Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования

«Московский физико-технический институт

(государственный университет)»

Факультет управления и прикладной математики

Кафедра теоретической и прикладной информатики

**Проектирование и разработка подсистемы виртуализации звука в Parallels Desktop**

Выпускная квалификационная работа

(магистерская диссертация)

Направление подготовки: 03.04.01 Прикладная математика и физика

Выполнил:

студент 973 группы Бондарь Артем Олегович

Научный руководитель:

д. ф.-м.н, профессор Тормасов Александр Геннадьевич

Научный консультант:

Преподаватель Мелехова Анна Леонидовна

Москва 2015

Оглавление

1.2 Виртуализация мультимедиа 5

1.3 Системы реального времени 6

1.3 Проблемы виртуализации средств поддержки мультимедиа 11

2. Постановка задачи 16

3. Обзор существующих решений работы со звуком 19

3.1 Архитектурные подходы к виртуализации аудио потока 19

3.2 Существующие аудио библиотеки 21

3.2.1 Portaudio 22

3.2.2 RTAudio 23

3.2.3 JUCE 24

3.2.4 Core Audio 25

3.3 Заключение 27

4. Ход работ 28

4.1. Сравнительный анализ архитектурных подходов по виртуализации аудио устройств 28

4.1.1 Формальная модель аудио потока 28

4.1.2 Оценка мультипликативной константы модели 31

4.1.3 Интерпретация модели как Марковской цепи 33

4.1.4 Оценка метрики, характеризующей качество аудио потока 34

4.1.5 Сравнение архитектурных подходов в рамках приведенной формализации 36

4.2 Архитектура потоковой виртуализации аудио данных 39

4.3 Разработка аудио библиотеки 41

5. Оценка результатов работы 44

6. Заключение 45

7. Список использованной литературы 47

1. Введение в проблематику1.1 Виртуализационные технологии.

Парадигма облачных вычислений в последние годы обрела высокую популярность в индустрии информационных технологий, что, в свою очередь, поспособствовало все более бурному развитию технологий аппаратной виртуализации [1]. Эта технология позволяет одновременно исполняться нескольким операционным системам в рамках одной физической машины. Такой сценарий работы системы реализуется посредством изоляции операционных систем от реального оборудования при помощи дополнительного программного слоя абстракции - гипервизора [2, 3]. Его задача - эмулировать поведение реальных устройств таким образом, что бы операционные системы могли работать так же, как если бы они имели эксклюзивный доступ к оборудованию.

Существующие конфигурации виртуализированных систем могут быть разделены на два широких класса: гипервизоры работающие без использования сервисной (хостовой) ОС и гипервизоры использующие сервисную ОС (гипервизоры первого и второго типа [38]). Гипервизоры первого типа по сути являются микро-ОС, так как сами могут работать с оборудованием, обслуживая запросы изолированных ОС (гостевые ОС). Примером такого гипервизороа являются VMWare ESXi [4]. Гипервизоры же второго типа не содержат в себе драйверов для работы с аппаратурой и исполняются как сервис хостовой ОС, которая и обслуживает все запросы по работе с оборудованием, приходящие от гостевых ОС. Такой метод виртуализации реализован в проектах KVM [5], VMWare Workstation [6], Parallels Desktop [7]. В рамках этой работы мы остановим свое внимание на гипервизорах второго типа.

Виртуализированные системы имеют ряд свойств, обеспечивших высокую популярность решений по аппаратной виртуализации в IT индустрии [39]. Среди них особенно стоит выделить высокий уровень утилизации ресурсов, надежность и гибкость в управления системой. Действительно, виртуальные машины изолированы друг от друга гипервизором и в любой момент могут быть скопированы или перемещены между реальными машинами, в том числе и без остановки (технология живой миграции [8]). Это позволяет реализовывать множество сценариев, повышающих надежность системы и ее эффективность.

Но при этом необходимо понимать, что виртуализационное ПО должно эффективно эмулировать очень широкий спектр аппаратуры, что подчас требует от виртуальной машины работы с учетом ряда ограничений, вызванных спецификой работы физических устройств. Одной из областей, в которой подобная специфика вносит ряд серьезных затруднений при написании виртуализационного ПО, это виртуализация средств мультимедиа.

## 1.2 Виртуализация мультимедиа

 Вопросы виртуализации устройств мультимедиа и в частности аудио устройств приобрели особенную актуальность с развитием разработок виртуальных машин, рассчитанных для использования на рынке десктопной виртуализации (продукты вроде Virtual Box, Parallels Desktop и VMWare Workstation). В этом случае основной целью использования виртуальной машины является запуск приложений, не совместимых с используемой ОС. Таким приложением может являться приложение по работе с мультимедиа, и в коммерческом виртуализационном решении качество эмуляции мультимедиа устройств имеет большое значение.

 В то же время цифровая аудио система имеет жесткие временные ограничения: если мы опаздаем с генерацией фрагмента аудио данных, то аудиопоток будет прерван и пользователь услышит искажения сигнала. Иными словами аудио система является системой реального времени, то есть она должна гарантировать выполнения вычислений с задержкой, не превышающей заданную. Имплементация таких систем сопряжена с рядом трудностей даже в рамках невиртуализированной среды, и тем более является проблемной в случае использования технологий виртуализации. В следующих главах мы рассмотрим причины этих затруднений и рассмотрим особенности написания таких систем.

## 1.3 Системы реального времени

Рассмотрим некоторую вычислительную систему, задачей которой является исполнение некоторой процедуры в ответ на внешнее событие. Если результат этих вычислений актуален и ценен только в течении определенного интервала времени с момента получения события (deadline), то такая система называется системой реального времени [9]. Разделяют такие системы на два подтипа:

- Система мягкого реального времени - система в которой ответ тем менее актуален, чем дольше он запоздал к deadline.

- Система жесткого реального времени - система в котором ответ теряет актуальность полностью после опоздания к deadline.

Примерами систем жесткого реального времени являются цифровые контроллеры в двигателе автомобиля, где запаздывание ответа может привести к автокатастрофе.

К системам мягкого реального времени причисляют большинство цифровых бытовых мультимедиа систем, в частности аудио и видео системы, в которых запаздывания приводят к искажению воспроизводимого сигнала, которые легко улавливаются человеческими органами восприятия.

Проблемы разработки систем жесткого реального времени мы оставим в рамках этой работы без должного внимания, в связи с тем, что это большая область, не имеющая прямого отношения к вопросам, поставленным в этой работе, и сконцентруемся на вопросах написания систем мягкого реального времени.

Написание систем мягкого реального времени является хорошо изученной областью, и современные ОС умеют давать гарантии исполнения пользовательского кода с указанными ограничениями в случае выполнения следующих правил [13]:

- Код, чувствительный к задержкам, должен исполняться в системе с максимальным приоритетом

- Запрещено использовать системные вызовы, потенциально блокирующие исполнение (такие как обращения к файловой системе и сети, выделение/освобождение памяти)

 - Запрещено использовать блокирующие примитивы синхронизации. Исключением является использование вызовов типа trylock(), исполняющихся за ограниченное сверху время

Как уже было указано выше, аудио система ОС является системой реального времени. Такая классификация применима в связи с тем, что в случае, если генератор аудио потока не успеет сгенерировать пакет данных к моменту, когда те должны будут начать воспроизводиться, то аудио контроллер вынужден будет остановить воспроизведение звука на некоторый промежуток времени, равный времени запаздывания, что будет звучать для слушателя как щелчок. В случае нескольких таких опазданий, разделенных короткими промежутками времени звук будет сильно искажен лишним шумом, природа которого заключается в резком изменении спектральной картины на границах выпадений аудио данных цифрового потока. Широкое использование персональных компьютеров в музыкальной индустрии приводит к тому, что они часто оказываются подключенными к усиливающей аппаратуре, для которой такие резкие звуковые скачки могут быть нежелательны, а иногда и опасны. В случае же записи выпадение данных приводит к искаженному записанному сигналу, что в случае профессионального использования неприемлемо.

Принимая во внимание правила написание кода реального времени исполнения, мы можем построить концептуальную схему работы аудио системы (Рис 1):

 -  Для аудио генератора выделяется необходимая память, загружаются все необходимые ресурсы с жесткого диска, или из сети, либо инициируется процесс асинхронной подкачки данных в буфер.

 -  Далее запускается процесс воспроизведения: аудио контроллер периодически запрашивает у звукового модуля данные (как правило при помощи функции обратного вызова) и аудио модуль либо генерирует их, либо берет уже подгруженные в память данные (если они уже успели загрузиться), либо оповещает звуковой контроллер об отсутствии аудио данных.

Рис 1. Архитектура аудиопотока (output) в современных ОС

 Для входящего потока аудио данных (input) работают та же схема, но меняется направление движения аудио данных.

 Как видно из описания, следование правилам написания системы мягкого реального времени не может дать полной гарантии непрерывности звукового потока в операционных системах с многозадачностью (каковыми являются большинство современных ОС). И тем не менее такой подход обеспечивает приемлемое качество звука - это становится возможным благодаря продуманным политикам выставления задержки (latency) в воспроизведении аудио потока.

 Задержка между началом генерации аудио данных и стартом их воспроизведения (либо между началом записи звука и началом приема аудио данных записывающим приложением), так же называемая пребуферизацией, - это основной инструмент снижения рисков нарушения непрерывности аудио потока в современных операционных системах [14, 15, 16]. И это кажется вполне естественным - чем выше задержка, тем больше готовых для воспроизведения данных буферизировано, а значит в случае, если генератор по каким-либо причинам нарушит расписание синтеза новых данных, то у него будет возможность наверстать упущенное, пока аудио подсистема проигрывает/обрабатывает уже готовые для этого данные (более подробно этот вопрос будет рассмотрен в разделе 4.1). С другой стороны, задержки в воспроизведении звука свыше 180 мс в случае воспроизведения мультимедиа [17] и 400 мс для VoIP систем [18] уже различимы человеком и неприемлемы. Для систем работающих с цифровой обработкой звука этот параметр еще ниже: при задержках в 2-5 мс тембральная картина меняется, а при превышении порога в 10 мс эта задержка становится явно различимой, и не позволяет качественно работать с полученным аудио сигналом [19]. Для достижения высокого качества работы аудио системы разработчики операционных систем стараются выдерживать баланс между стабильностью аудио потока и величиной задержки. Для современных операционных систем характерны следующие минимальные величины задержки [14]:

 - OSX - 2мс

 - Windows - зависит от реализации драйвера (8 - 400мс). При использовании драйверов ASIO [20] - снижена до 1 - 2мс

 - Unix-до10мс

Но при этом очень важно понимать, что до этого момента мы рассматривали задержку между началом генерации данных и стартом их воспроизведения (так же известную как величина пребуферизации), тогда как общая величина задержки в системе включает в себя еще и составляющую, связанную с не мгновенностью передачи контрольных сообщений (далее по тексту - нежелательная задержка). Природа этой составляющей заключается в том, что контрольные сигналы как правило даются пользователем системы через устройства ввода/вывода, обработка которых ведется с гораздо более низким приоритетом и меньшим вниманием к скорости отзывчивости. Эта часть задержки может давать существенный вклад в общую величину, при этом, по понятным причинам, она не влияет на стабильность воспроизведения аудиоданных.

Таким образом популярные операционные системы позволяют работать со звуком в режиме мягкого реального времени благодаря нескольким основным факторам:

- Код аудио рендеринга/обработки исполняется с наивысшим приоритетом

- Аудио подсистемы написаны с соблюдением правил написания кода мягкого реального времени исполнения

- Использованы максимально эффективные алгоритмы и методы работы с потоком аудио данных для снижения нежелательной составляющей задержки

- В системе выставляется максимальная задержка в рамках приемлемых значений

## 1.3 Проблемы виртуализации средств поддержки мультимедиа

Теперь же рассмотрим случай виртуализированой системы на базе гипервизора второго типа. В такой конфигурации гостевая и хостовая операционные системы изолированы друг от друга гипервизором, и виртуальная машина делегирует работу гостевой ОС c оборудованием хостовой ОС. Не вдаваясь в детали реализации, общую архитектуру виртуальной машины, опирающейся на технологию аппаратной виртуализации на платформе x86 можно описать следующим образом [35] (рис 2):



Рис 2. Архитектура виртуальной машины, опирающейся на средства аппаратной виртуализации

- Виртуальная машина – это пользовательский процесс в хостовой операционной системе, который выделяет под нужды гостевой операционной системы память, и передает управление коду монитора виртуальных машин

- Монитор виртуальных машин сохраняет контекст операционной системы и переходит в собственный контекст исполнения, изолированный от хостового. В нем монитор виртуальных машин переводит процессор в режим поддержки виртуализации и передает исполнение гостевому коду. В случае, если гостевой код выполняет обращение к аппаратуре, то такой запрос перехватывается, и исполнение возвращается монитору виртуальных машин. Если этот запрос требует обращения к реальному оборудованию, то он передается в пользовательский процесс, который исполняет его.

При такой архитектуре указанные выше проблемы существующих имплементаций аудио подсистемы в современных ОС существенно усугубляются следующими факторами:

- Гостевая ОС отдает аудио данные не напрямую в оборудование, а в буфер виртуальной машины (эмулированный DMA буфер), из которой их забирает для воспроизведения хостовая ОС (либо же гостевая ОС получает буферизированные данные, которые должны быть еще сгенерированы на хосте для случая записи звука). Это означает, что нежелательная составляющая задержки аудио потока складывается как минимум из задержки хостовой и гостевой ОС.

- Виртуальная машина исполняется с приоритетом ниже, чем real-time, и поэтому в случае высокой загрузки хостовой ОС планировщик может не давать времени на исполнение виртуальной машине в течение произвольно долгого количества времени.

- Код эмуляции наиболее популярных виртуальных машин (KVM, XEN, Parallels Desktop, косвенно VMWare ESX, Microsoft HyperV), написан с нарушениями требований к написанию кода реального времени исполнения, и поэтому процедуры эмуляции могут исполняться произвольное долгое время (например в процессе выделения памяти, либо взятия блокировок). А это означает, что даже код гостевой ОС, удовлетворяющий требованиям написания кода реального времени исполнения, теряет описанные гарантии из за того, что время исполнения кода эмуляции не ограниченно сверху.

Из-за перечисленных проблем виртулизированный аудио тракт не может давать даже гарантий исполнения мягкого реального времени. Более того, этот результат невозможно улучшить без серьезной модификации эмуляционного кода существующих виртуальных машин, так как те грубо нарушают правила написания систем реального времени. Однако виртуальная машина реального времени исполнения имеет и свои слабые стороны: сложность такой системы заметно возрастет, а ограничение времени исполнения эмуляционных операций приведет к деградации производительности. Поэтому в коммерческих продуктах такой подход является чрезмерным. Из инструментов борьбы за качество мультимедиа в виртуализированной среде у разработчиков виртуальных машин остаются только следующие:

- Управление приоритетами исполнения виртуальной машины

- Управление задержкой аудио тракта

- Использование максимально эффективных алгоритмов и подходов к написанию аудио тракта.

Среди перечисленных методов наименее популярным является подход, связанный с управлением приоритета потоков. Это связано с тем что такое управление часто требует серьезных архитектурных изменений и наталкивается на трудноразрешимые проблемы приоритезации, вроде инверсии приоритетов. Поэтому в коммерческих продуктах такой подход применяется редко, и встретить его можно лишь в виде академических разработок на базе Xen [10,12].

В свою очередь управление задержкой, как наиболее предсказуемый подход, применяется довольно часто в таких проектах как KVM-Qemu (по умолчанию выставлена задержка в 10мс [21]), Virtual Box (по умолчанию - 20 мс [22]). В продукте Parallels Desktop задержка выставляется в зависимости от типа хостовой ОС и составляет 20 миллисекунд для OSX и 50 для Windows/linux.

Помимо этого немалый вклад в надежность аудио системы дает продуманный выбор архитектурных решений и алгоритмов, так как в виртуализированной среде модель потока данных в сравнении с той, что реализована в операционных системах, работающих без средств виртуализации, является более сложной. В потоке данных появляется новый узел, ответственный за обработку и передачу данных.

Рис 3. Архитектура виртуализированного аудио потока

 На рисунке 3 изображения схема работы виртуализированного аудио

потока. На этой схеме все вызовы являются синхронными, но процессы внутри кода эмуляции могут быть асинхронными, более того подход к передаче данных сильно различается в различных виртуальных машинах и это связано с особенностью работы DMA буфера аудио контроллера.

Этот буфер реализован как кольцевой буфер, и драйвер аудио карты следит за количеством проигранных данных, наблюдая за значением первого из указателей буфера. Это означает, что гостевая операционная система начнет генерировать новый квант данных только если указатель был смещен (данные воспроизведены с точки зрения гостя). Но при этом после перемещения указателя с начала кванта данных на его конец делает эти данные невалидными для компонентов виртуальной машины, так как после перемещения указателя гостевая ОС считает эту часть буфера уже проигранной аудио устройством, а значит свободной для новой записи. С учетом вышесказанного, логика работы эмулированного DMA буфера может быть реализована несколькими способами – обзор этих подходов дан в главе 3.1, а анализ их преимуществ и недостатков дан в главе 4.1.

# 2. Постановка задачи

В рамках данной работы мы рассмотри виртуальную машину Parallels Desktop. В силу специфики приложения в качестве хостовой ОС мы будем рассматривать только систему OS X. Аудио подсистема эмулирует AC’97- совместимую аудио карточку [24]. Текущая архитектура воплощает следующие решения:

1. В качестве хостового аудио компонента используется библиотека PortAudio [25]

2. Передача данных из DMA буфера в аудио модуль хостовой ОС организована опосредованно при помощи кольцевого буфера

3. Доступ к контрольным регистрам аудио карточки обрабатывается частью приложения, работающей в пользователськом режиме, тогда как монитор виртуальной машины работает в режиме проброса запросов.

Для описанной конфигурации в течении нескольких лет эксплуатации были выявлены следующие проблемы:

- Высокая нежелательная задержка вносимая хостовым аудио компонентом

- Низкая производительность аудио подсистемы при изменении параметров эмулируемого аппаратного микшера аудио карты. Связано с тем, что любой запрос на изменение (например автоматическая подстройка громкости звука в реальном времени) вызывает длительный процесс доступа из монитора виртуальных машин к пользовательскому приложению. Особенно критично эта проблема проявлялась в случае использования VoIP приложений, автоматически подстраивающих громкость микрофона в процессе работы.

- Проблемы перечисления присутствующих в системе аудио устройств, связанной с особенностями реализации библиотеки PortAudio.

- Жесткая привязка кода эмуляции в пользовательском приложении к спецификации AC’97.

- Невозможность использования системных преобразователей форматов, предоставляемых Core Audio в OS X в библиотеке PortAudio в случае если устройство не поддерживает формат, запрашиваемый гостевой ОС.

Описанные проблемы являются очевидным следствием принятых архитектурных решений (1-е и 3-е), и их разрешение требует отказа от описанных подходов и серьезной переработки аудио стека, и в этом смысле кажется абсолютно естественным оценить и уместность 2-го архитектурного решения, что бы в случае если описанный подход окажется неудачным и в дальнейшем станет новой ключевой проблемой, то в будущем не было бы необходимости очередного пересмотра архитектуры.

Таким образом, в рамках данной работы мы ставим себе цель улучшить характеристики аудио подсистемы продукта Parallels Desktop, последовательно выполнив следующие подзадачи:

- Анализ и выбор наиболее эффективного подхода к трансферу аудио данных между эмулированным DMA буфером и хостовой аудио подсистемой.

- Анализ существующих аудио библиотек и выбор наиболее подходящей вместо PortAudio.

- Проектирование новой архитектуры на базе принятых решений (новая библиотека, подход к эмуляции, перенос AC’97 логики в монитор виртуальных машин)

При этом необходимо учитывать тот факт, что в аудио системе стабильность воспроизведения является необходимым условием – т.е. уменьшение задержки не должно достигаться за счет ухудшения стабильности аудио подсистемы. В главе 3.1 дан обзор двух основных подходов к виртуализации аудио карты – их детальный анализ приведен в 4.1. В главе 3.2 дан обзор существующих библиотек реализующих поддержку мультимедиа, а в главах 4.2-4.3 описана имплементация аудио подсистемы на базе выбранных решений.

# 3. Обзор существующих решений работы со звуком

## 3.1 Архитектурные подходы к виртуализации аудио потока

Основная проблема при организации потока аудио данных из гостевой системы заключается в том, что подсистемы работают в разных режимах: гостевая работает в режиме push, тогда как хостовая в режиме pull [26]. Поэтому встает вопрос в том, каким образом связать эти два потока данных с учетом необходимости поддержания синхронности скорости генерации и потребления данных.



Рис 3. Схема работы аудио потока в случае воспроизведения данных из виртуальной машины

Решать данную проблему можно при помощи двух различных подходов:

- Запрос на трансфер данных между хостовой аудио системой и виртуальной машиной происходит по требованию хостовой системы. При этом данные при записи берутся напрямую из DMA, и гостевая система оповещается о том, что они проиграны только когда хостовая система запросит новый квант данных. Соответственно в случае записи звука запись идет напрямую в буфер. Такой подход реализован в проекте QEMU.

- Между DMA регионом виртуальной машины и хостовой аудио системой устанавливается дополнительный single-consumer-single- producer буфер, через который осуществляется передача данных. В таком случае виртуальная машина и хостовая аудио подсистема работают в асинхронном режиме. Такой подход использован в Virtual Box и текущей версии Parallels Desktop.

Среди очевидных преимуществ первого подхода можно выделить то, что нет двух дополнительных трансферов данных, но в то же время можно заметить, что в этом случае мы лишаемся преимуществ асинхронной работы (а именно возможностью более гибкой работы с задержкой аудио потока). Поэтому делать окончательный вывод о преимуществе одного подхода над другим можно только на базе более детального исследования, ход которого и результаты будут представлены в главе 4.1.

## 3.2 Существующие аудио библиотеки

В предыдущей главе мы рассмотрели различные подходы к виртуализации звука в предположении, что генерация аудиоданных и их передача в звуковое устройство происходит без существенных временных затрат, связанных с процессингом аудио (преобразование форматов, применение громкости, и т. д.). На практике же библиотеки, предоставляющие доступ к аудио оборудованию, оптимизируя свое поведения под конкретные паттерны использования, подчас жертвуя производительностью при других подходах к использованию. Поэтому выбор подходящей аудио библиотеки позволит улучшить характеристики аудио подсистемы.

Среди требований к аудио библиотеке особенно стоит выделить следующие:

1. Поддержка non-interleaved форматов PCM8, PCM16, PCM24, PCM32.

2. Поддержка преобразования частоты дискретизации и микширования каналов

3. Отсутствие связывания каналов воспроизведения и записи

4. Низкая дополнительная задержка, вносимая в аудио тракт.

5. Библиотека должна работать на операционной системе OSX

Пункт 3 требует пояснений: некоторые аудио библиотеки были изначально написаны для процессинга аудио-данных в формате реального времени, что накладывает ряд архитектурных ограничений на процесс функционирования аудио системы. В нашем случае устройства воспроизведения и записи являются отдельными устройствами, и процесс передачи данных по этим каналам не должен быть синхронизирован, тогда как процессинг данных в реальном времени предполагает связывание этих двух потоков одной функцией обратного вызова, которая дает данные на обработку и ждет в ответ обработанных данных.

Таким образом, имея ряд требований к аудио библиотеке, мы можем сделать обзор на наиболее популярные и принять решение о целесообразности использования одной из них.

### 3.2.1 Portaudio

Библиотека Portaudio[24] использовалась в для работы с аудио в проекте Parallels Desktop. Основным ее преимуществом являлась кроссплатформенность. Это было важным свойством, так как до недавнего времени кодовая база этой виртуальной машины использовалась для нескольких продуктов – в том числе и для Parallels Workstation, которая работала на ОС Windows и Linux. Но в то же время в процессе использования было выявлено довольно большое количество проблем, среди которых выделяются следующие:

- Библиотека предназначена в первую очередь для обработки аудио данных в реальном времени – это означает, что библиотека в одной функции обратного вызова как передает буфер с данными со входа, так и требует заполнить буфер с данными на выход. Это означает, что виртуальное устройство записи и виртуальное устройство воспроизведения приходится синхронизировать на уровне доступа к аудио библиотеки. Такое поведение вызывает ряд проблеем: начиная от невозможности выставления различных задержек этим двум потокам, заканчивая необходимостью синхронизации контрольных вызовов.

- Нестабильность библиотеки при многократном открытии/закрытии аудио устройств. Такое поведение было критически необходимо, так как в ОС Windows открытие аудио устройства приводило к удержанию в рабочем состоянии мультимедийного таймера, что снижало время жизни работы от батареи. В OSX же многократное открытие/закрытие устройств приводило к проблемам с выставлением аудио формата, при котором аудио устройство начинало неверно интерпретировать цифровые аудио данные.

- Изменение идентификаторов аудио устройств после ре-инициализации библиотеки. Как было уже указано – в процессе работы приложения приходится довольно часто ре-инициализировать библиотеку Portaudio, но при этом она не гарантирует неизменность идентификаторов аудио устройств, что приводило к необходимости использования решений, усложняющих архитектуру.

Таким образом как только было принято решение отказаться от поддержки ОС Windows и Linux в Parallels Desktop, возникла возможность перейти на более подходящую аудио библиотеку.

Более того: при внимательном анализе кода движка веб браузеров Chromium [28] обнаруживается, что архитектурные решения и код работы с аудио был частично взят из Portaudio, но со временем от этого кода все сильнее отказывались, пока аудио компонент не был окончательно переписан.

### 3.2.2 RTAudio

RTAudio [29] – популярная кроссплатформенная C++ аудио библиотека, предоставляющая простой API для работы с аудио оборудованием. Причиной интереса к этой библиотеки являлось то, что в отличии от других эта библиотека не предполагала работу в дуплексном режиме как основной подход, что означало что функции обратного вызова для устройств записи и воспроизведения не синхронизированы. Согласно документации, библиотека предоставляла все инструменты, необходимые для работы виртуализационного ПО, в том числе и:

- Поддержка динамической смены устройств

- Поддержка различных форматов, в том числе и non-interleaved signed integer (необходимое условие поддержки PCM)

- Поддержка конфигурации величины задержки, отсутствие дополнительной буферизации.

Но в то же время анализ исходного показал, что использование этой

библиотеки представляется невозможным в связи со следующими проблемами:

- В коде библиотеки, поддерживающей работу с аудио в системе OSX не поддерживается преобразование частоты дискретизации – при открытии устройства идет проверка поддерживаемых аппаратурой форматов, и в случае выставления экзотической частоты дискретизации гостевого аудио потока библиотека возвращает ошибку.

- Библиотека не позволяет выставить битрейт – предоставляется возможность использования только 32-битного потока

Таким образом, библиотека не удовлетворяла основным поставленным требованиям, и могла бы быть использована только с серьезными модификациями, сравнимыми по трудоемкости реализации с написанием свой библиотеки.

### 3.2.3 JUCE

JUCE [30] – кросплатформенный фреймворк, предоставляющий широкий инструментарий, включающий в себя модули работы с двухмерной графикой, звуком, интерфейсом пользователя и рядом других возможностей. Этот фреймворк часто используется для создания виртуальных инструментов в формате VST [31], для которых высокая производительность обработки и передачи аудиоданных является критически важным свойством. Именно поэтому JUCE был выбран в качестве возможного решения. Среди плюсов этой библиотеки необходимо выделить:

- Высокое внимание разработчиков к вопросам производительности аудио стека.

- Поддержка необходимых форматов задекларирована в документации.

Но в то же время детальный анализ выявил следующие проблемы:

- Как и в библиотеке RTAudio поддержка различных частот

дискретизации работала только в случае поддержки этих частот аудио устройством – встроенная конвертация отсутствовала

- Бедные возможности управления устройствами – нет функционала подписки на изменения в списке аудио устройств

- Отсутствие поддержки PCM форматов.

Таким образом, адаптация этого фреймворка к коду эмуляции с доработкой под нужды проекта является более трудоемкой, чем написание своего модуля работы со звуком.

### 3.2.4 Core Audio

Core Audio [31] – это компонент операционной системы OS X, предоставляющий низкоуровневый интерфейс для работы с аудио устройствами на языке C. Фактически перечисленные выше фреймворки реализуют свой функционал для операционной системе OS X, используя Core Audio. Особенностью этой библиотеки является возможность тонкой настройки параметров аудио тракта и широкий набор средств по обработке форматов аудио данных. Фактически, все требования выставленные к аудио компоненту, согласно документации, могут быть удовлетворены средствами Core Audio. Главным минусом такого подхода является необходимость написания полноценного компонента, отвечающего за работу с аудио оборудованием, и предоставляющего высокоуровневый интерфейс эмуляционному коду. Но как показал анализ существующих решений – их расширение по трудоемкости сопоставимо, а иногда и превосходит по сложности написание своего фреймворка, в то время как создание своего аудио компонента позволит более эффективно использовать особенности архитектуры виртуализированного аудио тракта и добиться большего качества аудио в продукте Parallels Desktop.

## 3.3 Заключение

Анализ существующих решений в области поддержки аудио и виртуализации звука привели к следующим выводам:

- Существующие библиотеки, предоставляющие высокоуровневый интерфейс доступа к аудио оборудованию обладают рядом особенностей, из-за которых их адаптация к аудио стеку проекта Parallels Desktop по трудоемкости сравнится или превысит сложность написания собственного аудио компонента. Поэтому было принято решение написать свой модуль работы с аудио оборудованием на базе Core Audio

- Необходимо провести детальное исследование двух подходов к виртуализации аудио потока, так как умозрительно сложно оценить преимущество одного над другим.

Ход описанных работ, а так же детали конечной реализации нового аудио стека в продукте Parallels Desktop даны в главе 4.

# 4. Ход работ

## 4.1. Сравнительный анализ архитектурных подходов по виртуализации аудио устройств

### 4.1.1 Формальная модель аудио потока

Для оценки целесообразности использования одного из подходов к виртуализации звука введем в рассмотрение следующую формальную модель, в которой время является дискретной величиной, с характерной величиной . Эту величину можно легко интерпретировать – это длина минимального кванта аудиоданных, запрашиваемого модулем ОС, отвечающей за взаимодействие с аудио оборудованием. В реальных системах эта величина порядка 1 миллисекунды.



Рис 5. Формальная модель аудио системы

Здесь:

 - количество буферезированных аудио данных (с)

 – скорость генерации аудиоданных (с/с)

 – количество воспроизведенных аудиоданных (с/с)

Введем динамическое уравнение, связывающее эти величины:

Для дальнейшего рассмотрения без доказательства примем следущие гипотезы:

- Передвижение данных между буфером и аудио устройством происходит мгновенно (). Этот эта предпосылка основана на экспериментальных данных: передача одного кванта аудио при характерном времени мс по продолжительности не превышает мкс. Таким образом:

- Воспроизведение происходит с некоторой задержкой , где – оптимальное количество буферизированных аудиоданных, а – задержка генератора связанная с пост-процессингом данных (применением громкости, преобразование форматов и т. д.). Таким образом:

В рамках введенной формализации рассмотрим следующую ситуацию: в качестве генератора данных выступает аудио подсистема ОС, буфером выступает DMA буфер между аудио устройством и ядром ОС. Анализ реализации аудио подсистем в открытых ОС позволяет сделать следущее предположение о логике работы генератора аудио данных:

- В случае, если количество буферизированных данных превышает , генератор не создаёт новых данных

- В случае, если количество данных в буфере снижается в сравнении с оптимальным, то ОС с некоторой вероятностью (обусловленной различным поведением системы на разных уровнях загрузки) переходит в режим форсированной генерации аудиоданных для восполнения буфера.

- В то же время в любой момент времени может произойти просроченный вызов генератора, связанный с несовершенством работы планировщика задач.

Учитывая описанную выше логику можно смоделировать изменения значения величины следующим образом:

 (4)

- Такая модель требует пояснений: – это вероятность того, что в квант времени генератор не будет вызван; - вероятность перейти в форсированный режим генерации данных. Так же в случае выпадения (), у нас в любом случае происходит генерация данных, так как в этом случае аудио подсистема проигрывает пустые данные. И помимо того, сильным предположением является то, что форсированный режим осуществляется с постоянной мультипликативной константой . этот факт требует экспериментального подтверждения и качественной оценки величины, который будет дан в следующем разделе.

### 4.1.2 Оценка мультипликативной константы модели

Для оценки величины мы в среде OSX напишем простейшее аудио приложение, которое воспроизводит синусоидальный сигнал при помощи библиотеки Core Audio. Эта библиотека зовет функцию обратного вызова в пользовательском коде, которая должна генерировать аудио данные. При одним из аргументов в этом вызове является структура, позволяющая оценить состояние аудио потока. Используя эти данные, мы можем оценить изменение величины , измеренное через равные интервалы времени мс. Запустим тестовый стенд в агрессивном окружении –на операционной системе одновременно с нашим приложением будет работать несколько процессов, активно потребляющих CPU, имея при этом высокий приоритет исполнения. На рисунке 6 представлен один из графиков изменений величины , с течением времени на одном из запусков.



Рис 6. График функции . Обе величины измерены в миллисекундах

Для оценки значения величины мультипликатора генерации , рассчитаем значения величины . В этом случае мы должны рассмотреть распределение положительных значений , для поиска возможных значений . Одно из таких распределений представлено на рисунке 7.



Рис 7. Гистограмма распределения неотрицательных значений

В приведенной гистограмме особенно выделяются два значения: 0 и 1. И опираясь на описанную модель эти два пика приобретают очевидную интерпретацию – пик в нуле соответствует стабильному режиму работы, когда количество произведенных данных на каждом шаге соответствует количеству проигранных данных за этот квант времени, в то время как пик в единице соответствует режиму ускоренной генерации аудиоданных. Таким образом случае форсированного режима:

А величина принимает вид:

### 4.1.3 Интерпретация модели как Марковской цепи

Учитывая полученные результаты мы можем перейти к следущей интерпретации нашей модели: можно представить как Марковский процесс с набором состояний: [0, , , … ] и матрицей переходов:

Упрощенно логика переходов между состояниями в такой системе заключается в следующем: с вероятностью мы переходим в состояние с большим значением величины буфера, с вероятностью буфер опустошается на величину . Соответственно в состоянии 0 буфер остается пустым, если только генератор в форсированном режиме не начнет опережать потребителя, и буфер снова начнет наполняться. А в состоянии генератор создает данные с нормальной скоростью, а значит наполненность буфера изменится только если случится дедлайн вызова с вероятностью . В описанной модели легко найти величину, характеризующую качество работы аудио подсистемы – это вероятность того, что на шаге наша система окажется в состоянии 0, если в начальный момент времени она была в состоянии .

### 4.1.4 Оценка метрики, характеризующей качество аудио потока

Оценим вероятность того, что на шаге система окажется в состоянии 0, если в начальный момент времени она была в состоянии . Для этого рассмотрим произвольный путь, по которому система может попасть из состояния в состояние 0 за шагов. Обозначим каждый шаг в этом пути , где -1 сопоставлен переходу системы в состояние с более низким значением, 0 – значение шага, если система осталась в том же состоянии, и +1 сопоставлен переходу с большем значением. Докажем утверждение:

*Утверждение 1*. Вероятность осуществления траектории переходов системы равна

 (5)

где - количество шагов , - количество шагов , - количество шагов проведенных в состоянии , - количество шагов проведенных в состоянии , - количество шагов проведенных в состоянии .

Для доказательства этого утверждения рассмотрим последовательность траекторий , сконструированных следующим образом:

и докажем, что для каждой из таких последовательностей вероятность исполнения может быть представлена в виде (5) с использованием метода математической индукции:

Базис индукции:

Таким образом при любых значениях , вероятность может быть выражена в указанном виде, причем степени соответствуют количеству шагов сделанных в этой последовательности.

Шаг индукции:

Пусть для последовательности шагов вероятность ее представлена в виде (5). Используя свойство независимости Марковской цепи рассчитаем вероятность выполнения последовательности :

Таким образом при любом значении и выражение соответствует виду (5). Утверждение доказано для в том числе и для N.

*Утверждение 2*. Вероятность осуществления траектории переходов системы из состояния в состояние 0 равна:

где

С учетом утверждения 1 это утверждение можно переформулировать следующим образом: в любой траектории переводящей систему из состояния в состояние 0 найдется как минимум шагов . Для описанной конфигурации цепочки это утверждение является очевидным.

Таким образом искомую вероятность можно рассчитать как:

Имея аналитический вид вероятности попадание системы в состояние 0 мы находим инструмент управления рисками аудиотракта: учитывая то, что увеличивая значение мы уменьшаем вероятность попадания системы в нулевое состояние на произвольном шаге N.

### 4.1.5 Сравнение архитектурных подходов в рамках приведенной формализации

Теперь же рассмотрим виртуализированную подсистему в рамках приведенной формализации. Выбирая один из описанных подходов к виртуализации аудио мы стремимся к качеству воспроизведения мультимедиа сопоставимому с качеством невиртуализированной системы. Т. е. предполагается, что мы некоторым образом сможем приблизить значение к значению . Оценим каким образом в рамках текущей модели меняются параметры при переходе к виртуализированному режиму.

Для этого будем считать что в новой системе генератором аудио данных является виртуализационное ПО, и при этом задержка в воспроизведении во втором случае (когда между DMA буфером гостевой система и DMA буфером хостовой системы установлен дополнительный буфер) учетом двойной буферизации может быть рассчитана как:

В первом же случае (данные напрямую передаются из гостевого DMA буффера в хостовый), задержка в воспроизведении с точки зрения нашей модели соответствует величине . Является очевидным следующее утверждение: в случае 2-го подхода к виртуализации звука мы имеем полноценную возможность управлять значением величины , изменяя значнеие , тогда как 1-й подход лишен такой возможности.

Однако мы не учитываем тот факт, что серьезное расхождение в оценки изменения времени в каждом из подходов к виртуализации может существенно повлиять на значения величин 𝜀 и 𝑢. В случае если это так, то возможно что в первом случае мы сможем добиться некоторого удовлетворительного значения , при более низком значении . Для того, что бы оценить справедливость такого утверждения воспользуемся следующим стендом: описанное выше приложение, генерирующее синусоидальный сигнал мы запустим на двух стендах:

- В первом случае будет использована ВМ, работающая с прототипом аудио подсистемы, работающим по первому сценарию (прямой доступ к гостевому DMA).

- Во втором случае аудио подсистема будет реализована по втором сценарию – с использованием промежуточного аудио буфера. Но при этом глубина пребуферизации для него будет выставлена в 0, для того, что бы исключить его потенциальное влияние на значения 𝜀 и 𝑢.

В обоих случаях стенд будет запущен на высоконагруженной хостовой ОС, и на уровне хостовой аудио библиотеки мы будем как и в предыдущих измерениях следить за изменением величины 𝐵(𝑡). Но в этот раз мы обратим внимание не на абсолютную величину изменения загрузки в каждый из квантов времени, а на количество событий выпадения данных и их ускоренного восстановления в случае неполного буфера. Каждый запуск длился 240 секунд, всего для каждой конфигурации было проведено 4 запуска. Результаты приведены в таблице 1.

|  |  |  |
| --- | --- | --- |
|  | 𝜀, %  | 𝑢, % |
| Прямой доступ к DMA  | 4,2 ± 0,1  | 93,3 ± 0,6  |
| Доступ через буффер  | 4,1 ± 0,1 | 92,7 ± 0,8 |

Таблица 1. Оценка значений 𝜀 и 𝑢 для разных подходов к виртуализации аудио подсистемы

 Таким образом, мы видим, что значения этих двух величин не претерпевают значительных изменений в различных режимах виртуализации. Это кажется вполне адекватным в свете того факта, что эти две вероятности в первую очередь охарактеризованы заложенными в планировщик алгоритмами выдачи кванта времени исполнения. Таким образом обе системы получают примерно одинаковое количество времени на обработку запросов на генерацию аудио данных, и успевают отработать с близкими значениями времени отклика.

 Имея ввиду этот факт мы можем прийти к выводу, что подход, использующий промежуточный буфер оказывается более привлекательным, так как при сопоставимом с первым подходом времени отклика он позволяет увеличивать величину пребуферизации, тем самым компенсируя снижение приоритета процесса аудио генератора.

## 4.2 Архитектура потоковой виртуализации аудио данных

Данный в предыдущей главе анализ показал, что оптимальным подходом к виртуализации аудио потока аудио данных является использование буфера для передачи данных между виртуальной аудио картой и хостовой частью аудио подсистемы. При этом как уже было указано мы стремимся перенести логику работы AC97 аудио карты в код эмуляции монитора виртуальных машин. Это позволяет нам создать унифицированный̆ объект аудио потока, состоящий̆ из:

- Неблокирующего кольцевого буфера [33]

- Описания формата, используемого для передачи данных внутри кольцевого буфера

При этом сферы ответственности компонентов распределяются следующим образом:

- Монитор виртуальных машин занимается обработкой запросов к ac97-cовместимой эмулируемой аудио карте, конфигурирует унифицированный аудио поток, переносит аудио данные из DMA ,буфера в аудио поток, а так же передает контрольные запросы в часть виртуальной машины, исполняющейся в режиме пользователя

- Пользовательское приложение обрабатывает запросы от монитора на запуск/остановку аудио потока и передает их в аудио библиотеку. Так же этот компонент занимается управлением выбора реального устройства воспроизведения и задержками виртуализированного аудио потока

- Аудио библиотека является обёрткой вокруг интерфейса Core Audio, позволяющей использовать унифицированный̆ аудио поток в

качестве источника данных и информации о формате воспроизведения. Так же она дает пользовательской части приложения простой интерфейс для работы с информацией̆ о присутствующих в системе реальных аудио устройствах, учитывая особенности работы виртуальных машин

Разработка велась итеративно со следующими стадиями:

- Расширение кольцевого буфера до унифицированного аудио

потока

- Пошаговый перенос кода обработки запросов к ac97-

совместимой эмулируемой карте из пользовательского приложения в монитор виртуальных машин

- Написание кода конфигурации параметров унифицированного аудио потока, изменяющего эти параметры в при доступе к mmio регистрам эмулируемой аудио карты

- Изменение интерфейса взаимодействия монитора виртуальных машин и пользовательского приложения

- Переход на использование унифицированного аудио потока в качестве единственного источника данных и конфигурации аудио в пользовательском коде эмуляции.

## 4.3 Разработка аудио библиотеки

Как было указано в главе 3, существующие аудио библиотеки не удовлетворяли требованиям к хостовой части аудио подсистемы виртуальной машины, и поэтому было принято решение написать свой аудио компонент на базе интерфейса, предоставляемого Core Audio. Основной цикл работы в этом случае выглядит так [14]:

- Клиент создает обьект Audio Unit (HAL Audio Unit), выставляет необходимый формат аудио данных, параметры воспроизведения, регистрирует функцию обратного вызова и запускает воспроизведение

- Core Audio вызывает функцию обратного вызова как только необходимо произвести трансферт данных. В качестве параметра передается указатель на буфер, в котором содержатся данные, либо который необходимо заполнить данными. Так продолжается до момента остановки.

Такой подход в явном виде не учитывает ряд особенностей работы с аудио устройствами в среде OSX, которые так же должны быть учтены в аудио библиотеке:

- В списке устройств, предоставляемом библиотекой Core Audio нет “устройства по умолчанию”. Но есть параметр системы, который указывает на текущее устройство, которое считается таким. Из этого следует, что смена “устройства по умолчанию” должна быть обработана нашей аудио библиотекой (библиотека Portaudio не решала эту проблему, что приводило к излишней запутанности кода эмуляции).

- Тестирование прототипа, использующего Core Aduio для воспроизведения звука показало, что несмотря на ряд заверений

разработчиков Apple, эта аудио библиотека не умеет работать с форматами, в которых частота дискретизации не поддерживается аудио устройством, используемым для работы. Эта проблема может быть решена при помощи компонента Core Audio – AudioConverter, принцип работы которого совпадает с принципом работы HAL Audio Unit и основан на функции обратного вызова. Особенностью этого конвертера является то, что в качестве параметра при конвертации передается количество фреймов аудио данных, которые нужно получить на выходе из конвертера. Такой подход является очень удобным в случае воспроизведения: HAL Audio Unit запрашивает определенное количество квантов аудио в своей функции обратного вызова. Мы это же количество квантов запрашиваем у Audio Converter Unit’а, а тот в свою очередь выберет из нашего унифицированного аудио потока ровно столько фреймов, сколько ему необходимо (а в случае несовпадения частоты дискретизации ему может понадобиться как большее количество фреймов, так и меньшее). Но в этом случае для устройства записи возникает закономерная проблема: получив в функции обратного вызова пакет данных конкретной длинны – мы не знаем, какое количество квантов в гостевом формате выйдет из этого пакета после трансформации, а значит мы не знаем какой параметр выдать в трансформатор. Проблема заключается в том, что при некратном значении отношения хостовой и гостевой частоты дискретизации при конвертации запрашиваются пакеты данных фиксированного размера, и если из аудио устройства придет некратное размеру пакета количество данных, то часть из них не будет использована и без дополнительных действие со стороны аудио библиотеки пропадет, что для пользователя будет звучать как искажение звука. Обзор существующих решений показал что с этой проблемой борются созданием дополнительного кольцевого буфера между HAL Audio Unit и Audio Converter Unit.

- Функции обратного вызова в Core Audio не потоко-безопасны в том смысле, что могут быть вызваны после остановки и даже закрытия аудио устройства. Ряд библиотек не учитывал эту особенность и потому имел серьезные проблемы со стабильностью.

Учитывая выявленные выше особенности, было принято решение использовать следующую архитектуру:

CAudioPipeline – обертка вокруг унифицированного аудио потока, реализующую простую read/write семантику, и инкапсулирующего внутри себя логику по предобразованию аудио форматов.

CAudioUnit – обьект, инкапсулирующий логику инициализации и работы с HAL Audio Unit.

CAudioUnitManager – singletone обьект, занимающийся маршрутизацией запросов между CAudioUnit’ми и Core Audio. Является точкой входа для сторонних компонентов, и разрешает проблемы исполнения функций обратного вызова в устаревшем контексте при помощи механизма подсчета поколений аудио юнитов.

Отдельным компонентом является менеджер устройств, который позволяет подписываться на изменения в списке устройств в системе и получении информации о них. Так как его функционал востребован сразу несколькими частями приложения, то его код вынесен в отдельную библиотеку.

# 5. Оценка результатов работы

Оценка результатов работы была проведена на базе измерения следущих метрик:

1. Полная величина задержки при неизменном значении величины пребуферизации

2. Частота возникновения искажений в VoIP приложениях при автоматической подстройке громкости аудио

Для проверки первого утверждения был собран следующий стенд: на физической машине MacPro 2 x 2,8 GHz Quad-Core Intel Xeon с установленной на ней OS X 10.10 линейный выход было соединен с линейным входом шнуром 3.5 jack. Внутри виртуальной машины (2 virtual CPU cores) Windows 8 было установлено приложение Audacity [36], позволяющее стартовать одновременную воспроизведение аудио сигнала и запись. В качестве аудио сигнала была выбрана синусоида, а в качестве метрики – задержка между началом воспроизведения и началом записи сигнала. Результаты измерений приведены в таблице 2.

|  |  |  |
| --- | --- | --- |
|  | Старая архитектура  | Новая архитектура  |
| Задержка полная, мс  | 119 ±15 | 86 ±11  |

Таблица 2. Сравнение величин полной задежки

Таким образом, полная задержка виртуализированного аудиотракта была снижена на 27% при неизменной величине пребуферизации.

 На том же стенде удалось обнаружить качественный скачек в скорости обработки изменения громкости аудиоустройств. Для этого на том же стенде было установлено приложение Lync [37] и совершен звонок. На старой аудио архитектуре были очевидны искажения аудио сигнала. В версии с новой архитектурой эти помехи устранены.

# 6. Заключение

В рамках данной работы был приведен анализ существующей архитектуры виртуализированного аудио потока в продукте Parallels Desktop, выявлены слабые стороны, и предоставлен анализ и сравнение возможных архитектурных изменений, улучшающих аудио характеристики продукта. Были проанализированы следующие вопросы, раскрывающие подходы к улучшению системы:

- Вопрос уместности дополнительного слоя буферизации между эмулируемой аудио картой и хостовой аудио системой. Формальный анализ этой проблемы показал, что используемый в текущей реализации подход, предполагающий использование дополнительного буфера, является более предпочтительным перед подходами, реализованными в рамках других виртуализационных продуктов.

- Проблема выбора подходящей аудио библиотеки. Обзор существующих решений (в том числе и ранее использованного в продукте) показал, что они не удовлетворяют требованиям со стороны виртуализационного ПО, и, следовательно, наилучшим решением являлось написание собственной аудио библиотеки.

Таким образом, анализ показал, что наиболее оптимальным решением в вопросе улучшения аудио харатктеристик продукта Parallels Desktop является использование текущей архитектуры аудио потока (с рядом некритичных изменений) и имплементация собственной библиотеки для работы с аудио устройствами, учитывающей специфику виртуализацотнного ПО.

Проведенная оценка результатов работы показала, что выбранный подход оправдал себя, и полная задержка аудио тракта упала почти на треть, при неизменной величине пребуферизации.

Тем не менее, стоит отметить ряд вопросов, которые все еще остаются актуальными, являясь по сути логическим продолжением данной работы:

- Развитие формальной модели виртуализированного аудиотракта. Анализ с учетом нестационарной природы нагрузки системы.

- Обзор методик управлением приоритетами виртуальной машины и анализ их применимости.

- Анализ методологий предсказания аудиосигнала в случаях простоя генератора.

В заключение хотелось бы особенно поблагодарить Константина Озеркова и Анну Мелехову за неоценимую помощь в работе.

# 7. Список использованной литературы

1. ADAMS, K. AND AGESEN, O. 2006. A comparison of software and hardware techniques for x86 virtualiza- tion. In Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XII). 2–13.

2. AMD. AMD64 Virtualization Codenamed “Pacifica” Technology: Secure Virtual Machine Architecture Reference Manual, May 2005.

3. INTEL CORPORATION. Intel Virtualization Technology Specifica- tion for the IA-32 Intel Architecture, April 2005.

4. http://www.vmware.com/products/esxi-and-esx/overview

5. http://www.linux-kvm.org/page/Main\_Page

6. http://www.vmware.com/ru/products/workstation

7. http://www.parallels.com/ru/products/desktop

8. Christopher Clark , Keir Fraser , Steven Hand , Jacob Gorm Hansen , Eric Jul , Christian Limpach , Ian Pratt , Andrew Warfield, Live migration of virtual machines, Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation, p.273-286, May 02-04, 2005

9. MANACHER, G.K. Production and stabihzation of real-time task schedules. J. ACM 14' 3 (July 1967), 439-465.

10. Raoul Rivas , Ahsan Arefin , Klara Nahrstedt, Janus: a cross-layer soft real- time architecture for virtualization, Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, June 21-25, 2010, Chicago, Illinois

11. Vincent Legout , Matthieu Lemerre Paravirtualizing Linux in a real-time hypervisor, ACM SIGBED Review - 2nd Workshop on Embed With Linux

12. RT-Xen: towards real-time hypervisor scheduling in xen

13. Martin, James (1965). Programming Real-time Computer Systems.

Englewood Cliffs, NJ: Prentice-Hall Inc. p. 4. ISBN 013-730507-9.

14. http://www.portaudio.com/docs/latency.html

15. https://developer.apple.com/library/mac/documentation/MusicAudio/Concep

tual/CoreAudioOverview/CoreAudioEssentials/CoreAudioEssentials.html

16. https://msdn.microsoft.com/en-

us/library/windows/desktop/dd756612(v=vs.85).aspx

17. A. C. Younkin and P. J. Corriveau "Determining the amount of audio-video synchronization errors perceptible to the average end-user", IEEE Trans. Broadcast., vol. 54, pp.623 -627 2008

18. ITU-T Recommendation G.114

19. http://www.yamahaproaudio.com/europe/en\_gb/training\_support/selftrainin

g/audio\_quality/chapter5/05\_absolute\_latency/

20. http://www.asio4all.com/

21. http://www.linux-kvm.org/page/Main\_Page

22. https://www.virtualbox.org

23. http://www.di.ens.fr/~guatto/sbac13.pdf

24. http://www.intel.com/content/dam/www/public/us/en/documents/product-

specifications/high-definition-audio-specification.pdf

25. http://www.portaudio.com/

26. https://developer.apple.com/library/mac/documentation/MusicAudio/Concep

tual/AudioUnitProgrammingGuide/AudioUnitDevelopmentFundamentals/A

udioUnitDevelopmentFundamentals.html

27. http://www.intel.com/content/dam/www/public/us/en/documents/technical-

specifications/extensible-host-controler-interface-usb-xhci.pdf

28. http://blogs.msdn.com/b/usbcoreblog/archive/2013/04/11/usb-2-1-2-0-1-1- device-enumeration-changes-in-windows-8.aspx

29. https://chromium.googlesource.com/chromium/src.git

30. https://www.music.mcgill.ca/~gary/rtaudio

31. http://www.juce.com/features

32. http://jvstwrapper.sourceforge.net/vst20spec.pdf

33. https://developer.apple.com/library/mac/documentation/MusicAudio/Concep

tual/CoreAudioOverview/Introduction/Introduction.html

34. John David Valois, Lock-free data structures, Rensselaer Polytechnic Institute, Troy, NY, 1996

35. http://www.intel.com/content/www/us/en/processors/architectures-software- developer-manuals.html

36. http://sourceforge.net/projects/audacity/

37. https://support.office.com/en-ca/article/Install-Skype-for-Business-Lync-

8a0d4da8-9d58-44f9-9759-5c8f340cb3fb

38. Popek, Gerald J.; Goldberg, Robert P. (1974). "Formal requirements for virtualizable third generation architectures". Communications of the ACM 17 (7): 412–421. doi:10.1145/361011.361073. Retrieved 2015-03-01

39. https://software.intel.com/en-us/articles/the-advantages-of-using-virtualization-technology-in-the-enterprise