] // Sustainability. 2022. 01. Т. 14.

[2] Notification Chains in Linux Kernel [Электронный ресурс]. Режим доступа: https://0xax.gitbooks.io/linux-insides/content/Concepts/linux-cpu-4.html (дата обращения: 15.02.2022).

[3] notifier.h - include/linux/notifier.h - Linux source code (v5.13) - Bootlin [Электронный ресурс]. Режим доступа: https://elixir.bootlin.com/linux/v5.13/source/include/linux/notifier.h#L54 (дата обращения: 15.02.2022).

[4] usbmon — The Linux Kernel documentation [Электронный ресурс]. Режим доступа: https://www.kernel.org/doc/html/latest/usb/usbmon.html (дата обращения: 16.02.2022).

[5] udevadm(8) - Linux manual page [Электронный ресурс]. Режим доступа: https://man7.org/linux/man-pages/man8/udevadm.8.html (дата обращения: 16.02.2022).

[6] libudev [Электронный ресурс]. Режим доступа: https://www.freedesktop.org/software/systemd/man/libudev.html (дата обращения: 16.02.2022).

[7] usb.h - include/linux/usb.h - Linux source code (v5.13) - Bootlin [Электронный ресурс]. Режим доступа: https://elixir.bootlin.com/linux/v5.13/source/include/linux/usb.h#L632 (дата обращения: 15.02.2022).

[8] Document Library | USB-IF [Электронный ресурс]. Режим доступа: https://www.usb.org/documents?search=&type%5B0%5D=55&items_per_page=50 (дата обращения: 15.02.2022).

[9] mod_devicetable.h - include/linux/mod_devicetable.h - Linux source code (v5.13) - Bootlin [Электронный ресурс]. Режим доступа:

https://elixir.bootlin.com/linux/v5.13/source/include/linux/mod_devicetable.h#L121 (дата обращения: 15.02.2022).

[10] NVIDIA Management Library (NVML) [Электронный ресурс]. Режим доступа: https://developer.nvidia.com/nvidia-management-library-nvml (дата обращения: 15.02.2022).

[11] NVIDIA [Электронный ресурс]. Режим доступа: https://www.nvidia.com/en-us/ (дата обращения: 15.02.2022).

[12] NVIDIA System Management Interface [Электронный ресурс]. Режим доступа: https://developer.nvidia.com/nvidia-system-management-interface (дата обращения: 15.02.2022).

[13] Invoking user-space applications from the kernel – IBM Developer [Электронный ресурс]. Режим доступа: https://developer.ibm.com/articles/l-user-space-apps/ (дата обращения: 15.02.2022).

[14] umh.h - include/linux/umh.h - Linux source code (v5.13) - Bootlin [Электронный ресурс]. Режим доступа: https://elixir.bootlin.com/linux/v5.13/source/include/linux/umh.h#L42 (дата обращения: 15.02.2022).

[15] C99 standard note [Электронный ресурс]. Режим доступа: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1256.pdf (дата обращения: 15.02.2022).

[16] GCC, the GNU Compiler Collection [Электронный ресурс]. Режим доступа: https://gcc.gnu.org/ (дата обращения: 15.02.2022).

[17] Visual Studio Code - Code Editing. Redefined [Электронный ресурс]. Режим доступа: https://code.visualstudio.com (дата обращения: 16.02.2022).

ПРИЛОЖЕНИЕ А

Листинг 3.11 – Исходный код программы

```c
#include <linux/module.h>
#include <linux/slab.h>
#include <linux/usb.h>
#include <linux/seq_file.h>
#include <linux/proc_fs.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Artem Sarkisov");
MODULE_VERSION("1.0");

static struct proc_dir_entry *proc_root = NULL;

static struct proc_dir_entry *proc_gpu_status_file = NULL;

// Wrapper for usb_device_id with added list_head field to track devices.
typedef struct int_usb_device
{
    struct usb_device_id dev_id;
    struct list_head list_node;
} int_usb_device_t;

bool gpu_state = true;

struct usb_device_id allowed_devs[] = {
    {USB_DEVICE(0x13fe, 0x4301)}, // 0x13fe, 0x4300
};

// Declare and init the head node of the linked list.
LIST_HEAD(connected_devices);

// Match device with device id.
static bool is_dev_matched(struct usb_device *dev, const struct usb_device_id *dev_id)
{
    // Check idVendor and idProduct, which are used.
    if (dev_id->idVendor != dev->descriptor.idVendor || dev_id->idProduct !=
        dev->descriptor.idProduct)
    {
        return false;
    }

    return true;
}

// Match device id with device id.
```

```
44 static bool is_dev_id_matched(struct usb_device_id *new_dev_id, const struct
       usb_device_id *dev_id)
45 {
46     // Check idVendor and idProduct, which are used.
47     if (dev_id->idVendor != new_dev_id->idVendor || dev_id->idProduct !=
           new_dev_id->idProduct)
48     {
49         return false;
50     }
51
52     return true;
53 }
54
55 // Check if device is in allowed devices list.
56 static bool *is_dev_allowed(struct usb_device_id *dev)
57 {
58     unsigned long allowed_devs_len = sizeof(allowed_devs) / sizeof(struct usb_device_id);
59
60     int i;
61     for (i = 0; i < allowed_devs_len; i++)
62     {
63         if (is_dev_id_matched(dev, &allowed_devs[i]))
64         {
65             return true;
66         }
67     }
68
69     return false;
70 }
71
72 // Check if changed device is acknowledged.
73 static int count_not_acked_devs(void)
74 {
75     int_usb_device_t *temp;
76     int count = 0;
77
78     list_for_each_entry(temp, &connected_devices, list_node)
79     {
80         if (!is_dev_allowed(&temp->dev_id))
81         {
82             count++;
83         }
84     }
85
86     return count;
87 }
88
89 // Add connected device to list of tracked devices.
```

```
90  static void add_int_usb_dev(struct usb_device *dev)
91  {
92      int_usb_device_t *new_usb_device = (int_usb_device_t
            *)kmalloc(sizeof(int_usb_device_t), GFP_KERNEL);
93      struct usb_device_id new_id = {USB_DEVICE(dev->descriptor.idVendor,
            dev->descriptor.idProduct)};
94      new_usb_device->dev_id = new_id;
95      list_add_tail(&new_usb_device->list_node, &connected_devices);
96  }
97
98  // Delete device from list of tracked devices.
99  static void delete_int_usb_dev(struct usb_device *dev)
100 {
101     int_usb_device_t *device, *temp;
102     list_for_each_entry_safe(device, temp, &connected_devices, list_node)
103     {
104         if (is_dev_matched(dev, &device->dev_id))
105         {
106             list_del(&device->list_node);
107             kfree(device);
108         }
109     }
110 }
111
112 // Handler for USB insertion.
113 static void usb_dev_insert(struct usb_device *dev)
114 {
115     printk(KERN_INFO "gpufreezer:␣device␣connected␣with␣PID␣'%d'␣and␣VID␣'%d'\n",
116           dev->descriptor.idProduct, dev->descriptor.idVendor);
117     add_int_usb_dev(dev);
118     int not_acked_devs = count_not_acked_devs();
119
120     if (!not_acked_devs)
121     {
122         printk(KERN_INFO "gpufreezer:␣there␣are␣no␣any␣forbidden␣devices␣connected,␣
              skipping␣GPU␣freezing\n");
123     }
124     else
125     {
126         printk(KERN_INFO "gpufreezer:␣there␣are␣%d␣not␣allowed␣devices␣connected,␣
              freezing␣GPU\n", not_acked_devs);
127         if (gpu_state)
128         {
129             gpu_state = false;
130         }
131     }
132 }
133
```

```
134 // Handler for USB removal.
135 static void usb_dev_remove(struct usb_device *dev)
136 {
137     printk(KERN_INFO "gpufreezer: device disconnected with PID '%d' and VID '%d'\n",
138            dev->descriptor.idProduct, dev->descriptor.idVendor);
139     delete_int_usb_dev(dev);
140     int not_acked_devs = count_not_acked_devs();
141 
142     if (not_acked_devs)
143     {
144         printk(KERN_INFO "gpufreezer: there are %d not allowed devices connected, nothing
              to do\n", not_acked_devs);
145     }
146     else
147     {
148         if (!gpu_state)
149         {
150             gpu_state = true;
151         }
152     }
153 }
154 
155 void show_int_message(struct seq_file *m, const char *const f, const long num)
156 {
157     char tmp[256];
158     int len;
159 
160     len = snprintf(tmp, 256, f, num);
161     seq_write(m, tmp, len);
162 }
163 
164 void print_gpu_state(struct seq_file *m)
165 {
166     show_int_message(m, "%d", gpu_state);
167 }
168 
169 static int show_gpu_state(struct seq_file *m, void *v)
170 {
171     print_gpu_state(m);
172     return 0;
173 }
174 
175 static int proc_gpu_state_open(struct inode *sp_inode, struct file *sp_file)
176 {
177     return single_open(sp_file, show_gpu_state, NULL);
178 }
179 
180 static int proc_release(struct inode *sp_node, struct file *sp_file)
```

```
{
    return 0;
}

// Override open and close file functions.
static const struct proc_ops usb_ops = {
    proc_read : seq_read,
    proc_open : proc_gpu_state_open,
    proc_release : proc_release,
};

// Handler for event's notifier.
static int notify(struct notifier_block *self, unsigned long action, void *dev)
{
    // Events, which our notifier react.
    switch (action)
    {
    case USB_DEVICE_ADD:
        usb_dev_insert(dev);
        break;
    case USB_DEVICE_REMOVE:
        usb_dev_remove(dev);
        break;
    default:
        break;
    }

    return 0;
}

// React on different notifies.
static struct notifier_block usb_notify = {
    .notifier_call = notify,
};

// Module init function.
static int __init gpufreezer_init(void)
{
    if ((proc_root = proc_mkdir("gpufreezer_log", NULL)) == NULL)
    {
        return -1;
    }

    if ((proc_gpu_status_file = proc_create("gpu_state", 066, proc_root, &usb_ops)) ==
        NULL)
    {
        return -1;
    }
```

```
228
229     usb_register_notify(&usb_notify);
230     printk(KERN_INFO "gpufreezer: module loaded\n");
231     return 0;
232 }
233
234 // Module exit function.
235 static void __exit gpufreezer_exit(void)
236 {
237     if (proc_gpu_status_file != NULL)
238     {
239         remove_proc_entry("gpu_state", proc_root);
240     }
241
242     if (proc_root != NULL)
243     {
244         remove_proc_entry("gpufreezer_log", NULL);
245     }
246
247     usb_unregister_notify(&usb_notify);
248     printk(KERN_INFO "gpufreezer: module unloaded\n");
249 }
250
251 module_init(gpufreezer_init);
252 module_exit(gpufreezer_exit);
```

Листинг 3.12 – Исходный код программы

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <nvml.h>
4
5  #define GPU_TO_ACTIVATE '1'
6  #define GPU_TO_DEACTIVATE '0'
7
8  nvmlDevice_t device;
9  nvmlPciInfo_t pciInfo;
10
11 // Modify the drain state of the GPU
12 static void modify_drain_state(nvmlEnableState_t newState)
13 {
14    nvmlDeviceModifyDrainState(&pciInfo, newState);
15 }
16
17 int main()
18 {
19    nvmlReturn_t result;
20    char gpu_status;
21
```

```
22    // Initialize NVML
23    result = nvmlInit();
24    if (result != NVML_SUCCESS)
25    {
26       return 1;
27    }
28
29    // Retrieve information about the device by GPU index
30    result = nvmlDeviceGetHandleByIndex_v2(0, &device);
31    if (result != NVML_SUCCESS)
32    {
33       nvmlShutdown();
34       return 1;
35    }
36
37    // Retrieve information about the first GPU on the system
38    result = nvmlDeviceGetPciInfo(device, &pciInfo);
39    if (result != NVML_SUCCESS)
40    {
41       nvmlShutdown();
42       return 1;
43    }
44
45    // Observer for handling gpu status file
46    while (1)
47    {
48       char *gpu_status_filename = "/proc/gpufreezer_log/gpu_state";
49       FILE *gpu_status_fp = fopen(gpu_status_filename, "r");
50       if (gpu_status_fp == NULL)
51       {
52          return 1;
53       }
54
55       char gpu_status_cur;
56       gpu_status_cur = fgetc(gpu_status_fp);
57
58       if (gpu_status_cur != gpu_status)
59       {
60          switch (gpu_status_cur)
61          {
62          case GPU_TO_ACTIVATE:
63             nvmlDeviceModifyDrainState(&pciInfo, NVML_FEATURE_DISABLED);
64             break;
65          case GPU_TO_DEACTIVATE:
66             nvmlDeviceModifyDrainState(&pciInfo, NVML_FEATURE_ENABLED);
67             break;
68          }
69          gpu_status = gpu_status_cur;
```

```
70        }
71
72        fclose(gpu_status_fp);
73        sleep(2);
74    }
75
76    // Clean up NVML
77    nvmlShutdown();
78    return 0;
79 }
```