Red Hat Enterprise Linux 6

Handbuch zur Ressourcenverwaltung

Verwaltung von Systemressourcen unter Red Hat Enterprise Linux 6

Ausgabe 1

Last Updated: 2017-10-16

Red Hat Enterprise Linux 6 Handbuch zur Ressourcenverwaltung

Verwaltung von Systemressourcen unter Red Hat Enterprise Linux 6
Ausgabe 1

Martin Prpič
Red Hat Engineering Content Services
mprpic@redhat.com

Rüdiger Landmann
Red Hat Engineering Content Services
r.landmann@redhat.com

Douglas Silas
Red Hat Engineering Content Services
dhensley@redhat.com

Rechtlicher Hinweis

Zusammenfassung

Verwaltung von Systemressourcen unter Red Hat Enterprise Linux 6

Inhaltsverzeichnis

KAPITEL 1. EINFÜHRUNG IN KONTROLLGRUPPEN

Red Hat Enterprise Linux 6 bietet ein neues Kernel-Feature: *Kontrollgruppen*, manchmal auch kurz *Cgroups* (von engl. "Control Groups") genannt. Mithilfe von Kontrollgruppen können Sie Ressourcen — wie z.B. CPU-Zeit, Systemspeicher, Netzwerkbandbreite oder eine Kombination dieser Ressourcen — unter einer benutzerdefinierten Gruppe von Aufgaben (Prozessen) auf dem System aufteilen. Sie können die von Ihnen konfigurierten Kontrollgruppen überwachen, Kontrollgruppen den Zugriff auf bestimmte Ressourcen verwehren und sogar im laufenden Betrieb des Systems die Kontrollgruppen neu konfigurieren. Der Dienst **`cgconfig`** (»*Kontrollgruppenkonfiguration* «) kann so konfiguriert werden, dass er beim Systemstart automatisch startet und Ihre definierten Kontrollgruppen wieder einrichtet, wodurch die Kontrollgruppen über Neustarts hinweg also persistent sind.

Der Einsatz von Kontrollgruppen ermöglicht Systemadministratoren eine feingranulare Kontrolle beim Zuweisen, Priorisieren, Verwehren, Verwalten und Überwachen von Systemressourcen. Hardware-Ressourcen können geschickt auf verschiedene Aufgaben und Benutzer verteilt werden, um die Effizienz insgesamt zu steigern.

1.1. ORGANISATION VON KONTROLLGRUPPEN

Kontrollgruppen sind hierarchisch organisiert, ganz wie Prozesse, und untergeordnete Kontrollgruppen (auch "Kind" oder "Child"-Kontrollgruppen) erben einige der Attribute der übergeordneten Kontrollgruppen (auch "Eltern" oder "Parent"-Kontrollgruppen). Es gibt jedoch Unterschiede zwischen den beiden Modellen.

Das Linux-Prozessmodell

Alle Prozesse auf einem Linux-System sind einem einzigen übergeordneten Prozess untergeordnet: dem **`init`**-Prozess, der vom Kernel zum Zeitpunkt des Systemstarts gestartet wird und der andere Prozesse startet (welche wiederum eigene untergeordnete Prozesse starten können). Da alle Prozesse von einem einzigen, übergeordneten Prozess abstammen, handelt es sich beim Linux-Prozessmodell um eine einzige Hierarchie bzw. einen einzigen Baum.

Zudem erbt jeder Linux-Prozess mit Ausnahme von **`init`** die Umgebungsvariablen (wie z.B. die PATH-Variable)[1] und andere bestimmte Attribute (wie z.B. offene Dateideskriptoren) von seinem übergeordneten Prozess.

Das Kontrollgruppenmodell

Kontrollgruppen weisen einige Ähnlichkeiten mit Prozessen auf:

- Sie sind hierarchisch
- Untergeordnete Kontrollgruppen erben bestimmte Attribute ihrer übergeordneten Kontrollgruppe

Der wesentliche Unterschied besteht jedoch darin, dass viele verschiedene Hierarchien von Kontrollgruppen gleichzeitig auf dem System existieren können. Wenn das Linux-Prozessmodell ein einziger Baum von Prozessen ist, dann kann das Kontrollgruppenmodell als einer oder mehrere, miteinander nicht verbundene Bäume von Aufgaben (d.h. Prozesse) beschrieben werden.

Mehrere, separate Kontrollgruppenhierarchien sind erforderlich, da jede Hierarchie mit *einem oder mehreren Subsystemen* verknüpft ist. Ein Subsystem [2] repräsentiert eine einzelne Ressource, wie z.B. CPU-Zeit oder Speicher. Red Hat Enterprise Linux 6 bietet neun Kontrollgruppensubsysteme, die nachfolgend nach Name und Funktion aufgeführt sind.

Verfügbare Subsysteme in Red Hat Enterprise Linux

- **blkio** — dieses Subsystem setzt Grenzen für Eingabe/Ausgabe-Zugriff auf Blockgeräte wie z.B. physische Laufwerke (Festplatten, Solid State Drives, USB, etc.).
- **cpu** — dieses Subsystem verwendet den Scheduler, um Kontrollgruppenaufgaben Zugriff auf die CPU zu gewähren.
- **cpuacct** — dieses Subsystem generiert automatische Berichte über CPU-Ressourcen, die von Aufgaben in einer Kontrollgruppe verwendet werden.
- **cpuset** — dieses Subsystem weist einzelne CPUs (auf einem System mit mehreren CPUs) und Speicherknoten zu Aufgaben in einer Kontrollgruppe zu.
- **devices** — dieses Subsystem erlaubt oder verwehrt den Zugriff auf Geräte für Aufgaben in einer Kontrollgruppe.
- **freezer** — dieses Subsystem pausiert Aufgaben in einer Kontrollgruppe oder setzt diese fort.
- **memory** — dieses Subsystem setzt Grenzwerte für den Speicherverbrauch durch Aufgaben in einer Kontrollgruppe und erstellt automatische Berichte zu Speicherressourcen, die von diesen Aufgaben verwendet werden.
- **net_cls** — dieses Subsystem markiert Netzwerkpakete mit einem Class-Identifier (classid), der es dem Linux-Traffic-Controller (**tc**) ermöglicht, Pakete zu identifizieren, die von einer bestimmten Kontrollgruppe stammen.
- **ns** — das *Namensraum*-Subsystem.

ANMERKUNG

In der Literatur über Kontrollgruppen wie z.B. Handbuchseiten oder Kernel-Dokumentation stoßen Sie ggf. auf den Begriff *Ressourcen-Controller* oder kurz *Controller*. Beide Begriffe sind Synonyme für »Subsystem« und rühren daher, dass ein Subsystem normalerweise eine Ressource einplant oder eine Grenze für die Kontrollgruppe in der Hierarchie festlegt, mit der es verknüpft ist.

Die Definition eines Subsystems (Ressourcen-Controller) ist sehr allgemein: Ein Subsystem ist etwas, das auf eine Gruppe von Aufgaben, d.h. Prozessen, einwirkt.

1.2. BEZIEHUNGEN ZWISCHEN SUBSYSTEMEN, HIERARCHIEN, KONTROLLGRUPPEN UND AUFGABEN

Denken Sie daran, dass Systemprozesse in Kontrollgruppenterminologie als Aufgaben bezeichnet werden.

Sehen Sie nachfolgend einige einfache Regeln, die für die Beziehungen zwischen Subsystemen, Hierarchien von Kontrollgruppen und Aufgaben maßgeblich sind, sowie eine Erklärung der Auswirkungen dieser Regeln.

Regel 1

Jedes einzelne Subsystem (wie z.B. **cpu**) kann mit maximal einer Hierarchie verknüpft werden.

*Demnach kann das **cpu**-Subsystem nie mit zwei verschiedenen Hierarchien verknüpft werden.*

Regel 2

Eine einzelne Hierarchie kann mit einem oder mehreren Subsystemen verknüpft sein.

Demnach können die **`cpu`** *- und* **`memory`** *- Subsysteme (oder eine beliebige Anzahl an Subsystemen) mit einer einzelnen Hierarchie verknüpft werden, vorausgesetzt, dass keine von beiden mit einer anderen Hierarchie verknüpft ist.*

Regel 3

Jedes Mal, wenn eine neue Hierarchie auf dem System erstellt wird, sind alle Aufgaben auf dem System anfangs Mitglieder der standardmäßigen Kontrollgruppe dieser Hierarchie, die auch *Root-Kontrollgruppe* genannt wird. Für jede einzelne von Ihnen erstellte Hierarchie gilt, dass jede Aufgabe auf dem System Mitglied von *genau einer* Kontrollgruppe in dieser Hierarchie sein kann. Eine einzelne Aufgabe kann in mehreren Kontrollgruppen sein, vorausgesetzt, diese Kontrollgruppen befinden sich in verschiedenen Hierarchien. Sobald eine Aufgabe zu einem Mitglied einer zweiten Kontrollgruppe in derselben Hierarchie gemacht wird, wird sie von der ersten Kontrollgruppe in dieser Hierarchie entfernt. Zu keinem Zeitpunkt kann eine Aufgabe Mitglied in zwei Kontrollgruppen in derselben Hierarchie sein.

Demnach gilt, falls das **`cpu`** *- und das* **`memory`** *-Subsystem mit einer Hierarchie namens* **`cpu_and_mem`** *verknüpft sind, und das* **`net_cls`** *-Subsystem mit einer Hierarchie namens* **`net`** *, dann könnte ein laufender* **`httpd`** *-Prozess ein Mitglied in einer Kontrollgruppe in* **`cpu_and_mem`** *und in einer Kontrollgruppe in* **`net`** *sein.*

Die Kontrollgruppe in **`cpu_and_mem`** *, in der der* **`httpd`** *-Prozess Mitglied ist, kann seine CPU-Zeit auf die Hälfte dessen setzen, was anderen Prozessen zugewiesen ist, und kann seinen Speicherverbrauch auf maximal* **`1024`** *MB begrenzen. Zusätzlich kann die Kontrollgruppe in* **`net`** *, in der er ebenfalls Mitglied ist, seine Übertragungsrate auf* **`30`** *Megabytes pro Sekunde begrenzen.*

Wenn die erste Hierarchie erstellt wird, werden alle Aufgaben auf dem System Mitglieder von mindestens einer Kontrollgruppe: Der Root-Kontrollgruppe. Beim Einsatz von Kontrollgruppen ist jede Systemaufgabe demnach grundsätzlich Mitglied in mindestens einer Kontrollgruppe.

Regel 4

Jeder Prozess (Aufgabe) auf dem System, der sich selbst klont, erzeugt einen untergeordneten Prozess (Aufgabe). Der untergeordnete Prozess wird automatisch Mitglied all jener Kontrollgruppen, bei denen auch der übergeordnete Prozess Mitglied ist. Der untergeordnete Prozess kann anschließend bei Bedarf anderen Kontrollgruppen zugewiesen werden, anfänglich erbt er jedoch immer die Kontrollgruppen (in Prozessterminologie die "Umgebung") seines übergeordneten Prozesses.

Nehmen wir nun also an, dass die **`httpd`** *-Aufgabe ein Mitglied der Kontrollgruppe namens* **`half_cpu_1gb_max`** *in der* **`cpu_and_mem`** *-Hierarchie ist sowie ein Mitglied der Kontrollgruppe* **`trans_rate_30`** *in der* **`net`** *-Hierarchie. Wenn der* **`httpd`** *-Prozess sich selbst klont, werden seine untergeordneten Prozesse demnach automatisch Mitglieder der* **`half_cpu_1gb_max`** *-Kontrollgruppe und der* **`trans_rate_30`** *-Kontrollgruppe. Sie erben genau dieselben Kontrollgruppen, denen auch ihr übergeordneter Prozess angehört.*

Von diesem Zeitpunkt an sind die über- und untergeordneten Aufgaben jedoch völlig unabhängig voneinander: Wenn Sie die Kontrollgruppen ändern, denen eine Aufgabe angehört, wirkt sich dies auf keine Weise auf die andere Aufgabe aus. Ebenso wirkt sich auch das Ändern der Kontrollgruppen einer übergeordneten Aufgabe in keiner Weise auf mehrere Ebenen darunter liegende Aufgaben aus. Kurz: jegliche untergeordnete Aufgaben erben zu Beginn grundsätzlich die Mitgliedschaften zu genau denselben Kontrollgruppen wie die übergeordnete Aufgabe, doch diese Mitgliedschaften können später geändert oder gelöscht werden.

1.3. AUSWIRKUNGEN AUF DIE RESSOURCENVERWALTUNG

- Da eine Aufgabe in jeder Hierarchie nur einer einzigen Kontrollgruppe zugewiesen sein kann, gibt es nur eine Art und Weise, auf die eine Aufgabe von einem Subsystem begrenzt oder beeinflusst wird. Dies Verhalten ist logisch - also ein Feature, kein Manko.

- Sie können mehrere Subsysteme zusammen gruppieren, so dass sie alle Aufgaben in einer einzelnen Hierarchie beeinflussen. Da Kontrollgruppen in dieser Hierarchie verschiedene Parameter gesetzt haben, werden diese Aufgaben unterschiedlich beeinflusst.

- Unter Umständen kann es manchmal nötig sein, eine Hierarchie zu *refaktorieren* (umzugestalten). Ein Beispiel hierfür ist das Entfernen eines Subsystems von einer Hierarchie, die mit mehreren Subsystemen verknüpft ist, um es anschließend mit einer neuen, separaten Hierarchie zu verknüpfen.

- Falls der Bedarf, Subsysteme unter separaten Hierarchien aufzuteilen, nicht mehr besteht, können Sie umgekehrt auch eine Hierarchie entfernen und deren Subsysteme einer vorhandenen Hierarchie zuordnen.

- Dieser Aufbau ermöglicht eine sehr einfache Implementierung von Kontrollgruppen, bei der z.B. einige wenige Parameter für bestimmte Aufgaben in einer einzelnen Hierarchie festgelegt werden, die z.B. nur mit den CPU- und Speichersubsystemen verknüpft ist.

- Dieser Aufbau ermöglicht jedoch auch sehr ausgefeilte Konfigurationen: Jede Aufgabe (Prozess) auf einem System kann ein Mitglied jeder Hierarchie sein, jede davon mit einem einzelnen Subsystem verknüpft. Solch eine Konfiguration gibt dem Systemadministrator vollständige Kontrolle über alle Parameter für jede einzelne Aufgabe.

---

[1] Der übergeordnete Prozess ist dazu in der Lage, die Umgebungsvariablen zu ändern, bevor diese an den untergeordneten Prozess weitergereicht werden.

[2] Beachten Sie, dass Subsysteme in den libcgroup-Handbuchseiten und anderer Dokumentation auch *Ressourcen-Controller* oder einfach *Controller* genannt werden

KAPITEL 2. VERWENDUNG VON KONTROLLGRUPPEN

Der einfachste Weg, mit Kontrollgruppen zu arbeiten, ist durch Installation des libcgroup-Pakets, das eine Reihe von Befehlszeilentools samt zugehöriger Handbuchseiten rund um Kontrollgruppen liefert. Es ist zwar auch möglich, Hierarchien *einzuhängen* und die Kontrollgruppenparameter (nicht-persistent) mithilfe der auf jedem System verfügbaren Shell-Befehle und Hilfsprogramme einzurichten. Allerdings vereinfacht der Einsatz der Hilfsprogramme des libcgroup-Pakets den Vorgang und bietet erweiterte Optionen. Aus diesem Grund konzentriert sich dieses Handbuch auf die libcgroup-Befehle. In den meisten Fällen haben wir zusätzlich die entsprechenden Shell-Befehle angegeben, um die zugrunde liegenden Mechanismen besser zu beschreiben. Wir empfehlen jedoch die Verwendung der libcgroup-Befehle, wo immer dies möglich ist.

ANMERKUNG

Um Kontrollgruppen einsetzen zu können, vergewissern Sie sich zunächst, dass das libcgroup-Paket auf Ihrem System installiert ist, indem Sie den folgenden Befehl als Root ausführen:

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

2.1. DER CGCONFIG-DIENST

Der **cgconfig**-Dienst, der mit dem libcgroup-Paket installiert wird, bietet einen bequemen Weg zum Einrichten von Hierarchien, zum Verknüpfen von Subsystemen mit Hierarchien, und zum Verwalten von Kontrollgruppen innerhalb dieser Hierarchien. Wir empfehlen Ihnen, zur Verwaltung von Hierarchien und Kontrollgruppen auf Ihrem System **cgconfig** zu verwenden.

Der **cgconfig**-Dienst wird auf Red Hat Enterprise Linux 6 standardmäßig nicht automatisch gestartet. Wenn Sie den Dienst mit dem **chkconfig**-Befehl starten, liest er die Konfigurationsdatei der Kontrollgruppe, **/etc/cgconfig.conf**. Kontrollgruppen werden daher von Sitzung zu Sitzung neu erstellt und sind somit persistent. Abhängig vom Inhalt der Konfigurationsdatei kann **cgconfig** Hierarchien erstellen, benötigte Dateisysteme einhängen, Kontrollgruppen erstellen und Subsystemparameter für jede Gruppe setzen.

Die standardmäßige **cgconfig.conf**-Datei, die mit dem libcgroup-Paket installiert wird, erstellt eine separate Hierarchie für jedes Subsystem, hängt diese ein, und verknüpft diese Subsysteme mit diesen Hierarchien.

Wenn Sie den **cgconfig**-Dienst stoppen (mit dem Befehl **service cgconfig stop**), hängt er sämtliche Hierarchien, die er eingehängt hatte, wieder aus.

2.1.1. Die cgconfig.conf-Datei

Die **cgconfig.conf**-Datei beinhaltet zwei Haupttypen von Einträgen — *mount* und *group*. Mount-Einträge erstellen Hierarchien, hängen diese als virtuelle Dateisysteme ein und fügen Subsysteme zu diesen Hierarchien hinzu. Mount-Einträge werden unter Verwendung der folgenden Syntax definiert:

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

Siehe Beispiel 2.1, »Erstellen eines Mount-Eintrags« für ein Anwendungsbeispiel.

Beispiel 2.1. Erstellen eines Mount-Eintrags

Das folgende Beispiel erstellt eine Hierarchie für das **cpuset**-Subsystem:

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

Die entsprechenden Shell-Befehle:

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

Group-Einträge erstellen Kontrollgruppen und setzen Subsystemparameter. Group-Einträge werden unter Verwendung der folgenden Syntax definiert:

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

Beachten Sie, dass der **permissions**-Abschnitt optional ist. Um Berechtigungen für einen Gruppeneintrag zu definieren, verwenden Sie die folgende Syntax:

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

Siehe Beispiel 2.2, »Erstellen eines Gruppeneintrags« für ein Anwendungsbeispiel:

Beispiel 2.2. Erstellen eines Gruppeneintrags

Das folgende Beispiel erstellt eine Kontrollgruppe für sql-Daemons mit Berechtigungen für Benutzer in der Gruppe **sqladmin** zum Hinzufügen von Aufgaben zu der Kontrollgruppe, sowie den Benutzer **root** zum Modifizieren der Subsystemparameter.

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

```
        } admin {
            uid = root;
            gid = root;
        }
    } cpu {
        cpu.shares = 100;
    }
}
```

In Kombination mit dem Beispiel des Mount-Eintrags in Beispiel 2.1, »Erstellen eines Mount-Eintrags« lauten die entsprechenden Shell-Befehle:

```
~]# 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
```

ANMERKUNG

Sie müssen den **cgconfig**-Dienst neu starten, damit die Änderungen in der **/etc/cgconfig.conf**-Datei wirksam werden:

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

Bei der Installation von libcgroup wird eine Beispiel-Konfigurationsdatei unter **/etc/cgconfig.conf** gespeichert. Das Rautenzeichen ('#') am Anfang einer Zeile kommentiert diese Zeile aus und macht sie dadurch für den **cgconfig**-Dienst unsichtbar.

2.2. ERSTELLEN EINER HIERARCHIE UND VERKNÜPFEN VON SUBSYSTEMEN

WARNUNG

Die folgende Anleitung zum Erstellen einer neuen Hierarchie und dem Verknüpften von Subsystemen geht davon aus, dass noch keine Kontrollgruppen auf Ihrem System konfiguriert sind. In diesem Fall werden diese Anleitungen keine Auswirkungen auf den Betrieb des Systems haben. Wenn Sie dagegen anpassbare Parameter in einer Kontrollgruppe mit Aufgaben ändern, kann sich dies sofort auf diese Aufgaben auswirken. Dieses Handbuch macht Sie darauf aufmerksam, wenn zum ersten Mal die Änderung eines anpassbaren Kontrollgruppenparameters dargestellt wird, der sich auf eine oder mehrere Aufgaben auswirken kann.

Auf einem System, auf dem bereits Kontrollgruppen konfiguriert sind (entweder manuell oder mittels **cgconfig**-Dienst), werden diese Befehle fehlschlagen, wenn Sie nicht zunächst vorhandene Hierarchien aushängen - was sich jedoch auf den Betrieb des Systems auswirkt. Experimentieren Sie mit diesen Anweisungen nicht auf Produktionssystemen.

Um eine Hierarchie zu erstellen und Subsysteme damit zu verknüpfen, bearbeiten Sie den **mount**-Abschnitt der **/etc/cgconfig.conf**-Datei als Root. Einträge im **mount**-Abschnitt haben das folgende Format:

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

Beim nächsten Start von **cgconfig** wird es die Hierarchie erstellen und die Subsysteme damit verknüpfen.

Das folgende Beispiel erstellt eine Hierarchie namens **cpu_and_mem** und verknüpft die **cpu**, **cpuset**, **cpuacct** und **memory** Subsysteme damit.

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

Alternative Methode

Sie können auch Shell-Befehle und Hilfsprogramme nutzen, um die Hierarchien zu erstellen und Subsysteme damit zu verknüpfen.

Erstellen Sie als Root einen *Einhängepunkt* für die Hierarchie. Der Name des Einhängepunkts sollte den Namen der Kontrollgruppe enthalten:

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

Zum Beispiel:

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

Verwenden Sie als Nächstes den **mount**-Befehl, um die Hierarchie einzuhängen und gleichzeitig ein oder mehrere Subsysteme einzuhängen. Zum Beispiel:

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

Wobei *subsystems* eine kommagetrennte Liste mit Subsystemen und *name* der Name der Hierarchie ist. Eine kurze Beschreibung aller verfügbaren Subsysteme finden Sie in Verfügbare Subsysteme in Red Hat Enterprise Linux, und Kapitel 3, *Subsysteme und anpassbare Parameter* liefert detaillierte Erläuterungen.

Beispiel 2.3. Verwendung des mount-Befehls zum Verknüpfen mit Subsystemen

In diesem Beispiel existiert bereits ein Verzeichnis namens **/cgroup/cpu_and_mem**, das als Einhängepunkt für die Hierarchie dienen wird, die wir erstellen. Wir werden die Subsysteme **cpu**, **cpuset** und **memory** mit einer Hierarchie verknüpfen, die wir **cpu_and_mem** nennen, und mit dem **mount**-Befehl die **cpu_and_mem**-Hierarchie unter **/cgroup/cpu_and_mem** einhängen:

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

Sie können alle verfügbaren Subsysteme zusammen mit ihren derzeitigen Einhängepunkten (d.h. der Einhängepunkt der Hierarchie, mit der sie verknüpft sind) mithilfe des **lssubsys**-Befehls [3] einsehen :

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

Aus dieser Ausgabe lässt sich Folgendes ablesen:

- Die **cpu**, **cpuset** und **memory** Subsysteme sind mit einer Hierarchie verknüpft, die unter **/cgroup/cpu_and_mem** eingehängt ist
- Die **net_cls**, **ns**, **cpuacct**, **devices**, **freezer** und **blkio** Subsysteme sind bislang mit keiner Hierarchie verknüpft, wie das Fehlen eines Einhängepunktes zeigt.

2.3. VERKNÜPFEN UND ENTFERNEN VON SUBSYSTEMEN MIT/VON EINER VORHANDENEN HIERARCHIE

Um ein Subsystem mit einer vorhandenen Hierarchie zu verknüpfen, es von einer vorhandenen Hierarchie zu entfernen, oder es in eine andere Hierarchie zu verlegen, bearbeiten Sie als Root den **mount**-Abschnitt der **/etc/cgconfig.conf**-Datei, und verwenden Sie dabei dieselbe Syntax wie in Abschnitt 2.2, »Erstellen einer Hierarchie und Verknüpfen von Subsystemen« beschrieben. Beim nächsten Start von **cgconfig** wird es die Subsysteme je nach spezifizierten Hierarchien erkennen.

Alternative Methode

Um ein nicht verknüpftes Subsystem zu einer vorhandenen Hierarchie hinzuzufügen, hängen Sie die Hierarchie neu ein. Geben Sie dabei das zusätzliche Subsystem im **mount**-Befehl an, zusammen mit der **remount**-Option.

Beispiel 2.4. Neu Einhängen einer Hierarchie zum Hinzufügen eines Subsystems

Der **lssubsys**-Befehl zeigt die **cpu**, **cpuset** und **memory** Subsysteme verknüpft mit der **cpu_and_mem**-Hierarchie:

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

Hängen Sie die **cpu_and_mem**-Hierarchie neu ein. Verwenden Sie dazu die **remount**-Option und geben Sie **cpuacct** in der Liste der Subsysteme an:

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

Der **lssubsys**-Befehl zeigt nun **cpuacct** verknüpft mit der **cpu_and_mem**-Hierarchie:

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

Ebenso können Sie ein Subsystem von einer vorhandenen Hierarchie entfernen, indem Sie die Hierarchie neu einhängen und den Namen des Subsystems aus den -o Optionen weglassen. Um beispielsweise das **cpuacct**-Subsystem zu entfernen, lassen Sie es beim Einhängen einfach weg:

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

2.4. AUSHÄNGEN EINER HIERARCHIE

Sie können eine Hierarchie von Kontrollgruppen *aushängen* mithilfe des **umount**-Befehls:

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

Zum Beispiel:

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

Falls die Hierarchie derzeit leer ist (also nur die Root-Kontrollgruppe enthält), wird die Hierarchie beim Aushängen deaktiviert. Falls die Hierarchie andere Kontrollgruppen enthält, verbleibt die Hierarchie aktiv im Kernel, selbst wenn sie nicht länger eingehängt ist.

Um eine Hierarchie zu entfernen, vergewissern Sie sich, dass alle untergeordneten Kontrollgruppen entfernt wurden, bevor Sie die Hierarchie aushängen, oder verwenden Sie den **cgclear**-Befehl, mithilfe dessen Sie eine Hierarchie deaktivieren können, selbst wenn diese nicht leer ist – siehe Abschnitt 2.11, »Entladen von Kontrollgruppen « .

2.5. ERSTELLEN VON KONTROLLGRUPPEN

Verwenden Sie den **cgcreate**-Befehl, um Kontrollgruppen zu erstellen. Die Syntax für **cgcreate** lautet wie folgt: **cgcreate -t *uid*:*gid* -a *uid*:*gid* -g *subsystems*:*path*** , wobei gilt:

- **-t** (optional) – definiert einen Benutzer (durch die Benutzer-ID, uid), der Besitzer der **tasks**-Pseudodatei für diese Kontrollgruppe sein soll. Der Benutzer kann Aufgaben zur Kontrollgruppe hinzufügen.

  ANMERKUNG

  Beachten Sie, dass der einzige Weg zum Entfernen einer Aufgabe aus einer Kontrollgruppe darin besteht, diese Aufgabe in eine andere Kontrollgruppe zu verlegen. Um eine Aufgabe zu verlegen, muss der Benutzer über Schreibzugriff auf die *Ziel*-Kontrollgruppe verfügen. Schreibzugriff auf die Quell-Kontrollgruppe ist dagegen unerheblich.

- **-a** (optional) – definiert einen Benutzer (durch die Benutzer-ID, uid) und eine Gruppe (durch die Gruppen-ID, gid), der bzw. die Besitzer aller Pseudodateien außer **tasks** für diese Kontrollgruppe sein soll. Dieser Benutzer kann den Zugriff, den Prozesse in dieser Kontrollgruppe auf Systemressourcen besitzen, ändern.

- **-g** – spezifiziert die Hierarchie, in der die Kontrollgruppe erstellt werden soll, als eine kommagetrennte Liste von *subsystems* (Subsystemen), die mit diesen Hierarchien verknüpft sind. Falls sich die Subsysteme in dieser Liste in verschiedenen Hierarchien befinden, wird die Gruppe in jeder dieser Hierarchien erstellt. Der Liste der Hierarchien folgt ein Doppelpunkt und der *path* (Pfad) zur untergeordneten Gruppe relativ zur Hierarchie. Der Pfad darf nicht den Einhängepunkt der Hierarchie enthalten.

  Beispielsweise heißt die Kontrollgruppe im Verzeichnis **/cgroup/cpu_and_mem/lab1/** schlicht **lab1** – ihr Pfad ist bereit eindeutig bestimmt, da es maximal eine Hierarchie für jedes Subsystem gibt. Beachten Sie zudem, dass die Gruppe von allen Subsystemen gesteuert wird, die in den Hierarchien existieren, in der die Kontrollgruppe erstellt wird, obwohl diese Subsysteme nicht im **cgcreate**-Befehl angegeben wurden – siehe Beispiel 2.5, »cgcreate-Verwendung«.

Da alle Kontrollgruppen in derselben Hierarchie dieselben Controller haben, hat die untergeordnete Gruppe dieselben Controller wie ihre übergeordnete Gruppe.

> **Beispiel 2.5. cgcreate-Verwendung**
>
> Nehmen wir ein System an, auf dem die **cpu** und **memory** Subsysteme zusammen unter der **cpu_and_mem**-Hierarchie eingehängt sind und der **net_cls**-Controller unter einer separaten Hierarchie namens **net**. Führen wir nun Folgendes aus:

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

Der **cgcreate**-Befehl erstellt zwei Gruppen namens **test-subgroup**, eine in der **cpu_and_mem**-Hierarchie und eine in der **net**-Hierarchie. Die **test-subgroup**-Gruppe in der **cpu_and_mem**-Hierarchie wird gesteuert vom **memory**-Subsystem, obwohl wir dies nicht im **cgcreate**-Befehl angegeben haben.

Alternative Methode

Um direkt eine untergeordnete Gruppe zur Kontrollgruppe zu erstellen, verwenden Sie den Befehl **mkdir**:

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

Zum Beispiel:

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

2.6. ENTFERNEN VON KONTROLLGRUPPEN

Entfernen Sie Kontrollgruppen mit dem **cgdelete**-Befehl, der eine ähnliche Syntax wie der **cgcreate**-Befehl hat. Führen Sie Folgendes aus: **cgdelete *subsystems*:*path***, wobei gilt:

- *subsystems* ist eine kommagetrennte Liste von Subsystemen.
- *path* ist der Pfad zur Kontrollgruppe relativ zum Root der Hierarchie.

Zum Beispiel:

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

cgdelete kann mit der Option **-r** auch rekursiv alle Untergruppen entfernen.

Wenn Sie eine Kontrollgruppe löschen, werden alle ihre Aufgaben in die übergeordnete Gruppe verlegt.

2.7. EINSTELLEN VON PARAMETERN

Setzen Sie Subsystemparameter, indem Sie den **cgset**-Befehl von einem Benutzerkonto ausführen, das über die Berechtigungen zur Änderung der relevanten Gruppe verfügt. Falls beispielsweise **/cgroup/cpuset/group1** existiert, spezifizieren Sie mit dem folgenden Befehl die CPUs, auf die diese Gruppe Zugriff hat:

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

Die Syntax für **cgset** lautet: **cgset -r *parameter=value path_to_cgroup*** , wobei gilt:

- *parameter* ist der zu setzende Parameter, welcher der Datei im Verzeichnis der jeweiligen Kontrollgruppe entspricht
- *value* ist der Wert des Parameters

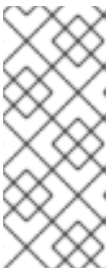
- *path_to_cgroup* ist der Pfad zur Kontrollgruppe *relativ zum Root der Hierarchie*. Um beispielsweise den Parameter der Root-Gruppe zu setzen (falls **/cgroup/cpuacct/** existiert), führen Sie Folgendes aus:

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

Alternativ können Sie auch Folgendes ausführen, da **.** relativ zur Root-Gruppe ist (d.h. die Root-Gruppe selbst):

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

Beachten Sie jedoch, dass **/** die bevorzugte Syntax ist.



ANMERKUNG

Nur eine kleine Anzahl von Parametern können für die Root-Gruppe gesetzt werden (wie z.B. der **cpuacct.usage**-Parameter, der in den obigen Beispielen gezeigt wurde). Da der Root-Gruppe sämtliche verfügbaren Ressourcen gehören, wäre es sinnlos, alle vorhandenen Prozesse zu begrenzen, indem bestimmte Parameter wie z.B. der **cpuset.cpu**-Parameter definiert werden.

Um den Parameter von **group1** einzustellen, die eine Untergruppe der Root-Gruppe ist, führen Sie Folgendes aus:

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

Der nachgestellte Schrägstrich am Ende des Gruppennamens (z.B. **cpuacct.usage=0 group1/**) ist optional.

Die Werte, die Sie mit **cgset** einstellen können, hängen ggf. von den Werten ab, die an höherer Stelle in der Hierarchie eingestellt wurden. Falls beispielsweise **group1** ausschließlich CPU 0 auf einem System verwenden darf, können Sie **group1/subgroup1** nicht zur Verwendung von CPUs 0 und 1 oder zur ausschließlichen Verwendung von CPU 1 einstellen.

Sie können auch **cgset** verwenden, um die Parameter aus einer Kontrollgruppe in eine andere, vorhandene Kontrollgruppe zu kopieren. Zum Beispiel:

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

Die Syntax zum Kopieren von Parametern mit **cgset** lautet: **cgset --copy-from *path_to_source_cgroup path_to_target_cgroup***, wobei gilt:

- *path_to_source_cgroup* ist der Pfad zur Kontrollgruppe, deren Parameter kopiert werden sollen, relativ zur Root-Gruppe der Hierarchie
- *path_to_target_cgroup* ist der Pfad zur Ziel-Kontrollgruppe, relativ zur Root-Gruppe der Hierarchie

Vergewissern Sie sich, dass alle obligatorischen Parameter (falls vorhanden) für die verschiedenen Subsysteme gesetzt sind, bevor Sie Parameter von einer Gruppe auf eine andere kopieren, da ansonsten der Befehl fehlschlagen wird. Weitere Informationen über obligatorische Parameter finden Sie in Wichtig — Pflichtparameter.

Alternative Methode

Um die Parameter einer Kontrollgruppe direkt einzustellen, fügen Sie Werte in die relevante Pseudodatei des Subsystems unter Verwendung des **echo**-Befehls ein. Der folgende Befehl fügt beispielsweise den Wert **0-1** in die Pseudodatei **cpuset.cpus** der Kontrollgruppe **group1** ein:

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

Mit diesem Wert werden die Aufgaben in dieser Kontrollgruppe auf CPUs 0 und 1 auf dem System beschränkt.

2.8. VERSCHIEBEN EINES PROZESSES IN EINE KONTROLLGRUPPE

Sie können einen Prozess in eine Kontrollgruppe verschieben, indem Sie den Befehl **cgclassify** ausführen:

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

Die Syntax für **cgclassify** lautet: **cgclassify -g *subsystems*:*path_to_cgroup pidlist***, wobei gilt:

- *subsystems* ist eine kommagetrennte Liste mit Subsystemen, oder *****, zum Starten der Prozesse in den Hierarchien, die mit allen verfügbaren Subsystemen verknüpft sind. Falls Kontrollgruppen desselben Namens in mehreren Hierarchien existieren, beachten Sie, dass die **-g** Option die Prozesse in jeder dieser Gruppen verlegt. Vergewissern Sie sich, dass die Kontrollgruppe innerhalb aller Hierarchien existiert, deren Subsysteme Sie hier spezifizieren.
- *path_to_cgroup* ist der Pfad zur Kontrollgruppe innerhalb ihrer Hierarchien
- *pidlist* ist eine kommagetrennte Liste von *Process Identifier* (PIDs)

Sie können auch die Option **--sticky** vor der Prozess-ID *pid* einfügen, um jegliche untergeordnete Prozesse in derselben Kontrollgruppe beizubehalten. Wenn Sie diese Option nicht setzen und der **cgred**-Daemon ausgeführt wird, werden untergeordnete Prozesse Kontrollgruppen anhand der in **/etc/cgrules.conf** gefundenen Einstellungen zugewiesen. Der Prozess selbst jedoch bleibt in der Kontrollgruppe, in der er gestartet wurde.

Mithilfe von **cgclassify** können Sie mehrere Prozesse gleichzeitig verschieben. Dieser Befehl verschiebt beispielsweise die Prozesse mit den PIDs **1701** und **1138** in die Kontrollgruppe **group1/**:

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

Beachten Sie, dass die PIDs, die verschoben werden sollen, durch Leerzeichen getrennt sind und dass sich die angegebenen Kontrollgruppen in verschiedenen Hierarchien befinden sollten.

Alternative Methode

Schreiben Sie den *Prozess-Identifier* (PID) in die **tasks**-Datei einer Kontrollgruppe, um einen Prozess direkt in diese Kontrollgruppe zu verschieben. Um beispielsweise einen Prozess mit der PID **1701** in eine Kontrollgruppe unter **/cgroup/lab1/group1/** zu verschieben:

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

2.8.1. Der cgred-Daemon

Cgred ist ein Daemon, der Prozesse anhand der in der Datei **/etc/cgrules.conf** gesetzten Parameter in Kontrollgruppen verschiebt. Einträge in der **/etc/cgrules.conf**-Datei können in einer der folgenden zwei Formate vorliegen:

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

Zum Beispiel:

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

Dieser Eintrag definiert, dass alle Prozesse, die dem Benutzer namens **maria** gehören, auf das **devices**-Subsystem anhand der in der Kontrollgruppe **/usergroup/staff** spezifizierten Parameter zugreifen. Um bestimmte Befehle mit bestimmten Kontrollgruppen zu verknüpfen, fügen Sie den Parameter *command* wie folgt hinzu:

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

Dieser Eintrag definiert nun, dass wenn der Benutzer namens **maria** den **ftp**-Befehl verwendet, der Prozess automatisch in die **/usergroup/staff/ftp**-Kontrollgruppe in der Hierarchie verlegt wird, die das **devices**-Subsystem enthält. Beachten Sie jedoch, dass der Daemon den Prozess erst in die Kontrollgruppe verlegt, nachdem die entsprechende Bedingung erfüllt wurde. Deshalb ist es möglich, dass der **ftp**-Prozess für kurze Zeit in der falschen Gruppe läuft. Falls der Prozess sofort Unterprozesse startet, während er sich noch in der falschen Gruppe befindet, werden diese Unterprozesse unter Umständen nicht verschoben.

Einträge in der **/etc/cgrules.conf**-Datei können die folgende, zusätzliche Notation enthalten:

- **@** — wird dies als Präfix für *user* verwendet, kennzeichnet dies eine Gruppe statt eines einzelnen Benutzers. So sind **@admins** beispielsweise alle Benutzer in der Gruppe **admins**.
- ***** — steht für "alle". So steht ***** im Feld **subsystem** beispielsweise für alle Subsysteme.
- **%** — steht für ein Element, das dem Element in der darüber liegenden Zeile entspricht. Zum Beispiel:

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

2.9. STARTEN EINES PROZESSES IN EINER KONTROLLGRUPPE

WICHTIG

Einige Controller besitzen Pflichtparameter, die Sie setzen müssen, bevor Sie eine Aufgabe in eine Kontrollgruppe verlegen können, die eine dieser Subsysteme verwendet. Bevor Sie beispielsweise eine Aufgabe in eine Kontrollgruppe verlegen, die das **cpuset**-Subsystem verwendet, müssen die Parameter **cpuset.cpus** und **cpuset.mems** definiert sein.

Die Beispiele in diesem Abschnitt verdeutlichen die korrekte Syntax für den Befehl, funktionieren jedoch nur auf Systemen, auf denen die relevanten Pflichtparameter für jegliche in diesen Beispielen verwendeten Controller gesetzt wurden. Falls Sie die relevanten Controller nicht bereits konfiguriert haben, können Sie die Beispiel-Befehle aus diesem Abschnitt nicht direkt kopieren und damit rechnen, dass sie auf Ihrem System funktionieren.

Siehe Abschnitt 3.10, »Weitere Informationsquellen« für eine Beschreibung, welche Parameter für bestimmte Subsysteme obligatorisch sind.

Sie können Prozesse in einer Kontrollgruppe starten, indem Sie den **cgexec**-Befehl ausführen. Dieser Befehl startet beispielsweise den **lynx**-Webbrowser innerhalb der **group1**-Kontrollgruppe unter Berücksichtigung der Beschränkungen, die das **cpu**-Subsystem dieser Gruppe auferlegt:

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

Die Syntax für **cgexec** lautet: **cgexec -g *subsystems*:*path_to_cgroup* *command* *arguments***, wobei gilt:

- *subsystems* ist eine kommagetrennte Liste mit Subsystemen, oder *****, zum Starten der Prozesse in den Hierarchien, die mit allen verfügbaren Subsystemen verknüpft sind. Falls Kontrollgruppen desselben Namens in mehreren Hierarchien existieren, beachten Sie (wie bereits beim **cgset**-Befehl in Abschnitt 2.7, »Einstellen von Parametern« beschrieben), dass die **-g** Option die Prozesse in jeder dieser Gruppen erstellt. Vergewissern Sie sich, dass die Kontrollgruppe innerhalb aller Hierarchien existiert, deren Subsysteme Sie hier spezifizieren.
- *path_to_cgroup* ist der Pfad zu der Kontrollgruppe relativ zur Hierarchie.
- *command* ist der auszuführende Befehl
- *arguments* sind etwaige Parameter für den Befehl

Sie können auch die Option **--sticky** vor *command* einfügen, um jegliche untergeordneten Prozesse in derselben Kontrollgruppe zu halten. Wenn Sie diese Option nicht setzen und der **cgred**-Daemon ausgeführt wird, werden untergeordnete Prozesse anhand der in **/etc/cgrules.conf** gefundenen Einstellungen den Kontrollgruppen zugewiesen. Der Prozess selbst jedoch bleibt in der Kontrollgruppe, in der er gestartet wurde.

Alternative Methode

Wenn Sie einen neuen Prozess starten, erbt dieser die Gruppe seines übergeordneten Prozesses. Demzufolge können Sie, um einen Prozess in einer bestimmten Kontrollgruppe zu starten, Ihren Shell-Prozess auch in die gewünschte Gruppe verlegen (siehe Abschnitt 2.8, »Verschieben eines Prozesses in eine Kontrollgruppe«) und anschließend den Prozess von dieser Shell aus starten. Zum Beispiel:

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

Beachten Sie, dass nach dem Beenden von **lynx** Ihre existierende Shell sich immer noch in der **group1**-Kontrollgruppe befindet. Ein noch besseres Vorgehen ist deshalb:

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

2.9.1. Starten eines Dienstes in einer Kontrollgruppe

Sie können bestimmte Dienste in einer Kontrollgruppe starten. Dienste, die in Kontrollgruppen gestartet werden können, müssen die folgenden Eigenschaften haben:

- sie müssen eine **/etc/sysconfig/*servicename***-Datei verwenden
- sie müssen die **daemon()**-Funktion von **/etc/init.d/functions** verwenden, um den Dienst zu starten

Damit ein Dienst, der diese Voraussetzungen erfüllt, in einer Kontrollgruppe startet, fügen Sie in dessen Datei im **/etc/sysconfig**-Verzeichnis einen Eintrag in der Form **CGROUP_DAEMON="*subsystem*:*control_group*"** ein, wobei *subsystem* ein mit einer bestimmten Hierarchie verknüpftes Subsystem und *control_group* eine Kontrollgruppe in dieser Hierarchie ist. Zum Beispiel:

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

2.10. ABRUFEN VON INFORMATIONEN ZU KONTROLLGRUPPEN

2.10.1. Suchen von Prozessen

Führen Sie den folgenden Befehl aus, um die Kontrollgruppe zu ermitteln, zu der ein Prozess gehört:

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

Führen Sie den folgenden Befehl aus, wenn Sie die PID für den Prozess herausfinden möchten:

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

2.10.2. Suchen von Subsystemen

Führen Sie den folgenden Befehl aus, um die Subsysteme zu finden, die in Ihrem Kernel verfügbar sind, sowie deren Anordnung in Hierarchien:

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

Führen Sie den folgenden Befehl aus, um stattdessen die Einhängepunkte von bestimmten Subsystemen herauszufinden:

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

wobei *subsystems* eine Liste der Subsysteme ist, für die Sie die Informationen wünschen. Beachten Sie, dass der **lssubsys -m**-Befehl nur den Einhängepunkt der obersten Ebene für jede Hierarchie ausgibt.

2.10.3. Suchen von Hierarchien

Wir empfehlen Ihnen, Hierarchien unter **/cgroup** einzuhängen. Ist dies auf Ihrem System der Fall, sehen Sie sich einfach den Inhalt dieses Verzeichnisses an, um eine Liste der Hierarchien zu erhalten. Falls **tree** auf Ihrem System installiert ist, führen Sie es aus, um einen Überblick über alle Hierarchien und der darin enthaltenen Kontrollgruppen zu erhalten:

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

2.10.4. Suchen von Kontrollgruppen

Führen Sie den folgenden Befehl aus, um die Kontrollgruppen auf einem System anzuzeigen:

```
~]$ lscgroup
```

Sie können die Ausgabe auf eine bestimmte Hierarchie beschränken, indem Sie einen Controller und den Pfad im Format ***controller*:*path*** angeben. Zum Beispiel:

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

Dieser Befehl listet nur Untergruppen der **adminusers**-Kontrollgruppe in der Hierarchie auf, mit der das **cpuset**-Subsystem verknüpft ist.

2.10.5. Anzeigen der Parameter von Kontrollgruppen

Führen Sie den folgenden Befehl aus, um die Parameter von bestimmten Kontrollgruppen anzuzeigen:

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

wobei *parameter* eine Pseudodatei ist, die Werte für ein Subsystem enthält, und *list_of_cgroups* eine Liste mit kommagetrennten Kontrollgruppen ist. Zum Beispiel:

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

Dieser Befehl zeigt die Werte für **cpuset.cpus** und **memory.limit_in_bytes** für die Kontrollgruppen **lab1** und **lab2**.

Falls Sie die Namen der Parameter selbst nicht kennen, nutzen Sie einen Befehl wie:

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

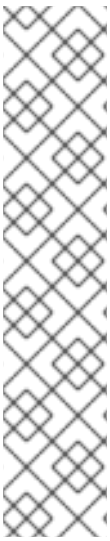
2.11. ENTLADEN VON KONTROLLGRUPPEN

WARNUNG

Der Befehl **cgclear** zerstört alle Kontrollgruppen in allen Hierarchien. Falls Sie diese Hierarchien nicht in einer Konfigurationsdatei gespeichert haben, wird es schwierig sein, sie zu rekonstruieren.

Verwenden Sie den Befehl **cgclear**, um ein vollständiges Kontrollgruppendateisystem zu entfernen.

Alle Aufgaben in der Kontrollgruppe werden daraufhin dem Root-Knoten der Hierarchien zugewiesen, alle Kontrollgruppen werden entfernt und das Dateisystem selbst wird aus dem System ausgehängt, wodurch alle vormals eingehängten Hierarchien zerstört werden. Abschließend wird das Verzeichnis gelöscht, unter dem das Kontrollgruppendateisystem eingehängt war.



ANMERKUNG

Wenn Sie den **mount**-Befehl zum Erstellen von Kontrollgruppen verwenden (anstelle des **cgconfig**-Dienstes), wird ein Eintrag in der **/etc/mtab**-Datei erstellt (die Tabelle der eingehängten Dateisysteme). Diese Änderung erfolgt ebenso in der **/proc/mounts**-Datei. Wenn jedoch Kontrollgruppen mit dem **cgclear**-Befehl und anderen **cgconfig**-Befehlen entladen werden, wird eine direkte Kernel-Schnittstelle verwendet, welche die Änderungen nicht in die **/etc/mtab**-Datei überträgt, sondern die neuen Informationen nur in die **/proc/mounts**-Datei schreibt. Aus diesem Grund sind die ausgehängten Kontrollgruppen nach dem Entladen durch den **cgclear**-Befehl unter Umständen nach wie vor in der **/etc/mtab**-Datei sichtbar und werden deshalb auch noch beim Ausführen des **mount**-Befehls angezeigt. In diesen Fällen ist es empfehlenswert, sich für eine genaue Aufstellung aller eingehängten Kontrollgruppen auf die **/proc/mounts**-Datei zu beziehen.

2.12. WEITERE INFORMATIONSQUELLEN

Werfen Sie für eine definitive Dokumentation der Kontrollgruppenbefehle einen Blick auf die Handbuchseiten, die mit dem libcgroup-Paket geliefert werden. Die Abschnittsnummern finden Sie in der nachfolgenden Liste mit Handbuchseiten.

Die libcgroup-Handbuchseiten

- **man 1 cgclassify** – der **cgclassify**-Befehl wird verwendet, um laufende Aufgaben auf eine oder mehrere Kontrollgruppen zu verlegen.

  man 1 cgclear – der **cgclear**-Befehl wird verwendet, um alle Kontrollgruppen in einer Hierarchie zu löschen.

  man 5 cgconfig.conf – Kontrollgruppen werden in der **cgconfig.conf**-Datei definiert.

  man 8 cgconfigparser – der **cgconfigparser**-Befehl analysiert die **cgconfig.conf**-Datei und hängt Hierarchien ein.

  man 1 cgcreate – der **cgcreate**-Befehl erstellt neue Kontrollgruppen in Hierarchien.

man 1 cgdelete – der **cgdelete**-Befehl entfernt die angegebenen Kontrollgruppen.

man 1 cgexec – der **cgexec**-Befehl führt Aufgaben in den angegebenen Kontrollgruppen aus.

man 1 cgget – der **cgget**-Befehl zeigt Kontrollgruppenparameter an.

man 5 cgred.conf – **cgred.conf** ist die Konfigurationsdatei für den **cgred**-Dienst.

man 5 cgrules.conf – **cgrules.conf** enthält die Regeln, anhand derer bestimmt wird, wann Aufgaben zu bestimmten Kontrollgruppen gehören.

man 8 cgrulesengd – der **cgrulesengd**-Dienst verteilt Aufgaben auf Kontrollgruppen.

man 1 cgset – der **cgset**-Befehl setzt Parameter für eine Kontrollgruppe.

man 1 lscgroup – der **lscgroup**-Befehl listet die Kontrollgruppen in einer Hierarchie auf.

man 1 lssubsys – der **lssubsys**-Befehl listet die Hierarchien auf, die die angegebenen Subsysteme enthalten.

---

[3] Der **lssubsys**-Befehl ist Teil der Hilfsprogramme, die vomlibcgroup-Paket bereitgestellt werden. Sie müssen libcgroup installieren, um ihn verwenden zu können: Siehe Kapitel 2, *Verwendung von Kontrollgruppen*, falls Sie **lssubsys** nicht ausführen können.

KAPITEL 3. SUBSYSTEME UND ANPASSBARE PARAMETER

Subsysteme sind Kernel-Module, die sich der Kontrollgruppen bewusst sind. Typischerweise sind dies Ressourcen-Controller, die verschiedenen Kontrollgruppen in unterschiedlichem Umfang Systemressourcen zuweisen. Subsysteme könnten jedoch für jede andere Interaktion mit dem Kernel programmiert werden, bei der der Bedarf besteht, verschiedene Gruppen von Prozessen unterschiedlich zu behandeln. Das *Application Programming Interface*(auch kurz API oder Programmierschnittstelle genannt) zur Entwicklung neuer Subsysteme ist in **cgroups.txt** in der Kernel-Dokumentation dokumentiert, die auf Ihrem System unter **/usr/share/doc/kernel-doc-*kernel-version*/Documentation/cgroups/** installiert ist (geliefert vom kernel-doc-Paket). Die neueste Version der Kontrollgruppendokumentation steht zudem online unter http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt zur Verfügung. Beachten Sie jedoch, dass sich die in der aktuellsten Dokumentation genannten Features unter Umständen von jenen unterscheiden, die in dem Kernel auf Ihrem System zur Verfügung stehen.

Statusobjekte, die Subsystemparameter für eine Kontrollgruppe beinhalten, werden als *Pseudodateien* innerhalb des virtuellen Dateisystems der Kontrollgruppe dargestellt. Diese Pseudodateien können mithilfe von Shell-Befehlen, oder ihren entsprechenden Systemaufrufen bearbeitet werden. So ist **cpuset.cpus** beispielsweise eine Pseudodatei, die definiert, auf welche CPUs eine Kontrollgruppe zugreifen darf. Angenommen, **/cgroup/cpuset/webserver** ist eine Kontrollgruppe für den Webserver, der auf einem System läuft, und wir führen den folgenden Befehl aus:

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

Der Wert **0,2** wird dadurch in die **cpuset.cpus**-Pseudodatei geschrieben und schränkt daher jegliche Aufgaben, deren PIDs in **/cgroup/cpuset/webserver/tasks** aufgeführt sind, so ein, dass nur CPU 0 und CPU 2 auf dem System verwendet werden dürfen.

3.1. BLKIO

Das Block-I/O-Subsystem (**blkio**) überwacht und steuert den I/O-Zugriff auf Blockgeräte von Aufgaben in Kontrollgruppen. Werden in einige dieser Pseudodateien Werte geschrieben, schränkt dies den Zugriff oder die Bandbreite ein; werden von einigen dieser Pseudodateien Werte gelesen, liefert dies Informationen über I/O-Operationen.

blkio.weight

spezifiziert den relativen Anteil (das *Gewicht*) von Block I/O-Zugriff, der einer Kontrollgruppe standardmäßig zur Verfügung steht, im Bereich **100** bis **1000**. Dieser Wert wird für bestimmte Geräte durch den **blkio.weight_device**-Parameter außer Kraft gesetzt. Um einer Kontrollgruppe beispielsweise ein standardmäßiges Gewicht von **500** für den Zugriff auf Blockgeräte zuzuweisen, führen Sie Folgendes aus:

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

blkio.weight_device

spezifiziert den relativen Anteil (das *Gewicht*) von Block I/O-Zugriff, der einer Kontrollgruppe standardmäßig für bestimmte Geräte zur Verfügung steht, im Bereich **100** bis **1000**. Der Wert dieses Parameters setzt den Wert des **blkio.weight**-Parameters für die angegebenen Geräte außer Kraft. Werte werden im Format *major:minor weight* angegeben, wobei *major* und *minor* die in *Linux Allocated Devices* (auch *Linux-Geräteliste* genannt, verfügbar unter http://www.kernel.org/doc/Documentation/devices.txt) spezifizierten Gerätetypen und Knotennummern sind. Um einer Kontrollgruppe beispielsweise ein Gewicht von **500** für den Zugriff auf **/dev/sda** zuzuweisen, führen Sie Folgendes aus:

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

In der *Linux Allocated Devices* Notation steht **8:0** für **/dev/sda**.

blkio.time

zeigt die Zeit an, die eine Kontrollgruppe I/O-Zugriff auf bestimmte Geräte hatte. Jeder Eintrag umfasst drei Felder: *major*, *minor* und *time*. *major* und *minor* sind die Gerätetypen und Knotennummern, die in den *Linux Allocated Devices* definiert sind. *time* ist die Zeitspanne in Millisekunden (ms).

blkio.sectors

zeigt die Anzahl an Sektoren an, die auf bestimmte Geräte bzw. von bestimmten Geräten durch eine Kontrollgruppe übertragen wurden. Jeder Eintrag umfasst drei Felder: *major*, *minor* und *sectors*. *major* und *minor* sind die Gerätetypen und Knotennummern, die in den *Linux Allocated Devices* definiert sind. *sectors* ist die Anzahl der Plattensektoren.

blkio.io_service_bytes

zeigt die Anzahl an Bytes an, die auf bestimmte Geräte bzw. von bestimmten Geräten durch eine Kontrollgruppe übertragen wurden. Jeder Eintrag umfasst vier Felder: *major*, *minor*, *operation* und *bytes*. *major* und *minor* sind die Gerätetypen und Knotennummern, die in den *Linux Allocated Devices* definiert sind. *operation* steht für die Art der Operation (**read**, **write**, **sync** oder **async**) und *bytes* ist die Anzahl der übertragenen Bytes.

blkio.io_serviced

zeigt die Anzahl an I/O-Operationen an, die auf bestimmten Geräten von einer Kontrollgruppe durchgeführt wurden. Jeder Eintrag umfasst vier Felder: *major*, *minor*, *operation* und *number*. *major* und *minor* sind die Gerätetypen und Knotennummern, die in den *Linux Allocated Devices* definiert sind. *operation* steht für die Art der Operation (**read**, **write**, **sync** oder **async**) und *number* ist die Anzahl der Operationen.

blkio.io_service_time

zeigt die Zeitspanne zwischen der Absendung einer Anfrage und deren Abschluss bei I/O-Operationen an, die auf bestimmten Geräten von einer Kontrollgruppe durchgeführt wurden. Jeder Eintrag umfasst vier Felder: *major*, *minor*, *operation* und *time*. *major* und *minor* sind die Gerätetypen und Knotennummern, die in den *Linux Allocated Devices* definiert sind. *operation* steht für die Art der Operation (**read**, **write**, **sync** oder **async**) und *time* ist die Zeitspanne in Nanosekunden (ns). Die Zeit wird in Nanosekunden angegeben statt in größeren Einheiten, damit dieser Bericht auch für Solid-State-Geräte aussagekräftig ist.

blkio.io_wait_time

zeigt die Zeitspanne an, die I/O-Operationen auf bestimmten Geräten von Kontrollgruppen auf den Service der Scheduler-Queues warten. Wenn Sie diesen Bericht interpretieren, beachten Sie Folgendes:

- Die ausgewiesene Zeit kann größer sein als die Zeit, die insgesamt vergangen ist, da die ausgewiesene Zeit die kumulierte Zeit aller I/O-Operationen für die Kontrollgruppe darstellt, nicht die Zeit, die die Kontrollgruppe selbst auf I/O-Operationen gewartet hat. Um die Zeit herauszufinden, die die Gruppe als Ganzes gewartet hat, verwenden Sie **blkio.group_wait_time**.

- Falls das Gerät eine **queue_depth** > 1 hat, beinhaltet die ausgewiesene Zeit nur die Zeit, bis die Anfrage zum Gerät gesendet wird, nicht die Wartezeit, während das Gerät die Anfragen neu ordnet.

Jeder Eintrag umfasst vier Felder: *major*, *minor*, *operation*, and *bytes*. *major* und *minor* sind die Gerätetypen und Knotennummern, die in den *Linux Allocated Devices* definiert sind. *operation* steht für die Art der Operation (**read**, **write**, **sync** oder **async**) und *time* ist die Zeitspanne in Nanosekunden (ns). Die Zeit wird in Nanosekunden angegeben statt in größeren Einheiten, damit dieser Bericht auch für Solid-State-Geräte aussagekräftig ist.

blkio.io_merged

zeigt die Anzahl an BIOS-Anfragen an, die in Anfragen für I/O-Operationen von einer Kontrollgruppe zusammengeführt werden. Jeder Eintrag umfasst zwei Felder: *number* und *operation*. *number* ist die Anzahl der Anfragen und *operation* steht für die Art der Operation (**read**, **write**, **sync** oder **async**).

blkio.io_queued

zeigt die Anzahl an Anfragen an, die in der Warteschlange stehen für I/O-Operationen von einer Kontrollgruppe. Jeder Eintrag umfasst zwei Felder: *number* und *operation*. *number* ist die Anzahl der Anfragen und *operation* steht für die Art der Operation (**read**, **write**, **sync** oder **async**).

blkio.avg_queue_size

zeigt die durchschnittliche Größe der Warteschlange für I/O-Operationen von einer Kontrollgruppe an, über die Gesamtdauer der Existenz der Gruppe hinweg. Die Größe der Warteschlange wird jedes Mal ermittelt, wenn eine Warteschlange für diese Kontrollgruppe ein Zeitfenster zugeteilt bekommt. Beachten Sie, dass dieser Bericht nur verfügbar ist, falls **CONFIG_DEBUG_BLK_CGROUP=y** auf dem System gesetzt ist.

blkio.group_wait_time

zeigt die Gesamtdauer (in Nanosekunden — ns) an, die eine Kontrollgruppe auf die Zuteilung eines Zeitfensters für eine ihrer Warteschlangen wartet. Der Bericht wird jedes Mal aktualisiert, wenn eine Warteschlange für diese Kontrollgruppe ein Zeitfenster erhält. Falls Sie also diese Pseudodatei lesen, während die Kontrollgruppe auf ein Zeitfenster wartet, enthält der Bericht nicht die Zeit, die auf die Operation gewartet wird, die derzeit in der Warteschlange steht. Beachten Sie, dass dieser Bericht nur verfügbar ist, falls **CONFIG_DEBUG_BLK_CGROUP=y** auf dem System gesetzt ist.

blkio.empty_time

zeigt die Gesamtdauer (in Nanosekunden — ns) an, die eine Kontrollgruppe ohne jegliche ausstehende Anfragen ist. Der Bericht wird jedes Mal aktualisiert, wenn eine Warteschlange für diese Kontrollgruppe eine ausstehende Anfrage hat. Falls Sie also diese Pseudodatei lesen, während die Kontrollgruppe keine ausstehende Anfragen hat, enthält der Bericht nicht die Zeit, die in diesem Moment im leeren Zustand verbracht wird. Beachten Sie, dass dieser Bericht nur verfügbar ist, falls **CONFIG_DEBUG_BLK_CGROUP=y** auf dem System gesetzt ist.

blkio.idle_time

zeigt die Gesamtdauer (in Nanosekunden — ns) an, die der Scheduler für eine Kontrollgruppe untätig bleibt, während er auf eine bessere Anfrage wartet als jene, die bereits in anderen Warteschlangen stehen oder die von anderen Kontrollgruppen stammen. Der Bericht wird jedes Mal aktualisiert, wenn eine Gruppe nicht länger untätig ist. Falls Sie also diese Pseudodatei lesen,

während die Kontrollgruppe untätig ist, enthält der Bericht nicht die Zeit, die derzeit untätig verbracht wird. Beachten Sie, dass dieser Bericht nur verfügbar ist, falls **CONFIG_DEBUG_BLK_CGROUP=y** auf dem System gesetzt ist.

blkio.dequeue

zeigt an, wie oft Anfragen für I/O-Operationen von einer Kontrollgruppe von bestimmten Geräten aus der Warteschlange entfernt wurden. Jeder Eintrag umfasst drei Felder: *major*, *minor* und *number*. *major* und *minor* sind die Gerätetypen und Knotennummern, die in den *Linux Allocated Devices* definiert sind. *number* ist die Anzahl der Anfragen der Gruppe, die aus der Warteschlange entfernt wurden. Beachten Sie, dass dieser Bericht nur verfügbar ist, falls **CONFIG_DEBUG_BLK_CGROUP=y** auf dem System gesetzt ist.

blkio.reset_stats

setzt die in anderen Pseudodateien aufgezeichneten Statistiken zurück. Schreiben Sie einen ganzzahligen Wert in diese Datei, um die Statistiken für diese Kontrollgruppe zurückzusetzen.

3.2. CPU

Das **cpu**-Subsystem disponiert CPU-Zugriff für Kontrollgruppen. Der Zugriff auf CPU-Ressourcen kann anhand der folgenden Parameter disponiert werden, wobei sich jeder in einer separaten *Pseudodatei* innerhalb des virtuellen Dateisystems der Kontrollgruppe befindet:

cpu.shares

beinhaltet einen ganzzahligen Wert, der einen relativen Anteil der CPU-Zeit bestimmt, der den Aufgaben in einer Kontrollgruppe zur Verfügung steht. Aufgaben in zwei Kontrollgruppen, bei denen **cpu.shares** auf **1** gesetzt ist, erhalten beispielsweise die gleiche CPU-Zeit. Aufgaben in einer Kontrollgruppe, bei der **cpu.shares** auf **2** gesetzt ist, erhalten die doppelte CPU-Zeit im Vergleich zu Aufgaben in einer Kontrollgruppe, bei denen **cpu.shares** auf **1** gesetzt ist.

cpu.rt_runtime_us

spezifiziert eine Zeitspanne in Mikrosekunden (µs, hier als "us" dargestellt) für die längste kontinuierliche Periode, in der die Aufgaben in einer Kontrollgruppe Zugriff auf CPU-Ressourcen besitzen. Das Einrichten dieser Grenze verhindert, dass Aufgaben in einer Kontrollgruppe CPU-Zeit für sich allein beanspruchen. Falls die Aufgaben in einer Kontrollgruppe in der Lage sein sollten, mindestens vier von fünf Sekunden auf CPU-Ressourcen zuzugreifen, setzen Sie **cpu.rt_runtime_us** auf **4000000** und **cpu.rt_period_us** auf **5000000**.

cpu.rt_period_us

spezifiziert eine Zeitspanne in Mikrosekunden (µs, hier als "us" dargestellt), wie häufig der Zugriff einer Kontrollgruppe auf CPU-Ressourcen neu zugewiesen werden soll. Falls die Aufgaben in einer Kontrollgruppe in der Lage sein sollten, mindestens vier von fünf Sekunden auf CPU-Ressourcen zuzugreifen, setzen Sie **cpu.rt_runtime_us** auf **4000000** und **cpu.rt_period_us** auf **5000000**.

3.3. CPUACCT

Das CPU-Accounting-Subsystem (**cpuacct**) generiert automatische Berichte zu CPU-Ressourcen, die von Aufgaben in einer Kontrollgruppe (einschließlich der Aufgaben von untergeordneten Gruppen) beansprucht werden. Drei Berichte stehen zur Verfügung:

cpuacct.stat

zeigt die Anzahl der CPU-Zyklen (in den auf dem System durch **USER_HZ** definierten Einheiten) an, die von Aufgaben in dieser Kontrollgruppe samt Untergruppen sowohl in Benutzer-Modus und System (Kernel-) Modus beansprucht werden.

cpuacct.usage

zeigt die gesamte CPU-Zeit (in Nanosekunden) an, die von allen Aufgaben in dieser Kontrollgruppe (inklusive in dieser Hierarchie untergeordnete Aufgaben) beansprucht wird.

cpuacct.usage_percpu

zeigt die CPU-Zeit (in Nanosekunden) an, die auf jeder CPU von allen Aufgaben in dieser Kontrollgruppe beansprucht wird (inklusive in der Hierarchie untergeordnete Prozesse).

3.4. CPUSET

Das **cpuset**-Subsystem weist Kontrollgruppen individuelle CPUs und Speicherknoten zu. Jedes 'cpuset' kann anhand der folgenden Parameter bestimmt werden, wobei sich jeder in einer separaten *Pseudodatei* innerhalb des virtuellen Dateisystems der Kontrollgruppe befindet:

WICHTIG

Einige Controller besitzen Pflichtparameter, die Sie setzen müssen, bevor Sie eine Aufgabe in eine Kontrollgruppe verlegen können, die eine dieser Subsysteme verwendet. Bevor Sie beispielsweise eine Aufgabe in eine Kontrollgruppe verlegen, die das **cpuset**-Subsystem verwendet, müssen die Parameter **cpuset.cpus** und **cpuset.mems** definiert sein.

cpuset.cpus (obligatorisch)

spezifiziert die CPUs, auf die Aufgaben in dieser Kontrollgruppe Zugriff besitzen. Dies ist eine kommagetrennte Liste im ASCII-Format mit Bindestrichen ("-") zur Darstellung von Bereichen. Zum Beispiel:

```
0-2,16
```

repräsentiert CPUs 0, 1, 2 und 16.

cpuset.mems (obligatorisch)

spezifiziert die Speicherknoten, auf die Aufgaben in dieser Kontrollgruppe Zugriff besitzen. Dies ist eine kommagetrennte Liste im ASCII-Format mit Bindestrichen ("-") zur Darstellung von Bereichen. Zum Beispiel:

```
0-2,16
```

repräsentiert Speicherknoten 0, 1, 2 und 16.

cpuset.memory_migrate

beinhaltet ein Flag (**0** oder **1**), das festlegt, ob eine Seite im Speicher auf einen neuen Knoten migrieren soll, falls sich die Werte in **cpuset.mems** ändern. Standardmäßig ist Speicher-Migration deaktiviert (**0**) und Seiten bleiben auf dem Knoten, dem sie ursprünglich zugewiesen wurden, auch

wenn dieser Knoten nicht länger einer der Knoten ist, die aktuell in **cpuset.mems** definiert sind. Falls aktiviert (**1**), migriert das System Seiten auf Speicherknoten im Rahmen der durch **cpuset.mems** definierten neuen Parameter und behält deren relative Platzierung bei, falls möglich. So werden beispielsweise Seiten auf dem zweiten Knoten auf der Liste, der ursprünglich via **cpuset.mems** definiert wurde, dem zweiten Knoten auf der Liste zugewiesen, jetzt definiert durch **cpuset.mems**, falls dieser Platz zur Verfügung steht.

cpuset.cpu_exclusive

beinhaltet ein Flag (**0** oder **1**), das definiert, ob andere 'cpusets' und deren über- und untergeordneten Sets die für diese 'cpuset' definierten CPUs gemeinsam nutzen können. Standardmäßig (**0**) werden keine CPUs exklusiv nur einem 'cpuset' zugewiesen.

cpuset.mem_exclusive

beinhaltet ein Flag (**0** oder **1**), das definiert, ob andere 'cpusets' die für diese 'cpuset' definierten Speicherknoten gemeinsam nutzen können. Standardmäßig (**0**) werden keine Speicherknoten exklusiv nur einem 'cpuset' zugewiesen. Das Reservieren von Speicherknoten für den exklusiven Gebrauch von einem 'cpuset' (**1**) ist im Endeffekt das gleiche, wie die Aktivierung eines Speicher-Hardwalls mit **cpuset.mem_hardwall**.

cpuset.mem_hardwall

beinhaltet ein Flag (**0** oder **1**), das definiert, ob Kernel-Zuweisungen von Speicherseiten und Puffer-Daten auf die Speicherknoten beschränkt werden soll, die für dieses 'cpuset' angegeben wurden. Standardmäßig (**0**) werden Seiten und Puffer-Daten von Prozessen, die zu mehreren Benutzern gehören, gemeinsam genutzt. Mit einer aktivierten Hardwall (**1**) kann die Benutzerzuweisung für jede Aufgabe getrennt gehalten werden.

cpuset.memory_pressure

eine schreibgeschützte Datei, die einen laufenden Durchschnittswert des *Speicherdrucks* enthält, der von Prozessen in diesem 'cpuset' erzeugt wird. Der Wert in dieser Pseudodatei wird automatisch aktualisiert, wenn **cpuset.memory_pressure_enabled** aktiviert ist. Ansonsten enthält die Pseudodatei den Wert **0**.

cpuset.memory_pressure_enabled

beinhaltet ein Flag (**0** oder **1**), das definiert, ob das System den von Prozessen in dieser Kontrollgruppe generierten *Speicherdruck* errechnen soll. Errechnete Werte werden in **cpuset.memory_pressure** ausgegeben und repräsentiert die Rate, mit der Prozesse derzeit verwendeten Speicher freizusetzen versuchen. Diese Rate wird als ganzzahliger Wert der Anzahl der Versuche zur Zurückgewinnung von Speicher pro Sekunde, multipliziert mit 1000, dargestellt.

cpuset.memory_spread_page

beinhaltet ein Flag (**0** oder **1**), das definiert, ob der Puffer des Dateisystems gleichmäßig auf den für dieses 'cpuset' zugewiesenen Speicherknoten verteilt werden soll. Standardmäßig (**0**) werden keine Versuche unternommen, Speicherseiten für diesen Puffer gleichmäßig zu verteilen und Puffer werden auf demselben Knoten platziert, auf dem der Prozess läuft, der sie erstellte.

cpuset.memory_spread_slab

beinhaltet ein Flag (**0** oder **1**), das definiert, ob Kernel-Slab-Caches für Datei-Eingabe/-Ausgabe Operationen gleichmäßig auf dem 'cpuset' verteilt werden sollen. Standardmäßig (**0**) werden keine Versuche unternommen, um Kernel-Slab-Caches gleichmäßig zu verteilen und Slab-Caches werden auf demselben Knoten platziert, auf dem der Prozess, der sie erstellte, läuft.

cpuset.sched_load_balance

beinhaltet ein Flag (`0` oder `1`), das definiert, ob der Kernel in diesem 'cpuset' die Lasten zwischen den CPUs verteilt. Standardmäßig (`1`) verteilt der Kernel Lasten, indem er Prozesse von überlasteten CPUs auf weniger ausgelastete CPUs verlegt.

Beachten Sie jedoch, dass das Setzen dieses Flags in einer Kontrollgruppe keinerlei Auswirkungen hat, wenn die Lastverteilung in einer übergeordneten Kontrollgruppe aktiviert ist, da in diesem Fall die Lastverteilung bereits auf höherer Ebene stattfindet. Um die Lastverteilung in einer Kontrollgruppe zu deaktivieren, müssen Sie demzufolge auch in allen übergeordneten Gruppen in der Hierarchie die Lastverteilung deaktivieren. In diesem Fall sollten Sie berücksichtigen, ob die Lastverteilung für andere Gruppen auf gleicher Ebene (also "Geschwister" der fraglichen Kontrollgruppe) aktiviert sein sollte.

cpuset.sched_relax_domain_level

beinhaltet einen ganzzahligen Wert zwischen `-1` und einem kleinen, positiven Wert, welcher die Bandbreite der CPUs darstellt, innerhalb welcher der Kernel versuchen soll, die Last zu verteilen. Dieser Wert ist bedeutungslos, wenn **`cpuset.sched_load_balance`** deaktiviert ist.

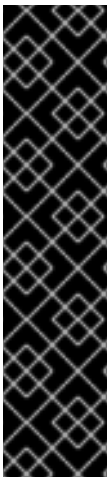
Der genaue Effekt dieses Wertes variiert abhängig von der Systemarchitektur, doch die folgenden Werte sind typisch:

Die Werte von cpuset.sched_relax_domain_level

Wert	Effekt
`-1`	System-Standardwert für Lastverteilung verwenden
`0`	Keine unmittelbare Lastverteilung durchführen. Lastverteilung erfolgt nur periodisch
`1`	Umgehend die Last auf Threads auf demselben Kern verteilen
`2`	Umgehend die Last auf Kernen in demselben Paket verteilen
`3`	Umgehend die Last auf CPUs auf demselben Knoten oder Blade verteilen
`4`	Umgehend die Last auf mehreren CPUs auf Architekturen mit Non-Uniform Memory Access (NUMA) verteilen
`5`	Umgehend die Last auf allen CPUs auf Architekturen mit NUMA verteilen

3.5. DEVICES

Das **devices**-Subsystem ermöglicht oder verwehrt Aufgaben in einer Kontrollgruppe Zugriff auf Geräte.



WICHTIG

Das Device-Whitelist-Subsystem (**devices**) ist als Technologievorschau in Red Hat Enterprise Linux 6 enthalten.

Technologievorschau-Features werden derzeit unter Red Hat Enterprise Linux 6 Subskriptionsdiensten nicht unterstützt, funktionieren möglicherweise nicht vollständig und sind im Wesentlichen ungeeignet für die Verwendung in der Produktion. Dennoch sind die Features für höheren Kundenkomfort und zur Verbesserung des Bekanntheitsgrads der Features enthalten. Diese Features sind unter Umständen hilfreich in Nicht-Produktionsumgebungen. Kritik und Vorschläge für weitere Funktionalitäten sind für diese Technologievorschau-Features ausdrücklich erwünscht, bevor diese vollständig unterstützt werden.

devices.allow

spezifiziert die Geräte, auf die Aufgaben in einer Kontrollgruppe Zugriff haben. Jeder Eintrag besitzt vier Felder: *type*, *major*, *minor* und *access*. Die in den Feldern *type*, *major* und *minor* verwendeten Werte entsprechen Gerätetypen und Knotennummern, die in den *Linux Allocated Devices*, auch bekannt als die *Linux-Geräteliste*, definiert sind und unter http://www.kernel.org/doc/Documentation/devices.txt abrufbar ist.

type

type kann einen der drei folgenden Werte besitzen:

- **a** — umfasst alle Geräte, sowohl *zeichenorientierte Geräte* als auch *blockorientierte Geräte*
- **b** — definiert ein Blockgerät
- **c** — definiert ein Zeichengerät

major, minor

major und *minor* sind via *Linux Allocated Devices* definierte Geräteknotennummern. Die Major- und Minor-Nummern werden durch einen Doppelpunkt getrennt. Ein Beispiel: **8** ist die Major-Nummer, die SCSI-Plattenlaufwerke definiert und die Minor-Nummer **1** definiert die erste Partition auf dem ersten SCSI-Plattenlaufwerk. **8:1** definiert daher komplett diese Partition und entspricht der Position im Dateisystem **/dev/sda1**.

***** steht für alle Major- bzw. alle Minor-Geräteknoten, z.B. **9:*** (alle RAID-Geräte) oder ***:*** (alle Geräte).

access

access entspricht einer Sequenz aus einem oder mehreren Buchstaben:

- **r** — ermöglicht Prozessen das Lesen von dem angegebenen Gerät.
- **w** — ermöglicht Prozessen das Schreiben auf das angegebene Gerät.
- **m** — ermöglicht Prozessen das Erstellen von Gerätedateien, die noch nicht existieren.

Wenn *access* beispielsweise als `r` definiert wird, können Prozesse nur von dem angegebenen Gerät lesen. Wenn *access* jedoch als `rw` definiert wird, können Prozesse von dem Gerät lesen und auf dieses schreiben.

devices.deny

spezifiziert Geräte, auf die Aufgaben in einer Kontrollgruppe keinen Zugriff haben. Die Syntax der Einträge entspricht `devices.allow`.

devices.list

zeigt die Geräte an, für die die Zugriffskontrolle für Aufgaben in dieser Kontrollgruppe eingestellt wurde.

3.6. FREEZER

Das `freezer`-Subsystem setzt Aufgaben in einer Kontrollgruppe zeitweilig aus oder setzt sie fort.

freezer.state

`freezer.state` hat drei mögliche Werte:

- **FROZEN** — Aufgaben in der Kontrollgruppe sind ausgesetzt.
- **FREEZING** — Das System ist gerade dabei, Aufgaben in der Kontrollgruppe auszusetzen.
- **THAWED** — Aufgaben in der Kontrollgruppe wurden fortgesetzt.

Führen Sie die folgenden Schritte aus, um einen bestimmten Prozess auszusetzen:

1. Verlegen Sie den Prozess in eine Kontrollgruppe in einer Hierarchie, mit der das `freezer`-Subsystem verknüpft ist.
2. Setzen Sie diese Kontrollgruppe aus, um den darin enthaltenen Prozess auszusetzen.

Es ist nicht möglich, einen Prozess in eine ausgesetzte ("FROZEN") Kontrollgruppe zu verlegen.

Beachten Sie, dass Werte für **FROZEN** und **THAWED** nach `freezer.state` geschrieben werden können, wohingegen **FREEZING** nur gelesen, nicht aber geschrieben werden kann.

3.7. MEMORY

Das `memory`-Subsystem erstellt automatische Berichte zu Speicherressourcen, die von den Aufgaben in einer Kontrollgruppe verwendet werden und setzt Grenzwerte für den Speicherverbrauch durch diese Aufgaben fest.

memory.stat

zeigt eine größere Bandbreite von Speicher-Statistiken an, wie in der folgenden Tabelle beschrieben:

Tabelle 3.1. Werte, die von memory.stat angezeigt werden

Statistik	Beschreibung
cache	Seiten-Cache, inklusive **tmpfs** (**shmem**), in Bytes
rss	anonymer und Swap-Cache-Speicher, *exklusive* **tmpfs** (**shmem**), in Bytes
mapped_file	Größe von Memory-Mapped Dateien, inklusive **tmpfs** (**shmem**), in Bytes
pgpgin	Anzahl der in den Seitenspeicher geschriebenen Seiten
pgpgout	Anzahl der aus dem Seitenspeicher gelesenen Seiten
swap	Swap-Verwendung, in Bytes
active_anon	anonymer und Swap-Cache-Speicher auf aktiver least-recently-used (LRU) Liste, inklusive **tmpfs** (**shmem**), in Bytes
inactive_anon	anonymer und Swap-Cache-Speicher auf inaktiver LRU-Liste, inklusive **tmpfs** (**shmem**), in Bytes
active_file	Dateigestützter Speicher auf aktiver LRU-Liste, in Bytes
inactive_file	Dateigestützter Speicher auf inaktiver LRU-Liste, in Bytes
unevictable	Speicher, der nicht zurückgefordert werden kann, in Bytes
hierarchical_memory_limit	Speichergrenze für die Hierarchie, welche die **memory**-Kontrollgruppe enthält, in Bytes
hierarchical_memsw_limit	Grenze für Speicher plus Swap für die Hierarchie, welche die **memory**-Kontrollgruppe enthält, in Bytes

Zusätzlich verfügt jede dieser Dateien (mit Ausnahme von **hierarchical_memory_limit** und **hierarchical_memsw_limit**) über ein Gegenstück mit vorangestelltem **total_**, das nicht nur über die Kontrollgruppe berichtet, sondern auch über alle zugehörigen Untergruppen. Zum Beispiel zeigt **swap** den Swap-Verbrauch einer Kontrollgruppe an, und **total_swap** zeigt den gesamten Swap-Verbrauch einer Kontrollgruppe einschließlich aller untergeordneten Gruppen an.

Wenn Sie die von **memory.stat** angezeigten Werte interpretieren, beachten Sie, wie die verschiedenen Statistiken zusammenhängen:

- **active_anon** + **inactive_anon** = anonymer Speicher + Datei-Cache für **tmpfs** + Swap-Cache

  Demzufolge gilt: **active_anon** + **inactive_anon** ≠ **rss**, da **rss** nicht **tmpfs** einschließt.

- **active_file** + **inactive_file** = Cache - Größe von **tmpfs**

memory.usage_in_bytes

zeigt den aktuellen gesamten Speicherverbrauch von Prozessen in der Kontrollgruppe an (in Bytes).

memory.memsw.usage_in_bytes

zeigt die Summe des aktuellen Speicherverbrauchs plus Swap-Space-Verbrauch von Prozessen in der Kontrollgruppe an (in Bytes).

memory.max_usage_in_bytes

zeigt den maximalen Speicherverbrauch von Prozessen in der Kontrollgruppe an (in Bytes).

memory.memsw.max_usage_in_bytes

zeigt den maximalen Swap- und Speicherverbrauch von Prozessen in der Kontrollgruppe an (in Bytes).

memory.limit_in_bytes

legt die maximale Menge an Benutzerspeicher fest (inklusive Datei-Cache). Falls keine Einheiten angegeben werden, wird dieser Wert als Bytes interpretiert. Es ist jedoch möglich, Suffixe zur Darstellung von größeren Einheiten zu verwenden — **k** oder **K** für Kilobytes, **m** oder **M** für Megabytes und **g** oder **G** für Gigabytes.

Sie können **memory.limit_in_bytes** nicht dazu verwenden, um die Root-Kontrollgruppe zu begrenzen; nur Gruppen an niedrigerer Stelle in der Hierarchie können Sie Werte zuordnen.

Schreiben Sie **-1** in **memory.limit_in_bytes**, um jegliche vorhandene Grenzen zu entfernen.

memory.memsw.limit_in_bytes

legt die maximale Menge für die Summe von Speicher und Swap-Space fest. Falls keine Einheiten angegeben werden, wird dieser Wert als Bytes interpretiert. Es ist jedoch möglich, Suffixe zur Darstellung von größeren Einheiten zu verwenden — **k** oder **K** für Kilobytes, **m** oder **M** für Megabytes und **g** oder **G** für Gigabytes.

Sie können **memory.memsw.limit_in_bytes** nicht dazu verwenden, um die Root-Kontrollgruppe zu begrenzen; nur Gruppen an niedrigerer Stelle in der Hierarchie können Sie Werte zuordnen.

Schreiben Sie **-1** in **memory.memsw.limit_in_bytes**, um jegliche vorhandene Grenzen zu entfernen.

memory.failcnt

zeigt an, wie oft der in **memory.limit_in_bytes** festgelegte Wert für die Speichergrenze erreicht wurde.

memory.memsw.failcnt

zeigt an, wie oft der in **memory.memsw.limit_in_bytes** festgelegte Grenzwert für die Summe von Speicher und Swap-Space erreicht wurde.

memory.force_empty

wenn auf **0** gesetzt, wird der Speicher von allen Seiten gelöscht, die von Aufgaben in dieser Kontrollgruppe verwendet werden. Diese Schnittstelle kann nur benutzt werden, wenn die Kontrollgruppe keine Aufgaben besitzt. Falls kein Speicher freigesetzt werden kann, wird es in eine

übergeordnete Kontrollgruppe verschoben, falls möglich. Verwenden Sie **memory.force_empty**, bevor Sie eine Kontrollgruppe verschieben, um das Verschieben von nicht mehr verwendeten Seiten-Caches auf die übergeordnete Kontrollgruppe zu vermeiden.

memory.swappiness

spezifiziert die Tendenz des Kernels, Prozessspeicher, der von Aufgaben in dieser Kontrollgruppe beansprucht wird, aus dem Speicher auszulagern (Swap), anstatt die Seiten vom Seiten-Cache zurückzufordern. Dies entspricht der Tendenz, wie sie auch für das gesamte System in **/proc/sys/vm/swappiness** definiert wird und wird genauso berechnet. Der Standardwert ist **60**. Werte unter **60** verringern die Tendenz des Kernels, Prozessspeicher auszulagern. Werte über **60** erhöhen die Tendenz des Kernels, Prozessspeicher auszulagern und bei Werten größer als **100** beginnt der Kernel, Seiten auszulagern, die Teil des Adressraums der Prozesse in dieser Kontrollgruppe sind.

Beachten Sie, dass der Wert **0** nicht verhindert, dass Prozessspeicher ausgelagert wird; die Auslagerung kann nach wie vor erfolgen, wenn Systemspeicher knapp wird, da die globale Logik zur Verwaltung des virtuellen Speichers den Wert der Kontrollgruppe nicht liest. Um Seiten vollständig zu sperren, verwenden Sie **mlock()** anstelle von Kontrollgruppen.

Sie können Tendenz zur Auslagerung (Swappiness) für die folgenden Gruppen nicht ändern:

- die Root-Kontrollgruppe, welche die Swappiness-Angabe aus **/proc/sys/vm/swappiness** verwendet.
- eine Kontrollgruppe, die über untergeordnete Gruppen verfügt.

memory.use_hierarchy

beinhaltet ein Flag (**0** oder **1**), das definiert, ob Speicherverbrauch in einer Hierarchie von Kontrollgruppen berücksichtigt werden soll. Falls aktiviert (**1**), fordert der Speicher-Controller Speicher von untergeordneten Prozessen sowie von solchen Prozessen, die ihre Speichergrenze überschreiten, zurück. Standardmäßig (**0**) fordert der Controller keinen Speicher von untergeordneten Aufgaben zurück.

3.8. NET_CLS

Das **net_cls**-Subsystem markiert Netzwerkpakete mit einem Class-Identifier (classid), der es dem Linux-Traffic-Controller (**tc**) ermöglicht, Pakete zu identifizieren, die von einer bestimmten Kontrollgruppe stammen. Der Traffic-Controller kann so konfiguriert werden, dass Paketen aus verschiedenen Kontrollgruppen verschiedene Prioritäten zugewiesen werden.

net_cls.classid

net_cls.classid enthält einen einzelnen Wert in Hexadezimal-Format, der ein Traffic-Control-*Handle* anzeigt. So stellt **0x100001** beispielsweise das Handle dar, das im von iproute2 verwendeten Format konventionell **10:1** geschrieben wird.

Das Format für diese Handles lautet: **0x*AAAABBBB***, wobei *AAAA* die Major-Nummer in Hexadezimal und *BBBB* die Minor-Nummer in Hexadezimal ist. Führende Nullen können Sie weglassen; **0x10001** ist dasselbe wie **0x00010001** und steht für **1:1**.

Werfen Sie einen Blick auf die Handbuchseite von **tc** für Informationen darüber, wie Sie den Traffic-Controller zur Verwendung der Handles konfigurieren, die **net_cls** zu den Netzwerkpaketen hinzufügt.

3.9. NS

Das **ns**-Subsystem bietet einen Weg, um Prozesse in separate *Namensräume* zu gruppieren. Innerhalb eines bestimmten Namensraums können Prozesse miteinander agieren, sind jedoch von den Prozessen in anderen Namensräumen isoliert. Diese separaten Namensräume werden manchmal als *Container* bezeichnet, wenn sie zur Virtualisierung auf Betriebssystemebene eingesetzt werden.

3.10. WEITERE INFORMATIONSQUELLEN

Subsystem-spezifische Kernel-Dokumentation

Alle der folgenden Dateien befinden sich unter dem **/usr/share/doc/kernel-doc-<*kernel_version*>/Documentation/cgroups/**-Verzeichnis (welches vom kernel-doc-Paket gestellt wird).

- **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**

ANHANG A. VERSIONSGESCHICHTE

Version 1-3.400 **2013-10-31** **Rüdiger Landmann**
Rebuild with publican 4.0.0

Version 1-3 **2012-07-18** **Anthony Towns**
Rebuild for Publican 3.0

Version 1.0-5 **Thu May 19 2011** **Martin Prpič**
Red Hat Enterprise Linux 6.1 GA Release des *Handbuch zur Ressourcenverwaltung*.

Version 1.0-4 **Tue Mar 1 2011** **Martin Prpič**
Mehrere Beispiele korrigiert — BZ#667623, BZ#667676, BZ#667699
Verdeutlichung des cgclear-Befehls — BZ#577101
Verdeutlichung des lssubsystem-Befehls — BZ#678517
Einfrieren eines Prozesses — BZ#677548

Version 1.0-3 **Wed Nov 17 2010** **Rüdiger Landmann**
Remount-Beispiel korrigiert — BZ#612805

Version 1.0-2 **Thu Nov 11 2010** **Rüdiger Landmann**
Feedback-Anleitung für Vorab-Version entfernt

Version 1.0-1 **Wed Nov 10 2010** **Rüdiger Landmann**
Korrekturen von QE (Qualitätssicherung) —BZ#581702 and BZ#612805

Version 1.0-0 **Tue Nov 9 2010** **Rüdiger Landmann**
Vollständige Version für GA