



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, искусственный интеллект и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

---

# РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

## *К КУРСОВОЙ РАБОТЕ*

### *НА ТЕМУ:*

*«Регулировка уровня громкости клавиатурой с  
возможностью использования заданной шкалы  
настроек»*

Студент ИУ7-76Б  
(Группа)

\_\_\_\_\_  
(Подпись, дата)

В. В. Леонов  
(И. О. Фамилия)

Руководитель курсовой работы

\_\_\_\_\_  
(Подпись, дата)

Н. Ю. Рязанова  
(И. О. Фамилия)

*2023 г.*

# СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b>	<b>4</b>
<b>1 Аналитический раздел</b>	<b>5</b>
1.1 Постановка задачи . . . . .	5
1.2 Анализ способов обработки прерываний . . . . .	5
1.3 Анализ аудио подсистем . . . . .	8
1.3.1 Open Sound System . . . . .	8
1.3.2 Advanced Linux Sound Architecture . . . . .	9
1.3.3 JACK Audio Connection Kit . . . . .	10
1.3.4 Сравнение рассмотренных решений . . . . .	11
<b>2 Конструкторский раздел</b>	<b>13</b>
2.1 IDEF0–диаграммы . . . . .	13
2.2 Разработка алгоритмов . . . . .	14
2.3 Структура ПО . . . . .	17
<b>3 Технологический раздел</b>	<b>18</b>
3.1 Выбор языка и среды программирования . . . . .	18
3.2 Точки входа в модуль . . . . .	18
3.3 Обработка прерываний клавиатуры . . . . .	19
3.4 Функция изменения громкости . . . . .	21
3.5 Сборка разработанного модуля . . . . .	23
<b>4 Исследовательский раздел</b>	<b>24</b>
4.1 Демонстрация последовательного изменения громкости . . . . .	24
4.2 Демонстрация использования заданной шкалы настроек . . . . .	25
<b>ЗАКЛЮЧЕНИЕ</b>	<b>26</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</b>	<b>27</b>
<b>ПРИЛОЖЕНИЕ А</b>	<b>28</b>

## ВВЕДЕНИЕ

Системный уровень громкости — это важный аспект, который в значительной степени влияет на способность человека эффективно и комфортно работать на компьютере. К сожалению, не все компьютеры оснащены удобным механизмом регулировки этого параметра. Необходимость в изменении громкости возникает в разные моменты времени.

Клавиатура является очень важным инструментом для работы с компьютером и используется большим количеством пользователей. Поэтому добавление функции регулировки громкости с помощью клавиатуры с возможностью использования заданной шкалы настроек имеет смысл и актуально.

Цель работы — разработать загружаемый модуль ядра Linux для регулировки уровня громкости клавиатурой с возможностью использования заданной шкалы настроек.

# **1 Аналитический раздел**

## **1.1 Постановка задачи**

В соответствии с темой на курсовую работу по дисциплине «Операционные системы» необходимо разработать загружаемый модуль ядра Linux для регулировки уровня громкости клавиатурой с возможностью использования заданной шкалы настроек.

Для решения поставленной задачи необходимо:

- провести анализ способов обработки прерываний;
- провести анализ существующих аудио подсистем;
- разработать алгоритмы и описать структуру разрабатываемого ПО;
- разработать ПО, предоставляющее заявленную функциональность;
- исследовать разработанное ПО.

## **1.2 Анализ способов обработки прерываний**

Одной из основных задач компьютера является обеспечение связи и обмена данными с внешними устройствами. В системах, использующих монолитное ядро, все системные операции основаны на прерываниях. Прерывания делятся на три категории: системные вызовы, исключения и аппаратные прерывания. Аппаратные прерывания генерируются такими устройствами, как клавиатура, мышь или таймер, и являются асинхронными по своей природе, то есть они не зависят от других текущих процессов в системе.

Существует два типа аппаратных прерываний: быстрые и медленные. В современных системах Linux единственным быстрым прерыванием является прерывание таймера, которое является привилегированным и обрабатывается от начала до конца без какой-либо другой обработки, происходящей в процессоре. В это время все прерывания, включая прерывания других процессоров в системе, отключены. Обработчики прерываний должны выполняться как можно быстрее, чтобы минимизировать прерывание.

Все остальные прерывания являются медленными. Они состоят из двух частей — верхней половины и нижней половины. Верхняя половина выполняет

начальную работу по настройке, необходимую для прерывания, в то время как нижняя половина выполняет фактическую обработку прерывания. Верхняя половина подготавливает действия, необходимые для обработки прерывания, и прекращает взаимодействие с контроллером прерывания после выполнения своих задач. Это позволяет системе полностью разрешить прерывание и восстановить предыдущую маску прерывания. Нижняя половина выполняется при разрешенных прерываниях.

Веделяют три типа нижних половин:

- soft IRQ;
- тасклеты;
- очереди работ.

Тасклет — частный случай реализации soft IRQ. Обработчик одного тасклета выполняется только на одном процессоре, на котором выполнялась верхняя половина прерывания. Их можно создавать как статически, так и динамически.

Тасклеты считаются устаревшим механизмом, поэтому в современных версиях ядра их не используют [1].

В отличие от тасклетов, очереди работ могут блокироваться, поэтому обязательно должны быть неделимыми. Управление очередями работ реализовано иначе, чем у тасклетов. В ядре описаны структуры, описывающие работу, выполняемую обработчиком действия, очередь работ, поток ядра, выполняющий работу, пул рабочих, а также отношения между пулом рабочих и очередью работ.

Очереди работ могут выполняться на разных процессорах и имеют большие задержки, в отличие от тасклетов, которые выполняются через небольшой промежуток времени после того, как были запланированы.

Очередь работ описывается структурой `struct workqueue_struct` и приведена в листингах 1.1–1.2 [2].

Листинг 1.1 – Структура `struct workqueue_struct` (часть 1)

```
1 struct workqueue_struct {
2     struct list_head    pwqs;        /* WR: all pwqs of this wq */
3     struct list_head    list;        /* PR: list of all workqueues */
4     struct mutex         mutex;       /* protects this wq */
```

## Листинг 1.2 – Структура struct workqueue\_struct (часть 2)

```

1      int          work_color; /* WQ: current work color */
2      int          flush_color; /* WQ: current flush color */
3      atomic_t      nr_pwqs_to_flush; /* flush in progress */
4      struct wq_flusher *first_flusher; /* WQ: first flusher */
5      struct list_head flusher_queue; /* WQ: flush waiters */
6      struct list_head flusher_overflow; /* WQ: flush overflow list */
7      struct list_head maydays; /* MD: pwqs requesting rescue */
8      struct worker  *rescuer; /* MD: rescue worker */
9      int          nr_drainers; /* WQ: drain in progress */
10     int          saved_max_active; /* WQ: saved pwq max_active */
11     struct workqueue_attrs *unbound_attrs; /* PW: only for unbound wqs */
12     struct pool_workqueue *dfl_pwq; /* PW: only for unbound wqs */
13 #ifdef CONFIG_SYSFS
14     struct wq_device *wq_dev; /* I: for sysfs interface */
15 #endif
16 #ifdef CONFIG_LOCKDEP
17     char          *lock_name;
18     struct lock_class_key key;
19     struct lockdep_map lockdep_map;
20 #endif
21     char          name[WQ_NAME_LEN]; /* I: workqueue name */
22     /*
23      * Destruction of workqueue_struct is RCU protected to allow walking
24      * the workqueues list without grabbing wq_pool_mutex.
25      * This is used to dump all workqueues from sysrq.
26      */
27     struct rcu_head rcu;
28     /* hot fields used during command issue, aligned to cacheline */
29     unsigned int    flags _____cacheline_aligned; /* WQ: WQ* flags */
30     struct pool_workqueue __percpu *cpu_pwqs; /* I: per-cpu pwqs */
31     struct pool_workqueue __rcu *numa_pwq_tbl[]; /* PWR: unbound pwqs
32         indexed by node */
32 };

```

Для описания работ в очереди определена структура struct work\_struct, которая приведена в листинге 1.3 [3].

## Листинг 1.3 – Структура struct work\_struct

```

1  struct work_struct {
2      atomic_long_t data;
3      struct list_head entry;
4      work_func_t func;
5 #ifdef CONFIG_LOCKDEP
6      struct lockdep_map lockdep_map;
7 #endif
8  };

```

Таким образом, при помощи очередей работ можно описать собственный обработчик аппаратных прерываний и изменять функциональность внешних устройств.

## 1.3 Анализ аудио подсистем

Аудио драйверы реализованы в виде модулей ядра и взаимодействуют со звуковой картой через специальные фреймворки, которые обеспечивают последовательный и единообразный интерфейс для звуковых драйверов, облегчая приложениям доступ к аудиоустройствам.

Когда приложение запрашивает воспроизведение звука, запрос отправляется в соответствующий фреймворк, который взаимодействует со звуковым драйвером. Затем драйвер преобразует цифровые аудиоданные в аналоговые сигналы и отправляет их на звуковую карту для вывода.

Входящий звук обрабатывается аналогичным образом. Звуковая карта преобразует входящие аналоговые сигналы в цифровые данные и отправляет их драйверу, который затем передает их в фреймворк для обработки. Затем приложение, которое их запросило, получает обработанные данные.

Процесс преобразования цифровых аудиоданных в аналоговые сигналы и наоборот известен как цифро-аналоговое преобразование (ЦАП) и аналого-цифровое преобразование (АЦП) соответственно. Качество этих преобразований зависит от возможностей звуковой карты и драйвера.

Наиболее распространенные фреймворки для работы со звуком:

1. Open Sound System;
2. Advanced Linux Sound Architecture;
3. JACK Audio Connection Kit.

### 1.3.1 Open Sound System

Open Sound System (OSS) — это старейшая из рассматриваемых аудио подсистем, разработанная для переносимости и обеспечения единого интерфейса для приложений, связанных со звуком. Это проект с открытым исходным кодом, который доступен для широкого спектра компьютерных операционных систем, включая Linux, FreeBSD, Solaris и MacOS [4].

К основным особенностям OSS относятся:

- поддержка широкого спектра аудиоаппаратуры, включая звуковые карты и USB-аудиоустройства;
- поддержка широкого спектра аудиоформатов, включая WAV, AIFF, MP3, Ogg Vorbis и FLAC;
- поддержка широкого спектра аудиоприложений, включая аудиоплееры, аудиоредакторы и программы записи звука;
- поддержка многоканального звука и объемного звучания;
- Поддержка аудио с низкой задержкой, что позволяет использовать аудио в реальном времени в таких приложениях, как синтез музыки и игры;
- поддержка аудиоплагинов, позволяющих использовать эффекты и виртуальные инструменты сторонних производителей;
- поддержка микширования звука, позволяющая смешивать несколько источников звука;
- поддержка маршрутизации аудиосигналов, позволяющая передавать аудиосигналы между приложениями;
- поддержка потокового аудио, позволяющая передавать аудио по сети;
- поддержка захвата аудио, позволяющая захватывать аудио из различных источников;
- поддержка аудиоэффектов, позволяющая использовать эффекты и виртуальные инструменты сторонних производителей;
- поддержка визуализации аудио, позволяющая визуально представлять аудиосигналы.

### 1.3.2 Advanced Linux Sound Architecture

Advanced Linux Sound Architecture (ALSA) — это программная структура и часть ядра Linux, которая предоставляет интерфейс для драйверов устройств звуковых карт [5].



ALSA обеспечивает поддержку всех типов аудиоустройств, включая цифровые, MIDI и аналоговые, а также поддержку расширенных функций, таких как обработка эффектов в реальном времени, 3D аудио и объемный звук.

Фреймворк предоставляет набор API, которые позволяют приложениям получать доступ к функциям звуковой карты, таким как микшер, эквалайзер и MIDI-порты. Эти API включают функции для настройки и управления звуковой картой, воспроизведения и записи звука, а также управления аудио-эффектами.

Advanced Linux Sound Architecture также обеспечивает поддержку широкого спектра аудиоформатов, включая WAV, MP3, OGG и MIDI, более того существует поддержка различных аудиоэффектов, включая реверберацию, эхо, хорус, искажение и аппаратное микширование, которое позволяет смешивать несколько аудиопотоков.

### 1.3.3 JACK Audio Connection Kit

JACK Audio Connection Kit (JACK) — это система подключения и обработки аудио для Linux, MacOS и Windows. Это аудиосервер с низкой задержкой, который позволяет нескольким приложениям подключаться к одному аудиоустройству и обмениваться аудиосигналами между ними. Он подходит для создания сложных аудиоустановок, таких как те, что используются в студиях звукозаписи, системах живого звука и других профессиональных аудиоприложениях [6].

JACK разработан для простоты использования и обеспечения максимально возможного качества звука. Он поддерживает широкий спектр аудиоформатов, включая 16- и 24-битное аудио с частотой дискретизации до 192 кГц. Он также поддерживает различные аппаратные средства ввода/вывода аудио, включая USB, FireWire и Ethernet.

Фреймворк позволяет направлять аудиосигналы между приложениями, а также на внешнее оборудование. Он поддерживает широкий спектр плагинов для обработки звука, таких как эквалайзеры, компрессоры, лимитеры и задержки, а также множество других эффектов.

Является программным обеспечением с открытым исходным кодом и доступен бесплатно. Его поддерживает большое сообщество пользователей и

разработчиков, которые постоянно работают над улучшением программы и добавлением новых функций.

JACK работает по принципу архитектуры сервер–клиент, где сервер JACK управляет аудиоустройствами и приложениями, а клиенты JACK используют сервер для подключения к аудиоустройствам и взаимодействия с другими приложениями.

Сервер JACK предоставляет высококачественный аудиопоток с низкой задержкой, синхронизированный с системными часами. Этот поток делится на аудиокадры, каждый из которых представляет собой небольшую единицу аудиоданных, передаваемых между клиентами JACK.

Клиентами JACK могут быть аудиоприложения или плагины, такие как синтезаторы, сэмплеры, процессоры эффектов и цифровые аудио рабочие станции (DAW). JACK позволяет подключать эти приложения в режиме реального времени, обеспечивая модульный и гибкий подход к обработке звука.

Для функционирования и работы необходимо установить сервер и клиенты JACK. После установки вы можете запустить сервер JACK и запустить клиенты JACK, которые автоматически подключатся к серверу. Графический инструмент JACK Patchbay упрощает маршрутизацию аудиосигналов между клиентами.

### **1.3.4 Сравнение рассмотренных решений**

В соответствии с поставленной задачей необходимо разработать загружаемый модуль ядра Linux, а JACK Audio Connection Kit позволяет работать только в пространстве пользователя, соответственно его использование в данной работе невозможно.

В таблице 1.1 представлено сравнение OSS и ALSA.

Таблица 1.1 – Сравнение OSS и ALSA

Преимущества OSS	Преимущества ALSA
Поддержка старых устройств	Поддержка современных устройств
Кроссплатформенность	Linux ориентированность
Более простое API	Большой функционал
	Лучшая поддержка USB устройств
	Лучшая поддержка MIDI устройств
	Лучшая поддержка Bluetooth устройств

## Выводы

В данном разделе были рассмотрены способы обработки прерываний от внешних устройств, а также основные аудио подсистемы.

Для реализации поставленной задачи и отслеживания нажатых клавиш на клавиатуре необходимо разработать загружаемый модуль ядра, в котором будут перехватываться прерывания клавиатуры.

Для обработки прерываний выбраны очереди работ, так как для добавления нового softIRQ требуется перекомпиляция ядра, а использование такслетов нецелесообразно, так как они признаны устаревшими и в будущем будут выведены из ядра.

В результате сравнительного анализа аудио подсистем выбран подход с использованием фреймворка ALSA ввиду большого функционала и широкой совместимости с современными звуковыми устройствами.

## 2 Конструкторский раздел

### 2.1 IDEF0–диаграммы

На рисунках 2.1–2.2 приведены диаграммы, описывающие последовательность действий, выполняемых в ПО.

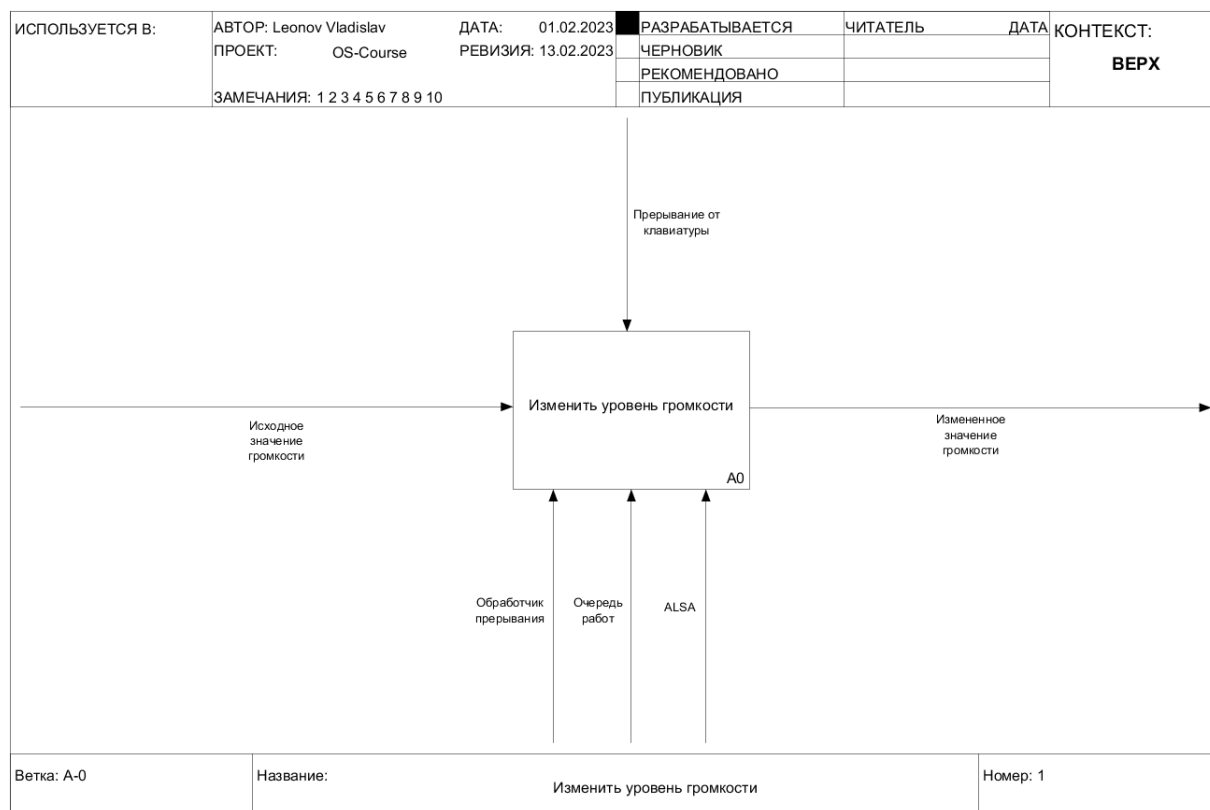


Рисунок 2.1 – Уровень A0

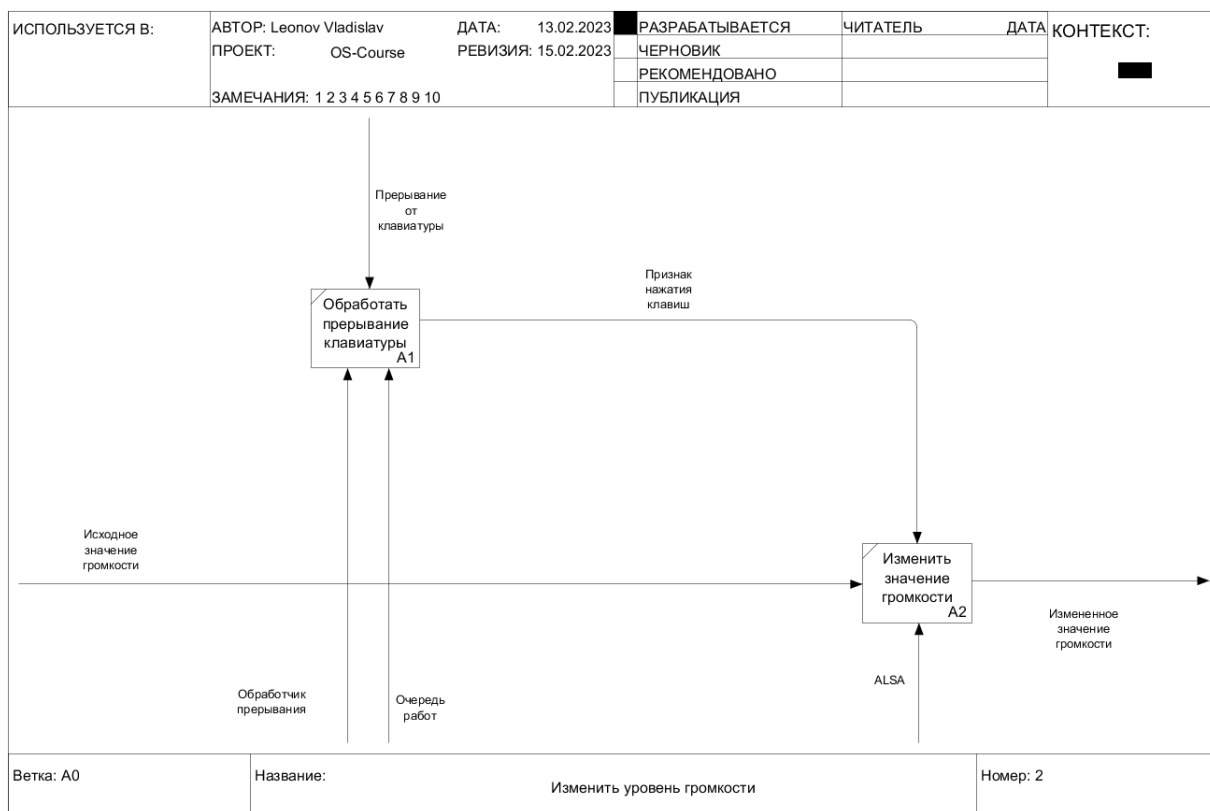


Рисунок 2.2 – Уровень A1-A2

## 2.2 Разработка алгоритмов

Одновременное нажатие следующих клавиш выполняет следующие действия:

- ALT и 0 — установить уровень громкости 0%;
- ALT и 1 — установить уровень громкости 10%;
- ALT и 2 — установить уровень громкости 20%;
- ALT и 3 — установить уровень громкости 30%;
- ALT и 4 — установить уровень громкости 40%;
- ALT и 5 — установить уровень громкости 50%;
- ALT и 6 — установить уровень громкости 60%;
- ALT и 7 — установить уровень громкости 70%;
- ALT и 8 — установить уровень громкости 80%;

- ALT и 9 — установить уровень громкости 90%;
- ALT и – — уменьшить текущий уровень громкости на 1%;
- ALT и = — увеличить текущий уровень громкости на 1%.

Алгоритм обработки прерывания от клавиатуры изображен на рисунке 2.3.

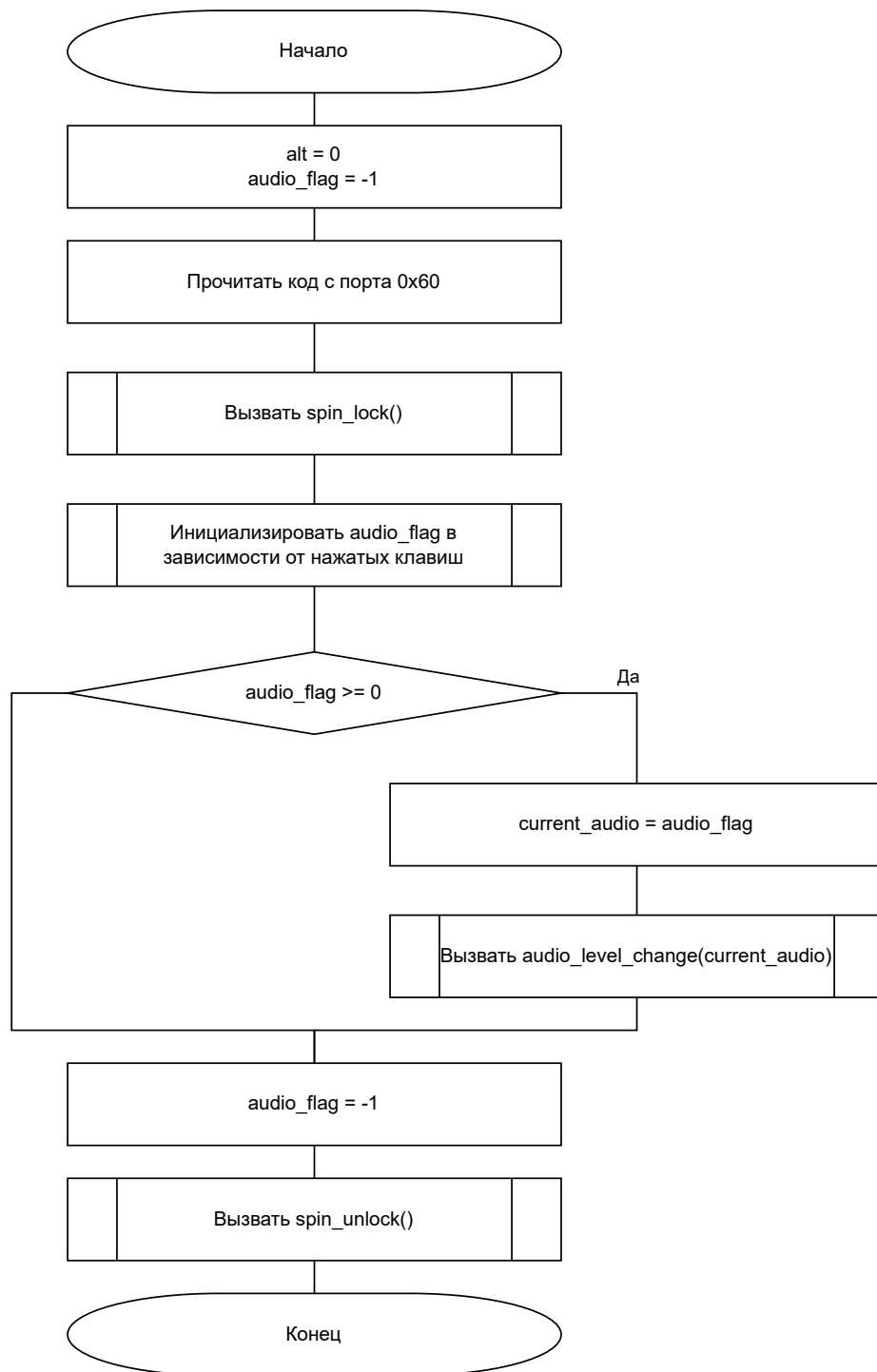


Рисунок 2.3 – Алгоритм обработки прерывания от клавиатуры

Алгоритм изменения громкости изображен на рисунке 2.4.



Рисунок 2.4 – Алгоритм изменения громкости

## 2.3 Структура ПО

Разработанное программное обеспечение включает в себя:

- загружаемый модуль ядра, осуществляющий перехват прерываний от клавиатуры;
- вспомогательный модуль для регулировки громкости.

На рисунке 2.5 приведена структура ПО.

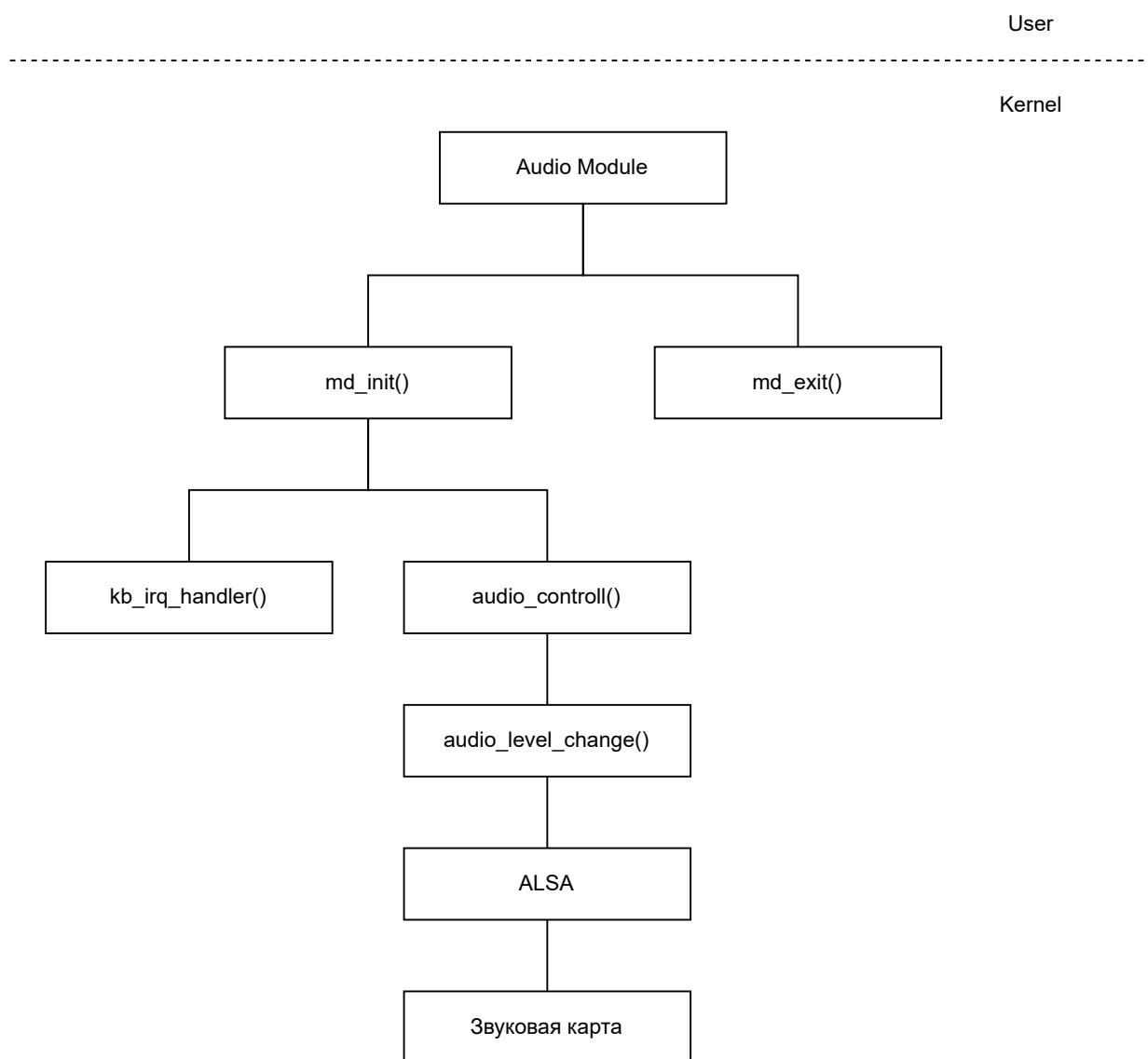


Рисунок 2.5 – Структура ПО



## 3 Технологический раздел

### 3.1 Выбор языка и среды программирования

В качестве языка программирования был выбран язык C. Данный ЯП используется для разработки всех модулей ядра и драйверов операционной системы Linux.

Среда программирования — Microsoft Visual Studio Code [7]. Данное решение является кроссплатформенным и бесплатным, а также имеет открытый исходный код.

### 3.2 Точки входа в модуль

В листингах 3.1–3.2 представлены точки входа в модуль — `init` и `exit` — в которых происходит создание и уничтожение очереди работ, а также регистрация и deregистрация обработчика прерываний от клавиатуры.

Листинг 3.1 – Точки входа в модуль (часть 1)

```
1 static int __init md_init(void)
2 {
3     int ret;
4     printk(KERN_INFO "Module Initialization.\n");
5     ret = request_irq(KB_IRQ, (irq_handler_t)kb_irq_handler, IRQF_SHARED,
6         "Custom Handler", (void *) (kb_irq_handler));
7     if (ret != 0)
8         printk(KERN_INFO "Cannot Request IRQ for keyboard.\n");
9
10    work = kmalloc(sizeof(struct work_struct), GFP_KERNEL);
11    if (work)
12    {
13        INT_WORK((struct work_struct *)work, audio_controll);
14    }
15    wq = create_workqueue("keyboard_queue");
16
17    printk(KERN_INFO "Module Inited.\n");
18    return 0;
19 }
```

### Листинг 3.2 – Точки входа в модуль (часть 2)

```
1 static void __exit md_exit(void)
2 {
3     free_irq(KB_IRQ, (void *) (kb_irq_handler));
4     flush_workqueue(wq);
5     destroy_workqueue(wq);
6
7     if (work)
8         kfree(work);
9
10    printk(KERN_INFO "Module Exit.");
11 }
12
13 module_init(md_init);
14 module_exit(md_exit);
```

## 3.3 Обработка прерываний клавиатуры

В листингах 3.3–3.6 представлен обработчик прерываний клавиатуры.

### Листинг 3.3 – Обработчик прерываний клавиатуры

```
1 irq_handler_t kb_irq_handler(int irq, void *dev_id, struct pt_regs *regs)
2 {
3     spin_lock(&k_lock);
4     scancode = inb(0x60);
5     spin_unlock(&k_lock);
6
7     queue_work(wq, work);
8
9     return (irq_handler_t)IRQ_HANDLED;
10 }
```

### Листинг 3.4 – Обработчик нижней половины прерывания(часть 1)

```
1 void audio_controll(struct work_struct *work)
2 {
3     static int alt = 0;
4     static long audio_flag = -1;
5
6     spin_lock(&k_lock);
7     switch (scancode)
8     {
9     case ALT_PRESSED:
10         alt = 1;
11         break;
```

Листинг 3.5 – Обработчик нижней половины прерывания(часть 2)

```
1      case ALT_UNPRESSED:
2          alt = 0;
3          break;
4      case LEVEL0_PRESSED:
5          if (alt)
6              audio_flag = 0;
7          break;
8      case LEVEL1_PRESSED:
9          if (alt)
10             audio_flag = 10;
11         break;
12     case LEVEL2_PRESSED:
13         if (alt)
14             audio_flag = 20;
15         break;
16     case LEVEL3_PRESSED:
17         if (alt)
18             audio_flag = 30;
19         break;
20     case LEVEL4_PRESSED:
21         if (alt)
22             audio_flag = 40;
23         break;
24     case LEVEL5_PRESSED:
25         if (alt)
26             audio_flag = 50;
27         break;
28     case LEVEL6_PRESSED:
29         if (alt)
30             audio_flag = 60;
31         break;
32     case LEVEL7_PRESSED:
33         if (alt)
34             audio_flag = 70;
35         break;
36     case LEVEL8_PRESSED:
37         if (alt)
38             audio_flag = 80;
39         break;
40     case LEVEL9_PRESSED:
41         if (alt)
42             audio_flag = 90;
43         break;
```

### Листинг 3.6 – Обработчик нижней половины прерывания(часть 3)

```
1      case LEVELMINUS_PRESSED:
2          if (alt && current_audio > 0)
3              audio_flag = current_audio - 1;
4          break;
5      case LEVELPLUS_PRESSED:
6          if (alt && current_audio < 100)
7              audio_flag = current_audio + 1;
8          break;
9      default:
10         break;
11     }
12
13     if (audio_flag >= 0)
14     {
15         printk(KERN_INFO "System Volume Changing Started.\n");
16         current_audio = audio_flag;
17         char buffer[10];
18         sprintf(buffer, "%d", current_audio);
19         char *argv[] = {"/home/leerycorsair/os_course/audio_controller",
20             buffer, NULL};
21         static char *envp[] = {
22             "HOME=",
23             "TERM=linux",
24             "PATH=/sbin:/bin:/usr/sbin:/usr/bin", NULL};
25
26         call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC);
27         printk(KERN_INFO "System Volume Changed to %d%%.\n", current_audio);
28     }
29
30     audio_flag = -1;
31     spin_unlock(&k_lock);
32 }
```

## 3.4 Функция изменения громкости

В листинге 3.7 приведена структура, описывающая звуковое устройство для которого создается функция изменения громкости.

### Листинг 3.7 – Структура `system_volume_controller`

```
1 struct audio_controller
2 {
3     long min, max;
4     snd_mixer_t *handle;
5     snd_mixer_selem_id_t *sid;
6     const char *card;
7     const char *selem_name;
8 } a_controller;
```

В данной структуре созданы следующие поля:

- `min` и `max` — минимальный и максимальный физический уровень громкости;
- `handle` — микшерное устройство;
- `sid` — идентификатор микшерного элемента;
- `card` — имя устройства воспроизведения;
- `selem_name` — имя микшерного элемента.

В листингах 3.8–3.9 представлена функция изменения уровня громкости.

### Листинг 3.8 – Функция изменения громкости (часть 1)

```
1 void audio_level_change(long audio_level)
2 {
3     a_controller.card = "default";
4     a_controller.selem_name = "Master";
5
6     if (snd_mixer_open(&(a_controller.handle), 0) < 0)
7         return;
8
9     if (snd_mixer_attach(a_controller.handle, a_controller.card) < 0)
10         return;
11
12     if (snd_mixer_selem_register(a_controller.handle, NULL, NULL) < 0)
13         return;
14
15     if (snd_mixer_load(a_controller.handle) < 0)
16         return;
17
18     snd_mixer_selem_id_alloca(&(a_controller.sid));
19     snd_mixer_selem_id_set_index(a_controller.sid, 0);
20     snd_mixer_selem_id_set_name(a_controller.sid, a_controller.selem_name);
```

### Листинг 3.9 – Функция изменения громкости (часть 2)

```
1     snd_mixer_elem_t *elem = snd_mixer_find_selem(a_controller.handle,
2         a_controller.sid);
3     if (elem == NULL)
4         return;
5     snd_mixer_selem_get_playback_volume_range(elem, &(a_controller.min),
6         &(a_controller.max));
7     if (snd_mixer_selem_set_playback_volume_all(elem, audio_level *
8         a_controller.max / 100) < 0)
9         return;
10 }
```

## 3.5 Сборка разработанного модуля

Для компиляции загружаемого модуля используется утилита make. Конфигурационный файл makefile приведен в листинге 3.10 [8].

### Листинг 3.10 – makefile для сборки

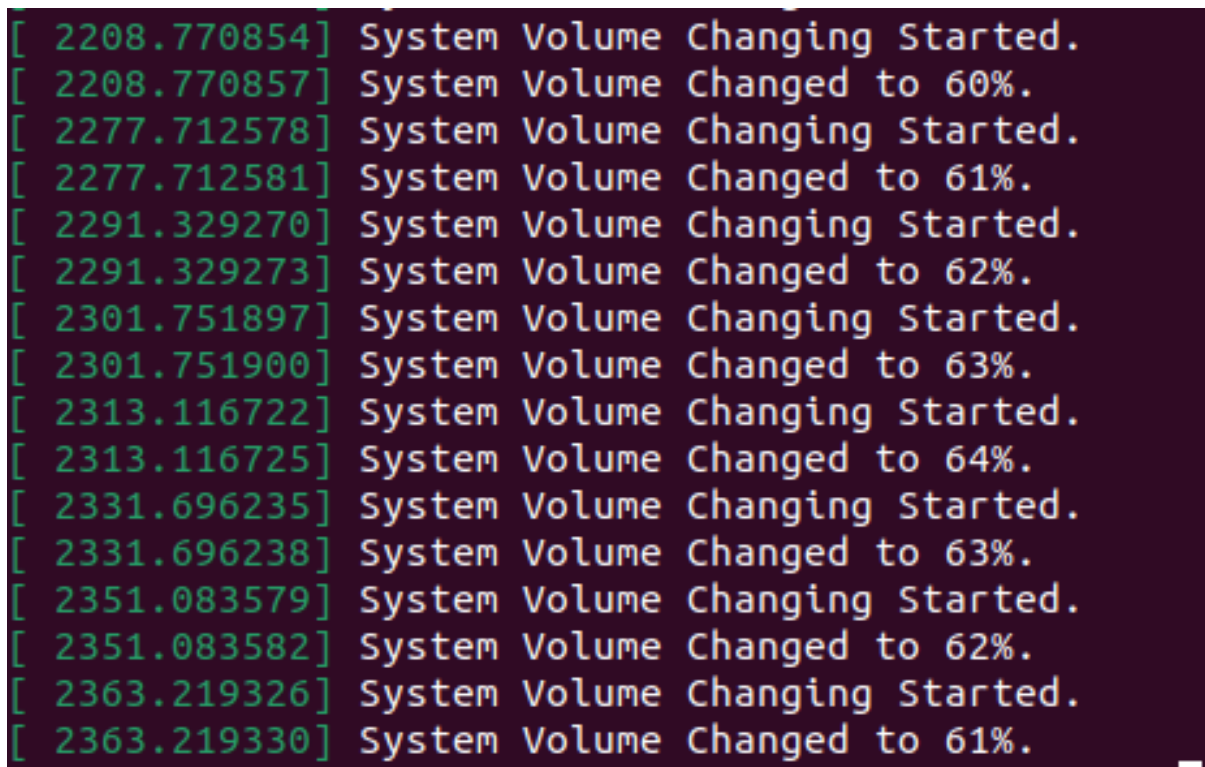
```
1 CONFIG_MODULE_SIG=n
2 CURRENT = $(shell uname -r)
3 KDIR = /lib/modules/$(CURRENT)/build
4 PWD = $(shell pwd)
5 TARGET = audio_module
6 obj-m := $(TARGET).o
7
8
9 default: audio_controller
10     $(MAKE) -C$(KDIR) M=$(PWD) modules
11
12 audio_controller: audio_controller.c
13     gcc -o $$@ $$@.c -lasound
```

## 4 Исследовательский раздел

Для разработки и тестирования данной работы использовался ноутбук Honor MagicBook Pro 2021 и операционная система Linux Ubuntu 22.04.

### 4.1 Демонстрация последовательного изменения громкости

На рисунке 4.1 изображены логи последовательного изменения громкости.

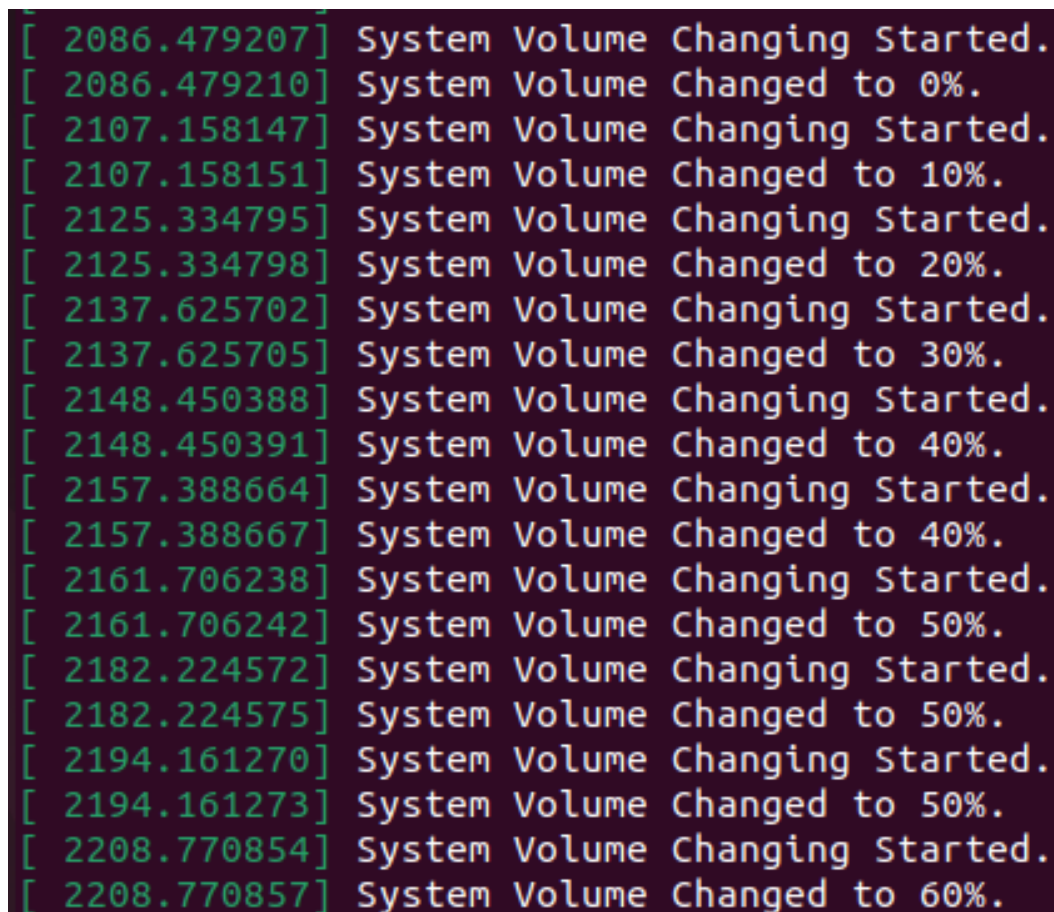
A screenshot of a terminal window with a dark purple background. It displays a series of system logs for volume changes. Each log entry consists of a timestamp in green brackets followed by a message in orange text. The messages alternate between 'System Volume Changing Started.' and 'System Volume Changed to X%.' where X is a percentage value. The volume starts at 60% and increases in 1% increments up to 64%, then decreases to 63%, then 62%, and finally 61%.

```
[ 2208.770854] System Volume Changing Started.  
[ 2208.770857] System Volume Changed to 60%.  
[ 2277.712578] System Volume Changing Started.  
[ 2277.712581] System Volume Changed to 61%.  
[ 2291.329270] System Volume Changing Started.  
[ 2291.329273] System Volume Changed to 62%.  
[ 2301.751897] System Volume Changing Started.  
[ 2301.751900] System Volume Changed to 63%.  
[ 2313.116722] System Volume Changing Started.  
[ 2313.116725] System Volume Changed to 64%.  
[ 2331.696235] System Volume Changing Started.  
[ 2331.696238] System Volume Changed to 63%.  
[ 2351.083579] System Volume Changing Started.  
[ 2351.083582] System Volume Changed to 62%.  
[ 2363.219326] System Volume Changing Started.  
[ 2363.219330] System Volume Changed to 61%.
```

Рисунок 4.1 – Логи последовательного изменения громкости

## 4.2 Демонстрация использования заданной шкалы настроек

На рисунке 4.2 изображены логи при использовании заданной шкалы настроек громкости.

A screenshot of a terminal window with a dark purple background. It displays a series of system logs in a monospaced font. Each log entry consists of a timestamp in green and a message in white. The messages indicate the start of a volume change and the resulting percentage. The volume is changed from 0% to 60% in 10% increments, with some steps (40% and 50%) being repeated.

```
[ 2086.479207] System Volume Changing Started.  
[ 2086.479210] System Volume Changed to 0%.  
[ 2107.158147] System Volume Changing Started.  
[ 2107.158151] System Volume Changed to 10%.  
[ 2125.334795] System Volume Changing Started.  
[ 2125.334798] System Volume Changed to 20%.  
[ 2137.625702] System Volume Changing Started.  
[ 2137.625705] System Volume Changed to 30%.  
[ 2148.450388] System Volume Changing Started.  
[ 2148.450391] System Volume Changed to 40%.  
[ 2157.388664] System Volume Changing Started.  
[ 2157.388667] System Volume Changed to 40%.  
[ 2161.706238] System Volume Changing Started.  
[ 2161.706242] System Volume Changed to 50%.  
[ 2182.224572] System Volume Changing Started.  
[ 2182.224575] System Volume Changed to 50%.  
[ 2194.161270] System Volume Changing Started.  
[ 2194.161273] System Volume Changed to 50%.  
[ 2208.770854] System Volume Changing Started.  
[ 2208.770857] System Volume Changed to 60%.
```

Рисунок 4.2 – Логи при использовании заданной шкалы настроек громкости



## ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы:

- проанализированы способы обработки прерываний;
- проанализированы существующие аудио подсистемы;
- разработаны алгоритмы и описана структура разработанного ПО;
- разработано ПО, предоставляющее заявленную функциональность;
- выполнено исследование разработанного ПО.

Таким образом, разработанное программное обеспечение отвечает поставленной задаче — загружаемый модуль ядра Linux позволяет регулировать уровень громкости клавиатурой с возможностью использования заданной шкалы настроек.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Modernizing the tasklet API? [Электронный ресурс]. — Режим доступа: <https://lwn.net/Articles/830964/> (дата обращения: 04.02.2023).
2. workqueue.c Linux source code [Электронный ресурс]. — Режим доступа: <https://elixir.bootlin.com/linux/v5.15/source/kernel/workqueue.c#L256> (дата обращения: 11.02.2023).
3. workqueue.h Linux source code [Электронный ресурс]. — Режим доступа: <https://elixir.bootlin.com/linux/v5.15/source/include/linux/workqueue.h#L97> (дата обращения: 11.02.2023).
4. Open Sound System [Электронный ресурс]. — Режим доступа: <http://www.opensound.com/oss.html> (дата обращения: 11.02.2023).
5. ALSA [Электронный ресурс]. — Режим доступа: [https://www.alsa-project.org/wiki/Main\\_Page](https://www.alsa-project.org/wiki/Main_Page) (дата обращения: 11.02.2023).
6. JACK [Электронный ресурс]. — Режим доступа: <https://jackaudio.org/> (дата обращения: 11.02.2023).
7. VS Code [Электронный ресурс]. — Режим доступа: <https://code.visualstudio.com/> (дата обращения: 11.02.2023).
8. Makefile [Электронный ресурс]. — Режим доступа: <https://www.gnu.org/software/make/manual/make.html> (дата обращения: 11.02.2023).

## ПРИЛОЖЕНИЕ А

Листинг А.1 – Листинг audio\_module.c(часть 1)

```
1 #include <linux/kernel.h>
2 #include <linux/module.h>
3 #include <linux/workqueue.h>
4 #include <linux/interrupt.h>
5 #include <linux/kmod.h>
6 #include <linux/slab.h>
7 #include <asm/io.h>
8
9 #define KB_IRQ 1
10
11 #define ALT_PRESSED 56    // Alt pressed
12 #define ALT_UNPRESSED 184 // Alt unpresse
13
14 #define LEVEL0_PRESSED 11    // Key 0 pressed 0%
15 #define LEVEL1_PRESSED 2     // Key 1 pressed 10%
16 #define LEVEL2_PRESSED 3     // Key 2 pressed 20%
17 #define LEVEL3_PRESSED 4     // Key 3 pressed 30%
18 #define LEVEL4_PRESSED 5     // Key 4 pressed 40%
19 #define LEVEL5_PRESSED 6     // Key 5 pressed 50%
20 #define LEVEL6_PRESSED 7     // Key 6 pressed 60%
21 #define LEVEL7_PRESSED 8     // Key 7 pressed 70%
22 #define LEVEL8_PRESSED 9     // Key 8 pressed 80%
23 #define LEVEL9_PRESSED 10    // Key 9 pressed 90%
24 #define LEVELMINUS_PRESSED 12 // Key – pressed decrease
25 #define LEVELPLUS_PRESSED 13 // Key = pressed increase
26
27 unsigned char scancode;
28 long current_audio = 0;
29
30 MODULE_LICENSE("GPL");
31 MODULE_AUTHOR("Leonov Vladislav");
32
33 DEFINE_SPINLOCK(k_lock);
34
35 static struct workqueue_struct *wq;
36 static struct work_struct *work;
```

## Листинг А.2 – Листинг audio\_module.c(часть 2)

```

1 void audio_controll(struct work_struct *work)
2 {
3     static int alt = 0;
4     static long audio_flag = -1;
5
6     spin_lock(&k_lock);
7     switch (scancode)
8     {
9         case ALT_PRESSED:
10             alt = 1;
11             break;
12         case ALT_UNPRESSED:
13             alt = 0;
14             break;
15         case LEVEL0_PRESSED:
16             if (alt)
17                 audio_flag = 0;
18             break;
19         case LEVEL1_PRESSED:
20             if (alt)
21                 audio_flag = 10;
22             break;
23         case LEVEL2_PRESSED:
24             if (alt)
25                 audio_flag = 20;
26             break;
27         case LEVEL3_PRESSED:
28             if (alt)
29                 audio_flag = 30;
30             break;
31         case LEVEL4_PRESSED:
32             if (alt)
33                 audio_flag = 40;
34             break;
35         case LEVEL5_PRESSED:
36             if (alt)
37                 audio_flag = 50;
38             break;
39         case LEVEL6_PRESSED:
40             if (alt)
41                 audio_flag = 60;
42             break;
43         case LEVEL7_PRESSED:
44             if (alt)
45                 audio_flag = 70;
46             break;

```

### Листинг А.3 – Листинг audio\_module.c(часть 3)

```

1      case LEVEL8_PRESSED:
2          if (alt)
3              audio_flag = 80;
4          break;
5      case LEVEL9_PRESSED:
6          if (alt)
7              audio_flag = 90;
8          break;
9      case LEVELMINUS_PRESSED:
10         if (alt && current_audio > 0)
11             audio_flag = current_audio - 1;
12         break;
13     case LEVELPLUS_PRESSED:
14         if (alt && current_audio < 100)
15             audio_flag = current_audio + 1;
16         break;
17     default:
18         break;
19 }
20
21 if (audio_flag >= 0)
22 {
23     printk(KERN_INFO "System Volume Changing Started.\n");
24     current_audio = audio_flag;
25     char buffer[10];
26     sprintf(buffer, "%d", current_audio);
27     char *argv[] = {"/home/leerycorsair/os_course/audio_controller",
28                     buffer, NULL};
29     static char *envp[] = {
30         "HOME=",
31         "TERM=linux",
32         "PATH=/sbin:/bin:/usr/sbin:/usr/bin", NULL};
33
34     call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC);
35     printk(KERN_INFO "System Volume Changed to %d%%.\n", current_audio);
36 }
37
38 audio_flag = -1;
39 spin_unlock(&k_lock);
40
41 irq_handler_t kb_irq_handler(int irq, void *dev_id, struct pt_regs *regs)
42 {
43     spin_lock(&k_lock);
44     scancode = inb(0x60);
45     spin_unlock(&k_lock);

```

#### Листинг А.4 – Листинг audio\_module.c(часть 4)

```

1    queue_work(wq, work);
2
3    return (irq_handler_t)IRQ_HANDLED;
4 }
5
6 static int __init md_init(void)
7 {
8     int ret;
9     printk(KERN_INFO "Module Initialization.\n");
10    ret = request_irq(KB_IRQ, (irq_handler_t)kb_irq_handler, IRQF_SHARED,
11        "Custom Handler", (void *) (kb_irq_handler));
12    if (ret != 0)
13        printk(KERN_INFO "Cannot Request IRQ for keyboard.\n");
14
15    work = kmalloc(sizeof(struct work_struct), GFP_KERNEL);
16    if (work)
17    {
18        INIT_WORK((struct work_struct *)work, audio_controll);
19    }
20    wq = create_workqueue("keyboard_queue");
21
22    printk(KERN_INFO "Module Inited.\n");
23    return 0;
24 }
25
26 static void __exit md_exit(void)
27 {
28     free_irq(KB_IRQ, (void *) (kb_irq_handler));
29     flush_workqueue(wq);
30     destroy_workqueue(wq);
31
32     if (work)
33         kfree(work);
34
35     printk(KERN_INFO "Module Exit.");
36 }
37
38 module_init(md_init);
39 module_exit(md_exit);

```

## Листинг А.5 – Листинг audio\_controller.c

```

1
2 #include <alsa/asoundlib.h>
3
4 struct audio_controller
5 {
6     long min, max;
7     snd_mixer_t *handle;
8     snd_mixer_selem_id_t *sid;
9     const char *card;
10    const char *selem_name;
11 } a_controller;
12
13 void audio_level_change(long audio_level)
14 {
15     a_controller.card = "default";
16     a_controller.selem_name = "Master";
17
18     if (snd_mixer_open(&(a_controller.handle), 0) < 0)
19         return;
20
21     if (snd_mixer_attach(a_controller.handle, a_controller.card) < 0)
22         return;
23
24     if (snd_mixer_selem_register(a_controller.handle, NULL, NULL) < 0)
25         return;
26
27     if (snd_mixer_load(a_controller.handle) < 0)
28         return;
29
30     snd_mixer_selem_id_alloca(&(a_controller.sid));
31     snd_mixer_selem_id_set_index(a_controller.sid, 0);
32     snd_mixer_selem_id_set_name(a_controller.sid, a_controller.selem_name);
33
34     snd_mixer_elem_t *elem = snd_mixer_find_selem(a_controller.handle,
35         a_controller.sid);
36     if (elem == NULL)
37         return;
38
39     snd_mixer_selem_get_playback_volume_range(elem, &(a_controller.min),
40         &(a_controller.max));
41     if (snd_mixer_selem_set_playback_volume_all(elem, audio_level *
42         a_controller.max / 100) < 0)
43         return;
44
45     snd_mixer_close(a_controller.handle);
46 }

```