



★ 4.84

Оценка

1510.02

Рейтинг

Selectel

IT-инфраструктура для бизнеса



Andrei Yemelianov

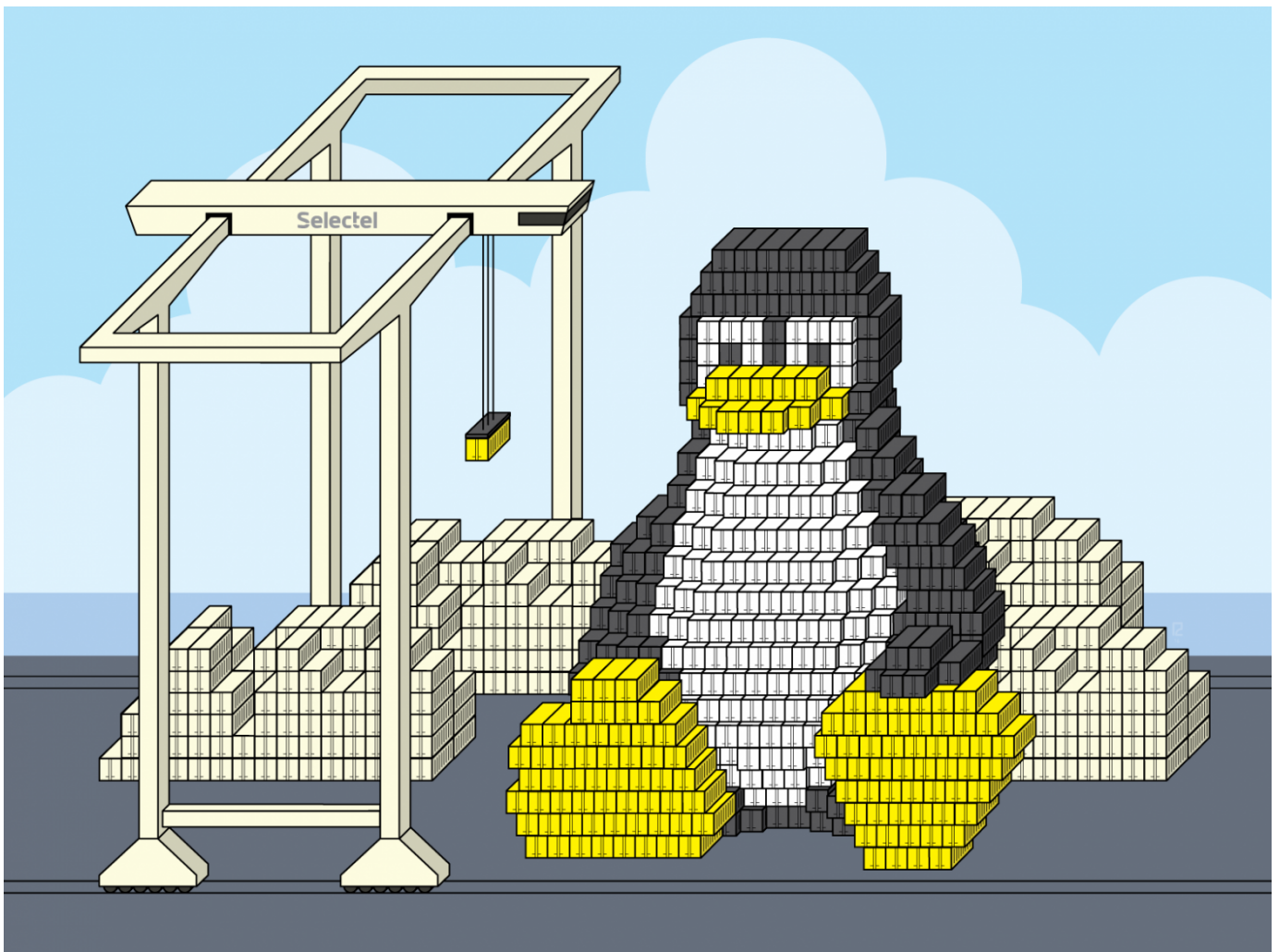
14 июн 2016 в 16:09

Механизмы контейнеризации: cgroups

🕒 11 мин 👁 88K

Блог компании Selectel

Тutorial



Продолжаем цикл статей о механизмах контейнеризации. В прошлый раз мы говорили об изоляции процессов с помощью механизма «пространств имён» (namespaces). Но для



либо приложение в изолированном окружении, мы должны быть уверены в том, что этому приложению выделено достаточно ресурсов и что оно не будет потреблять лишние ресурсы, нарушая тем самым работу остальной системы. Для решения этой задачи в ядре Linux имеется специальный механизм – cgroups (сокращение от control groups, контрольные группы). О нём мы расскажем в сегодняшней статье.

Тема cgroups сегодня особенно актуальна: в ядро версии 4.5, вышедшей в свет в январе текущего года, была официально добавлена новая версия этого механизма – group v2. В ходе работы над ней cgroups был по сути переписан заново.

Почему потребовались столь радикальные изменения? Чтобы ответить на этот вопрос, рассмотрим в деталях, как была реализована первая версия cgroups.

Cgroups: краткая история

Разработка cgroups была начата в 2006 году сотрудниками Google Полом Менеджем и Рохитом Сетом. Термин «контрольная группа» тогда ещё не использовался, а вместо него употреблялся термин «контейнеры процессов» (process containers). Собственно, сначала они и не ставили перед собой цели создать cgroups в современном понимании. Изначальный замысел был гораздо скромнее: усовершенствовать механизм cpuset, предназначенный для распределения процессорного времени и памяти между задачами. Но со временем всё переросло в более масштабный проект.

В конце 2007 года название process containers было заменено на control groups. Это было сделано, чтобы избежать разночтений в толковании термина «контейнер» (в то время уже активно развивался проект OpenVZ, и слово «контейнер» стало употребляться в новом, современном значении).

В 2008 году механизм cgroups был официально добавлен в ядро Linux (версия 2.6.24). Что нового появилось в этой версии ядра по сравнению с предыдущими?

Ни одного системного вызова, предназначенного специально для работы с cgroups, добавлено не было. В числе главных изменений следует назвать файловую систему cgroups, известную также под названием cgroupfs.

В init/main.c были добавлены отсылки к функциям для активации cgroups во время загрузки: cgroup_init и cgroup_init_early. Были незначительно изменены функции, используемые для порождения и завершения процесса – fork() и exit().

В виртуальной файловой системе /proc появились новые директории: /proc/{pid}/cgroup (для каждого процесса) и /proc/cgroups (для системы в целом).

Архитектура

Механизм cgroups состоит из двух составных частей: ядра (cgroup core) и так называемых подсистем. В ядре версии 4.4.0.21 таких подсистем 12:

- `blkio` – устанавливает лимиты на чтение и запись с блочных устройств;
- `cpuacct` – генерирует отчёты об использовании ресурсов процессора;
- `cpu` – обеспечивает доступ процессов в рамках контрольной группы к CPU;
- `cpuset` – распределяет задачи в рамках контрольной группы между процессорными ядрами;
- `devices` – разрешает или блокирует доступ к устройствам;
- `freezer` – приостанавливает и возобновляет выполнение задач в рамках контрольной группы;
- `hugetlb` – активирует поддержку больших страниц памяти для контрольных групп;
- `memory` – управляет выделением памяти для групп процессов;
- `net_cls` – помечает сетевые пакеты специальным тэгом, что позволяет идентифицировать пакеты, порождаемые определённой задачей в рамках контрольной группы;
- `netprio` – используется для динамической установки приоритетов по трафику;
- `pids` – используется для ограничения количества процессов в рамках контрольной группы.

Вывести список подсистем на консоль можно с помощью команды:

```
$ ls /sys/fs/cgroup/
```

```
blkio      cpu,cpuacct  freezer     net_cls      perf_event
cpu         cpuset       hugetlb     net_cls,net_prio  pids
cpuacct    devices      memory      net_prio     systemd
```

Каждая подсистема представляет собой директорию с управляющими файлами, в которых прописываются все настройки. В каждой из этих директорий имеются следующие управляющие файлы:

- `cgroup.clone_children` – позволяет передавать дочерним контрольным группам свойства родительских;
- `tasks` – содержит список PID всех процессов, включённых в контрольные группы;
`cgroup.procs` – содержит список TGID групп процессов, включённых в контрольные группы;
- `cgroup.event_control` – позволяет отправлять уведомления в случае изменения статуса контрольной группы;
- `release_agent` – содержится команда, которая будет выполнена, если включена опция `notify_on_release`. Может использоваться, например, для автоматического удаления пустых контрольных групп;
- `notify_on_release` – содержит булеву переменную (0 или 1), включающую (или наоборот отключающую), выполнение команды, указанной в `release_agent`.

У каждой подсистемы имеются также собственные управляющие файлы. О некоторых из них мы расскажем ниже.

Чтобы создать контрольную группу, достаточно создать вложенную директорию в любой из подсистем. В эту вложенную директорию будут автоматически добавлены управляющие файлы (ниже мы расскажем об этом более подробно). Добавить процессы в группу очень просто: нужно просто записать их PID в управляющий файл `tasks`.

Совокупность контрольных групп, встроенных в подсистему, называется иерархией. Попробуем разобрать принципы функционирования cgroups на простых практических примерах.

Иерархия cgroups: практическое знакомство

Пример 1: управление процессорными ресурсами

Выполним команду:

```
$ mkdir /sys/fs/cgroup/cpuset/group0
```

С помощью этой команды мы создали контрольную группу, в которой содержатся следующие управляющие файлы:

```
$ ls /sys/fs/cgroup/cpuset/group0

group.clone_children    cpuset.memory_pressure
cgroup.procs           cpuset.memory_spread_page
cpuset.cpu_exclusive   cpuset.memory_spread_slab
cpuset.cpus            cpuset.mems
cpuset.effective_cpus  cpuset.sched_load_balance
cpuset.effective_mems  cpuset.sched_relax_domain_level
cpuset.mem_exclusive   notify_on_release
cpuset.mem_hardwall    tasks
cpuset.memory_migrate
```

Пока что в нашей группе никаких процессов нет. Чтобы добавить процесс, нужно записать его PID в файл `tasks`, например:

```
$ echo $$ > /sys/fs/cgroup/cpuset/group0/tasks
```

Символами `$$` обозначается PID процесса, выполняемого текущей командной оболочкой.

Этот процесс не закреплён ни за одним ядром CPU, что подтверждает следующая команда:

```
$ cat /proc/$$/status |grep '_allowed'
Cpus_allowed: 2
Cpus_allowed_list: 0-1
Mems_allowed: 00000000,00000001
Mems_allowed_list: 0
```

Вывод этой команды показывает, что для интересующего нас процесса доступны 2 ядра CPU с номерами 0 и 1.

Попробуем «привязать» этот процесс к ядру с номером 0:

```
$ echo 0 >/sys/fs/cgroup/cpuset/group0/cpuset.cpus
```

Проверим, что получилось:

```
$ cat /proc/$$/status |grep '_allowed'  
Cpus_allowed: 1  
Cpus_allowed_list: 0  
Mems_allowed: 00000000,00000001  
Mems_allowed_list: 0
```

Пример 2: управление памятью

Встроим созданную в предыдущем примере группу ещё в одну подсистему:

```
$ mkdir /sys/fs/cgroup/memory/group0
```

Далее выполним:

```
$ echo $$ > /sys/fs/cgroup/memory/group0/tasks
```

Попробуем ограничить для контрольной группы group0 потребление памяти. Для этого нам понадобится прописать соответствующий лимит в файле memory.limit_in_bytes:

```
$ echo 40M > /sys/fs/cgroup/memory/group0/memory.limit_in_bytes
```

Механизм cgroups предоставляет очень обширные возможности управления памятью.

Например, с его помощью мы можем оградить критически важные процессы от попадания под

горячую руку OOM-killer'а:

```
$ echo 1 > /sys/fs/cgroup/memory/group0/memory.oom_control
$ cat /sys/fs/cgroup/memory/group0/memory.oom_control
oom_kill_disable 1
under_oom 0
```

Если мы поместим в отдельную контрольную группу, например, ssh-демон и отключим для этой группы OOM-killer, то мы можем быть уверены в том, что он не будет «убит» при преувеличении потребления памяти.

Пример 3: управление устройствами

Добавим нашу контрольную группу ещё в одну иерархию:

```
$ mkdir /sys/fs/cgroup/devices/group0
```

По умолчанию у группы нет никаких ограничений доступа к устройствам:

```
$ cat /sys/fs/cgroup/devices/group0/devices.list
a *:* rwm
```

Попробуем выставить ограничения:

```
$ echo 'c 1:3 rwm' > /sys/fs/cgroup/devices/group0/devices.deny
```

Эта команда включит устройство /dev/null в список запрещённых для нашей контрольной группы. Мы записали в управляющий файл строку вида 'с 1:3 rwm'. Сначала мы указываем тип устройства – в нашем случае это символьное устройство, обозначаемое буквой с (сокращение от character device). Два других типа устройств – это блочные (b) и все возможные устройства (a). Далее следуют мажорный и минорный номера устройства. Узнать

номера можно с помощью команды вида:

```
$ ls -l /dev/null
```

Вместо `/dev/null`, естественно, можно указать любой другой путь. Вывод этой команды выглядит так:

```
crw-rw-rw- 1 root root 1, 3 May 30 10:49 /dev/null
```

Первая цифра в выводе – это мажорный, а вторая – минорный номер.

Три последние буквы означают права доступа: `r` – разрешение читать файлы с указанного устройства, `w` – разрешение записывать на указанное устройство, `m` – разрешение создавать новые файлы устройств.

Далее выполним:

```
$ echo $$ > /sys/fs/cgroup/devices/group0/tasks  
$ echo "test" > /dev/null
```

При выполнении последней команды система выдаст сообщение об ошибке:

```
-bash: /dev/null: Operation not permitted
```

С устройством `/dev/null` мы никак взаимодействовать не можем, потому что доступ закрыт.

Восстановим доступ:

```
$ echo a > /sys/fs/cgroup/devices/group0/devices.allow
```


В результате выполнения этой команды в файл `/sys/fs/cgroup/devices/group0/devices.allow` будет добавлена запись `a *:* gwm`, и все ограничения будут сняты.

Cgroups и контейнеры

Из приведённых примеров понятно, в чём заключается принцип работы cgroups: мы помещаем определённые процессы в группу, которую затем «встраиваем» в подсистемы. Разберём теперь более сложные примеры и рассмотрим, как cgroups используются в современных инструментах контейнеризации на примере LXC.

Установим LXC и создадим контейнер:

```
$ sudo apt-get install lxc debootstrap bridge-utils
$ sudo lxc-create -n ubuntu -t ubuntu -f /usr/share/doc/lxc/examples/lxc-veth.cor
$ lxc-start -d -n ubuntu
```

Посмотрим, что изменилось в директории cgroups после создания и запуска контейнера:

```
$ ls /sys/fs/cgroup/memory

cgroup.clone_children  memory.limit_in_bytes      memory.swappiness
cgroup.event_control  memory.max_usage_in_bytes  memory.usage_in_bytes
cgroup.procs          memory.move_charge_at_immigrate memory.use_hierarchy
cgroup.sane_behavior  memory.numa_stat           notify_on_release
lxc                   memory.oom_control         release_agent
memory.failcnt        memory.pressure_level      tasks
memory.force_empty    memory.soft_limit_in_bytes
```

Как видим, в каждой иерархии появилась директория `lxc`, которая в свою очередь содержит поддиректорию `Ubuntu`. Для каждого нового контейнера в директории `lxc` будет создаваться отдельная поддиректория. PID всех запускаемых в этом контейнере процессов будут записываться в файл `/sys/fs/cgroup/cpu/lxc/[имя контейнера]/tasks`

Выделять ресурсы для контейнеров можно как с помощью управляющих файлов cgroups, так и с помощью специальных команд lxc, например:

```
$ lxc-cgroup -n [имя контейнера] memory.limit_in_bytes 400
```

Аналогичным образом дело обстоит с контейнерами Docker, systemd-nspawn и другими.

Недостатки cgroups

На протяжении почти 10 лет существования механизм cgroups неоднократно подвергался критике. Как отметил автор одной статьи на LWN.net, разработчики ядра cgroups активно не любят. Причины такой нелюбви можно понять даже из приведённых в этой статье примеров, хоть мы и старались подавать их максимально нейтрально, без эмоций: встраивать контрольную группу в каждую подсистему по отдельности очень неудобно. Присмотревшись повнимательней, мы увидим, что такой подход отличается крайней непоследовательностью.

Если мы, например, создаём вложенную контрольную группу, то в некоторых подсистемах настройки родительской группы наследуются, а в некоторых – нет.

В подсистеме cgroup любое изменение в родительской контрольной группе автоматически передаётся вложенным группам, а в других подсистемах такого нет и нужно активировать параметр clone.children.

Об устранении этих и других недостатков cgroups разговоры в сообществе разработчиков ядра шли очень давно: один из первых текстов на эту тему датируется началом 2012 года.

Автор этого текста, инженер Facebook Течжен Хе, прямо указал, что главная проблема cgroups заключается в неправильной организации, при которой подсистемы подключаются к многочисленным иерархиям контрольных групп. Он предложил использовать одну и только одну иерархию, а подсистемы добавлять для каждой группы отдельно. Такой подход повлёк за собой серьёзные изменения вплоть до смены названия: механизм изоляции ресурсов теперь называется cgroup (в единственном числе), а не cgroups.

Разберёмся более подробно в сути реализованных нововведений.

Cgroup v2: что нового

Как уже было отмечено выше, cgroup v2 был включён в ядро Linux начиная с версии ядра 4.5. При этом старая версия поддерживается тоже. Для версии 4.6 уже существует патч, с помощью которого можно отключить поддержку первой версии при загрузке ядра.

На текущий момент в cgroup v2 можно работать только с тремя подсистемами: blkio, memory и PID. Уже появились (пока что в тестовом варианте) патчи, позволяющие управлять ресурсами CPU.

Cgroup v2 монтируется при помощи следующей команды:

```
$ mount -t cgroup2 none [точка монтирования]
```

Предположим, мы смонтировали cgroup 2 в директорию /cgroup2. В этой директории будут автоматически созданы следующие управляющие файлы:

- `cgroup.controllers` – содержит список поддерживаемых подсистем;
- `cgroup.procs` – по завершении монтирования содержит список всех выполняемых процессов в системе, включая процессы-зомби. Если мы создадим группу, то для неё тоже будет создан такой файл; он будет пустым, пока в группу не добавлены процессы;
- `cgroup.subtree_control` – содержит список подсистем, активированных для данной контрольной группы; по умолчанию пуст.

Эти же самые файлы создаются в каждой новой контрольной группе. Также в группу добавляется файл `cgroup.events`, который в корневой директории отсутствует.

Новая группа создаётся так:

```
$ mkdir /cgroup2/group1
```

Чтобы добавить для группы подсистему, нужно записать имя этой подсистемы в файл

cgroup.subtree_control:

```
$ echo "+pid" > /cgroup2/group1/cgroup.subtree_control
```

Для удаления подсистемы используется аналогичная команда, только на место плюса ставится минус:

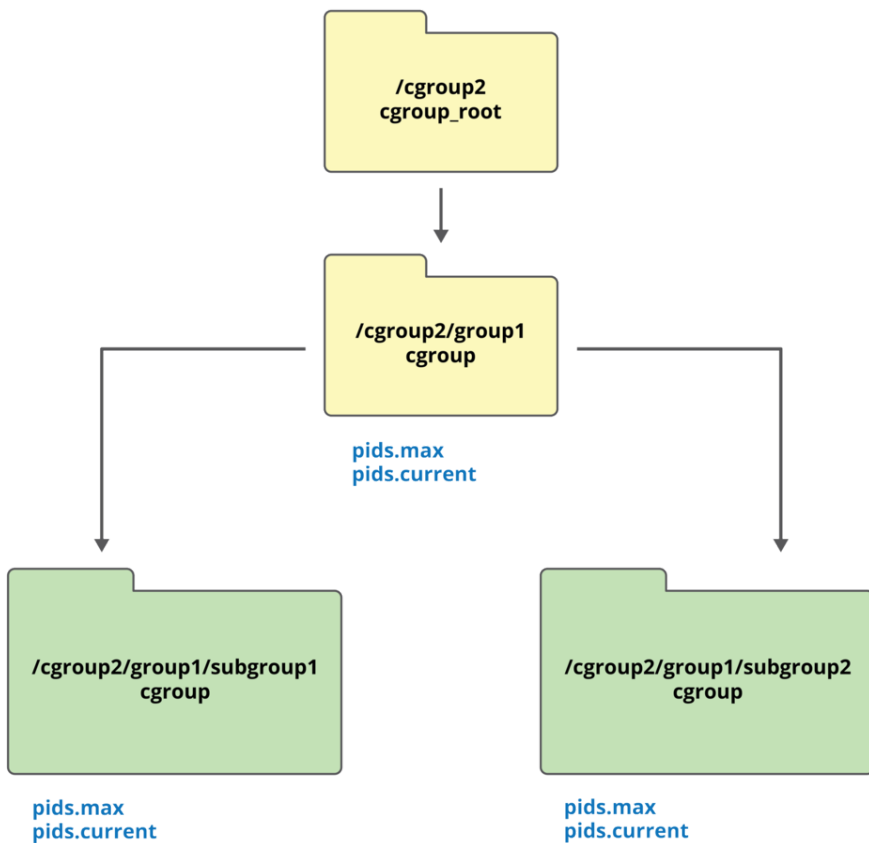
```
$ echo "-pid" > /cgroup2/group1/cgroup.subtree_control
```

Когда для группы активируется подсистема, в ней создаются дополнительные управляющие файлы. Например, после активации подсистемы PID в директории появятся файлы `pids.max` и `pids.current`. Первый из этих файлов используется для ограничения числа процессов в группе, а второй – содержит информацию о числе процессов, включённых в группу на текущий момент.

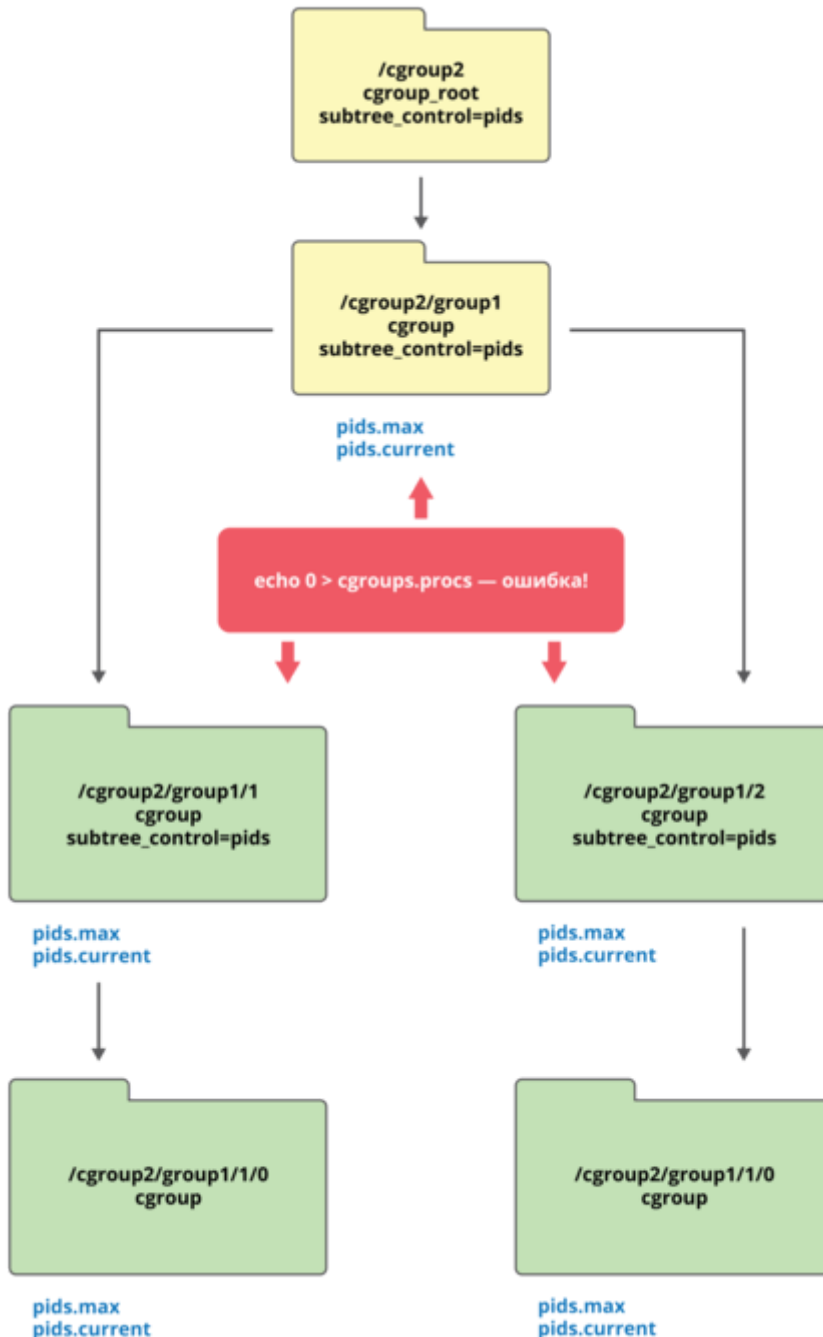
Внутри уже имеющихся групп можно создавать подгруппы:

```
$ mkdir /cgroup2/group1/subgroup1  
$ mkdir /cgroup2/group1/subgroup2  
$ echo "+memory" > /cgroup2/group1/cgroup.subtree_control,
```

Все подгруппы наследуют характеристики родительской группы. В только что приведённом примере подсистема PID будет активирована как для группы `group1`, так и для обеих вложенных в неё подгрупп; в них также будут добавлены файлы `pids.max` и `pids.current`. Сказанное можно проиллюстрировать с помощью схемы:



Чтобы избежать недоразумений с вложенными группами (см. выше), в cgroup v2 действует следующее правило: нельзя добавить процесс во вложенную группу, если в ней уже активирована какая-либо подсистема:



В первой версии cgroups процесс мог входить в несколько подгрупп одновременно, если эти подгруппы входили в разные иерархии, встроенные в разные подсистемы. Во второй версии один процесс может принадлежать только к одной подгруппе, что позволяет избежать путаницы.

Заключение

В этой статье мы рассказали, как устроен механизм cgroups и какие изменения были внесены в его новую версию. Если у вас есть вопросы и дополнения — добро пожаловать в комментарии.

Для всех, кто хочет глубже погрузиться в тему, приводим список ссылок на интересные материалы:

- <https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt> – документация первой версии cgroups;
- <https://www.kernel.org/doc/Documentation/cgroup-v2.txt> – документация cgroup v2;
- <https://www.youtube.com/watch?v=PzpG40WiEfM> – лекция Течжена Хе о нововведениях cgroup v2;
- <https://events.linuxfoundation.org/sites/events/files/slides/2014-KLF.pdf> – презентация доклада о cgroup v2 с подробными разъяснениями всех нововведений и изменений.

Если вы по тем или иным причинам не можете оставлять комментарии здесь, добро пожаловать в наш корпоративный блог.

Теги: cgroups, cgroup v2, linux, linux kernel, контейнеризация, контейнеры, selectel, селектел

Хабы: Блог компании Selectel

Редакторский дайджест



Присылаем лучшие статьи раз в месяц



Selectel

IT-инфраструктура для бизнеса

ВКонтакте Telegram



146

0

Карма Рейтинг

Андрей Емельянов @AndreiYemeljanov

Пользователь

 Комментарии 4

Публикации

ЛУЧШИЕ ЗА СУТКИ ПОХОЖИЕ



anton-artemenko

вчера в 14:01

Идеальные паразиты человека и «тихая пандемия»: привет, ветрянка и герпес



Простой



13 мин



12K

Обзор



+49



55



48

21 час назад

Герои напильника и паяльника: итоги сезона DIY



8 мин



8.2K



Сезон DIY

Спецпроект



+41



42



6



spiritus_sancti

22 часа назад

RGB-усилители. Особенности, проблемы, выбор



Простой



6 мин



5.4K

Тutorial



+39



24



11



ru_vds

18 часов назад

Профилирование Python — почему и где тормозит ваш код



Средний



10 мин



3.2K

Tutorial

Перевод

 +31 90 3

ViktorSergeev

15 часов назад

WebOne: даём жизнь старым браузерам

 6 мин  3.1K +27 26 8

AnnaVIMorozova

18 часов назад

Как повысить эффективность коммуникаций в команде: учимся решать конфликты

 7 мин  2.2K +19 43 2

it_union

1 час назад

ЗАО Гейм Инсайт Труп

 Простой  4 мин  2.5K +18 6 5

Yu-Leo

23 часа назад

Обзор электронной книги Meebook P10 Pro

 Простой  9 мин  6.2K[Обзор](#) +18 18 8

fedorborovitsky

вчера в 14:13

Российский софт в идеальном шторме

 6 мин  7.6K[Мнение](#)

29.08.2023, 14:59

Механизмы контейнеризации: cgroups / Хабр

+15

16

38

zatim

58 минут назад

Видеокарта VGA для микроконтроллера

Средний

17 мин

545

Тutorial

+13

8

3

Показать еще

ВАКАНСИИ КОМПАНИИ «SELECTEL»

QA Team Lead (web)

Selectel · Санкт-Петербург

Python engineer в команду Compute

Selectel · Можно удаленно

Практикант в инженерно-технических отдел

Selectel · Санкт-Петербург

QA Fullstack Engineer в команду разработки Выделенных серверов

Selectel · Можно удаленно

Python-разработчик в команду Биллинговой платформы

Selectel · Можно удаленно

Больше вакансий на Хабр Карьере

ИНФОРМАЦИЯ

Сайт

selectel.ru

Дата регистрации

16 марта 2010

Дата основания

11 сентября 2008

Численность

501–1 000 человек

Местоположение

Россия

https://habr.com/ru/companies/selectel/articles/303190/

18/23

Представитель

Ульяна Малышева

ССЫЛКИ

Выделенный сервер от 26 рублей в день

[selectel.ru](#)

Сервер для 3D - моделирования и рендеринга

[selectel.ru](#)

Физический сервер от 800 рублей в месяц

[selectel.ru](#)

Облачные серверы от 280 рублей в месяц

[selectel.ru](#)

FAQ

[slc.tl](#)

Реферальная программа

[slc.tl](#)

Telegram-канал о технологиях

[t.me](#)


Telegram-канал про карьеру в IT

[t.me](#)

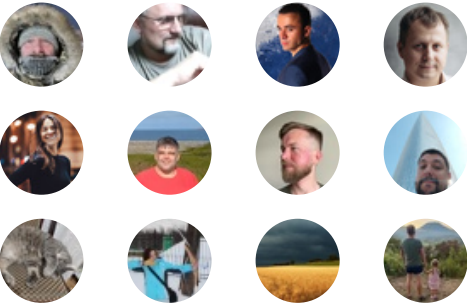
Вакансии

[slc.tl](#)

В КОНТАКТЕ

 Selectel

91 363 подписчика



Подписаться на новости

ВИДЖЕТ

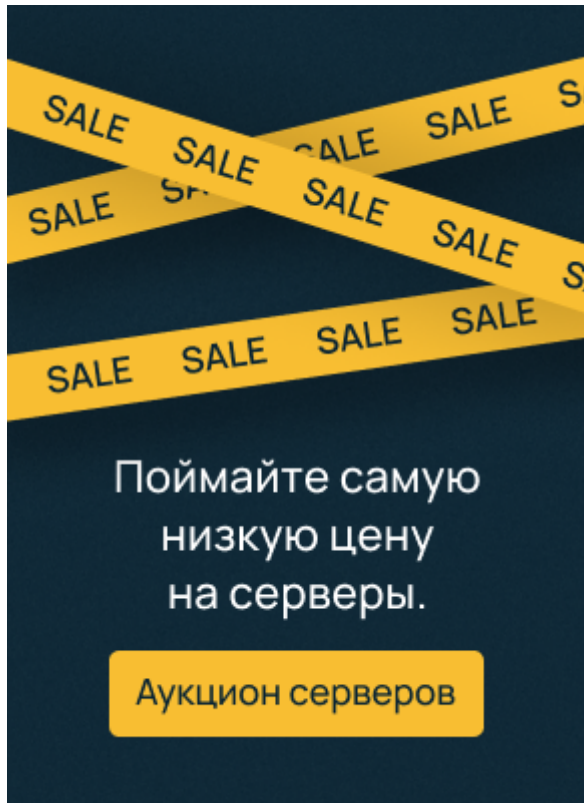
Selectel Career Wave

Стажировка в Backend
и DevOps для студентов
Санкт-Петербурга

Подай заявку до 10 сентября



ВИДЖЕТ



БЛОГ НА ХАБРЕ

5 часов назад

Всего два месяца — и новый релиз ядра Linux. Что появилось в ядре 6.5, что изменилось и что удалили. Новые возможности

 4.7K  4

вчера в 07:59

Гигабайт в мире мини-ПК: 96 ГБ ОЗУ и Intel Core i9-13900H в форм-факторе NUC

 8.5K  28

25 авг в 08:34

Новый ремонт Nintendo Switch Lite: как меня обманул продавец, но я все починил. Отвал процессора

 8.1K  45

24 авг в 18:07

MLOps от Gucci и оценка уровня Data Driven'ности в компании

 1.6K  0

23 авг в 17:31

Из Zero в Hero: как нетехническому специалисту работать со сложным продуктом

2.5K 1

Ваш аккаунт	Разделы	Информация	Услуги
Войти	Статьи	Устройство сайта	Корпоративный блог
Регистрация	Новости	Для авторов	Медийная реклама
	Хабы	Для компаний	Нативные проекты
	Компании	Документы	Образовательные
	Авторы	Соглашение	программы
	Песочница	Конфиденциальность	Стартапам
			Спецпроекты



Настройка языка

Техническая поддержка

Вернуться на старую версию