Red Hat Enterprise Linux 6

Управление системными ресурсами

Управление системными ресурсами в Red Hat Enterprise Linux 6
Редакция 1

Last Updated: 2017-10-13

Red Hat Enterprise Linux 6 Управление системными ресурсами

Управление системными ресурсами в Red Hat Enterprise Linux 6
Редакция 1

Martin Prpič
Отдел инженерной документации Red Hat
mprpic@redhat.com

Rüdiger Landmann
Отдел инженерной документации Red Hat
r.landmann@redhat.com

Douglas Silas
Отдел инженерной документации Red Hat
dhensley@redhat.com

Юридическое уведомление

Аннотация

Управление системными ресурсами в Red Hat Enterprise Linux 6

Содержание

ГЛАВА 1. ВВЕДЕНИЕ В КОНТРОЛЬНЫЕ ГРУППЫ

Red Hat Enterprise Linux 6 предоставляет новые возможности ядра — *контрольные группы* или *cgroups* (от англ. control groups). Контрольные группы позволяют распределять между процессами ресурсы — процессорное время, память, доступ к сети. Настройки существующих групп можно изменять динамически, запрещать и разрешать их доступ к ресурсам. С помощью **cgconfig** можно обеспечить сохранение конфигурации cgroups между перезагрузками.

Cgroups позволяют осуществлять тонкий контроль распределения, приоритизации и управления системными ресурсами. При этом ресурсы оборудования эффективно распределяются между пользователями и заданиями.

1.1. ОРГАНИЗАЦИЯ КОНТРОЛЬНЫХ ГРУПП

Подобно процессам, контрольные группы организованы иерархически, и дочерние группы наследуют атрибуты родительских. Существует две основных модели организации групп (см. ниже).

Модель процессов Linux

Все процессы в Linux являются дочерними по отношению к **init**, который запускается во время загрузки системы и запускает другие процессы. Таким образом, иерархия процессов Linux представляет собой дерево с единственным корнем.

Все процессы за исключением **init** наследуют настройки окружения (переменную PATH и пр.) [1] и другие атрибуты родительского процесса (дескрипторы файлов и пр.).

Модель контрольных групп

Основные сходства контрольных групп и процессов:

- организованы в виде иерархии;
- дочерние группы выборочно наследуют атрибуты родительской группы.

Основное отличие заключается в том, что в системе одновременно может существовать множество независимых иерархий контрольных групп.

Каждая иерархия может соответствовать одной или нескольким *подсистемам*. Подсистема[2] определяет отдельный тип ресурса, например процессорное время или память. Всего в Red Hat Enterprise Linux 6 есть 9 подсистем (см. ниже).

Подсистемы в Red Hat Enterprise Linux

- **blkio**: ограничение ввода-вывода блочных устройств (дисков, USB и т.п.).
- **cpu**: использует планировщик для управления доступом к процессору.
- **cpuacct**: генерирует отчеты об использовании процессорных ресурсов.
- **cpuset**: отвечает за выделение процессоров и узлов памяти в многопроцессорных системах .
- **devices**: отвечает за управление доступом заданий к устройствам.
- **freezer**: останавливает и возобновляет работу заданий контрольной группы.

- **memory**: накладывает ограничения и генерирует отчеты об использовании памяти.
- **net_cls**: позволяет присвоить сетевым пакетам идентификатор класса (classid), который помогает контроллеру **tc** идентифицировать пакеты, поступающие из заданной контрольной группы.
- **ns**: подсистема пространства имен.

ПРИМЕЧАНИЕ

Возможно, вы уже сталкивались с терминами *контроллер* и *контроллер ресурсов* в документации ядра. Оба термина являются синонимами «подсистемы».

Само определение подсистемы довольно расплывчато — это механизм управления группами процессов.

1.2. СВЯЗИ МЕЖДУ ПОДСИСТЕМАМИ, ИЕРАРХИЯМИ, КОНТРОЛЬНЫМИ ГРУППАМИ И ЗАДАЧАМИ

Системные процессы в терминологии cgroup называются задачами.

Далее приведены правила связей между подсистемами, иерархиями и задачами.

Правило 1

Отдельная подсистема может быть сопоставлена только одной иерархии.

*Например, подсистема **cpu** не может быть одновременно подключена к двум разным иерархиям.*

Правило 2

С одной иерархией может быть связано несколько подсистем.

*Например, **cpu** и **memory** могут быть подключены к одной и той же иерархии при условии, что ни одна из подсистем не связана с другой иерархией.*

Правило 3

При создании новой иерархии все системные задачи по умолчанию будут назначены контрольной группе этой иерархии — так называемой *корневой группе*. Каждое задание в созданной иерархии может принадлежать *только одной* группе в этой иерархии. Задача может принадлежать различным группам, если они расположены в разных иерархиях. Поэтому как только задача входит в состав новой группы в этой же иерархии, она автоматически удаляется из исходной группы.

*Так, например, если подсистемы **cpu** и **memory** назначены иерархии **cpu_and_mem**, а подсистема **net_cls** — иерархии **net**, то процесс **httpd** может принадлежать одной группе в **cpu_and_mem** и одной в **net**.*

*Группа в **cpu_and_mem**, которой принадлежит процесс **httpd**, может сократить процессорное время, выделяемое другим процессам, до половины, а память — до **1024** МБ. Группа в **net**, в состав которой **httpd** также входит, может ограничить скорость передачи **30** мегабайтами в секунду.*

При создании новой иерархии все системные задачи по умолчанию будут назначены корневой группе. То есть любая системная задача принадлежит как минимум одной группе.

Правило 4

Если процесс создает дочерние процессы, они автоматически войдут в состав групп, которым принадлежит родительский процесс. Впоследствии дочерние процессы можно перенести в другие группы.

Представим, например, процесс ***httpd****, входящий в состав группы* ***half_cpu_1gb_max*** *в иерархии* ***cpu_and_mem*** *и группы* ***trans_rate_30*** *в иерархии* ***net****. Порождаемые им процессы будут автоматически принадлежать* ***half_cpu_1gb_max*** *и* ***trans_rate_30****.*

Дочерний процесс лишь изначально наследует группы родительского процесса и впоследствии не зависит от него. То есть изменение группы одного процесса никак не скажется на другом.

1.3. ОСОБЕННОСТИ РАБОТЫ С КОНТРОЛЬНЫМИ ГРУППАМИ

- Так как задача может принадлежать лишь одной группе в иерархии, для ее управления будет использоваться всего одна подсистема.
- Несколько подсистем можно сгруппировать так, чтобы они контролировали все задачи в заданной иерархии. Так как настройки групп в этой иерархии могут отличаться, результаты могут отличаться.
- Структура иерархии может меняться. Например, может потребоваться исключить подсистему, к которой подключено несколько других подсистем, и добавить ее в другую иерархию.
- И наоборот, если необходимость в разделении подсистем отпала, можно удалить одну иерархию и добавить ее подсистемы в другую.
- Можно изменять параметры отдельных задач в иерархии, используя подсистему в качестве критерия выбора.
- Возможна и более тонкая настройка задач. Так, например, каждая задача может состоять в разных иерархиях, к каждой из которых подключена всего одна подсистема. В этом случае администратор сможет одновременно управлять всеми настройками процесса.

---

[1] Родительский процесс может изменять настройки окружения до передачи их дочернему процессу.

[2] В документации и на справочных страницахlibcgroup подсистемы также называются *контроллерами ресурсов* или просто *контроллерами*.

ГЛАВА 2. РАБОТА С КОНТРОЛЬНЫМИ ГРУППАМИ

Управление контрольными группами осуществляется с помощью инструментов из пакета libcgroup. В принципе, параметры групп можно изменить на время текущего сеанса с помощью стандартных команд оболочки, но libcgroup значительно упрощает этот процесс и предоставляет больше возможностей. В данном руководстве рассматривается настройка групп с использованием libcgroup, но приводятся и эквивалентные команды оболочки.

ПРИМЕЧАНИЕ

В режиме root:

```
~]# yum install libcgroup
```

2.1. СЛУЖБА CGCONFIG

cgconfig устанавливается вместе с libcgroup и рекомендуется для создания иерархий, их связи с подсистемами и управления группами.

В Red Hat Enterprise Linux 6 **cgconfig** не запускается по умолчанию. При ее запуске с помощью **chkconfig** будет прочитан файл конфигурации **/etc/cgconfig.conf**. Каждый раз при запуске группы будут создаваться заново и таким образом сохранять постоянство. Исходя из настроек в **/etc/cgconfig.conf**, **cgconfig** может создавать иерархии, подключать файловые системы, создавать контрольные группы и настраивать подсистемы для каждой группы.

В исходном **/etc/cgconfig.conf** определены настройки для создания и подключения иерархии для каждой подсистемы, и для связи подсистем с этими иерархиями.

При остановке **cgconfig** (команда **service cgconfig stop**) все иерархии будут отключены.

2.1.1. cgconfig.conf

Файл **cgconfig.conf** содержит два типа записей — *mount* и *group*. В секции mount создаются иерархии, которые затем подключаются как виртуальные файловые системы. Созданным иерархиям могут быть назначены подсистемы. Формат:

```
mount {
<подсистема> = <путь>;
…
}
```

См. Пример 2.1, «Создание секции mount».

Пример 2.1. Создание секции mount

Пример создания иерархии для подсистемы **cpuset**:

```
mount {
    cpuset = /cgroup/cpu;
}
```

что эквивалентно командам:

```
~]# mkdir /cgroup/cpu
~]# mount -t cgroup -o cpu cpu /cgroup/cpu
```

Записи *group* содержат определения групп и параметры подсистем. Например:

```
group <имя> {
    [<права>]
    <подсистема> {
        <параметр> = <значение>;
        …
    }
    …
}
```

Секция прав доступа не является обязательной. Ее формат:

```
perm {
    task {
        uid = <пользователь>;
        gid = <группа>;
    }
    admin {
       uid = <администратор>;
       gid = <админ_группа>;
    }
}
```

См. Пример 2.2, «Создание секции group».

Пример 2.2. Создание секции group

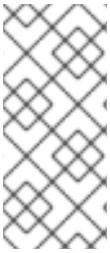
В следующей секции создается контрольная группа для служб sql. Пользователям в группе **sqladmin** разрешается добавлять задачи в группу, а пользователю **root** разрешается изменять параметры подсистем.

```
group daemons/sql {
    perm {
        task {
            uid = root;
            gid = sqladmin;
        } admin {
            uid = root;
            gid = root;
        }
    } cpu {
        cpu.shares = 100;
    }
}
```

Эквивалентный набор команд оболочки в комбинации с секцией mount (см. Пример 2.1, «Создание секции mount») будет выглядеть так:

```
~]# mkdir -p /cgroup/cpu/daemons/sql
```

```
~]# chown root:root /cgroup/cpu/daemons/sql/*
~]# chown root:sqladmin /cgroup/cpu/daemons/sql/tasks
~]# echo 100 > /cgroup/cpu/daemons/sql/cpu.shares
```



ПРИМЕЧАНИЕ

Чтобы изменения в **/etc/cgconfig.conf** вступили в силу, надо перезапустить **cgconfig**:

```
~]# service cgconfig restart
```

При установке libcgroup будет создан шаблон файла **/etc/cgconfig.conf**. Знаки # в начале строки отделяют комментарии, которые **cgconfig** будет игнорировать.

2.2. СОЗДАНИЕ ИЕРАРХИИ И ПОДКЛЮЧЕНИЕ ПОДСИСТЕМ

ПРЕДУПРЕЖДЕНИЕ

Приведенные ниже инструкции подразумевают, что контрольные группы еще не настроены. Изменение параметров в существующих группах может повлиять на производительность задач в их составе.

Если в системе уже настроены группы (вручную или с помощью **cgconfig**), приведенные здесь команды завершатся неудачей, так как для их работы необходимо отключить существующие иерархии, что повлияет на производительность системы. Не рекомендуется экспериментировать с этими настройками в критически важных системах.

Чтобы создать иерархию и подключить к ней подсистемы, надо добавить соответствующие записи в секцию **mount** в **/etc/cgconfig.conf** (в режиме root). Формат:

```
подсистема = /cgroup/иерархия;
```

При следующем запуске **cgconfig** будет создана иерархия и подключены подсистемы.

Далее будет создана иерархия **cpu_and_mem**, к которой будут подключены подсистемы **cpu**, **cpuset**, **cpuacct**, **memory**.

```
mount {
    cpuset  = /cgroup/cpu_and_mem;
    cpu     = /cgroup/cpu_and_mem;
    cpuacct = /cgroup/cpu_and_mem;
    memory  = /cgroup/cpu_and_mem;
}
```

Создание иерархии в оболочке

Иерархии могут быть созданы с помощью команд оболочки.

В режиме root создайте каталог с именем контрольной группы, который будет служить точкой подключения иерархии:

```
~]# mkdir /cgroup/имя
```

Например:

```
~]# mkdir /cgroup/cpu_and_mem
```

Подключите иерархию и добавьте в нее подсистемы:

```
~]# mount -t cgroup -o подсистемы иерархия /cgroup/иерархия
```

Здесь *подсистемы* — список подсистем, разделенных запятой (см. Подсистемы в Red Hat Enterprise Linux и Глава 3, *Подсистемы и настраиваемые параметры*).

Пример 2.3. Подключение подсистем с помощью mount

В следующем примере в качестве точки подключения иерархии будет использоваться существующий каталог **/cgroup/cpu_and_mem**. К иерархии с именем **cpu_and_mem** будут подключены подсистемы **cpu**, **cpuset** и **memory**. Наконец, **cpu_and_mem** будет подключена в **/cgroup/cpu_and_mem**:

```
~]# mount -t cgroup -o cpu,cpuset,memory cpu_and_mem /cgroup/cpu_and_mem
```

Список доступных подсистем и соответствующих точек подключения можно получить с помощью команды **lssubsys** [3]:

```
~]# lssubsys -am
cpu,cpuset,memory /cgroup/cpu_and_mem
net_cls
ns
cpuacct
devices
freezer
blkio
```

Исходя из этого, можно сделать следующие выводы:

- иерархии **/cgroup/cpu_and_mem** назначены подсистемы **cpu**, **cpuset**, **memory**;
- подсистемы **net_cls**, **ns**, **cpuacct**, **devices**, **freezer** и **blkio** еще не принадлежат никаким иерархиям, о чем свидетельствует отсутствие соответствующих точек подключения.

2.3. ДОБАВЛЕНИЕ И УДАЛЕНИЕ ПОДСИСТЕМ ИЗ ИЕРАРХИИ

Чтобы добавить, удалить или переместить подсистему в другую иерархию, надо внести изменения в секцию **mount** в **/etc/cgconfig.conf** (см. Раздел 2.2, «Создание иерархии и

подключение подсистем»). Изменения вступят в силу после перезапуска **cgconfig**.

Создание иерархии в оболочке

Чтобы добавить независимую подсистему в иерархию, сначала нужно отключить иерархию. После этого можно добавить подсистему в команду **mount** и указать параметр **remount**.

Пример 2.4. Отключение иерархии для добавления подсистемы

Приведенная команда показывает, что с иерархией **cpu_and_mem** связаны подсистемы **cpu**, **cpuset** и **memory**.

```
~]# lssubsys -am
cpu,cpuset,memory /cgroup/cpu_and_mem
net_cls
ns
cpuacct
devices
freezer
blkio
```

Заново подключим **cpu_and_mem**, указав параметр **remount**, и добавим **cpuacct** в список подсистем:

```
~]# mount -t cgroup -o remount,cpu,cpuset,cpuacct,memory cpu_and_mem
/cgroup/cpu_and_mem
```

Вывод **lssubsys** теперь выглядит так:

```
~]# lssubsys -am
cpu,cpuacct,cpuset,memory /cgroup/cpu_and_mem
net_cls
ns
devices
freezer
blkio
```

Аналогичным образом можно удалить подсистему из иерархии. Нужно просто повторно подключить иерархию, намеренно опустив имя подсистемы после аргумента "-o". Команда отключения подсистемы **cpuacct**:

```
~]# mount -t cgroup -o remount,cpu,cpuset,memory cpu_and_mem
/cgroup/cpu_and_mem
```

2.4. ОТКЛЮЧЕНИЕ ИЕРАРХИИ

Для отключения иерархии из файловой системы используется **umount**:

```
~]# umount /cgroup/имя
```

Например:

```
~]# umount /cgroup/cpu_and_mem
```

Пустая иерархия (содержащая только корневую контрольную группу) будет автоматически отключена после отсоединения из файловой системы. Если же она содержит другие группы, она останется активной на уровне ядра.

Прежде чем приступить к удалению иерархии, надо удалить все подчиненные группы, и уже затем отключить иерархию из файловой системы. В противном случае можно принудительно отключить иерархию с помощью **cgclear** (см. Раздел 2.11, «Удаление контрольных групп»).

2.5. СОЗДАНИЕ КОНТРОЛЬНЫХ ГРУПП

Формат команды создания групп: **cgcreate -t *uid*:*gid* -a *uid*:*gid* -g *подсистемы*:*путь***.

- **-t** (необязательный): определяет пользователя (по UID) и группу (по GID), которым будет принадлежать псевдофайл **tasks** создаваемой группы. Пользователь сможет добавлять и удалять задания из группы.

  ПРИМЕЧАНИЕ

  Единственный метод удаления задач из группы заключается в их перемещении в другую группу. При этом пользователь должен обладать правами записи во вторую группу.

- **-a** (необязательный): определяет пользователя (по UID) и группу (по GID), которым будут принадлежать все псевдофайлы группы кроме **tasks**. Этот пользователь сможет изменять уровень доступа заданий к ресурсам.

- **-g** (необзятельный): разделенный запятой список подсистем, определяющих иерархии, которым будет принадлежать группа. Список завершается двоеточием, после которого следует путь к группе. Точка подключения иерархии не указывается.

  К примеру, группа в **/cgroup/cpu_and_mem/lab1/** будет обозначена как **lab1**, так как ее путь уже известен в силу того, что заданной подсистеме соответствует всего одна иерархия. Группа контролируется всеми подсистемами в указанных иерархиях, даже если некоторые подсистемы не были напрямую перечислены в строке **cgcreate** (см. Пример 2.5, «Пример cgcreate»).

Так как контрольные группы в иерархии используют одни и те же контроллеры, дочерние группы тоже их унаследуют.

Пример 2.5. Пример cgcreate

Представим, что подсистемы **cpu** и **memory** подключены в иерархию **cpu_and_mem**, а **net_cls** — в **net**.

```
~]# cgcreate -g cpu,net_cls:/test-subgroup
```

cgcreate создаст две группы с именем **test-subgroup** — одну в иерархии **cpu_and_mem**, а вторую в **net**. Первая группа будет также находиться под контролем подсистемы **memory**, хотя это не указано явно.

Создание иерархии в оболочке

Чтобы создать дочернюю группу, используйте команду **mkdir**:

```
~]# mkdir /cgroup/иерархия/имя/новая_группа
```

Например:

```
~]# mkdir /cgroup/cpuset/lab1/group1
```

2.6. УДАЛЕНИЕ КОНТРОЛЬНЫХ ГРУПП

Формат команды удаления групп: **cgdelete *подсистемы*:*путь***.

- *подсистемы* — список подсистем, разделенных запятой;
- *путь* — путь к контрольной группе из текущей иерархии.

Например:

```
~]# cgdelete cpu,net_cls:/test-subgroup
```

-r осуществляет рекурсивное удаление подгрупп.

При удалении группы ее задания будут перемещены в родительскую группу.

2.7. НАСТРОЙКА ПАРАМЕТРОВ

При наличии соответствующих прав пользователь может изменять параметры подсистемы с помощью **cgset**. К примеру, чтобы указать процессоры, к которым у группы должен быть доступ:

```
cpuset]# cgset -r cpuset.cpus=0-1 group1
```

Формат команды: **cgset -r *параметр=значение путь***.

- *параметр* — устанавливаемый параметр, которому соответствует файл в каталоге группы;
- *значение* — присвоенное параметру значение;
- *путь* — путь к группе из корня иерархии. Например, команда настройки параметра корневой группы **/cgroup/cpuacct/**, будет выглядеть так:

  ```
  cpuacct]# cgset -r cpuacct.usage=0 /
  ```

  Корневая группа также может быть обозначена как **.**, то есть команда может выглядеть так:

  ```
  cpuacct]# cgset -r cpuacct.usage=0 .
  ```

  Рекомендуется использовать первый вариант команды.

ПРИМЕЧАНИЕ

В корневой группе можно настроить лишь ограниченное число параметров (включая приведенный выше **cpuacct.usage**). Это объясняется тем, что корневой группе принадлежат все ресурсы, поэтому изменение настроек на этом уровне ограничит все существующие процессы, что нецелесообразно.

Чтобы изменить параметр в **group1**, которая входит в состав корневой группы:

```
cpuacct]# cgset -r cpuacct.usage=0 group1
```

Добавление "/" после названия группы (**cpuacct.usage=0 group1/**) необязательно.

Значения параметров в группах также зависят от того, какие значения заданы на верхних уровнях иерархии. К примеру, если **group1** доступен только процессор CPU 0, то **group1/subgroup1** тоже будет доступен только этот процессор.

С помощью **cgset** можно копировать параметры одной группы в другую:

```
~]# cgset --copy-from group1/ group2/
```

Формат: **cgset --copy-from *путь1 путь2***.

- *путь1* — путь к исходной контрольной группе;
- *путь2* — путь к группе-получателю.

Прежде чем приступить к копированию, надо убедиться, что необходимые параметры подсистем установлены, иначе команда не сможет завершить работу (см. Обязательные параметры).

Создание иерархии в оболочке

Параметры можно добавить в псевдофайл подсистемы напрямую с помощью **echo**. Ниже приведен пример добавления значения **0-1** в **cpuset.cpus** контрольной группы **group1**:

```
~]# echo 0-1 > /cgroup/cpuset/group1/cpuset.cpus
```

В результате заданиям этой контрольной группы будет разрешено обращаться только к процессорам 0 и 1.

2.8. ПЕРЕНОС ПРОЦЕССА В КОНТРОЛЬНУЮ ГРУППУ

Процесс можно переместить с помощью команды **cgclassify**:

```
~]# cgclassify -g cpu,memory:group1 1701
```

Формат: **cgclassify -g *подсистемы*:*путь PID***.

- *подсистемы* — список подсистем, разделенных запятой. Если указать *****, то будут выбраны все подсистемы, которым принадлежат указанные процессы. Если в разных иерархиях есть группы с одним и тем же именем, **-g** переместит процессы в каждую группу.
- *путь* — путь к группе из корня иерархии.

- *PID* — список идентификаторов процессов, разделенных запятой.

Дополнительный аргумент **-- sticky** перед идентификатором процесса оставляет дочерние процессы в текущей контрольной группе. Если аргумент не указан, и в то же время выполняется **cgred**, дочерние процессы будут переназначены в соответствии с настройками в **/etc/cgrules.conf**. Сам процесс будет оставаться в исходной контрольной группе.

С помощью **cgclassify** можно сразу переместить несколько процессов. Следующая команда переместит процессы с PID **1701** и **1138** в группу **group1**:

```
~]# cgclassify -g cpu,memory:group1 1701 1138
```

Идентификаторы процессов разделяются пробелом, а указанные группы должны принадлежать разным иерархиям.

Создание иерархии в оболочке

Чтобы напрямую переместить процесс в группу, запишите его *PID* в файл **tasks** этой группы. Команда переноса процесса **1701** в **/cgroup/lab1/group1/** будет выглядеть так:

```
~]# echo 1701 > /cgroup/lab1/group1/tasks
```

2.8.1. cgred

Служба **cgred** отвечает за перенос заданий в контрольные группы на основе заданных в **/etc/cgrules.conf** параметров. Записи в **/etc/cgrules.conf** могут следовать следующим форматам:

- *пользователь иерархии контрольная_группа*
- *пользователь*:*команда иерархии контрольная_группа*

Например:

```
maria   devices  /usergroup/staff
```

В этом примере все процессы, принадлежащие пользователю **maria**, будут обращаться к подсистеме **devices** в соответствии с заданными для группы **/usergroup/staff** параметрами. Чтобы сопоставить отдельные команды контрольным группам, укажите их после имени пользователя:

```
maria:ftp  devices  /usergroup/staff/ftp
```

То есть если пользователь **maria** выполняет команду **ftp**, процесс будет автоматически перемещен в группу **/usergroup/staff/ftp** в иерархии, содержащей подсистему **devices**. Стоит отметить, что перенос процесса будет осуществлен только при выполнении указанного условия, поэтому может оказаться так, что **ftp** некоторое время будет выполняться не в той группе. Более того, если процесс создает другие процессы во время выполнения в другой группе, нет гарантии, что дочерние процессы будут корректно перемещены.

Записи в **/etc/cgrules.conf** могут включать дополнительные выражения:

- @ перед именем пользователя обозначает не отдельного пользователя, а группу. Например, **@admins** включает всех пользователей в группе **admins**.

- ***** охватывает все компоненты. Так, ***** в поле подсистем обозначает все подсистемы.
- **%** копирует элементы из предыдущей строки. Например:

```
@adminstaff  devices  /admingroup
@labstaff  %  %
```

2.9. ЗАПУСК ПРОЦЕССА В КОНТРОЛЬНОЙ ГРУППЕ

ВАЖНО

Для некоторых подсистем существуют обязательные параметры, которые необходимо установить, прежде чем приступать к переносу заданий. Так, например, прежде чем перенести задание в группу, использующую подсистему **cpuset**, необходимо определить значения **cpuset.cpus** и **cpuset.mems**.

Приведенные в этой секции примеры будут работать при условии, что заданы обязательные параметры для используемых контроллеров.

Раздел 3.10, «Дополнительные ресурсы» содержит перечень параметров.

Для запуска процессов в контрольной группе используется **cgexec**. Ниже будет запущен браузер **lynx** в пределах группы **group1**. Этот процесс унаследует ограничения доступа, определенные подсистемой **cpu** для этой группы:

```
~]# cgexec -g cpu:group1 lynx http://www.redhat.com
```

Формат: **cgexec -g *подсистемы*:*путь* *команда* *аргументы***.

- *подсистемы* — список подсистем, разделенных запятой. Если указать *****, будут выбраны все подсистемы, которым принадлежат указанные процессы. Если в разных иерархиях есть группы с одним и тем же именем, **-g** создаст процессы в каждой группе.
- *путь* — путь к группе из корня иерархии.
- *команда* — выполняемая команда.
- *аргументы* — аргументы команды.

Дополнительный аргумент **--sticky** перед командой оставляет дочерние процессы в текущей группе. Если аргумент не указан, и в то же время выполняется **cgred**, дочерние процессы будут переназначены в соответствии с настройками в **/etc/cgrules.conf**. Сам процесс будет оставаться в исходной контрольной группе.

Создание иерархии в оболочке

При запуске нового процесса он унаследует группу создавшего его процесса. Поэтому чтобы запустить процесс в выбранной группе, можно перенести в эту группу процесс оболочки (см. Раздел 2.8, «Перенос процесса в контрольную группу») и уже затем запустить процесс из этой оболочки.

```
~]# echo $$ > /cgroup/lab1/group1/tasks
lynx
```

После выхода из **lynx** оболочка все еще будет состоять в группе **group1**. Поэтому лучше предпочесть следующий метод:

```
~]# sh -c "echo \$$ > /cgroup/lab1/group1/tasks && lynx"
```

2.9.1. Запуск службы в контрольной группе

Требования к запускаемым в группе службам:

- они должны использовать файл **/etc/sysconfig/*имя_службы***;
- для запуска должны использовать функцию **daemon()** из **/etc/init.d/functions**.

Чтобы запустить службу в контрольной группе, добавьте запись **CGROUP_DAEMON="*подсистема*:*группа*"** в ее файл **/etc/sysconfig**. Пример:

```
CGROUP_DAEMON="cpuset:daemons/sql"
```

2.10. ПОЛУЧЕНИЕ ИНФОРМАЦИИ

2.10.1. Поиск процессов

Команда поиска контрольной группы, которой принадлежит процесс:

```
~]$ ps -O cgroup
```

Если известен PID процесса, можно выполнить следующее:

```
~]$ cat /proc/PID/cgroup
```

2.10.2. Поиск подсистем

Чтобы выполнить поиск подсистем, доступных в ядре, и получить информацию об их связи с иерархиями:

```
~]$ cat /proc/cgroups
```

Следующая команда поможет найти точки подключения подсистем:

```
~]$ lssubsys -m подсистемы
```

подсистемы — разделенный запятой список подсистем. **lssubsys -m** вернет только верхние точки подключения иерархий.

2.10.3. Поиск иерархий

Иерархии рекомендуется подключать в каталог **/cgroup**. Поэтому вывод содержимого этого каталога покажет список иерархий. Если в системе установлена программа **tree**, с ее помощью можно получить список иерархий и их групп:

```
~]$ tree /cgroup/
```

2.10.4. Поиск контрольных групп

Получение списка групп:

```
~]$ lscgroup
```

Можно показать результаты для определенной иерархии. Для этого в строке команды укажите ***подсистема*:*путь***. Пример:

```
~]$ lscgroup cpuset:adminusers
```

В результате будут показаны подгруппы в составе группы **adminusers**, принадлежащей иерархии, к которой подключена подсистема **cpuset**.

2.10.5. Просмотр параметров контрольных групп

Чтобы получить список параметров выбранных групп, выполните

```
~]$ cgget -r параметр группы
```

где *параметр* — псевдофайл с настройками подсистемы, а *группы* — список групп, разделенных пробелами.

```
~]$ cgget -r cpuset.cpus -r memory.limit_in_bytes lab1 lab2
```

Эта команда покажет значения **cpuset.cpus** и **memory.limit_in_bytes** для групп **lab1** и **lab2**.

Если точные названия параметров неизвестны, можно выполнить следующее:

```
~]$ cgget -g cpuset /
```

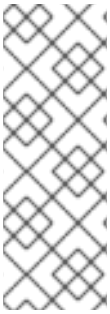
2.11. УДАЛЕНИЕ КОНТРОЛЬНЫХ ГРУПП

ПРЕДУПРЕЖДЕНИЕ

Для удаления всех групп в иерархии используется команда **cgclear**. Если структура иерархии не сохранена в файле конфигурации, восстановить ее будет непросто.

С помощью **cgclear** можно полностью очистить файловую систему контрольных групп.

Все задания групп будут перемещены на верхний уровень иерархии, сами группы будут удалены, а файловая система будет отключена. Наконец, будет удален каталог, в который была подключена файловая система контрольной группы.



ПРИМЕЧАНИЕ

Если группы были созданы с помощью команды **mount**, то в файлах **/etc/mtab** и **/proc/mounts** были созданы соответствующие записи. При удалении групп с помощью **cgclear** изменения будут отражены только в **/proc/mounts**. Файл **/etc/mtab** не будет изменен, поэтому результат выполнения команды **mount** будет содержать удаленные группы. Актуальный список групп можно найти в **/proc/mounts**.

2.12. ДОПОЛНИТЕЛЬНЫЕ РЕСУРСЫ

Информацию о командах управления контрольными группами можно найти на справочных страницах libcgroup.

Справочные страницы

- **man 1 cgclassify**: с помощью **cgclassify** осуществляется перемещение выполняемых заданий между группами.

  man 1 cgclear: команда **cgclear** удаляет все группы в иерархии.

  man 5 cgconfig.conf: файл **cgconfig.conf** содержит определения групп.

  man 8 cgconfigparser: команда **cgconfigparser** осуществляет разбор файла **cgconfig.conf** и подключает иерархии в файловую систему.

  man 1 cgcreate: команда **cgcreate** создает группы в иерархиях.

  man 1 cgdelete: команда **cgdelete** удаляет группы.

  man 1 cgexec: команда **cgexec** запускает задания в группах.

  man 1 cgget: команда **cgget** показывает список параметров группы.

  man 5 cgred.conf: справочная страница файла конфигурации службы **cgred**.

  man 5 cgrules.conf: файл **cgrules.conf** содержит правила, помогающие определить принадлежность заданий группам.

  man 8 cgrulesengd: справочная страница службы **cgrulesengd**, которая распределяет задания между группами.

  man 1 cgset: справочная страница команды настройки контрольных групп.

  man 1 lscgroup: справочная страница команды просмотра групп в иерархии.

  man 1 lssubsys: справочная страница команды просмотра иерархий, содержащих заданные подсистемы.

[3] **lssubsys** входит в состав пакета libcgroup (см. Глава 2, *Работа с контрольными группами*).

ГЛАВА 3. ПОДСИСТЕМЫ И НАСТРАИВАЕМЫЕ ПАРАМЕТРЫ

Подсистемы представляют собой модули ядра, отвечающие за выделение системных ресурсов контрольным группам. Можно отдельно запрограммировать подсистемы с целью создания индивидуального подхода к управлению группами процессов. *Интерфейс программирования приложений* (API, Application Programming Interface) для разработки новых подсистем задокументирован в файле **cgroups.txt** в каталоге **/usr/share/doc/kernel-doc-*версия_ядра*/Documentation/cgroups/**. Последнюю версию документа можно найти на http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt. Стоит помнить, что описание новых функций может не совпадать с установленным ядром.

Объекты состояния содержат параметры подсистем для контрольной группы и представлены в виде *псевдофайлов* в виртуальной файловой системе. Управление псевдофайлами осуществляется с помощью команд оболочки или эквивалентных системных вызовов. Например, если в **cpuset.cpus** определены процессоры, к которым разрешен доступ контрольной группы, а **/cgroup/cpuset/webserver** — контрольная группа веб-сервера, то команда

```
~]# echo 0,2 > /cgroup/cpuset/webserver/cpuset.cpus
```

запишет **0,2** в **cpuset.cpus**, тем самым разрешив заданиям, PID которых перечислены в **/cgroup/cpuset/webserver/tasks**, использовать процессоры 0 и 2.

3.1. BLKIO

blkio (Block I/O) — подсистема управления вводом-выводом блочных устройств. Для ограничения доступа следует записать значения в соответствующие псевдофайлы, а их чтение позволяет получить интересующую информацию.

blkio.weight

Определяет относительный вес (от **100** до **1000**) ввода-вывода контрольной группы. Для отдельных устройств это значение можно переопределить величиной **blkio.weight_device**. Пример присвоения значения **500**:

```
echo 500 > blkio.weight
```

blkio.weight_device

Определяет относительный вес (от **100** до **1000**) ввода-вывода для конкретного устройства, доступного контрольной группе. Этот параметр переопределяет **blkio.weight**. Формат: *старший_номер*:*младший_номер вес*. Номера устройств определены в *списке устройств Linux* (см. http://www.kernel.org/doc/Documentation/devices.txt). Так, команда присвоения веса **500** для доступа к **/dev/sda** выглядит так:

```
echo 8:0 500 > blkio.weight_device
```

В списке устройств Linux номера **8:0** соответствуют **/dev/sda**.

blkio.time

Возвращает время доступа ввода-вывода к заданным устройствам. Каждая запись содержит *старший_номер*, *младший_номер* и *время*. Номера устройств определены в *списке устройств Linux*, а *время* указывается в миллисекундах.

blkio.sectors

Возвращает число перемещаемых между устройствами секторов. Запись содержит *старший_номер*, *младший_номер* и *число_секторов*. Номера устройств определены в *списке устройств Linux*.

blkio.io_service_bytes

Возвращает число байт, переносимых между устройствами. Запись содержит *старший_номер*, *младший_номер*, *операция* и *число_байт*. Номера устройств определены в *списке устройств Linux*, а *операция* может принимать значения **read**, **write**, **sync**, **async**.

blkio.io_serviced

Возвращает число операций ввода-вывода для указанных устройств. Запись содержит *старший_номер*, *младший_номер*, \n*операция* и *число_операций*. Номера устройств определены в *списке устройств Linux*, а *операция* может принимать значения **read**, **write**, **sync**, **async**.

blkio.io_service_time

Возвращает время между выдачей запроса ввода-вывода и его завершением. Запись содержит *старший_номер*, *младший_номер*, *операция* и *время*. Номера устройств определены в *списке устройств Linux*, *операция* может принимать значения **read**, **write**, **sync**, **async**, а *время* указывается в наносекундах.

blkio.io_wait_time

Возвращает время ожидания операций ввода-вывода. Некоторые моменты следует отметить отдельно:

- Время ожидания может превышать общее время, так как суммируется время ожидания всех операций ввода-вывода для выбранной группы. Для определения времени ожидания группы используется **blkio.group_wait_time**.
- Если **queue_depth** > 1, результат будет включать только время до отправки запроса устройству. Время ожидания переорганизации запросов устройством учитываться не будет.

Запись содержит поля *старший_номер*, *младший_номер*, *операция* и *время*. Номера устройств определены в *списке устройств Linux*, *операция* может принимать значения **read**, **write**, **sync**, **async**, а *время* указывается в наносекундах.

blkio.io_merged

Возвращает число запросов BIOS, объединенных с другими запросами ввода-вывода. Записи содержат поля *число_запросов* и *операция*. Параметр *операция* может принимать значения **read**, **write**, **sync**, **async**.

blkio.io_queued

Возвращает число запросов в очереди ввода-вывода группы. Записи содержат поля *число_запросов* и *операция*. Параметр *операция* может принимать значения **read**, **write**, **sync**, **async**.

blkio.avg_queue_size

Возвращает средний размер очереди ввода-вывода за время существования контрольной группы. Длина очереди проверяется каждый раз, когда группе предоставляется рабочее

время. Эти данные будут доступны, только если определена переменная **CONFIG_DEBUG_BLK_CGROUP=y**.

blkio.group_wait_time

Возвращает общее время (в наносекундах), которое группа провела в ожидании времени обслуживания. Результат обновляется каждый раз, когда группе предоставляется рабочее время. Поэтому если группе было выделено время, после того как вы открыли файл для чтения, будет доступен лишь предыдущий результат. Данные будут доступны, только если определена переменная **CONFIG_DEBUG_BLK_CGROUP=y**.

blkio.empty_time

Возвращает общее время (в наносекундах), которое группа провела без ожидающих запросов. Результат обновляется каждый раз при появлении ожидающего запроса. Данные будут доступны, только если определена переменная **CONFIG_DEBUG_BLK_CGROUP=y**.

blkio.idle_time

Возвращает общее время (в наносекундах), которое планировщик провел в ожидании более подходящего запроса по сравнению с запросами в других очередях или из других групп. Результат обновляется каждый раз, когда группа выходит из состояния бездействия. Поэтому если открыть файл в период бездействия группы, информация о последнем состоянии бездействия будет недоступна. Данные будут доступны, только если определена переменная **CONFIG_DEBUG_BLK_CGROUP=y**.

blkio.dequeue

Сообщает, сколько раз запросы ввода-вывода удалялись устройствами из очереди. Запись содержит *старший_номер*, *младший_номер* и *число_запросов*. Номера устройств определены в *списке устройств Linux*. Эти данные будут доступны, только если определена переменная **CONFIG_DEBUG_BLK_CGROUP=y**.

blkio.reset_stats

Обнуляет статистику в других псевдофайлах.

3.2. CPU

Подсистема **cpu** отвечает за управление доступом контрольных групп к процессорам. Доступ предоставляется в зависимости от перечисленных ниже параметров. Каждый параметр хранится в отдельном *псевдофайле* в виртуальной файловой системе контрольной группы:

cpu.shares

Целое значение, определяющее относительную величину доступного заданиям процессорного времени. Например, задания в двух контрольных группах, для которых **cpu.shares** имеет значение **1**, получат равное время доступа к процессорам, а задания в группе, для которой **cpu.shares** имеет значение **2**, получат в два раза больше процессорного времени.

cpu.rt_runtime_us

Определяет максимальный период времени (в микросекундах), в течение которого задания в контрольной группе могут использовать процессорные ресурсы. Такое ограничение позволяет предотвратить монопольное использование ресурсов одной контрольной группой.

Например, если заданиям в контрольной группе необходимо предоставить периодический доступ длиной 4 секунды каждые 5 секунд, установите значение **cpu.rt_runtime_us** в **4000000**, а **cpu.rt_period_us** в **5000000**.

cpu.rt_period_us

Определяет интервал (в микросекундах), по истечении которого контрольная группа повторно получит доступ к процессорным ресурсам. Например, если заданиям в контрольной группе необходимо предоставить периодический доступ длиной 4 секунды каждые 5 секунд, установите значение **cpu.rt_runtime_us** в **4000000**, а **cpu.rt_period_us** в **5000000**.

3.3. CPUACCT

Подсистема **cpuacct** создает отчеты о занятости процессорных ресурсов. Существует три типа отчетов:

cpuacct.stat

Возвращает число циклов процессора (в единицах, заданных с помощью **USER_HZ**), затраченных на обработку заданий контрольной группы в пользовательском и системном режиме.

cpuacct.usage

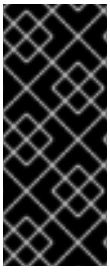
Возвращает суммарное время (в наносекундах), в течение которого процессорные ресурсы были заняты обработкой заданий контрольной группы (включая задания на низких уровнях иерархии).

cpuacct.usage_percpu

Возвращает время (в наносекундах), в течение которого ресурсы каждого процессора были заняты обработкой всех заданий контрольной группы (включая задания на низких уровнях иерархии).

3.4. CPUSET

Подсистема **cpuset** выделяет процессоры и узлы памяти контрольным группам. Ниже перечислены параметры cpuset. Каждый параметр хранится в отдельном *псевдофайле* в виртуальной файловой системе контрольной группы:



ВАЖНО

Для некоторых подсистем существуют обязательные параметры, которые необходимо определить, прежде чем приступать к переносу заданий в контрольную группу. Так, например, чтобы перенести задание в группу, использующую **cpuset**, необходимо определить значения **cpuset.cpus** и **cpuset.mems**.

cpuset.cpus (обязательный)

Определяет процессоры, к которым могут обращаться задания в группе. Представляет собой список значений ASCII, разделенных запятой. Для обозначения диапазона используется дефис. Пример:

```
0-2,16
```

Здесь перечислены процессоры 0, 1, 2 и 16.

cpuset.mems (обязательный)

Определяет узлы памяти, к которым могут обращаться задания в группе. Представляет собой список значений ASCII, разделенных запятой. Для обозначения диапазона используется дефис. Пример:

```
0-2,16
```

Здесь перечислены узлы памяти 0, 1, 2 и 16.

cpuset.memory_migrate

Флаг (**0** или **1**), который вызывает перенос страницы памяти на другой узел при изменении значений в **cpuset.mems**. По умолчанию эта функциональность отключена (**0**) и страницы остаются на исходном узле, даже если узел не включен в список в **cpuset.mems**. Если же флаг установлен (**1**), страницы будут перенесены на узлы памяти, заданные в **cpuset.mems**. При этом по возможности будет поддерживаться их относительное размещение; так, например, страницы со второго узла в исходном списке будут перенесены на второй узел в обновленном списке **cpuset.mems**.

cpuset.cpu_exclusive

Флаг (**0** или **1**), позволяющий совместно использовать назначенный заданному набору cpuset процессор другими cpuset. По умолчанию процессоры не ограничены одним набором cpuset (**0**).

cpuset.mem_exclusive

Флаг (**0** или **1**), позволяющий совместно использовать назначенные заданному набору cpuset узлы памяти другими cpuset. По умолчанию узлы памяти не ограничены одним набором cpuset (**0**). Их резервирование для одного набора cpuset (**1**) эквивалентно установке флага **cpuset.mem_hardwall**.

cpuset.mem_hardwall

Флаг (**0** или **1**), позволяющий ограничить выделение страниц памяти и данных буфера узлами памяти, заданными в текущем наборе cpuset. По умолчанию (**0**) страницы и данные буфера могут совместно использоваться процессами, принадлежащими разным пользователям. Установка флага (**1**) отделяет выделение заданий для конкретного пользователя от других.

cpuset.memory_pressure

Этот файл доступен только для чтения и содержит среднее значение *нагрузки памяти* в результате выполнения процессов, соответствующих текущему набору cpuset. Если флаг **cpuset.memory_pressure_enabled** отключен (**0**), то значение в этом файле равно нулю, а при активации **cpuset.memory_pressure_enabled** оно будет рассчитано автоматически.

cpuset.memory_pressure_enabled

Флаг (**0** или **1**), отвечающий за расчет *нагрузки памяти*, вызванной работой процессов в составе заданной контрольной группы. Значения рассчитываются умножением числа попыток возвращения занятой памяти в секунду на 1000 и характеризуют скорость освобождения

процессами используемой памяти. Результаты сохраняются в файл **cpuset.memory_pressure**

cpuset.memory_spread_page

Флаг (**0** или **1**), позволяющий равномерно распределить буферы файловой системы между узлами памяти для заданного cpuset. По умолчанию распределение выполняться не будет (**0**), поэтому буферы будут помещаться на тот же узел, где выполняется создавший их процесс.

cpuset.memory_spread_slab

Флаг (**0** или **1**), отвечающий за равномерное распределение slab-блоков кэша для операций ввода-вывода в пределах текущего набора cpuset. По умолчанию распределение выполняться не будет (**0**), поэтому блоки кэша будут помещаться на тот же узел, где выполняется создавший их процесс.

cpuset.sched_load_balance

Флаг (**0** или **1**), отвечающий за распределение нагрузки между процессорами в составе заданного cpuset. По умолчанию (**1**) ядро распределяет нагрузку посредством переноса процессов на менее загруженные процессоры.

Установка этого флага в контрольной группе не возымеет эффекта, если в родительской группе включено распределение нагрузки. Следует его отключить во всех родительских группах в иерархии, а также оценить, нужно ли распределение в контрольных группах на том же уровне иерархии.

cpuset.sched_relax_domain_level

Содержит целое значение, начиная с **-1** и заканчивая небольшим позитивным числом, определяющее диапазон процессоров, между которыми будет распределяться нагрузка. Не имеет эффекта, если отключен флаг **cpuset.sched_load_balance**.

В таблице перечислены типичные значения, хотя действия могут отличаться в зависимости от архитектуры.

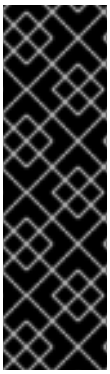
Значения cpuset.sched_relax_domain_level

Значение	Действие
-1	Использовать исходное системное значение для распределения нагрузки
0	Не выполнять распределение нагрузки сразу, а делать это периодически
1	Сразу перераспределить нагрузку между потоками одного ядра
2	Сразу перераспределить нагрузку между ядрами одного пакета
3	Сразу перераспределить нагрузку между процессорами на одном узле или blade-модуле

Значение	Действие
4	Сразу перераспределить нагрузку между несколькими процессорами в системах с неоднородным доступом к памяти (NUMA)
5	Сразу перераспределить нагрузку между всеми процессорами в системах с NUMA

3.5. DEVICES

Подсистема **devices** отвечает за управление доступом заданий контрольной группы к устройствам.



ВАЖНО

Подсистема **devices** включена в Red Hat Enterprise Linux 6 в роли экспериментальной версии.

Предварительный выпуск включает функции, которые в настоящее время не поддерживаются службой подписки Red Hat Enterprise Linux 6, предоставляются в ознакомительных целях и не рекомендуются для использования в критически важных системах. Отзывы и пожелания будут приниматься до тех пор, пока не будет выпущена полнофункциональная версия.

devices.allow

Задает устройства, к которым разрешен доступ заданий. Каждая запись содержит четыре поля: *тип*, *старший_номер*, *младший_номер*, *доступ*. Значения типа и номеров определены в *списке устройств Linux* (см.http://www.kernel.org/doc/Documentation/devices.txt).

тип

Допустимые значения типа:

- **a** применимо ко всем *символьным* и *блочным* устройствам;
- **b** — блочное устройство;
- **c** — символьное устройство.

старший_номер, младший_номер

Старший и *младший* номера разделены двоеточием и идентифицируют устройство Linux. Например, **8:1** (где **8** — номер, соответствующий дискам SCSI, а **1** обозначает первый раздел на первом диске SCSI) соответствует разделу **/dev/sda1**.

"*****" используется в качестве шаблона. Например, **9:*** обозначает устройства RAID, а ***:*** — все устройства.

доступ

Допустимые значения:

- **r** разрешает осуществлять чтение из заданного устройства;
- **w** разрешает осуществлять запись на заданные устройства;
- **m** разрешает создавать файлы устройств, если они не существуют.

К примеру, если задано значение **r**, задания смогут выполнять чтение с заданного устройства, а **rw** разрешит и чтение, и запись.

devices.deny

Устройства, к которым запрещен доступ заданий. Формат записей аналогичен **devices.allow**.

devices.list

Возвращает устройства, для которых было настроено управление доступом заданий.

3.6. FREEZER

Подсистема **freezer** отвечает за остановку и возобновление заданий контрольной группы.

freezer.state

Допустимые значения:

- **FROZEN**: задания приостановлены;
- **FREEZING**: система в стадии приостановки заданий;
- **THAWED**: работа заданий возобновлена.

Чтобы приостановить работу конкретного процесса, выполните следующее:

1. Переместите процесс в группу в иерархии, к которой подключена подсистема **freezer**.
2. Остановите работу этой группы, тем самым остановив работу процесса.

Перемещение процесса в остановленную группу (frozen) не разрешается.

Значения **FROZEN** и **THAWED** могут быть записаны в **freezer.state**, в то время как значение **FREEZING** изменить нельзя.

3.7. MEMORY

Подсистема **memory** создает отчеты об использовании ресурсов памяти и позволяет наложить ограничения с помощью следующих параметров:

memory.stat

Возвращает статистику памяти (см. таблицу).

Таблица 3.1. memory.stat

Значение	Описание
cache	кэш страниц (в байтах), включая **tmpfs** (**shmem**)
rss	анонимный кэш и кэш подкачки (в байтах) *за исключением* **tmpfs** (**shmem**)
mapped_file	размер файлов в карте памяти (в байтах), включая **tmpfs** (**shmem**)
pgpgin	число страниц, помещаемых в память
pgpgout	число страниц, извлекаемых из памяти
swap	использование пространства подкачки (в байтах)
active_anon	анонимный кэш и кэш подкачки (в байтах) в активном списке LRU (Least Recently Used) включая **tmpfs** (**shmem**)
inactive_anon	анонимный кэш и кэш подкачки (в байтах) в неактивном списке LRU (Least Recently Used) включая **tmpfs** (**shmem**)
active_file	память с файловой поддержкой в активном списке LRU (в байтах)
inactive_file	память с файловой поддержкой в неактивном списке LRU (в байтах)
unevictable	память, которую нельзя вернуть (в байтах)
hierarchical_memory_limit	лимит памяти для иерархии, содержащей контрольную группу **memory** (в байтах)
hierarchical_memsw_limit	суммарный лимит памяти и пространства подкачки для иерархии, содержащей контрольную группу **memory** (в байтах)

Перечисленным файлам за исключением **hierarchical_memory_limit** и **hierarchical_memsw_limit** соответствует файл **total_** с аналогичной информацией для всех дочерних групп. Так, если **swap** сообщает об использовании пространства подкачки группой, то **total_swap** — о суммарном использовании пространства и группой, и ее подчиненными группами.

Значения соотносятся следующим образом:

- **active_anon** + **inactive_anon** = (анонимная память) + (файловый кэш для **tmpfs**) + (кэш подкачки).

  Как следствие, **active_anon** + **inactive_anon** ≠ **rss**, так как **rss** не включает **tmpfs**.

- **active_file** + **inactive_file** = (размер кэша **tmpfs**).

memory.usage_in_bytes

Суммарный размер памяти, занятой процессами заданной контрольной группы (в байтах).

memory.memsw.usage_in_bytes

Суммарный размер пространства подкачки, занятого процессами заданной контрольной группы (в байтах).

memory.max_usage_in_bytes

Максимальный размер памяти, занятой процессами заданной контрольной группы (в байтах).

memory.memsw.max_usage_in_bytes

Максимальный размер пространства подкачки, занятого процессами заданной контрольной группы (в байтах).

memory.limit_in_bytes

Задает максимальный размер памяти (включая файловый кэш). По умолчанию используются байты. Допускается использование приставок **k** и **K** для килобайтов, **m** и **M** для мегабайтов, **g** и **G** для гигабайтов.

memory.limit_in_bytes накладывает ограничения не на корневую группу, а на группы нижних уровней.

Для отмены ограничений присвойте значение **-1**.

memory.memsw.limit_in_bytes

Задает максимальный размер памяти и пространства подкачки. Если единицы не указаны, по умолчанию используются байты. Допускается использование приставок **k** и **K** для килобайтов, **m** и **M** для мегабайтов, **g** и **G** для гигабайтов.

memory.memsw.limit_in_bytes накладывает ограничения не на корневую группу, а на группы нижних уровней.

Для отмены ограничений присвойте значение **-1**.

memory.failcnt

Счетчик случаев достижения лимита, заданного в **memory.limit_in_bytes**.

memory.memsw.failcnt

Возвращает число случаев, когда лимит, заданный в **memory.memsw.limit_in_bytes**, был достигнут.

memory.force_empty

Если **0**, память будет очищена от страниц, которые использовались заданиями контрольной группы. Если же память невозможно освободить, содержимое будет перенесено в родительскую группу. Прежде чем удалять группу, рекомендуется очистить память.

memory.swappiness

Заставляет ядро осуществлять подкачку памяти, используемой заданиями контрольной группы, вместо возврата страниц из кэша страниц. Это аналогично поведению, определенному для всей системы в файле **/proc/sys/vm/swappiness**. Значение по умолчанию — **60**. Значения меньше 60 уменьшают вероятность подкачки, а более высокие — увеличивают вероятность подкачки, а значения больше **100** приводят к подкачке страниц пространства адресов.

0 не отменяет подкачку полностью — в силу того глобальная логика управления виртуальной памятью не использует это значение, страницы будут подкачиваться при недостатке памяти. Чтобы запретить подкачку, рекомендуется использовать **mlock()**.

Это поведение нельзя изменить для следующих групп:

- для корневой группы, настройки которой определены в **/proc/sys/vm/swappiness**,
- для контрольной группы с существующими подчиненными группами.

memory.use_hierarchy

Флаг, отвечающий за возврат памяти. **1** заставляет подсистему **memory** освободить память, используемую процессом, который превышает максимально разрешенную квоту, и его потомками. По умолчанию (**0**) память освобождаться не будет.

3.8. NET_CLS

Подсистема **net_cls** присваивает сетевым пакетам идентификатор classid, который помогает контроллеру **tc** идентифицировать пакеты, поступающие из заданной контрольной группы. Настройки **tc** можно изменить так, чтобы пакетам из различных групп назначался разный приоритет.

net_cls.classid

net_cls.classid содержит шестнадцатеричное значение, идентифицирующее *обработчик* трафика. Например, значение **10:1** здесь будет представлено как **0x1001**.

Формат: **0x*AAAABBBB***, где *AAAA* — старший номер, а *BBBB* — младший. Нули в начале можно опустить, то есть значение **1:1** может быть записано как **0x10001**, что эквивалентно **0x00010001**.

Справочная страница **tc** содержит информацию о настройке обработчиков, которые **net_cls** назначает сетевым пакетам.

3.9. NS

Подсистема **ns** позволяет сгруппировать процессы в отдельное *пространство имен* , где они могут взаимодействовать друг с другом, но будут изолированы от внешних процессов.

3.10. ДОПОЛНИТЕЛЬНЫЕ РЕСУРСЫ

Документация

Перечисленные ниже файлы расположены в каталоге **/usr/share/doc/kernel-doc-<*версия_ядра*>/Documentation/cgroups/**, который создается при установке пакета kernel-doc.

- **blkio** — **blkio-controller.txt**
- **cpuacct** — **cpuacct.txt**
- **cpuset** — **cpusets.txt**
- **devices** — **devices.txt**
- **freezer** — **freezer-subsystem.txt**
- **memory** — **memory.txt**

ПРИЛОЖЕНИЕ A. ИСТОРИЯ ИЗМЕНЕНИЙ

Издание 1-2.400 **2013-10-31** **Rüdiger Landmann**

Rebuild with publican 4.0.0

Издание 1-2 **2012-07-18** **Anthony Towns**

Rebuild for Publican 3.0

Издание 1.0-5 **Thu May 19 2011** **Martin Prpič**

Руководство по управлению ресурсами в Red Hat Enterprise Linux 6.1.

Издание 1.0-4 **Tue Mar 1 2011** **Martin Prpič**

Исправления в соответствии с BZ#667623, BZ#667676, BZ#667699.

Уточнено описание команды cgclear: BZ#577101.

Уточнено описание команды lssubsystem: BZ#678517.

Блокирование процесса: BZ#677548.

Издание 1.0-3 **Wed Nov 17 2010** **Rüdiger Landmann**

Откорректирован пример mount: BZ#612805.

Издание 1.0-2 **Thu Nov 11 2010** **Rüdiger Landmann**

Удалены инструкции по отправлению отзывов для предыдущей версии.

Издание 1.0-1 **Wed Nov 10 2010** **Rüdiger Landmann**

Контроль качества: BZ#581702 и BZ#612805.

Издание 1.0-0 **Tue Nov 9 2010** **Rüdiger Landmann**

Полная версия для главного выпуска.