**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 «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

# РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

## *К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ*

## *НА ТЕМУ:*

## *«Методы внесения изменений в ядро Linux»*

Студент ИУ7-53Б _____ **Р. Р. Хамзина**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель _____ **А. А. Оленев**
(Подпись, дата) (И.О.Фамилия)

*2021 г.*

**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)**

УТВЕРЖДАЮ

Заведующий кафедрой ИУ-7
_____ И. В. Рудаков
« _____ » _____ 20 ____ г.

# З А Д А Н И Е
## на выполнение научно-исследовательской работы

по теме Методы внесения изменений в ядро Linux

Студент группы ИУ7-53Б

Хамзина Регина Ренатовна
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

учебная

Источник тематики (кафедра, предприятие, НИР) НИР

График выполнения НИР: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

***Техническое задание***

Классифицировать методы внесения изменений в ядро Linux. В ходе выполнения научно-исследовательской работы необходимо выделить критерии сравнения методов. На основании выделенных критериев провести сравнение методов.

***Оформление научно-исследовательской работы:***

Расчетно-пояснительная записка на 15-25 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
Презентация на 8-10 слайдах.

Дата выдачи задания « ___ » _____ 20__ г.

**Руководитель НИР** _____ Р. Р. Хамзина
(Подпись, дата) (И.О.Фамилия)

**Студент** _____ А. А. Оленев
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

# Содержание

# Введение

В жизненный цикл программного обеспечения входит его техническая поддержка. В рамках технической поддержки проводится исправление обнаруженных за время существования программы ошибок, изменение реализованных функций, расширение возможностей программного обеспечения, например, добавление поддержки отдельных библиотек или инструментов. Описанные действия требуют внесения изменений в программное обеспечение. Один из элементов операционной системы, в который вносятся изменения — ядро операционной системы.

Целью данной научно-исследовательской работы является классификация методов внесения изменений в ядро Linux.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- классифицировать существующие методы внесения изменений в ядро Linux;
- выделить критерии оценки методов;
- провести сравнение методов на основании выделенных критериев;
- отразить результаты сравнения в выводе.

# 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 [1]. Причиной этого является большое число участников разработки. Каждый разработчик находит ошибки, представляет 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 [2]. 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 [3].

Одной из характеристик ядра Linux является динамическая загрузка модулей ядра [4]. 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 [5]. 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 [6].

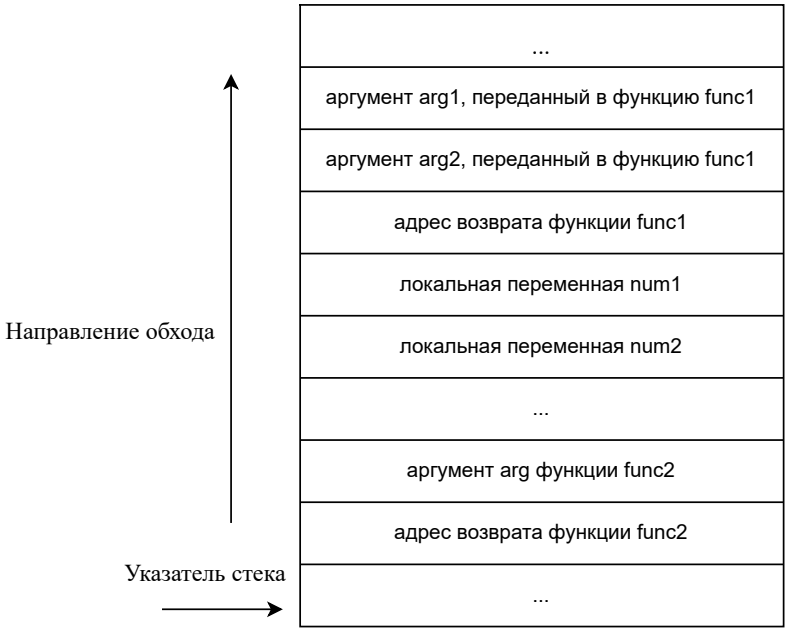


Рисунок 1.1 – Метод обхода стека

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 [7]:

- 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 [8]: появляется время простоя, которое состоит из времени простоя приложения и простоя ядра, что показано на рисунке 2.1.

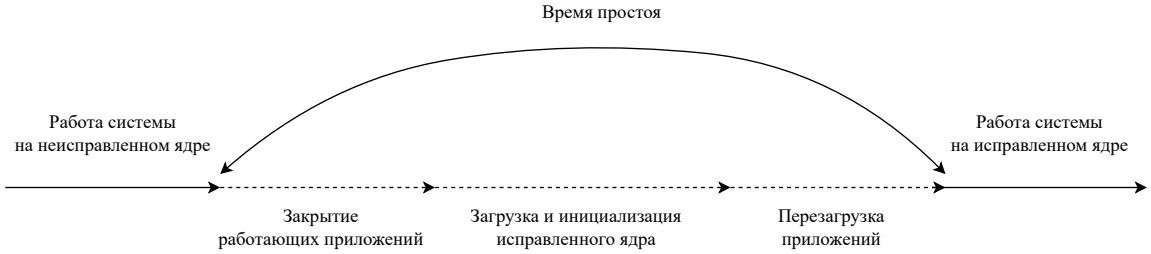


Рисунок 2.1 – Время простоя при перезагрузке системы

Для снижения времени простоя появились модификации метода, которые эффективно управляют перезагрузкой ядра. Одно из решений — метод контрольной точки [9].

В данном методе используется системный вызов *kexec* [8] 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 [10]. 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.

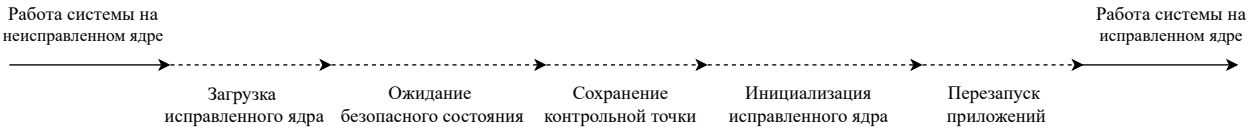


Рисунок 2.2 – Метод контрольной точки

Существует метод, который сокращает время простоя путем одновременного выполнения приложений и перезагрузки системы — метод теневой перезагрузки [11]. 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 [12] 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.


Рисунок 2.3 – Метод теневой перезагрузки

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

Метод теневой перезагрузки приводит к простою приложения, а в методе 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 [13] 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 [6] в качестве дополнительной 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 неизмененный 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.

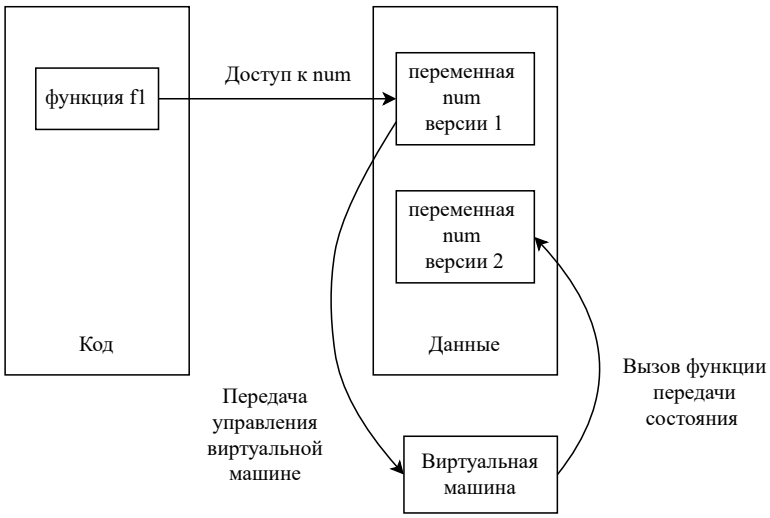


Рисунок 2.4 – Согласованность данных в методе переноса

На последнем этапе отключается контроль исходных структур данных. Для этого используется метод обхода стека [6]. В случае, если соответствующие исходные структуры используются, адрес возврата исходной функции заменяется адресом функции-заглушки, которая позволяет 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 [14] 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 [15]. 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.


Рисунок 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 [16] — 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, например, добавление поля в структуру данных. Кроме того, возникают сложности с изменением 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;
- число машин, необходимых для применения патча.

## 2.3 Сравнение методов

Обозначим введенные критерии оценки методов следующим образом:

- К1 — необходимость перезагрузки системы;
- К2 — наличие времени простоя;
- К3 — возможность семантических изменений;
- К4 — число машин, необходимых для применения патча.

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 | есть | есть | есть | 1 |
| Теневой перезагрузки | есть | есть | есть | 1 |
| Переноса | нет | есть | есть | 2 |
| Динамический | нет | нет | нет | 1 |

По результатам сравнения самым эффективным методом внесения изменений в ядро Linux оказался динамический метод. Главными недостатками методов контрольной точки и теневой перезагрузки 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.

# Заключение

В ходе выполнения данной работы были классифицированы методы внесения изменений в ядро операционной системы Linux.

На основании результатов сравнения методов можно сделать вывод о том, что наиболее эффективным методом является динамический метод, так как не требует перезагрузки системы, решает проблему появления времени простоя и не является ресурсозатратным.

Цель, поставленная в начале работы была достигнута. В ходе ее выполнения были решены следующие задачи:

- были классифицированы существующие методы внесения изменений в ядро Linux;
- были выделены критерии оценки изученных методов;
- было проведено сравнение методов на основании выделенных критериев;
- были отражены результаты сравнения в выводе.

# Литература


[1] Alami Adam. The sustainability of quality in free and open source software // New York, Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings. 2020. 06. С. 222–225. URL: https://www.researchgate.net/publication/346043137_The_sustainability_of_quality_in_free_and_open_source_software.

[2] Linux Kernel History Report 2020 / The Linux Foundation. San Francisco: The Linux Foundation, 2020. 08. URL: https://www.linuxfoundation.org/tools/linux-kernel-history-report-2020/.

[3] Fox Richard. Linux with Operating System Concepts. London: Chapman and Hall/CRC, 2021. 11. С. 1–620. URL: https://www.researchgate.net/publication/355948100_Linux_with_Operating_System_Concepts.

[4] Zhen Yanjie, Zhang Wei. Is It Possible to Automatically Port Kernel Modules? // New York, APSys '18: Proceedings of the 9th Asia-Pacific Workshop on Systems. 2018. 08. С. 1–8. URL: https://www.researchgate.net/publication/327811028_Is_It_Possible_to_Automatically_Port_Kernel_Modules.

[5] Koyuncu Anil, Bissyandé Tegawendé. Impact of Tool Support in Patch Construction // New York, roceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2017. 07. с. 237–248. URL: https://dl.acm.org/doi/10.1145/3092703.3092713.

[6] Chen Haibo, Chen Rong, Zhang Fengzhe. Live updating operating systems using virtualization // Ottawa, Proceedings of the Second International Conference on Virtual Execution Environments. 2006. 06. С. 35–44. URL: https://www.researchgate.net/publication/221137837_Live_updating_operating_systems_using_virtualization.

[7] Terada Ken, Yamada Hiroshi. Shortening Downtime of Reboot-Based Kernel Updates Using Dwarf // Tokyo, IEICE Transactions on Information and Systems. 2018. 12. С. 2991–3004. URL: https://www.researchgate.net/publication/329349663_Shortening_Downtime_of_Reboot-Based_Kernel_Updates_Using_Dwarf.

[8] Siniavine Maxim, Goel Ashvin. Seamless kernel updates // Budapest, Proceedings of the International Conference on Dependable Systems and Networks. 2013. 06. C. 1–12. URL: https://www.researchgate.net/publication/261452974_Seamless_kernel_updates.

[9] Sanidhya Kashyap Changwoo Min Byoungyoung Lee. Instant OS Updates via Userspace Checkpoint-and-Restart // Denver, USENIX Annual Technical Conference (USENIX ATC 16). 2016. 06. C. 605–619. URL: https://www.usenix.org/conference/atc16/technical-sessions/presentation/kashyap.

[10] Begunkov Pavel. Checkpoint and Restore of File Locks in Userspace // New York, CEE-SECR '17: Proceedings of the 13th Central and Eastern European Software Engineering Conference in Russia. 2017. 10. C. 1–4. URL: https://www.researchgate.net/publication/322260073_Checkpoint_and_restore_of_file_locks_in_userspace.

[11] Yamada Hiroshi, Kono Kenji. Traveling Forward in Time to Newer Operating Systems using ShadowReboot // New York, ACM SIGPLAN Notices. 2013. 03. C. 121–130. URL: https://dl.acm.org/doi/10.1145/2451512.2451536.

[12] Lagar-Cavilla H. Andrés, Whitney. SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing // New York, Proceedings of the 4th ACM European Conference on Computer Systems, EuroSys'09. 2009. 01. C. 1–12. URL: https://www.researchgate.net/publication/221351721_SnowFlock_Rapid_Virtual_Machine_Cloning_for_Cloud_Computing.

[13] Potter Shaya, Nieh Jason. Reducing downtime due to system maintenance and upgrades // San Diego, LISA '05: Proceedings of the 19th conference on Large Installation System Administration Conference. 2005. 12. C. 6–6. URL: https://dl.acm.org/doi/10.5555/1251150.1251156.

[14] Makris Kristis, Ryu Kyung. Dynamic and adaptive updates of non-quiescent subsystems in commodity operating system kernels // Lisbon, Operating Systems Review - SIGOPS. 2007. 06. C. 327–340. URL: https://www.researchgate.net/publication/221351757_Dynamic_and_adaptive_updates_of_non-quiescent_subsystems_in_commodity_operating_system_kernels.

[15] Kaashoek M., Arnold Jeffrey. Ksplice: Automatic Rebootless Kernel Updates // Nuremberg, EuroSys '09: Fourth EuroSys Conference 2009. 2009. 04. C. 187–198. URL: https://dl.acm.org/doi/10.1145/1519065.1519085.

[16] Arras Paul-Antoine, Andronidis. SaBRe: load-time selective binary rewriting // Berlin, International Journal on Software Tools for Technology Transfer. 2022. 01. C. 829–832. URL: https://www.researchgate.net/publication/358018186_SaBRe_load-time_selective_binary_rewriting.