Санкт-Петербургский государственный университет
Математико-механический факультет

Программная инженерия

Черепанов Алексей Олегович

Разработка unikernel операционной системы на основе Embox

Бакалаврская работа

Научный руководитель:
проф. каф. СП, д.ф.-м.н., проф. А. Н. Терехов

Рецензент:
Разработчик
ООО «Эмбокс» А. И. Калмук

Санкт-Петербург
2020

SAINT PETERSBURG STATE UNIVERSITY

Software Engineering

Cherepanov Aleksei

Development of unikernel operating system based on Embox

Bachelor's Thesis

Scientific supervisor:
professor Andrey Terekhov

Reviewer:
Developer,
«Embox» Co. Ltd Aleksandr Kalmuk

Saint Petersburg
2020

Оглавление

Введение

Современные компьютеры имеют огромные ресурсы, которые они далеко не всегда используют полностью. Модель, когда серверы в определённые моменты времени перегружены, а большую часть времени незначительно используют вычислительную мощность машины или вовсе простаивают, является экономически невыгодной, неэффективной и сложной в управлении. В качестве решения данной проблемы можно вместо множества серверов использовать виртуализацию, которая позволяет запускать одну или несколько операционных систем в рамках другой операционной системы на одной вычислительной машине. Виртуальные машины, которые работают на одном компьютере, в контексте мультизадачности позволяют задействовать все ресурсы рабочей станции и, как следствие, оптимизировать энергопотребление. Виртуализация предоставляет повышение изоляции, то есть снижение вероятности сбоев от взаимного влияния программ; безопасность благодаря распределению задач администрирования. Виртуализация позволяет ограничить права каждого администратора только самыми необходимыми. В контексте описанной проблемы существенную роль играет распределение ресурсов, когда каждая виртуальная машина получает столько ресурсов, сколько ей необходимо, но не более того. Виртуализация предоставляет портативность и возможность дешёвого клонирования системы: легко мигрировать виртуальную машину с одного физического компьютера на другой.

Несмотря на то, что виртуальные машины могут работать на одном и том же физическом оборудовании, они по-прежнему логически отделены друг от друга. Обычно оптимальное разделение ресурсов памяти и процессорного времени между виртуальными машинами достигается при помощи гипервизора. Гипервизор управляет физическими ресурсами вычислительной машины и распределяет эти ресурсы, позволяя запускать операционные системы одновременно. Существует два типа гипервизоров. Гипервизоры 1-го

типа, иногда называемые «автономными гипервизорами», запускаются непосредственно на аппаратном обеспечении хост-машины для управления оборудованием и гостевыми виртуальными машинами. Одним из ярких примеров гипервизоров 1-го типа является гипервизор Xen [20]. Xen представляет собой кроссплатформенный гипервизор, поддерживающий не только аппаратную виртуализацию, но и паравиртуализацию. Его отличительной особенностью является минимальный объем кода, поскольку большая часть компонентов вынесена за его пределы. За счет поддержки паравиртуализации и аппаратной виртуализации Xen относят также к гибридному типу гипервизоров. Ко 2-му типу гипервизоров относятся гипервизоры, которые запускаются на обычной ОС, как и другие приложения в системе. В этом случае гостевая ОС выполняется как процесс на хосте, а гипервизоры разделяют гостевую ОС и ОС хоста. Одним из самых популярных гипервизоров 2-го типа является Oracle VM Virtualbox [12]. Главными его особенностями являются модульность всей системы, возможность работы виртуальной машины в качестве терминального сервера, а также возможность использовать внешние устройства как виртуальные диски без дополнительной поддержки устройства со стороны виртуальной операционной системы.

Гипервизор предоставляет намного более высокую степень изоляции, чем процесс в ОС. Так, если мы хотим добиться от системы безопасности, то мы будем запускать гипервизор, а внутри будем запускать операционные системы, нацеленные на безопасность, но в итоге мы получаем неоптимальное использование ресурсов: излишний слой обеспечения безопасности операционной системы поверх слоя предоставляемого гипервизором. Для решения этой проблемы можно обратить свое внимание в сторону паравиртуализации (когда операционная система знает, что она находится в виртуальной среде) unikernel систем. Unikernels — это специализированные образы машин [17] с единым адресным пространством, созданные с использованием библиотечных операционных систем. Обычно unikernel системы используются для сетевых приложений и запускаются при

помощи гипервизора. Виртуальные машины редко используются в качестве основы для unikernel систем, потому что имеют более сложный интерфейс по сравнению с гипервизором. Unikernel системы используются для экономии ресурсов. В данном примере unikernel убирает накладные расходы на операционную систему, работающую на гипервизоре. Ведь по сравнению с полноценной ОС, в unikernel значительно меньше излишней функциональности, что облегчает разработку и ускоряет работу системы. Во-вторых, unikernel системы предоставляют усиленную безопасность, которая достигается за счет ограничения количества функций, включенных в ядро, а также за счет его уникальности. Если рассматривать вектор атаки на операционную систему с повышением прав доступа, то в обычной операционной системе злоумышленник может получить полный доступ ко всем системным функциям операционной системы. В случае с unikernel системой злоумышленник, добравшись до внутреннего уровня привилегий, получает лишь ограниченный набор функциональности, заложенный разработчиком для минимального функционирования системы под конкретную задачу.

Рассматривая облачные системы (среда, в которой ресурсы сдаются в аренду и оплачиваются в зависимости от использования), использование unikernel систем означает грамотное использование ресурсов и инфраструктуры. К плюсам такого выбора также можно добавить маленький размер облака и отсутствие лишних компонентов в облаке, кроме необходимых инструментов, что также повышает уровень безопасности.

Важным аспектом является то, что unikernel системы создаются с прицелом на использование одного приложения, то есть многие привычные механизмы и интерфейсы классических операционных систем отсутствуют. С одной стороны, это позволяет экономить ресурсы, но с другой — это также увеличивает сложность разработки и интеграции третьесторонних компонентов. В качестве примера можно привести unikernel операционную систему MirageOS [11], одним из основных идеологических аспектов которой является

написание компонентов на высокоуровневом функциональном языке программирования OCaml. Таким образом, данное решение не оставляет разработчикам другого пути, кроме как реализовывать необходимые компоненты самостоятельно на языке OCaml, игнорируя аналоги. Например, MirageOS включает в себя функциональные реализации в чистом виде таких протоколов, как TCP/IP, DNS, SSH, HTTP и XMPP. Используемая в данной работе ОСРВ Embox [31] является POSIX-совместимой ОС, что значительно облегчает интеграцию уже готовых компонентов. Встроенная система сборки Mybuild [30] позволяет интегрировать третьесторонний код вместе со скриптами сборки, что часто позволяет переносить программы и библиотеки из среды GNU/Linux с нулевыми или минимальными изменениями. Например, консольный текстовый редактор nano был интегрирован[1] в Embox путем добавления его исходного кода в систему сборки без внесения каких-либо дополнительных изменений.

Если же рассматривать Embox в качестве unikernel системы, то можно выделить следующие аспекты. Embox во многом разрабатывалась для встроенных систем, а потому ориентирована на малое потребление ресурсов. Embox имеет встроенную систему сборки Mybuild, позволяющую добавлять только необходимые модули, обеспечивая безопасность. В контексте unikernel систем Embox имеет преимущество в виде POSIX-совместимости, что позволяет использовать уже существующие компоненты, а не переписывать их с нуля. Таким образом можно рассматривать Embox как хорошего кандидата на unikernel систему.

Использование POSIX-совместимого эмбокса в качестве unikernel системы позволит с низкими расходами ассоциировать преимущества POSIX с преимуществами unikernel систем. Однако, Embox портирован на платформу гипервизора Xen не полностью: реализованы портирование, таймер, вывод отладочной информации, но этого недостаточно. Для того, чтобы использовать Embox в качестве unikernel системы, не хватает реализации сетевого интерфейса, а также

[1] https://github.com/embox/embox/tree/master/third-party/nano

всех сопутствующих механизмов, включая механизм общей памяти, разделяемой между доменами.

1. Постановка задачи

Целью данной работы является создание прототипа unikernel на базе ОСРВ Embox для архитектуры гипервизора Xen. Для достижения поставленной цели были сформулированы следующие задачи.

- Провести обзор предметной области, изучить архитектуру гипервизора Xen и ОСРВ Embox, изучить существующие решения
- Реализовать и интегрировать механизм общей памяти в ОСРВ Embox для поддержки архитектуры гипервизора Xen
- Реализовать и интегрировать сетевой интерфейс в Embox
- Оценить сетевую производительность Embox в качестве unikernel системы

2. Обзор предметной области

2.1. Обзор ОСРВ Embox

Embox — это кроссплатформенная операционная система реального времени на языке Си, в которой весь архитектурно-зависимый код (например, ядро, драйверы, приложения командной строки, файловая система и т.п.) представлен в виде отдельных модулей. Модули включаются в сборку с помощью встроенной системы Mybuild. Благодаря такому решению достаточно всего лишь прописать в конфигурационном файле необходимые модули, а Mybuild соберёт также и все зависимости этих модулей. Это позволяет создать минимальное по объёму решение для конкретной задачи. С точки зрения unikernel систем это позволяет создать единое с ядром приложение, исключая излишнюю функциональность. Более того, это увеличивает безопасность системы: гарантируется, что невозможно выполнить приложения, не включенные в конечный образ.

Немаловажным фактом является то, что Embox предоставляет слой POSIX-совместимости, что позволяет разработчикам повторно не реализовывать уже существующие решения под целевую платформу, а использовать внушительное количество готового ПО, изначально разработанного под Linux. Это позволяет сократить время и стоимость разработки.

2.2. Обзор гипервизора Xen

Xen Project - гипервизор гибридного типа, позволяющий нескольким ОС исполняться на одном аппаратном обеспечении одновременно. Xen представляет из себя “тонкий” гипервизор: большая часть функциональности вынесена за его пределы и делегирована специально выбранной операционной системе. Xen стартует на аппаратном обеспечении без операционной системы и запускает виртуальную машину Domain0 Host (далее: dom0). На практике это обычно дистрибутив Linux. При помощи специальных инструментов Xen

можно запускать дополнительные виртуальные машины: DomainU Guest (далее: domU). Dom0 отвечает за фактическое взаимодействие с аппаратурой, такой как блочные устройства или сетевая карта. Чтобы виртуальные машины типа domU могли общаться с аппаратурой, Xen прикрепляет к каждой domU виртуальные устройства. Модель разделённого драйвера — основной механизм построения драйверов в архитектуре Xen (рис. 1). У самого гипервизора Xen нет драйверов периферийных устройств. Однако эти драйверы есть у Dom0. Общие драйверы передают данные на диск или пакеты на отправку настоящей аппаратуре через dom0. Обе части могут быть реализованы как модули ядра и загружаться динамически.

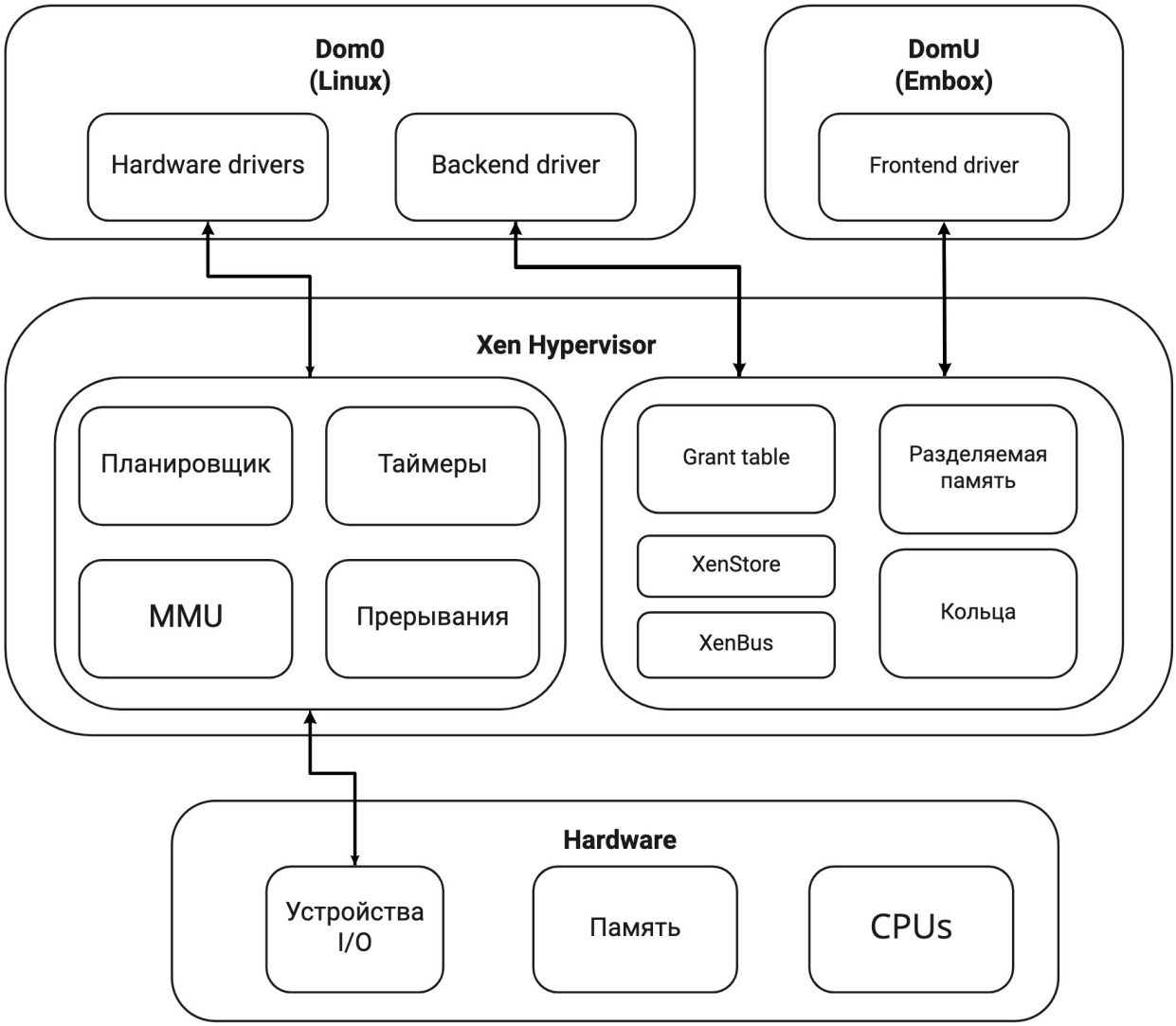

Рис. 1: Модель разделённого драйвера в архитектуре гипервизора Xen

Связь DomU с железом происходит посредством разделённых драйверов. На стороне DomU реализован фронтенд драйвер, на стороне Dom0 — бэкенд. Они связываются друг с другом с помощью механизма разделяемой памяти, механизма Grant table, каналов событий и механизма передачи управляющих команд XenStore.

Суть такого подхода в простоте и скорости. В случае обычной виртуализации для гостевой ОС эмулируется железо, например, определённая сетевая карта, и эта ОС взаимодействует с ней через драйвер для конкретного железа. Гипервизор в этом случае должен транслировать команды, отправляемые гостевой ОС на эмулированный процессор, в коды для реального процессора, на котором непосредственно запущен сам гипервизор. Это приводит к большим затратам. В случае с Xen мы используем паравиртуализацию. Здесь гостевая операционная система знает, что она находится в виртуальной машине. У неё должен быть специальный драйвер, который работает с абстрактным железом без архитектурной специфики. Гостевая ОС присылает специальные простые команды гипервизору, который быстро обрабатывает их и переводит в коды настоящего аппаратного обеспечения.

У гостевой машины есть возможность обращаться к гипервизору при помощи гиперколов — интерфейса гипервизора. Эти специальные команды позволяют передать поток исполнения в ядро и выполнить операцию с повышенными привилегиями. Например, операции с памятью, кроссдоменные сообщения и многое другое. Это позволяет гипервизору контролировать каждое изменение состояния системы и пресекать небезопасные вызовы гостевых доменов.

2.3. Обзор существующих решений

В данном разделе будут рассмотрены существующие реализации domU с открытым исходным кодом для архитектуры гипервизора Xen. Это необходимо для определения основных подходов к реализации и выявления опорных технологий, которые необходимо реализовать

в Embox, а также для определения реализаций, с которыми будет проводиться сравнение производительности.

2.3.1. Unikernels

Haskell Lightweight Virtual Machine [4] является unikernel системой, работающей поверх гипервизора Xen. Время загрузки составляет миллисекунды, а объем используемой памяти составляет всего 4 MB. HaLVM представлена в виде набора библиотек, соединенных с Xen поверх основного порта. Разработчики могут выбрать необходимые библиотеки для разных уровней работы, а неиспользуемые библиотеки просто не будут добавлены в конечный образ. HaLVM написан на языке Haskell — это высокоуровневый, ленивый, чисто функциональный язык. Он позволяет не только конструировать сложные программы при помощи абстракций и механики композиции, но и получать доступ к памяти напрямую, чтобы реализовывать драйверы, а также для того, чтобы писать императивные участки низкоуровневого кода.

Одной из наиболее популярных операционных систем для Xen является MirageOS. Она является ярким примером unikernel системы, запускаемой на гипервизоре, и используется для безопасных высокопроизводительных сетевых приложений для различных облачных вычислений и мобильных платформ. В ней реализованы все механизмы для общения с гипервизором Xen, однако данная ОС полностью написана на высокоуровневом функциональном языке программирования OCaml.

Другая unikernel система, работающая с гипервизором Xen — LING [7]. Она позволяет запускать приложения, написанные на функциональном языке Erlang, без операционной системы. От момента запуска виртуальной машины до начала работы приложения проходит всего 50 миллисекунд — это примерно в сотни раз меньше, чем нужно для запуска ОС Linux в сочетании с Erlang. Размер образа машины при этом составляет примерно 1 Мб. LING и MirageOS используют высокоуровневые языки и вычислительное окружение, чтобы предоставить API к функциональности операционной системы.

2.3.2. Mini-OS

Другой операционной системой, работающей как гость с гипервизором Xen, является Mini-OS [10]. Это крошечное ядро ОС, распространяемое вместе с исходным кодом Xen Project Hypervisor. Оно было разработано в рамках университетского проекта и в основном используется в качестве операционной системы для доменов-заглушек при разработке Dom0. Кроме того, Mini-OS используется в качестве обучающего репозитория и основы для разработки unikernel систем в роли гостя в окружении гипервизора Xen. Mini-OS можно считать минимально жизнеспособным продуктом с минимальным набором реализованных функций, что позволяет изучить реализацию взаимодействия с гипервизором, не отвлекаясь на внутреннюю архитектуру операционной системы. Однако это не позволяет использовать Mini-OS в качестве тестового стенда, потому что из-за ограниченных возможностей оказывается невозможным развернуть тестовый сервер на его основе.

2.3.3. Linux

В репозитории Linux также есть реализация драйвера сети как бэкенда для dom0 [8], так и фронтенда для domU [9]. В Linux представлено универсальное решение, которое максимально покрывает всю возможную функциональность гипервизора Xen. Но данное решение обладает множеством зависимостей от внутренней архитектуры ядра Linux, что не позволяет сделать полноценную легковесную unikernel систему. Однако Linux является открытым программным обеспечением, что позволяет множеству людей дополнять и модернизировать его. Общие знания различных специалистов приводят к тому, что Linux имеет одну из самых производительных реализаций.

Операционная система Alpine [1] является легковесным дистрибутивом, основанным на Linux и ориентированным на безопасность. Alpine запускается на гипервизоре Xen, а также

имеет POSIX-совместимый интерфейс, что позволяет провести сравнительное тестирование сетевой производительности, исключая из сравнения производительность тестового сервера. При этом Alpine переняла у Linux все преимущества сетевого стека. Более того, Alpine предоставляет открытый исходный код, что позволяет всем желающим разработчикам внести свой вклад и улучшить имеющиеся компоненты.

Одной из немногих unikernel систем, поддерживающих POSIX-совместимый интерфейс, является Rumprun [14]. Rumprun построен на rump kernels [13], что позволяет создать необходимый программный стек, включая необходимые для конкретной задачи компоненты в ядро. Это в свою очередь позволяет исполнять существующее неизменённое POSIX-совместимое программное обеспечение в качестве unikernel системы. Однако для функционирования программного стека Rumprun нуждается в компонентах, которые идейно похожи на драйверы, потому что тесно связаны с операционной системой. Разработчики предоставляют некоторые бесплатные многокомпонентные драйверы, которые можно включить в ядро. Например, различные файловые системы, драйверы устройств PCI и стеки протоколов TCP/IP и SCSI.

2.3.4. Вывод

В данном разделе были рассмотрены популярные unikernel системы, работающие на гипервизоре Xen. Всех их объединяет тот факт, что они написаны на функциональных языках программирования, что в свою очередь сделано для обеспечения безопасности и корректности за счёт производительности. В контексте данной работы этот факт исключает возможность объективного тестирования реализаций сетевого драйвера. Более того, этот факт затрудняет запуск существующих POSIX-совместимых приложений на базе этих unikernel систем, которые можно было бы использовать в качестве тестовых. Иначе их пришлось бы реализовать заново на функциональных языках программирования, на которых написано само ядро.

MiniOS был выбран для изучения архитектуры разделённых

драйверов, а также в качестве опорной минимальной реализации фронтенд драйвера. В роли хост-домена был использован дистрибутив Linux, поэтому его исходный код использовался для изучения поведения бэкенд драйвера во время инициализации и взаимодействия с фронтенд драйвером Embox. Легковесный дистрибутив Alpine Linux был выбран для получения эталонных значений в процессе тестирования, потому что Alpine перенял у Linux все преимущества сетевого стека.

3. Реализация механизма общей памяти

3.1. Развертывание окружения разработки

Использование физической машины для разработки и тестирования может повлечь за собой большие накладные расходы на переустановку системы в случае критических ошибок. Учитывая оценку рисков частоты появления таких критических состояний системы, было решено развернуть специальную среду разработки с использованием Vagrant [18] в связке с Virtualbox [12]. Это позволило тестировать систему изолированно и детерминировано с точки зрения начального состояния системы.

В качестве хост-домена dom0 был использован дистрибутив Ubuntu 18.04 LTS «Bionic Beaver» [16] — дистрибутив с длительной поддержкой — с установленным пакетом программ Xen [29]. В качестве domU использована ОСРВ Embox.

В процессе разработки я научился изменять бэкенд драйвер сети на стороне Dom0 для получения отладочной информации о текущем состоянии системы. Мной были написаны Bash-скрипты для быстрой модификации и подключения измененного драйвера к ядру Linux. Для отладки Embox в окружении Vagrant были настроены порты для подключения удалённого дебаггера gdb [23]. Также настроено удалённое подключение редактора исходного кода с графической оболочкой VSCode [19] к файловой системе dom0 для модификации исходного кода бэкенд драйвера. Для непрерывной интеграции и тестирования был использован уже настроенный Travis CI[2] в связке с GitHub. Всё это позволило ускорить процесс поиска ошибок и, как следствие, ускорить разработку в целом.

В процессе разработки также использовалась утилита arping [21] для отладки процесса приема пакетов драйвером на стороне Embox до того, как он был подключен к сетевому стеку TCP/IP. Это позволило проверить целостность входящих сетевых пакетов. Утилита

[2] `https://travis-ci.org/github/embox/embox`

arping запускалась на 60 секунд совместно с запущенной программой отслеживания сетевых пакетов tcpdump [28], которая выводила все пакеты в шестнадцатеричном формате. В процессе данного тестирования пакеты также выводились на стороне Embox для выявления несоответствия.

3.2. Выделение памяти и трансляция адресов

Гипервизор Xen имеет страничное распределение памяти с размером страницы 4КБ. В терминах архитектуры Xen существуют 3 типа адресов страниц (рис. 2). Первый — Machine frame number (MFN) — номер страницы в реальном адресном пространстве гипервизора. Архитектура памяти Xen представляет ещё один уровень трансляции аналогично тому, как операционная система транслирует виртуальные адреса процесса в машинную память: во время инициализации гостевого домена Xen выделяет машинную память, которую впоследствии транслирует, тем самым предоставляя механизм изоляции доменов друг от друга. При этом ядро каждого домена после трансляции видит полученный диапазон адресов как непрерывное адресное пространство, будто он запущен непосредственно на аппаратном обеспечении без операционной системы. Такие адреса называются Pseudo-physical frame number (PFN) — псевдофизические номера страниц. Данные страницы ядро самостоятельно транслирует в Virtual frame number (VFN) — виртуальные адресные пространства для дочерних процессов внутри операционной системы. Отличительной особенностью архитектуры Xen является то, что гостевой домен может не просто знать о существовании машинных адресов, но и манипулировать ими с помощью гиперколов. Так, например, в Xen есть понятие автотрансляции машинной памяти в псевдофизические адреса ядра ОС, однако, эта функция недоступна для паравиртуализированных доменов. В рамках данной работы реализована обработка исключений в процессе инициализации драйвера, связанных с возможным неподдерживаемым окружением, в

котором происходит запуск Embox.

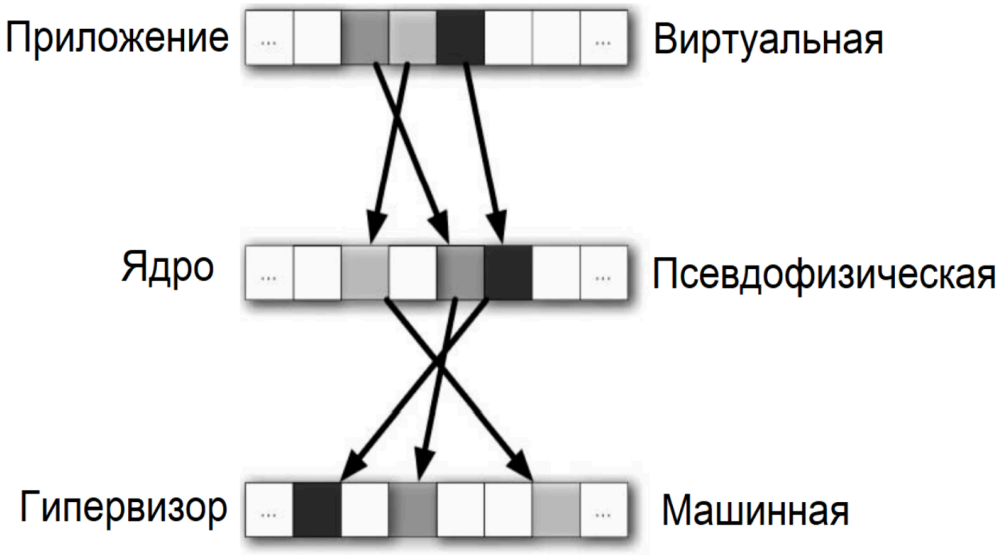

Рис. 2: Структура памяти в Xen

В условиях поставленной задачи я реализовал инициализацию памяти следующим образом. При инициализации нового домена гипервизор Xen выделяет страницу общей памяти, доступной как гипервизору, так и гостевому домену, в которую располагает структуру с необходимой конфигурационной информацией об окружении домена. Именно оттуда Embox извлекает необходимые адреса начала и конца адресного пространства, таблицу трансляции псевдофизических адресов домена в машинные адреса гипервизора, а также адрес начала таблицы страниц. Стоит отметить, что таблица страниц является архитектурно зависимым элементом системы. Embox поддерживает 32-х битную архитектуру x86, а конфигурация Xen для запуска Embox в качестве гостевой системы предполагает таблицу страниц с 3 уровнями вложенности.

Для всех доступных псевдофизических адресов запускается итерация с шагом в 4КБ (размер страницы памяти). Каждый рассматриваемый адрес разбивается с помощью макросов на 3 смещения внутри таблиц страниц разного уровня. И если отсутствует таблица какого-то уровня или запись в таблице, то именно здесь происходит выделение и инициализация ячейки памяти с помощью обращений к гипервизору.

После того, как таблица страниц построена, необходимо инициализировать аллокатор страниц. В Embox уже был реализован аллокатор страниц для архитектуры x86, однако модель его работы не подходила для взаимодействия с гипервизором Xen. Существующий аллокатор выбирал область памяти для разметки, исходя из статической разметки образа ядра, загружаемого в оперативную память. Однако во время инициализации ОС в качестве гостевого домена в паравиртуализированном окружении гипервизора Xen записывает в начало адресного пространства структуру с конфигурационной информацией, поэтому неизвестно, где будет первый доступный для домена псевдофизический адрес. Чтобы решить эту проблему, я переиспользовал функциональность существующего аллокатора, но изменил его под конкретную задачу, добавив возможность динамически выбирать адреса начала и конца памяти для разметки.

Все операции по управлению памятью производятся с помощью гиперколов. Однако как аргументы принимаются в основном машинные адреса. Именно поэтому в Embox необходимо было реализовать трансляцию виртуальных адресов в псевдофизические, а псевдофизические в машинные с помощью таблицы соответствия. Адрес этой таблицы располагается в структуре конфигурационной информации, которую гипервизор создал во время начальной инициализации. В процессе построения таблицы страниц, которая доступна не только гостевому домену, но и хост-домену, таблица трансляции постепенно заполнялась машинными адресами при помощи гипервизора.

Полученная реализация была выделена в отдельный модуль и интегрирована во встроенную систему сборки Mybuild. Реализован интерфейс для доступа к функциям выделения страниц и трансляции адресов. Определены необходимые зависимости, и модуль включён в сборку.

3.3. Механизм Grant Table

Grant table — это механизм реализации общей памяти для Xen доменов. У каждого домена есть своя собственная таблица, в которой описаны все страницы памяти, доступ к которым он предоставляет другим доменам. Также там описано, какие права какому домену предоставляются. Grant table представляет из себя массив структур, в которых хранится информация об адресах, доменах и правах доступа. Доступ осуществляется через Grant reference, а именно через индекс по таблице. Таким образом, физическое устройство страницы памяти инкапсулируется.

Реализация Grant table в Embox подразумевает вызов гиперкола для запроса у гипервизора индивидуальной для домена Grant table. Результатом данного вызова являются машинные адреса страниц, где находится Grant table, сформированный гипервизором Xen. Получив адреса, Embox локально выделяет память для будущей таблицы и выполняет ещё один гиперкол, который отображает страницы, полученные от гипервизора в локальное адресное пространство Embox. Таким образом, Embox получает доступ к своей Grant table.

В Embox я реализовал Grant table как отдельный модуль. Также был определён интерфейс. Функция предоставления доступа принимает машинные адреса страниц, а также идентификатор домена и права в виде битовой маски. Стоит отметить, что любые изменения прав доступа являются изменением значений в соответствующей структуре, а гипервизор Xen осуществляет защиту прикрепленных страниц от несанкционированного доступа.

Grant table интегрирован в Embox как модуль во внутреннюю систему сборки Mybuild. Был продуман интерфейс взаимодействия с модулем Grant table. Также были прописаны зависимости модулей для удобства сборки.

4. Реализация сетевого интерфейса

В процессе изучения существующих решений и спецификаций гипервизора Xen был выявлен набор технологий, необходимых для успешного функционирования фронтенд драйвера сети. Процесс создания сетевого драйвера Embox для архитектуры гипервизора Xen можно поделить на следующие этапы [2].

4.1. Протокол обмена сообщениями

Для взаимодействия между частями драйверов паравиртуализированых устройств в окружении гипервизора Xen используется такая структура данных, как кольцо или циклический двусвязный список. Они предоставляют простую абстракцию передачи сообщений, построенную над механизмом разделяемой памяти, предоставляемым гипервизором.

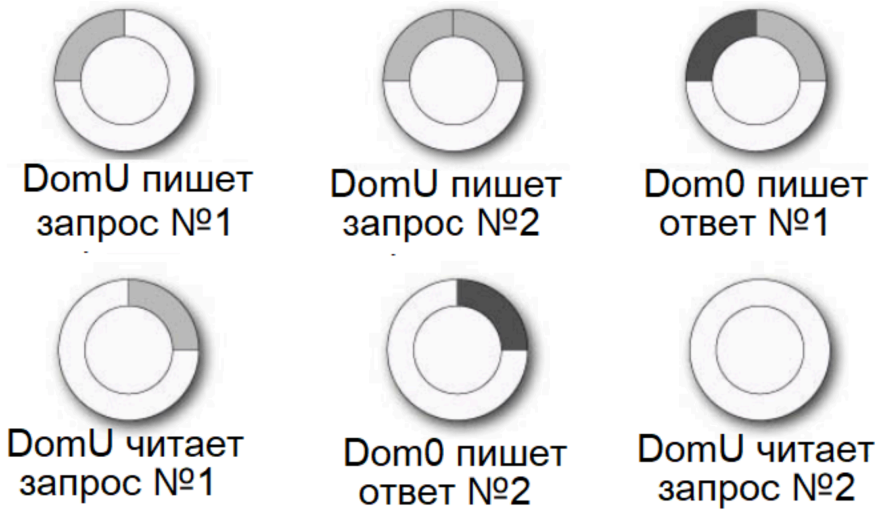

Рис. 3: Протокол обмена сообщениями между частями драйвера

I/O кольца предоставляют метод асинхронной коммуникации между доменами. Один домен кладет запрос в кольцо (рис. 3), в то время как другой забирает его и кладет в кольцо ответ, причем оба они пишутся в одну и ту же область памяти. Защита от переписывания одним ответом другого запроса достигается за счёт того, что структура запроса и структура ответа совпадают по

размеру. Кольцо также имеет внутренние приватные и публичные поля, которые хранят информацию о текущем расположении указателей: указатель продюсера опубликованных запросов, указатель потребителя опубликованных запросов, а также приватные указатели на ещё не опубликованные новые запросы и на неопубликованные прочитанные ответы.

Для синхронизации двух доменов, изменяющих один и тот же разделяемый участок памяти, были использованы барьеры — макросы с вставками ассемблерного кода. Вызов барьера гарантировал консистентность данных в кольце. Для разных задач используются разные барьеры: барьеры для записи и для чтения. Барьеры вызываются непосредственно после обновления данных в кольце и перед оповещением другой части драйвера.

Xen предоставляет интерфейс для инициализации и разметки областей памяти как кольца для передачи сообщений. Фронтенд должен выделить область памяти и проинициализировать на ней структуры кольца. После этого он должен вычислить машинный адрес страницы, на котором находится каждое кольцо, и с помощью механизма Grant table разрешить доступ хост-домену к кольцам.

Для сетевого драйвера было реализовано два кольца: одно для приема сообщений и другое для передачи. Через кольца передаются только управляющие запросы и ответы, а сами данные пишутся в страницы общей памяти с последующим использованием механизма Grant table. В процессе обмена сообщениями используются макросы для получения элемента кольца или для оповещения другой стороны драйвера о новых данных.

События — способ оповещения в Xen. Физические прерывания конвертируются в события и направляются соответствующим доменам. Посредством событий происходит оповещение между частями разделённых драйверов, например, о поступлении новых данных или о завершении обработки запроса или ответа. Все события проходят через Event channel, один “конец” которого находится в dom0, а второй в domU. Данный механизм уже был реализован в Embox.

Процесс получения пакета выглядит следующим образом. Кольцо для получения сообщений полностью заполняется запросами на этапе инициализации. Запросы включают в себя идентификационный номер и индекс в таблице Grant table, по которому лежит пустая страница с разрешением на запись. Когда приходит пакет, бэкенд берет запрос из кольца и пишет на его место ответ, в котором указывается Grant reference в Grant table на страницу, куда был записан пакет и сдвиг. Указатель продюсера запросов сдвигается, и с помощью макросов отправляется оповещение через определённый Event channel, что состояние указателей обновилось, а значит, настало время прочитать ответ. Фронтенд драйвер получает прерывание и вызывает функцию обработки пакета. Страница проверяется на использование другим доменом по флагам в структуре Grant table и освобождается. Пакет читается и отправляется в сетевой стек. Происходит проверка, не пришли ли новые пакеты. Если да, то обработка продолжается. В конце обработки указатель потребителя ответов сдвигается. Данный подход экономит процессорное время, ведь бэкенд видит все указатели, а потому не отправляет лишних прерываний, если очередь обработки пакетов не пуста. Если очередь заполнена, пакеты просто отбрасываются.

Кольцо для отправки запросов изначально пустует и заполняется только по необходимости, если возникнет пакет для отправки. Когда это произойдет, фронтенд пишет запрос в страницу, заранее закрепленную за запросом. Затем происходит трансляция виртуального адреса страницы сначала в псевдофизический, а затем с помощью таблицы трансляции в машинный, понятный гипервизору. С помощью механизма Grant table разрешаем доступ host-домену, после чего пишем легковесный запрос в кольцо, указывая лишь индекс в Grant table. После мы оповещаем бэкенд через Event channel. Когда бэкенд обработает запрос и запишет ответ со статусом, он отправит оповещение, что можно прочитать ответ и очистить память.

4.2. Механизм XenStore

XenStore — это иерархическое пространство имен для хранения информации, разделяемое между доменами. XenStore расположен в dom0 и поддерживает транзакции и атомарные операции. Примитивы междоменной связи, предоставляемые Xen, являются очень низкоуровневыми (виртуальные прерывания и разделяемая память). XenStore реализован поверх этих примитивов и обеспечивает некоторые операции более высокого уровня (чтение ключа, запись ключа, перечисление каталога, уведомление, когда ключ изменяет значение). Он предназначен для конфигурирования и хранения информации о состоянии, но не для передачи больших данных.

Каждый домен получает свой собственный путь в XenStore. Когда значения изменяются в XenStore, соответствующие драйверы уведомляются. Обычно XenStore используется для хранения информации о доменах во время их выполнения и в качестве механизма создания и управления устройствами domU.

Именно через XenStore идёт первоначальная конфигурация драйвера сети и его синхронизация с бэкендом. Сначала необходимо узнать префикс адреса бэкенда в XenStore, закрепленного за фронтендом, чтобы знать, откуда брать конфигурационную информацию. Например, бэкенд выделяет mac-адрес по умолчанию для фронтенда драйвера. Далее необходимо записать индексы в таблице Grant table, соответствующие кольцам получения и отправки сообщений, в соответствующие ветки дерева XenStore, установить номера прерываний для Event channel.

Гипервизор Xen поддерживает разделенный канал оповещения частей драйвера. В XenStore выставляется соответствующий флаг, а также выставляются два номера прерываний, а не один, как в объединённом канале для отправки и для принятия пакетов соответственно. Это позволяет не перегружать канал и не вызывать лишний раз функции, которые не требуется вызывать в данный момент, что оптимизирует производительность.

В XenStore выставляется также другая конфигурационная информация, необходимая для синхронизированной работы с бэкэндом, например, флаг “feature-rx-copy”, который указывает, что можно провести более быструю операцию копирования страницы с данными с помощью гипервизора вместо перемещения ее из домена в домен.

Также именно в XenStore выставляются состояния каждой из частей драйвера для определения конечным автоматом другой стороны драйвера. Во время инициализации гостевого домена бэкенд драйвер на стороне Linux производит начальную инициализацию и переходит из состояния “Initialising” в состояние “InitWait”, что означает, что он ожидает инициализации фронтэнда драйвера на стороне гостевого домена, а именно: инициализации колец и канала сообщений, а также выставления соответствующей информации в XenStore. После инициализации фронтенд меняет свое состояние на “Connected” и дожидается такого же состояния у бэкенда. После того, как оба состояния окажутся “Connected”, части драйвера считаются настроенными и готовыми к работе.

В процессе изучения существующих решений было выявлено, что в Linux для изменения и отслеживания состояний частей драйвера используется реализация интерфейса XenBus. XenBus — это API-интерфейс ядра, используемый драйверами и приложениями для взаимодействия с XenStore. В XenBus используется механизм транзакций, чтобы несколько операций обновления данных в XenStore воспринимались как одна атомарная операция. В Linux это используется для изоляции кода, специфичного для Xen, за относительно абстрактным интерфейсом.

Однако XenBus является опциональным при портировании ядра на Xen. Детально изучив процесс настройки, было решено, что XenBus является избыточным механизмом, потому что данные в XenStore обновляются поэтапно и по очереди с использованием конечного автомата состояний частей драйвера, т.е. конкуренция за разделяемый объект отсутствует. Даже при одновременной

инициализации нескольких гостевых доменов конкуренция за элемент в XenStore отсутствует, потому что каждому из гостевых доменов выдаётся свой собственный адрес в иерархии XenStore для хранения конфигурационной информации. Более того, реализация такого примитивного конечного автомата для синхронизации фронтенд и бэкенд сторон драйвера не требует такой избыточной логики, поэтому было принято решение реализовать взаимодействие Embox с XenStore напрямую, а логику синхронизации заложить непосредственно в процесс инициализации драйвера.

4.3. Интеграция

Обязательным условием запуска гостевого домена в окружении гипервизора Xen является наличие сконфигурированного на хост-домене виртуального коммутатора (bridge [22]). Он позволяет домену иметь свою локальную сеть, связанную с другими доменами, работающими на гипервизоре Xen. Также можно настроить NAT, чтобы у гостевого домена была возможность выходить в Глобальную сеть Интернет. Bridge в Xen устанавливается как интерфейс, а все гостевые домены к нему подключаются. Мной были написаны скрипты для настройки интерфейса коммутатора и установки локальной сети гостевого домена с использованием встроенных программ ip [25] и bridge. Также мной были написаны скрипты для настройки сети на стороне Embox. С помощью программ ifconfig [24] и route [27] были настроены интерфейсы lo (Loopback) и eth0 для общения с внешним миром. Были настроены ip и mac адреса для корректной работы внутри сети, выделенной коммутатором.

В архитектуре Embox существует интерфейс для драйверов сети, а также определённая последовательность действий, необходимых для успешной регистрации устройства в ядре. Выделяется управляющая структура, которая хранит в себе адреса функций сетевого драйвера. Она регистрируется в системе вместе с сущностью сетевого девайса. Именно там хранится приватная архитектурно зависимая структура

данных.

В процессе разработки были испытаны разные конфигурации сетевого интерфейса для большей производительности. Было выявлено, что большую скорость обмена данными обеспечивает разделённый канал сообщений: отдельный канал для оповещения о новых данных в кольце приёма сообщений и отдельный канал для оповещения об изменениях в кольце передачи. В процессе интеграции я зарегистрировал оба номера прерывания, в которые преобразуются события из Event channel, сначала в гипервизоре при помощи гиперколов, а потом в Embox при помощи внутреннего интерфейса. Момент появления прерываний не определён, поэтому используются механизмы синхронизации, которые уже были реализованы в Embox. Для обеспечения монопольного доступа к критическим секциям кода используются функции блокировки переключения потока исполнения, что обеспечивает атомарность операций получения идентификационных номеров запросов, доступных для записи отправляемых данных, а также для исключения повторного вызова функции обработки кольца приёма сообщений и функции очистки кольца передачи сообщений.

После обработки кольца приёма сообщений пакеты передаются в сетевой стек Embox через установленный интерфейс взаимодействия. При этом данные оборачиваются в понятную Embox'у структуру sk_buff, представляющую буфер сетевого пакета. Обратный процесс происходит при отправке сообщений. Из сетевого стека драйвер получает структуру sk_buff, извлекает из неё данные, записывает пакет на страницу общей памяти, пишет запрос с номером страницы в Grant table в кольцо и оповещает бэкенд.

Драйвер в контексте архитектуры Embox также был представлен в виде загружаемого отдельного модуля. Были прописаны необходимые зависимости: память, утилиты ip и route, механизм Grant table, а также необходимые для функционирования сети модули, реализующие стек TCP/IP.

5. Оценка сетевой производительности Embox

5.1. Обзор метрик

Задача апробации драйвера сети в Embox подразумевает под собой непрерывное тестирование новой функциональности на отсутствие ошибок, сравнение полученных результатов по выбранным метрикам с эталонными значениями другой реализации, а также выявление узких мест и их последующая оптимизация. Также необходимо получить результаты производительности как всей unikernel системы, так и отдельно добавленной функциональности. Для оценки сетевой производительности были проанализированы существующие метрики для тестирования серверов. Из них были выбраны основные, удовлетворяющие контексту данной работы.

В первую очередь необходимо выяснить задержку или время отклика — время между запросом и ответом. Необходимо оценить скорость отклика веб-сервиса, работающего в окружении Embox, а именно: сколько времени проходит от отправки запроса до получения первых данных. Данные значения учитывают производительность сетевого драйвера, а также встроенного в Embox стека TCP/IP.

Вторым важным аспектом является количество одновременных подключений. Количество активных соединений напрямую влияет на производительность системы: диспетчеризация для каждого пакета транспортного уровня может повлечь замедление с увеличением возможных назначений. Простаивающие соединения потребляют память, которая могла бы быть использована для кэша, что также может приводить к замедлению. Необходимо эмпирическим путём выявить пороговое значение количества подключений, которое предполагает оптимум по производительности системы.

Финальным параметром будет оцениваться пропускная способность — максимальное количество бит в секунду, которое можно передать по сети. Классическим методом тестирования пропускной способности

является передача значительного по размеру файла с одной системы на другую и последующее измерение времени копирования файла. Значение пропускной способности получается, разделив размер файла на время передачи. Данный подход позволяет оценить не только производительность конкретного драйвера сети, но и производительность системы в целом, потому что он учитывает такие накладные расходы, как задержка, размер окна приема TCP и другие ограничения системы. Это позволит оценить параметр достижимой производительности.

5.2. Процесс тестирования

Организация стенда для тестирования включила в себя настройку окружения, а также установку пакета специализированных программ тестирования сетевой производительности.

Для проверки целостности и качества соединений была использована утилита ping [26]. Тестирование проводилось 5 раз. Непрерывная отправка ICMP пакетов длилась в течение 60 минут в каждом тесте с частотой 1 пакет в секунду. В результате было получено количество успешно отвеченных пакетов, потерянных пакетов и пакетов с ошибкой, а также среднее время обработки пакета, включая минимальное и максимальное время отклика.

Для определения стабильности времени подключения в утилите ping был использован флаг, отвечающий за непрерывную отправку ICMP пакетов. Утилита ping с данным флагом запускалась 10 раз на 45 минут. После этого работоспособность сервера проверялась контрольным запросом к корневой странице при помощи утилиты wget [3], а также повторным тестированием утилитой ping с флагом, отвечающим за непрерывную отправку пакетов, или без него в течение 15 минут.

Для тестирования пропускной способности, чтобы исключить оценку посторонних накладных расходов, таких как различная реализация серверного приложения, было решено использовать сервер

httpd [6] для всех гостевых систем. Это серверное программное обеспечение для работы с протоколом HTTP в режиме демона. Демон — это компьютерная программа в системах класса UNIX, запускаемая самой системой и работающая в фоновом режиме без прямого взаимодействия с пользователем. Данная программа уже была портирована в Embox в качестве третьестороннего программного обеспечения с минимальными изменениями. Для нагрузочного тестирования в качестве клиента использовалась утилита httperf [5]. Ключевым её отличием является то, что httperf пытается отправлять непрерывный поток запросов с заданной скоростью независимо от того, отвечают на них или нет. Это позволит показать не только максимальную нагрузку веб-сервера, но, что более важно, его поведение при перегрузке. Данное свойство позволяет замерять реальное время ожидания ответа. В тестировании было использовано порядка 8 различных конфигураций по 10 запусков тестов в каждой. Количество запросов варьировалось от 1 до 100 с шагом 20. Количество одновременных сессий от 1 до 40 с шагом в 10. Также тестирование включало в себя варьирование размера файла от 100КБ до 100МБ.

Для проведения полного анализа была выбрана ещё одна утилита для стрессового тестирования — seige [15]. Данная утилита позволяет проверить сервер в условиях, максимально приближенных к реальным. Отличительной чертой программы Siege является то, что она позволяет имитировать обращения к сайту сразу нескольких пользователей. Утилита имитирует пользователя путем недетерминированных по времени обращений к серверу и переходу по ссылкам с уже полученной страницы. Более того, конфигурация позволяет задать количество одновременных подключений к серверу, что позволит выявить пороговое значение производительности веб-сервиса. Было произведено 10 запусков Seige по 600 секунд каждый. Имитировалось поведение 1, 10, 20 и 40 пользователей. Размер получаемой страницы также варьировался от 100КБ до 100МБ.

Эталонные значения для сравнения были получены с использованием минималистичного дистрибутива Alpine Linux,

нацеленного на безопасность и сетевую производительность. Поддержка POSIX позволяет запустить серверное приложение для тестирования на стороне domU, а наследование реализаций большинства механизмов Linux гарантирует высокую сетевую производительность.

5.3. Оптимизация сетевого драйвера

В процессе изучения архитектуры Xen и в частности реализации модели разделенного драйвера в Linux были выявлены возможные оптимизации для улучшения значений показателей по выбранным для тестирования метрикам.

Изначальный вариант реализации включал в себя объединённый канал оповещения частей драйверов о появлении новых данных. При появлении оповещения от бэкенда драйвера вызывалось сразу несколько процессов, блокирующих поток исполнения во избежание гонок: сначала обработка кольца приёма, затем процесс очистки кольца передачи. На стороне бэкенда также на одно оповещение от фронтенда последовательно пробуждались сразу несколько процессов, что влекло за собой обработку колец, которые в этой обработке не нуждались. Гипервизор Xen позволяет использовать разделённый канал обмена событиями, чтобы оповещать только о событиях в конкретном кольце. Для этого я инициировал два канала обмена событиями и создал два различных обработчика прерываний: для кольца приёма сообщений и для кольца отправки.

Гипервизор Xen предоставляет хосту возможность использовать преимущества архитектуры для оптимизации процесса взаимодействия с частями драйверов гостевых доменов. Так, во время конфигурации фронтенда драйвера я указал специальный флаг feature-rx-copy в XenStore, указывающий, что бэкенду следует копировать переданные фронтендом драйвера страницы при помощи внутренних механизмов гипервизора Xen вместо того, чтобы использовать передачу страницы от домена к домену. В последнем случае гипервизор действительно

копирует страницу из одного места в другое вместо дешёвой смены указателей и прав доступа.

В первоначальной версии драйвера при появлении прерывания для обработки колец драйвер сети блокировал исполнение до окончания обработки, игнорируя возможность переключения контекста для обработки другого кольца. Это сильно замедляло процесс, потому что драйверу сети необходимо одновременно обрабатывать как поток отправки, так и поток получения пакетов. В противном же случае, когда одна очередь приоритетнее другой, возможно существенное увеличение времени отклика системы. Решение данной проблемы оказалось в более аккуратном использовании механизма передачи событий. Гипервизор Xen позволяет отправлять оповещение только в том случае, когда очередь сообщений пуста. Эта функция позволяет сократить количество отправляемых прерываний. Из этого последовала возможность ограничения критических секций в обработке колец до нескольких инструкций, что в свою очередь позволило системе гибче распределять нагрузку двух потоков обработки колец между собой.

Ещё одна оптимизация появилась из концепции, что пакет передаётся между доменами при помощи механизма общей памяти, поэтому повреждение данных при передаче могут говорить о проблемах более серьезных, чем потеря пакета. Вероятность повреждения данных в стабильно работающей системе близка к нулю, поэтому я отключил проверку контрольной суммы пакета на стороне Embox.

5.4. Сравнительный анализ результатов

Стоит отметить, что результаты были получены в искусственной среде с использованием Vagrant и VirtualBox, поэтому в первую очередь их стоит рассматривать с точки зрения сравнения, а не с точки зрения реальной производительности, которая могла бы быть больше на репрезентативной машине.

Я провёл тесты и получил результаты, представленные в таблице 1. Среднее значение времени отклика, полученное при помощи утилиты

	Embox до оптимизации	Embox после оптимизации	Alpine
min	3.1	0.4	0.2
max	25.6	9.4	5.4
avg	12.2	5.2	1.5

Таблица 1: Сравнение времени отклика (мс)

ping, в Alpine Linux составляет 1.5мс, минимальное значение составляет 0.2, а максимальное — 5.4мс. В Embox среднее значение времени отклика составляло 12.2мс до оптимизации с минимальным значением 3.1мс, а максимальным 25.6мс. Однако после оптимизации показатель среднего значения уменьшился почти в 5 раз и теперь составляет 5.2мс. Время отклика сократилась и стало более стабильным: минимальное значение составило 0.4мс, а максимальное 9.4мс.

Стоит отметить, что за каждое нагрузочное тестирование с использованием утилиты ping с флагом, отвечающим за непрерывную отправку пакетов, приходило более 100000 пакетов, и ни один из них не был пропущен или не отвечен. Более того, Embox продолжал функционировать в штатном режиме.

Данные о пропускной способности были получены при помощи утилиты httperf в связке с сервером httpd. У Alpine Linux пропускная способность в среднем составила 29.3мб/с. В Embox до оптимизации пропускная способность составляла 2.78мб/с, но после оптимизации увеличилась почти в три с половиной раза — в среднем 9.47мб/с. Данное значение является максимальным. При увеличении размера файла пропускная способность не снижается: максимальное отклонение значения пропускной способности не превышает 0.2Мб/с. При увеличении количества запросов производительность не снижается.

Результаты, полученные при помощи утилиты Seige, незначительно отличались от результатов httperf из-за одинаковых значений конфигурации. Расхождение в значениях было меньше 8%, это связано с недетерминированным значением времени отправки запросов в Seige.

Программы httperf и seige позволили получить наглядную статистику по подключениям, а также позволили за несколько

итераций определить оптимальную конфигурацию сервера. Подбор порогового значения количества одновременных подключений оказался зависимым в большей степени от реализации сервера httpd, а не от реализации драйвера, поэтому пороговые значения в Embox и в Alpine Linux совпали.

Тестирование позволило выявить и оптимизировать не только узкие места в самом драйвере, но и ряд проблем, находящихся в архитектурно-зависимой части Embox, например, ошибка в таймере, которая также была исправлена. Тестирование показало, что Embox в качестве unikernel системы может работать как с небольшими по размеру запросами, так и с большими файлами без уменьшения скорости передачи. Нагрузочное тестирование с имитацией реального поведения пользователей показали небольшое отклонение значений скорости относительно значительного увеличения числа последовательных сессий и запросов, что говорит о стабильной производительности драйвера.

6. Заключение

В процессе данной работы был реализован и протестирован прототип unikernel системы на основе ОСРВ Embox для архитектуры гипервизора Xen. Были достигнуты следующие основные результаты.

- Произведен обзор предметной области, изучена архитектура гипервизора Xen и ОСРВ Embox, изучены существующие unikernel системы, минимальная реализация Mini-OS и полная реализация Linux выбраны в качестве опорных реализаций модели разделенного драйвера, а POSIX-совместимый дистрибутив Alpine Linux выбран для сравнения производительности

- Реализован и интегрирован в систему сборки Embox механизм общей памяти для архитектуры гипервизора Xen, включающий в себя реализацию механизма Grant table и реализацию механизма трансляции адресов

- Реализован и интегрирован сетевой интерфейс Embox: протокол обмена сообщениями между доменами с помощью колец, произведена настройка фронтенда драйвера при помощи XenStore без использования XenBus, произведена интеграция во внутреннюю систему сборки Mybuild, драйвер подключен к стеку TCP/IP

- Произведена оценка сетевой производительности Embox в качестве unikernel системы, достигнуты стабильные результаты при нагрузочном тестировании, произведена оптимизация драйвера

Список литературы

[1] Alpine Linux. — 2020. — Access mode: https://alpinelinux.org (online; accessed: 01.06.2020).

[2] Chisnall David. The Definitive Guide to the Xen Hypervisor. — 2007. — Access mode: https://www.amazon.com/Definitive-Hypervisor-Pearson-Software-Development/dp/0133582493.

[3] GNU Wget 1.20. — 2019. — Access mode: https://www.gnu.org/software/wget/manual/wget.html (online; accessed: 01.06.2020).

[4] HaLVM. — 2018. — Access mode: https://galois.com/project/halvm/ (online; accessed: 01.06.2020).

[5] Httperf. — 2020. — Access mode: https://github.com/httperf/httperf (online; accessed: 01.06.2020).

[6] Hyper Text Transfer Protocol Daemon. — 2020. — Access mode: https://github.com/embox/embox/tree/master/src/cmds/net/httpd (online; accessed: 01.06.2020).

[7] Ling. — 2015. — Access mode: https://github.com/cloudozer/ling (online; accessed: 01.06.2020).

[8] Linux backend driver realization. — 2020. — Access mode: https://github.com/torvalds/linux/tree/master/drivers/net/xen-netback (online; accessed: 01.06.2020).

[9] Linux netfront driver realization. — 2019. — Access mode: https://github.com/torvalds/linux/blob/master/drivers/net/xen-netfront.c (online; accessed: 01.06.2020).

[10] Mini-OS. — 2018. — Access mode: https://wiki.xenproject.org/wiki/Mini-OS (online; accessed: 01.06.2020).

[11] MirageOS. — 2020. — Access mode: https://mirage.io/ (online; accessed: 01.06.2020).

[12] Oracle VM Virtualbox. — 2020. — Access mode: https://www.virtualbox.org (online; accessed: 01.06.2020).

[13] Rump kernels. — 2019. — Access mode: http://rumpkernel.org (online; accessed: 01.06.2020).

[14] Rumprun. — 2020. — Access mode: https://github.com/rumpkernel/rumprun (online; accessed: 01.06.2020).

[15] Seige. — 2020. — Access mode: https://linux.die.net/man/1/siege (online; accessed: 01.06.2020).

[16] Ubuntu 18.04 LTS «Bionic Beaver. — 2018. — Access mode: https://releases.ubuntu.com/18.04.4/ (online; accessed: 01.06.2020).

[17] Unikernels. — 2020. — Access mode: http://unikernel.org (online; accessed: 01.06.2020).

[18] Vagrant. — 2020. — Access mode: https://www.vagrantup.com/ (online; accessed: 01.06.2020).

[19] Visual Studio Code. — 2020. — Access mode: https://code.visualstudio.com (online; accessed: 01.06.2020).

[20] Xen project. — 2019. — Access mode: https://wiki.xenproject.org/wiki/Main_Page (online; accessed: 01.06.2020).

[21] arping. — 2019. — Access mode: https://www.man7.org/linux/man-pages/man8/arping.8.html (online; accessed: 01.06.2020).

[22] bridge. — 2002. — Access mode: https://man7.org/linux/man-pages/man8/bridge.8.html (online; accessed: 01.06.2020).

[23] gdb. — 2019. — Access mode: https://www.gnu.org/software/gdb/ (online; accessed: 01.06.2020).

[24] ifconfig. — 2020. — Access mode: https://github.com/embox/embox/tree/master/src/cmds/net/ifconfig (online; accessed: 01.06.2020).

[25] ip. — 2001. — Access mode: https://man7.org/linux/man-pages/man8/ip.8.html (online; accessed: 01.06.2020).

[26] ping. — 1986. — Access mode: https://linux.die.net/man/8/ping (online; accessed: 01.06.2020).

[27] route. — 2020. — Access mode: https://github.com/embox/embox/tree/master/src/cmds/net/route (online; accessed: 01.06.2020).

[28] tcpdump. — 2020. — Access mode: https://www.tcpdump.org (online; accessed: 01.06.2020).

[29] xen-system-amd64. — 2020. — Access mode: https://packages.debian.org/ru/sid/xen-system-amd64 (online; accessed: 01.06.2020).

[30] Документация к встроенной в Embox системе сборки Mybuild. — 2020. — Access mode: https://non-descriptive.github.io/embox-mdbook/ru/embox_user_manual_ru.html (online; accessed: 01.06.2020).

[31] Операционная система реального времени Embox. — 2020. — Access mode: https://www.embox.rocks (online; accessed: 01.06.2020).