Подходы к фрагментации системы паравиртуализации

Василий Сартаков
ksys labs
sartakov@ksyslabs.org

Николай Голиков
ksys labs
golikov@ksyslabs.org

***Аннотация*—Повторное использование разработанного для Linux программного обеспечения в микроядерном окружении возможно благодаря использованию системы паравиртуализации L4Linux. Представляя собой модифицированное ядро Linux, L4Linux наследует его основной недостаток: исполнение в общем адресном пространстве (пусть и пользовательском) большого количества компонент. Преодолеть этот недостаток можно благодаря фрагментации, то есть разделению ядра L4Linux на независимые компоненты, исполняемые в разных адресных пространствах. В работе обосновывается необходимость такой фрагментации, описываются уже существующие и собственный подходы к декомпозиции и рекомпозиции системы, предварительные результаты работы.**

***Keywords*—Фрагментация; Паравиртуализация; Рекомпозиция системы;**

I. Введение

Изоляция компонентов - один из основополагающих принципов построения защищенных систем. Изоляция существенно понижает возможности компрометации всей системы при компрометации одного ее компонента.

Среди множества разнообразных архитектур Операционных Систем (ОС) с точки зрения изоляции компонентов ядра можно выделить две архитектуры: монолитно-модульную и микроядерную. К первой относятся популярные ОС, такие как GNU/Linux, BSD-системы, Windows, некоторые варианты коммерческих Unix. В этих системах основные компоненты: планировщик процессов, менеджер памяти, средства IPC, драйвера устройств, сетевой стек и другие - исполняются в едином адресном пространстве и с высоким уровнем привилегий. Компрометация любого из этих компонентов ведет к компрометации всего ядра, а за счет высокого уровня привилегий, на котором исполняется код ядра, и всей системы в целом. Ко второй можно отнести семейство экспериментальных проектов на основе ядра L4: ОС Singularity, MINIX3, ядро GNU/Mach и другие. Такие ОС содержат в своем ядре лишь минимальный набор компонентов, необходимых для разделения вычислительных ресурсов между пользовательскими процессами и обеспечения коммуникации между ними. Все остальные компоненты: драйверы устройств, сетевой стек и прочие - вынесены в пространство пользователя. Такой подход позволяет лучшим образом изолировать их друг от друга.

С точки зрения разработчиков конечных аппаратно-программных комплексов и устройств такие архитектурные свойства, как: защищенность, отказоустойчивость, производительность - не всегда являются решающими. Действительность такова, что скорость выхода устройства на рынок в области встраиваемых систем зависит в первую очередь от наличия уже готового ПО для нового устройства, в частности драйверов устройств и компонентов аппаратной платформы. Это достигается благодаря повторному использованию (code reuse) существующих наработок.

Прямой перенос существующего кода из монолитно-модульных систем в микроядерные невозможен без модификаций и специализированной поддержки. Прикладные программы в ОС GNU/Linux разрабатываются на основе POSIX-стандарта, в то время как, например, микроядро Fiasco.OC и его окружение L4Re (или Genode) с этим стандартом несовместимы. Тем не менее в ряде микроядерных проектов разработаны слои сопряжения (wrappers) для повторного использования кода из монолитно-модульных систем в окружении микроядерных.

Одним из таких слоев сопряжения для микроядра L4 является система паравиртуализации L4Linux [1]. Особенностью этого слоя сопряжения является то, что он позволяет исполнять двоичные программы ОС GNU/Linux в окружении ядра L4 без перекомпиляции. Достигается это при помощи переноса ядра Linux с аппаратной платформы на систему сообщений L4: вместо использования аппаратных вызовов, записей в регистры, управлением TLB и устройствами, ядро L4Linux использует вызовы L4. Как следствие, ядро L4Linux исполняется не в привилегированном режиме пространства ядра, а в качестве обычной пользовательской программы. При этом доступ к аппаратному обеспечению, например, при работе драйверов, обеспечивается при помощи отображения аппаратных страниц в адресное пространство L4Linux.

Перенос ядра Linux в пользовательское пространство позволяет повторно использовать драйвера из ядра Linux для работы в микроядерном окружении, а так же исполнять двоичные программы GNU/Linux. В то же время микроядро гарантирует изоляцию компонентов системы по отношению к системе паравиртуализации, при том что компоненты L4Linux по-прежнему находятся в едином адресном пространстве. Как следствие, компрометация любого компонента L4Linux приводит к компрометации всей системы паравиртуализации L4Linux. Кроме того, как показывает анализ архитектуры L4Linux [2], текущая реализация паравиртуализации не позволяет использовать аппаратные средства защиты стека, такие как NX-бит и рандомизацию адресного пространства (ASLR). NX-бит и ASLR применимы к отдельным программам, где в момент

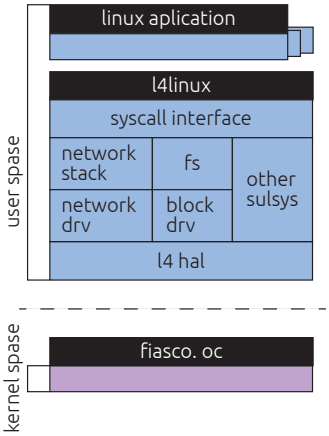

Рис. 1: L4Linux

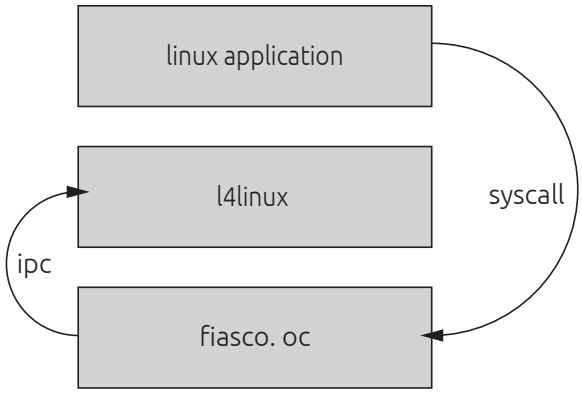

Рис. 2: Обработка системных вызовов в L4Linux

старта можно выделить, в каких регионах памяти будут находиться сегменты данных, кода и стек. В текущей реализации система паравиртуализации представляет собой монолитный регион виртуальной памяти (dataspace), внутри которого располагаются различные части L4Linux. Эти части управляются паравиртуализированным ядром, а значит, в момент выделения dataspace невозможно определить, какая часть этого пространства будет использована под данные, а какая - под код.

Решить проблему монолитного L4Linux можно применив к нему уже знакомую идеологию микроядра - разделив L4Linux на самостоятельные компоненты и связав их между собой посредствам IPC. Такое разделение мы называем термином *фрагментация*[1]. При фрагментации L4Linux каждый отделённый компонент становится самостоятельной частью окружения микроядра, т.е. программой, и, как следствие, к этой программе можно применить механизмы защиты стека. Кроме того, после отделения каждый компонент исполняется в собственном адресном пространстве, что существенно усложняет вторжение и компрометацию системы.

Другим положительным эффектом разделения L4Linux может являться повышение отказоустойчивости компонентов системы, а также производительность: несмотря на то, что при фрагментации возникают дополнительные накладные расходы, связанные с пересылкой сообщений, микроядерное окружение может быть перекомпановано с целью минимизации межпроцессорного взаимодействия. В частности, исследовательская мультиядерная (multikernel) система Barellfish [3] была разработана, как ответ на проблему деградации производительности в SMP-системах. В проектах seL4 [4] и NewtOS [5] так же рассматриваются эти проблемы. Проведенные ранее исследования выявили, что для реализации микроядерной архитектуры L4 так же спроведлива проблема деградации производительности при работе в SMP-системах [6]. Таким образом, фрагментация L4Linux преследует две цели: повышение защищенности компонентов системы через изоляцию и использование аппаратных средств защиты памяти, а также повышение производительности и отказоустойчивости засчет рекомпозии и избыточного дублирования элементов микроядерного окружения.

Данная работа посвящена промежуточным результатам процесса разработки архитектуры фрагментированной системы паравиртуализации. В ней описаны существующие подходы по разделению L4Linux на независимые части, плюсы и минусы этих подходов, а также описана собственная методология по построению минималистичных окружений, ориентированных на запуск конкретных приложений.

II. Архитектура L4Linux

L4Linux представляет собой ядро Linux, перенесенное с аппаратной платформы на программный интерфейс, предоставляемый ядром L4 и его окружением. На рис. 1 представлена общая схема системы паравиртуализации. Как можно заметить на схеме, нижний уровень ядра Linux, выполняющий функции взаимодействия с апаратурой, заменен слоем L4 Hal. С использованием этого слоя все аппаратные операции, такие, например, как: работа с памятью, прерываниями, регистрами процессора и др. - заменяются L4 IPC, а функции аппаратной платформы обеспечивают программы окружения. Как следствие, паравиртуализированное ядро L4Linux исполняется в пользовательском пространстве внутри отдельного региона памяти. Далее мы рассмотрим механизмы взаимодействия компонент ядра L4Linux с компонентами окружения и устройствами.

A. Драйвера устройств и компоненты L4Linux

В монолитно-модульной системе драйвера устройств находятся в привилегированном пространстве и параллельно работают вместе с другими компонентами ядра. В микроядерной системе драйвера устройств исполняются в форме отдельной программы в виртуальной адресном пространстве. Для доступа к физическому адресному пространству устройств драйвер запрашивает у окружения

[1]Иногда в западной литературе используется термин кернелизация (kernelization)

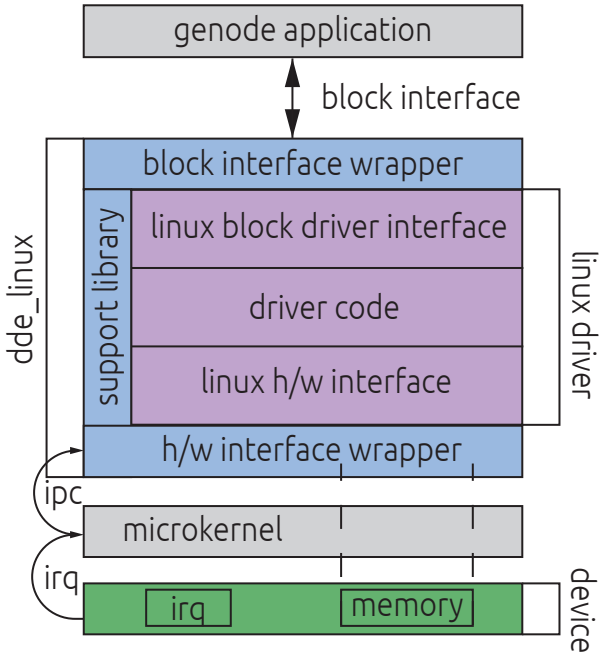

Рис. 3: Драйвер устройства

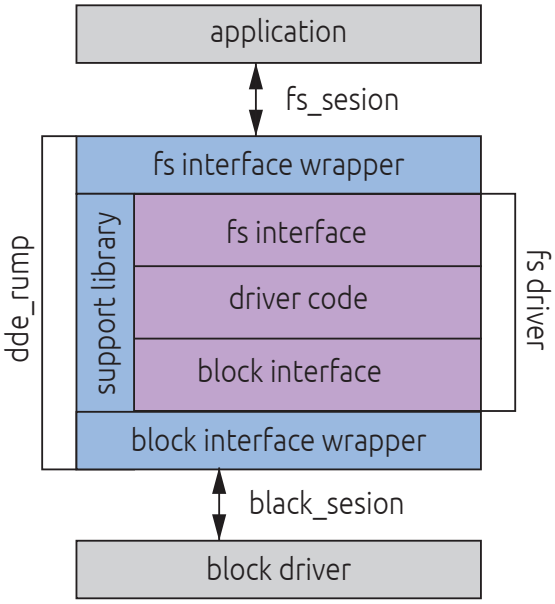

Рис. 4: Драйвер файловой системы

(обычно у процесса init) отображение диапазона физической памяти в свое адресное пространство. При наличии capability, то есть разрешения на эту конкретную операцию, сервис init выполняет эту операцию.

Работа драйвера внутри паравиртуализированного ядра L4Linux ничем не отличается от работы драйвера в форме отдельного процесса. Механизм взаимодействия между устройством и соответствующим драйвером идентичен - через запрос отображения физической памяти в адресное пространство L4Linux.

Помимо доступа к адресному пространству устройства, драйверу необходим механизм доставки прерываний. В случае исполнения в режиме ядра, драйвер может зарегистрировать обработчик по номеру прерывания, используя для этого механизмы ядра. При работе в пространстве пользователя у драйвера нет возможности ни работать с контроллером прерывания напрямую, ни использовать примитивы ядра. Вместо этого, аппаратные прерывания транслируются в сообщения. Чтобы процесс мог получать и обрабатывать прерывания, создается специальный поток (внутри процесса или контекста исполнения) для обработки прерываний. Далее этот поток регистрируется как обработчик прерываний с заданными номерами. Запущенный на исполнение, поток блокируется на вызове ожидания сообщения. Когда в системе возникает прерывание, оно поступает на обработку в микроядро, код обработки прерывания просматривает таблицу зарегистрированных обработчиков, и, в случае если прерыванию с данным номером соответствует поток-обработчик, данному потоку будет отправлено сообщение. При поступлении сообщения поток-обработчик разблокируется и вызовет код для обработки данного прерывания.

B. Программы L4Linux

Помимо работы самого ядра L4Linux в пространстве пользователя, использование паравиртуализации позволяет запускать двоичные программы, скомпилированные ранее под Linux. Каждая программа (процесс) L4Linux исполняется в отдельном адресном пространстве микроядерного окружения. Взаимодействие между программами L4Linux и ядром L4Linux осуществляется через механизм сообщений, поверх которого реализована эмуляция системных вызовов ядра Linux.

В Linux процессы пространства пользователя взаимодействуют с ядром при помощи интерфейса системных вызовов. Например, выполняя системный вызов на ARM-системе пользовательский процесс загружает в регистр r7 номер системного вызова, а в регистры r6-r1 загружаются параметры. После этого выполняется инструкция SVC (SWI) и генерируется исключение supervisor call (software interrupt). В результате этого управление передается на соответствующий обработчик исключения, в котором реализован диспетчер системных вызовов. В системах на базе ядер L4 взаимодействие процессов с микроядром реализовано сходным образом, с той лишь разницей, что в L4 имеется только один системный вызов (против более чем 200 системных вызовов Linux), который используется для передачи сообщений IPC.

В L4 системах имеется возможность назначить потоку свой поток обработки исключений (exception handler). В результате, когда поток создаст исключение, микроядро не вызывает код системного вызова, а отправляет сообщение IPC потоку обработчику. На этой возможности базируется реализация интерфейса системных вызовов в L4Linux:

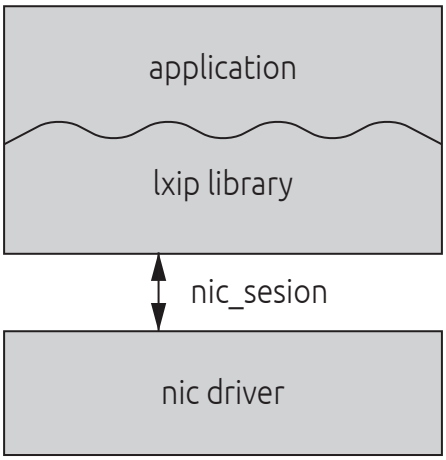

Рис. 5: Сетевой стек линукс в Genode

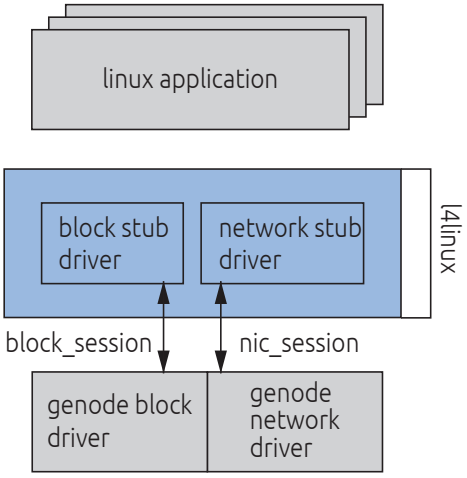

Рис. 6: L4Linux в окружении Genode

сервер L4Linux служит обработчиком исключений для всех процессов L4Linux. При доставке сообщения серверу об исключительной ситуации вызывается соответствующий код диспетчера системных вызовов L4Linux (рис. 2). Так как обработчиком исключительных ситуаций для программ L4Linux является само ядро L4Linux, то, фактически, у программ L4Linux нет возможности коммуницировать с программами окружения микроядра - при попытке отправить системный вызов этот запрос обрабатывается ядром L4Linux. Иными словами, программы L4Linux изолированы от программ окружения, а передавать сообщения программам окружения может только ядро L4Linux, так как у него нет особого обработчика исключительных ситуаций.

Работа с виртуальным адресным пространством пользовательских программ L4Linux осуществляется по такому же принципу. Процессу L4 может быть назначен свой обработчик исключений типа ошибка страницы (pagefault handler, pager). Для процессов L4Linux таким обработчиком является сервер L4Linux. Когда в процессе возникает ошибка обращения к странице, сервер L4Linux получает сообщение IPC и отображает необходимую страницу в адресное пространство процесса.

Сервер L4Linux при запуске запрашивает у окружения регион памяти, который впоследствии используется для процессов L4Linux. Чтобы процессы, размещенные в данном регионе, могли исполняться, L4Linux запрашивает этот регион, как доступный на исполнение. Именно это является фундаментальной причиной невозможности применения средств защиты стека (ASLR, NX-бит) для отдельных сегментов программ L4Linux и для ядра L4Linux целиком.

III. Извлечение компонент

Попытки исполнения отдельных компонент ядра Linux в окружении микроядра уже предпринимались разными командами разработчиков и исследователей. Одним из первых таких проектов был DDE (Device Driver Environment) Kit [7]. DDKEkit представляет собой набор библиотек и заголовочных файлов. Этот слой сопряжения, с одной стороны, реализует донорский интерфейс ОС, необходимый для работы компонента, а с другой - интерфейс, принятый для работы с данным классом устройств или сервисов в микроядерном окружении. Примерами DDEkit могут служить dde_linux, dde_oss, dde_ipxe и dde_rump, созданные для микроядерного фреймфорка Genode, а также DDEkit/linux, созданный для ОС, основанных на ядрах L4 и впоследствии перенесенный в ОС MINIX3 и GNU/Hurd. В отличие от фрагментации L4Linux, в рамках этих проектов преследовалась цель переноса функционала компонентов из оригинальных ядер в окружение микроядра. Фрагментация L4Linux позволяет сохранить функционал всего ядра, а не только какой-то отдельной части. Приведем примеры некоторых подходов при переносе компонент.

A. Драйвера устройств

Для реализации поддержки доступа к запоминающим устройствам, подключенным по интерфейсу USB (USB storage), в фреймворке Genode используется USB-стек и соответствующие драйверы устройств, перенесенные из ядра Linux с помощью dde_linux. Схема драйвера представлена на рис. 3 Для работы с аппаратным обеспечением в dde_linux используются механизмы доступа к памяти устройств и обработки прерываний аналогичные тем, что используются в L4Linux. Интерфейс блочного устройства Linux, предоставляемый драйвером, преобразуется в интерфейс блочного устройства, принятый во фреймфорке Genode.

B. Драйвера файловыйх систем

Другим примером использования DDEKit может служить драйвер файловой системы ext2, перенесенный в окружение Genode. Для работы данного драйвера используется компонент dde_rump, реализующий слой совместимости с исходными текстами проекта Rump [8]. Rump представляет собой проект по переносу некоторых компонентов ОС NetBSD в пространство пользователя. В

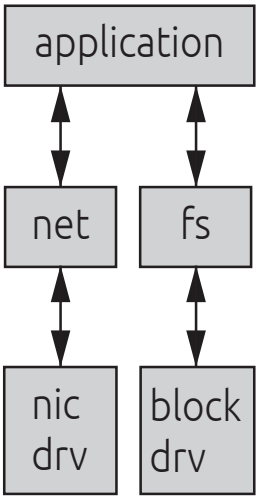

Рис. 7: Вариант декомпозиции L4Linux

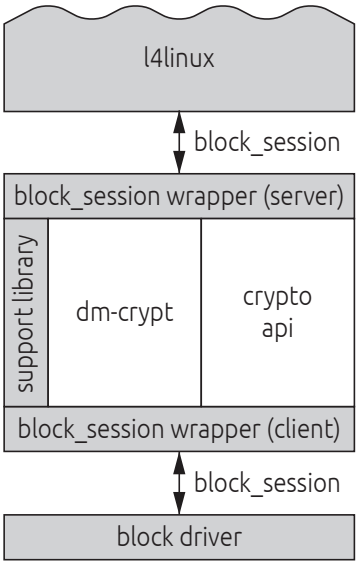

Рис. 8: dm-crypt в микроядерном окружении

данном драйвере, при помощи dde_rump, обеспечивается совместимость между интерфейсом блочного устройства, принятым в Rump и в Genode, с интерфейсом файловой системы. Схема устройства драйвера представлена на рис. 4.

Микроядерное окружение не ограничивает использование какого-то определенного механизма сопряжения. Для создания обычной системы, использующей сервис файловой системы и сервис блочного устройства, могут быть одновременно использованы как и dde_linux так и dde_rump. Для поддержки совместимости необходим зафиксированный API, благодаря которому сервис файловой системы может обращаться к сервису носителя, а то, что лежит за этим API - интерфейс драйвера Linux или NetBSD - значения не имеет.

C. Сеть

Одним из первых компонентов, повторно использованных из других операционных систем, в микроядерных окружениях был сетевой стек и набор сетевых драйверов. Например, стек lwip или стек ядра Linux lxip. Как и в примере с блочным устройством и файловой системой, для использования заимствованного компонента необходимо объявить и придерживаться API (для сетевого устройства это интерфейс сетевого устройства - Nic_session). На рис. 5 представлена схема сетевой подсистемы Genode.

D. L4Linux в окружении Genode

Существует вариант L4Linux, адаптированный для работы в окружении Genode. В отличии от L4Linux, работающего в окружении L4Re, в Genode L4Lnux не имеет непосредственного доступа к аппаратному обеспечению, а использует паравиртульные драйверы (рис. 7). Паравиртуальный драйвер (stub driver) представляет собой слой совместимости между интерфейсом устройства, предоставляемым драйвером в Linux, и интерфейсом, предоставляемым драйвером аналогичного устройства в Genode.

Несмотря на отсутствие доступа к аппаратной платформе из L4Linux в Genode, обозначенные выше проблемы, связанные с изоляцией компонент, остаются актуальными. В тоже время, можно расценивать извлечение драйверов устройств из пространства L4Linux в качестве первого шага к декомпозиции. Задачей нашего проекта является продвинуться вперед на два шага: разделить оставшиеся части ядра на блоки (шаг 1) и разработать механизм и методологию избыточного дублирования компонент с целью минимизации межпроцессного взаимодействия (шаг 2).

IV. Фрагментация ядра L4Linux

Описанные в разделе III подходы по использованию компонент Linux в окружении можно упрощенно представить в следующим виде:

- Выбирается изолированная подсистема ядра Linux
- Анализируется API подсистемы и зависимости в заголовочных файлах и других подсистемах
- Зависимые заголовочные файлы вместе с исходным кодом извлекаются исходного кода ядра Linux
- API подсистемы накладывается на механизм сообщений микроядерного окружения.

К этому списку добавляется дополнительный шаг, если целью является не просто исполнение какой-то подсистемы в окружении микроядра, но и сохранение связи с оставшимися подсистемами L4Linux. Например, при извлечении драйверов сетевых устройств с сохранением возможности передачи пакетов в L4Linux. Таким дополнительным шагом является перенос надсистемной части API (то есть

интерфейса взамодействия ядра и подсистемы со стороны ядра) на интерфейс сообщений микроядра (рис. 7).

Этот принцип извлечения компонент ядра является универсальным, различаются лишь методы его использования. Как упоминалось ранее, помимо разделения L4Linux на компоненты, мы ставим целью реализовать рекомпозицию системы паравиртуализации с целью уменьшения межпроцессного взаимодействия. То есть критичной для нас является возможность подготовки минимального набора компонентов L4Linux, исполняемых на одном ядре, для обеспечения функционирования соответствующей программы L4Linux. Как следствие, нам важна не сама по себе подсистема, сколько возможность объединить минимальный набор подсистем в *сборку*.

Отталкиваясь от необходимости комбинирования изолированных компонент L4Linux, мы проанализировали структуру Linux с целью выделения минимального набор подсистем. Было выявлено, что для обеспечения работы пользовательских процессов необходимо реализовать, либо частично перенести из Linux следующие компоненты: загрузчик процессов, диспетчер системных вызовов, обработчик исключений типа "ошибка страницы". Эти подсистемы являются необходимыми компонентами, используемыми каждой программой. Опционально, для реализации дополнительных функций отдельных программ, можно отделить виртуальную файловую систему (VFS) и сетевой стек в виде сервиса, чтобы несколько процессов могли пользоваться одним и тем же сетевым стеком, как это происходит в Linux.

V. Ход работы

Для опробации метода рекомпозиции мы решили реализовать сервис блочного шифрованного устройства, используемого программой L4Linux. Выбор именно этого примера обусловлен несколькими причинами. Во-первых, криптографическая подсистема ядра в значительной степени изолирована от остальных компонент. Во-вторых, блочный ввод-вывод является важным элементом ядра, и его извлечение полезно для последующих экспериментов с блочными устройствами. В-третьих, минимальный набор компонентов ядра, криптографическая подсистема, блочный ввод-вывод и пользовательская программа L4Linux, использующая эти сервисы, легко компонуются на одном или нескольких ядрах процессора, что позволит продемонстрировать изменения производительности. Так же тестовую систему мы дополнили драйвером сетевой карты и TCP/IP стеком, превратив тем самым тестовый стенд в модель сетевого файлового сервера (рис. 8).

Так как драйвера блочного устройства ввода-вывода в значительной степени изолированы, их извлечение из ядра не представляет существенной сложности (подобная работа уже выполнялась [9] ранее для USB-устройств). Что касается других подсистем, например *device mapper* (специализированная подсистема ядра, занимающаяся отображением физических устройств в высокоуровневые виртуальные блочные устройства), их извлечение их потребует разработки нового API. В частности, подсистема device mapper должна быть выделена в отдельный слой сопряжения между блочным драйвером и драйвером файловой системы, так как драйвера файловой системы не работают в Linux с устройствами напрямую. Криптографическая подсистема *crypto api* должна быть извлечена целиком из ядра и реализована в виде отдельной библиотеки.

На момент написания статьи проводятся работы по переносу подсистемы device mapper в пространство микроядра. При разработке слоя совместимости между device mapper и окружением микроядра было принято решение не переносить подсистему целиком, а обеспечить возможность использования драйверов, реализующих интерфейс device mapper target. В качестве первого драйвера был взят dm-crypt.

VI. Заключение

Данная работа является начальным этапом в серии работ, посвященных фрагментации и рекомпозиции системы паравиртуализации L4Linux с целью повышения отказоустойчивости, изоляции компонентов и производительности системы паравиртуализации в микроядерном окружении. С научной точки зрения данный проект ставит перед собой цели переосмысления архитектурных представлений об организации современных операционных систем и методов улучшения их свойств. В работе представлены первые шаги в этом направлении: описаны мотивация и текущие подходы к организации окружений микроядерных ОС, рассмотрены архитектурные особенности реализации различных компонентов окружений в контексте повторного использования кода. Кроме того, описаны направления разработки и исследований, а также методы достижения результатов.

В последующих работах мы ставим целью полное описание методологии фрагментации и рекомпозиции, применения этого метода к более сложным подсистемам L4Linux, сбор эмпирических данных о производительности и защищенности фрагментированных компонентов.

Список литературы

[1] Lackorzynski Adam [и др.]. L4Linux porting optimizations // Master's thesis, Technische Universitat Dresden. 2004.

[2] Сартаков В. А., Тарасиков А. С. Защита микроядерного окружения L4Re от атак на стек // Безопасность информационных технологий. 2014. Т. 4.

[3] The multikernel: a new OS architecture for scalable multicore systems / Andrew Baumann, Paul Barham, Pierre-Evariste Dagand [и др.] // Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles / ACM. 2009. С. 29–44.

[4] Klein Gerwin, Derrin Philip, Elphinstone Kevin. Experience report: seL4: formally verifying a high-performance microkernel // ACM Sigplan Notices / ACM. Т. 44. 2009. С. 91–96.

[5] Keep net working - on a dependable and fast networking stack / Tomas Hruby, Dirk Vogt, Herbert Bos [и др.] // In Dependable Systems and Networks. IEEE, 2012. С. 1–12.

[6] Сартаков Василий, Тарасиков Александр. Анализ производительности сетевой подсистемы микроядерного окружения Genode // Tools & Methods of Program Analysis. 2013.

[7] Weisbach Hannes. DDEKit Approach for Linux User Space Drivers. 2011.

[8] Kantee Antti. Flexible Operating System Internals: The Design and Implementation of the Anykernel and Rump Kernels // Doctoral dissertaions, Aalto University. 2012.

[9] Vogt Dirk. USB for the L4 Environment. Ph.D. thesis: Master's thesis, TU Dresden. 2008.