Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Московский физико-технический институт

(государственный университет)»

Факультет радиотехники и кибернетики

Кафедра Теоретической и Прикладной Информатики

**Исследование и реализация способов расширения номенклатуры виртуальных устройств, подключаемых к гипервизорному виртуализационному решению.**

Выпускная квалификационная работа

(бакалаврская работа)

Направление подготовки: 03.03.01 Прикладные математика и физика

Выполнил:

студент 211 группы Фофанов Владислав Сергеевич

Научный руководитель: Симановский Андрей Юрьевич

Москва 2016

Оглавление

[Введение 3](#_Toc454895945)

[1. Описание гипервизорного виртуализационного решения 6](#_Toc454895946)

[2. Классификация способов виртуализации устройств 9](#_Toc454895947)

[1. Полная эмуляция 9](#_Toc454895948)

[2. Проброс 9](#_Toc454895949)

[3. Аппаратная виртуализация 10](#_Toc454895950)

[4. Паравиртуализация 10](#_Toc454895951)

[3. Использование системы подгружаемых хостовых модулей для виртуализации широкого спектра устройств. 11](#_Toc454895952)

[4. Open ToolsGate как механизм передачи данных при паравиртуализации 13](#_Toc454895953)

[5. Устройство базового механизма Open ToolsGate 15](#_Toc454895954)

[6. Базовый вариант интерфейса для передачи данных устройств на основе Open ToolsGate 18](#_Toc454895955)

[7. Расширенный вариант интерфейса для передачи данных устройств на основе Open ToolsGate 20](#_Toc454895956)

[8. Сравнение производительности вариантов 23](#_Toc454895957)

[9. Дальнейшие идеи по расширению номенклатуры 25](#_Toc454895958)

[Заключение 26](#_Toc454895959)

[Список литературы 27](#_Toc454895960)

# Введение

 **Актуальность темы исследования.** На сегодняшний день в развитии компьютерной индустрии важную роль играет разработка, создание и модернизация внешних устройств, подключаемых к компьютеру. Более современные устройства, такие как камеры, передают большие объемы данных через различные интерфейсы для воспроизведения их на компьютере. Вместе с этим, все большую роль играют технологии виртуализации, применительно к частным и корпоративным решениям. Для сохранения актуальности виртуализации нужно расширять спектр возможностей виртуальных машин в соответствии с современными технологическими достижениями. Таким образом, для развития виртуализации необходимо поддерживать виртуализацию новых устройств, учитывая их запросы касательно скорости передачи данных.

 Данная дипломная работа описывает методы расширения интерфейса взаимодействия устройств с гостевыми операционными системами, что позволяет виртуализовать новые классы устройств на должном уровне и, таким образом, расширить номенклатуру виртуальных устройств.

 **Объектом исследования** является интерфейс передачи данных между гостевыми и хостовой операционной системами.

 **Целью дипломной работы** является расширение номенклатуры виртуальных устройств, подключаемых к гипервизорному виртуализационному решению путем расширения интерфейса взаимодействия между гостевой и хостовой операционными системами.

 **Используемая методология и степень разработанности темы.** Решение данной задачи базируется на теории виртуализации памяти в гипервизорном виртуализационном решении и использовании данных об организации виртуального адресного пространства гостевой операционной системы. Данная тема является очень узкоспециализированной, поэтому исследования, описываемые в этой работе, являются уникальными.

 **Теоретическая значимость.** Разработаны и реализованы две модели передачи данных между гостевой и хостовой операционными системами на основе общих страниц памяти для монитора виртуальных машин (VMM) и гостевой части виртуальной машины.

 **Практическая** **значимость.** Для разработанных моделей измерена скорость передачи данных. Полученный интерфейс предлагается к использованию для виртуализируемых устройств в качестве альтернативы существующим методам передачи данных между гостевой и хостовой операционными системами.

 **Структура исследования.** Работа состоит из введения, девяти глав и заключения. В первой главе описывается устройство гипервизорного вируализационного решения – платформы, для которой мы проводим расширение номенклатуры. Во второй главе рассматриваются четыре способы виртуализации устройств: эмуляция, проброс, аппаратная виртуализация и паравиртуализация. В третьей главе мы рассмотрим концепцию модульных устройств как способ расширить номенклатуру виртуальных устройств. В четвертой главе мы рассмотрим вопрос о возможных направлениях деятельности, связанной с созданием интерфейса между гостевой и хостовой операционными системами в зависимости от способа виртуализации устройств. В пятой главе мы рассмотрим основной механизм, используемый нами при создании интерфейса – Open ToolsGate (OTG). В шестой и седьмой главах описываются две модели передачи данных на основе механизма Open ToolsGate: базовая и расширенная. В восьмой главе приведены результаты измерения скорости передачи данных рассматриваемых моделей. И, наконец, в последней главе указаны дальнейшие направления деятельности для улучшения интерфейса и расширения номенклатуры.

# 1. Описание гипервизорного виртуализационного решения

Прежде всего, вкратце рассмотрим основные принципы работы платформы для нашей исследовательской работы. Главной составляющей гипервизорного виртуализационного решения является, собственно, сам гипервизор.

**Гипервизор** или Virtual Machine Manager— програмная схема, позволяющая одновременное, параллельное выполнение нескольких операционных систем на одном и том же хост-компьютере [1]. Гипервизор также играет роль диспетчера ресурсов системы, таких как память, процессоры, устройства... Гипервизор обеспечивает изоляцию гостевых операционных систем друг от друга.

 Гипервизоры разделяются по типу управления ресурсами хоста:

 1. Гипервизор, который устанавливается непосредственно на аппаратную часть без предварительной установки операционной системы. Гипервизор такого типа, по сути, сам является «облегченной» версией хостовой операционной системы, выполняя функции распределения памяти и процессоров.

 2. Гипервизор, который работает поверх установленной операционной системы в нулевом кольце (т.е. в качестве ядерного модуля). Гостевые операционные системы используют аппаратные ресурсы не напрямую, а через хостовою операционную систему (с помощью гипервизора).

 В рамках данной дипломной работы мы будем рассматривать гипервизоры второго типа, как наиболее распространённые. Взаимосвязь базовых компонентов гипервизорного виртуализационного решения, использующего гипервизор второго типа, изображена на рисунке 1.1.



Рис. 1.1 Устройство гипервизорного виртуализационного решения

 Как можно увидеть из схемы, гипервизорное виртуализационное решение состоит из следующих основных компонент:

 1. Приложение уровня пользователя – процесс в хостовой операционной системе, выполняющий такие функции, как [1]:

 - Получение данных и команд от реальных устройств для последующей передачи их гостевой операционной системе.

 - Работа с памятью гостевой и хостовой операционных систем, в том числе и для установления взаимодействия между ними.

 - Реализация некоторых дополнительных функций гипервизорного виртуализационного решения.

 2. Гипервизор – драйвер хостовой операционной системы, выполняющий две важные функции. Во-первых, гипервизор является диспетчером виртуальных машин. Он регистрирует новые виртуальные машины, отслеживает состояние уже зарегистрированных, а также отвечает за распределение системных ресурсов между виртуальными машинами. Во-вторых, гипервизор осуществляет переключение между приложением уровня пользователя и VMM (еще один компонент, используемый для виртуализации), используя механизм HyperSwitch. Данный механизм подменяет текущий контекст выполнения на контекст искусственно созданного процесса, в рамках которого исполняется VMM [2].

 3. Монитор Виртуальных Машин или VMM – представляет из себя файл с кодом, загружаемый в искусственно созданный процесс, переключение к которому происходит с помощью механизма HyperSwitch [1][3]. Основные функции VMM:

 - Эмуляция тех машинных инструкций, которые не могут быть исполнены непосредственно на процессоре. Такие инструкции могут возникать в следствие различия базовых архитектур у гостевой и хостовой операционных систем, а также в следствие наличия инструкций, непосредственное исполнение которых невозможно для гостевой операционной системы.

 - Эмуляция различных аппаратных событий для гостевой операционной системы, например, прерываний [3].

 - Выполняет некоторые промежуточные действия по обработке взаимодействия между гостевой и хостовой операционными системами. Степень участия в этой обработке очень сильно зависит от способа виртуализации устройств, о чем будет подробнее рассказано в дальнейшем.

 В дальнейшем, наши исследования будут широко опираться на устройство этих компонент.

# 2. Классификация способов виртуализации устройств

 Виртуализацию компьютеров можно условно разделить на три направления: виртуализация процессора, виртуализация памяти и виртуализация устройств [2]. В рамках данной дипломной работы мы будем рассматривать в основном виртуализацию устройств.

 Виртуализация устройств подразделяется на четыре типа:

 1. Полная эмуляция – такой тип виртуализации устройств, при котором происходит эмуляция всего стека устройства для гостевой операционной системы. Приложение уровня пользователя получает данные с устройства через какой-либо интерфейс (возможно, высокоуровневый), а затем конвертирует их в низкоуровневые запросы, которые используются непосредственно устройствами. Затем, с помощью HyperSwitch, происходит передача управления VMM, который эмулирует весь аппаратный стек данного устройства (устройство, шина, контроллер...). VMM передает запрос виртуальному устройству, и тот начинает его обрабатывать как это бы делало реальное устройство. В итоге, после обработки стеком виртуальных устройств, запрос приходит в гостевой драйвер устройства для последующей стандартной обработки. При такой модели гость полностью уверен в реальности виртуальных устройств, и, как следствие, в том, что он находиться на реальной аппаратуре [2].

 2. Проброс – технология виртуализации устройств, при которой данные берутся непосредственно с устройства и в неизмененном виде передаются драйверу гостевой операционной системы. Для этой цели в кольцо уровня ноль хостовой операционной системы устанавливается программа-перехватчик. Она перехватывает запросы от устройств прежде чем их получит драйвер устройства хостовой операционной системы. Перехватчик посылает данные в приложение уровня пользователя, а оттуда данные передаются в VMM. VMM эмулирует событие прихода данных с устройства для гостя, таким образом передавая данные драйверу устройства гостевой операционной системы. Таким образом, создается т.н. мост между устройством и драйвером гостя. Дополнительно, программа перехватчик посылает сигнал хостовой операционной системе об отключении устройства. Это позволяет, по сути, передать устройство под управление гостевой операционной системе [2][1].

 3. Аппаратная виртуализация (Intel VT-d) – расширение техники проброса путем добавления в чипсет нового модуля – IOMMU. Данный модуль, при необходимости, перехватывает запросы устройств на DMA операции, и заменяет физическую память хостовой операционной системы на физическую память гостевой операционной системы. Таким образом, устройства могут совершать DMA операции напрямую с гостем, что значительно увеличивает производительность виртуальных устройств [2].

 4. Паравиртуализация – техника виртуализации, при которой в качестве средства передачи данных между устройствами и гостевой операционной системой используется какой-либо нестандартный интерфейс (отличный от стандартных протоколов взаимодействия между устройствами и драйверами). Для того, чтобы реализовать такой метод, необходимо внести некоторые изменения в ядро гостевой операционной системы, чтобы она была готова к передаче данных через нестандартный интерфейс, что является одним из главных недостатков такого метода. Однако, при использовании данного метода, можно для каждого устройства наладить взаимодействие с гостевой операционной системой через свой собственный, подходящий для данного устройства, интерфейс. При этом, варьировать интерфейс можно не только в плане формата передаваемых данных, но и в плане уровня приема/передачи этих данных (более/менее высокоуровневый интерфейс) [2][4].

# 3. Использование системы подгружаемых хостовых модулей для виртуализации широкого спектра устройств.

Цель данной исследовательской работы – расширение номенклатуры виртуальных устройств. Для реализации задуманного в рамках дипломной работы была спроектирована общая система виртуализации устройств на основе подгружаемых хостовых модулей. Рассмотрим принцип работы данной системы.

 В первую очередь, нам необходимо поместить информацию о загружаемых модулях (название, полный путь...) в реестр параметров виртуальной машины. Во время старта работы, виртуальная машина будет считывать данные из реестра, и на его основе составлять список подгружаемых модулей.

 Далее, на основе записей из реестра, виртуальная машина загружает модули к себе (в свое адресное пространство). Однако, чтобы виртуальная машина могла взаимодействовать с кодом модуля, необходимо чтобы в модуле был реализован определенный интерфейс обратных вызовов, который виртуальная машина может использовать при необходимости.

 Основное преимущество данной модульной системы в том, что при прописанном заранее формате интерфейса обратных вызовов и доступе к реестру виртуальной машины, ее может использовать пользователь без доступа к исходному коду виртуальной машины. Это позволит разработчикам виртуализировать произвольные устройства, используя модули в качестве эмулируемого устройства или прослойки между реальным устройством и гостевой операционной системой.

 Тем не менее, для того, чтобы эта система работала, необходима связь между гостевой операционной системой и модулями в виде интерфейса для передачи данных, при этом, данный интерфейс должен быть достаточно универсальным, чтобы им могли пользоваться любые виртуализированные устройства, а также достаточно быстрым, чтобы обеспечивать потребности самых современных устройств. Это очень важная проблема, без решения которой концепция модулей не имеет возможности к осуществлению. Именно решению данной проблемы посвящена дальнейшая часть дипломной работы.

# 4. Open ToolsGate как механизм передачи данных при паравиртуализации

 Как было заключено в предыдущей главе, наша основная задача - расширить интерфейс взаимодействия между устройствами и гостевой операционной системой. Для начала, давайте рассмотрим, какие стандартные интерфейсы используются для взаимодействия устройств с процессором и памятью:

 Взаимодействие с памятью:

 1. DMA (Direct Memory Access) – способ взаимодействия устройств с памятью в обход процессора для экономия процессорного времени и ускорения чтения/записи в память устройством. IOMMU работает именно с этим механизмом.

 Взаимодействие с процессором:

 1. MMIO (Memory Mapped Input/Output) – способ взаимодействия устройств с процессором через стандартный интерфейс обращения к памяти. При использовании MMIO процессор для чтения/записи в регистры устройства использует стандартные системные вызовы READ и WRITE. Процессор пишет данные по виртуальным адресам, которые поступают в MMU для конвертации в физические. В MMU адрес записи чтения конвертируется в физический, но при этом, данный физический адрес лежит вне диапазона существующей физической памяти. Поэтому, когда запрос попадает на шину, он отправляется не в память, а к соответствующему устройству.

 2. IO Ports – способ взаимодействия с устройством с использованием интерфейса в виде системных вызовов IN и OUT. Поскольку данные через него передаются побайтно за одну команду, данный интерфейс имеет сравнительно небольшую скорость. Поэтому, на сегодняшний день, он мало применяется.

 Заметим, что данные стандартные интерфейсы взаимодействия с устройствами в том или ином виде используются и в большинстве методов виртуализации устройств. Так, при использовании полной эмуляции, проброса или аппаратной виртуализации, данные, тем или иным образом, передаются через один из этих интерфейсов. Поэтому мы не можем использовать эти типы виртуализации устройств в нашем исследовании.

 Остается один тип виртуализации – паравиртуализация. В этом случае мы сами устанавливаем интерфейс взаимодействия между устройством и гостевой операционной системой. Именно создание эффективного интерфейса передачи данных и является непосредственной целью данной дипломной работы. Основой нашего, более широкого, интерфейса будет уже реализованный в рамках Parallels Desktop интерфейс передачи данных – Open ToolsGate.

# 5. Устройство базового механизма Open ToolsGate

Устройство базового механизма Open ToolsGate (OTG), на основе которого будет разработан интерфейс передачи данных между устройствами и гостевой операционной системой, изображено на рисунке 4.1.



Рис. 4.1 Устройство базового механизма OTG

 Прежде всего стоит отметить, что в силу некоторых программных ограничений, базовый механизм OTG оперирует блоками данных, размером 4 килобайта. Также, стоит обратить внимание на то, что инициатором обмена с использованием данного механизма может быть только гостевая операционная система (т.е. со стороны хостовой операционной системы невозможно подать сигнал о начале обмена данными).

 Передача данных происходит следующим образом:

 1. Гость инициирует передачу путем вызова ключевой (небезопасной для прямого исполнения) инструкции – RDPMC. Предварительно гость должен записать аргументы передачи в различные регистры. Самыми важными аргументами являются:

- указатель на буфер данных для обмена

- фактический размер данных, содержащийся в буфере

- общий размер данных, которые мы хотим передать/получить через данную операцию

- идентификатор данной операции

- текущее состояние данной операции

После заполнения регистров данными и вызова RDPMC происходит переключение в VMM с целью дальнейшей обработки запроса.

2. В VMM происходит копирование данных в буфер VMM (который он разделяет с приложением уровня пользователя на уровне физической памяти). После копирования определяется, нужна ли какая-либо обработка данных на стороне VMM, и нужна ли последующая обработка на стороне приложения уровня пользователя. Если обработка на его стороне нужна (а чаще всего так и бывает), то после окончания обработки в VMM происходит HyperSwitch, и управление передается приложению уровня пользователя.

3. На стороне приложения уровня пользователя запрос первым делом попадает к диспетчеру OTG запросов. Тот смотрит на идентификатор запроса, и определяет его текущее состояние. Если запрос имеет состояние SEND, то это означает, что данные передаются с гостевой на хостовую операционную систему. В этом случае данные считываются в буфер для входящих данных и запрос завершается. Если запрос находится в состоянии EXECUTE, то это означает, что все данные с гостя получены, и запрос отправляется на обработку к обработчику данного запроса. Обработчик определяется из данных, посланных в регистрах вместе с запросом и может вызвать другую функцию для обработки, например, функцию драйвера нужного устройства. Наконец, если запрос находится в состоянии RECEIVE, то это означает, что данных с гостя нет, а нам надо передавать те данные, которые вернул нам наш обработчик запроса. В этом случае мы забираем часть выходного буфера, размером 4 килобайта, и передаем ее гостю посредством предоставленного совместного буфера (просто копируем туда данные).

В итоге, данный интерфейс позволяет передавать нам данные с гостя или получать их. Однако, он оперирует буферами размером 4 килобайта, что не подходит для передачи данных для ресурсозатратных устройств. Нам требуется более эффективный и удобный интерфейс, которым можно было бы пользоваться для передачи данных с любого уровня стека драйверов гостевой операционной системы.

В последующих двух частях описывается результат данного исследования – два варианта интерфейса, основывающиеся на вышеописанном механизме передачи данных Open ToolsGate.

# 6. Базовый вариант интерфейса для передачи данных устройств на основе Open ToolsGate

Концептуальная схема базового варианта интерфейса для передачи данных между гостевой и хостовой операционными системами представлен на рисунке 5.1.



Рис. 5.1 Базовый вариант интерфейса передачи данных

 Основная преимущество данного интерфейса по сравнению с базовым механизмом OTG, это возможность использовать буферы данных, размером более 4 килобайт. Для достижения этой цели, мы создаем статическую библиотеку в гостевой операционной системе, которая будет поставляться всем пользователям данного интерфейса (вместе с заголовочным файлом, содержащим определение функций интерфейса). Функция, поставляемая данной библиотекой, будет принимать на вход гостевой пользовательский буфер с данными и возвращать указатель на возвращаемый буфер с данными. Внутри себя функция производит разбивку пользовательского буфера на фрагменты, размером 4 килобайта, а затем производит циклическую посылку этих фрагментов с одним и тем же идентификатором операции. После отправления всех сегментов данных, функция ожидает получения возвращаемого буфера. Буфер принимается аналогичным способом – по 4-рех килобайтным сегментам. После получения всех сегментов, из них собирается исходный буфер и отправляется на выход функции интерфейса. Благодаря этому, пользователь данного интерфейса со стороны гостевой операционной системы может не беспокоится по поводу деталей работы базового механизма OTG, а использовать достаточно простой интерфейс для обмена данными с хостовой операционной системой. Стоит также заметить, что изменения, вносимые на хостовую часть механизма OTG, довольно незначительны, т.к. в рамках диспетчеризации входящих OTG запросов существует механизм сборки/разбиения буферов из сегментов.

 Тем не менее, данный механизм нельзя назвать достаточно эффективным, поскольку в рамках передачи буфера данных с гостевой на хостовую операционную систему при передачи каждого 4-рех килобайтного сегмента данных происходят следующие действия, сильно уменьшающие общую скорость передачи данных:

 1. VMexit при переключении с гостевой операционной системы на VMM.

 2. Копирование данных из гостевой памяти в общий буфер VMM и приложения уровня пользователя.

 3. HyperSwitch при переключении контекста между VMM и хостовой операционной системой.

 В расширенной модели интерфейса все эти недостатки устранены в значительной степени.

# 7. Расширенный вариант интерфейса для передачи данных устройств на основе Open ToolsGate

 Концептуальная схема расширенного варианта интерфейса для передачи данных между гостевой и хостовой операционными системами представлен на рисунке 6.1.



Рис. 6.1 Расширенный вариант интерфейса передачи данных

 В расширенном варианте интерфейс, предоставляемый пользователю, аналогичен интерфейсу базового варианта (тот же API, также поставляется в виде библиотека + заголовочный файл). Различие заключается в способе передачи данных.

 Когда пользователь вызывает функцию расширенного интерфейса для передачи данных, функция не передает буфер по 4-рех килобайтным сегментам (как это было в базовом механизме). Вместо этого, в 4-рех килобайтный буфер, используемый для передаваемых данных при вызове RDPMC, записываются адреса виртуальных страниц, на которых лежит гостевой пользовательский буфер. На каждую запись выделяется 4 байта из 4096. Адрес записывается не напрямую, а с 12-битным смещением (т.к. это адрес страницы, то эти биты нулевые). Это позволяет использовать 44-битный диапазон адресов, т.е. 16 Терабайт памяти.

 После вызова RDPMC и передачи управления в VMM, вызывается обработчик OTG запросов на стороне VMM. Внутри него мы используем интерфейс виртуальной машины для получения физического адреса тех страниц, линейные адреса которых ранее были записаны в буфер обмена. В итоге, проделав данную операцию для всех адресов из списка, мы получаем аналогичный буфер, только заполненный уже гостевыми физическими, а не виртуальными, адресами. Дополнительно, в рамках получения физического адреса страницы, делается проверка на присутствие страницы в физической памяти. Если страница отсутствует, то мы приостанавливаем выполнение кода обработчика, а затем эмулируем PAGE\_FAULT для данной страницы в гостевой операционной системе. Гость обрабатывает данный PAGE\_FAULT, а затем перезапускает инструкцию RDPMC, чтобы мы могли корректно получить гостевой физический адрес страницы. После того, как адреса всех страниц из списка конвертированы в гостевые физические, мы перекопируем буфер с адресами в буфер, разделяемый VMM и приложением уровня пользователя, и передаем второму управление.

 В приложении уровня пользователя, опять же, используется интерфейс виртуальной машины для считывания содержимого гостевых физических страниц (или для записи в них). В итоге, мы считываем содержимое всех физических страниц, содержащихся в списке, в специальный хостовый буфер, который передаем обработчику на стороне хоста. Возвращаемые данные передаются таким же способом (через запись по физическим адресам из списка).

 В результате, данный метод устранил все недостатки базового варианта интерфейса для обмена данными. При использовании расширенного метода, нам не нужно копировать все данные во время обработки в VMM (мы копируем только 4-рех килобайтный буфер со списком адресов). Также, поскольку адреса страниц буфера передаются на хостовую операционную систему за один вызов RDPMC (если размер буфера данных не превышает 4 Мегабайта), то и VMexit, и HyperSwitch происходит всего один раз за весь цикл обмена данными (при использовании базового метода, при передаче 4 Мегабайт данных, число переключений контекста равно 1024). Благодаря этому, скорость передачи данных расширенного метода должна значительно превосходить скорость базового метода.

 Стоит также заметить, что в случае данных, размером более 4 Мегабайт, требуется надстройка над расширенным механизмом передачи данных, которая будет разбивать буфер на 4-рех Мегабайтные сегменты и передавать их по отдельности. Данная надстройка, по своей концепции, практически идентична концепции базового интерфейса передачи данных.

# 8. Сравнение производительности вариантов

 В предыдущих двух главах были описаны два механизма передачи данных на основе OTG. Было сказано об их преимуществах и недостатках относительно друг друга. Для того, чтобы иметь более четкое представление о возможности использования данных интерфейсов для передачи данных между устройствами и гостевой операционной системой, был проведен ряд измерений скорости передачи данных в зависимости от размера передаваемых данных и механизма передачи (базовый или расширенный). Результаты этих измерений приведены в таблице 7.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 Мб | 2Мб | 3Мб | 4Мб |
| Базовый | 78 Мб/сек | 83 Мб/сек | 85 Мб/сек | 80 Мб/сек |
| Расширенный | 121 Мб/сек | 126 Мб/сек | 132 Мб/сек | 134 Мб/сек |

Табл. 7.1 Зависимость скорости передачи от механизма и размера данных

 Из таблицы видно, что скорость передачи данных с использованием базового механизма практически не зависит от размера данных. Это лего объясняется тем, что число ресурсозатратных операций при каждом запросе (копирование, VMexit, HyperSwitch) прямо пропорционально размеру данных.

 С другой стороны, скорость передачи данных с использованием расширенного механизма, хоть и превосходит скорость базового механизма, но всего лишь примерно в полтора раза, что довольно неожиданно, учитывая тот факт, что наиболее очевидные проблемы производительности были базового механизма были устранены в дополнительном. Также можно заметить, что с ростом объема данных, скорость растет очень медленно, что не согласуется с теорией, т.к. передача любого блока данных, размером до 4 Мегабайт, должна происходить за одно переключение из гостевой в хостовую операционную систему, т.е. примерно за одинаковое время. Все это указывает на наличие некоторых ресурсозатратных операции, которые не были учтены при расчетах и разработке расширенной версии передачи данных.

 Вместе с тем хотелось бы отметить, что скорость передачи данных у обоих механизмов довольно высока (для сравнения, для передачи видеопотока качества Full HD с частотой кадров 60 в секунду требуется канал со скоростью примерно 200Мб/сек). Это означает, что даже данные варианты интерфейса вполне применимы в качестве каналов обмена данными между устройствами и гостевой операционной системой.

# 9. Дальнейшие идеи по расширению номенклатуры

Прежде всего хотелось бы отметить, что расширенный вариант интерфейса передачи данных не является законченным в текущем виде. При разработке способа передачи данных не было учтено влияние некоторых операций на производительность интерфейса. В связи с этим, дальнейшая работа по улучшению расширенного интерфейса имеет место быть. Например, потенциально ресурсозатратной может быть операция получения физического адреса страницы из виртуального в VMM, которая проводится над каждой страницей из списка. В качестве альтернативы, можно попробовать не конвертировать адрес каждой страницы, а просто скопировать соответствующую таблицу страниц из дерева отображения памяти текущего гостевого процесса (возможно сделать средствами виртуальной машины).

 Так же, необходимо протестировать данные механизмы непосредственно при передачи данных для конкретных устройств. Наиболее разумно в качестве теста использовать паравиртуализацию камеры, т.к. её диапазон необходимой скорости передачи данных примерно равен скорости передачи данных с использованием механизмов на основе OTG.

 Наконец, следует окончательно интегрировать данный интерфейс в систему модулей виртуальных устройств для предоставления возможности виртуализовать любые устройства на ее основе. При этом, не исключено, что придется добавить еще какие-либо интерфейсы (например, интерфейс передачи прерываний).

# Заключение

В заключении хотелось бы обобщить все результаты данной дипломной работы:

 1. Было проведено исследование касательно возможности расширения номенклатуры виртуальных устройств путем расширения интерфейса взаимодействия устройств с гостевой операционной системой. Были рассмотрены различные типы виртуализации устройств, и, в итоге, было решено использовать паравиртуализацию в качестве способы виртуализации устройств в деле расширения интерфейса взаимодействия.

 2. Был изучен базовый механизм передачи данных с гостевой на хостовую операционную систему – Open ToolsGate. Рассмотрены его сильные и слабые стороны.

 3. Была написана гостевая библиотека, реализующая базовый интерфейс передачи данных между гостевой и хостовой операционными системами на основе OTG. Были сформулированы главные недостатки данной модели.

 4. Была написаны гостевая библиотека, а также были внесены изменения в код гипервизорного виртуализационного решения, с целью реализации расширенного интерфейса передачи данных на основе механизма OTG, в котором были устранены недостатки базовой версии интерфейса. Также, было рассмотрено дальнейшее направление улучшения данного интерфейса.

 5. Были проведены тесты, связанные с паравиртуализацией камеры, с использованием расширенного механизма передачи данных. По результатам тестов была показана состоятельность данного механизма применительно к использованию в виртуализации устройств, требующих широкий канал передачи данных.

# Список литературы

1. searchservervirtualization.techtarget.com URL: http://searchservervirtualization.techtarget.com/ (дата обращения: 20.06.2016).

2. Chris Wolf, Erick M. Halter Virtualization: From the Desktop to the Enterprise. 2005.

3. Dan Kusnetzky Virtualization: A Manager's Guide. 1 изд. .

4. Thomas Olzak, James Sabovik, Jason Boomer Microsoft Virtualization: Master Microsoft Server, Desktop, Application, and Presentation Virtualization. 1 изд. .