A. U. Budkina ,V.A. Sartakov

Fragmented network subsystem with traffic filtering for microkernel environment

Key words: Microkernel environment, network subsystem, traffic filtering

The TCP/IP stack in a microkernel operating system executed in a user space, which requires the development of a distributed network infrastructure within a single software environment. Its functions are the organization of interaction between the components of the stack with different processes, as well as the organization of filtering mechanisms and routing of internal network traffic. Use of architectural approaches applicable in monolithic-modular systems is impossible, because the network stack is not a shareable component of the system. As a consequence, the microkernel environment requires development of special network subsystem. In this work we provide overview of major conceptions of network architectures in microkernel environments. Also, we provide own architecture which supports filtering of internal network traffic. We evaluate the architecture by development of high-performance "key-value" store.

А. Ю. Будкина, В. А. Сартаков

ФРАГМЕНТИРОВАННАЯ СЕТЕВАЯ ПОДСИСТЕМА МИКРОЯДЕРНОГО ОКРУЖЕНИЯ С ПОДДЕРЖКОЙ ФИЛЬТРАЦИИ ТРАФИКА

Введение

Концепция разделения механизмов и политик (separation of mechanism and policy) [1], подразумевающая изоляцию кода, принимающего решения (policy), от механизмов, реализующих процесс принятия решения (mechanism), является основополагающей в архитектуре микроядерных операционных систем (ОС). Политики с точки зрения ОС — это драйвера, файловые системы, стеки протоколов, пользовательские программы и т.д. В микроядерных ОС эти компоненты системы исполняются в пользовательском пространстве и изолированы от механизмов. Механизмы в микроядерных системах — компоненты ядра, предоставляющие примитивы по управлению системными ресурсами: созданием процессов, выделением памяти, отображением памяти, запросом прерываний и т. п. Эти примитивы требуют привилегированного доступа, они не могут быть исполнены в пространстве пользователя, в отличие от политик, и находятся в ядре.

Вынесение из привилегированного пространства таких компонент ядра, как TCP/IP- стек, порождает новые архитектурные решения. Если в монолитно-модульных системах TCP/IP стек находится в пространстве ядра, он один и он обрабатывает все пакеты системы, то в микроядерных системах TCP/IP стеков может быть множество, они могут по-разному взаимодействовать с другими компонентами системы (драйверами, программами), создавая, по факту, внутреннюю сеть.

Такая внутренняя сеть требует дополнительного контроля, поскольку используемые в микроядерных окружениях системы безопасности на основе мандатных ссылок (capability-based security) не ограничивают движения сетевых пакетов . Например, имея два процесса, в каждом из которых присутствует сетевой стек, мы можем запретить их прямое взаимодействие, используя систему безопасности. В тоже время, эти компоненты могут взаимодействовать посредством виртуальной сети, ведь оба процесса должны использовать виртуальный сетевой коммутатор или физическое сетевое устройство. Также вне зависимости от того, каким образом организован сетевой стек (является ли он общим для всех процессов, или процессы организуются в сеть множеством стеков), при использовании технологий виртуализации необходимо создавать виртуальную сетевую инфраструктуру, компоненты которой необходимо изолировать, а доступ разграничивать.

Помимо внутренней сети между компонентами системы, вынесение стека из привилегированного режима приводит к фрагментации самого стека, то есть расщеплению стека на составные части с уменьшенной связностью. Иными словами разные фрагменты стека, во-первых, могут быть реплицированы, что может способствовать повышению отказоустойчивости системы, а во-вторых, набор функций копии стека может
быть существенно уменьшен до минимально необходимого программе.

В рамках работы рассматриваются проблемы организации внутренней архитектуры сетевой подсистемы микроядерного окружения, приведен анализ различных подходов и возможных архитектур, представлена архитектура фрагментированного стека и соответствующей виртуальной сетевой инфраструктуры. Для тестирования производительности разработанной сетевой инфраструктуры был создан прототип высокопроизводительного хранилища «ключ-значение» (key-value store), защищенный от вторжений посредством встроенного межсетевого экрана. Дублирование сетевых стеков позволило существенно повысить производительность хранилища (24% в сравнении с GNU/Linux), а фрагментация дублированных стеков позволило уменьшить размер используемого кода каждым стеком на 33%.

Сетевая инфраструктура микроядерного окружения

В монолитно-модульных операционных системах, таких как Windows, Linux, FreeBSD, стек протоколов TCP/IP выполняется в пространстве ядра (рис. 1). Все приложения разделяют общий сетевой стек, используя программный интерфейс сокетов для обращения к нему. В ядре выполняется множество функций: коммутация пакетов между физическими интерфейсами, маршрутизация, инкапсуляция и деинкапсуляция пакетов для передачи в приложение и из него. Элементы инфраструктуры сетевой безопасности, например, межсетевой экран Netfilter, являются частью сетевого стека и также исполняются в режиме ядра.

В микроядерных ОС стеки протоколов, сетевое ПО, прикладное ПО, драйвера устройств — все находится в пространстве пользователя. Разделение компонент на отдельные программы позволяет комбинировать между собой разные компоненты сетевой подсистемы, выстраивая внутреннюю коммуникацию в соответствии с политиками безопасности. Как следствие, открывается возможность разделения TCP/IP стека на отдельные фрагменты, а так же отделение от него, если это необходимо, неиспользуемых элементов.

Важной проблемой при разработке метода фрагментации сетевой подсистемы является производительность. Разделение подсистемы производится посредством переноса компонент в отдельные адресные пространства, взаимодействие между которыми осуществляется при помощи сообщений. Каждое такое сообщение затрагивает ядро, требует переключения контекста, чего не происходит в монолитно-модульных системах. Это значит, что архитектура фрагментированной сетевой подсистемы должна содержать минимальное количество переключений контекста, но в то же время соответствовать принципу изоляции максимального числа компонентов. Количество компонентов сетевой подсистемы, выполняющихся в отдельных процессах может быть различным, однако в сетевой подсистеме всегда присутствует минимальный набор компонент, сочетание и взаимодействие которых можно определить в архитектуре. К этим компонентам относятся:
сетевые драйвера, механизм маршрутизации, пакетный фильтр, реализации протоколов, высокоуровневые сетевые интерфейсы (сокеты, порты). Рассмотрим несколько вариантов компоновки сетевой подсистемы.

Монолитный стек в пользовательском пространстве

Одним из подходов при построении сетевой подсистемы микроядерной ОС может быть идея единого стека с интегрированным межсетевым экраном, драйверами устройств,

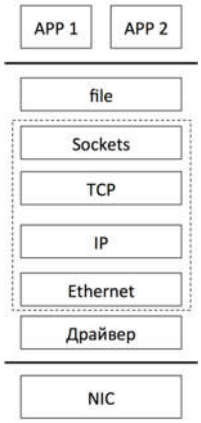

Рис. 1. Сетевая подсистема Linux

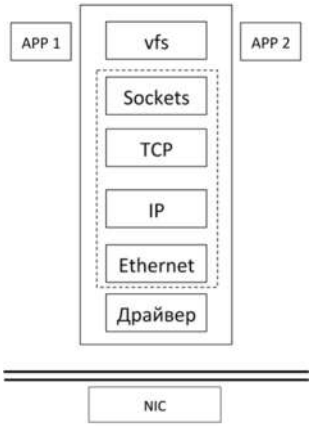

Рис. 2. Монолитный стек в пространстве пользователя

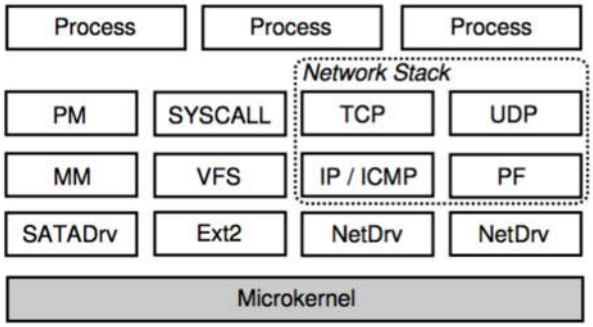

Рис. 3. Сетевая подсистема NewtOS

протоколами и др., но, в отличие от монолитно-модульных систем, исполняемого в пользовательском пространстве (рис. 2). Прикладные программы могут совместно использовать этот стек, эмулирующий POSIX socket API. На низком уровне процесс TCP/IP стека через встроенные драйвера работает напрямую с сетевыми устройствами.

С точки зрения производительности такая компоновка обладает рядом преимуществ и недостатков. Так как стек и драйвера находятся в одном адресном пространстве, то передача данных из драйвера в стек осуществляется без дополнительных переключения контекста и пересылки сообщений. Физические адреса сетевых устройств отображаются в виртуальное адресное пространство драйвера, а информация о прерываниях доставляется при помощи сообщений. Далее драйвер извлекает пакет, который затем обрабатывается в стеке. На верхнем уровне обработанный пакет доставляется через разделяемую память и сообщения соответствующим программам. В тоже время производительность такого стека деградирует, если стек и программы исполняются на разных ядрах процессора. Этот вывод был сделан в рамках работы [2] по анализу производительности сетевой подсистемы микроядерного окружения Genode [3]. Причиной деградации производительности являются издержки, возникающие при синхронизации кэшей ядер процессора при одновременной работе с разделяемыми регионами памяти.

Множество стеков

В микроядерных окружениях семейства L4, таких как L4Re и Genode, используется концепция множества сетевых стеков, каждый из которых встроен в сетевое приложение. На низком уровне стеки могут соединяться с драйвером устройства или с виртуальным коммутатором. Такой подход обладает как положительными, так и отрицательными качествами. С одной стороны, безусловно, внутренняя среда становится набором сетевых приложений, коммуницирующих между собой. Использование изолированных драйверов позволяет обеспечивать отказоустойчивость, а встроенный в приложение стек уменьшает количество переключений контекста. В тоже время, при такой конфигурации невозможно использовать средства фильтрации трафика. Контроль трафика является «policy» и должен обеспечиваться программами, входящими в TCB (Trusted Computing Base). Кроме того, если предположить, что у пользовательской программы есть доступ к механизму формирования правил фильтрации, то эти правила могут быть деактивированы пользовательской программой посредством сбоя или вторжения. Таким образом, использование TCP/IP стека в качестве компонента сетевой программы невозможно при необходимости использования инструментов фильтрации трафика.

Фрагментированный стек операционной системы NewtOS

Проблема деградации производительности микроядерных систем на SMP-архитектурах рассматривалась в различных исследовательских проектах. Например, в рамках проекта Barellfish [4] была предложена мультиядерная (multikernel) архитектура, основная идея

которой заключается отказе от межядерной коммуникации, отказу от разделяемой памяти в пользу репликации, а вместо IPC должны использоваться только сообщения. Как следствие, в SMP-системах на каждом ядре процессора исполняется отдельное 4ядро, что и привело к появлению термина «мультиядерный».

Данный подход был использован при разработке сетевой подсистемы в проекте NewtOS [5] для построения отказоустойчивой сетевой подсистемы микроядерной ОС. В ней сетевая подсистема разделяется на несколько однопоточных компонентов, сообщающихся между собой с помощью асинхронного межпроцессного взаимодействия. Протоколы TCP, UDP, IP, а также механизм маршрутизации пакетов, межсетевой экран и сетевой драйвер выполняются в разных пользовательских процессах (рис. 3). Проблема большого количества переключений контекста была решена с помощью организации механизма передачи сообщений между компонентами, в котором ядро не принимает участие. Также производительность была повышена за счет параллельного выполнения каждого из компонентов сетевой подсистемы на отдельном ядре процессора. Максимальная производительность от казоустойчивой сетевой подсистемы достигала 70% от производительности стека ядра Linux.

Фрагментированный стек микроядерного окружения

При разработке модели сетевой подсистемы микроядерного окружения мы отталкивались от следующих базовых идей:

• В системе присутствует множество стеков, каждый из которых взаимодействует только с одной программой

• Драйвера устройств отделены от стеков

• Компоненты стеков и программ группируются по соответствующим ядрам.

Использование множества стеков взаимодействующих с соответствующими программами таким образом, что стек является отдельным процессом, который, с одной стороны взаимодействует с программой, а с другой - с сетевым драйвером или виртуальным свитчем, превращает сетевую подсистему в виртуальную сеть программ (рис. 4). На высоком уровне коммуникация в виртуальной сети осуществляется через интерфейс сокетов, а на низком - при помощи виртуальной коммутации Ethernet-пакетов. Кроме того, в виртуальной сети могут присутствовать так же аппаратные и паравиртуализированные виртуальные машины: взаимодействие посредством Ethernet-пакетов позволяет объединять вместе любые компоненты микроядерной системы. Независимый стек позволяет осуществлять высокоуровневую фильтрацию трафика и делать это независимо от программы. Загрузка и обработка правил фильтрации трафика не является прямой задачей пользовательской программы, а следовательно у неё не должно быть к доступа к этим функциям стека.

Разделение драйверов и стеков осуществлено в соответствии с идеологией микроядерных систем с целью уменьшения влияние драйверов на работоспособность пользовательских программ. Драйвера зачастую являются точкой отказа, и их изоляция необходима. Кроме того, без отделения стека от драйвера невозможно использовать виртуальный свитч для внутренней коммутации пакетов.

Использование виртуального свитча может привести к возникновению издержек, связанных с совместным доступом к разделяемой памяти программ, исполняемых на разных ядрах. В тоже время, для тех процессов, которые не используют виртуальный свитч для коммутации и напрямую взаимодействуют с сетевым устройством, мы группируем компоненты стека, сетевой драйвер и программу таки образом, чтобы вся обработка сетевых пакетов происходила на одном процессорном ядре [6]. Современное сетевое оборудование позволяет осуществлять доставку сетевых пакетов драйверам, исполняемым на разных ядрах, что позволяет обеспечить движение пакетов по программному стеку строго в рамках одного процессорного ядра. Именно эту конфигурацию компонент мы используем далее при разработке высокопроизводительного хранилища типа «ключ-значение».

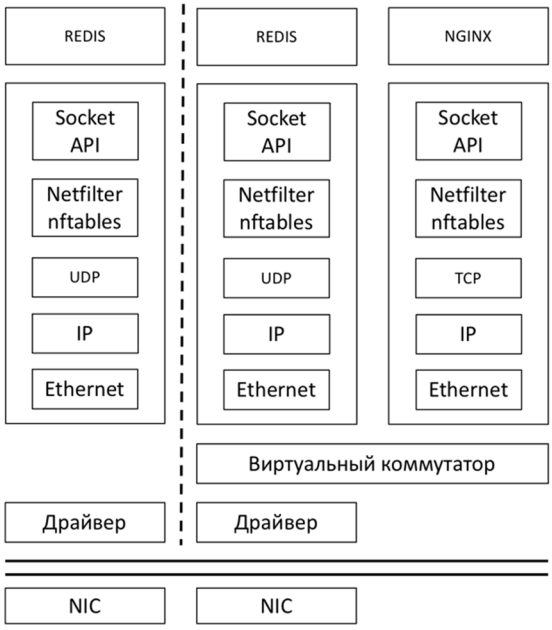

Рис. 4: Фрагментированный стек

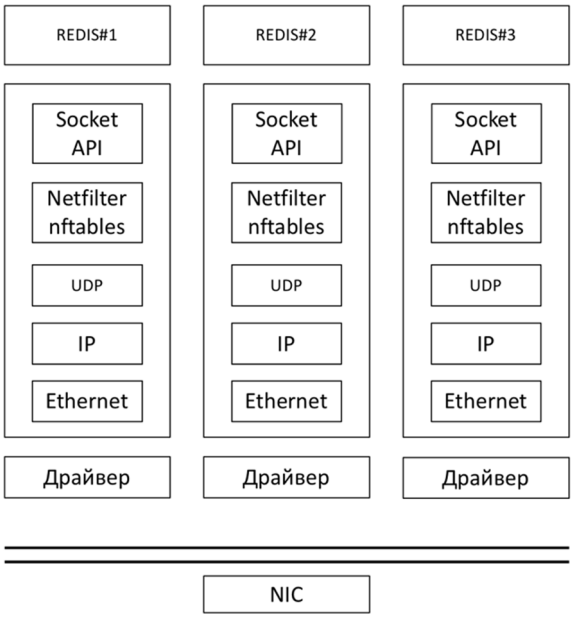

Рис. 5: Хранилище «ключ - значение»

Концепция фрагментированного стека отличается от мультиядерной идеологии тем, что коммуникации между ядрами все же присутствуют, а значит количество дублированных элементов меньше, что уменьшает сложность системы. В отличие от сетевой подсистемы проекта NewtOS, обработка пакетов разных сетевых протоколов выполняется в одном процессе, что способствует повышению производительности, но уменьшает отказоустойчивость.

Хранилище «ключ-значение»

Для апробации концепции фрагментированного стека с поддержкой фильтрации трафика мы разработали высокопроизводительное хранилище типа «ключ-значение» (Key-Value Store, KVS). В KVS входят несколько компонентов (рис. 5): Набор программ, реализующих функции хранилища, набор сетевых стеков с поддержкой функции фильтрации трафика, драйвера сетевого устройства, сетевая карта.

В качестве программы, реализующей функции хранилища, мы использовали Redis. Redis является открытым хранилищем. распространяемым под лицензией BSD. Мы осуществили перенос кода хранилища в микроядерную среду, заменив сетевые и системные компоненты Redis соответствующим функционалом микроядерного окружения. Мы используем четыре экземпляра хранилища, каждый из которых исполнялся на отдельном ядре.

У каждого из четырех экземпляров присутствует собственный сетевой стек, исполняемый в качестве отдельной программы, но на том же ядре, где и экземпляр хранилища. В этом эксперименте мы используем протокол UDP для взаимодействия с хранилищем, как следствие, мы извлекли из стека компоненты, не участвующие в передаче UDP-пакетов. В стек были встроены функции анализа трафика по средствам правил, которые являются частью стека, но могут динамически обновляются системной утилитой в
процессе работы.

Каждый сетевой стек связан с собственным экземпляром сетевого драйвера. Сетевой драйвер, как и стек, исполняется на одном ядре вместе с соответствующим экземпляром хранилища. При этом, в устройстве используется только одна сетевая карта. Такая схема работы возможна при использовании современных сетевых устройств, способных осуществлять доставку пакетов последовательно каждому обработчику на каждом ядре. Фактически, сетевая карта циклически доставляет пакеты разным драйверам и нет единого пространства, где аккумулируются все сетевые пакеты. Благодаря этому,

мы можем использовать четыре независимых хранилища, каждое из которых обладает собственным сетевым стеком и драйвером.

Реализация

В качестве основы прототипа KVS мы использовали микрогипервизор NOVA с микроядерным окружением Genode. NOVA является гипервизором первого порядка построенным в соответствии с идеологией микроядер, наследует L4 API и использует модель управления доступом на основе мандатных ссылок (capability-based security). Genode — фреймверк, поддерживающий множество различных микроядерных проектов, таких как Fiasco.OC, seL4, L4Ka и другие. Genode представляет собой прикладное программное обеспечение, требующее для своей работы микроядро.

В качестве сетевого устройства мы использовали Intel X540, для которого был разработан отдельный драйвер. Несмотря на наличие возможности использования слоя сопряжения DDE (Device-Driver-Environment), позволяющего переносить исходный код сетевых драйверов iPXE или Linux в окружение Genode, мы разработали собственный драйвер самостоятельно с целью минимизации его исходного кода.

Для сетевой подсистемы мы использовали библиотеку LXIP, являющейся исходным кодом сетевого стека ядра Linux, адаптированной для работы в среде Genode. Стек LXIP монолитен, в том смысле, что исходный код стека не позволял методами конфигурации отделить поддержку неиспользуемых протоколов. Такая функциональность была добавлена и, исключив поддержку неиспользуемых протоколов, мы уменьшили размер исходного кода LXIP на 33%. Дополнительно, в соответствии с концепцией, сетевой стек был изолирован от программы и драйвера, а так же был расширен поддержкой nftables.

Система nftables является частью ядра Linux начиная с версии 3.13. Наряду с системой iptables, система nftables используется для управления межсетевым экраном Netfilter. Она состоит из утилиты для задания правил nft и модулей ядра nftables. Утилита nft используется для преобразования текстовых правил фильтра в байткод и последующей передачи байткода в ядро по протоколу Netlink. Для поддержки Netlink в Linux используются библиотеки libmnl и libnftnl. Эти библиотеки были перенесены в программную среду Genode, а netfilter был интегрирован в сетевой стек LXIP. Каждый сетевой стек получил возможность использовать собственный набор правил, которые размещаются в виртуальной файловой системе и управляются конфигурационным файлом.

Тестирование

Тестирование разработанного прототипа хранилища «ключ-значение» проводилось при помощи утилиты redis-benchmark. Нам необходимо было выяснить, как изменилась производительность хранилища в сравнении со стандартным решением на основе ОС Linux, а так же влияют ли правила сетевого фильтра на производительность системы. Каждый экземпляр ключевого хранилища мог оперировать с 512 мегабайтами оперативной памяти, длина ключа и значения были стандартными. В качестве аппаратной платформы использовался Intel Core i5-4440 с четырьмя ядрами и отключенным параметром hyperthreading. Для сравнения производительности использовалась так же ОС Fedora 20.

При использовании только одного экземпляра хранилища, среднее количество запросов в секунду (RPS) для ОС Linux составляло 78034, в то время как для одного экземпляра хранилища среднее RPS для Genode составляло 96376. При использовании четырех экземпляров, при полной загрузке сетевого канала, мы получили средний RPS GNU/LInux 278691, против среднего RPS в Genode равного 385504. Как можно заметить, при использовании одного экземпляра хранилища прирост производительности микроядерного хранилища составляет 21%, а при использовании четырех экземпляров, прирост производительности микроядерного хранилища составляет 28%. Так же в ходе

экспериментов мы использовали различные правила netfilter, но деградации (стабильно более трех процентов) зафиксировать не удалось.

Заключение

В рамках данного проекта была поставлена задача разработки сетевой подсистемы микроядерного окружения. Для этого были проанализированы существующие подходы различных микроядерных проектов (таких как Genode, L4Re, NewtOS, Barellfish и других), были рассмотрены их сильные и слабые стороны в контексте различных применений.

Как было показано, в отличие от монолитно-модульных проектов, в микроядерных проектах существуют множества сетевых стеков, соединяющие компоненты системы в цифровую сеть. Различия подходов сводятся к области применений и целям проектов. Для некоторых проектов необходима высокая степень отказоустойчивости, и это достигается благодаря дублированию компонентов стека. В других проектах необходима гибкость конфигурирования и производительность без дополнительной защиты, и в этом случае стек является частью программы.

Нами был предложен альтернативный вариант архитектуры сетевой подсистемы, разработанный для создания высокопроизводительных и защищенных сетевых приложений. Мы используем с каждым процессом собственный стек. При этом в стеке, с одной стороны, отсутствует поддержка неиспользуемых протоколов, с другой - присутствуют инструменты фильтрации трафика, правила которых загружаются динамически в процессе работы компонентами, входящими в ТСВ. На основе разработанной концепции сетевой подсистемы было создано хранилище типа «ключ-значение», продемонстрировавшее рост производительности в сравнении с Linux на 21% при однопоточном исполнении и на 28% для исполнения в четыре потока. Кроме того, за счет удаления поддержки неиспользуемых протоколов, мы смогли уменьшить размер исходного текста стека на 33%.

СПИСОК ЛИТЕРАТУРЫ:

[1] Butler W Lampson and Howard E Sturgis. Reflections on an operating system design. Communications of the ACM, 19(5):251–265, 1976.

[2] Василий Сартаков and Александр Тарасиков. Анализ производительности сетевой подсистемы микроядерного окружения genode. In Материалы Международной научно-практической конференции Инструменты И Методы Аанализа Программ, pages 146–157, 2013.

[3] Genode Labs GmbH. Genode operating system framework.

[4] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. The multikernel: a new os architecture for scalable multicore systems. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pages 29–44. ACM, 2009.

[5] Tomas Hruby, Dirk Vogt, Herbert Bos, and Andrew S. Tanenbaum. Keep net working - on a dependable and fast networking stack. In In Dependable Systems and Networks, pages 1–12. IEEE, 2012.

[6] Василий Сартаков and Николай Голиков. Подходы к фрагментации системы паравиртуализации. pages 184–190, 2014.

REFERENCES:

[1] Butler W Lampson and Howard E Sturgis. Reflections on an operating system design. Communications of the ACM, 19(5):251–265, 1976.

[2] Vasily Sartakov and Aleksandr Tarasikov. Analiz proizvoditel'nosti setevoj podsistemy mikrojadernogo okruzhenija Genode. In Materialy Mezhdunarodnoj nauchno-prakticheskoj konferencii Instrumenty I Metody Aanaliza Programm, pages 146–157, 2013.
[3] Genode Labs GmbH. Genode operating system framework.
[4] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. The multikernel: a new os architecture for scalable multicore systems. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pages 29–44. ACM, 2009.
[5] Tomas Hruby, Dirk Vogt, Herbert Bos, and Andrew S. Tanenbaum. Keep net working - on a dependable and fast networking stack. In In Dependable Systems and Networks, pages 1–12. IEEE, 2012.
[6] Vasily Sartakov and Nikolaj Golikov. Podhody k fragmentacii sistemy paravirtualizacii. pages 184–190, 2014.