DOI: 10.15514/ISPRAS-2026-38(4)-21

Методика сравнения и анализа безопасности подсистем обработки прерываний в ядрах операционных систем и устойчивости к штормам прерываний

З.А. Антипов, ORCID: 0009-0002-6936-274X <zaantipov@edu.hse.ru>
НИУ Высшая школа экономики,
101978, Россия, г. Москва, ул. Мясницкая, д. 20.

Аннотация. В работе предлагается методика сравнения операционных систем (ОС) по устойчивости к шторму прерываний – интенсивному потоку аппаратных событий, при котором обработка прерываний вытесняет полезную нагрузку и может привести к отказу в обслуживании (denial of service, DoS). Методика объединяет качественную оценку механизмов защиты и количественный вычислительный эксперимент. В качестве критериев рассматриваются маскирование источника, объединение обработки прерываний, адаптивный переход к опросу, управление прерываниями, сигнализируемыми сообщениями, привязка прерываний к процессорам, вытесняемость ядра и ограничение процессорного времени обработчиков. Экспериментальный стенд построен на основе эмулятора QEMU, виртуального устройства – генератора прерываний и плагина Tiny Code Generator (TCG), который выполняет статистическое профилирование ядра и пользовательского пространства без модификации гостевой ОС. Предлагаемый подход позволяет сопоставлять архитектурные механизмы защиты с измеряемым снижением доли процессорного времени, доступной полезной нагрузке.

Ключевые слова: операционная система; обработка прерываний; шторм прерываний; отказ в обслуживании; безопасность ядра; микроядро; эмулятор QEMU; компонент TCG; профилирование.

Для цитирования: Антипов З.А. Методика сравнения и анализа безопасности подсистем обработки прерываний в ядрах операционных систем и устойчивости к штормам прерываний. Труды ИСП РАН, 2026, том 38, вып. 4, часть 2, стр. 109–122. DOI: 10.15514/ISPRAS-2026-38(4)-21

Благодарности: Автор выражает большую благодарность Анне Мелеховой и Евгению Баскову за помощь в постановке задачи, ее решении и подготовке данной статьи.

Methodology for Comparing and Analyzing the Security of Interrupt-Handling Subsystems in Operating-System Kernels and Their Resilience to Interrupt Storms

Z.A. Antipov, ORCID: 0009-0002-6936-274X <zaantipov@edu.hse.ru>
National Research University, Higher School of Economics,
20, Myasnitskaya st., Moscow, 101978, Russia.

Abstract. This paper proposes a methodology for comparing operating systems (OSs) with respect to their resilience to interrupt storms–high-rate streams of hardware events in which interrupt handling displaces the application workload and may result in a denial-of-service (DoS) condition. The methodology combines a qualitative assessment of protection mechanisms with a quantitative computational experiment. The evaluation criteria include interrupt-source masking, interrupt coalescing (combining multiple events into a single delivery), adaptive switching to polling, management of message-signaled interrupts, interrupt affinity, kernel preemptibility, and limits on the processor time consumed by handlers. The experimental environment is based on the QEMU emulator, a virtual interrupt-generator device, and a Tiny Code Generator (TCG) plugin that performs statistical profiling of kernel and user-space execution without modifying the guest OS. The proposed approach makes it possible to relate architectural protection mechanisms to the measured reduction in the share of processor time available to the application workload.

Keywords: operating system; interrupt handling; interrupt storm; denial of service; kernel security; microkernel; QEMU; TCG plugin; statistical profiling.

For citation: Antipov Z. Methodology for Comparing and Analyzing the Security of Interrupt-Handling Subsystems in Operating-System Kernels and Their Resilience to Interrupt Storms. Trudy ISP RAN/Proc. ISP RAS, 2026, vol. 38, issue 4, part 2, pp. 109–122 (in Russian). DOI: 10.15514/ISPRAS-2026-38(4)-21

Acknowledgements. The author expresses great gratitude to Anna Melekhova and Evgeny Baskov for their assistance in formulating the problem, solving it, and preparing this article.

1. Введение

Подсистема обработки прерываний входит в доверенную вычислительную базу (trusted computing base, TCB), то есть в совокупность аппаратных и программных компонентов, от корректности которых зависит соблюдение политики безопасности операционной системы. Внешнее или виртуальное устройство может асинхронно остановить выполнение текущей задачи и передать управление привилегированному коду. Запрос прерывания (interrupt request, IRQ) сам по себе является штатным аппаратным механизмом, однако его неограниченная генерация превращает путь обработки прерываний в канал истощения процессорного времени. Штормом прерываний в работе называется режим, при котором входы в ядро, диспетчеризация, планирование, межпроцессное взаимодействие и работа драйверов систематически вытесняют полезную нагрузку – прикладную работу, ради которой запущена система.

Практическая актуальность проблемы подтверждается исследованиями и публичными бюллетенями безопасности. Метод ExpRace управляет моментами возникновения аппаратных прерываний и за счет этого повышает воспроизводимость эксплуатации состояний гонки в ядрах операционных систем Linux, Microsoft Windows и macOS [1]. В модели недоверенного драйвера показано, что злонамеренный компонент способен сформировать шторм прерываний и перевести систему в состояние активной блокировки (livelock) [2]. В 2025 году NVIDIA раскрыла уязвимость с идентификатором из перечня Common Vulnerabilities and Exposures (CVE) – CVE-2024-53881, позволявшую гостевой системе вызвать шторм прерываний на узле виртуализации и отказ в обслуживании; эта уязвимость также была включена в еженедельный бюллетень Агентства по кибербезопасности и защите инфраструктуры США (Cybersecurity and Infrastructure Security

Agency, CISA) [3, 4]. В 2026 году в Национальной базе данных уязвимостей США была опубликована CVE-2026-43488: неочищенное состояние ошибки расширяемого интерфейса хост-контроллера (eXtensible Host Controller Interface, xHCI) вызывало продолжающийся шторм прерываний и тяжелые системные отказы [5]. Следовательно, шторм прерываний является не только проблемой производительности, но и наблюдаемым сценарием нарушения доступности.

Несмотря на практическую значимость, задача сравнения операционных систем по устойчивости к такому сценарию остается недостаточно разработанной. Систематический обзор механизмов безопасности ядра классифицирует контроль доступа, защиту памяти, безопасную загрузку, контроль целостности и аппаратно поддерживаемую защиту, но не предлагает метрики устойчивости подсистемы прерываний [6]. Сравнения монолитных и микроядерных систем обычно измеряют задержки межпроцессного взаимодействия, системных вызовов и прикладную производительность [7], а критерии оценки доверенных систем сосредоточены на политике безопасности и уровне уверенности в ее реализации [8]. Работы, непосредственно связанные с прерываниями, исследуют эксплуатацию гонок или изоляцию отдельного недоверенного драйвера [1-2], но не задают воспроизводимой методики сопоставления разных операционных систем. Таким образом, существующие подходы покрывают отдельные аспекты проблемы, но не связывают архитектурные механизмы, экспериментальную нагрузку и единую шкалу сравнения.

Различие архитектур усиливает этот методический пробел. В монолитном ядре Linux общие механизмы регистрации, маскирования и синхронизации прерываний, их привязка к процессорам, переход сетевых драйверов к опросу, объединение прерываний и отключение необслуживаемых линий реализуются ядром и драйверными подсистемами [12-17]. В микроядерной системе ядро предоставляет минимальный механизм доставки, тогда как сброс причины прерывания, маскирование устройства, выбор приоритета и переключение к опросу могут выполняться пользовательским драйвером [18-19]. Поэтому одинаковая частота аппаратных событий создает различный профиль нагрузки: в него могут входить ядро, планировщик, межпроцессное взаимодействие, драйвер и системный сервер.

Цель работы – разработать методику и экспериментальную платформу для сравнения подсистем обработки прерываний в операционных системах и оценки их устойчивости к штормам прерываний. Для достижения цели решаются следующие задачи:

1. Описать шторм прерываний как сценарий отказа в обслуживании для подсистемы обработки прерываний;
2. Выделить механизмы операционной системы, влияющие на устойчивость к интенсивному потоку прерываний;
3. Сформулировать критерии и весовые коэффициенты для сравнения операционных систем;
4. Определить требования к профилировщику, который должен учитывать нагрузку как в ядре, так и в пользовательском пространстве;
5. Реализовать профилировщик как плагин TCG эмулятора QEMU;
6. Разработать план вычислительного эксперимента и критерии проверки корректности профиля.

Объект исследования – подсистемы обработки прерываний в операционных системах. Предмет исследования – механизмы, влияющие на устойчивость операционной системы к штормам прерываний, и способы измерения распределения нагрузки между ядром, драйверами, пользовательскими серверами и полезной нагрузкой.

Научная новизна работы состоит в формализации набора критериев сравнения устойчивости операционных систем к штормам прерываний и в связывании этих критериев с экспериментальным профилированием. Практическая значимость заключается в разработке

стенда на основе интерфейса плагинов TCG эмулятора QEMU: инструмент не требует изменения гостевой операционной системы, собирает стековые выборки выполнения и формирует данные для построения графика профиля.

2. Обзор предметной области

2.1 Микроядерная архитектура

Монолитное ядро размещает в привилегированном адресном пространстве планировщик, управление памятью, драйверы устройств, файловые системы и сетевой стек. Такое размещение сокращает число переходов между доменами защиты, но расширяет объем кода, ошибка в котором способна повлиять на всю систему. Микроядерный подход, напротив, оставляет в ядре минимальный набор механизмов и переносит значительную часть служб в изолированные пользовательские процессы [9-10].

К минимальным механизмам микроядра обычно относят управление адресными пространствами и потоками, планирование и межпроцессное взаимодействие (inter-process communication, IPC). Драйверы и системные службы выполняются как отдельные процессы, поэтому отказ такого компонента не обязан непосредственно повреждать память ядра; возможность локализации и перезапуска драйверов рассматривается как одно из преимуществ многосерверной архитектуры [10].

С точки зрения безопасности микроядерная архитектура стремится сократить TCB. Для микроядра seL4 функциональная корректность подтверждена машинно-проверяемым доказательством [11]. Однако доказательство корректности ядра относится к определенной модели и конфигурации и не означает автоматической устойчивости всей платформы к перегрузке: доступность по-прежнему зависит от аппаратуры, драйверов, планирования и политики распределения ресурсов.

2.2 Обработка шторма прерываний в Linux и микроядрах

В Linux базовый уровень обработки прерываний предоставляет интерфейсы регистрации обработчиков, маскирования, синхронизации и настройки типа прерывания [12]. В системе с симметричной многопроцессорной обработкой (symmetric multiprocessing, SMP) привязка прерываний к процессорам позволяет распределять поток событий или изолировать его от критической нагрузки [13]. Эти механизмы находятся в ядре и доступны драйверам через общую подсистему прерываний.

Для сетевых устройств Linux использует механизм обработки событий NAPI. Это устоявшееся название; в современной документации оно не рассматривается как расшифровываемая аббревиатура [14]. После первичного прерывания драйвер планирует ограниченный опрос очереди и временно сокращает число новых входов в обработчик. Объединение прерываний может дополнительно настраиваться средствами ethtool, если ее поддерживает сетевой адаптер [15]. Для устройств шины Peripheral Component Interconnect (PCI) руководство Linux описывает прерывания, сигнализируемые сообщениями (Message Signaled Interrupts, MSI), и расширение MSI-X, включая их маскирование и распределение векторов [16].

Linux также отслеживает необслуживаемые прерывания. Если ни один обработчик многократно не подтверждает источник, программа самодиагностики из файла kernel/irq/spurious.c учитывает такие события и может отключить проблемную линию, чтобы прекратить бесполезное потребление процессорного времени [17]. Этот механизм не заменяет корректную работу драйвера, но ограничивает последствия постоянно активного или ошибочно маршрутизированного источника.

В seL4 право на управление прерыванием представлено объектом полномочий IRQControl, из которого создается объект IRQHandler. Прерывание связывается с объектом уведомления,

а после обработки пользовательский драйвер вызывает функцию seL4_IRQHandler_Ack; до получения подтверждения повторная доставка через тот же обработчик блокируется [18]. Ядро тем самым предоставляет механизм доставки и подтверждения, но политика сброса причины на устройстве, маскирования источника и перехода к опросу остается ответственностью пользовательской подсистемы драйверов.

Планировщик seL4 выбирает готовый поток с наивысшим приоритетом. Поэтому пользовательский поток драйвера при низком приоритете может не успевать обслуживать устройство, а при чрезмерно высоком – вытеснять полезную нагрузку. В расширении seL4 для систем смешанной критичности (mixed-criticality systems, MCS) планировочный контекст содержит бюджет процессорного времени; этот механизм позволяет ограничить потребление процессора обработчиком прерываний [19].

Из сопоставления следует, что устойчивость микроядерной системы определяется не только механизмом доставки прерывания, но и политикой пользовательских драйверов: учетом статистики, маскированием источника, адаптивным опросом, объединение прерываний, привязкой к процессорам и бюджетами потоков. Это вывод из различного размещения механизма и политики, а не утверждение о безусловном превосходстве одной архитектуры над другой.

2.3 Профилирование шторма прерываний

Для исследования отказа в обслуживании необходимо наблюдать весь путь выполнения: вход в ядро, диспетчеризацию прерывания, переключение контекста, межпроцессное взаимодействие, работу пользовательского драйвера и возврат к полезной нагрузке. Встроенный профилировщик гостевой операционной системы не всегда пригоден для сравнения, поскольку его наличие, формат данных и накладные расходы различаются между системами.

Интерфейс программирования приложений (application programming interface, API) плагинов Tiny Code Generator (TCG) эмулятора QEMU позволяет регистрировать обработчики событий трансляции и выполнения гостевого кода [20]. В предлагаемом профилировщике обработчик выполнения используется как источник периодических выборок. В выборку входят счетчик команд (program counter, PC), регистр управления CR3, определяющий текущее адресное пространство, и стек вызовов, восстановленный по кадровому указателю; назначение регистров архитектуры x86 описано в руководстве Intel [21].

Адреса из выборок сопоставляются с символами файлов формата исполняемых и компонуемых объектов (Executable and Linkable Format, ELF). Результат представляется как график профиля исполнения (flame graph): ширина блока соответствует числу выборок, в которых встречается соответствующий стек вызовов [22]. Для пользовательского пространства дополнительно используется значение CR3, позволяющее разделить выборки разных адресных пространств.

2.4 Литературный обзор и исследовательский пробел

Работы о безопасности ядер в основном систематизируют механизмы предотвращения и обнаружения эксплуатации. В обзоре Z. Ali и соавторов исследования сгруппированы вокруг контроля доступа, защиты памяти, безопасной загрузки, контроля целостности, аппаратной поддержки, статического и динамического анализа и фаззинга [6]. Такой обзор задает широкий контекст безопасности ядра, однако шторм прерываний и доступность полезной нагрузки не выделены как самостоятельный объект сравнительной оценки.

Исследования архитектуры ядер обычно отвечают на другой вопрос. Классические работы по микроядрам рассматривают минимизацию ядра, стоимость межпроцессного взаимодействия и прикладную производительность [7, 9], а работы о MINIX 3 и seL4 – изоляцию отказов и формальную корректность [10-11]. Эти результаты необходимы для

выбора сравниваемых механизмов, но сами по себе не определяют интенсивность прерываний, при которой система теряет доступность.

Методики оценки доверенных систем также имеют иной масштаб. Критерии, рассмотренные NIST предназначены для объективной оценки доверия, политики доступа, аудита и доказательств корректности реализации [8]. Они позволяют сравнивать классы защищенности, но не задают динамический эксперимент с управляемым потоком аппаратных событий и не связывают результат с долей процессорного времени полезной нагрузки.

Работы, непосредственно посвященные безопасности прерываний, подтверждают возможность атак, но решают локальные задачи. ExpRace использует управление прерываниями для эксплуатации состояний гонки [1], но методика сравнения ОС не освещена в работах подобного класса.

Следовательно, исследовательский пробел состоит в отсутствии воспроизводимой методики, которая одновременно создает одинаковую нагрузку прерываниями, учитывает размещение механизмов защиты в ядре и пользовательском пространстве, измеряет сохранение процессорного времени для полезной нагрузки и предоставляет общую шкалу для сопоставления архитектур. Дальнейшие разделы вводят такую модель, критерии и экспериментальный стенд.

3. Модель и алгоритмы

3.1 Модель шторма прерываний

В исследуемом сценарии встроенное программное обеспечение или управляемое им устройство рассматривается как внешний источник прерываний. Интенсивность потока описывается числом аппаратных событий, поступающих за выбранный интервал времени.

Каждое прерывание запускает цепочку обработки. В микроядерной операционной системе эта цепочка обычно не ограничивается входом в ядро: управление может перейти к планировщику, механизму межпроцессного взаимодействия, пользовательскому драйверу и системному серверу. Поэтому измеряется не только число событий, но и распределение процессорного времени между компонентами системы.

В микроядерной операционной системе стоимость обработки прерывания включает не только выполнение ядра, но и работу пользовательского драйвера или сервера. Поэтому профилировщик разделяет выборки как минимум на пространство ядра и пользовательское пространство. Более точная модель учитывает конкретные процессы: значение CR3 различает адресные пространства, после чего виртуальный адрес выборки сопоставляется с исполняемым файлом компонента и его таблицей символов.

3.2 Алгоритм профилирования

Профилировщик реализуется как плагин подсистемы динамической трансляции Tiny Code Generator (TCG) эмулятора QEMU. При трансляции блоков плагин регистрирует функцию обратного вызова для выполнения каждой гостевой инструкции с помощью qemu_plugin_register_vcpu_insn_exec_cb и режима чтения регистров QEMU_PLUGIN_CB_R_REGS. Полная выборка снимается не на каждой инструкции, а после истечения интервала, заданного параметром delay в микросекундах. Для каждого виртуального процессора (vCPU) хранится отдельное состояние: дескрипторы счетчика команд, кадрового указателя и регистра CR3, время следующей выборки и буферы чтения регистров и памяти гостевой системы.

При установке плагин поддерживает 64-разрядную архитектуру x86-64 и 32-разрядную архитектуру Intel Architecture 32 (IA-32, i386). Для x86-64 он ищет регистры rip, rbp, rsp и cr3, а для IA-32 – eip, ebp, esp и cr3. Если для виртуального процессора не удалось найти счетчик

команд, кадровый указатель или CR3, профилирование этого процессора отключается. Во время выборки плагин читает регистры и проходит по цепочке кадров стека в памяти гостя: по адресу кадрового указателя считывается предыдущий кадр, а по смещению, равному ширине указателя, – адрес возврата. Глубина стека ограничена константой MAX_DEPTH = 64.

4. Программная реализация и вычислительный эксперимент

4.1 Архитектура профилировщика

Практическая часть работы представляет собой профилировщик, реализованный как плагин TCG эмулятора QEMU. Выбор подхода обусловлен двумя требованиями.

1. Инструмент должен работать без модификации гостевой микроядерной операционной системы.
2. Инструмент должен учитывать выполнение ядра, пользовательских драйверов и системных серверов.

Интерфейс плагинов TCG удовлетворяет этим требованиям: плагин динамически подключается к эмулятору, получает сведения о гостевой архитектуре и регистрирует функции обратного вызова для событий выполнения [20]. Основные модули профилировщика:

1. модуль разбора параметра out, задающего выходной файл, и параметра delay, задающего интервал выборки;
2. модуль определения архитектуры гостевой системы и ширины указателя;
3. модуль регистрации обработчиков обратного вызова QEMU для виртуальных процессоров и блоков трансляции;
4. модуль хранения состояния каждого виртуального процессора на основе таблицы состояний (scoreboard);
5. модуль выборочного профилирования и чтения счетчика команд, кадрового указателя и CR3;
6. модуль восстановления стека вызовов по кадровому указателю;
7. модуль записи необработанных выборок в двоичном формате;
8. внешний агрегатор символов ELF ядра и пользовательских процессов, сопоставляющий CR3 с адресными пространствами и формирующий график профиля исполнения.

Такая структура отделяет сбор необработанных записей от их интерпретации и позволяет использовать один инструмент для разных сценариев нагрузки.

4.2 Виртуальное устройство для генерации прерываний

Для воспроизводимого исследования шторма прерываний, помимо профилировщика, требуется виртуальное устройство QEMU, которое имитирует поведение контроллера и позволяет задавать режимы генерации прерываний.

Устройство должно быть простым, детерминированным и управляемым из гостевой операционной системы. Его задача – не моделировать целиком сетевой адаптер или контроллер энергонезависимой памяти с интерфейсом Non-Volatile Memory Express (NVMe), а воспроизводить одиночные и пакетные прерывания, периодическую генерацию и сигналы по фронту или по уровню. Разработанная реализация isa-irq-storm использует совместимый с архитектурой Industry Standard Architecture (ISA) интерфейс портов ввода-вывода размером не менее 0x20 байт; базовый адрес по умолчанию равен 0x560. Параметры задаются

свойствами QEMU iobase, iosize, irq, burst, period-us, start-enabled и level-triggered; общий механизм системной эмуляции QEMU описан в [23].

Регистр IRQ_STORM_REG_CTRL управляет включением генератора и типом сигнала (Листинг 1). Бит IRQ_STORM_CTRL_ENABLE запускает таймер, а IRQ_STORM_CTRL_LEVEL выбирает прерывание по уровню. Регистр IRQ_STORM_REG_IRQ возвращает номер линии ISA, заданный свойством irq. Регистр IRQ_STORM_REG_BURST определяет число импульсов в пакетном режиме: при каждом срабатывании таймера устройство вызывает qemu_irq_pulse() от одного до IRQ_STORM_MAX_BURST раз.

```
#define IRQ_STORM_REG_CTRL       0x00
#define IRQ_STORM_REG_IRQ        0x01
#define IRQ_STORM_REG_BURST      0x02
#define IRQ_STORM_REG_STATUS     0x03
#define IRQ_STORM_REG_PERIOD_US  0x04
#define IRQ_STORM_REG_PULSES_LO   0x08
#define IRQ_STORM_REG_PULSES_HI   0x0c
#define IRQ_STORM_REG_TIMER_CB    0x10
#define IRQ_STORM_REG_CFG_WRITES  0x14
#define IRQ_STORM_REG_EN_TOGGLES  0x18
#define IRQ_STORM_REG_ACK         0x1c
```

Листинг 1. Карта регистров виртуального устройства генерации прерываний.
Listing 3.1. Register map of the virtual interrupt-storm generator.

Регистр IRQ_STORM_REG_STATUS отражает состояние генератора, линии прерывания и режима по уровню. Регистр IRQ_STORM_REG_PERIOD_US задает период таймера в микросекундах; после изменения периода срок следующего срабатывания пересчитывается от текущего времени. Регистры IRQ_STORM_REG_PULSES_LO и IRQ_STORM_REG_PULSES_HI образуют 64-битный счетчик импульсов. IRQ_STORM_REG_TIMER_CB хранит число срабатываний таймера, IRQ_STORM_REG_CFG_WRITES – число изменений конфигурации, а IRQ_STORM_REG_EN_TOGGLES – число включений и отключений генератора. Запись ненулевого значения в IRQ_STORM_REG_ACK опускает линию в режиме по уровню вызовом qemu_irq_lower().

Поведение таймера различается для двух типов сигнала. В режиме по фронту одно срабатывание генерирует пакет коротких импульсов. В режиме по уровню первое срабатывание поднимает линию вызовом qemu_irq_raise() и оставляет ее активной до записи в IRQ_STORM_REG_ACK или отключения генератора. Этот режим воспроизводит ситуацию, в которой причина прерывания не сброшена и система повторно входит в путь обработки.

4.3 Методика исследования

Методика исследования строится вокруг трех компонентов: эмулятора QEMU с виртуальным генератором прерываний, гостевой микроядерной операционной системы и профилировщика TCG. Устройство создает управляемый поток событий, гостевая система обрабатывает его штатными механизмами, а профилировщик измеряет распределение нагрузки между ядром, пользовательскими компонентами и полезной нагрузкой.

Схема экспериментального стенда показана на рис. 1.

Базовая последовательность эксперимента выглядит следующим образом:

1. Запустить гостевую операционную систему в QEMU без активной генерации прерываний и снять базовый профиль;

2. Включить виртуальное устройство в режиме одиночных прерываний по фронту и проверить корректность доставки;
3. Включить периодическую генерацию с разными значениями IRQ_STORM_REG_PERIOD_US;
4. Включить пакетный режим и изменять число импульсов в пакете;
5. Повторить эксперименты для прерываний по уровню, включая сценарий, в котором линия остается активной до явного сброса состояния;
6. Собрать выборки для каждого сценария;
7. Сравнить изменение долей входа в ядро, диспетчеризации прерываний, межпроцессного взаимодействия, пользовательского драйвера, сервера и полезной нагрузки.

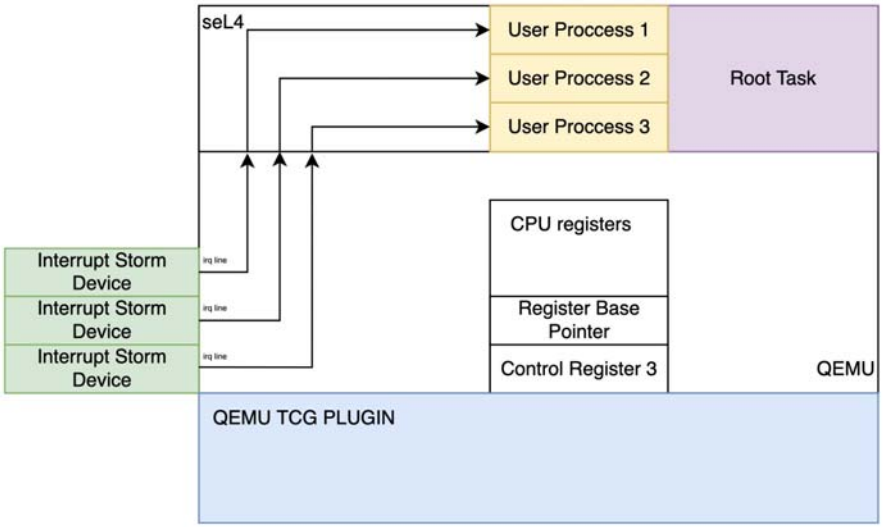

Рис. 1. Архитектура экспериментального стенда.
Fig. 1. Architecture of the experimental setup.

Отдельно исследуются механизмы, которые могут уменьшать ущерб от шторма прерываний или менять профиль нагрузки.

1. Объединение прерываний. Устройство или драйвер объединяет несколько событий в одну доставку. В эксперименте механизм оценивается по уменьшению числа фактических прерываний при сохранении числа обработанных событий.
2. Адаптивный опрос. Вместо постоянной доставки прерываний драйвер или сервер временно проверяет состояние устройства самостоятельно. Это сокращает число входов в обработчик, но увеличивает вычислительную работу пользовательского компонента.
3. Механизм NAPI Linux и очереди NVMe. NAPI сочетает прерывания с ограниченным опросом при высокой сетевой нагрузке [14], а спецификация NVMe предусматривает несколько очередей ввода-вывода и настройку векторов прерываний [24]. В микроядерной системе аналогичная политика реализуется пользовательским драйвером или сервером ввода-вывода.

4. Маскирование при перегрузке. Если обработчик не успевает завершить предыдущую работу, новые события временно запрещаются. Для прерываний по уровню это особенно важно: пока причина не сброшена, линия остается активной.
5. Маскирование до обработки. Система сначала запрещает повторные прерывания от устройства, затем обрабатывает накопленное состояние и только после этого снова разрешает доставку.
6. Балансировка между процессорами. Для гостевой SMP-системы сравниваются ручная привязка прерываний к одному виртуальному процессору, распределение между несколькими виртуальными процессорами и автоматическая балансировка.
7. Приоритеты потоков обработки. В микроядерной операционной системе обработка может продолжаться в пользовательском потоке драйвера или сервера. Эксперимент сравнивает задержки полезной нагрузки при меньшем, равном и большем приоритете обработчика.
8. Ограничение процессорного времени пользовательского обработчика. Проверяется наличие бюджета планировочного контекста. В MCS-конфигурации seL4 такой механизм ограничивает время, которое драйвер может потребить при шторме, и сохраняет часть процессорного ресурса для полезной нагрузки [19].
9. Вытесняемость ядра и длительность невытесняемых участков. Длительное выполнение ядра без вытеснения увеличивает задержки критических потоков, поэтому эксперимент фиксирует способность системы ограничивать время нахождения в критических путях.
10. Управление MSI и MSI-X. Отдельно определяется, где находится политика настройки, маскирования и маршрутизации прерываний, сигнализируемых сообщениями. В seL4 такая политика может выполняться пользовательским драйвером; следовательно, аппаратное маскирование должно оцениваться как свойство всей драйверной подсистемы, а не только ядра.

Для непосредственного сравнения операционных систем вводится интегральная оценка устойчивости к штурму прерываний. Каждый механизм получает частную оценку r_i в диапазоне от 0 до 1: 0 означает отсутствие механизма, 0,5 – частичную или ручную поддержку, 1 – воспроизводимо включаемый механизм. Итоговая оценка вычисляется как

$$R = \sum_i w_i R_i, \sum_i w_i = 1$$

где веса w_i выбираются в зависимости от профиля угроз. Приведенные ниже коэффициенты являются исходной экспертной гипотезой, а не универсальными эмпирическими константами: больший вес присваивается механизмам, непосредственно ограничивающим частоту доставки или процессорное время обработчика. Перед сравнением конкретных систем веса фиксируются, а после серии экспериментов проверяется устойчивость вывода к их изменению. Для базового сравнения используется следующий набор весов:

1. маскирование источника прерывания до или во время обработки – 0,15;
2. корректная обработка прерываний по уровню и сброс причины – 0,12;
3. объединение прерываний или пакетная обработка – 0,10;
4. адаптивный переход к опросу, включая схемы, аналогичные NAPI, – 0,12;
5. балансировка и привязка прерываний к процессорам – 0,08;
6. политика приоритетов потоков обработчиков относительно полезной нагрузки – 0,10;
7. ограничение процессорного времени пользовательского обработчика, например через бюджет планировочного контекста в MCS-конфигурации seL4, – 0,13;

8. вытесняемость ядра и ограничение длительности невытесняемых участков обработки – 0,08;
9. управление MSI и MSI-X, включая место настройки маскирования и маршрутизации, – 0,07;
10. наблюдаемость: счетчики прерываний, статистика перегрузки и возможность сопоставить источник с потоком, драйвером и потребленным процессорным временем – 0,05.

Для системы реального времени увеличиваются веса бюджета обработчика, вытесняемости ядра и задержек полезной нагрузки. Для сетевого или дискового устройства больший вес получают объединение обработки прерываний, адаптивный опрос и управление MSI и MSI-X. Интегральная оценка не заменяет измерения, а задает предварительную общую шкалу сравнения Linux, seL4, seL4 в конфигурации MCS и других систем.

Таким образом, методика исследует не только факт отказа в обслуживании, но и реакцию системы на разные стратегии защиты. Особое внимание уделяется прерываниям по уровню: при ошибочной конфигурации или злонамеренном встроенном программном обеспечении линия может оставаться активной до сброса либо маскирования причины.

4.4 План вычислительного эксперимента

Эксперимент сравнивает профиль микроядерной операционной системы в нормальном режиме и при искусственном шторме прерываний. Поток событий создает виртуальное устройство, а профилировщик TCG собирает выборки выполнения. Основной количественный показатель – нормированная производительность полезной нагрузки $U(\lambda) = W(\lambda) / W_0$, где W_0 – число единиц полезной работы за фиксированный интервал без искусственного шторма, $W(\lambda)$ – тот же показатель при интенсивности прерываний λ. Порог отказа θ задается до эксперимента; для иллюстративного сценария может использоваться $\theta = 0,5$. План эксперимента:

1. собрать базовый профиль без искусственного шторма прерываний;
2. запустить сценарии с разной частотой генерации прерываний;
3. сравнить режимы прерываний по уровню и по фронту;
4. сравнить одиночные прерывания и пакетный режим;
5. проверить маскирование, объединение прерываний и адаптивный опрос;
6. для SMP-конфигурации сравнить ручную и автоматическую балансировку прерываний между виртуальными процессорами;
7. запустить поток полезной нагрузки и сравнить меньший, равный и больший приоритет пользовательского обработчика;
8. для MCS-конфигурации повторить эксперимент с ограниченным бюджетом планировочного контекста обработчика;
9. для каждого сценария измерить долю времени в пространстве ядра, пользовательских компонентов и полезной нагрузки;
10. построить график профиля исполнения для каждого сценария;
11. сравнить рост долей входа в ядро, межпроцессного взаимодействия, планировщика, пользовательского драйвера и сервера;
12. определить порог интенсивности как минимальное λ, при котором $U(\lambda) \le \theta$ в течение заданного окна наблюдения; для системы реального времени дополнительным условием считается нарушение установленного срока выполнения.

План позволяет раздельно оценить влияние частоты прерываний, типа сигнала, планирования обработчиков и защитных механизмов на доступность полезной нагрузки.

4.5 Ожидаемые результаты и проверка адекватности

Ожидается, что при росте интенсивности прерываний график профиля исполнения покажет увеличение доли стеков ядра. Для проверки этого было запущено два теста. В первом сценарии прерывания генерировались виртуальным девайсом каждые 50 микросекунд (рис. 2), во втором – каждые 1000 микросекунд (рис. 3).

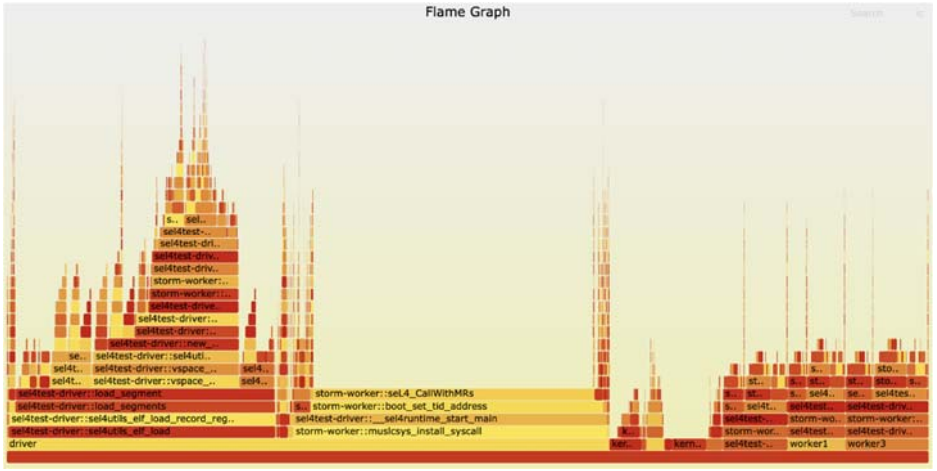

Рис. 2. График профиля нагрузки для генерации прерываний по фронту каждые 50 мкс.
Fig. 2. Flame graph for edge-triggered interrupt generation every 50 µs.

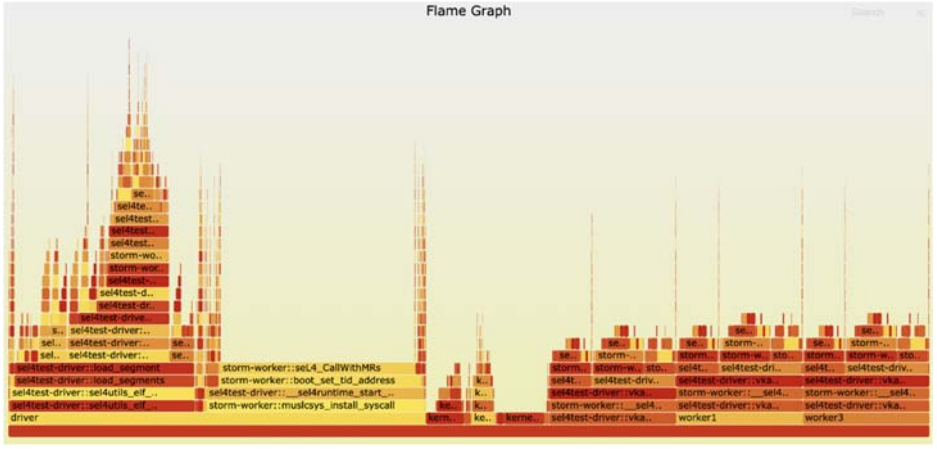

Рис. 3. График профиля нагрузки для генерации прерываний по фронту каждые 1000 мкс.
Fig. 3. Flame graph for edge-triggered interrupt generation every 1000 µs.

Для анализа получившихся результатов можно оценить отношение замеров (samples) времени нахождения исполнения в ядерной функции c_handle_interrupt() и одного из worker процессов (табл. 1).

Табл. 1. Предварительная форма представления ожидаемых результатов эксперимента.
Table 1. Preliminary format for presenting the expected experimental results.

Сценарий	Количество замеров (kernel::c_handle_interrupt/worker1)	Соотношение
Прерывания каждые 50 мкс.	1236087/2218361	0.56
Прерывания каждые 1000 мкс.	2142627/7106781	0.3

Можно заметить, что в случае увеличения нагрузки, за счет возрастания частоты генерации прерываний, увеличивается и доля времени нахождения в ядре. Это говорит о том, что выработанная методика и разработанные инструменты позволяют оценить необходимые метрики для проведения всех запланированных экспериментов.

5. Заключение

В работе шторм прерываний рассмотрен как сценарий отказа в обслуживании операционной системы. Для исследования сформулирована модель нагрузки и разработан стенд на основе QEMU, виртуального генератора прерываний и профилировщика TCG. Профилировщик собирает стековые выборки и значение CR3 в двоичном формате; внешний агрегатор сопоставляет адреса с символами и строит график профиля нагрузки.

Анализ Linux выявляет несколько механизмов ограничения шторма: маскирование, отключение необслуживаемых прерываний, объединение обработки, переход NAPI к опросу и привязку к процессорам [12-17]. В микроядерной системе часть этой политики переносится в пользовательские драйверы и серверы [18, 19]. Следовательно, малый и верифицированный код ядра уменьшает доверенную вычислительную базу, но не устраняет необходимость контролировать доступность драйверной подсистемы.

Дальнейшая работа включает автоматическое сопоставление адресов с символами гостевой системы, эксперименты на нескольких операционных системах, сравнение и оценка реализованных там механизмов и расширение режимов виртуального устройства. Для снижения зависимости от кадрового указателя планируется исследовать аппаратно поддерживаемую трассировку переходов. Это также упростит профилирование систем с рандомизацией размещения адресного пространства (address space layout randomization, ASLR).

Список литературы / References

[1]. Lee Y., Min C., Lee B. ExpRace: Exploiting Kernel Races through Raising Interrupts // 30th USENIX Security Symposium, 2021, pp. 2363-2380. Available at: https://www.usenix.org/conference/usenixsecurity21/presentation/lee-yoochan, accessed 16.07.2026.

[2]. Boyd-Wickizer S., Zeldovich N. Tolerating Malicious Device Drivers in Linux. 2010 USENIX Annual Technical Conference, 2010, pp. 9-22. Available at: https://www.usenix.org/conference/usenix-atc-10/tolerating-malicious-device-drivers-linux, accessed 16.07.2026.

[3]. NVIDIA. Security Bulletin: NVIDIA GPU Display Driver – January 2025. CVE-2024-53881. Available at: https://nvidia.custhelp.com/app/answers/detail/a_id/5614/, accessed 16.07.2026.

[4]. Cybersecurity and Infrastructure Security Agency. Vulnerability Summary for the Week of January 27, 2025. Available at: https://www.cisa.gov/news-events/bulletins/sb25-034, accessed 16.07.2026.

[5]. National Vulnerability Database. CVE-2026-43488: usb: xhci: Prevent interrupt storm on host controller error. Available at: https://nvd.nist.gov/vuln/detail/CVE-2026-43488, accessed.

[6]. Ali Z., Aslam N., Marotta A., Tiberti W., Cassioli D. A Systematic Review of Kernel-Level Security Mechanisms, Vulnerability Detection and Mitigation in Modern Operating Systems. Sensors, 2026, vol. 26, no. 8, Art. 2452. DOI: 10.3390/s26082452.

[7]. Härtig H., Hohmuth M., Liedtke J., Schönberg S., Wolter J. The Performance of µ-Kernel-Based Systems. Proceedings of the 16th ACM Symposium on Operating Systems Principles, 1997, pp. 66-77. DOI: 10.1145/268998.266660.

[8]. Dinkel C. A Review of U.S. and European Security Evaluation Criteria. NISTIR 4774. National Institute of Standards and Technology, 1992. DOI: 10.6028/NIST.IR.4774.

[9]. Liedtke J. On Micro-Kernel Construction. Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles, 1995, pp. 237-250. DOI: 10.1145/224056.224075.

[10]. Tanenbaum A. S., Herder J. N., Bos H. Can We Make Operating Systems Reliable and Secure? Computer, 2006, vol. 39, no. 5, pp. 44-51. DOI: 10.1109/MC.2006.156.

[11]. Klein G. et al. seL4: Formal Verification of an OS Kernel. Proceedings of the 22nd ACM Symposium on Operating Systems Principles, 2009, pp. 207-220. DOI: 10.1145/1629575.1629596.

[12]. The Linux Kernel documentation. Linux generic IRQ handling. Available at: https://docs.kernel.org/core-api/genericirq.html, accessed 16.07.2026.

[13]. The Linux Kernel documentation. SMP IRQ affinity. Available at: https://docs.kernel.org/core-api/irq/irq-affinity.html, accessed 16.07.2026.

[14]. The Linux Kernel documentation. NAPI. Available at: https://docs.kernel.org/networking/napi.html, accessed 16.07.2026.

[15]. Kerrisk M. ethtool(8) – Linux manual page. Available at: https://man7.org/linux/man-pages/man8/ethtool.8.html, accessed 16.07.2026.

[16]. The Linux Kernel documentation. The MSI Driver Guide HOWTO. Available at: https://docs.kernel.org/PCI/msi-howto.html, accessed 16.07.2026.

[17]. Linux kernel source tree. kernel/irq/spurious.c. Available at: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/kernel/irq/spurious.c, accessed 16.07.2026.

[18]. seL4 documentation. Interrupts tutorial. Available at: https://docs.sel4.systems/Tutorials/interrupts.html, accessed 16.07.2026.

[19]. seL4 documentation. MCS tutorial. Available at: https://docs.sel4.systems/Tutorials/mcs.html, accessed 16.07.2026.

[20]. QEMU documentation. QEMU TCG Plugins. Available at: https://www.qemu.org/docs/master/devel/tcg-plugins.html.

[21]. Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manuals. Available at: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html, accessed 16.07.2026.

[22]. Gregg B. Flame Graphs. Available at: https://www.brendangregg.com/flamegraphs.html, accessed 16.07.2026.

[23]. QEMU documentation. System Emulation. Available at: https://www.qemu.org/docs/master/system/, accessed 16.07.2026.

[24]. NVM Express, Inc. NVM Express Base Specification, Revision 2.3, 2025. Available at: https://nvmexpress.org/specifications/, accessed 16.07.2026.

Информация об авторе / Information about the Author

Захар Алексеевич Антипов – магистрант ФКН НИУ ВШЭ. Научные интересы: архитектура операционных систем; микроядерные системы; обработка прерываний; безопасность систем; виртуализация; профилирование производительности.

Zakhar Alekseevich Antipov – Master's student at the Faculty of Computer Science at the National Research University Higher School of Economics. Research interests: operating-system architecture; microkernel systems; interrupt handling; systems security; virtualization; performance profiling.