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 **3**

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ **4**

ВВЕДЕНИЕ **5**

1 Анализ предметной области **6**

1.1 Способы модификации ядра Linux 6

1.1.1 Модификация ядра Linux 6

1.1.2 Задачи модификации ядра Linux 7

1.2 Базовые понятия и термины 8

2 Существующие решения **9**

2.1 Обзор методов модификации ядра Linux 9

2.1.1 Методы модификации ядра Linux, основанные на перекомпиляции ядра 9

2.1.2 Методы модификации ядра Linux, основанные на встраивании модулей ядра 10

2.1.3 Kernel Live Patching 13

2.1.4 eBPF 15

2.2 Критерии сравнения методов модификации ядра 17

2.3 Сравнение методов модификации ядра 18

ЗАКЛЮЧЕНИЕ **19**

ОПРЕДЕЛЕНИЯ

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

Application Programming Interface — «Интерфейс прикладного программирования» - набор инструментов программирования, который позволяет 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» - модуль ядра, который может быть загружен в операционную систему во время ее работы и может быть выгружен из нее в любое время. Они расширяют функциональность ядра без необходимости перезагрузки системы.[2]

Translation Lookaside Buffer — «Буфер ассоциативной трансляции» - кэш, используемый процессором для ускорения перевода виртуальных адресов в физические адреса. Он используется для сокращения времени, необходимого для доступа к ячейке пользовательской памяти.[3]

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

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

ОС — Операционная система

API — Application Programming Interface

LKM — Loadable Kernel Modules

TLB — Translation Lookaside Buffer

ВВЕДЕНИЕ

Расширяемость ядра долгое время была желаемым свойством различных операционных систем, так как это позволяет пользователям с различными потребностями совместно использовать ту же кодовую базу, что и сама ОС. Эта концепция со временем широко распространилась, и такие ОС как Linux были разработаны с возможностью легкого расширения ядра.

Сегодня различные расширения ядра обеспечивают критически важные функции для обеспечения безопасности и эффективности в облачных системах. К сожалению, безопасность ядра не является идеальной, и множество ошибок в ядре продолжают вызывать проблемы[4][5].

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 сравнения методов модификации ядра;
- Классифицировать существующие методы модификации ядра.

1 Анализ предметной области

1.1 Способы модификации ядра Linux

Прежде чем приступить к обзору и анализу существующих методов модификации ядра Linux, необходимо определить, что подразумевается под модификацией ядра Linux.

1.1.1 Модификация ядра Linux

В данном разделе описаны основные 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 любые части ядра Linux, не затрагивая саму структуру ядра. Любые изменения, которые включают в себя изменение структуры ядра Linux, называются **переписыванием ядра Linux**, что выходит за рамки данного исследования.

- Любой дополнительный функционал, который добавляется в ядро Linux, должен оперировать существующими структурами ядра Linux. Иными словами, здесь и далее, мы говорим, что новые функции должны работать на уровне ядра Linux, а не на уровне пользователя.

  Современные операционные системы, в частности Linux, разделяет виртуальную память на пространство пользователя и пространство ядра. В классическом представлении, существует 4 кольца защиты, однако их точное количество зависит от множества факторов, включая архитектуру процессора, архитектуру операционной системы и т.д. Схематично это представлено на рисунке 1.1.

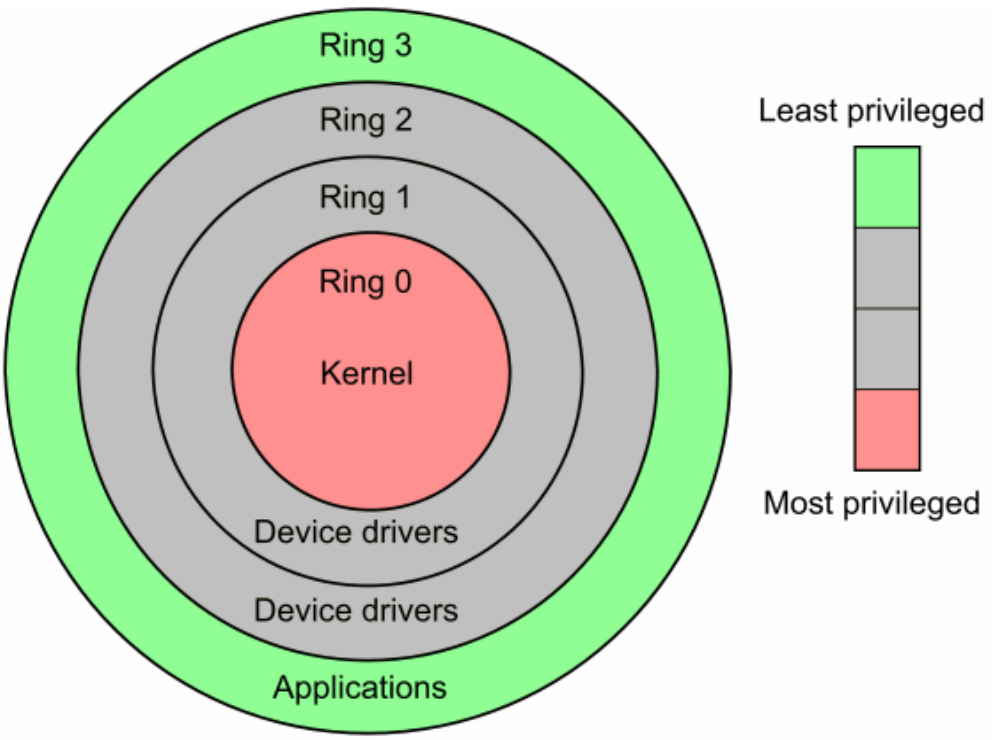

Рисунок 1.1 – Кольца защиты

– Главное различие между уровнем ядра и уровнем пользователя заключается в том, что в то время пока процессы на уровне пользователя оперируют ограниченными участками памяти, то процессы на уровне ядра имеют доступ ко всей памяти компьютера.

1.1.2 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 некоторые преимущества, однако, в большинстве случаев, они либо не являются необходимыми, либо недостатки таких подходов нивелируют эти преимущества. Общее правило, которое может быть сформировано для всех методов, 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 самой программы, на тему того, не существует 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 все остальные варианты были исчерпаны.
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, у которых должен быть доступ к низкоуровневым ресурсам, которые не могут быть предоставлены другими способами (например, прерывания).

2. Реализация алгоритмов, которые должны быть выполнены с высокой точностью по времени и/или пространству (например, мониторинг ресурсов системы или совместное использование ресурсов).

3. Написание программ, которые должны быть доступны всем пользователям системы (например, демонов).

4. Также следует перейти в пространство ядра, где накладные расходы, такие как смена пространств пользователь-ядро, становится неприемлемым для правильной работы написанного кода. Чаще всего в таких случаях мы говорим об облачных вычислениях.

1.2 Базовые понятия и термины

TODO: Добавить описание терминов и базовых понятий.
TODO: А это вообще нужно?

2 Существующие решения

2.1 Обзор методов модификации ядра Linux

В настоящее время существует множество методов модификации ядра Linux. Все они различаются по своим особенностям и применяемым технологиям. В данном разделе рассмотрены наиболее популярные методы модификации ядра Linux.

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, которое можно использовать для запуска 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 нового модуля в систему необходимо будет перекомпилировать все ядро Linux. Таким образом, этот метод модификации ядра Linux не подходит для систем, в которых необходимо добавлять новые модули в систему 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)[6]. Таким образом, данный метод модификации ядра Linux не подходит для людей, которые не имеют достаточной квалификации связанной с языком C или достаточных знаний связанных с Linux API. Иными словами данный подход требует крайне высокой квалификации и опыта программиста, что

замедляет процесс разработки и внедрения новых модулей в систему.

3. Модифицированное ядро Linux не является частью исходного кода ядра Linux.

   Поэтому, если в исходный код ядра Linux вносятся изменения, то исходный код модифицированного ядра Linux должен быть изменен вручную. (???)

4. Если допустить ошибку при добавлении нового кода в ядро Linux, то есть шанс его повредить и сделать систему полностью нерабочей. При возникновении ошибки в ядре, то вся система также критично завершает работу.

Однако у статического метода модификации ядра Linux есть и **преимущества**:

1. Скорость работы системы. В статическом методе модификации ядра Linux не используется динамическая подгрузка 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 позволяет увеличить 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. (???)

2.1.2 Методы модификации ядра Linux, основанные на встраивании модулей ядра

Второй метод модификации ядра Linux, основанный на встраивании модулей ядра, появился в 1995 году, когда в версию 1.2 была добавлена поддержка

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.

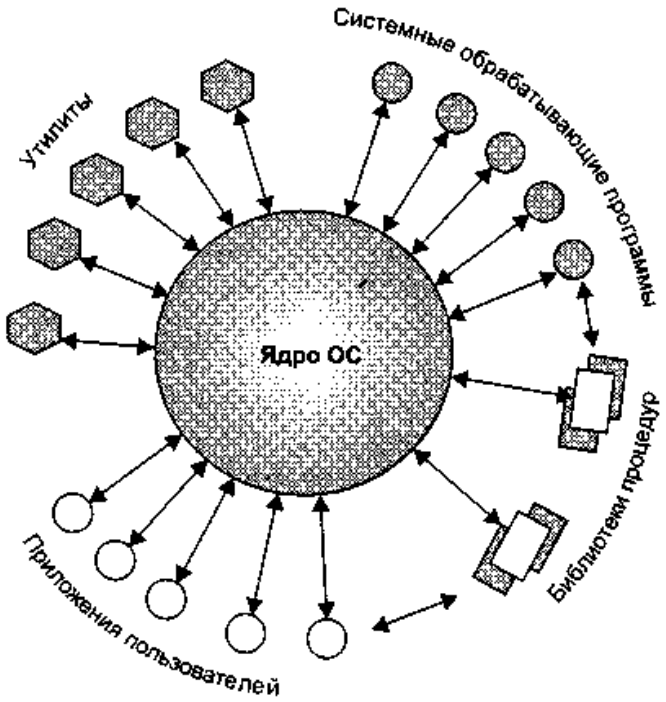

Рисунок 2.1 – Взаимодействие ядра с загружаемыми модулями и пользовательскими приложениями.

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, которые 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, модуль ядра может быть загружен в ядро Linux во время работы системы.

2. Модифицированное ядро Linux является частью исходного кода ядра Linux.

   Поэтому, если в исходный код ядра Linux вносятся изменения, то исходный код модифицированного ядра Linux автоматически изменяется.

3. Модуль ядра может сэкономить оперативную память, потому что появляется возможность загружать их только тогда, когда они действительно нужны, в то время как все части базового ядра все время остаются загруженными в реальном хранилище, а не только в виртуальном.

4. Еще одним преимуществом LKM является то, что он помогают диагностировать системные проблемы.

   Ошибка в драйвере устройства, связанном с ядром, может остановить загрузку системы, что может вызвать проблемы определить, какая часть базового ядра вызвала проблемы. Однако если тот же драйвер устройства является LKM, базовое ядро запускается и работает еще до загрузки драйвера устройства. Если система завершает работу после того, как базовое ядро запущено, то появляется возможность отследить проблему до вызывающего проблемы драйвера устройства и не загружать его до тех пока, проблема не будет решена.

Недостатки:

1. LKM может привести к проблемам с производительностью системы.

   Поскольку LKM загружаются в ядро Linux во время работы системы, они могут увеличить время загрузки системы.

2. В некоторых случаях LKM может привести к некоторым проблемам.

   Например, LKM может привести к проблемам совместимости, если они не совместимы с базовым ядром.

3. Еще одним из незначительных критических замечаний по поводу предпочтения модульного ядра статическому ядру является так называемый штраф за фрагментацию.

   Базовое ядро всегда распаковывается в реальную непрерывную память своими программами установки; таким образом, базовый код ядра никогда не фрагментируется. Как только система находится в состоянии, в котором модули могут быть вставлены, например, после того, как были смонтированы файловые системы, содержащие модули, вполне вероятно, что любая новая вставка кода ядра приведет к фрагментации ядра, что приведет к незначительному снижению производительности, используя больше записей TLB, что приводит к большему количеству промахов TLB.

4. Также 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 относительно новая функция ядра Linux, которая позволяет обновлять ядро Linux без перезагрузки системы. Исправление ядра в реальном времени является важным компонентом стратегии управления серверами Linux и устранения уязвимостей.
Что Live Patching делает, так это создают модуль ядра из исправленного кода, а затем с помощью инструмента ftrace (трассировка функции) перенаправляют от устаревшей функции к новой.

Преимущества:

1. Позволяет обновлять ядро Linux без перезагрузки системы.

2. Обновление системы требует времени и высокого уровня навыков системного администрирования. Live Patching избавляет персонал от рутинной работы по рутинному обслуживанию многочисленных серверов.

3. Обновления проходят достаточно быстро, что позволяет оперативное разворачивание новых функций.

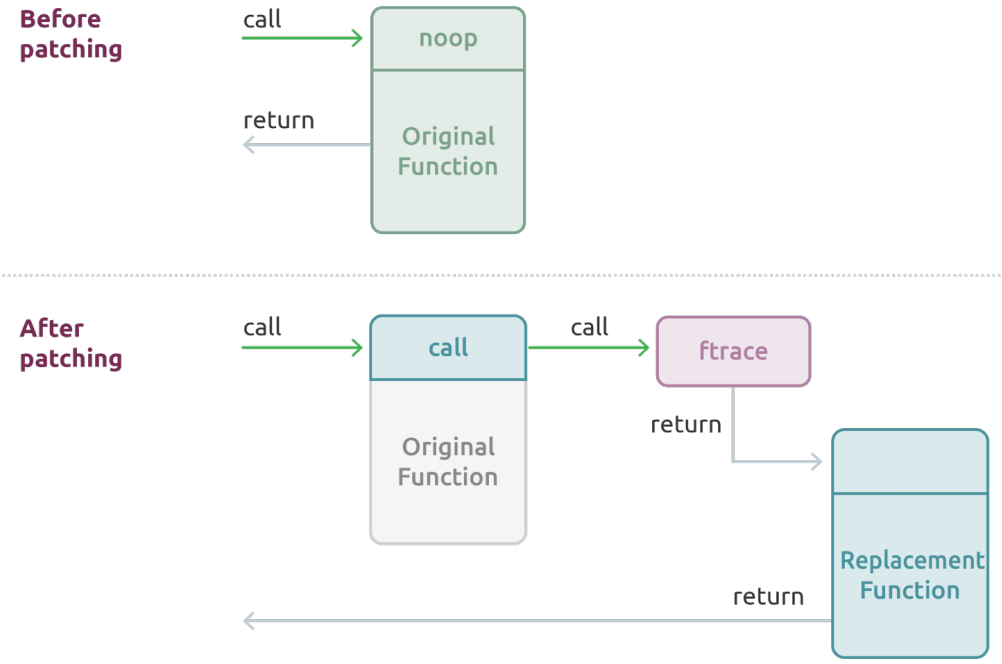

Рисунок 2.2 – Концепт Live Patching

Недостатки:

1. Внесение исправлений в ядро по-прежнему довольно 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.

2. Live Patching может применяться только к небольшим и конкретным частям кода ядра и не может использоваться для каких-либо 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 все изменения.

3. Не все ядра поддерживают Live Patching. В различных ядрах используются разные методы управления процессом исправления и создания исправле-

ний, а некоторые из них разработаны исключительно под определенные семейства Linux.[7]

2.1.4 eBPF

eBPF (extended Berkeley Packet Filter) - это новая технология, которая позволяет встраивать программы в ядро Linux, не изменяя его исходный код. В своей основе eBPF использует привилегированную способность ядра видеть и контролировать все ресурсы системы. С помощью eBPF есть возможность запускать изолированные программы в привилегированном контексте, таком как ядро операционной системы.

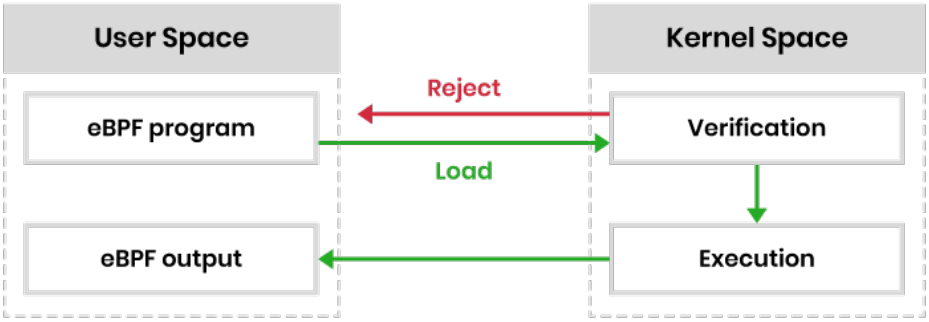

Рисунок 2.3 – Алгоритм eBPF

Преимущества:

1. eBPF позволяет встраивать программы в ядро Linux, не изменяя его исходный код.

2. Поскольку программа eBPF не изменяет код ядра, появляется возможность сохранить правила управления доступом для изменений на уровне кода. Кроме того, программы написанные с помощью eBPF имеют этап проверки, который предотвращает чрезмерное использование ресурсов.

3. eBPF может перенести обработку пакетов из пространства ядра в пространство пользователя.

Иными словами eBPF является JIT-компилятором. После компиляции байт-кода вызывается eBPF, а не новая интерпретация байт-кода для каждого метода.

Недостатки:

1. Главный минус eBPF заключается в том, что количество программ в песочнице ограничено.

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

2. В случае если необходимо внести изменения в ядро, то eBPF просто не обладает данной возможностью.

3. Поскольку eBPF выполняет программы в привилегированном контексте, то это может привести к утечке конфиденциальной информации.

   В конце данного раздела стоит сделать оговорку касательно eBPF.

В реальности eBPF **не существует**, или если быть точнее, не является полноценным инструментом модификации ядра де-факто.

eBPF — это всего лишь 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 лишь наборы инструкций, которые позволяют расширить возможности уже существующего ядра. Но не смотря на это, по формальным признакам, eBPF подходит под все критерии, которые были описаны в начале данной работы, являясь при этом крайне мощным 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 2.1 – Критерии сравнения методов модификации ядра

Критерий	**Описание**
Производительность	Производительность программ.
Безопасность	Наличие гарантии, что внесенный код не вызовет остановку системы.
Скорость разработки	Скорость разработки модификации.
Скорость загрузки	Скорость загрузки модификаций при запуске системы.
Гибкость	Возможность метода подстроиться под любые поставленные бизнес-задачи.
Простота отладки	Является ли написанные модификации простыми в отладке.
Поддержка	Поддержка метода разработчиками ядра при его написании.
Простота развёртывания	Насколько сложно развёртывать модификации на большом количестве машин.

2.3 Сравнение методов модификации ядра

TODO: добавить комментарии к сравнению

Таблица 2.2 – Сравнение методов модификации ядра

Критерий	Рекомпиляция	LKM	Live Patching	eBPF
Производительность	✓	✓	✓	✓
Безопасность	✗	✗	✗	✓
Скорость разработки	✗	✓	✗	✓
Скорость загрузки	✓	✓	✓	✓
Гибкость	✓	✓	✗	✗
Простота отладки	✗	✓/✗[1]	✗	✓
Поддержка	✓	✓	✓	✗
Простота развёртывания	✗	✓	✓	✓

[1]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.

ЗАКЛЮЧЕНИЕ

TODO: *Заключение* В ходе работы были изучены методы модификации ядра Linux. Были изучены основные принципы работы и преимущества каждого из методов. Были изучены критерии сравнения методов модификации ядра. Было проведено сравнение методов модификации ядра. Были сделаны выводы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Определение API [Электронный ресурс]. — Режим доступа: https://www.oxfordlearnersdictionaries.com/definition/english/api?q=API.
2. The Linux Kernel Module Programming Guide / P. J. Salzman [и др.]. — Режим доступа: https://sysprog21.github.io/lkmpg/.
3. Определение TLB [Электронный ресурс]. — Режим доступа: https://pages.cs.wisc.edu/~remzi/OSTEP/vm-tlbs.pdf.
4. Количество багов ядра (по модулям) [Электронный ресурс]. — Режим доступа: https://bugzilla.kernel.org/report.cgi?y_axis_field=component&cumulate=0&z_axis_field=&format=pie&x_axis_field=&no_redirect=1&query_format=report-graph&short_desc_type=allwordssubstr&short_desc=&bug_status=NEW&bug_status=ASSIGNED&bug_status=REOPENED&longdesc_type=allwordssubstr&longdesc=&bug_file_loc_type=allwordssubstr&bug_file_loc=&keywords_type=allwords&keywords=&cf_kernel_version_type=allwordssubstr&cf_kernel_version=&deadlinefrom=&deadlineto=&bug_id=&bug_id_type=anyexact&emailassigned_to1=1&emailtype1=substring&email1=&emailassigned_to2=1&emailreporter2=1&emailcc2=1&emailtype2=substring&email2=&emailtype3=substring&email3=&chfieldvalue=&chfieldfrom=&chfieldto=Now&j_top=AND&f1=noop&o1=noop&v1=&action=wrap.
5. Количество багов ядра (по версии ядра) [Электронный ресурс]. — Режим доступа: https://bugzilla.kernel.org/report.cgi?bug_status=NEW&bug_status=ASSIGNED&bug_status=REOPENED&cumulate=0&y_axis_field=cf_kernel_version&width=1024&height=600&action=wrap&format=table.
6. API ядра Linux [Электронный ресурс]. — Режим доступа: https://www.kernel.org/doc/html/latest/.
7. 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/.