Red Hat Enterprise Linux 6

Guia de Gerenciamento de Recursos

Gerenciando recursos de sistema no Red Hat Enterprise Linux 6

Edição 1

Last Updated: 2017-10-13

Red Hat Enterprise Linux 6 Guia de Gerenciamento de Recursos

Gerenciando recursos de sistema no Red Hat Enterprise Linux 6
Edição 1

Martin Prpič
Red Hat Serviços de Conteúdo de Engenharia
mprpic@redhat.com

Rüdiger Landmann
Red Hat Serviços de Conteúdo de Engenharia
r.landmann@redhat.com

Douglas Silas
Red Hat Serviços de Conteúdo de Engenharia
dhensley@redhat.com

Nota Legal

Copyright © 2011 Red Hat, Inc.

This document is licensed by Red Hat under the Creative Commons Attribution-ShareAlike 3.0 Unported License. If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Resumo

Gerenciando recursos de sistema no Red Hat Enterprise Linux 6

Índice

CAPÍTULO 1. INTRODUÇÃO AO GRUPOS DE CONTROLE (CGROUPS) 3
1.1. COMO OS GRUPOS DE CONTROLE SÃO ORGANIZADOS 3
O Modelo de Processo do Linux 3
O Modelo Cgroup 3
1.2. RELACIONAMENTO ENTRE SUBSISTEMAS, HIERARQUIAS, GRUPOS DE CONTROLE E TAREFAS 4
1.3. IMPLICAÇÕES PARA O GERENCIAMENTO DE RECURSOS 5

CAPÍTULO 2. USANDO GRUPOS DE CONTROLE 7
2.1. O SERVIÇO CGCONFIG 7
2.1.1. O Arquivo cgconfig.conf 7
2.2. CRIANDO UMA HIERARQUIA E ANEXANDO SUBSISTEMAS 9
Método alternativo 10
2.3. ANEXANDO E DESANEXANDO SUBSISTEMAS DE UMA HIERARQUIA EXISTENTE 11
Método alternativo 11
2.4. DESMONTANDO UMA HIERARQUIA 12
2.5. CRIANDO GRUPOS DE CONTROLE 12
Método alternativo 13
2.6. REMOVENDO GRUPOS DE CONTROLE 13
2.7. DEFININDO PARÂMETROS 13
Método alternativo 15
2.8. MOVENDO UM PROCESSO PARA UM GRUPO DE CONTROLE 15
Método alternativo 15
2.8.1. O Daemon cgred 16
2.9. INICIANDO UM PROCESSO EM UM GRUPO DE CONTROLE 16
Método alternativo 17
2.9.1. Iniciando um Serviço em um Grupo de Controle 18
2.10. OBTENDO INFORMAÇÕES SOBRE OS GRUPOS DE CONTROLE 18
2.10.1. Encontrando um Processo 18
2.10.2. Encontrando um Subsistema 18
2.10.3. Encontrando Hierarquias 18
2.10.4. Encontrando Grupos de Controles 19
2.10.5. Exibindo Parâmetros dos Grupos de Controles 19
2.11. DESCARREGANDO GRUPOS DE CONTROLES 19
2.12. RECURSOS ADICIONAIS 20

CAPÍTULO 3. SUBSISTEMAS E PARÂMETROS AJUSTÁVEIS 22
3.1. BLKIO 22
3.2. CPU 24
3.3. CPUACCT 25
3.4. CPUSET 25
3.5. DEVICES 28
3.6. FREEZER 29
3.7. MEMORY 30
3.8. NET_CLS 32
3.9. NS 33
3.10. RECURSOS ADICIONAIS 33

APÊNDICE A. HISTÓRICO DE REVISÃO 34

CAPÍTULO 1. INTRODUÇÃO AO GRUPOS DE CONTROLE (CGROUPS)

O Red Hat Enterprise Linux 6 fornece um novo recurso do kernel: *control groups* (grupos de controle), que são chamados por seu nome curto *cgroups* neste guia. Os cgroups permitem alocar recursos — tais como tempo da CPU, memória do sistema, largura de banda de rede ou combinações destes recursos — entre grupos de usuários definidos de tarefas (processos) rodando em um sistema. Você pode monitorar os cgroups que configurar, negar acesso dos cgroups a certos recursos e mesmo reconfigurar seus cgroups dinamicamente em um sistema em execução. O serviço **`cgconfig`** ("*control group config*") pode ser configurado para iniciar no momento do boot e restabelecer seus cgroups pré definidos, fazendo-os persistentes através de reboots.

Usando cgroups, os administradores de sistema ganham controle refinado sobre a alocação, prioridades, negação, gerenciamento e recursos de monitoramento do sistema. Recursos de hardware podem ser divididos inteligentemente entre tarefas e usuários, aumentando a eficiência geral.

1.1. COMO OS GRUPOS DE CONTROLE SÃO ORGANIZADOS

Os cgroups são organizados hierarquicamente, como processos, e cgroups filhos herdam alguns atributos de seus pais. Entretanto, existem diferenças entre os dois modelos.

O Modelo de Processo do Linux

Todos os processos em um sistema Linux são processos filhos de um pai comum: o processo **`init`**, que é executado pelo kernel no momento de boot e inicia todos os outros processos (que por sua vez iniciam seus próprios processos filhos). Por causa que todos os processos descendem de um pai único, o modelo de processamento do Linux é uma hierarquia única, ou árvore.

Adicionalmente, todos os processos do Linux exceto o **`init`** herdam o ambiente (tal como a variável PATH) [1] e outros certos atributos (tal como abrir arquivos descritores) de seu processo pai.

O Modelo Cgroup

Os cgroups são similares aos processos em:

- eles seguem uma hierarquia, e
- Os cgroups filhos herdam certos atributos de seus cgroups pais.

A diferença fundamental é que muitas hierarquias diferentes de cgroups podem existir simultaneamente em um sistema. Se o modelo de processo Linux é uma árvore única de processos, então o modelo cgroup é uma ou mais árvores desconectadas e separadas de tarefas (exemplo: processos).

Múltiplas hierarquias separadas de cgroups são necessárias porque cada hierarquia é anexada a *um ou mais subsistemas*. Um subsistema [2] representa um recurso único, tal como tempo de CPU ou memória, O Red Hat Enterprise Linux 6 fornece nove subsistemas cgroups, listados abaixo por nome e função.

Subsistemas Disponíveis no Red Hat Enterprise Linux

- **`blkio`** — este subsistema define limites de acesso de entrada/saída para e a partir de dispositivos de bloco tais como drives físicos (disco, estado sólido, USB, etc).
- **`cpu`** — este subsistema usa o agendador para fornecer acesso às tarefas de cgroups para o CPU.

- **cpuacct** — este subsistema gera relatórios automáticos nos recursos de CPU usados pelas tarefas em um cgroup.
- **cpuset** — este subsistema atribui CPUs individuais (em sistema multicore) e nós de memória para atribuir em um cgroup.
- **devices** — este subsistema permite ou nega acesso aos dispositivos por tarefas em um cgroup.
- **freezer** — este subsistema suspende ou retoma tarefas em um cgroup.
- **memory** — este subsistema define limites no uso de memória pelas tarefas em um cgroup e gera relatórios automáticos nos recursos de memória usados por essas tarefas.
- **net_cls** — este subsistema marca os pacotes de rede com um identificador de classe (classid) que permite que o controlador de tráfego do Linux (**tc**) identifique os pacotes que originam de um cgroup em particular.
- **ns** — o subsistema do *namespace*.

NOTA

Você pode encontrar o termo *controlador de recursos* (resource controller) ou simplesmente *controlador* (controller) nas literaturas do cgroup tais como as páginas man ou documentação do kernel. Ambos destes termos são sinônimos com “subsistemas”, e partem do fato que um subsistema tipicamente agenda um recurso ou aplica um limite aos cgroups na hierarquia que está anexada.

A definição de um subsistema (controlador de recurso) é bem geral: é algo que age sobre um grupo de tarefas, como exemplo os processos.

1.2. RELACIONAMENTO ENTRE SUBSISTEMAS, HIERARQUIAS, GRUPOS DE CONTROLE E TAREFAS

Se lembre que os processos de sistemas são chamados tarefas no terminologia do cgroup.

Aqui estão algumas regras simples que governam o relacionamento entre subsistemas, hierarquias de cgrupos e tarefas junto com as explicações dessas regras.

Regra 1

Qualquer subsistema único (tal como **cpu**) pode ser conectado no máximo a uma hierarquia.

*Como uma consequencia, o subsistema **cpu** nunca pode ser anexado à duas hierarquias diferentes.*

Regra 2

Uma hierarquia única pode ter um ou mais subsistemas anexados a ele.

*Como uma consequencia, os subsistemas **cpu** e **memory** (ou qualquer número de subsistemas) podem ser anexados a uma única hierarquia, desde que cada uma não está anexada a qualquer outra hierarquia.*

Regra 3

Cada vez que uma nova hierarquia é criada nos sistemas, todas as tarefas no sistema são inicialmente

membros do cgroup padrão desta hierarquia, que é conhecido como *root cgroup*. Para qualquer hierarquia única que você criar, cada tarefa do sistema pode ser um membro de *exatamente um* cgroup nessa hierarquia. Uma tarefa única pode estar em múltiplos cgroups, desde que cada um desses cgroups está em uma hierarquia diferente. Tão logo que uma tarefa é feita um membro de um segundo cgroup na mesma hierarquia é removido do primeiro cgroup dessa hierarquia. Nunca uma tarefa estará em dois cgroups diferentes na mesma hierarquia.

Como uma consequencia, se os subsistemas ***cpu*** *e* ***memory*** *estiverem anexados à uma hierarquia chamada* ***cpu_and_mem****, e o subsistema* ***net_cls*** *estiver anexado a uma hierarquia chamada* ***net****, então um processo em execução* ***httpd*** *poderia ser um membro de qualquer um cgroup no* ***cpu_and_mem****, e qualquer um cgroup no* ***net****.*

O cgroup no ***cpu_and_mem*** *de que o processo* ***httpd*** *é um membro, poderá restringir seu tempo de CPU para a metade desse tempo alocado para outros processos e limitar seu uso de memória para um máximo de* ***1024*** *MB. Além disso, o cgroup em* ***net*** *de que é um membro poderá limitar sua taxa de transmissão para* ***30*** *megabytes por segundo.*

Quando a primeira hierarquia é criada, toda tarefa no sistema é um membro de ao menos um cgroup: o cgroup root. Quando usar os cgroups, portanto, cada sistema de tarefa está sempre em ao menos um cgroup.

Regra 4

Qualquer processo (tarefa) no sistema que se divide cria um processo filho (tarefa). A tarefa filho automaticamente se torna membro de todos os cgroups de que seu pai é membro. A tarefa filho pode então ser movida para cgroups diferentes conforme necessitado, mas inicialmente sempre herda os cgroups (o "ambiente" dos cgroups na terminologia do processo) das tarefas de seu pai.

Como uma consequencia, considere a tarefa ***httpd*** *que é um membro do cgroup chamado* ***half_cpu_1gb_max*** *na hierarquia* ***cpu_and_mem*** *e um membro do cgroup* ***trans_rate_30*** *na hierarquia* ***net****. Quando o processo* ***httpd*** *se divide, seu processo filho automaticamente se torna um membro do cgroup* ***half_cpu_1gb_max*** *e do cgroup* ***trans_rate_30****. Ele herda os mesmos cgroups exatos que as tarefas de seu pai pertencem.*

A partir desse ponto, as tarefas pai e filho são completamente independentes uma da outra: mudando os cgroups a que uma tarefa pertence não afeta a outra. Nem mudar o cgroups de uma tarefa pai afetará de alguma maneira de seus netos. Para resumir: qualquer tarefa filho sempre herda inicialmente as afiliações dos exatos mesmos cgroups de sua tarefa pai, mais essas afiliações podem ser mudadas ou removidas depois.

1.3. IMPLICAÇÕES PARA O GERENCIAMENTO DE RECURSOS

- Pelo motivo que uma tarefa pode pertencer a somente um cgroup único em qualquer hierarquia, há somente uma maneira que uma tarefa pode ser limitada ou afetada por qualquer subsistema único. Isto é lógico: um recurso, não uma limitação.

- Você pode agrupar diversos subsistemas juntos para que então eles afetem todas as tarefas em uma hierarquia única. Por causa que os cgroups nessa hierarquia possuem diferentes parâmetros definidos, estas tarefas serão afetadas diferentemente.

- Pode ser necessário as vezes *refazer* uma hierarquia. Um exemplo seria remover um subsistema de uma hierarquia que possui diversos subsistemas anexados, e anexando-os a uma nova e separada hierarquia.

- Reciprocamente, se a necessidade para dividir subsistemas entre hierarquias separadas é reduzido, você pode remover uma hierarquia e anexar seus subsistemas a um já existente.

- A estrutura permite um uso de cgroup simples, tal como definir poucos parâmetros para tarefas especificas em uma hierarquia única, tal como uma com apenas os subsistemas de cpu e memória anexados.
- A estrutura também pode permitir uma configuração altamente específica: cada tarefa (processo) em um sistema pode ser um membro de cada hierarquia, cada qual possui um subsistema único anexado. Tal configuração daria ao administrador de sistemas controle absoluto sobre todos os parâmetros para cada tarefa única.

---

[1] O processo pai é capaz de alterar o ambiente antes de passa-lo para outro processo filho

[2] Você deve estar atento que subsistemas também são chamados *controladores de recursos* (resource controllers), ou simplesmente *controladores* (controllers), nas páginas man libcgroup e outras documentações.

CAPÍTULO 2. USANDO GRUPOS DE CONTROLE

A maneira mais fácil de trabalhar com cgroups é instalar o pacote libcgroup, que contém um número de utilitários de linha de comando relacionados ao cgroup e suas páginas man associadas. É possível *montar* hierarquias e definir parâmetros de cgroup (não persistentes) usando comandos shell e utilitários disponíveis em qualquer sistema. Entretanto, usar os utilitários fornecidos do libcgroup simplifica o processo e extende suas capabilidades. Portanto, este guia foca principalmente em comandos do libcgroup. Na maioria dos casos nós incluimos os comandos shell equivalentes para ajudar a descrever o mecanismo subjacente. Entretanto, nós recomendamos que você use os comandos libcgroup sempre que for prático.

NOTA

Para usar os cgroups, primeiro certifique-se que o pacote libcgroup está instalado no seu sistema rodando, como root:

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

2.1. O SERVIÇO CGCONFIG

O serviço **cgconfig** instalado com o pacote libcgroup fornece uma maneira conveniente para criar hierarquias, anexar subsistemas à hierarquias e gerenciar cgroups com essas hierarquias. Nós recomendamos que você use o **cgconfig** para gerenciar hierarquias e cgroups no seu sistema.

O serviço **cgconfig** não é iniciado por padrão no Red Hat Enterprise Linux 6. Quando você inicia o serviço com **chkconfig**, ele lê o arquivo de configuração — **/etc/cgconfig.conf**. Os cgroups são portanto recriados de sessão para sessão e se tornam persistentes. Dependendo dos conteúdos do arquivo de configuração, o **cgconfig** pode criar hierarquias, montar sistemas de arquivos necessários, criar cgroups e definir parâmetros de subsistemas para cada grupo.

O arquivo padrão **/etc/cgconfig.conf** instalado com o pacote libcgroup cria e monta uma hierarquia individual para cada subsistema e anexa os subsistemas à estas hierarquias.

Se você parar o serviço **cgconfig** (com o comando **service cgconfig stop**), ele desmonta todas as hierarquias que montou.

2.1.1. O Arquivo cgconfig.conf

O arquivo **/etc/cgconfig.conf** contém dois tipos principais de entrada — *mount* e *group*. As entradas do Mount criam e montam hierarquias como sistemas de arquivos virtuais, e anexam subsistemas àquelas hierarquias. Entradas Mount são definidas usando a seguinte sintaxe:

```
mount {
    <controller> = <path>;
    …
}
```

Veja Exemplo 2.1, “Criando entradas mount” para um exemplo de uso.

Exemplo 2.1. Criando entradas mount

O seguinte exemplo cria uma hierarquia para o subsistema **cpuset**:

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

o equivalente ao comandos shell:

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

Entradas de grupo criam cgroups e definem parâmetros de subsistema. Entradas Group são definidas com a seguinte sintaxe:

```
group <name> {
    [<permissions>]
    <controller> {
        <param name> = <param value>;
        …
    }
    …
}
```

Note que a seção **permissions** é opcional. Para definir permissões para uma entrada de grupo, use a seguinte sintaxe:

```
perm {
    task {
        uid = <task user>;
        gid = <task group>;
    }
    admin {
       uid = <admin name>;
       gid = <admin group>;
    }
}
```

Veja Exemplo 2.2, “Criando uma entrada group” para exemplo de uso:

Exemplo 2.2. Criando uma entrada group

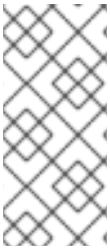
O seguinte exemplo cria um cgroup para daemons sql, com permissões para os usuários no grupo **sqladmin** para adicionar tarefas ao cgroup e ao usuário **root** para modificar parâmetros de subsistemas:

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

```
        cpu.shares = 100;
    }
}
```

Quando combinados com o exemplo da entrada de montagem no Exemplo 2.1, “Criando entradas mount”, os comandos shell equivalentes são:

```
~]# 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
```



NOTA

Você deve iniciar o serviço **cgconfig** para as mudanças no **/etc/cgconfig.conf** terem efeito:

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

Quando você instala o pacote libcgroup, é gravado um arquivo de configuração de amostra **/etc/cgconfig.conf**. Os simbolos hash ('#') no início de cada linha comentam aquela linha e a torna invisível ao serviço **cgconfig**.

2.2. CRIANDO UMA HIERARQUIA E ANEXANDO SUBSISTEMAS

ATENÇÃO

As seguintes instruções, que cobrem criar uma nova hierarquia e anexar subsistemas a ela, presumem que os cgroups não estão configurados em seu sistema. Neste caso, estas instruções não afetarão a operação do sistema. Mudar os parâmetros ajustáveis em um cgroup com tarefas, entretanto, podem imediatamente afetar essas tarefas. Este guia lhe alerta a primeira vez quando mudar um parâmetro cgroup ajustável que pode afetar uma ou mais tarefas.

Em um sistema nos quais os cgroups já estão configurados (tanto manualmente ou pelo serviço **cgconfig**) estes comandos falharão a menos que você primeiro desmonte hierarquias existentes, que afetarão a operação do sistema. Não experimente estas instruções em sistemas em produção.

Para criar uma hierarquia e anexar subsistemas a ela, edite a seção **mount** do arquivo **/etc/cgconfig.conf** como root. Entradas na seção **mount** possuem o seguinte formato:

```
subsystem = /cgroup/hierarchy;
```

Quando o **cgconfig** iniciar da próxima vez, ele criará a hierarquia e anexará subsistemas a ele.

O seguinte exemplo cria uma hierarquia chamada **cpu_and_mem** e anexa os subsistemas **cpu**, **cpuset**, **cpuacct**, e **memory** a ele.

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

Método alternativo

Você pode também usar os comandos shell e utilitários para criar hierarquias e anexar subsistemas a eles.

Crie um *ponto de montagem* para a hierarquia como root. Inclui o nome do cgroup no ponto de montagem.

```
~]# mkdir /cgroup/name
```

Por exemplo:

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

Depois, use o comando **mount** para montar a hierarquia e simultâneamente anexar um ou mais subsistema. Por exemplo:

```
~]# mount -t cgroup -o subsystems name /cgroup/name
```

Onde um *subsystems* é uma lista separada por vírgulas de subsistemas e *name* é o nome da hierarquia. Descrições rápidas de todos os subsistemas disponíveis são listados em Subsistemas Disponíveis no Red Hat Enterprise Linux, e Capítulo 3, *Subsistemas e Parâmetros Ajustáveis* fornece uma referência detalhada.

Exemplo 2.3. Usando o comando mount para anexar subsistemas

Neste exemplo, um diretório chamado **/cgroup/cpu_and_mem** já existe, que servirá como o ponto de montagem para a hierarquia que foi criamos. Anexaremos os subsistemas **cpu**, **cpuset** e **memory** a uma hierarquia que chamamos **cpu_and_mem**, e **mount** a hierarquia **cpu_and_mem** em **/cgroup/cpu_and_mem**:

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

Você pode listar todos os subsistemas disponíveis junto com seus atuais pontos de montagem (exemplo: onde a hierarquia a que estão anexados está montada) com o **lssubsys** [3]:

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

Este resultado indica que:

- os subsistemas **cpu**, **cpuset** and **memory** estão anexados à hierarquia montada em **/cgroup/cpu_and_mem**, e
- os subsistemas **net_cls**, **ns**, **cpuacct**, **devices**, **freezer** e **blkio** não estão ainda anexados a qualquer hierarquia, conforme ilustrado pela falta de um ponto de montagem correspondente.

2.3. ANEXANDO E DESANEXANDO SUBSISTEMAS DE UMA HIERARQUIA EXISTENTE

Para adicionar um subsistema a uma hierarquia existente, desanexe-o de uma outra hierarquia existente, ou mova-o para uma hierarquia diferente, edite a seção **mount** do arquivo **/etc/cgconfig.conf** como root, usando a mesma sintaxe descrita em Seção 2.2, “Criando uma Hierarquia e Anexando Subsistemas”. Quando **cgconfig** iniciar da próxima vez, ele reorganizará o subsistema de acordo às hierarquias que você especificar.

Método alternativo

Para adicionar um subsistema desanexado à uma hierarquia existente, remonte a hierarquia. Inclua o subsistema extra no comando **mount**, junto com a opção **remount**.

Exemplo 2.4. Remontando uma hierarquia para adicionar um subsistema

O comando **lssubsys** mostra os subsistemas **cpu**, **cpuset**, e **memory** anexados à hierarquia **cpu_and_mem**:

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

Remontaremos a hierarquia **cpu_and_mem**, usando a opção **remount** e incluindo o **cpuacct** na lista de subsistemas:

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

O comando **lssubsys** agora mostra o **cpuacct** anexado à hierarquia **cpu_and_mem**:

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

–

De forma análoga, você pode desanexar um subsistema de uma hierarquia existente remontando a hierarquia e omitindo o nome de subsistema das opções -o. Por exemplo, para então desanexar o subsistema **cpuacct**, simplesmente remonte-o omita-o:

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

2.4. DESMONTANDO UMA HIERARQUIA

Você pode *desmontar* uma hierarquia de cgroups com o comando **umount**:

```
~]# umount /cgroup/name
```

Por exemplo:

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

Se a hierarquia está atualmente vazia (que significa, ela possui somente o cgroup root), a hierarquia é desativada quando ela é desmontada. Se a hierarquia contém quaisquer outras opções, a hierarquia permanece ativa no kernel mesmo que ela não esteja mais montada.

Para remover uma hierarquia, garanta que todos os cgroups filhos são removidos antes de desmontar a hierarquia ou use o comando **cgclear** que pode desativar uma hierarquia mesmo quando ela não está vazia — consulte Seção 2.11, “Descarregando Grupos de Controles”.

2.5. CRIANDO GRUPOS DE CONTROLE

Use o comando **cgcreate** para criar cgroups. A sintaxe para o **cgcreate** é: **cgcreate -t *uid*:*gid* -a *uid*:*gid* -g *subsystems*:*path*** , onde:

- **-t** (opcional) — especifica um usuário (por ID, uid) e um grupo (por ID de grupo) para ter propriedade do pseudofile **tasks** para este cgroup. Este usuário pode adicionar ou remover tarefas a partir do cgroup.

  NOTA

  Note que a única maneira para remover uma tarefa a partir de um cgroup é move-lo para um cgroup diferente. Para mover uma tarefa, o usuário deve ter acesso de escrita ao cgroup de *destino*; acesso de escrita ao cgroup fonte não é importante nesse caso.

- **-a** (opcional) — especifica um usuário (por ID de usuário, uid) e um grupo (por ID de grupo, gid) para possuir todos os pseudofiles a não ser o **tasks** para este cgroup. Este usuário pode modificar o acesso que as tarefas neste cgroup possuem para recursos de sistema.

- **-g** — especifica a hierarquia na qual o cgroup deve ser criado, como uma lista separada por vírgulas dos *subsistemas* com essas hierarquias. Se os subsistemas nesta lista estão em hierarquias diferentes, o grupo é criado em cada uma dessas hierarquias. A lista de hierarquias é seguida por dois pontos e o *caminho* ao grupo filho relativo à hierarquia. Não inclua o ponto de montagem da hierarquia no caminho.

Por exemplo, o cgroup localizado no diretório **/cgroup/cpu_and_mem/lab1/** é chamada apenas **lab1** — seu caminho já está unicamente determinado porque não há, no máximo, uma hierarquia para um determinado subsistema. Note que também o grupo é controlado por todos os subsistemas que existem nas hierarquias na qual o cgroup é criado, mesmo que estes subsistemas não foram especificados no comando **cgcreate** — consulte Exemplo 2.5, “uso do cgcreate”.

Por causa que todos os cgroups na mesma hierarquia possuem os mesmos controladores, o grupo filho possui os mesmos controladores como seu pai.

Exemplo 2.5. uso do cgcreate

Considere um sistema onde os subsistemas **cpu** e **memory** são montados juntos à hierarquia **cpu_and_mem** e o controlador **net_cls** é montado em uma hierarquia separada chamada **net**. Agora, executaremos:

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

O comando **cgcreate** cria dois grupos chamados **test-subgroup**, um na hierarquia **cpu_and_mem** e um na hierarquia **net**. O grupo **test-subgroup** na hierarquia **cpu_and_mem** é controlado pelo subsistema **memory**, mesmo que nós não especificamos ele no comando **cgcreate**.

Método alternativo

Para criar um filho do cgroup diretamente, use o comando **mkdir**:

```
~]# mkdir /cgroup/hierarchy/name/child_name
```

Por exemplo:

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

2.6. REMOVENDO GRUPOS DE CONTROLE

Remova os cgroups com o **cgdelete**, que possui uma sintaxe similar ao **cgcreate**. Execute: **cgdelete *subsystems*:*path***, onde:

- *subsystems* é uma lista separada por vírgulas de subsistemas.
- *path* é o caminho para o cgroup relativo ao root da hierarquia.

Por exemplo:

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

O **cgdelete** pode também remover recursivamente todos os subgrupos com a opção **-r**.

Quando você deletar um cgroup, todas suas tarefas são movidas ao grupo pai.

2.7. DEFININDO PARÂMETROS

Defina os parâmetros de subsistemas rodando o comando **cgset** de uma conta de usuário com permissão para modificar o cgroup relevante. Por exemplo, se o **/cgroup/cpuset/group1** existe, especifique os CPUs aos quais este grupo tem acesso com o seguinte comando:

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

A sintaxe para **cgset** é: **cgset -r *parameter=value path_to_cgroup***, onde:

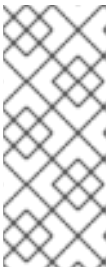
- *parameter* é o parâmetro a ser definido, que corresponde ao arquivo no diretório ao cgroup dado.
- *value* é o valor para o parâmetro
- O *path_to_cgroup* é o caminho para o cgroup *relativo ao root da hierarquia*. Por exemplo, para definir o parâmetro ao grupo root (se o **/cgroup/cpuacct/** existe), execute:

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

  Alternativamente, por causa que o **.** é relativo ao grupo root (que é, o próprio grupo root) você pode também executar:

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

  Note, entretanto que a **/** é a sintaxe preferida.

  NOTA

  Somente um número pequeno de parâmetros podem ser definidos para o grupo root (tal como o parâmetro **cpuacct.usage** mostrado nos exemplos acima). Isto é porque um grupo root possui propriedade sobre todos os recursos existentes, portanto, não faria sentido limitar todos os processos existentes definindo certos parâmetros, por exemplo o parâmetro **cpuset.cpu**.

  Para definir o parâmetro do **group1**, que é um subgroup de grupo root, execute:

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

  A barra no final do nome do grupo (por exemplo **cpuacct.usage=0 group1/**) é opcional.

Os valores que você pode definir com o **cgset** pode depender de valores maiores ajustados em uma determinada hierarquia. Por exemplo, se o **group1** é limitado para usar somente o CPU 0 em um sistema, você não pode definir o **group1/subgroup1** para usar os CPUs 0 e 1 ou usar somente o CPU 1.

Você também pode usar o **cgset** para copiar os parâmetros de um cgroup para outro cgroup existente. Por exemplo:

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

A sintaxe para copiar um parâmetro com o **cgset** é: **cgset --copy-from *path_to_source_cgroup path_to_target_cgroup***, onde:

- *path_to_source_cgroup* é o caminho ao cgroup cujos parâmetros devem ser copiados, relativos ao grupo root da hierarquia.
- *path_to_target_cgroup* é o caminho ao cgroup de destino, relativo ao grupo root da hierarquia

Certifique-se que quaisquer parâmetros obrigatórios para os vários subsistemas são definidos antes de você copiar os parâmetros de um grupo ao outro ou o comando falhará. Para mais informações sobre parâmetros obrigatórios, consulte: Importante - Parâmetros obrigatórios .

Método alternativo

Para definir os parâmetros diretamente em um cgroup, insira os valores ao subsistema de pseudo-file relevante usando o comando **echo**. Por exemplo, este comando insere o valor **0-1** ao pseudofile **cpuset.cpus** do cgroup **group1**:

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

Com este valor, as tarefas neste cgroup são restringidas à CPUs 0 e 1 no sistema:

2.8. MOVENDO UM PROCESSO PARA UM GRUPO DE CONTROLE

Mova um processo em um cgroup executando o comando **cgclassify**:

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

A sintáxe para **cgclassify** é: **cgclassify -g *subsystems*:*path_to_cgroup pidlist***, onde:

- *subsystems* é uma lista separada por vírgula de subsistemas ou ***** para iniciar o processo nas hierarquias associadas com todos os subsistemas disponíveis. Note que se os cgroups do mesmo nome existirem um múltiplas hierarquias, a opção **-g** move os processos em cada um desses grupos. Certifique-se que o cgroup exista dentro de cada uma das hierarquias cujos subsistemas você especifica aqui.
- *path_to_cgroup* é o caminho para o cgrupo dentro de sua hierarquia
- *pidlist* é uma lista separada por espaços do *process identifier* (PIDs)

Você também pode adicionar a opção **--sticky** antes do *pid* para manter qualquer processo filho no mesmo cgroup. Se você não definir esta opção e o daemon **cgred** estiver rodando, os processos filhos serão alocados aos cgroups baseados nas configurações encontradas em **/etc/cgrules.conf**. O próprio processo, no entanto, ficará no grupo de controle no qual você iniciou.

Ao usar o **cgclassify**, você poderá mover diversos processos simultaneamente. Por exemplo, este comando muda os processos com PIDs **1701** e **1138** para o cgroup **group1/**:

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

Observe que os PIDs a serem movidos são separados por espaços e que os grupos especificados devem estar em hierarquias diferentes.

Método alternativo

Para mover um processo para um cgroup diretamente, grave seu PID no arquivo **tasks** do cgroup. Por exemplo, para mover um processo com o PID **1701** para um cgroup em **/cgroup/lab1/group1/**:

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

2.8.1. O Daemon cgred

O **Cgred** é um daemon que move tarefas para cgroups de acordo com os parâmetros definidos no arquivo **/etc/cgrules.conf**. Entradas no arquivo **/etc/cgrules.conf** podem tomar uma ou duas formas:

- *user hierarchies control_group*
- *user:command hierarchies control_group*

Por exemplo:

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

Esta entrada especifica que qualquer processo que pertence ao usuário chamado **maria** acessará o subsistema de dispositivos de acordo com os parâmetros especificados no cgroup **/usergroup/staff**. Para associar comandos específicos aos cgroups específicos, adicione o parâmetro *command* como se segue:

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

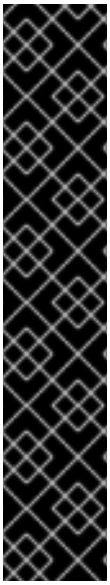
A entrada agora especifica que quando o usuário chamado **maria** usa o comando **ftp**, o processo é automaticamente movido ao cgroup **/usergroup/staff/ftp** na hierarquia que contém o subsistema **devices**. Note, entretanto, que o daemon move o processo ao cgroup somente depois que a condição apropriada é cumprida. Portanto, o processo **ftp** poderá rodar por um curto período no grupo incorreto. Além disso, se o processo rapidamente gera filhos enquanto no grupo incorreto, estes filhos não poderão ser movidos.

Entradas no arquivos **/etc/cgrules.conf** podem incluir a seguinte nota extra:

- **@** — quando pré-fixado para *user*, indica um grupo ao invés de um usuário individual. Por exemplo, **@admins** são usuários no grupo **admins**.
- ***** — representa "todos". Por exemplo, ***** no campo **subsystem** representa todos os subsistemas.
- **%** — representa um item igual ao item na linha acima. Por exemplo:

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

2.9. INICIANDO UM PROCESSO EM UM GRUPO DE CONTROLE



IMPORTANTE

Alguns subsistemas possuem parâmetros obrigatórios que devem ser definidos antes que você possa mover uma tarefa em um cgroup que usa quaisquer desses subsistemas. Por exemplo, antes de você mover uma tarefa em um cgroup que usa o subsistema **cpuset**, os parâmetros **cpuset.cpus** e **cpuset.mems** devem ser definidos para esse cgroup.

Os exemplos nesta seção ilustram a sintáxe correta para o comando, mas somente funcionam em sistemas cujos parâmetros obrigatórios relevantes foram definidos para qualquer controlador usado nestes exemplos. Se você já não houver configurado os controladores relevantes, você não conseguirá copiar os comandos de exemplo diretamente a partir desta seção e esperar que eles funcionem em seu sistema.

Consulte a Seção 3.10, “Recursos Adicionais” para uma descrição de qual os parâmetros são mandatórios para os subsistemas dados.

Inicie processos em um cgroup rodando o comando **cgexec**. Por exemplo, este comando inicia o navegador **lynx** dentro do cgroup **group1**, sujeito às limitações impostas nesse grupo pelo subsistema **cpu**:

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

A sintaxe para o **cgexec** is: **cgexec -g *subsystems*:*path_to_cgroup* *command* *arguments*** , onde:

- *subsystems* é uma lista de subsistemas separada por vírgulas, ou ***** para iniciar o processo nas hierarquias associadas com todos os subsistemas disponíveis. Note que, como acontece com o **cgset** descrito na Seção 2.7, “Definindo Parâmetros”, se os cgroups do mesmo nome existem em múltiplas hierarquias, a opção **-g** cria processos em cada um desses grupos. Certifique-se que o cgroup existe dentro de cada das hierarquias cujos subsistemas você especifica aqui.
- *path_to_cgroup* é o caminho para o cgroup relativo à hierarquia.
- *command* é o comando para executar
- *arguments* são quaisquer argumentos para o comando

Você também pode adicionar a opção **--sticky** antes de *command* para manter qualquer processo filho no mesmo cgroup. Se você não definir esta opção e o daemon do **cgred** estiver em execução, os processos filhos serão alocados aos cgroups baseados em configurações encontradas em **/etc/cgrules.conf**. O próprio processo, no entanto, ficará no grupo de controle no qual você o iniciou.

Método alternativo

Quando você iniciar um novo processo, ele herda o grupo de seu processo pai. Portanto, um método alternativo para iniciar um processo em um determinado cgroup é mover seu processo shell ao grupo (consulte a Seção 2.8, “Movendo um Processo para um Grupo de Controle”) e então inicie o processo desse shell. Por exemplo:

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

Note que depois de sair do **lynx**, seu shell aberto está ainda no cgroup **group1**. Portanto, uma maneira melhor ainda seria:

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

2.9.1. Iniciando um Serviço em um Grupo de Controle

Você pode iniciar certos serviços em um cgroup. Os serviços que podem ser iniciados em cgroups devem:

- usar um arquivo **/etc/sysconfig/*servicename***
- usar a função **daemon()** do **/etc/init.d/functions** para iniciar o serviço

Para fazer um serviço iniciar em um cgroup, edite seu arquivo no diretório **/etc/sysconfig** para incluir uma entrada na forma de **CGROUP_DAEMON="*subsystem*:*control_group*"** onde *subsystem* é um subsistema associado à hierarquia específica e *control_group* é um cgroup nesta hierarquia. Por exemplo:

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

2.10. OBTENDO INFORMAÇÕES SOBRE OS GRUPOS DE CONTROLE

2.10.1. Encontrando um Processo

Para encontrar um cgroup de onde um processo pertence, execute:

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

Ou se você souber o PID do processo, execute:

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

2.10.2. Encontrando um Subsistema

Para encontrar subsistemas que estão disponíveis em seu kernel e como eles são montados juntos às hierarquias, execute:

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

Ou, encontre os pontos de montagem de um determinado subsistema, execute:

```
~]$ lssubsys -m subsystems
```

onde *subsystems* é uma lista de subsistemas que você está interessado. Note que o comando **lssubsys -m** retorna somente o ponto de montagem de nível superior para cada hierarquia.

2.10.3. Encontrando Hierarquias

Nós recomendamos que você monte as hierarquias sob o **/cgroup**. Assumindo que este é o caso de

seu sistema, liste ou navegue pelo conteúdo desse diretório para obter uma lista de hierarquias. Se a **tree** (árvore) estiver instalada no seu sistema, execute-a para obter uma visão geral de todas as hierarquias dos cgroups dentro dela:

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

2.10.4. Encontrando Grupos de Controles

Para listar os cgroups em um sistema, execute:

```
~]$ lscgroup
```

Você pode restringir o resultado a uma determinada hierarquia especificando um controlador e o caminho no formato ***controller*:*path***. Por exemplo:

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

lista somente subgrupos do cgroup **adminusers** na hierarquia a qual o subsistema **cpuset** é anexado.

2.10.5. Exibindo Parâmetros dos Grupos de Controles

Para exibir os parâmetros de cgroups especificos, execute:

```
~]$ cgget -r parameter list_of_cgroups
```

onde *parameter* é um pseudofile que contém valores para um subsistema e *list_of_cgroups* é uma lista de cgroups separadas com espaços. Por exemplo:

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

exibe os valores do **cpuset.cpus** e **memory.limit_in_bytes** para cgroups **lab1** e **lab2**.

Se você não conhece os nomes dos parâmetros, use um comando como:

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

2.11. DESCARREGANDO GRUPOS DE CONTROLES

ATENÇÃO

O comando **cgclear** destrói todos os cgroups em todas hierarquias. Se você não possuir estas hierarquias armazenadas em um arquivo de configuração, você não conseguirá reconstruí-las imediatamente.

Para limpar um sistema de arquivos cgroup inteiro, use o comando **cgclear**.

Todas as tarefas no cgroup são realocadas para o nó root das hierarquias, todos os cgroups são removidos e o próprio sistema de arquivo é desmontado do sistema, desta maneira destruindo todas as hierarquias montadas anteriormente. Finalmente, o diretório onde o sistema de arquivos cgroup foi montado está atualmente deletado.

NOTA

Usando o comando **mount** para criar cgroups (ao contrário de cria-los usando o serviço **cgconfig**) resulta na criação de uma entrada no arquivo **/etc/mtab** (a tabela de sistemas de arquivos montada). Esta mudança é também refletida no arquivo **/proc/mounts**. Entretanto, o descarregamento dos cgroups com o comando **cgclear**, junto com os outros comandos **cgconfig**, usa uma interface do kernel direta que não reflete suas mudanças no arquivo **/etc/mtab** e somente escreve a nova informação no arquivo **/proc/mounts**. Dessa maneira, depois de descarregar os cgroups com o comando **cgclear**, os cgroups desmontados podem ainda estar visíveis no arquivo **/etc/mtab** e consequentemente, exibido quando o comando **mount** é executado. É recomendável consultar o arquivo **/proc/mounts** para uma lista precisa de todos os cgroups montados.

2.12. RECURSOS ADICIONAIS

A documentação definitiva para os comandos do cgroup são as páginas man fornecidas com o pacote libcgroup. Os números das seções são especificados na lista das páginas man abaixo.

As páginas man libcgroup

- **man 1 cgclassify** — o comando **cgclassify** é usado para mover tarefas em execução para um ou mais cgroups.

  man 1 cgclear — o comando **cgclear** é usado para deletar todos os cgroups em uma hierarquia.

  man 5 cgconfig.conf — cgroups são definidos no arquivo **cgconfig.conf**.

  man 8 cgconfigparser — o comando **cgconfigparser** analisa o arquivo **cgconfig.conf** e monta hierarquias.

  man 1 cgcreate — o comando **cgcreate** cria novos grupos nas hierarquias.

  man 1 cgdelete — o comando **cgdelete** remove determinados cgroups.

  man 1 cgexec — o comando **cgexec** executa tarefas em cgroups especificados.

  man 1 cgget — o comando **cgget** exibe parâmetros do cgroup.

  man 5 cgred.conf — **cgred.conf** é o arquivo de configuração para o serviço **cgred**.

  man 5 cgrules.conf — **cgrules.conf** contém as regras usadas para determinar quando as tarefas pertencem a certos cgroups.

  man 8 cgrulesengd — o serviço **cgrulesengd** distribui tarefas aos cgroups.

  man 1 cgset — o comando **cgset** define parâmetros para um cgroup.

man 1 lscgroup — o comando **lscgroup** lista os cgroups em uma hierarquia.

man 1 lssubsys — o comando lista as hierarquias contendo os subsistemas especificados.

---

[3] O comando **lssubsys** é um dos utilitários fornecidos pelo pacote libcgroup. Você deve instalar o libcgroup para usa-lo: consulte o Capítulo 2, *Usando Grupos de Controle* se você não consegue executar o comando **lssubsys**.

CAPÍTULO 3. SUBSISTEMAS E PARÂMETROS AJUSTÁVEIS

Subsistemas são módulos do kernel que estão atentos aos cgroups. Geralmente, são controladores de recursos que alocam diferentes níveis de recursos de sistema à diferentes cgroups. No entanto, os subsistemas podem ser programados por qualquer outra interação com o kernel onde existe a necessidade de tratar diferentes grupos de processos de maneiras diferentes. O *application programming interface* (API), para desenvolver os novos subsistemas, é documentado em **`cgroups.txt`** na documentação do kernel, instalado no seu sistema em **`/usr/share/doc/kernel-doc-kernel-version/Documentation/cgroups/`** (fornecido pelo pacote kernel-doc). A última versão da documentação do cgroups é também disponível on line em http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt. Note, entretanto que os recursos na última documentação podem não corresponder com aqueles disponíveis no kernel instalado no seu sistema.

Os *state objects* (objetos de estado) que contém os parâmetros de subsistemas para um cgroup, são representados como *pseudofiles* dentro de sistema de arquivos virtuais do cgroup. Estes pseudofiles podem ser manipulados pelo terminal ou suas chamadas de sistema equivalentes. Por exemplo o **`cpuset.cpus`** é um pseudofile que especifica quais CPUs um cgroup é permitido acessar. Se o **`/cgroup/cpuset/webserver`** é um cgroup para o servidor web que roda um sistema, executamos o seguinte comando:

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

O valor **`0,2`** é escrito no pseudofile **`cpuset.cpus`** e portanto limita quaisquer tarefas de quais os PIDs são listados no **`/cgroup/cpuset/webserver/tasks`** para usar somente a CPU 0 e CPU 2 no sistema.

3.1. BLKIO

O subsistema Bloco E/S (**`blkio`**) controla e monitora o acesso a E/S em dispositivos de bloco por tarefas nos cgroups. Escrever valores em algum desses pseudofiles limita o acesso ou largura de banda e a leitura de valores de alguns desses pseudofiles fornece informações sobre operações de E/S.

blkio.weight

especifica a proporção relativa (*weight*) do acesso do bloco E/S disponível por padrão a um cgroup, na abrangência de **`100`** a **`1000`**. Este valor é sobrescrito por dispositivos especificos pelo parâmetro **`blkio.weight_device`**. Por exemplo, para atribuir o peso padrão de **`500`** para um cgroup para o acesso aos dispositivos de bloco, execute:

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

blkio.weight_device

especifica a proporção relativa de acesso de E/S (*weight*) em dispositivos especificos disponíveis a um cgroup, na abrangencia de **`100`** a **`1000`**. O valor deste parâmetro sobrescreve o valor do parâmetro **`blkio.weight`** para dispositivos especificados. O valores têm o formato *maior:menor peso* onde o *maior* e *menor* são tipos de dispositivos e números de nós especificados no *Dispositivos Alocados do Linux*, caso contrário conhecido como *Lista de Dispositivos Linux* e disponíveis a partir de http://www.kernel.org/doc/Documentation/devices.txt. Por exemplo, para atribuir um peso de **`500`** a um cgroup para acessar o **`/dev/sda`**, rode:

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

Na anotação *Dispositivos Alocados do Linux*, **8:0** representa **/dev/sda**.

blkio.time

reporta o tempo que um cgroup teve acesso E/S a dispositivos especificos. As entradas possuem três campos: *major*, *minor*, e *time*. *Major* e *minor* são tipos de dispositivos e números de nós especificados no *Dispositivos Alocados do Linux*, e *time* é o período de tempo em milisegundos (ms).

blkio.sectors

reporta o número de setores transferidos para ou a partir de dispositivos especificos por um cgroup. As entradas possuem três campos: *major*, *minor*, e *sectors*. *Major* e *minor* são os tipos de dispositivos e números de nós especificos no *Dispositivos Alocados do Linux*, e *sectors* é o número de setores do disco.

blkio.io_service_bytes

reporta o número de bytes transferidos para ou a partir de dispositivos especificos por um cgroup. As entradas têm quatro campos: *major*, *minor*, *operation*, e *bytes*. *Major* e *minor* são tipos de dispositivos e números de nós especificados no *Dispositivos Alocados do Linux*, *operation* representa o tipo de operação (**leitura**, **escrita**, **sync**, ou **async**) e *bytes* é o número de bytes transferidos.

blkio.io_serviced

reporta o número de operações de E/S realizadas em dispositivos específicos por um cgroup. As entradas têm quatro campos: *major*, *minor*, *operation*, e *bytes*. *Major* e *minor* são tipos de dispositivos e números de nó especificados no *Dispositivos Alocados do Linux*, *operation* representa o tipo de operação (**leitura**, **escrita**, **sync**, ou **async**) e *number* representa o número de operações.

blkio.io_service_time

reporta

blkio.io_wait_time

reporta o tempo total das operações de E/S em dispositivos especificos que um cgroup gastou esperando por um serviço nas filas do agendador. Quando você interpretar este relatório, observe:

- O tempo reportado pode ser maior do que o tempo total decorrido, por causa que o tempo reportado é o total cumulativo de todas as operações de E/S para o cgroup em vez do tempo que o próprio cgroup gastou esperando por operações de E/S. Para encontrar o tempo que o cgroup como um todo gastou esperando, uso o **blkio.group_wait_time**.

- Se um dispositivo tiver um **queue_depth** > 1, o tempo reportado somente inclui o tempo até que o pedido seja despachado ao dispositivo, não qualquer tempo gasto esperando por um serviço enquanto o dispositivo re ordena os pedidos.

Entradas que possuem quatro campos: *major*, *minor*, *operation*, e *bytes*. *Major* e *minor* são tipos de dispositivos e números de nós especificados no *Linux Allocated Devices*, *operation* representa o tipo de operação (**leitura**, **escrita**, **sync**, ou **async** e *time* é o período de tempo em nanosegundos (ns). O tempo é reportado em nanosegundos em vez de uma unidade maior entao este relatório é significativo mesmo para dispositivos de estado sólido.

blkio.io_merged

reporta o número de pedidos da BIOS juntados em pedidos de operações de E/S por um cgroup. As entradas possuem dois campos: *number* e *operation*. *Number* é o número de pedidos, e *operation* representa o tipo de operação (**read**, **write**, **sync**, ou **async**).

blkio.io_queued

reporta o número de pedidos enfileirados para operações E/S por um cgroup. As entradas têm dois campos: *number* e *operation*. *Number* é o número de pedidos *operation* representa o tipo de operação (**`read`**, **`write`**, **`sync`**, ou **`async`**).

blkio.avg_queue_size

reporta o tamanho médio da fila para operações de E/S por um cgroup, sobre período de tempo inteiro para a existência do grupo. O tamanho da fila é amostrado toda vez que uma fila para este cgroup obter um timeslice. Observe que este relatório está disponível somente se **`CONFIG_DEBUG_BLK_CGROUP=y`** é definido no sistema.

blkio.group_wait_time

reporta o tempo total (em nanosegundos — ns) que um cgroup gastou esperando por um timeslice para uma de suas filas. O relatório é atualizado toda vez que uma fila deste cgroup obtem um timeslice, então se você ler este pseudofile enquanto o cgroup estiver esperando por um timeslice, o relatório não irá conter o tempo gasto em espera para a operação atualmente enfileirada. Observe que este relatório está disponível somente se o **`CONFIG_DEBUG_BLK_CGROUP=y`** estiver definido no sistema.

blkio.empty_time

reporta o tempo total (em nanosegundos — ns) que um cgroup gastou sem qualquer pedidos pendentes. O relatório é atualizado toda vez que uma fila para este cgroup tiver um pedido pendente, então se você ler este pseudofile enquanto o cgroup não tiver pedidos pendentes, o relatório não conterá o tempo gasto no estado vazio. Observe que este relatório está disponível somente se o **`CONFIG_DEBUG_BLK_CGROUP=y`** estiver definido no sistema.

blkio.idle_time

reporta o tempo total (em nanosegundos — ns) que o agendador gastou em tempo ocioso por um cgroup em antecipação de um pedido melhor do que os pedidos que já estão em outras fila ou de outros grupos. O relatório é atualizado toda vez que um grupo não está mais em ociosidade, então se você ler este pseudofile enquanto o grupo não está mais em ociosidade, o relatório não conterá o tempo gasto no estado atual de ociosidade. Observe que este relatório está somente disponível so o **`CONFIG_DEBUG_BLK_CGROUP=y`** estiver definido no sistema.

blkio.dequeue

relaciona o número de vezes de pedidos para operações de E/S por um cgroup que estavam desenfileirados por dispositivos específicos. As entradas têm três campos: *major*, *minor*, e *number*. *Major* e *minor* são tipos de dispositivos e números de nós especificados nos *Dispositivos Alocados do Linux*, e o *number* é o número de pedidos que o grupo estava desenfileirados. Note que este relatório é disponível somente se o **`CONFIG_DEBUG_BLK_CGROUP=y`** estiver definido no sistema.

blkio.reset_stats

reinicia as estatísticas gravadas nos outros pseudofiles. Escreve um número inteiro para este arquivo para reiniciar as estatísticas para este cgroup.

3.2. CPU

O subsistema **cpu** agenda o acesso de CPU para os cgroups. Acesso aos recursos da CPU podem ser agendados de acordo com os seguintes parâmetros, cada um em um *pseudofile* separado, dentro de um sistema de arquivo virtual do cgroup:

cpu.shares

contém um valor inteiro que especifica uma parte relativa do tempo disponível da CPU para tarefas em um cgroup. Por exemplo, tarefas em dois cgroups que possuem o **cpu.shares** definido para **1** receberão tempo de CPU igual, mas as tarefas em um cgroup que possuirem um **cpu.shares** definido para **2** receberão duas vezes mais tempo de CPU de tarefas em um cgroup do que onde o **cpu.shares** estiver definido para **1**.

cpu.rt_runtime_us

especifica um tempo de período em microsegundos (µs, representado aqui como "us") para o mais longo período contínuo das quais as tarefas em um cgroup têm acesso aos recursos de CPU. Ao estabelecer este limite, você evitará que as tarefas em um cgroup monopolizem o tempo de CPU. Se as tarefas em um cgroup precisam acessar os recursos da CPU por 4 segundos a cada 5 segundos, defina o **cpu.rt_runtime_us** para **4000000** e **cpu.rt_period_us** to **5000000**.

cpu.rt_period_us

especifica um tempo de período em microsegundos (µs, representado aqui como "us") para qual a frequência que um cgroup de controle à um recurso de CPU deve ser realocado. Se as tarefas em um cgroup precisam acessar os recursos da CPU por 4 segundos a cada 5 segundos, defina o **cpu.rt_runtime_us** para **4000000** e **cpu.rt_period_us** to **5000000**.

3.3. CPUACCT

O subsistema de accounting de CPU **cpuacct** gera relatórios automáticos nos recursos de CPU usados pelas tarefas em um cgroup, incluindo tarefas em grupos filhos. Três relatórios estão disponíveis:

cpuacct.stat

reporta o número de ciclos de CPU (nas unidades definidas por **USER_HZ** no sistema) consumida por tarefas neste cgroup e seus filhos em ambos modos de usuários e modo (kernel) de sistema.

cpuacct.usage

reporta o total de tempo de CPU (em nanosegundos) consumidos por todas as tarefas neste cgroup (incluindo tarefas abaixo da hierarquia)

cpuacct.usage_percpu

reporta o tempo de CPU (em nanosegundos) consumidos em cada CPU por todas as tarefas neste cgroup (incluindo tarefas menores na hierarquia).

3.4. CPUSET

O subsistema **cpuset** atribui CPUs individuais e nós de memória para cgroups. Cada cpuset pode ser especificada de acordo com os seguintes parâmetros, cada um em um *pseudofile* separado, dentro de um sistema de arquivo virtual de cgroup:

IMPORTANTE

Alguns subsistemas possuem parâmetros obrigatórios que você precisa definir antes de mover uma tarefa em um cgroup que usa qualquer destes subsistemas. Por exemplo, antes de mover uma tarefa em um cgroup que usa o subsistema **cpuset**, os parâmetros **cpuset.cpus** e **cpuset.mems** devem estar definidos para o cgroup.

cpuset.cpus (obrigatório)

especifica as CPUs cujas tarefas deste cgroup possuem acesso. Esta é uma lista separada por vírgulas em formato ASCII, com hífens ("**-**") que representam classes. Por exemplo,

```
0-2,16
```

representa CPUs 0, 1, 2 e 16.

cpuset.mems (obrigatório)

especifica os nós de memória cujas tarefas neste cgroup possuem acesso. Esta é uma lista separada por vírgulas em formato ASCII, com hífens ("**-**") que representam classes. Por exemplo,

```
0-2,16
```

representa nós de memória 0,1,2 e 16.

cpuset.memory_migrate

contém uma sinalização (**0** ou **1**) que especifica se a página na memória deve migrar para um novo nó se os valores em **cpuset.mems** mudarem. Por padrão, a migração de memória é desabilitada (**0**) e a página fica no nó para o qual eles foram alocados originalmente, até mesmo se este nó não for mais um dos nós agora especificados em **cpuset.mems**. Se habilitado (**1**), o sistema irá migrar páginas para nós de memória dentro dos novos parâmetros especificados pelo **cpuset.mems**, mantendo sua colocação relativa se possível, por exemplo, páginas no segundo nó na lista especificada inicialmente pelo **cpuset.mems** será alocada ao segundo nó na lista agora especificada por **cpuset.mems**, se este local estiver disponível.

cpuset.cpu_exclusive

contém um sinalizador (**0** ou **1**) que especifica se outras cpusets além desta e seus pais e filhos podem compartilhar as CPUs especificadas para este cpuset. Por padrão o (**0**), CPUs não são alocadas exclusivamente à uma cpuset.

cpuset.mem_exclusive

contém uma sinalização (**0** ou **1**) que especifica se outra cpuset pode compartilhar os nós de memória especificaods para este cpuset. Por padrão (**0**), os nós de memória não são alocados exclusivamente à uma cpuset. Ao reservar os nós de memória ao uso exclusivo de uma cpuset (**1**) é funcionalmente o mesmo que habilitar uma hardwall de memória com o **cpuset.mem_hardwall**.

cpuset.mem_hardwall

contém uma sinalização (**0** ou **1**) que especifica se alocações de kernel de página de memória e dados de buffer devem ser restringidos à nós de memórias especificados por esta cpuset. Por padrão (**0**), os dados de buffer e página são compartilhados com os processos que pertencem aos usuários múltiplos. Com o hardwall habilitado (**1**) cada alocação de usuário de tarefa pode ser mantido separadamente.

cpuset.memory_pressure

um arquivo de somente leitura que contém uma média de execução de *pressão de memória* criada por processos neste cpuset. O valor neste pseudofile é automaticamente atualizado quando o **cpuset.memory_pressure_enabled** é habilitado, caso contrário o psedudofile conterá o valor **0**.

cpuset.memory_pressure_enabled

contém uma sinalização (**0** ou **1**) que especifica se o sistema deve computar a *pressão de memória* criada pelos processos neste cgroup. Os valores computados são apresentados em **cpuset.memory_pressure** e representam a taxa na qual os processos tentam liberar a memória em uso, reportada como um valor inteiro de tentativas para requerer memória por segundo, multiplicada por 1000.

cpuset.memory_spread_page

contém uma sinalização (**0** ou **1**) que especifica se os buffers de sistema de arquivo devem ser espalhados de maneira igual entre os nós de memória alocados à este cpuset. Por padrão (**0**), nenhuma tentativa é feita para espalhar páginas de memória para estes buffers de forma igual, e os buffers são colocados no mesmo nó no qual o processo que o criou estiver rodando.

cpuset.memory_spread_slab

contém uma sinalização (**0** ou **1**) que especifica se o slab do kernel agrupamento de operações de entrada/saída do arquivo, deve ser distribuído igualmente entre as cpuset. Por padrão (**0**), não é feita nenhuma tentativa de distribuir caches de slab do kernel igualmente, e os caches de slab são colocados no mesmo nó no qual o processo que o criou estiver rodando.

cpuset.sched_load_balance

contém uma sinalização (**0** ou **1**) que especifica se o kernel irá balancear as cargas nas CPUs neste cpuset. Por default (**1**), o kernel balanceia cargas movendo processos de CPUs sobrecarregadas para CPUs menos usadas.

Observe, entretanto que definir este sinalizador em um cgroup não possui efeito se o balanceador de carga estiver ativado ou em qualquer cgroup pai, já que o balanceador de carga já está sendo usado em um nível maior. Portanto para desativar o balanceador de carga em um cgroup, desative o balanceador de carga também em cada um de seus pais na hierarquia. Neste caso, você deve também considerar se o balanceador de carga deve estar ativado para quaisquer irmãos no cgroup em questão.

cpuset.sched_relax_domain_level

contém um inteiro entre **-1** e um valor positivo pequeno, o qual representa a largura da classe da CPUs nos quais o kernel deve tentar balancear cargas. Este valor não é nulo se **cpuset.sched_load_balance** estiver desabilitada.

O efeito preciso deste valor varia de acordo com a arquitetura do sistema, mas os valores seguintes são típicos:

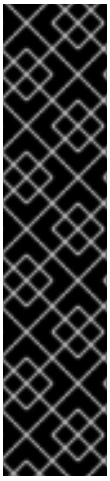
Valores de cpuset.sched_relax_domain_level

Valor	Efeito
-1	Usa o valor padrão do sistema para carregar balanceamento
0	Não realiza balanceamento de carga imediata, balanceia cargas somente periódicamente.

Valor	Efeito
1	Balanceia cargas imediatamente em todas as opções no mesmo núcleo.
2	Balanceia cargas imediatamente em núcleos do mesmo pacote
3	Balanceia cargas imediatamente em CPUs do mesmo nó ou blade
4	Balanceia cargas imediatamente em diversas CPUs em arquiteturas com acesso de memória não uniforme (NUMA)
5	Balanceia cargas imediatamente em todas as CPUs em arquiteturas com NUMA.

3.5. DEVICES

O subsistema **`devices`** permite ou nega acesso aos dispositivos por tarefas em um cgroup.



IMPORTANTE

O subsistema de dispositivo Whitelist (**`devices`**) é considerado uma Amostra de Tecnologia no Red Hat Enterprise Linux 6.

Recursos de *Amostra de Tecnologia* não são atualmente suportados nos serviços de subscrição do Red Hat Enterprise Linux 6, podem também não funcionar completamente e geralmente não estão aptos para uso em produção. Entretanto, a Red Hat inclui estes recursos no sistema operacional como uma conveniência para um cliente e para fornecer o recurso com total exposição. Você pode encontrar estes recursos úteis em um ambiente de não produção e também você é bem vindo para fornecer feedback e sugestões de funcionalidade para um recurso de amostra de tecnologia antes de tornar totalmente suportado.

devices.allow

especifica dispositivos para as quais tarefas em um cgroup possuem acesso. Cada entrada possui quatro campos: *type*, *major*, *minor*, e *access*. Os valores usados nos campos *type*, *major*, e *minor* correspondem aos tipos de dispositivos e números de nós especificados em *Linux Allocated Devices* (Dispositivos Alocados do Linux), ou conhecidos como *Linux Devices List* (Lista de Dispositivos Linux) e disponíveis em http://www.kernel.org/doc/Documentation/devices.txt.

tipo

type pode ter um dos seguintes três valores:

- **a** — se aplica à todos os dispositivos, ambos *dispositivos de carácter* e *dispositivos de bloco*
- **b** - especifica um dispositivo de bloco

- **c** - especifica um dispositivo de carácter.

maior, menor

major and *minor* são números de nós de dispositivos especificados pelo *Linux Allocated Devices*. Os números major e minor são separados por uma vírgula. Por exemplo, **8** é o número maior (major) que especifica os drives de disco SCSI, e o número menor (minor) **1** especifica a primeira partição no primeiro drive de disco do SCSI, portanto o **8:1** especifica totalmente esta partição, correspondente ao local de sistema de arquivo do **/dev/sda1**.

***** substitui todos os nós de dispositivos maiores e menores, por exemplo **9:*** (todos os dipositivos do RAID) ou ***:*** (todos os dispositivos).

access

access é a sequência de uma ou mais das seguintes letras:

- **r** — permite tarefas para ler a partir de um dispositivo especificado
- **w** — permite tarefas para gravar em um dispositivo especificado
- **m** — permite tarefas para criar arquivos de dispositivos que ainda não existem

Por exemplo, quando o *access* é especificado como **r**, as tarefas podem ler somente a partir de dispositivo especificado, mas quando o *access* é especificado como **rw**, ele pode ler ou gravar no dispositivo.

devices.deny

especifica dispositivos cujas tarefas em um cgroup não pode acessar. A sintáxe de entradas é idêntica à **devices.allow**.

devices.list

reporta os dispositivos para os quais os controles de acesso foram definidos para tarefas neste cgroup.

3.6. FREEZER

O subsistema **freezer** suspende ou retoma tarefas em um cgroup.

freezer.state

freezer.state possui três valores possíveis:

- **FROZEN** — tarefas no cgroup são suspendidas.
- **FREEZING** — o sistema está no processo de suspender tarefas no cgroup.
- **THAWED** — tarefas no cgroup foram retomadas.

Para suspender um especifico processo:

1. Mova o processo para um cgroup em uma hierarquia que possui o subsistema **freezer** anexado a ela.

2. Congele (freeze) o determinado cgroup para suspender o processo contido nele.

Não é possível mover um processo para um cgroup suspenso (congelado).

Observe que enquanto os valores **FROZEN** e **THAWED** podem ser gravados em **freezer.state**, **FREEZING** não pode ser gravado, somente lido.

3.7. MEMORY

O subsistema **memory** gera relatórios automáticos sobre recursos de memórias usados pelas tarefas em um cgroups e define limites em uso de memória por estas tarefas:

memory.stat

reporta um ampla classe de estatísticas de memória, conforme descrito na tabela:

Tabela 3.1. Valores reportados pela memory.stat

Estatística	Descrição
cache	cache de página, incluindo **tmpfs** (**shmem**), em bytes
rss	anônimo e cache de troca, *não* inclui o **tmpfs** (**shmem**), em bytes
mapped_file	tamanho dos arquivos mapeados da memória mapeada, incluindo **tmpfs** (**shmem**), em bytes
pgpgin	números de páginas paginadas na memória
pgpgout	número de páginas paginadas fora da memória
swap	o uso de troca (swap), em bytes
active_anon	anônimo e cache de troca (swap) na lista dos usados recentemente (LRU) ativa, incluindo **tmpfs** (**shmem**), em bytes
inactive_anon	anônimo e cache de troca na lista LRU inativa, incluindo o **tmpfs** (**shmem**), em bytes
active_file	memória de back up de arquivo na lista LRU ativa, em bytes
inactive_file	memória de back up de arquivo na lista LRU inativa, em bytes
unevictable	memória que não pode ser recuperada, em bytes
hierarchical_memory_limit	limite de memória para a hierarquia que contem o cgroup **memory**, em bytes

Estatística	Descrição
hierarchical_memsw_limit	memória mais o limite de troca para a hierarquia que contém o cgroup **memory**, em bytes

Adicionalmente, cada um destes arquivos que não são o **hierarchical_memory_limit** e o **hierarchical_memsw_limit** possui um duplicado pré fixado **total_** que reporta não somente no cgroup mas também em todos os filhos. Por exemplo, o **swap** reporta o uso de troca por um cgroup e o **total_swap** reporta o uso de troca total por um cgroup e todos seus filhos.

Quando você interpretar os valores reportados pelo **memory.stat**, observe que várias estatísticas são inter relacionadas:

- **active_anon** + **inactive_anon** = anonymous memory + file cache for **tmpfs** + swap cache

  Portanto o **active_anon** + **inactive_anon** ≠ **rss**, porque o **rss** não inclui o **tmpfs**.

- **active_file** + **inactive_file** = cache - size of **tmpfs**

memory.usage_in_bytes

reporta uso de memória atual total pelos processos nos cgroups (em bytes).

memory.memsw.usage_in_bytes

reporta a soma do uso atual de memória mais o espaço de troca usado por processos no cgroup (em bytes).

memory.max_usage_in_bytes

reporta o máximo de memória usada por processos nos cgroups (em bytes)

memory.memsw.max_usage_in_bytes

reporta a quantidade máxima de memória e de espaço troca usados pelos processos no cgroup (em bytes).

memory.limit_in_bytes

define a quantia máxima de memória de sistema (incluindo arquivo de cache). Se nenhuma unidade é especificada, o valor é interpretado como bytes. Entretanto, é possível usar sufixos para representar unidades maiores — **k** ou **K** para kilobytes, **m** ou **M** para Megabytes, e **g** ou **G** para Gigabytes.

Você não usar o **memory.limit_in_bytes** para limitar o cgroup root; você pode somente aplicar valores nos cgroups abaixo na hierarquia.

Digite **-1** para o **memory.limit_in_bytes** para remover quaisquer limites existentes.

memory.memsw.limit_in_bytes

define a quantia máxima para a soma de memória e uso de troca. Se nenhuma unidade for especificada, o valor é interpretado como bytes. No entanto, é possível usar sufixos para representar unidades maiores — **k** ou **K** para kilobytes, **m** ou **M** para Megabytes, e **g** ou **G** para Gigabytes.

Você não pode usar o **memory.memsw.limit_in_bytes** para limitar o cgrupo root; você pode somente aplicar valores a grupos abaixo na hierarquia.

Digite **-1** para o **memory.memsw.limit_in_bytes** para remover quaisquer limites existentes.

memory.failcnt

reporta o número de vezes que o limite de memória excedeu o valor definido no **memory.limit_in_bytes**.

memory.memsw.failcnt

reporta o número de vezes que a memória mais limite de espaço de troca alcançou o valor definido no **memory.memsw.limit_in_bytes**.

memory.force_empty

quando definido para **0**, esvazia a memória de todas as páginas usadas pelas tarefas neste cgroup. Esta interface pode ser usada somente quando o cgroup não possuir nenhuma tarefa. Se a memória não puder ser liberada, ela é movida para um cgroup pai se possível. Use o **memory.force_empty** antes de remover um cgroup para evitar páginas de cache fora de uso para o seu cgroup pai.

memory.swappiness

define a tendência do kernel para permutar memória de processo usada por tarefas neste cgroup ao invés de requerer páginas do cache de páginas. Esta é a mesma tendência, calculada da mesma forma, que o conjunto em **/proc/sys/vm/swappiness** para o sistema como um todo. O valor padrão é **60**. Valores mais baixos do que este diminuirão a tendência do kernel de permutar a memória do processo, e valores maiores que **60** aumentam a tendência do kernel de permutar a memória do processo e valores maiores que **100** permitem ao kernel permutar páginas que são parte do espaço de endereço dos processos neste cgroup.

Observe que o valor **0** não previne memória de processo a ser permutada; a permuta ainda acontece quando há escassez de memória do sistema porque o gerenciador lógico de memória virtual global não lê o valor do cgroup. Para bloquear páginas completamente, use o **mlock()** em vez de grupos.

Você não pode mudar a permuta dos seguintes grupos:

- o cgroup root, que usa a permuta definida em **/proc/sys/vm/swappiness**.
- um cgroup que tem um grupo filho abaixo dele.

memory.use_hierarchy

contém um sinalizador (**0** ou **1**) que especifica se o uso de memória deve ser contado por toda uma hierarquia de cgroups. Se ativado (**1**) o subsistema de memória requererá memória do processo filho que exceda seu limite de memória. Por padrão (**0**), o subsistema não requer memória para uma tarefa de um filho.

3.8. NET_CLS

O subsistema **net_cls** marca os pacotes de rede com um identificador de classe (classid) que permite que o controlador de trafego do Linux (**tc**) identifique os pacotes que originam de um cgroup em particular. O controlador de trafego pode ser configurado para atribuir prioridades diferentes à pacotes de cgroups diferentes.

net_cls.classid

net_cls.classid contém um valor único em formato hexadecimal que indica um controle de trafego *handle*. Por exemplo, **0x100001** representa o handle escrito tradicionalmente como **10:1** no formato usado pelo iproute2.

O formato desses handles é: **0x*AAAABBBB***, onde o *AAAA* é o número maior in hexadecimal e *BBBB* é o número menor em hexadecimal. você pode omitir os zeros à esquerda; **0x10001** é o mesmo que **0x00010001**, e representa **1:1**.

Consulte a página man do **tc** para aprender como configurar o controlador de trafego para usar os handles que o **net_cls** adiciona aos pacotes de rede.

3.9. NS

O subsistema **ns** fornece uma maneira para o grupo processar em *namespaces* separados. Dentro de um determinado namespace, os processos podem interagir um com outro mas são isolados dos processos rodando em outros namespaces. Estes namespaces separados são as vezes referidos como *containers* quando usados para virtualização de nível de sistema operacional.

3.10. RECURSOS ADICIONAIS

Documentação de Subsistemas Específicas do Kernel

Todos dos seguintes arquivos estão localizados sob o diretório **/usr/share/doc/kernel-doc-<*kernel_version*>/Documentation/cgroups/** (fornecido pelo pacote kernel-doc).

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

APÊNDICE A. HISTÓRICO DE REVISÃO

Revisão 1-4.400 **2013-10-31** **Rüdiger Landmann**
Rebuild with publican 4.0.0

Revisão 1-4 **2012-07-18** **Anthony Towns**
Rebuild for Publican 3.0

Revisão 1.0-5 **Thu May 19 2011** **Martin Prpič**
Lançamento Red Hat Enterprise Linux 6.1 GA do *Guia de Gerenciamento de Recursos*.

Revisão 1.0-4 **Tue Mar 1 2011** **Martin Prpič**
Ajustados múltiplos exemplos — BZ#667623, BZ#667676, BZ#667699
Clarificação do comando cgclear — BZ#577101
Clarificação do comando lssubsystem — BZ#678517
Congelando um processo — BZ#677548

Revisão 1.0-3 **Wed Nov 17 2010** **Rüdiger Landmann**
Corrige o exemplo de remontagem — BZ#612805

Revisão 1.0-2 **Thu Nov 11 2010** **Rüdiger Landmann**
Remove as instruções de feedback do pré lançamento

Revisão 1.0-1 **Wed Nov 10 2010** **Rüdiger Landmann**
Correções do QE — BZ#581702 and BZ#612805

Revisão 1.0-0 **Tue Nov 9 2010** **Rüdiger Landmann**
Versão de recursos completos para o GA