Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение эвм и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

«Классификация методов модификации ядра Linux»

Студент **ИУ7-55Б** _____ **Романов С. К.**

Руководитель _____ **Оленев А. А.**

2022 г.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ **5**

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ **6**

ВВЕДЕНИЕ **7**

1 Анализ предметной области **8**

1.1 Базовые понятия и термины 8

1.1.1 Ядро Linux 8

1.1.2 Виртуальная память 9

1.1.3 Пространство ядра и пространство пользователя 12

1.2 Способы модификации ядра Linux 13

1.2.1 Модификация ядра Linux 14

1.2.2 Задачи модификации ядра Linux 14

2 Существующие решения **17**

2.1 Обзор методов модификации ядра Linux 17

2.1.1 Методы модификации ядра Linux, основанные на перекомпиляции ядра 17

2.1.2 Метод модификации ядра Linux, основанный на встраивании модулей ядра 19

2.1.3 Kernel Live Patching 21

2.1.4 eBPF 23

2.2 Критерии сравнения методов модификации ядра 27

2.3 Сравнение методов модификации ядра 28

ЗАКЛЮЧЕНИЕ **30**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **31**

ПРИЛОЖЕНИЕ А **33**

ОПРЕДЕЛЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие термины с соответствующими определениями.

Application Programming Interface — «Интерфейс прикладного программирования» - набор инструментов программирования, который разрешает программе взаимодействовать с другой программой или операционной системой и помогает разработчикам программного обеспечения создавать свои собственные приложения (= части программного обеспечения)[1]

Loadable Kernel Module — «Загружаемый модуль ядра» - участок кода, который загружается в операционную систему во время работы операционной системы и выгружается из нее в любое время. Модули ядра расширяют функциональность ядра без необходимости перезагрузки системы.[2]

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие сокращения и обозначения.

ОС — Операционная система.

API — Application Programming Interface.

LKM — Loadable Kernel Modules.

eBPF — extended Berkeley Packet Filter.

ВВЕДЕНИЕ

При разрабатывании комплексных систем возникает необходимость создания надёжной системы на основе Linux. Для таких случаев появляется настраивается ядро системы таким образом, чтобы система работала, будучи заточенную под конкретную задачу. Ядро Linux хоть и проектировалось быть универсальным, однако возникают ситуации, когда базовая функциональность не решает поставленные задачи.

Так например, сегодня расширения ядра обеспечивают функциональность большинства облачных систем.[3]. Также в ядро Linux внедряются драйвера, которые обеспечивают работу устройств, подключаемых к компьютеру. Из-за этого возникает вопрос о модификации ядра Linux и внедрении в него новых функций.

Цель работы – классифицировать методы модификации ядра Linux.

Для достижения поставленной цели требуется решить следующие задачи:

- Провести обзор существующих методов модификации ядра Linux;
- Провести анализ методов модификации ядра Linux;
- Сформулировать критерии классификации методов модификации ядра;
- Классифицировать существующие методы модификации ядра.

1 Анализ предметной области

1.1 Базовые понятия и термины

Прежде чем приступить к теме исследования, следует описать предмет исследования и связанные с ним термины. В данной секции описаны понятия, которые будут использоваться в работе.

1.1.1 Ядро Linux

Ядро Linux — это основной компонент операционной системы Linux,2 поскольку ядро отвечает за управление ресурсами компьютера и за взаимодействие приложений с аппаратными средствами. Ядро должно в первую очередь выполнять две задачи:

1) Взаимодействие с аппаратными компонентами, обслуживая низкоуровневые элементы, входящие в состав аппаратной платформы.
2) Поддерживать среду выполнения для программ, которые будут выполняться в операционной системе.

Для выполнения приведенных выше задач, ядро Linux реализует ряд архитектурных атрибутов. Так, например, ядро разделено на отдельные подсистемы, каждый отвечающий за конкретные функции системы. Примерами таких функций – управление памятью, файловой системой, сетевыми интерфейсами, которые используются другими подсистемами. Такое разделение упрощает разработку и поддержку ядра, а также увеличивает гибкость ОС.

Linux считается монолитной системой, поскольку все службы системы объединяются в одном цельном ядре, что отличает данную концепцию от микроядерной архитектуры, где каждая служба реализуется в отдельном модуле-ядре. При запуске операционной системы ядро загружается в оперативную па-

мять компьютера и остается там до тех пор, пока операционная система не будет выключена. После того, как ядро загружено в оперативную память, ядро выполняет инициализацию и начинает планировать задачи. Когда конкретная задача загружается, ей присваивается виртуальное адресное пространство, в котором1 будет пребывать.

Обобщая вышесказанное, можно сказать следующее: ядро решает, какие ресурсы в каком порядке выделяются задаче для её выполнения. В действительности же ядро действует как интерфейс между пользовательскими приложениями и оборудованием.

Следует отметить, что не каждая задача получает равный доступ к ресурсам компьютера и подсистемам ядра, поскольку в ином случае, например, при возникновении ошибки в одной из задач, появляется шанс повредить как другие задачи, так и саму операционную систему. Также следует понимать, что это далеко не единственная причина такого ограничения, однако, чтобы понять, как ядро определяет к каким ресурсам и подсистемам процессы могут иметь доступ, необходимо для начала представить модель загрузки задач в оперативную память.

1.1.2 Виртуальная память

Виртуальная память - это способ организации памяти, при котором процессору предоставляется не физические адреса, а виртуальные адреса, которые в дальнейшем переводятся в физические адреса с помощью таблиц страниц.

Преимущество использования виртуальных адресов заключается в том, что операционной системе становится доступно управление представлением памяти, предоставляемой программному обеспечению. На практике каждое приложе-

ние использует собственный набор виртуальных адресов, которые будут отображаться в системе. Каждый раз, когда операционная система переключается между приложениями, происходит перепрограммирование карты паммяти. Это означает, что виртуальные адреса для текущего приложения будут сопоставлены с правильным физическим расположением в памяти[4].

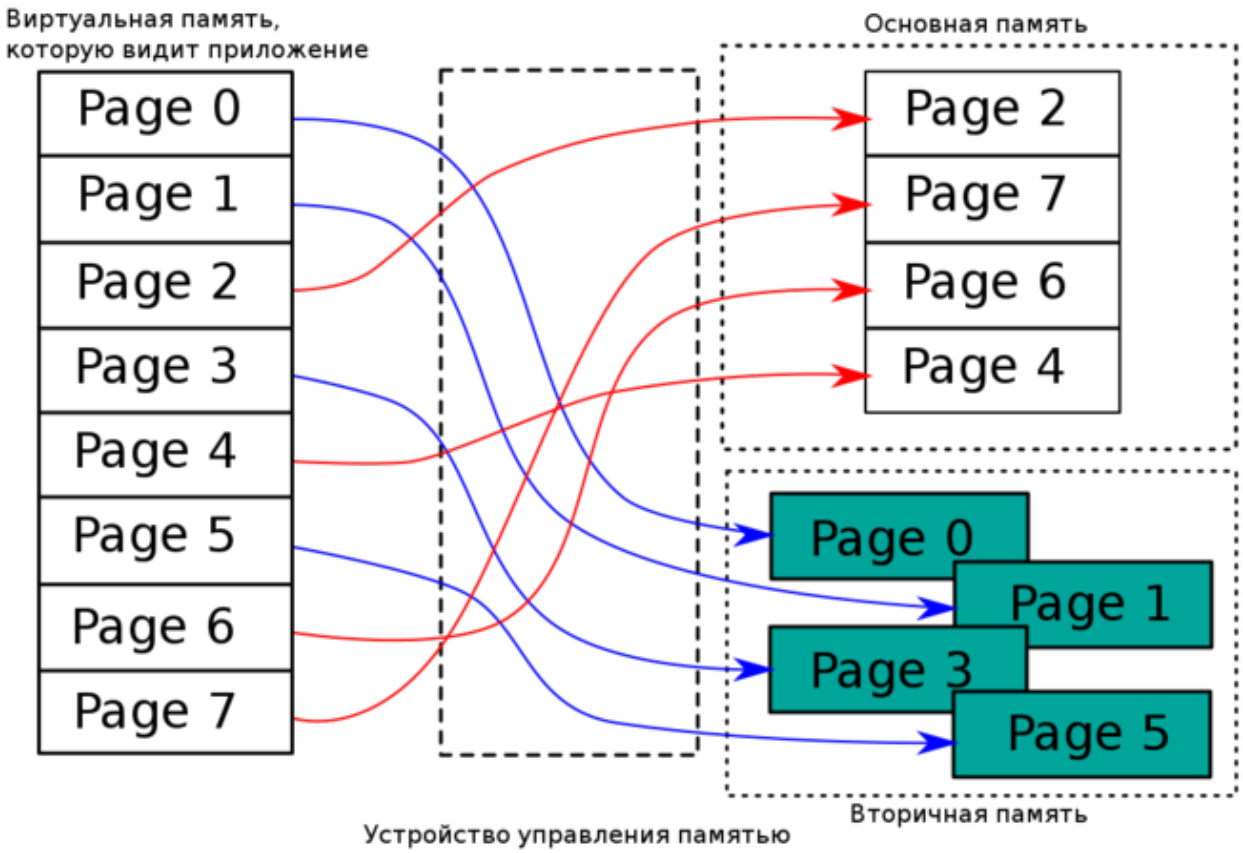

Рисунок 1 – Схема устройства виртуальной памяти[5].

Помимо прочего, для виртуальной памяти разработаны инструменты для защиты памяти от несанкционированного доступа. Виртуальная память распределяется на страницы с фиксированным размером, которые далее распределяются между процессами. В случае, если процесс попытается получить доступ к странице, которая не принадлежит ему, то процессор сгенерирует исключение, которое будет обработано операционной системой.

Ввиду этого факта, виртуальная память приобретает важное дополнительное

свойство. Различные программы могут быть запущены в отдельных изолированных областях памяти, где они имеют доступ только к тем областям данным, которые им необходимы для работы. Отсюда мы переходим к появлению важного двух важных понятий: пространство ядра и пространство пользователя.

1.1.3 Пространство ядра и пространство пользователя

Современные операционные системы, в частности Linux, разделяют виртуальную память на пространство ядра и пространство пользователя. В пространство ядра помещаются само ядро и необходимые для работы модули. Пространство пользователя в свою очередь разделяется на несколько частей. Каждый процесс получает собственное пространство виртуальной памяти, в котором хранятся его данные, стек и куча . Как было сказано выше, процессы в пространстве пользователя не имеют доступа к чужим областям памяти, однако такой возможностью обладают процессы запущенные в пространстве ядра, которые имеют доступ ко всем областям памяти. Впервые данная концепция появилась в системе Multics, которая включала в себя 8 «колец защиты»[1], однако в UNIX-системах де-факто используютсятолько два:

Кольцо 0 - отвечающее за ядро, и кольцо 3 - отвечающее за пользовательские приложения.

Кольца 1 и 2, предоставляемые некоторыми архитектурами процессоров, такие как x86, показали себя неэффективными ввиду сложности портирования на различные архитектуры процессоров, а также ряда других причин.

Схематично данная концепция представлено на рисунке 2.

[1]Такие архитектуры процессоров, как Intel x86, переняли эту концепцию и предоставляют 4 кольца защиты на уровне процессора, однако на практике 1-ое и 2-ое кольца практически нигде не используются.

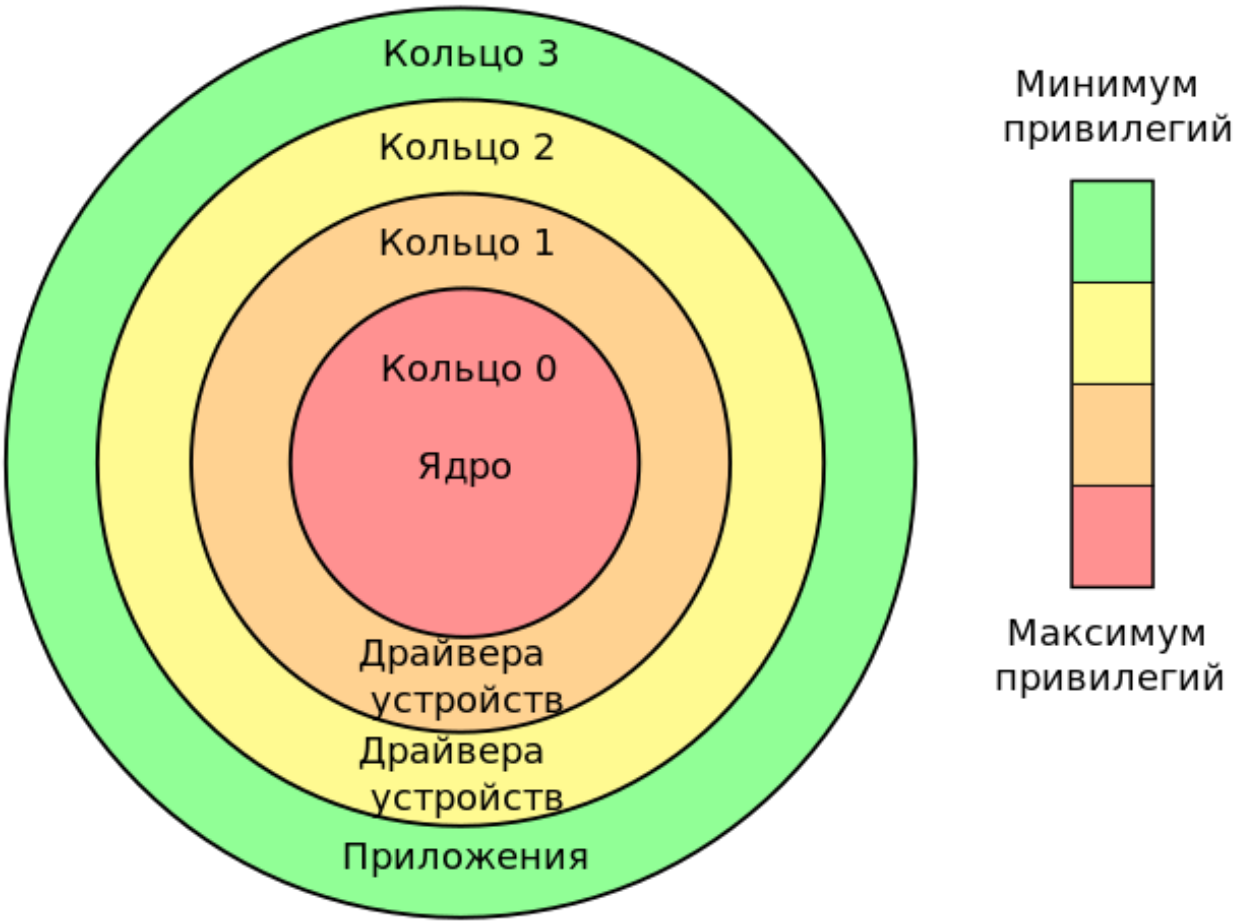

Рисунок 2 – Кольца защиты[6].

Идея колец защиты заключается в том, что каждое кольцо имеет собственный набор инструкций, которые процесс на данном кольце защиты может выполнять, и чем ближе кольцо к нулевому, тем больше прав имеет процесс. Однако поскольку в UNIX-системах используются только два кольца, то далее в работе будем к ним относиться как к пространству ядра и пространству пользователя для 0-го и 3-го колец соответственно, пренебрегая остальными.

1.2 Способы модификации ядра Linux

В данной секции будет обозначено, что подразумевается под модификацией ядра Linux для определения в дальнейшем того, какие методы модификации ядра Linux существуют.

1.2.1 Модификация ядра Linux

В данном разделе описаны критерии модификации ядра Linux.

- **Модификация ядра Linux** – это изменение кода ядра Linux, которое не включает в себя изменение структуры ядра Linux. Таким образом программист волен изменять части ядра Linux, не затрагивая саму структуру ядра. Изменения, которые включают в себя изменение структуры ядра Linux, называются **переписыванием ядра Linux**, что выходит за рамки данного исследования.
- Главное отличительная особенность модификаций ядра от других программ заключается в том, что работа модификаций происходит в пространстве ядра. Дополнительный функционал, который добавляется в ядро Linux, оперирует существующими структурами ядра Linux или создает собственные без ограничений со стороны ОС.
- Модификации ядра имеют доступ ко всему функционалу ядра Linux, включая системные вызовы, обмен данными с устройствами и т.д.
- Также модификации ядра манипулируют в любой области памяти, что позволяет взаимодействовать с другими модификациями ядра или самим ядром.

1.2.2 Задачи модификации ядра Linux

Конкретные задачи модификации ядра крайне обширны и во многом зависят от конкретно поставленной цели конкретного проекта. Вычисления на уровне ядра дают ряд преимуществ, такие как прямой доступ к оборудованию системы, ускорение работы программы за счет отсутствия необходимости переключения между пространствами, манипуляция данными в любой области

памяти и т.д. Однако для большого количества случаев такие преимущества либо не являются необходимыми, либо недостатки таких подходов нивелируют эти преимущества. Общее правило, которое может быть сформировано для всех методов модификации ядра, выглядит следующим образом:

- Если вычисления на уровне ядра не являются необходимыми, то они не должны быть реализованы.
- Если вычисления на уровне ядра все же являются необходимыми, то следует провести тщательный анализ, соизмеримое по времени с написанием программы, на тему того, нет ли других альтернатив.
- Только в том случае, если вычисления на уровне ядра являются необходимыми и других альтернатив не существует, то с особой осторожностью можно приступить к написанию таких программ.

Другими словами можно сказать следующее: модификация ядра Linux должна быть последним вариантом, когда остальные варианты исчерпали себя.

Следующий список дает примеры некоторых задач, которые должны быть решены на уровне ядра, однако ни в коей мере он не является полным или не лишенным исключений:

1) Написание приложений, таких как драйвера устройств, с доступом к низкоуровневым ресурсам, которые не могут быть предоставлены другими способами.
2) Реализация алгоритмов, которые должны быть выполнены с высокой точностью по времени и/или пространству (например, мониторинг ресурсов системы или совместное использование ресурсов)[7].
3) Написание программ, которые должны быть доступны всем пользователям системы.[7].
4) Также следует перейти в пространство ядра, где накладные расходы, такие как смена пространств пользователь-ядро, становится неприемлемыми для эффективной или корректной работы программы[7]. Чаще всего в

таких случаях речь идет об облачных вычислениях[8] или любых других вычислениях, требующих высокой производительности.

2 Существующие решения

2.1 Обзор методов модификации ядра Linux

В данном разделе рассмотрены методы модификации ядра Linux, которые используются при разрабатывании нового функционала ядра. Все они различаются по своим особенностям и применяемым технологиям.

2.1.1 Методы модификации ядра Linux, основанные на перекомпиляции ядра

На момент выпуска версии 1.0 ядра Linux существовал единственный метод модификации ядра Linux, основанный на перекомпиляции ядра. В этом методе исходный код Linux изменяется под конкретные задачи и перекомпилируется с использованием специальных опций компилятора. В результате получается модифицированное ядро Linux, котороеиспользуется для запуска обновленной системы. Однако, такой метод модификации ядра Linux имеет ряд недостатков:

1) Необходимость перекомпиляции ядра Linux для каждого нового модуля.

   Это означает, что при необходимости добавления новой модификации в систему, необходимо будет внести нужные изменения и перекомпилировать ядро Linux, что влечет за собой массу проблем. Так, например, при перекомпиляции ядра необходима полная остановка системы и таким образом этот метод модификации ядра не подходит для систем, в которых необходимо добавлять новые модули динамически.
2) Сложность добавления кода в ядро Linux.

   Для добавления кода в ядро Linux необходимо уметь работать с языком

и компилятором языка C, а также с инструментами конфигурации ядра Linux, знать интерфейс прикладного программирования ядра (API).[9]. Иными словами данный подход требует крайне высокой квалификации команды разработчиков, что замедляет процесс разработки и внедрения новых модулей в систему.

3) В случае если при добавлении нового кода в ядро Linux была допущена ошибка, то есть шанс повредить систему и сделать её нерабочей.

Однако у статического метода модификации ядра Linux есть и преимущества:

1) Скорость работы системы.

   В статическом методе модификации ядра Linux не используется динамическая загрузка модулей, поэтому модули ядра Linux загружаются в память только один раз, при запуске системы. Поэтому, в статическом методе модификации ядра Linux исключена возможность динамического добавления новых модулей в систему, что увеличивает скорость работы системы.

2) Отсутствие альтернативных методов модификации ядра Linux.

   Иногда необходимые дополнения не могут быть реализованы в виде модулей ядра Linux, поэтому в этом случае необходимо модифицировать исходный код ядра Linux.

3) В версии ядра 6.1 добавили поддержку языка Rust, что частично нивелировало сложность написания кода для ядра, поскольку данный язык дает разработчикам писать высокоуровневый код без потери производительности системы.

2.1.2 Метод модификации ядра Linux, основанный на встраивании модулей ядра

Второй метод модификации ядра Linux, основанный на встраивании модулей ядра, появился в 1995 году, когда в версию 1.2 добавили поддержку LKM - Loadable Kernel Module. В этом методе ядро Linux не перекомпилируется. Вместо этого используется специальный модуль ядра, который встраивается в ядро Linux. В результате получается модифицированное ядро Linux, которое можно использовать для работы системы.

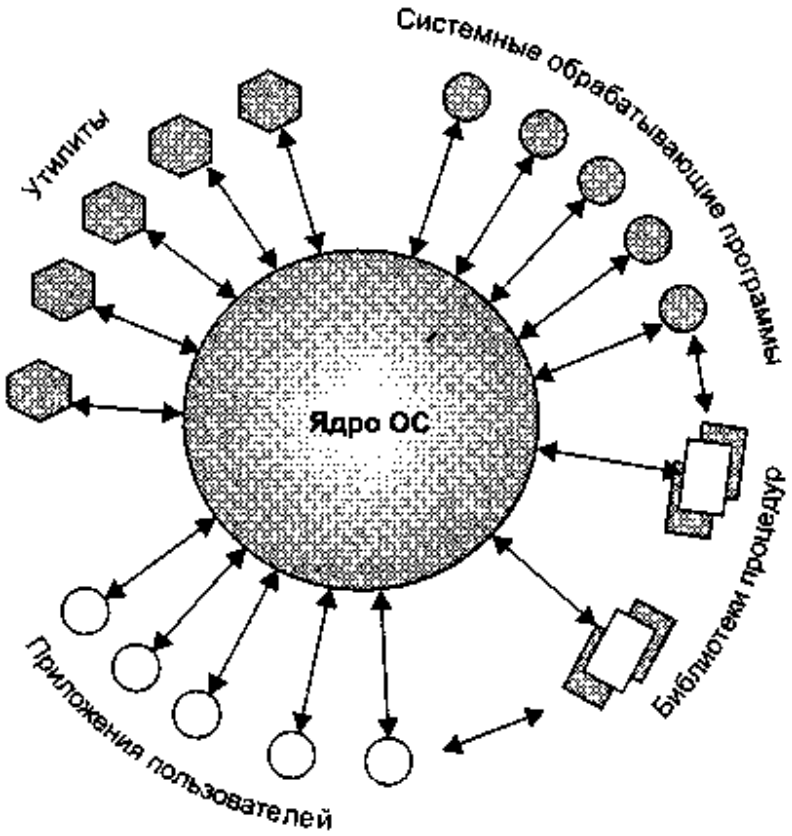

Рисунок 3 – Взаимодействие ядра с загружаемыми модулями и пользовательскими приложениями[10].

В качестве примера можно привести модуль ядра, который предназначен для работы с сетевыми интерфейсами. В этом модуле ядра реализованы функции, которые позволяют получить информацию о сетевых интерфейсах, а также управлять ими. При этом, модуль ядра не содержит в себе никаких функций, ко-

торые не относятся к работе с сетевыми интерфейсами. Таким образом, модуль ядра не засоряет ядро Linux ненужными функциями.

Плюсы:

1) Модуль ядра может быть загружен в ядро Linux во время работы системы.

   Таким образом, система продолжает работать, динамически меняя конфигурацию по мере необходимости.
2) Модуль ядра может сэкономить оперативную память, потому что появляется возможность загружать модификации только тогда, когда появляется их необходимость, в то время как все части базового ядра постоянно остаются загруженными в физическом хранилище, а не только в виртуальном.

3) Еще одно преимущество LKM – помощь в диагностировании проблем системы. Ошибка в драйвере устройства, связанном с ядром, может остановить загрузку системы, что вызывает сложности при определении проблемы, какая часть базового ядра вызвала проблемы. Однако если тот же драйвер устройства будет загружен в систему динамически, то базовое ядро способно запуститься и продолжать работу еще до загрузки драйвера устройства. Если система завершает работу после того, как базовое ядро было запущено, то появляется возможность отследить проблему до вызывающего проблемы драйвера устройства и не загружать до тех пор, пока проблема не будет решена.

   Недостатки:

1) LKM приводит к проблемам с производительностью системы.

   Поскольку LKM загружаются в ядро Linux во время работы системы, время загрузки системы увеличиввается.
2) В некоторых случаях LKM приводит к некоторым проблемам системы.

   Например, LKM приводит к проблемам совместимости, если нет совме-

стимости с базовым ядром.

3) Еще одним замечанием по поводу предпочтения модульного ядра статическому ядру является штраф за фрагментацию.

   Базовое ядро распаковывается в физическую непрерывную память; таким образом, базовый код ядра не подвергается фрагментированию. Как только система входит в состояние, в котором модули могут быть вставлены – любая новая вставка кода приведет к фрагментации ядра, что приведет к снижению производительности. Такое может произойти, например, после того, как были смонтированы файловые системы, содержащие модули ядра.

4) Так же как и при добавлении кода напрямую в ядро, если при работе модуля возникнет ошибка, то есть шанс, что система также экстренно завершит работу.

2.1.3 Kernel Live Patching

Kernel Live Patching - это функция ядра Linux, которая обновляет ядро Linux без перезагрузки системы. Первая программная реализация данной идеи принадлежит команде Джеффа Арнольда, студентов MIT, носит название Ksplice[11] а первая коммерческая версия была запущена в 2010 году. Исправление ядра в реальном времени используется при составлении стратегии управления серверами Linux и устранения уязвимостей.

Принцип Live Patching’а основан на том, что данный метод модификации ядра создает отдельный модуль из исправленного кода, а затем с помощью инструмента ftrace (трассировки функций) перенаправляет вызов от устаревшей функции к новой. На рисунке 4 показана схема работы представленного метода.

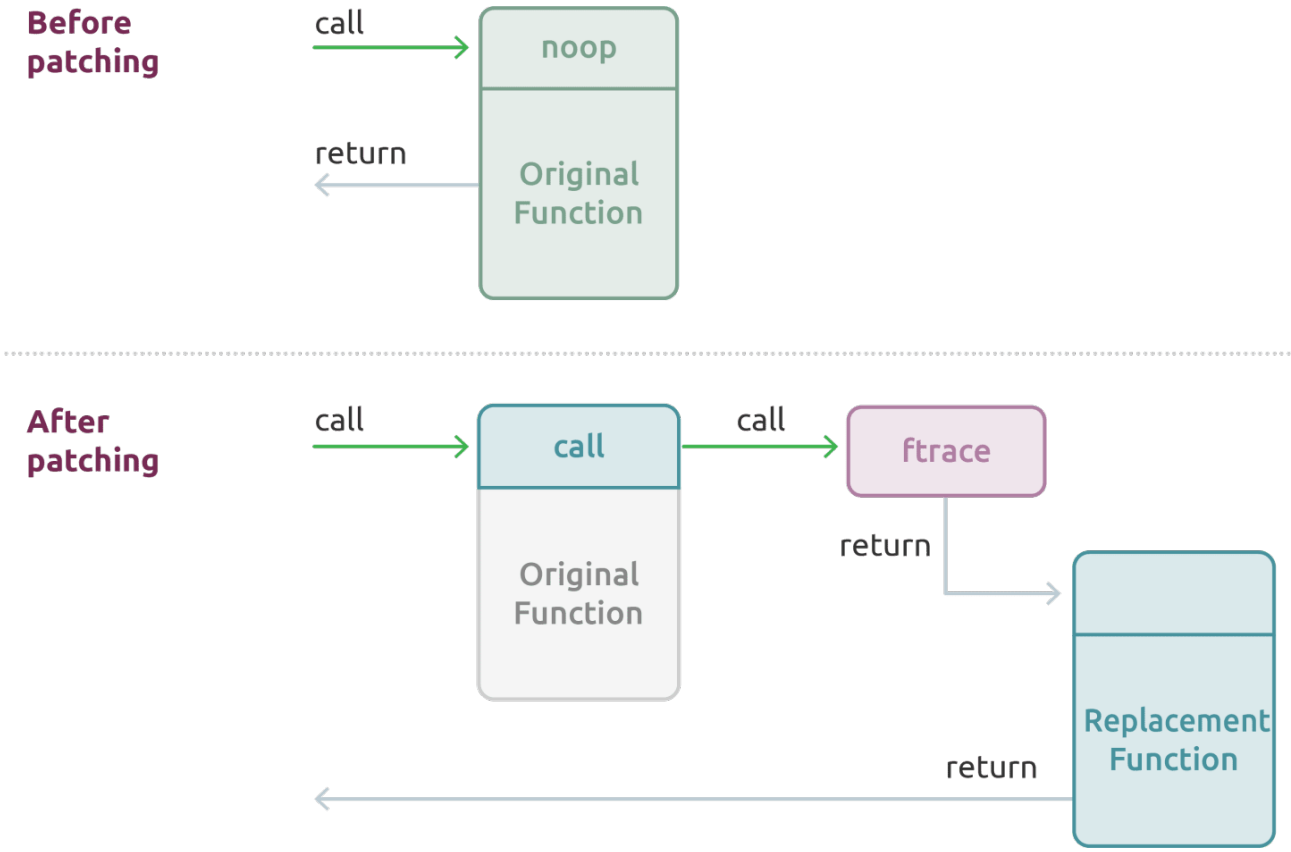

Рисунок 4 – Концепт Live Patching[12].

Преимущества:

1) Ядро Linux обновляется без перезагрузки системы.
2) Обновление системы требует времени и высокого уровня навыков системного администрирования. Live Patching избавляет персонал от рутинной работы по обслуживанию многочисленных серверов.
3) Обновления проходят быстро, что позволяет оперативное разворачивание новых функций.

Недостатки:

1) Внесение исправлений в ядро по-прежнему сложно — исправления должны быть написаны экспертами для каждой системы, и они зарезервированы только для важных исправлений безопасности. Даже в этом случае не гарантируется, что система не выйдет из строя. Так например различные оптимизации при компиляции могут незаметно изменить код таким образом, что это приводит к серьезным проблемам при применении

исправления[13].

2) Live Patching применяется только к небольшим и конкретным частям кода ядра и не используется для серьезных обновлений, которые затрагивают несколько компонентов или изменяют структуры данных.

   Изменения в структурах данных усложняют ситуацию, поскольку данные должны оставаться на месте и не могут быть расширены или переинтерпретированы. Хотя существуют методы, которые позволяют косвенно изменять структуры данных, некоторые изменения нельзя преобразовать в подобного рода исправления. В этой ситуации перезагрузка системы — единственный способ применить изменения.
3) Не все ядра поддерживают Live Patching. В различных ядрах используются разные методы управления процессом исправления и создания исправлений, а некоторые из них разработаны исключительно под определенные семейства Linux[14].

2.1.4 eBPF

eBPF (extended Berkeley Packet Filter) - это новая технология, которая позволяет встраивать программы в ядро Linux, не изменяя исходный код. В своей основе eBPF использует привилегированную способность ядра видеть и контролировать все ресурсы системы. С помощью eBPF запускаются изолированные программы в привилегированном контексте, которые взаимодействуют с ядром Linux.

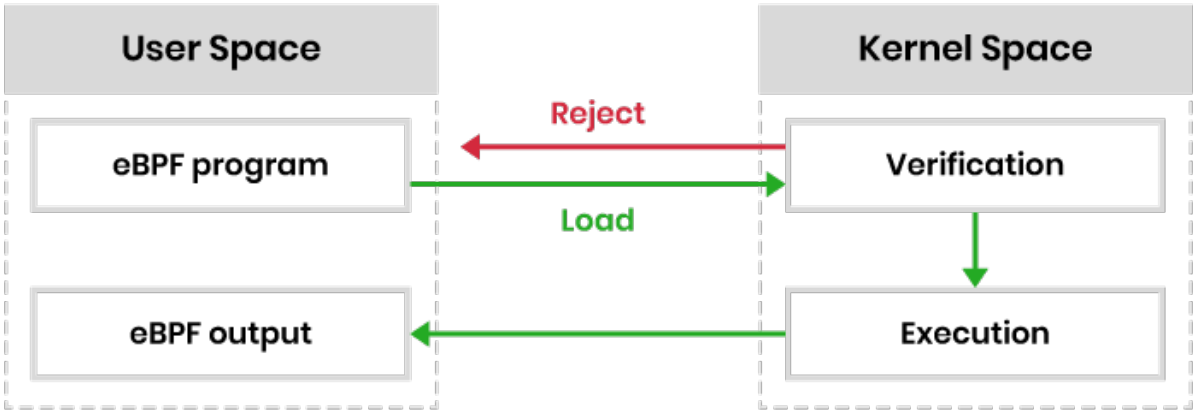

Рисунок 5 – Алгоритм eBPF[15].

eBPF работает по следующему принципу:

eBPF позволяет приложениям пространства пользователя упаковывать логику, которая будет выполняться в ядре Linux, в виде байт-кода. eBPF-программы вызываются ядром, когда происходят конкретные события, называемые «системными хуками». Примеры таких событий включают в себя системные вызовы, сетевые события и т.д.

Перед загрузкой в ядро eBPF-программа проходит конкретный набор проверок со стороны «верификатора». Только если все проверки пройдены успешно, программа eBPF преобразуется в байт-код, загружается в ядро и начинает ждать соответствующих событий. После того, как событие произошло, eBPF-программа выполняется в привилегированном контексте ядра. **Преимущества:**

1) eBPF позволяет встраивать программы в ядро Linux, не изменяя его исходный код. В то время как остальные методы вносят дополнительные факторы риска в работоспособность системы, eBPF, с данной точки, зрения безопасен. eBPF-программы перед компиляцией проходят строгую проверку со стороны одного из собственных модулей eBPF, «верификатора», в следствии чего можно быть уверенным в безопасности исполняемого кода.
2) eBPF позволяет писать программы на высокоуровневых языках програм-

мирования, что упрощает процесс разработки. Данная возможность обеспечивается с помощью компилятора LLVM, который позволяет компилировать код на различных языках в байт-код, который затем исполняется виртуальной машиной eBPF.

3) Также как и LKM, eBPF позволяет динамически загружать и выгружать программы во время работы системы. Поскольку eBPF является JIT-компилято, то он может компилировать программы в машинный код на лету, что позволяет изменять конфигурацию системы в реальном времени.

Недостатки:

1) Главный минус eBPF заключается в том, что количество функций, которыми программист манипулирует, сильно ограничено.

   eBPF обеспечивает повышенную безопасность, ограничивая доступ программ к ресурсам системы. Однако из-за такого ограничения, к каким частям ОС программа может получить доступ, функциональность данного метода также ограничена.

2) Поскольку eBPF выполняет программы в привилегированном контексте, то это может привести к утечке конфиденциальной информации. Исторически eBPF обладает множеством уязвимостей, которые могут быть использованы для получения нежелательного доступа к критическим ресурсам системы. Примером одной из таких уязвимостей является CVE-2021-4204[16], которая позволяет произвольному непривилегированному пользователю получить доступ к памяти ядра.

3) eBPF – молодой и пока еще развивающийся инструмент. В связи с этим в ходе разработки eBPF-программ могут возникать проблемы, связанные с отсутствием нужного инструментария, документации или её недостаточной информативностью.

В конце этого раздела стоит сделать оговорку касательно eBPF.

eBPF де-факто не является полноценным инструментом модификации ядра. eBPF — это набор инструкций, для которых ядро Linux предоставляет виртуальную машину, верификатор и некоторые вспомогательные функции. Программы запускаются внутри этого контекста выполнения и вызывают вспомогательные функции для расширения возможностей виртуальной машины. Во время выполнения программы eBPF на самом деле вызываются kprobe, или uprobe, или классификатор eXpress Data Path (XDP), или один из многих других типов программ пространства ядра, которые были выгружены в подсистему eBPF. Таким образом, eBPF-программы — это не полноценные модификации ядра Linux, а наборы инструкций, которые позволяют расширить возможности уже существующего модуля ядра. Но не смотря на это, по формальным признакам, eBPF подходит под все критерии, которые были описаны в начале данной работы, являясь при этом полезным инструментом при написании определенного типа программ, что и послужило причиной для его включения в данную работу.

2.2 Критерии сравнения методов модификации ядра

В данном разделе будут описаны критерии, которые будут использоваться для сравнения методов модификации ядра.

Таблица 1 – Критерии сравнения методов модификации ядра

Критерий	**Описание**
Производительность	Производительность программ.
Безопасность	Наличие гарантии, что внесенный код не вызовет остановку системы.
Скорость разработки	Является ли метод быстрым в разработке.
Гибкость	Возможность метода подстроиться под любые поставленные задачи.
Простота отладки	Является ли описанная модификация простой в отладке.
Поддержка	Поддержка метода разработчиками ядра при его написании.
Простота развёртывания	Является ли описанный метод простым в развёртывании на большом количестве машин.

2.3 Сравнение методов модификации ядра

В данном разделе будут описаны все описанные в работе методы модификации ядра, а также будет дано сравнение этих методов по всем критериям, указанным в таблице 1.

Таблица 2 – Сравнение методов модификации ядра

Критерий	Рекомпиляция	LKM	Live Patching	eBPF
Производительность	✓	✓	✓	✓
Безопасность	✗	✗	✗	✓
Скорость разработки	✗	✓	✗	✓
Гибкость	✓	✓	✗	✗
Простота отладки	✗	✓/✗[2]	✗	✓
Поддержка	✓	✓	✓	✗
Простота развёртывания	✗	✓	✓	✓

полезным Полученные результаты сводятся к следующему:

– Рекомпиляция ядра – самый гибкий метод, но самый сложный в развёртывании и отладке, а также требующий большего времени на разработку. Также, рекомпиляция ядра требует перезагрузки системы, что может быть нежелательно.

– LKM – второй по гибкости метод модификации ядра, однако, в отличие от рекомпиляции, не требует перезагрузки системы. Тем не менее LKM разделяет частично проблемы рекомпиляции ядра: медленная разработка, относительная сложность отладки, а также риск остановки системы при неправильной работе модуля.

[2]Несмотря на то, что в данном методе отладка происходит гораздо легче, чем при встраивании кода в само ядро, как при рекомпиляции или Live Patching'е, отладка модулей всё равно может вызвать ряд проблем.

– Live Patching – крайне ограниченный метод модификации ядра, который позволяет вносить изменения в ядро только в определённых случаях, чаще всего в виде исправления ошибок. Однако, он позволяет вносить изменения в уже загруженные функции ядра, без перезагрузки системы, что является большим плюсом.
– eBPF – самый простой и быстрый метод модификации ядра, однако, он ограничен в своих возможностях. Так например, eBPF не имеет возможности применять политики доступа к системным вызовам. Таким образом нельзя запретить пользователю открывать какой-либо файл, используя инструментарий eBPF. Взамен этому eBPF-программы предоставляет безопасный и высокопроизводительный способ выполнения кода на уровне ядра.

Из результатов сравнения было выявлено, что каждый из методов модификации ядра Linux различен относительно других методов по своим преимуществам и недостаткам. Таким образом, выбор метода модификации ядра Linux зависит в первую очередь от конкретных задач, которые необходимо решить, и нет единого метода, который был бы наилучшим для решения всех задач. Разработанная в ходе работы методика позволяет сравнивать методы модификации ядра Linux и определять, какой из методов подходит для решения конкретной задачи.

ЗАКЛЮЧЕНИЕ

В ходе данной работы были изучены:

- методы модификации ядра Linux;
- критерии сравнения методов модификации ядра;
- основные принципы работы и преимущества каждого из методов.

Был выполнен обзор существующих методов модификации ядра Linux, проведен анализ их преимуществ и недостатков. Были сформулированы критерии классификации методов модификации ядра Linux. Была проведена классификация методов модификации ядра Linux по критериям, сформулированным в ходе работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Определение API [Электронный ресурс]. Режим доступа: https://www.oxfordlearnersdictionaries.com/definition/english/api?q=API.

2 Salzman Peter Jay, Burian Michael, Pomerantz Ori [и др.]. The Linux Kernel Module Programming Guide. Режим доступа: https://sysprog21.github.io/lkmpg/.

3 Endorsed Linux distributions on Azure [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/en-us/azure/virtual-machines/linux/endorsed-distros.

4 Arm Developer. Virtual and physical addresses [Электронный ресурс]. Режим доступа: https://developer.arm.com/documentation/101811/0102/Virtual-and-physical-addresses.

5 Виртуальная память, Национальная библиотека им. Н. Э. Баумана [Электронный ресурс]. Режим доступа: https://ru.bmstu.wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C.

6 Кольца защиты. Рисунок [Электронный ресурс]. Режим доступа: https://upload.wikimedia.org/wikipedia/ru/2/2f/Priv_rings.svg.

7 What is RDS and why did we build it? [Электронный ресурс]. Режим доступа: https://oss.oracle.com/pipermail/rds-devel/2007-November/000228.html.

8 Adji Teguh Bharata, Nggilu Faisal Suryadi, Sumaryono Sujoko. Overhead Analysis as One Factor Scalibility of Private Cloud Computing for IAAS Service // International Journal of Scientific & Engineering Research, Volume 4, Issue 5. 2013.

9 API ядра Linux [Электронный ресурс]. Режим доступа: https://www.kernel.org/doc/html/latest/.

10 Loadable Kernel Module, Национальная библиотека им. Н. Э. Баумана [Электронный ресурс]. Режим доступа: https://ru.bmstu.wiki/LKM_(Loadable_Kernel_Module).

11 Arnold Jeff, Kaashoek M. Frans. Ksplice: Automatic Rebootless Kernel Updates.

12 Canonical, Kernel Livepatching [Электронный ресурс]. Режим доступа: https://canonical.com/blog/an-overview-of-live-kernel-patching.

13 Topics in live kernel patching [Электронный ресурс]. Режим доступа: https://lwn.net/Articles/706327/.

14 Linux Kernel Live Patching: What It Is and Who Needs It [Электронный ресурс]. Режим доступа: https://www.infosecurity-magazine.com/blogs/linux-kernel-live-patching/.

15 eBPF Explained: Use Cases, Concepts, and Architecture [Электронный ресурс]. Режим доступа: https://www.tigera.io/learn/guides/ebpf/.

16 CVE-2021-4204 [Электронный ресурс]. Режим доступа: https://nvd.nist.gov/vuln/detail/CVE-2021-4204.

ПРИЛОЖЕНИЕ А

Презентация к научно-исследовательской работе

Презентация содержит 13 слайдов.