# Оглавление

# Введение

TODO

# 1 Анализ предметной области

В данном разделе будут введены основные определения и будет обоснована актуальность задачи внесения изменений в ядро операционной системы Linux.

## 1.1 Актуальность

Программное обеспечение с открытым исходным кодом имеет такие преимущества, как скорость разработки и надежность. Причиной этого является большое число участников разработки. Каждый разработчик находит ошибки, предсавляет их решение и предлагает новый функционал. Так, осуществляется непрерывный процесс внесения изменений.

Ядро Linux - программное обеспечение с открытым исходным кодом, поэтому непрерывно разрабатывается специалистами со всего мира. Добавление новых функций, внесение усовершенствований, исправление ошибок делают актуальной проблему внесения изменений в ядро операционной системы Linux.

## 1.2 Основные определения

Ядро операционной системы - это программное обеспечение, которое предоставляет базовые функции для всех остальных частей операционной системы, управляет аппаратным обеспечением и распределяет системные ресурсы [1].

Одной из характеристик ядра Linux является динамическая загрузка модулей ядра. Другими словами, при необходимости существует возможность динамической загрузки и выгрузки исполняемого кода во время работы системы. Так, можно переопределять или дополнять функции ядра. Файлы с исправлениями встраиваются в виде загружаемых модулей ядра.

Технику внесения изменений в код или данные, позволяющую модифицировать поведение целевого алгоритма требуемым образом, называют патчингом [2]. Патчи - это небольшие добавочные изменения, вносимые в ядро.

Каждый патч содержит изменение ядра, реализующее одну независимую модификацию.

При применении патча к ядру операционной системы необходимо знать состояние функций ядра, используется ли функция в момент исправления или нет. Для этого используется метод обхода стека.

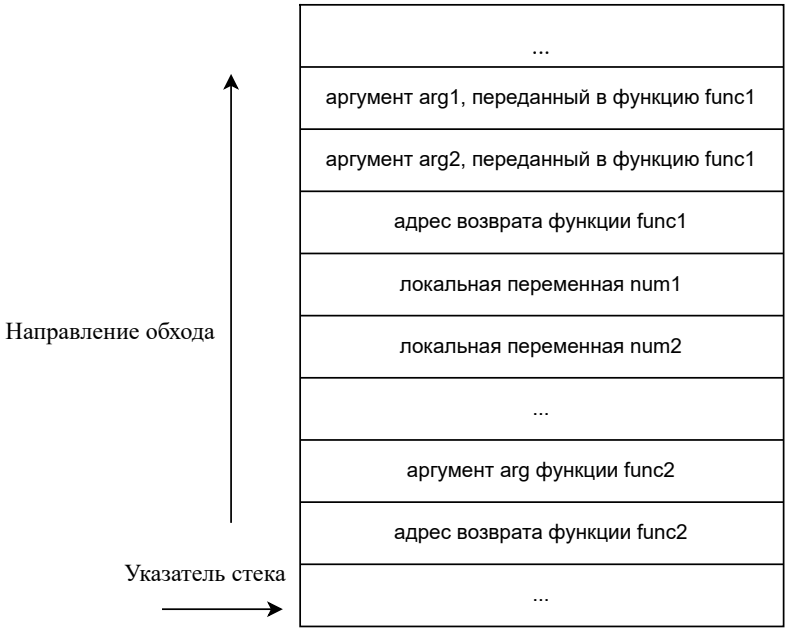


Рис. 1.1: Метод обхода стека

В методе обхода стека проверяется стек вызовов каждого потока ядра, показанный на рисунке 1.1. Необходимо определить, выполняется ли поток в функции, которую необходимо изменить. Для этого копия указателя стека уменьшается пока значение не достигнет нижней части стека. Функция используется, если адрес, принадлежащий этой функции можно найти в стеке.

## 1.3 Вывод

Были изучены основные определения и была обоснована важность проблемы изменения ядра операционной системы Linux.

# 2 Классификация существующих методов решения

В данном разделе будут описаны существующие методы решения, выделены критерии их оценки, и будет проведено сравнение описанных методов по выделенным критериям.

## 2.1 Методы решения

Существуют следующие методы изменения ядра Linux:

- требующие перезагрузки системы;
- переноса;
- динамические.

### 2.1.1 Методы, требующие перезагрузки системы

Первые применения патчей к ядру происходили по следующему алгоритму [3]:

- работающие приложения закрываются;
- происходит загрузка и инициализация исправленного ядра;
- приложения перезагружаются.

Так, неисправленное ядро заменялось исправленным ядром.

В процессе внесения изменений в ядро операционной системы описанным способом возникает следующая проблема: появляется время простоя, которое состоит из времени простоя приложения и простоя ядра, что показано на рисунке 2.1.


Рис. 2.1: Время простоя при перезагрузке системы

Для снижения времени простоя появились модификации метода, которые эффективно управляют перезагрузкой ядра. Одно из решений - метод контрольной точки [4].

В данном методе используется системный вызов *kexec* [5] для загрузки нового образа ядра. Этот механизм позволяет загружать новое ядро из работающего в данный момент ядра в основную память и сразу же начинает его выполнение.

Чтобы применить патч необходимо дождаться момента, когда система примет состояние, в котором выполнены два условия:

- все потоки ядра остановлены;
- структуры данных ядра согласованны.

Выполнение этих двух условий позволяет сделать контрольную точку, которая позволяет сохранить состояние приложений. Контрольная точка сохраняет состояния процессов, состоящих из их пространства памяти (разделы кода или данных, стека или кучи) и их внутренние состояния в ядре. Код контрольной точки проходит через структуры данных ядра, связанные с приложениями, и преобразует их в высокоуровневый формат, который не зависит от версии ядра.

После сохранения контрольной точки выполняется инициализация нового ядра. Исправленное ядро считывает контрольную точку и восстанавливает приложения, а затем перезапускает их.

Техника этого метода показана на рисунке 2.2.

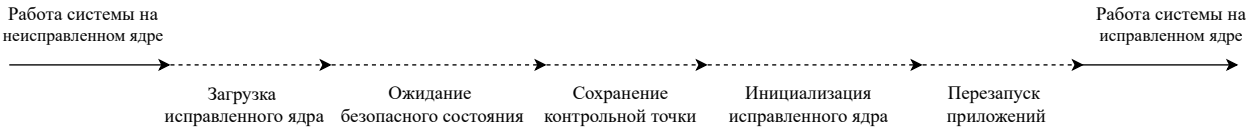


Рис. 2.2: Метод контрольной точки

Существует метод, который сокращает время простоя путем одновременного выполнения приложений и перезагрузки системы - метод теневой перезагрузки [6]. Перезагрузка операционной системы выполняется в фоновом режиме на виртуальной машине. Приложения могут продолжать выполняться на исходной машине.

После перезагрузки ядра на выделенной виртуальной машине, делается снимок системы, из которого восстанавливается файловая система на исходной машине.

Во время теневой перезагрузки пользовательские приложения могут изменять файлы исходной машины. Файлы могут быть изменены и на выделенной виртуальной машине. Так как файловая система восстанавливается из снимка, то изменения файлов на исходной машине теряются. То есть, файловая система откатывается до состояния, когда создавалась выделенная виртуальная машина.

Для согласованности файлов в методе теневой перезагрузки вводят понятие срока перезагрузки, в течение которого пользователи могут изменять файлы каталогов. Срок перезагрузки - это период, который начинается с создания выделенной виртуальной машины и завершается после создания снимка выделенной виртуальной машины. При восстановлении файловой системы сохраняются файлы, измененные во время срока перезагрузки на исходной машине, и добавляются другие файлы выделенной виртуальной машины.

Описанный метод представлен на рисунке 2.3.


Рис. 2.3: Метод теневой перезагрузки

Плюсом данных методов является отсутсвие необходимости дополнительных машин или общих хранилищ.

Метод теневой перезагрузки приводит к простою приложения, а метод контрольной точки восстановление процесса не может выполняться до тех пор, пока не завершится перезапуск ядра операционной системы, что приведет к простою ядра. Перезагрузка может привести к потере доступности критических задач или процессов, работающих в этой операционной системе.

Следующие методы решают проблему простоя исключением перезагрузки системы.

### 2.1.2 Метод переноса

Идея данного метода [7] заключается в следующем: на дополнительной машине запускается измененное ядро, на него переносятся запущенные процессы старого ядра, и оно останавливается. Так как использование дополнительной физической машины ресурсозатратно, в существующих решениях [8] в качестве допольнительной машины используется виртуальная машина, установленная на физической машине, требующей обновления ядра. Необходимо общее хранилище (сервер), которое подключено и к старому, и к новому ядрам. Патч применяется в три этапа.

На первом этапе собирается информация о состоянии операционной си-

стемы: подсчитывается число потоков, выполняемых в исправляемом коде, вызывается функция запуска, выполняется инициализация перед передачей управления виртуальной машине.

На втором этапе начинается исправление структур данных и функций. Если не измененный модуль используется, обе версии структур данных должны существовать во время процесса исправления. Для обеспечения согласованности структур данных, страницы исходных данных и новых данных защищены. Перехват доступа к ним контролируется виртуальной машиной. То есть, при попытке изменить отслеживаемую страницу управление будет передано виртуальной машине. В этот момент сравнивается содержимое двух версий и вызывается функция передачи состояния, как показано на рисунке 2.4.

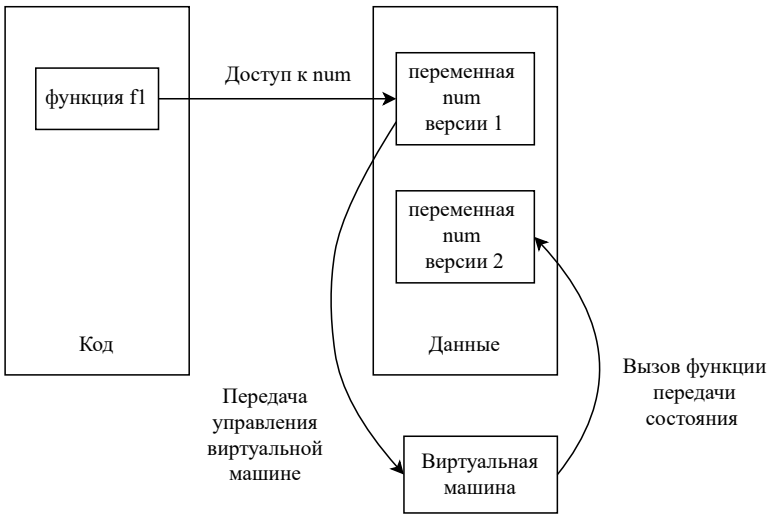


Рис. 2.4: Согласованность данных в методе переноса

На последнем этапе отключается контроль исходных структур данных. Для этого используется метод обхода стека. В случае, если соответствующая исходные структуры используются, адрес возврата исходной функции заменяется адресом функции-заглушки, которые позволяют определить, используется ли еще обновленная функция. Затем выполняется очистка кода старой версии и устанавливается флаг завершения патчинга.

При применении данного метода время простоя невелико. Главным минусом является высокое потребление ресурсов центрального процессора, сети

и объема памяти.

### 2.1.3 Динамические методы

Данные методы [9] [10] позволяют применять патчи во время выполнения процессов без перезагрузки и дополнительных ресурсов. Решение состоит из двух этапов, показанных на рисунке 2.5.

Для того, чтобы создать измененный код, проводится анализ обновленной и старой версий. Для этого собирается два варианта ядра: сборка исходного кода и сборка измененного кода. Файлы, полученные сборкой неизмененного кода называют предварительными объектными файлами, файлы, полученные в результате сборки кода патча - последующими объектными файлами. В отличие от поиска различий в исходном коде, анализ объектного кода позволяет понять, какие функции были изменены в патче. Большинство функций ядра, которые не были изменены патчем, будут иметь одинаковые объектные коды. Обнаруженные измененные функции помещаются в основной модуль для загрузки в ядро.


Рис. 2.5: Динамический метод

Следующий этап необходим для обнаружения встраиваемых функций и неуникальных символов. Для этого проводится сравнение работающего кода ядра и скомпилированного кода. Этот процесс называется предварительным сопоставлением.

Для применения данного метода необходимо определить состояние ядра, когда каждая заменяемая функция будет находиться в состоянии покоя, что является условием безопасного внесения изменений. В этом случае модуль может быть загружен в ядро. Проверка условия безопасного внесения изменений в случае неудачи возобновится через некоторое время. После нескольких неудачных попыток достичь безопасного состояния для применения патча процесс прерывается. Новые инструкции будут вставлены в необновленные функции путем бинарной перезаписи. Бинарная перезапись - это перенаправление вызова функции из исходной в исправленную, для чего используется прыжок к первым пяти или шести байтам функции.

Данные методы решают проблему с временем простоя.

Решения на основе динамических методов не поддерживают семантические изменения. Кроме того, возникают сложности с изменением типов и структур данных, нестабильных типов данных и функций ядра, которые всегда находятся в стеке вызовов потоков ядра.

## 2.2 Критерии оценки методов

Сравнение описанных методов изменения ядра Linux будет проводиться по следующим критериям:

- необходимость перезагрузки;
- наличие времени простоя;
- возможность семантических изменений;
- потребление ресурсов.

## 2.3 Сравнение методов

Обозначим введенные критерии оценки методов следующим образом:

- К1 - необходимость перезагрузки системы;
- К2 - наличие времени простоя;
- К3 - возможность семантических изменений;
- К4 - потребление ресурсов.

Результаты сравнения методов изменения ядра Linux представлены в таблице 2.1.

Таблица 2.1: Сравнение методов изменения ядра Linux

| Метод | К1 | К2 | К3 | К4 |
|---|---|---|---|---|
| Контрольной точки | необходима | есть | есть | 100 % |
| Теневой перезагрузки | необходима | есть | есть | 100 % |
| Переноса | отсутствует | есть | есть | 200 % |
| Динамический | отсутствует | нет | нет | 100 % |

По результатам сравнения самым эффективным методом внесения изменений в ядро Linux оказался динамический метод. Главными недостатками методов контрольной точки и теневой перзагрузки являются необходимость перезагрузки системы и наличие времени простоя. В методе переноса перезагрузка системы не требуется и время простоя снижено, но потребление ресурсов увеличивается в два раза. Кроме того методы, требующие перезагрузки системы, и метод переноса восполняют недостаток динамического метода - отсутствие возможности внесения семантических изменений.

## 2.4 Вывод

В данном разделе были рассмотрены существующие методы решения, выделены критерии их оценки, а также было проведено сравнение описанных методов по выделенным критериям.

# Выводы

TODO

# Литература


[1] Love Robert. Linux System Programming. 2013.

[2] Overview of Dynamic Operating System's Kernel Hooking Methods (Study Case of Linux Kernel). Режим доступа: https://www.flow3d.com/modeling-capabilities/waves/ (дата обращения: 04.10.2021).

[3] Terada Ken, Yamada Hiroshi. Shortening Downtime of Reboot-Based Kernel Updates Using Dwarf // IEICE Transactions on Information and Systems. 2018. 12. С. 2991–3004.

[4] Instant OS Updates via Userspace Checkpoint-and-Restart / Sanidhya Kashyap, Changwoo Min, Byoungyoung Lee [и др.] // USENIX Annual Technical Conference (USENIX ATC 16). 2016. 06. С. 605–619.

[5] Siniavine Maxim, Goel Ashvin. Seamless kernel updates // Proceedings of the International Conference on Dependable Systems and Networks. 2013. 06. С. 1–12.

[6] Yamada Hiroshi, Kono Kenji. Traveling Forward in Time to Newer Operating Systems using ShadowReboot // ACM SIGPLAN Notices. 2011. 07.

[7] Potter Shaya, Nieh Jason. Reducing downtime due to system maintenance and upgrades. 2005. 01. С. 6–6.

[8] Live updating operating systems using virtualization / Haibo Chen, Rong Chen, Fengzhe Zhang [и др.] // VEE 2006 - Proceedings of the Second International Conference on Virtual Execution Environments. 2006. 06. С. 35–44.

[9] Kaashoek M., Arnold Jeffrey. Ksplice: Automatic Rebootless Kernel Updates // Frans Kaashoek. 2009. 01.

[10] Makris Kristis, Ryu Kyung. Dynamic and adaptive updates of non-quiescent subsystems in commodity operating system kernels // Operating Systems Review - SIGOPS. 2007. 06. С. 327–340.