Рубрика 1. Информатика и информационные процессы

УДК *004.451.25*

Е. А. Еремина [1]***, А. Н. Простова*** [1]
[1]*ФГАОУ ВО «РГУ нефти и газа (НИУ) имени И.М. Губкина», г. Москва, Российская Федерация*

ИЗОЛЯЦИЯ ПРОЦЕССОВ В ОПЕРАЦИОННОЙ СИСТЕМЕ АЛЬТ С ИСПОЛЬЗОВАНИЕМ МЕХАНИЗМОВ CGROUPS V2

Аннотация. *Проблема управления вычислительными ресурсами в многозадачных операционных системах приобретает особую значимость с распространением сред контейнеризации и необходимостью одновременного выполнения задач с различными требованиями к ресурсам. Традиционные механизмы Linux – nice, cpulimit, ulimit – не обеспечивают комплексной групповой изоляции: nice задаёт лишь рекомендательный приоритет, cpulimit использует сигналы SIGSTOP/CONT, а ulimit ограничивает только сессию пользователя. Проблема интерференции рабочих нагрузок, известная как проблема «шумного соседа», остаётся ключевой в системном администрировании и информационной безопасности. Цель исследования: экспериментальная количественная оценка эффективности механизмов изоляции ресурсов cgroups v2 в дистрибутиве ОС Альт (ядро Linux 6.1.115). Методика. Для проверки гипотез H1a (изоляция CPU) и H1b (изоляция памяти) разработана серия контролируемых экспериментов с применением контроллеров cpuset и memory.max при варьировании нагрузки: 1–2 ядра, лимиты памяти 50–500 МБ, смешанный сценарий с приоритетными и фоновыми процессами. Измеряемые метрики: PSR (принадлежность ядру CPU), nr_switches (переключения контекста), memory.events (счётчики OOM Killer). Результаты. При использовании cpuset значение PSR для 100% наблюдений совпадало с назначенным ядром; nr_switches составил 3–4 переключения без межгрупповой миграции. При превышении лимита memory.max активировался механизм завершения процессов при нехватке памяти OOM Killer (oom_kill=1), не затрагивая другие группы. Выводы. Гипотезы H1a и H1b полностью подтверждены. Cgroups v2 демонстрирует принципиальные преимущества перед традиционными механизмами изоляции. Механизмы cpuset и memory.max могут поддерживать отдельные требования по разделению ресурсов и ограничению их исчерпания, однако сами по себе не доказывают соответствие приказу ФСТЭК России № 118 или PCI DSS 4.0.*

Ключевые слова: *cgroups v2; изоляция процессов; cpuset; memory.max; управление ресурсами; контейнеризация; информационная безопасность.*

Введение

Актуальность проблемы изоляции процессов возрастает с усложнением программного обеспечения, распространением сред контейнеризации и необходимостью одновременного выполнения задач с различными требованиями к ресурсам [1]. Проблема интерференции рабочих нагрузок (известная как проблема «шумного соседа») – когда один процесс дестабилизирует всю систему – остается ключевой в системном администрировании и информационной безопасности. Инструменты Linux широко применяются для моделирования телекоммуникационных сетей [2] и подготовки системных администраторов [1]. Платформы контейнеризации, например, Docker и Podman, занимают в задачах изоляции центральное место, и понимание механизмов ресурсной изоляции приобретает практическую значимость.

Традиционные механизмы управления ресурсами в Linux не обеспечивают комплексной групповой изоляции. Например, команда nice задаёт лишь рекомендательный приоритет [3], cpulimit периодически приостанавливает процесс сигналами SIGSTOP/CONT – такой подход устарел для многоядерных систем [4], ulimit задаёт лимиты на уровне сессии пользователя, но не группы процессов [5]. Всё это стало стимулом к разработке механизма Control Groups.

Объект исследования: механизмы cgroups v2 в ОС Альт.

Предмет исследования: контроллеры cpuset и memory.max и их эффективность в обеспечении изоляции ресурсов.

Цель исследования: экспериментальная количественная оценка эффективности механизмов изоляции ресурсов cgroups v2 в ОС Альт.

Задачи исследования:

1. Провести сравнительный анализ традиционных механизмов управления ресурсами Linux и механизма cgroups v2.
2. Развернуть экспериментальную среду и выполнить контрольные замеры без применения cgroups.

3. Количественно оценить изоляцию CPU с применением контроллера cpuset при изменении числа ядер и типов нагрузки.

4. Количественно оценить изоляцию памяти с применением контроллера memory.max при изменении лимитов.

5. Исследовать сценарий смешанной нагрузки и верифицировать отсутствие взаимного влияния групп.

6. Выработать практические рекомендации по применению cgroups v2 в соответствии с требованиями регуляторов.

Теоретические основы

Архитектура cgroups v2: унифицированная иерархия

Control Groups (cgroups) – это механизм ядра Linux. Он служит для группировки процессов, а также управления их доступом к системным ресурсам, таким как CPU, память, ввод-вывод. В отличие от cgroups v1, где каждый контроллер монтировался в отдельной иерархии, cgroups v2 предоставляет единую унифицированную иерархию для всех контроллеров, монтируемую в /sys/fs/cgroup [6]. Это устраняет несогласованность политик при одновременном использовании нескольких контроллеров и упрощает конфигурирование.

Иерархия cgroups v2 представляет собой дерево директорий в файловой системе типа cgroup2. Корневая cgroup (/) объединяет все системные процессы. Создание поддиректории в ее иерархии приводит к появлению дочерней группы. Ключевым правилом безопасности является то, что дочерние группы наследуют ограничения родителя, но не могут их превышать [6]. Единая модель иерархии выступает обязательным условием для обеспечения совместного использования cgroups с пространствами имён ядра Linux. Детерминированность изоляции в рамках данного исследования – это свойство механизма гарантировать соблюдение ресурсных ограничений при любых условиях нагрузки, без зависимости от случайных решений планировщика или мгновенного состояния системы.

Контроллеры cpuset и memory

Контроллер cpuset закрепляет процессы группы за конкретными ядрами CPU. Он исключает их конкуренцию с процессами из других групп. Это предотвращает атаки по сторонним каналам (cache-timing attacks), связанные с совместным использованием кэшей CPU. Ключевые параметры:

1. cpuset.cpus – список разрешённых ядер; cpuset.mems – список разрешённых NUMA-узлов;
2. cpuset.cpus.effective – фактически доступные ядра с учётом ограничений родителя.

Контроллер memory управляет оперативной памятью группы. Параметр memory.max задаёт жёсткий лимит, то есть при его превышении ядро активирует механизм завершения процессов при нехватке памяти (OOM Killer, Out-Of-Memory Killer). Принципиальным отличием от cgroups v1 является то, что в v2 также поддерживается параметр memory.high. Эта директива устанавливает мягкое ограничение с регулированием скорости выполнения процессов, не приводящее к их немедленному завершению [7]. Для детерминированного срабатывания OOM Killer используется именно memory.max. Параметр memory.swap.max = 0 не дает обойти жесткий лимит через файл подкачки.

Взаимодействие с пространствами имён (namespaces)

Механизм cgroups v2 тесно взаимодействует с пространствами имён ядра Linux [8]. В частности, пространство имён cgroup (CLONE_NEWCGROUP) обеспечивает изоляцию представления иерархии cgroups для процесса, который работает внутри контейнера.

Исходя из сказанного выше, процессы внутри контейнера, не имея доступа к родительским группам, видят собственный корень иерархии. Совместное использование cgroups v2 с pid namespace, mount namespace и network namespace дает ту самую изоляцию, которая используется в Docker и Podman. В документации ядра [6] этот подход описан как предпочтительный для контейнерных сред.

Рубрика 1. Информатика и информационные процессы

Литературный обзор

Значительная часть исследований посвящена сравнению производительности cgroups v1 и v2. В работе Wiedner и др. [9], представленной на IEEE NFV-SDN, экспериментально показано, что cgroups v2 демонстрирует меньшие задержки на высоких перцентилях (P95, P99) по сравнению с v1 при сопоставимой изоляции. Например, реализация v2 выполняет примерно на 2,4% меньше инструкций, а количество миграций процессов сокращается в два раза. В документации systemd по управлению ресурсами [10] параметр MemoryMax= сопоставляется с жёстким пределом памяти cgroups v2; при достижении предела применяется OOM-механизм, что подтверждает применимость контроллера для ограничения потребления памяти сервисами. В документации PostgreSQL [7] отмечается принципиальное различие между мягкими (memory.high) и жёсткими (memory.max) лимитами в cgroups v2 – это различие напрямую учитывается в нашем эксперименте.

Практические аспекты настройки cgroups в ОС Альт представлены в работе Крайнова и Пашиной [11]. Однако данное исследование ограничивается демонстрацией базовых возможностей и не содержит количественной оценки изоляции через метрики PSR, nr_switches и memory.events. Это составляет научную новизну настоящей работы. Таким образом, формализованный эксперимент с количественной оценкой метрик изоляции именно в ОС Альт в открытом доступе отсутствует.

Методология

Гипотезы исследования:

1. H1a: использование контроллера cpuset в cgroups v2 гарантирует, что процессы ограниченной группы не выполняются на ядрах CPU вне своего cpuset ни при каких условиях нагрузки. Условие принятия гипотезы: для 100% наблюдений PSR находится внутри cpuset.cpus; nr_switches не превышает базовое значение более чем на 20%.
2. H1b: использование контроллера memory.max с отключенным swap гарантирует, что процессы, превышающие установленный лимит памяти, завершаются механизмом OOM Killer до того, как нехватка памяти затронет остальные группы. Условие принятия гипотезы: oom_kill ≥ 1 при превышении лимита или oom_kill = 0 при соблюдении лимита.

Среда исследования: ОС Альт (ALT Linux, ядро Linux версии 6.1.115) [12], cgroups v2 активны по умолчанию, 2 логических ядра CPU (CPU0, CPU1), 4 ГБ ОЗУ.

Инструменты: bash, stress (v1.0.4), htop, vmstat, dmesg.

Количественные метрики:

– PSR из команды ps -e -o pid,psr,cmd – номер ядра CPU для верификации изоляции;

– nr_switches из /proc/PID/schedstat – количество переключений контекста (прокси-метрика стабильности);

– memory.events (max, oom, oom_kill) – счётчики событий памяти для оценки OOM Killer;

– dmesg – системный журнал ядра для подтверждения завершения процессов-нарушителей.

Для углубленной оценки латентности (lmbench lat_ctx, perf stat) и пропускной способности требуется дополнительное исследование.

Процедура эксперимента:

– контрольный эксперимент без cgroups для получения базовых значений;

– серия тестов cpuset с изменением числа ядер (1, 2) и типов нагрузки;

– серия тестов memory.max с изменением лимитов (50 МБ, 500 МБ, без лимита);

– сценарий смешанной нагрузки (приоритетный + фоновый процессы).

Экспериментальное исследование

Среда и подготовка

Перед началом экспериментов были выполнены проверка активности cgroups v2 и установка необходимых инструментов:

```
apt-get install stress
stress --version # stress 1.0.4
```

Наличие cgroup2 в /proc/mounts и контроллера cpuset в cgroup.controllers подтвердили активность cgroups v2. Дополнительно установлены утилита stress v1.0.4 и компилятор gcc.

Контрольный эксперимент (без cgroups)

Для получения базовых значений метрик два процесса stress запущены без применения cgroups:

```
stress -c 2 &
ps -e -o pid,psr,cmd | grep stress
```

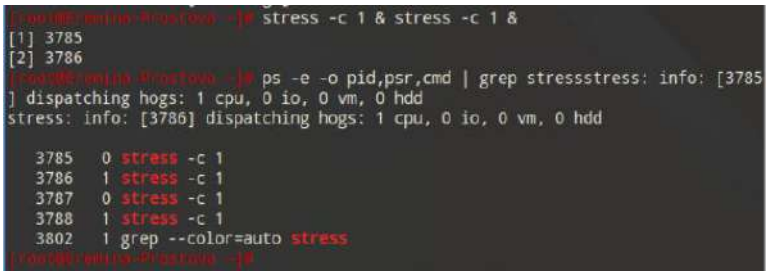

Рис. 1. Распределение процессов stress по ядрам CPU без использования cgroups

В контрольном эксперименте планировщик распределил процессы произвольно. Оба ядра загружены, однако администратор не может гарантировать, что критически важный процесс не будет конкурировать с менее приоритетным на одном ядре. Это подтверждает необходимость детерминированных механизмов изоляции.

Изоляция CPU через cpuset

Развертывание иерархии для двух изолированных групп group_A (ядро 0) и group_B (ядро 1):

```
# Включение cpuset в корне и создание иерархии
echo "+cpuset" > /sys/fs/cgroup/cgroup.subtree_control
mkdir /sys/fs/cgroup/cpu-isolation
echo "+cpuset" > /sys/fs/cgroup/cpu-isolation/cgroup.subtree_control
mkdir /sys/fs/cgroup/cpu-isolation/group_A
mkdir /sys/fs/cgroup/cpu-isolation/group_B

# Назначение ядер: group_A → 0, group_B → 1
echo "0-1" > /sys/fs/cgroup/cpu-isolation/cpuset.cpus
echo "0"   > /sys/fs/cgroup/cpu-isolation/group_A/cpuset.cpus
echo "1"   > /sys/fs/cgroup/cpu-isolation/group_B/cpuset.cpus

# Запуск процессов и помещение в группы
stress -c 1 & echo $! > /sys/fs/cgroup/cpu-isolation/group_A/cgroup.procs
stress -c 1 & echo $! > /sys/fs/cgroup/cpu-isolation/group_B/cgroup.procs

# Верификация (PSR = номер ядра)
ps -e -o pid,psr,cmd | grep stress
# 5161  0  stress -c 1   <- group_A, ядро 0
# 5163  1  stress -c 1   <- group_B, ядро 1
```

Рис. 2. Закрепление процессов stress за ядрами CPU через cpuset (вывод ps -e -o pid,psr,cmd)

Результат: процесс group_A выполнялся исключительно на ядре 0 (загрузка 100%), группа group_B – на ядре 1 (100%). Процессы не конкурировали за процессорное время.

Количественная оценка стабильности изоляции через nr_switches

```
STRESS_PID=$(pgrep -n stress)
cat /proc/$STRESS_PID/schedstat
```

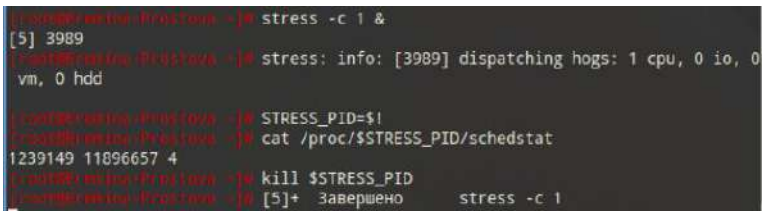

Рис. 3. Значение nr_switches=4 для процесса stress, изолированного через cpuset

Низкое значение nr_switches (4 переключения контекста за всё время жизни процесса) свидетельствует об отсутствии миграций между ядрами. Это подтверждает детерминированное выполнение на выделенном ядре.

Тестирование с изменением числа ядер

```
# Тест 1: одно ядро (ядро 0)
echo "0"   > /sys/fs/cgroup/cpu-isolation/test/cpuset.cpus
stress -c 1 & echo $! > /sys/fs/cgroup/cpu-isolation/test/cgroup.procs

# Тест 2: два ядра (0 и 1)
echo "0-1" > /sys/fs/cgroup/cpu-isolation/test/cpuset.cpus
stress -c 2 & echo $! > /sys/fs/cgroup/cpu-isolation/test/cgroup.procs
```

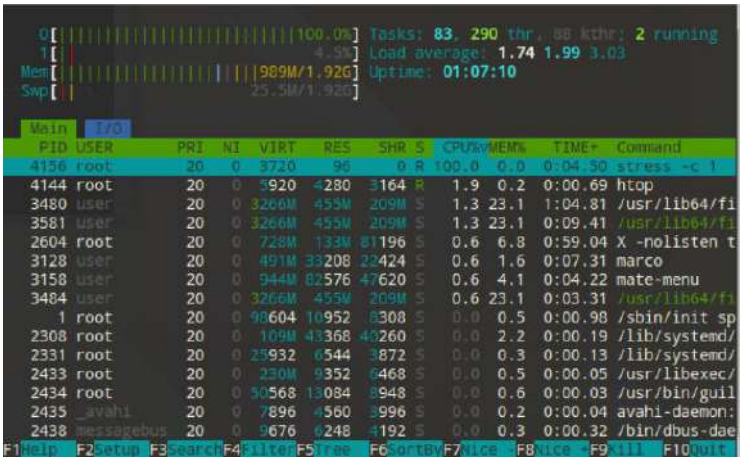

Рис. 4. htop при cpuset="0": ядро 0 загружено на 100%, ядро 1 свободно

Рис. 5. htop при cpuset="0-1": оба ядра загружены равномерно

Ограничение: CPU-контроллер (cpu.weight)

При попытке включить контроллер cpu командой echo "+cpu" > /sys/fs/cgroup/cgroup.subtree_control возникала ошибка «Отказано в доступе». Причиной является наличие realtime-процессов ядра вне корневой cgroup. Результат проверки представлен в таблице 1.

Таблица 1 – Realtime-процессы, препятствующие включению CPU-контроллера cgroups v2

PID	Класс планирования	RT-приоритет	Команда
15	SCHED_FIFO (FF)	99	[migration/0]

19	SCHED_FIFO (FF)	99	[migration/1]
41	SCHED_FIFO (FF)	50	[watchdog]

Согласно документации ядра [6], CPU-контроллер cgroups v2 не может быть включён при наличии RT-процессов (SCHED_FIFO/SCHED_RR) вне корневой cgroup, если ядро собрано с CONFIG_RT_GROUP_SCHED. Процессы migration и watchdog – системные realtime-процессы, которые не могут быть остановлены. Данное ограничение является задокументированным поведением и не является ошибкой.

Изоляция памяти через memory.max

Развертывание инфраструктуры для тестов памяти:

```
echo "+memory" > /sys/fs/cgroup/cgroup.subtree_control
mkdir -p /sys/fs/cgroup/mem-test/limited
```

Для экспериментов разработана тестовая программа на C, выделяющая заданный объём памяти с физическим заполнением (memset) для предотвращения lazy allocation. Заполнено в текстовом редакторе vim:

```
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
   int mb = argc > 1 ? atoi(argv[2]) : 100;
   char *p = malloc(mb * 1024 * 1024);
   if (p) { memset(p, 0, mb * 1024 * 1024); sleep(30); }
   return 0;
}

# Компиляция
gcc /tmp/oom_test.c -o /tmp/oom_test
```

Эксперимент 1: лимит 50 МБ, запрос 100 МБ (превышение лимита)

```
# Установка жесткого лимита 50 МБ + отключение swap
echo 52428800 > /sys/fs/cgroup/mem-test/limited/memory.max
echo 0        > /sys/fs/cgroup/mem-test/limited/memory.swap.max

# Запуск и помещение в группу
/tmp/oom_test 100 &
echo $! > /sys/fs/cgroup/mem-test/limited/cgroup.procs
sleep 5

# Анализ результатов
cat /sys/fs/cgroup/mem-test/limited/memory.current


dmesg | tail -5
```

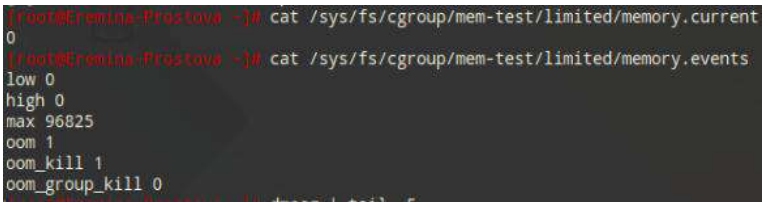

Рис. 6. memory.events: oom=1, oom_kill=1 – OOM Killer сработал при лимите 50 МБ

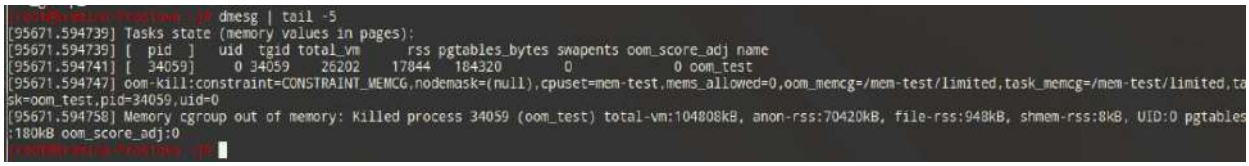

Рис. 7. dmesg: сообщение ядра о завершении процесса oom_test (pid=34059)

Аналогично провели еще два эксперимента. Второй – без лимита с штатным завершением, третий – с соблюдением лимита и штатным завершением. Результаты отмечены в таблице 3.

Сценарий смешанной нагрузки

```
# Создание изолированных групп для приоритетного и фонового процессов
mkdir /sys/fs/cgroup/cpu-isolation/priority
mkdir /sys/fs/cgroup/cpu-isolation/background
echo "0" > /sys/fs/cgroup/cpu-isolation/priority/cpuset.cpus
echo "1" > /sys/fs/cgroup/cpu-isolation/background/cpuset.cpus

# Приоритетная нагрузка (CPU) → ядро 0
stress -c 1 &
echo $! > /sys/fs/cgroup/cpu-isolation/priority/cgroup.procs

# Фоновая нагрузка (I/O) → ядро 1
dd if=/dev/urandom of=/dev/null &
echo $! > /sys/fs/cgroup/cpu-isolation/background/cgroup.procs
```

Рис. 8. htop: приоритетный stress на ядре 0 и фоновый dd на ядре 1 – без конкуренции

Результаты и обсуждение

Изоляция CPU (cpuset)

Контроллер cpuset обеспечивает жёсткую детерминированную изоляцию. Процессы группы не используют ядра, назначенные дргуим группам, несмотря на характер и интенсивность нагрузки. Значение PSR всегда соответствовало cpuset.cpus, а низкое значение nr_switches (4 переключения контекста) подтвердило отсутствие миграций. Следует отметить, что данный результат получен на двухъядерной системе. Применимость к многоядерным платформам и средам с NUMA требует отдельного экспериментального обоснования.

В отличие от cpu.weight (пропорциональное распределение при конкуренции за общие ядра), cpuset даёт абсолютную изоляцию. То есть, если ядро 0 простаивает, процесс из group_B не может его использовать, даже при перегрузке ядра 1. Это компромисс между гибкостью использования ресурсов и детерминизмом изоляции. Сводные результаты сценария смешанной нагрузки представлены в таблице 2.

Таблица 2 – Результаты сценария смешанной нагрузки

Процесс	Группа cgroup	Ядро CPU	Загрузка ядра	Конкуренция
stress -c 1	priority	0	100%	Отсутствует
dd (I/O)	background	1	100%	Отсутствует
stress -c 2 (контроль)	– (без cgroups)	0 или 1	~50% каждый	Присутствует

Результаты показали, что nr_switches процесса stress в группе priority составил 3 переключения контекста за период измерения, что сопоставимо с 4 при изолированном тесте. Это

подтверждает отсутствие конкуренции процесса stress с фоновым процессом dd за CPU и служит дополнительной верификацией в сценарии смешанной нагрузки.

Изоляция памяти (memory.max)

Серия экспериментов подтвердила детерминированность срабатывания OOM Killer при превышении жесткого лимита. Сводные результаты представлены в таблице 3.

Таблица 3 – Сводные результаты серии экспериментов на memory.max в cgroups v2

Эксперимент	Лимит	Запрос	Факт. выделено	Статус процесса	OOM Killer
Эксп. 1 (превышение)	50 МБ	100 МБ	~69 МБ	Завершён ядром	Активирован (oom_kill=1)
Эксп. 2 (без лимита)	max (нет)	300 МБ	300 МБ	Штатное завершение	Не активирован
Эксп. 3 (соблюдение)	500 МБ	400 МБ	~388 МБ	Штатное завершение	Не активирован (max=0)

Отключение swap (memory.swap.max = 0) обязательно, так как иначе процесс может превысить жёсткий лимит через файл подкачки, что делает невозможным детерминизм изоляции.

Подтверждение гипотез H1a и H1b

Гипотезы H1a и H1b полностью подтверждены. Для гипотезы H1a:

1. PSR всегда совпадал с cpuset.cpus (100% наблюдений);
2. nr_switches – 4 (изолированный тест) и 3 (смешанная нагрузка), что укладывается в критерий ≤ 20% отклонения.

Для гипотезы H1b:

1. при превышении лимита 50 МБ – oom_kill=1 в memory.events, подтверждено в dmesg;
2. процессы других групп не завершались принудительно. В контрольном эксперименте без cgroups процессы распределялись произвольно, что подтверждает необходимость детерминированных механизмов.

Соответствие требованиям регуляторов

Требования ФСТЭК России

Необходимо отметить, что в данной работе cgroups применяются без полноценной контейнерной изоляции (без namespace). Приказ ФСТЭК № 118 регулирует средства контейнеризации. Применение cgroups как самостоятельного механизма реализует требования о разделении ресурсов, но не заменяет полноценной namespace-изоляции. Полное соответствие требованиям достигается при совместном использовании cgroups v2 и namespaces. Приказ ФСТЭК России № 118 [13] устанавливает обязательное разделение процессов, обрабатывающих данные различных уровней конфиденциальности, с целью предотвращения несанкционированного доступа через разделяемые вычислительные ресурсы. Механизм cpuset реализует это требование. Например, процессы разных уровней доверия закрепляются за непересекающимися наборами ядер CPU. Это обеспечивает контролируемое закрепление процессов за наборами ядер в проверенной конфигурации, но не исключает все классы атак по сторонним каналам.

Требования PCI DSS 4.0

В контексте изоляции на уровне ядра релевантен также ГОСТ Р 59006-2020 «Защита информации. Доверенная загрузка» [14], регламентирующий контроль среды выполнения на уровне ядра ОС. Механизмы cgroups v2 могут рассматриваться как элемент обеспечения контролируемой среды выполнения, предусмотренной данным стандартом. Кроме того, управление вычислительными ресурсами регламентируется профилями защиты операционных систем в рамках оценочных уровней доверия (ОУД) по методологии ФСТЭК России, применяемой при сертификации средств защиты информации. Стандарт PCI DSS (версия 4.0, требования 6.3.3 и раздел A1 о многопользовательских средах) [15] требует защиты приложений, обрабатывающих данные платежных карт, от атак «исчерпание ресурсов» (resource exhaustion). Механизм memory.max обеспечивает эту защиту. Этот параметр задаёт жёсткий лимит. Он изолирует последствия утечки памяти или намеренного DoS в одном сервисе: OOM Killer завершает только процесс-нарушитель внутри его cgroup. Счетчик memory.events.max ведёт аудит

попыток исчерпания ресурсов, соответствуя требованиям PCI DSS к логированию инцидентов безопасности.

Заключение

Проведённые эксперименты подтверждают работоспособность и практическую пригодность механизмов cgroups v2 в составе ОС Альт. В качестве дальнейших шагов можно выделить измерение латентности с помощью lmbench или perf stat, сравнение поведения cpu.weight и cpuset, а также интеграцию cgroups v2 с пространствами имён для построения полноценной контейнерной изоляции.

Контроллер cpuset обеспечивает жёсткое закрепление процессов за выделенными ядрами CPU. Число переключений контекста nr_switches составило 4, миграции между ядрами отсутствуют. Контроллер memory.max также действует предсказуемо: при превышении лимита активируется OOM Killer (oom_kill=1), причём работа других групп не нарушается.

Гипотезы H1a и H1b полностью подтверждены. В рамках H1a процессы, ограниченные cpuset, ни при одном из приведенных сценариев не выполнялись на ядрах вне своего cpuset. В рамках H1b процессы, превысившие лимит memory.max, завершались OOM Killer до того, как нехватка памяти начинала влиять на остальные группы.

Таким образом, cgroups v2 имеет принципиальные преимущества перед традиционными механизмами (nice, cpulimit, ulimit) в средах с высокими требованиями к предсказуемости. Исследованные механизмы могут использоваться как часть более широкого комплекса мер для поддержки отдельных требований по разделению процессов и защите от исчерпания ресурсов; вывод о нормативном соответствии требует отдельной матрицы требований и процедуры оценки.

Практические рекомендации:

- cpuset необходим для жёсткого закрепления критических процессов за выделенными ядрами, особенно в системах реального времени и при обработке конфиденциальных данных;
- memory.max совместно с memory.swap.max=0 используется для детерминированного срабатывания OOM Killer и защиты от resource exhaustion;
- при планировании изоляции учитывать ограничение CONFIG_RT_GROUP_SCHED: CPU-контроллер (cpu.weight) недоступен при наличии RT-процессов ядра.

Список литературы

1. Носенко, Д. И. Сетевое и системное администрирование. Подготовка к Демонстрационному экзамену КОД 09.02.06-1-2025 : практикум / Д. И. Носенко, А. П. Золотарёв, А. Г. Уймин [и др.]. – Саратов : Профобразование, 2025. – 216 с. – ISBN 978-5-4488-2908-6. – URL: https://www.iprbookshop.ru/159510.html (дата обращения: 13.05.2025). – Текст : электронный.
2. Уймин, А. Г. Моделирование телекоммуникационной сети средствами сетевых инструментов Linux: инструменты создания цифровых двойников / А. Г. Уймин, О. Р. Никитин // I-methods. – 2023. – Т. 15, № 2. – EDN NFJDVH.
3. Nice and Renice Command in Linux [Электронный ресурс] // GeeksforGeeks. – 2025. – URL: https://www.geeksforgeeks.org/linux-unix/nice-and-renice-command-in-linux-with-examples/ (дата обращения: 14.05.2025). – Текст : электронный.
4. Cpulimit: CPU usage limiter for Linux [Электронный ресурс] // GitHub. – 2025. – URL: https://github.com/opsengine/cpulimit (дата обращения: 14.05.2025). – Текст : электронный.
5. Ulimit(3) – Linux manual page [Электронный ресурс] // man7.org. – 2025. – URL: https://man7.org/linux/man-pages/man3/ulimit.3.html (дата обращения: 14.05.2025). – Текст : электронный.
6. Control Group v2 – The Linux Kernel documentation [Электронный ресурс] // The Linux Kernel. – 2025. – URL: https://docs.kernel.org/admin-guide/cgroup-v2.html (дата обращения: 14.05.2025). – Текст : электронный.

7. Conway, J. Re: Getting out ahead of OOM [Электронный ресурс] / J. Conway // pgsql-admin Mailing List. – 2025. – URL: https://postgrespro.com/list/id/e52f00fc-ffe3-4745-8082-b492f44c6693@joeconway.com (дата обращения: 14.05.2025). – Текст : электронный.
8. Namespaces (7) – Linux manual page [Электронный ресурс] // man7.org. – 2024. – URL: https://man7.org/linux/man-pages/man7/namespaces.7.html (дата обращения: 14.05.2025). – Текст : электронный.
9. Wiedner, F. Control Groups Added Latency in NFVs: An Update Needed? / F. Wiedner, A. Daichendt, J. Andre, G. Carle // 2023 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN). – 2023. – P. 40–45. – DOI: 10.1109/NFV-SDN59219.2023.10329612.
10. systemd.resource-control — Resource Control Unit Settings [Электронный ресурс] // freedesktop.org. – 2025. – URL: https://www.freedesktop.org/software/systemd/man/latest/systemd.resource-control.html (дата обращения: 30.09.2025). – Текст : электронный.
11. Крайнов, П. А. Мониторинг системы контрольных групп процессов в ОС Альт / П. А. Крайнов, С. А. Пашина // Всероссийский сборник статей и публикаций института развития образования, повышения квалификации и переподготовки. – 2025. – URL: https://ropkip.ru/publication/415456 (дата обращения: 14.05.2025). – Текст : электронный.
12. ОС Альт – Документация [Электронный ресурс] // BaseALT. – 2025. – URL: https://docs.altlinux.org/ru-RU/index.html (дата обращения: 14.05.2025). – Текст : электронный.
13. Об утверждении Требований по безопасности информации к средствам контейнеризации : приказ ФСТЭК России от 4 июля 2022 г. № 118 [Электронный ресурс] // ГАРАНТ. – 2022. – URL: https://base.garant.ru/405955413/ (дата обращения: 14.05.2025). – Текст : электронный.
14. ГОСТ Р 59006-2020. Защита информации. Доверенная загрузка. Термины и определения [Электронный ресурс]. – М. : Стандартинформ, 2020. – URL: https://docs.cntd.ru/document/1200174521 (дата обращения: 13.05.2025). – Текст : электронный.
15. PCI Data Security Standard (PCI DSS) version 4.0.1 [Электронный ресурс] // PCI Security Standards Council. – 2024. – URL: https://www.pcisecuritystandards.org/document_library/ (дата обращения: 13.05.2025). – Текст : электронный.

References

1. Nosenko, D. I., Zolotarev, A. P., & Uymin, A. G. (2025). *Setevoe i sistemnoe administrirovanie. Podgotovka k Demonstratsionnomu ekzamenu KOD 09.02.06-1-2025* [Network and system administration: Practicum]. Profobrazovanie. ISBN 978-5-4488-2908-6. Retrieved May 13, 2025, from https://www.iprbookshop.ru/159510.html (accessed: 13.05.2025).
2. Uymin, A. G., & Nikitin, O. R. (2023). Modeling a telecommunication network using Linux network tools: Digital twin creation tools. *I-Methods*, *15*(2). EDN NFJDVH.
3. Nice and Renice Command in Linux. (2025). *GeeksforGeeks*. Retrieved May 14, 2025, from https://www.geeksforgeeks.org/linux-unix/nice-and-renice-command-in-linux-with-examples/ (accessed: 14.05.2025).
4. Cpulimit: CPU usage limiter for Linux. (2025). *GitHub*. Retrieved May 14, 2025, from https://github.com/opsengine/cpulimit (accessed: 13.05.2025).
5. Ulimit(3) – Linux manual page. (2025). *man7.org*. Retrieved May 14, 2025, from https://man7.org/linux/man-pages/man3/ulimit.3.html (accessed: 14.05.2025).
6. Control Group v2 – The Linux Kernel documentation. (2025). *The Linux Kernel*. Retrieved May 14, 2025, from https://docs.kernel.org/admin-guide/cgroup-v2.html (accessed: 14.05.2025).
7. Conway, J. (2025). Re: Getting out ahead of OOM. *pgsql-admin Mailing List*. Retrieved May 14, 2025, from https://postgrespro.com/list/id/e52f00fc-ffe3-4745-8082-b492f44c6693@joeconway.com (accessed: 14.05.2025).

8. Namespaces (7) – Linux manual page. (2024). *man7.org*. Retrieved May 14, 2025, from https://man7.org/linux/man-pages/man7/namespaces.7.html (accessed: 14.05.2025).
9. Wiedner, F., Daichendt, A., Andre, J., & Carle, G. (2023). Control groups added latency in NFVs: An update needed? In *2023 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)* (pp. 40–45). IEEE. DOI: 10.1109/NFV-SDN59219.2023.10329612.
10. freedesktop.org. (2025). systemd.resource-control — Resource control unit settings. Retrieved September 30, 2025, from https://www.freedesktop.org/software/systemd/man/latest/systemd.resource-control.html (accessed: 30.09.2025).
11. Kraynov, P. A., & Pashina, S. A. (2025). Monitoring the system of control groups of processes in the Alt OS. *All-Russian Collection of Articles and Publications of the Institute for Education Development*. Retrieved May 14, 2025, from https://ropkip.ru/publication/415456 (accessed: 14.05.2025).
12. ALT Linux – Documentation. (2025). *BaseALT*. Retrieved May 14, 2025, from https://docs.altlinux.org/ru-RU/index.html (accessed: 14.05.2025).
13. Federal Service for Technical and Export Control of Russia. (2022). *Ob utverzhdenii Trebovaniy po bezopasnosti informatsii k sredstvam konteynerizatsii* [On approval of Information Security Requirements for Containerization Tools] (Order No. 118). GARANT. Retrieved May 14, 2025, from https://base.garant.ru/405955413/ (accessed: 14.05.2025).
14. GOST R 59006-2020. (2020). *Zashchita informatsii. Doverennaya zagruzka. Terminy i opredeleniya* [Information protection. Trusted boot. Terms and definitions]. Standartinform. Retrieved May 14, 2025, from https://docs.cntd.ru/document/1200174521 (accessed: 13.05.2025).
15. PCI Security Standards Council. (2024). *PCI Data Security Standard (PCI DSS) version 4.0.1*. Retrieved May 14, 2025, from https://www.pcisecuritystandards.org/document_library/ (accessed: 13.05.2025).

Информация об авторах

Еремина Елизавета Алексеевна – студент ФГАОУ ВО «РГУ нефти и газа (НИУ) имени И.М. Губкина», г. Москва, Российская Федерация, e-mail: elizavetaeremina2@gmail.com

Простова Алиса Никитична – студент ФГАОУ ВО «РГУ нефти и газа (НИУ) имени И.М. Губкина», г. Москва, Российская Федерация, e-mail: prostrova2005@bk.ru

PROCESS ISOLATION IN THE ALT OPERATING SYSTEM USING CGROUPS V2 MECHANISMS

Eremina E. A.[1], Prostova A. N.[1]

[1]National University of Oil and Gas «Gubkin University»

Abstract. *The problem of computational resource management in multitasking operating systems is of particular importance in the context of widespread use of containerization environments and the need to simultaneously execute tasks with different resource requirements. Traditional Linux mechanisms – nice, cpulimit, and ulimit – do not provide comprehensive group isolation: nice sets only a recommended priority, cpulimit uses SIGSTOP/CONT signals, and ulimit restricts only the user session. The workload interference problem, known as the 'noisy neighbor' effect, remains a key challenge in system administration and information security. Purpose: experimental quantitative evaluation of the efficiency of cgroups v2 resource isolation mechanisms in the Alt Linux distribution (kernel 6.1.115). Methods. To verify hypotheses H1a (CPU isolation) and H1b (memory isolation), a series of controlled experiments was conducted using cpuset and memory.max controllers with varying workloads: 1–2 cores, memory limits of 50–500 MB, and a mixed scenario with priority and background processes. Measured metrics: PSR (CPU core affinity), nr_switches (context switches), memory.events (OOM Killer counters). Results. With cpuset, PSR matched the assigned core in 100% of observations; nr_switches was 3–4 without cross-group migration. When the memory.max limit was exceeded, the OOM Killer mechanism (oom_kill=1) was activated without affecting other groups. Conclusions. Hypotheses H1a and H1b are fully confirmed. Cgroups v2 demonstrates significant advantages over traditional isolation mechanisms. The mechanisms may support selected resource-separation and exhaustion-control requirements; however, the experiment alone does not demonstrate full compliance with FSTEC Order No. 118 or PCI DSS 4.0.*

Keywords: *cgroups v2; process isolation; cpuset; memory.max; resource management; containerization; information security.*

Information about the authors

Eremina Elizaveta Alekseevna – student Gubkin Russian State University of Oil and Gas (National Research University), Moscow, e-mail: elizavetaeremina2@gmail.com

Prostova Alisa Nikitichna – student Gubkin Russian State University of Oil and Gas (National Research University), Moscow, e-mail: prostrova2005@bk.ru