Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования

Московский государственный технический университет
имени Н. Э. Баумана
(МГТУ им. Н.Э.Баумана)

Факультет «Информатика и системы управления»
Кафедра «Программное обеспечение ЭВМ
и информационные технологии»

Расчётно-пояснительная записка
к курсовой работе по курсу
«Операционные системы»
на тему

Информирование индикатором клавиатуры получения сигнала с входа микрофона в ОС Linux

Исполнитель: студент ИУ7-71 Петухов И. С. _____

Руководитель: преподаватель ИУ7 Рогозин О. В. _____

Москва, 2016

Содержание

Введение

Персональный компьютер, так же сокращённо ПК, полагается для работы с человеком на прямую, то есть компьютер даёт возможность получить понятную информацию для человека. [1]

Все виды персональных компьютеров (настольные компьютеры или десктопы, неттопы, моноблоки, ноутбуки, нетбуки, планшеты и планшетные ноутбуки, карманные типы компьютеров и смартфоны) взаимодействуют с пользователем посредством устройств ввода-вывода.

Устройство ввода-вывода — компонент типовой архитектуры ЭВМ, предоставляющий компьютеру возможность взаимодействия с внешним миром и, в частности, с пользователями.

Устройства ввода — это, в основном, датчики преобразования неэлектрических величин (расположение в пространстве, давление, вязкость, скорость, ускорение, освещённость, температура, влажность, перемещение, количественные величины и т. п.) и электрических величин в электрические сигналы воспринимаемые процессором для дальнейшей их обработки в основном в цифровом виде. [2]

Важнейшим устройством ввода у персональных компьютеров является микрофон. **Микрофон** — электроакустический прибор, преобразующий акустические колебания в электрические колебания. [3]

Все ПК оборудованы микрофонным входом **Mic In** (Microphone In).

Устройства вывода — это преобразователи электрической цифровой информации в вид необходимый для получения требуемого результата, могущего быть как неэлектрической природы (механические, тепловые, оптические, звуковые), так и электрической природы (трансформаторы, нагреватели, электродвигатели,реле). [2]

Важнейшим устройством вывода у персональных компьютеров является акустическая система. **Акустическая система** — устройство для воспроизведения звука, состоит из акустического оформления и вмонтированных в него излучающих головок (обычно динамических). [4]

Все ПК оборудованы разъемом для акустичской системы (наушники, колонки).

С развитием персональных компьютеров и их минимизацией разъемы для наушников и микрофона объединились. Устройство, сочетающее микрофон и наушники, называется гарнитурой, а разъем для гарнитуры - **гарнитурный разъем**. Все мобильные устройства поддерживают гарнитуру. Обычно **в гарнитуру встраиваются кнопки**, позволяющие без прямого взаимодейсвтия с мобильным устройством быстро отвечать на звонки, изменять громкость, пролистывать музыку.

Смартфоны и планшеты работают на **мобильных операционных системах**. Современные операционные системы для мобильных устройств: Android, CyanogenMod, Cyanogen OS, Fire OS, Flyme OS, iOS, Windows Phone, BlackBerry OS, Firefox OS, Sailfish OS, Tizen, Ubuntu Touch. Устаревшие, ныне не поддерживаемые программные платформы: Symbian, Windows Mobile, Palm OS, webOS, Maemo, MeeGo, LiMo. [5]

Многие мобильные операционные системы поддерживают работу с кнопками гарнитуры. Эти ОС предоставляют удобный API для обработки нажатий кнопок. Существует большое колличество программ для мобильных устройств, использующие API от операционной системы. Например: **Headset Button Controller** - управление музыкальным проигрывателем и другими функциями телефона с проводной гарнитуры, превращает проводную гарнитуру в пульт дистанционного управления для вашего телефона. Работает как с 1-но кнопочной, так и с 3-х кнопочными гарнитурами. Действия можно программировать на любую из клавиш. Аналогичные программы: Headset Droid, Headset Volume Controller, JAYS Headset Control и Philips Headset. Пример кода на ОС Android представлен в листинге 1 [6].

Листинг 1 — Пример обработки кнопки гарнитуры

```
boolean onKeyDown(int keyCode, KeyEvent event) {
    AudibleReadyPlayer abc;
    switch (keyCode) {
    case KeyEvent.KEYCODE_MEDIA_FAST_FORWARD:
            // code for fast forward
            return true;
    case KeyEvent.KEYCODE_MEDIA_NEXT:
            // code for next
            return true;
    case KeyEvent.KEYCODE_MEDIA_PLAY_PAUSE:
            // code for play/pause
            return true;
    case KeyEvent.KEYCODE_MEDIA_PREVIOUS:
            // code for previous
            return true;
    case KeyEvent.KEYCODE_MEDIA_REWIND:
            // code for rewind
            return true;
    case KeyEvent.KEYCODE_MEDIA_STOP:
            // code for stop
            return true;
    }
    return false;
}
```

Также существуют альтернативные способы использования гарнитурного разъема. Периферийные устройства, например, глюкометр iHealth Lab (определяющий уровень сахара в крови), Irdroid – ИК-пульт для дистанционного управления телевизором, приставками и звуковыми компонентами и Flojack – устройство чтения NFC (организующее радиосвязь между находящимися рядом мобильными устройствами) расширяют возможности мобильных устройств, используя гарнитурный разъем.

Однако операционные системы Linux и Windows, ориентрированные на десктопы и ноутбуки, не поддерживают работу с гарнитурой.

В статье [7] автор собирает осцилограф и пользуется программой xoscope. Xoscope для Linux - цифровой осцилограф, который использует звуковую карту. Это единственное нестандартное использование микрофоного входа компьютера на не мобильной операционной системе.

В данной работе ставится задача написания программы, которая бы обнаруживала в ОС Linux нажатие кнопоки на гарнитуре.

Другим важнейшим устройством вывода у персональных компьютеров является **монитор или дисплей**.

Если в системе возникает критическая ошибка, она можеть быть выведена на экран или записана в логи. Иногда компьютер работает без дисплея, а возможности прочитать лог-файлы нет, так как диск зашифрован, а ошибка случается до расшировки. Из данной ситуации можно выйти, используя альтернативное взаимодействие с пользователем.

Когда в Linux случается kernel panic, ошибка может быть выведена через LED индикатор клаиатуры, используя азбуку Морзе. [8]

BIOS (Basic Input/Output System – базовая система ввода-вывода) - пограмма системного уровня, предназначенная для первоначального запуска компьютера, настройки оборудования и обеспечения функций ввода/вывода. BIOS записывается в микросхему постоянной памяти, которая расположена на системной плате. В персональных и портативных компьютерах система BIOS производит запуск компьютера и процедуру самотестирования (Power-On Self Test – POST). Программа, расположенная в микросхеме BIOS, загружается первой после включения компьютера. Программа определяет и проверяет установленное оборудование, настраивает устройства и готовит их к работе. При самотестировании, возможно, будет обнаружена неисправность оборудования, тогда процедура POST будет остановлена с выводом соответствующего сообщения или звукового сигнала. Если же все проверки прошли успешно, самотестирование завершается вызовом встроенной подпрограммы для загрузки операционной системы. Ну а если же программой будет выявлена серьезная ошибка, работа системы будет остановлена с выдачей звуковых сигналов, которые укажут на возникшую неисправность. [9]

Т.о. на низком уровне большую роль играет альтернативное взаимодействие компьютера с пользователем. Поэтому в данной работе разберем информирование пользователя компьютера LED индикаторами на уровне ядра в ОС Linux.

1 Аналитический раздел

1.1 Выбор подхода к реализации драйвера на ОС Linux

Одной из основных задач операционной системы является управление аппаратной частью. Ту программу или тот кусок программного кода, который предназначен для управления конкретным устройством, и называют обычно драйвером устройства. Необходимость драйверов устройств в операционной системе объясняется тем, что каждое отдельное устройство воспринимает только свой строго фиксированный набор специализированных команд, с помощью которых этим устройством можно управлять. Причем команды эти чаще всего предназначены для выполнения каких-то простых элементарных операций. Если бы каждое приложение вынуждено было использовать только эти команды, писать приложения было бы очень сложно, да и размер их был бы очень велик. Поэтому приложения обычно используют какие-то команды высокого уровня (типа «записать файл на диск»), а о преобразовании этих команд в управляющие последовательности для конкретного устройства заботится драйвер этого устройства. Поэтому каждое отдельное устройство, будь то дисковод, клавиатура или принтер, должно иметь свой программный драйвер, который выполняет роль транслятора или связующего звена между аппаратной частью устройства и программными приложениями, использующими это устройство. [10]

В Linux драйверы устройств бывают трех типов.

1.1.1 Драйверы первого типа

являются частью программного кода ядра (встроены в ядро). Соответствующие устройства автоматически обнаруживаются системой и становятся доступны для приложений. Обычно таким образом обеспечивается поддержка тех устройств, которые необходимы для монтирования корневой файловой системы и запуска компьютера. Примерами таких устройств являются стандартный видеоконтроллер VGA, контроллеры IDE-дисков, материнская плата, последовательные и параллельные порты.

1.1.2 Драйверы второго типа

представлены модулями ядра. Они оформлены в виде отдельных файлов и для их подключения (на этапе загрузки или впоследствии) необходимо выполнить отдельную команду подключения модуля, после чего будет обеспечено управление соответствующим устройством. Если необходимость в использовании устройства отпала, модуль можно выгрузить из памяти (отключить). Поэтому использование модулей обеспечивает большую гибкость, так как каждый такой драйвер может быть переконфигурирован без остановки системы. Модули часто используются для управления такими устройствами как SCSI-адаптеры, звуковые и сетевые карты.

1.1.3 Драйверы третьего типа

Для таких устройств программный код драйвера поделен между ядром и специальной утилитой, предназначенной для управления данным устройством. Например, для драйвера принтера ядро отвечает за взаимодействие с параллельным портом, а формирование управляющих сигналов для принтера осуществляет демон печати lpd, который использует для этого специальную программу-фильтр. Другие примеры драйверов этого типа — драйверы модемов и X-сервер (драйвер видеоадаптера).

1.1.4 Взаимодейсвтие пользователя с устройством

Но надо специально отметить, что во всех трех случаях непосредственное взаимодействие с устройством осуществляет ядро или какой-то модуль ядра. А пользовательские программы взаимодействуют с драйверами устройств через специальные файлы, расположенные в каталоге /dev и его подкаталогах. То есть взаимодействие прикладных программ с аппаратной частью компьютера в ОС Linux осуществляется по схеме 1.1.

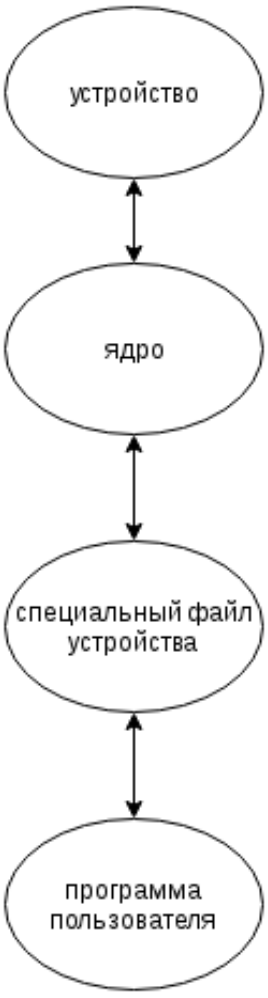

Рисунок 1.1 — Взаимодейсвтие пользователя с устройством

Такая схема 1.1 обеспечивает единый подход ко всем устройствам, которые с точки зрения приложений выглядят как обычные файлы.

1.1.5 Динамическая загрузка драйверов

За основу был взят подход динамической загрузки драйвера, который представляет собой загрузку при помощи отдельных модулей с расширением *.ko (объект ядра).

1.1.6 Загружаемые модули

Одной из хороших особенностей Linux является способность расширения функциональности ядра во время работы. Это означает, что вы можете добавить функциональность в ядро (и убрать её), когда система запущена и работает. Часть кода, которая может быть добавлена в ядро во время работы, называется модулем. Ядро Linux предлагает поддержку довольно большого числа типов (или классов) модулей, включая, но не ограничиваясь, драйверами устройств. Каждый модуль является подготовленным объектным кодом (не слинкованным для самостоятельной работы), который может быть динамически подключен в работающее ядро программой «insmod» и отключен программой «rmmod». [11]

1.1.7 Отличие между модулями ядра и приложениями

Хотя большинство малых и средних приложений выполняют от начала до конца одну задачу, каждый модуль ядра просто регистрирует себя для того, чтобы обслуживать в будущем запросы, и его функция инициализации немедленно прекращается. Иными словами, задача функции инициализации модуля заключается в подготовке функций модуля для последующего вызова; это как будто модуль сказал: «Вот я и вот что я могу делать». Функция выхода модуля вызывается только непосредственно перед выгрузкой модуля. Она сообщает ядру: «Меня больше нет». Такой подход к программированию подобен программируемой обработке событий, но пока не все приложения управляются событиями, как модули ядра. Другое сильное отличие между событийно - управляемым приложением и кодом ядра в функции выхода: в то время как приложение, которое прекращает работу, может быть ленивым при высвобождении ресурсов или избегать очистки всего, функция выхода модуля должна тщательно отменить все изменения, сделанные функцией инициализации, или эти куски останутся вокруг до перезагрузки системы. Возможность выгрузить модуль является одной из тех особенностей подхода модуляризации, потому что это помогает сократить время разработки; можно тестировать последовательные версии новых драйверов, не прибегая каждый раз к длительному циклу выключения/перезагрузки. Приложение может вызывать функции, которые не определены: стадия

линковки разрешает (определяет) внешние ссылки, используя соответствующие библиотечные функции. Printf является одной из таких вызываемых функций и определена в libc. Модуль, с другой стороны, связан только с ядром и может вызывать только те функции, которые экспортированы ядром, нет библиотек для установления связи. Например, функция printk, является версией printf, определённой в ядре и экспортированной для модулей. Она ведёт себя аналогично оригинальной функции с небольшими отличиями, главным из которых является отсутствие поддержки плавающей точки. Рисунок 1.2 показывает, как используются в модуле вызовы функций и указатели на функции, чтобы добавить ядру новую функциональность. [11]

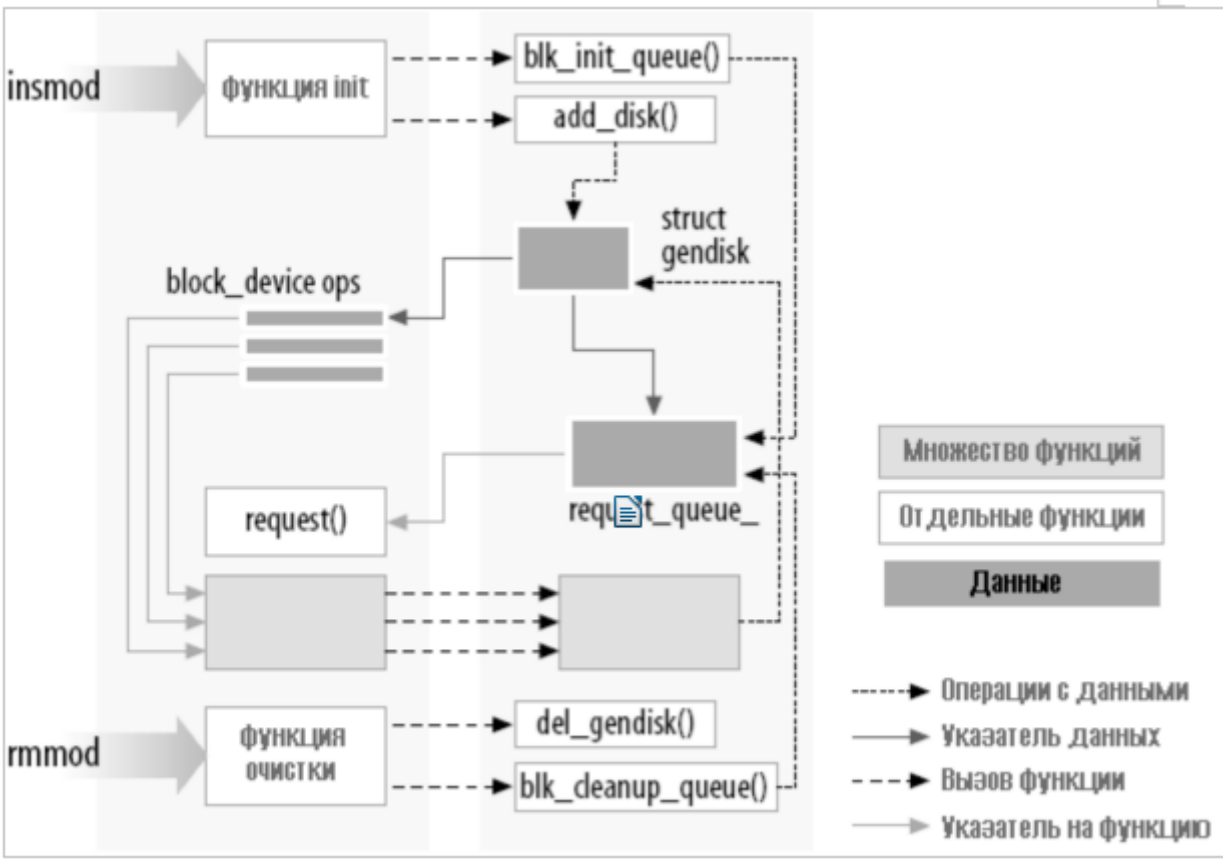

Рисунок 1.2 — Связи модуля в ядре

Файлы исходников никогда не должны подключать обычные заголовочные файлы, потому что нет библиотеки, связанной с модулями, и очень специальные ситуации будут только исключениями. Только функции, которые фактически являются частью самого ядра, могут быть использованы в модулях ядра. Всё относящееся к ядру объявлено в заголовках, находящихся в дереве исходных текстов ядра; наиболее часто используемые заголовки живут в include/linux и include/asm, но есть и другие подкаталоги в папке include для содержания материалов, связанных со специфичными подсистемами ядра. Ещё одно важное различие между программированием ядра и прикладным программированием в том, как каждое окружение обрабатывает ошибки: в то время, как ошибка сегментации является безвредной при разработке

приложений и всегда можно использовать отладчик для поиска ошибки в исходнике, ошибка ядра убивает по крайней мере текущий процесс, если не всю систему.

1.1.8 Пространство пользователя и пространство ядра

Модули работают в пространстве ядра, в то время как приложения работают в пользовательском пространстве. Это базовая концепция теории операционных систем.

На практике ролью операционной системы является обеспечение программ надёжным доступом к аппаратной части компьютера. Кроме того, операционная система должна обеспечивать независимую работу программ и защиту от несанкционированного доступа к ресурсам. Решение этих нетривиальных задач становится возможным, только если процессор обеспечивает защиту системного программного обеспечения от прикладных программ.

Каждый современный процессор позволяет реализовать такое поведение. Выбранный подход заключается в обеспечении разных режимов работы (или уровней) в самом центральном процессоре. Уровни играют разные роли и некоторые операции на более низких уровнях не допускаются; программный код может переключить один уровень на другой только ограниченным числом способов. Unix системы разработаны для использования этой аппаратной функции с помощью двух таких уровней. Все современные процессоры имеют не менее двух уровней защиты, а некоторые, например семейство x86, имеют больше уровней; когда существует несколько уровней, используются самый высокий и самый низкий уровни. Под Unix ядро выполняется на самом высоком уровне (также называемым режимом супервизора), где разрешено всё, а приложения выполняются на самом низком уровне (так называемом пользовательском режиме), в котором процессор регулирует прямой доступ к оборудованию и несанкционированный доступ к памяти. Unix выполняет переход из пользовательского пространства в пространство ядра, когда приложение делает системный вызов или приостанавливается аппаратным прерыванием. Код ядра, выполняя системный вызов, работает в контексте процесса - он действует от имени вызывающего процесса и в состоянии получить данные в адресном пространстве процесса. Код, который обрабатывает прерывания, с другой стороны, является асинхронным по отношению к процессам и не связан с каким-либо определённым процессом.

Ролью модуля является расширение функциональности ядра; код модулей выполняется в пространстве ядра. Обычно драйвер выполняет обе задачи, изложенные ранее: некоторые функции в модуле выполняются как часть системных вызовов, а некоторые из них отвечают за обработку прерываний.

1.2 Аудио разъемы

Разъём TRS (аббревиатура от англ. Tip, Ring, Sleeve — кончик, кольцо, гильза; подразумевается форма контактов на штекере) — распространённый разъём для передачи аудиосигнала. [12]

Однако, наверное мало кто из тех, кто сталкивался с этими разъемами в быту слышал, что они называются TRS, а все потому что у нас прижилось другое название. Его часто называют просто «джек», и это никак не связано с какой-нибудь личностью по имени Джек: на самом деле с английского языка слово «jack» переводится как «гнездо». Причем джеком правильно называть именно гнездо, то есть куда подключается, а это разъём-мама на панели, то есть на системном блоке или другом устройстве, а разъем-папу называют plug, что и переводится как «штекер». [13]

Джек (jack) - это разъем диаметром 1/4 дюйма (6,3 мм). Применяется в музыкальном оборудовании, чаще всего Вы с ними встретитесь при использовании:

а) микрофонов для любителей (чаще всего это караоке или для домашней записи)

б) электрогитар, бас-гитар, педалей («примочек»), усилителей для гитары.

в) профессиональных наушников

г) профессиональных звуковых плат

Мини-джек (mini-jack) - разъём диаметром 3,5 мм. По сравнению с джеком действительно «мини», и используется в устройствах, где размер действительно важен. Его вы можете встретить, купив:

а) наушники (обычные наушники для плеера, к примеру)

б) компьютерную акустичекую систему (её называют динамики или же колонки)

в) гарнитуру

г) плеер

д) звуковую плату потребительского уровня

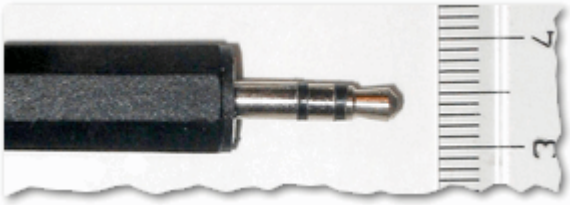

Рисунок 1.3 — Разъем мини-jack

На ноутбуках есть два вида разъемов, предназначеных для подключения акустической системы и микрофона. Они показаны на Рисунке 1.5. Для подключения

Рисунок 1.4 — Гарнитурный штекер

гарнитуры с стандартным совмещенным штекером (Рисунок 1.4) в левый разъем (Рисунок 1.5) используется переходник, показанный на рисунке 1.6.

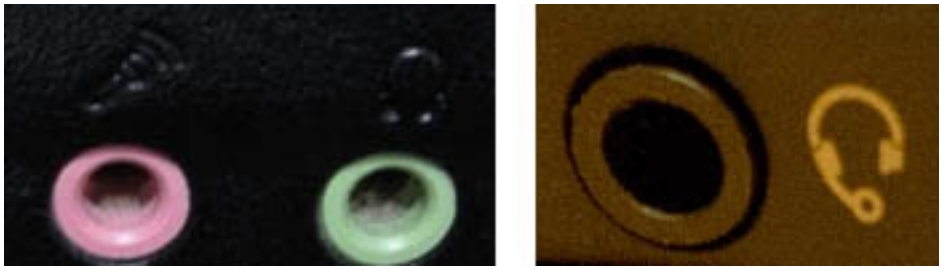

Рисунок 1.5 — Разъемы ноутбука

1.3 Звуковая архитектура

Рисунок 1.7 показывает подключение звука на ПК-совместимой системе. Аудио контроллер Южного моста, а также внешний кодек, подключённый к аналоговой звуковой схеме. [14]

1.3.1 Звуковая подсистема Linux

1.3.2 Open Sound System

Open Sound System (OSS) — унифицированный драйвер для звуковых карт и других звуковых устройств в различных UNIX-подобных операционных системах.

Рисунок 1.6 — Переходник для подключения гарнитурных наушников к обычной звуковой карте

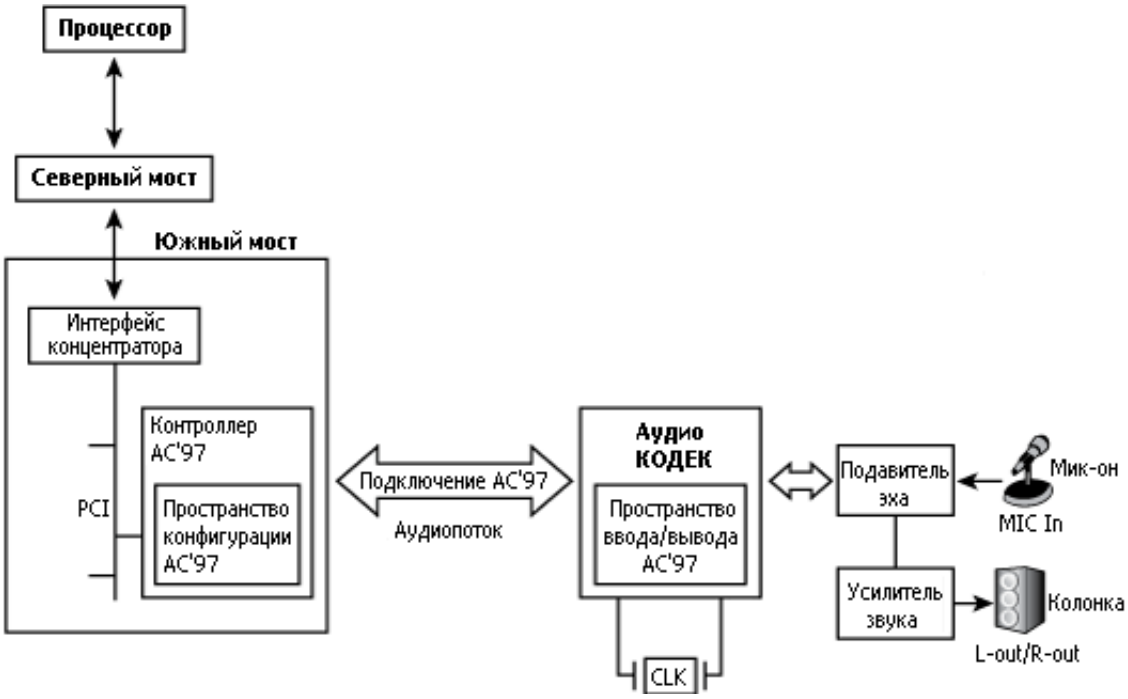

Рисунок 1.7 — Звук в среде ПК

OSS основан на Linux Sound Driver и в настоящее время работает на большом числе платформ: Linux, FreeBSD, OpenSolaris и т. д. [15]

/dev/dsp и /dev/audio — основные файлы устройств для цифровых приложений. Любые данные, записанные в эти файлы, воспроизводятся на DAC/PCM/DSP устройстве звуковой карты. Чтение из этих файлов возвращает звуковые данные, записанные с текущего входного источника (по умолчанию это Микрофонный вход).

При чтении из /dev/dsp мы получаем несжатый аудиопоток с микрофона компьютера через вход звуковой карты.

1.3.3 ALSA

ALSA (англ. Advanced Linux Sound Architecture, Продвинутая звуковая архитектура Linux) — архитектура звуковых драйверов, а также широкий их набор для операционной системы Linux, призванный сменить Open Sound System (OSS). ALSA тесно связана с ядром Linux. ALSA — программный микшер, который эмулирует совместимость для других слоев. Также предоставляет API для программистов и работает с низкой стабильной задержкой. [16]

ALSA является звуковой подсистемой, выбранной для ядра версии 2.6. Открытая Звуковая Система, OSS, звуковой уровень в ядре версии 2.4, в настоящее время устарел и не рекомендуется к использованию. Для перехода от OSS к ALSA последняя предоставляет эмуляцию OSS, которая позволяет приложениям, использующим API OSS, запускаться без изменений на ALSA. Звуковые ядра Linux, такие как ALSA и OSS, делают аудио приложения независимыми от базового оборудования.

Чтобы понять архитектуру звуковой подсистемы Linux посмотрите на Рисунок 1.8.

Основными частями подсистемы являются:

а) Звуковое ядро, которое является базовым кодом, состоящим из процедур и структур, доступных другим компонентам звукового уровня Linux. Как и уровни ядра, принадлежащие другим драйверным подсистемам, звуковое ядро обеспечивает уровень косвенности, что делает каждый компонент в звуковой подсистеме не зависящим от других. Ядро играет важную роль в экспорте API ALSA вышележащим приложениям. Узлами /dev/snd/*, показанными на Рисунке 1.8, которые создаются и управляются из ядром ALSA, являются: /dev/snd/controlC0 - узел управления (используемый в приложениях для управления уровнем громкости и тому подобному), /dev/snd/pcmC0D0p - устройство воспроизведения (p в конце имени устройства означает playback, воспроизведение), и /dev/snd/pcmC0D0c - записывающее устройство (c в конце имени устройства означает capture, захват). В этих именах устройств целое число после C является номером карты, а после D - номером устройства. ALSA драйвер для карты, которая имеет голосовой кодек для телефонии и стерео кодек для музыки, может экспортировать /dev/snd/pcmC0D0p для чтения аудио потоков, предназначенный для первого, и /dev/snd/pcmC0D1p для качественного музыкального канала для последнего.

б) Драйверы аудио контроллера зависят от оборудования контроллера. Например, для управления аудио контроллером, находящимся в Южном мосте Intel ICH, используется драйвер snd_intel8x0.

в) Интерфейсы аудиокодеков, которые помогают взаимодействию между контроллерами и кодеками. Для кодеков AC'97 используйте snd_ac97_codec и модули ac97_bus.

г) Уровень эмуляции OSS, который выступает в качестве посредника между приложениями, использующими OSS, и ядром с поддерживающим ALSA. Этот уровень экспортирует узлы /dev, изображающие поддержку уровня OSS в ядре версии 2.4. Эти узлы, такие как /dev/dsp, /dev/adsp и /dev/mixer, позволяют приложениям OSS работать поверх ALSA без изменений. Узел OSS /dev/dsp связан с узлами ALSA /dev/snd/pcmC0D0*, /dev/adsp соответствует /dev/snd/pcmC0D1*, а /dev/mixer связан с /dev/snd/controlC0.

д) Интерфейс procfs and sysfs для доступа к информации через /proc/asound/ и /sys/class/sound/.

е) Библиотека ALSA пользовательского пространства, alsa-lib, которая предоставляет объект libasound.so. Эта библиотека упрощает работу программиста приложения ALSA, предлагая несколько готовых процедур для доступа к драйверам ALSA.

ж) Пакет alsa-utils, который включает в себя такие утилиты, как alsamixer, amixer, alsactl и aplay. alsamixer или mixer используются для изменения громкости звуковых сигналов, таких как линейный вход, линейный выход или микрофон, а alsactl - для управления параметрами драйверов ALSA. aplay используется для воспроизведения звука через ALSA.

В данной работе для получения потока байтов от микрофонного разъема будем использовать API, предоставляемые ALSA.

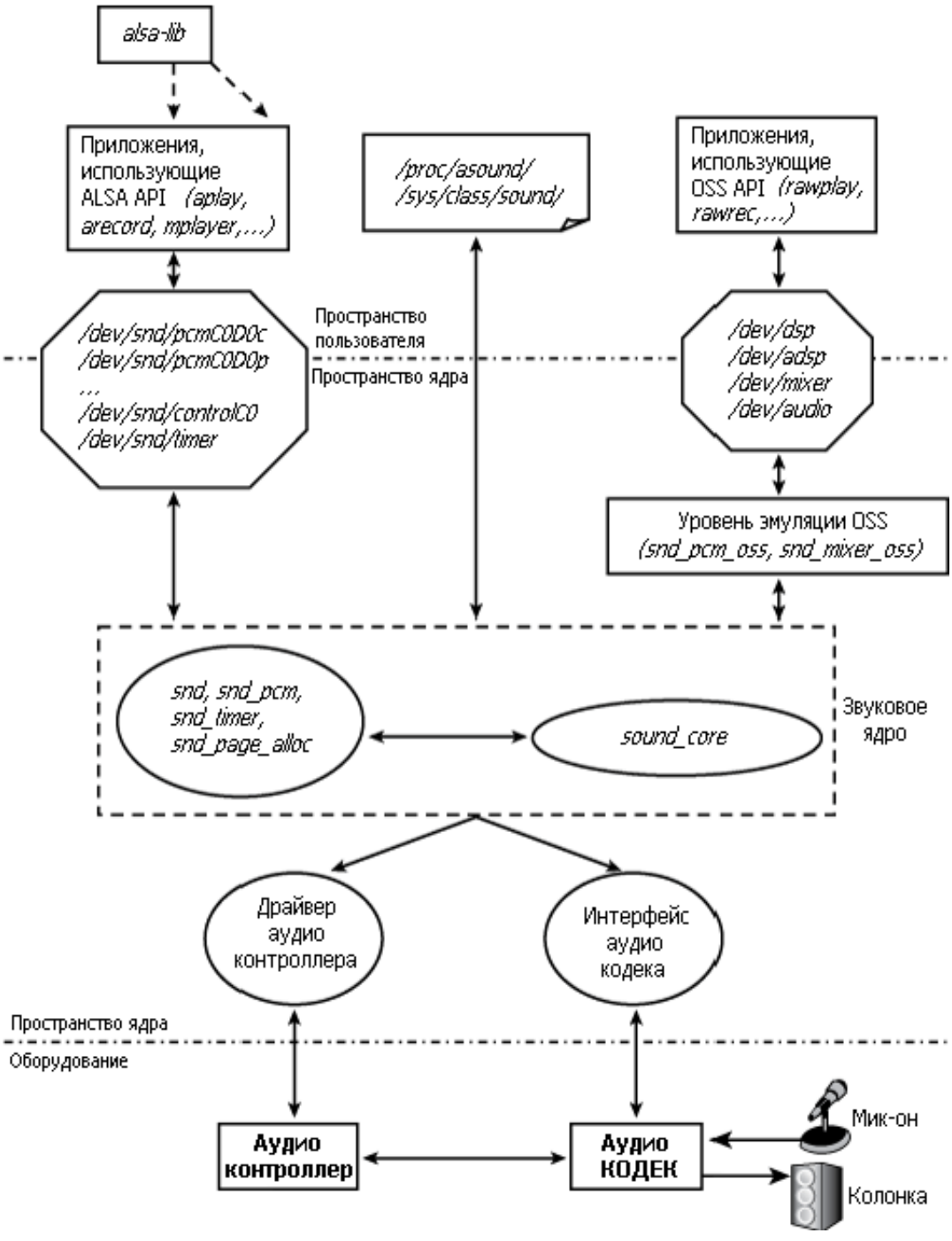

Рисунок 1.8 — Звуковая подсистема Linux (ALSA)

1.4 Исследование сигнала гарнитуры

При помощи ALSA API получим сырой поток байт с входа микрофона, несколько раз нажав на кнопку гарнитуры и построим график, учитывая, что единица данных при считывании входа микрофона - это 2-х байтовое число.

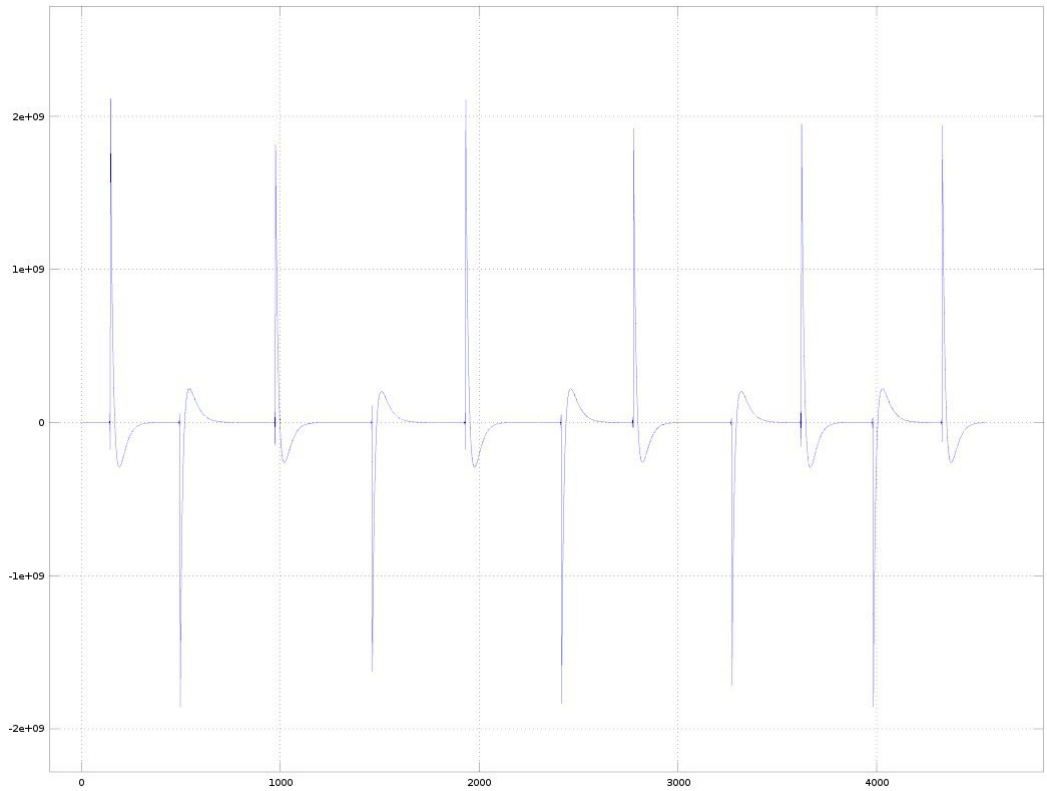

Рисунок 1.9 — Поток байтов при нажатии кнопок гарнитуры

На графике 1.9 мы видим 6 положительных пиков и 5 отрицательных пиков. Положительному пику соответствует нажатие кнопки на гарнитуре. Отрицательному пику соответствует отпускание кнопки гарнитуры.

На листингах 1.1 и 1.2 приведены примеры считанных байтов.

Листинг 1.1 — Положительный пик (в шестнадцатиричной системе)

```
1 fff5 ffed ffe8 fff8 fff7
2 ffe2 ff17 009c fff1 ff25 0179 fb90 0912
3 eded 2fff 7fff 7fff 7fff 7fff 7fff 7fff
4 7fff 7fff 7fff 7fff 7fff 7fff 7fff 7fff
5 7fff 7fff 7f9b 7ee7 778c 7188 6bd5 656d
6 6050 5b76 56de 5183 4d63 49
```

Листинг 1.2 — Отрицательный пик (в шестнадцатиричной системе)

```
1 fb6 ffb8 ffb6 ffb0 ffb7
2 ffc1 ffc5 ffc4 ffa3 ffa3 00b7 fd26 0324
3 fc9e 00d2 0801 8001 8001 8001 8001 8001
4 8001 8001 8001 8001 8001 8001 8001 8001
```

```
5 8001 8001 8001 80e0 8151 8870 8e3d 94c1
6 99f9 9ef7 a3bb a8be ac36 b2b0 b37c bc8c
7 c638 c039 c2be c6a2 c
```

Можно заметить, что нажатию кнопки гарнитуры соответствует несколько раз идущих подряд число $7fff_{16} = 32767_{10}$. А отпусканию кнопки гарнитуры соответствует несколько раз идущих подряд число $8001_{16} = -32768_{10}$.

2 Конструкторский раздел

На Рисунке 1.1 показана структура разрабатываемого проекта.

а) программа пользователя - программа, использующая ALSA API, определяющая нажатие на гарнитуре

б) специальный файл устройства - создан загружаемым модулем ядра для взаимодействия программы пользователя и модулем ядра.

в) ядро - на этом уровне работает модуль ядра, получающий информацию от программы пользователя через созданный модулем специальный файл устройства.

г) устройство - клавиатура, индикаторы которой будет включать/выключать модуль ядра.

2.1 Протокол взаимодействия модуля ядра и программы пользователя

Для взаимодействия модуля ядра и программы пользователя модулем ядра создан специальный файл, куда программа пользователя будет писать сообщения, а модуль ядра будет обрабатывать эти сообщения.

Сообщение - это нуль-терминированная строка (C-строка или ASCIZ-строка).

$$N\backslash 0$$

Где N - число, означающее номер нажатой кнопки. В данной работе поддерживается только один вид кнопки гарнитуры, поэтому всегда $N = 1$.

2.2 Модуль ядра

Алгоритм действий модуля ядра, при получении сообщения, показан на Рисунке 2.1

msg - сообщение из пространства пользователя. Так как поддерживается только один тип кнопки гарнитуры, первый символ сообщения msg сравнивается с единицей.

$my\_tasklet\_function$ - функция отложенного действия, запускающая первый раз таймер с функцией, изменяющей состояние LED индикатора клавиатуры.

Алгоритм отложенного действия показан на Рисунке 2.2

my_timer_func - функция, изменяющая состояние LED индикатора клавиатуры.

sos - массив числел, означающих длительность включения индикатора клавиатуры.

pos - текущий индекс массива *sos*.

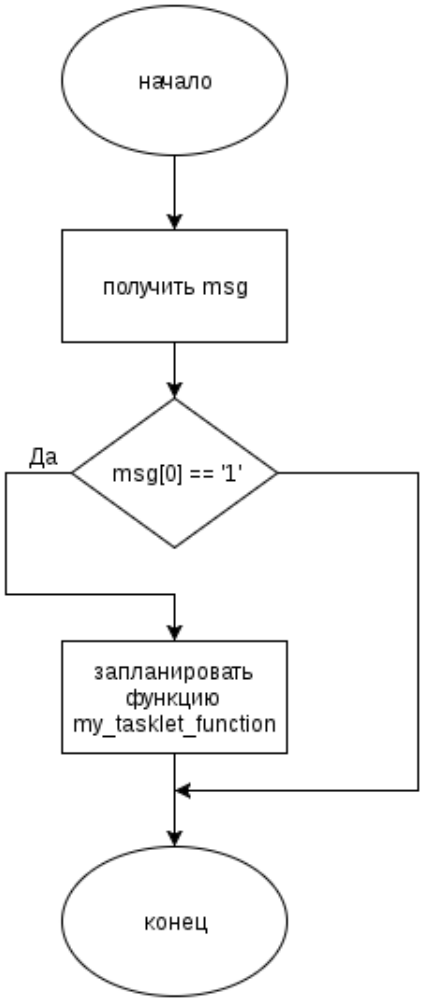

Рисунок 2.1 — Алгоритм работы модуля ядра

run - флаг, означающий, что в данный момент сообщение sos еще воспроизводится.

Но последовательность действий 2.1 может привести к печальным последствиям, если после выполнения строчки №3 и до выполнения строчки №4 Linux решит прекратить выполнение данного кода, и начнет выполнение другой задачи, а этой другой задачей м. б. этот же код. Тогда возможна ситуация, когда в ядре активны оба процесса, которые считают, что они выполняются в единственном экземпляре. Так как в ядре этот код работает с аппаратурой, то одновременное выполнение одной и той же программы приведет к kernel panic.

Листинг 2.1 — Опасный код

```
1 if (run > 0) {
2   return ;
3 }
4 run++;
```

Алгоритм функции таймера показан на Рисунке 2.3

pstatus - флаг включения индикатора клавиатуры.

Рисунок 2.2 — Алгоритм отложенного действия

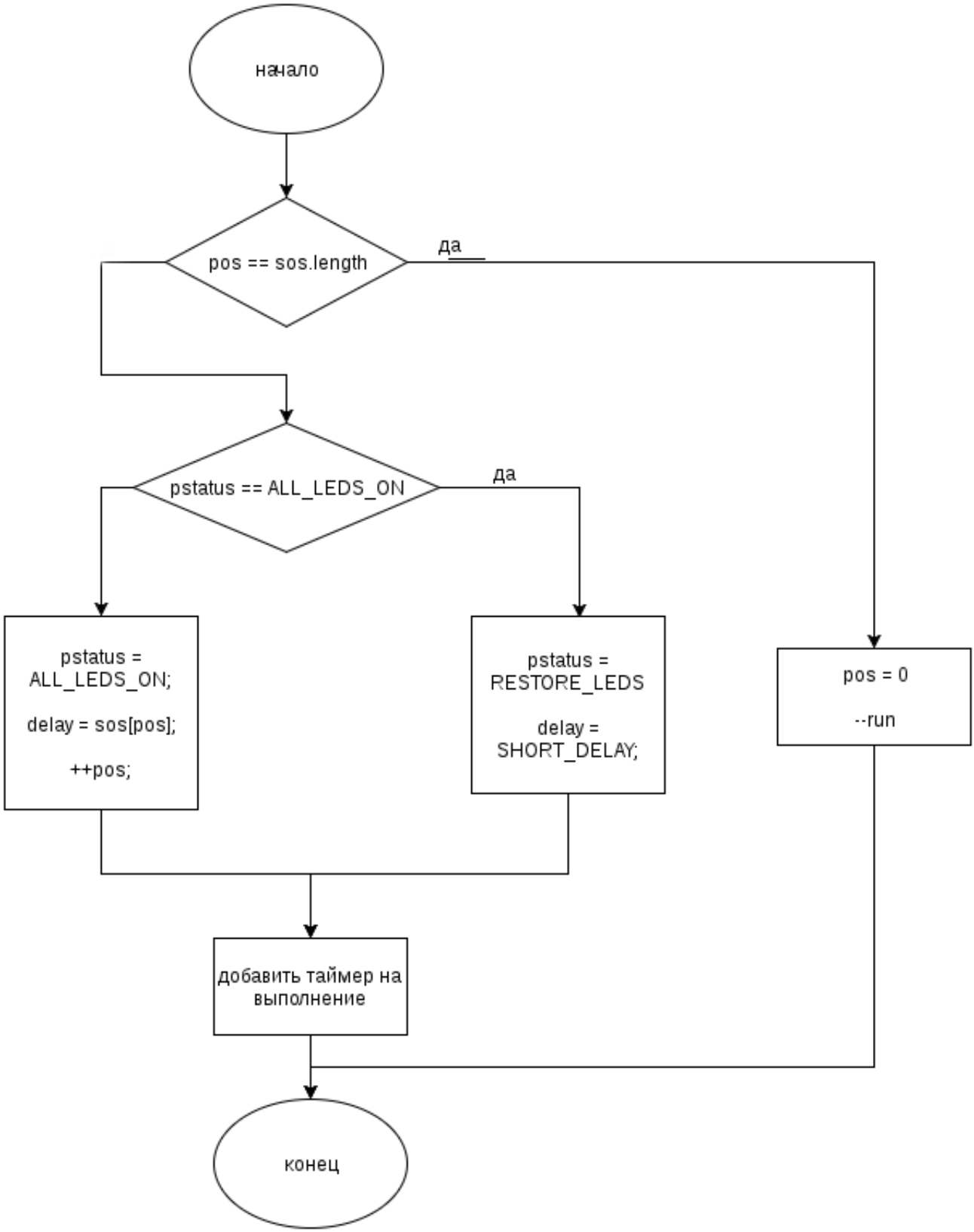

Рисунок 2.3 — Алгоритм функции таймера

3 Технологический раздел

3.1 Модуль ядра

Сам модуль ядра написана на языке C с использованием встроенного в ОС Linux компилятора GCC. Выбор языка основан на том, что исходный код ядра, предоставляемый системой, написан на C, и использование другого языка программирования в данном случае было бы нецелесообразным.

Листинг 3.1 — Функция, обрабатывающая запись в специальный файл

```
1  int module_dev_write(struct file *sp_file,const char __user *buf, size_t
       size, loff_t *offset) {
2      printk(KERN_INFO "module_dev: called write %d\n",size);
3
4      copy_from_user(msg, buf, size);
5      length_msg = size;
6
7      is_empty = 0;
8
9      int event;
10
11     if (msg[0] == '1') {
12         event = 1;
13         tasklet_schedule(&my_tasklet);
14     } else {
15         event = 0;
16     }
17
18     printk(KERN_INFO "module_dev: write event = %d\n", event);
19
20     return size;
21 }
```

3.2 Работа с индикаторами клавиатуры

Доступ к индикаторам клавиатуры реализован через виртуальную консоль, доступную в ядре через библиотеку linux/console_struct.h.

Отложенное действие реализовано через тасклет.

Для исключения возможности одновременного выполнения двух копий одной и той же программы в ядре используется mutex.

Листинг 3.2 — Поставновка таймера на выполнение

```
1  static DECLARE_MUTEX(mutex);
2
3
4  void my_tasklet_function(void) {
```

```
5
6       down(mutex);
7
8       int i;
9       printk(KERN_INFO "kbleds: loading\n");
10      printk(KERN_INFO "kbleds: fgconsole is %x\n", fg_console);
11
12      for (i = 0; i < MAX_NR_CONSOLES; i++) {
13              if (!vc_cons[i].d)
14                      break;
15              printk(KERN_INFO "poet_atkm: console[%i/%i] #%i, tty %lx\n", i,
16                      MAX_NR_CONSOLES, vc_cons[i].d->vc_num,
17                      (unsigned long)vc_cons[i].d->port.tty);
18      }
19
20      printk(KERN_INFO "kbleds: finished scanning consoles\n");
21      my_driver = vc_cons[fg_console].d->port.tty->driver;
22      printk(KERN_INFO "kbleds: tty driver magic %x\n", my_driver->magic);
23
24      /*
25       * Set up the LED blink timer the first time
26       */
27      init_timer(&my_timer);
28      my_timer.function = my_timer_func;
29      my_timer.data = (unsigned long)&my_timer_func_data;
30      my_timer.expires = jiffies + START_DELAY;
31      add_timer(&my_timer);
32
33      return;
34  }
```

Листинг 3.3 — Функция таймера, включающая/выключающая индикаторы

```
1  #define SHORT_DELAY   HZ / 5
2  #define LONG_DELAY    3 * HZ / 5
3  #define START_DELAY   HZ
4
5  #define ALL_LEDS_ON   0x07
6  #define RESTORE_LEDS  0xFF
7
8  int sos[9] = {SHORT_DELAY, SHORT_DELAY, SHORT_DELAY, LONG_DELAY,
       LONG_DELAY, LONG_DELAY, SHORT_DELAY, SHORT_DELAY, SHORT_DELAY};
9  int pos = 0;
10
11 static void my_timer_func(unsigned long ptr) {
12      int *pstatus = (int *)ptr;
13
```

```
14      if (pos == 9) {
15          pos = 0;
16          del_timer(&my_timer);
17          (my_driver->ops->ioctl) (vc_cons[fg_console].d->port.tty,
        KDSETLED, RESTORE_LEDS);
18          up(mutex);
19          return;
20      }
21
22      printk(KERN_INFO "kbleds: pstatus = %d\n", *pstatus);
23
24      int delay;
25
26      if (*pstatus == ALL_LEDS_ON) {
27          *pstatus = RESTORE_LEDS;
28          delay = SHORT_DELAY;
29      } else {
30          *pstatus = ALL_LEDS_ON;
31          delay = sos[pos];
32
33          ++pos;
34      }
35
36      (my_driver->ops->ioctl) (vc_cons[fg_console].d->port.tty, KDSETLED,
        *pstatus);
37
38      my_timer.expires = jiffies + delay;
39
40      add_timer(&my_timer);
41  }
```

3.3 Программа пользователя

Программа пользователя также написана на языке C, т.к. ALSA предоставляет свой API для этого языка.

Листинг 3.4 — Подготовка к считыванию входа микрофона

```
1  #define ALSA_PCM_NEW_HW_PARAMS_API
2
3  #include <alsa/asoundlib.h>
4
5    /* Open PCM device for recording (capture). */
6    rc = snd_pcm_open(&handle, "default", SND_PCM_STREAM_CAPTURE, 0);
7
8    if (rc < 0) {
9      fprintf(stderr,
```

```
10              "unable to open pcm device: %s\n",
11              snd_strerror(rc));
12      exit(1);
13    }
14
15    /* Allocate a hardware parameters object. */
16    snd_pcm_hw_params_alloca(&params);
17
18    /* Fill it in with default values. */
19    snd_pcm_hw_params_any(handle, params);
20
21    /* Set the desired hardware parameters. */
22
23    /* Interleaved mode */
24    snd_pcm_hw_params_set_access(handle, params,
25                        SND_PCM_ACCESS_RW_INTERLEAVED);
26
27    /* Signed 16-bit little-endian format */
28    snd_pcm_hw_params_set_format(handle, params,
29                                SND_PCM_FORMAT_S16_LE);
30
31    /* Two channels (stereo) */
32    snd_pcm_hw_params_set_channels(handle, params, 1);
33    // snd_pcm_hw_params_set_channels(handle, params, 2);
34
35    /* 44100 bits/second sampling rate (CD quality) */
36    val = 44100;
37    snd_pcm_hw_params_set_rate_near(handle, params, &val, &dir);
38
39    /* Set period size to 32 frames. */
40    frames = 32;
41    snd_pcm_hw_params_set_period_size_near(handle, params, &frames, &dir);
42
43    /* Write the parameters to the driver */
44    rc = snd_pcm_hw_params(handle, params);
45
46    if (rc < 0) {
47      fprintf(stderr,
48              "unable to set hw parameters: %s\n",
49              snd_strerror(rc));
50      exit(1);
51    }
52
53    /* Use a buffer large enough to hold one period */
54    snd_pcm_hw_params_get_period_size(params, &frames, &dir);
55
56    /* 2 bytes/sample, 2 channels */
```

```
57    size = frames * 2;
58
59    buffer = (char *) malloc(size);
60
61    snd_pcm_hw_params_get_period_time(params, &val, &dir);
```

Листинг 3.5 — Обнаружение нажатия

```
1     MIN_NECESSARY_MARKS = 8;
2     flag = 0;
3
4     while (1) {
5         rc = snd_pcm_readi(handle, buffer, frames);
6
7         for (i = 0; i <  size - 1; ++i) {
8           if (buffer[i] == 0x7f && buffer[i + 1] == (char)0xff) {
9             ++flag;
10            ++i;
11
12            if (flag > MIN_NECESSARY_MARKS) {
13                system("echo 1 > /proc/module_dir/dev");
14                flag = 0;
15            }
16          } else {
17                flag = 0;
18          }
19        }
20    }
```

Заключение

В результате выполнения данной курсовой работы был изучен подход динамического написания драйверов под ОС Linux, работа с звуковой системой и устройством ввода (гарнитурой).

Исследован сигнал, получаемый от гарнитуры на вход микрофона компьютера.

Разработана программа, обнаруживающая событие нажатия кнопки гарнитуры и протокол взаимодейсвтия этой программы с модулем ядра.

Разработан модуль ядра, работающий с клавиатурой, который включает/выключает LED индикаторы клавиатуры.

Список использованных источников

1. Какие компьютеры бывают, виды и типы компьютеров. — http://procomputer.su/osnovy-kompyutera/9-kakie-kompyutery-byvayut-vidy-i-tipy-kompyuterov. — [Online; accessed 25-October-2016].

2. *Wikipedia*. Устройства ввода-вывода. — https://ru.wikipedia.org/wiki/%D0%A3%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE_%D0%B2%D0%B2%D0%BE%D0%B4%D0%B0-%D0%B2%D1%8B%D0%B2%D0%BE%D0%B4%D0%B0. — 2016. — [Online; accessed 25-October-2016].

3. *Wikipedia*. Микрофон. — https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D1%84%D0%BE%D0%BD. — 2016. — [Online; accessed 25-October-2016].

4. *Wikipedia*. Акустическая система. — https://ru.wikipedia.org/wiki/%D0%90%D0%BA%D1%83%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0. — 2016. — [Online; accessed 25-October-2016].

5. *Wikipedia*. Мобильная операционная система. — https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B1%D0%B8%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0. — 2016. — [Online; accessed 25-October-2016].

6. *Krishnan, Yashwanth*. How to capture key events from bluetooth headset with android. — http://stackoverflow.com/questions/17819649/how-to-capture-key-events-from-bluetooth-headset-with-android/17983446#17983446. — [Online; accessed 25-October-2016].

7. Gambiarra em casa. — http://gambiarraemcasa.blogspot.ru/2014/06/osciloscopio-usb-usb-oscilloscope.html. — [Online; accessed 25-October-2016].

8. Morse code kernel panics. — https://lwn.net/Articles/21858/. — 2003. — [Online; accessed 25-October-2016].

9. РАСШИФРОВКА ЗВУКОВЫХ СИГНАЛОВ BIOS. — http://www.allmbs.ru/bios-01.html. — 2012. — [Online; accessed 25-October-2016].

10. *Костромин*. Linux для пользователя / Костромин. — БХВ-Петербург, 2002.

11. *Jessica McKellar Alessandro Rubini, Jonathan Corbet Greg Kroah-Hartman*. Linux Device Drivers / Jonathan Corbet Greg Kroah-Hartman Jessica McKellar, Alessandro Rubini. — O'Reilly Media, 2016.

12. *Wikipedia*. Разъём TRS. — https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%8A%D1%91%D0%BC_TRS. — 2016. — [Online; accessed 25-October-2016].

13. Разъем джек и мини-джек - Часть 1 - Как подключить микрофон к компьютеру. — http://kkg.by/kak-ustroen-computer/29-razem-dzhek-i-mini-dzhek-kak-podklyuchit-mikrofon-k-kompyuteru.html. — [Online; accessed 25-October-2016].

14. *Cox, Alan.* Essential Linux Device Drivers / Alan Cox.

15. *Wikipedia.* Open Sound System. — https://ru.wikipedia.org/wiki/Open_Sound_System. — 2016. — [Online; accessed 25-October-2016].

16. *Wikipedia.* ALSA. — https://ru.wikipedia.org/wiki/ALSA. — 2016. — [Online; accessed 25-October-2016].