Red Hat Enterprise Linux 7

リソース管理ガイド

cgroups を使用した RHEL 7 のシステムリソースの管理

Last Updated: 2021-07-15

Red Hat Enterprise Linux 7 リソース管理ガイド

cgroups を使用した RHEL 7 のシステムリソースの管理

Marie Doleželová
Red Hat Customer Content Services
mdolezel@redhat.com

Milan Navrátil
Red Hat Customer Content Services

Eva Majoršinová
Red Hat Customer Content Services

Peter Ondrejka
Red Hat Customer Content Services

Douglas Silas
Red Hat Customer Content Services

Martin Prpič
Red Hat Product Security

Rüdiger Landmann
Red Hat Customer Content Services

法律上の通知

概要

本書では、Red Hat Enterprise Linux 7 でこのようなリソースの CPU 時間、システムメモリー、ネットワーク帯域幅、組み合わせなどのリソースを割り当て可能な Linux カーネル機能、Linux カーネル機能について説明します。

目次

第1章 コントロールグループの概要(CGROUPS)

1.1. コントロールグループとは

本ガイドの *cgroups と呼ばれるコントロールグループは*、このようなリソースの CPU 時間、システムメモリー、ネットワーク帯域幅、システムで実行されているプロセスの階層順など、リソースの割り当てを可能にする Linux カーネル機能です。cgroups を使用することで、システム管理者は、システムリソースの割り当て、優先順位化、拒否、管理、監視まで細かく制御できます。ハードウェアリソースは、アプリケーションとユーザーに簡単に分割でき、全体的な効率を高めます。

コントロールグループは、階層的にグループやラベルのプロセスを階層的に適用し、リソース制限をそれらに適用する方法を提供します。従来は、すべてのプロセスがプロセスがプロセスが、プロセスの **niceness 値で変更できるシステムリソースの同様の量を受け取る可能性があります**。このアプローチにより、これらのアプリケーションの相対インポートに関係なく、多数のプロセスが複数のプロセスで受信するプロセス数を超えるリソースを受信するアプリケーションのことです。

Red Hat Enterprise Linux 7 では、cgroup 階層のシステムを systemd ユニットツリーにバインドすることにより、リソース管理設定をプロセスレベルからアプリケーションレベルに移行します。したがって、システムリソースの管理は、systemctl **コマンドを使用するか**、systemd ユニットファイルを変更して行うことができます。詳しくは *2章コントロールグループの使用*、を参照してください。

以前のバージョンの Red Hat Enterprise Linux では、libcgroup パッケージからの **cgconfig** コマンドを使用してカスタムの cgroup 階層を構築していたシステム管理者です。このパッケージは非推奨となっており、デフォルトの cgroup 階層との競合を簡単に作成できるため、使用は推奨されません。ただし、libcgroup **は、systemd** がまだ該当しない場合（特に **net-prio** サブシステムを使用）など、特定のケースに対応できます。*3章libcgroup ツールの使用*。

上記のツールは、Linux カーネルの cgroup コントローラー（サブシステムとしても知られている）と対話するための高レベルのインターフェースを提供します。リソース管理の主な cgroup コントローラーは **cpu、memory、** *および* **blkio** *Red Hat Enterprise Linux 7 で利用可能なコントローラー。コントローラー固有のカーネルドキュメント。*

1.2. デフォルトの CGROUP HIERARCHIES

デフォルトでは、systemd *はスライス、スコープ、およびサービスユニットの階層を自動的に作成し、cgroup ツリーに統一された構造を提供します。* ***systemctl*** *「コントロールグループの作成」。* ***また、systemd*** *Red Hat Enterprise Linux 7 で利用可能なコントローラー* ***/sys/fs/cgroup/*** *ディレクトリーに自動的にマウントします。*

警告

libcgroup *パッケージの非推奨の* ***cgconfig ツールは、systemd*** *でまだサポートされていないコントローラーの階層をマウントして処理するために利用できます（特に* ***net-prio*** *コントローラー）。* ***libcgropup ツールを使用して、systemd*** *がマウントするデフォルトの階層を変更する場合、予期しない動作が発生するためです。* ***libcgroup*** *ライブラリーは、Red Hat Enterprise Linux の今後のバージョンで削除されます。* ***cgconfig*** *の使用方法については、3章libcgroup ツールの使用。*

systemd のユニットタイプ

*システムで実行しているすべてのプロセスは、**systemd** init プロセスの子プロセスです。systemd は、リソース制御の目的に使用される 3 つのユニットタイプを提供します（**systemd** のユニットタイプの完全なリストについては、『Red Hat Enterprise Linux 7 システム管理者のガイド』の 「systemd によるサービス管理」の章を参照）。*

- ***サービス** - ユニット設定ファイルに基づいて **systemd** が起動したプロセスまたはプロセスのグループ。サービスは、指定したプロセスをカプセル化して、1 つのセットとして起動および停止できるようにします。サービスの名前は以下の方法で指定されます。*

  > *name.**service***

  name は、サービス名を表します。

- ***スコープ** - 外部で作成されたプロセスのグループ。スコープは、**fork()** 関数を介して任意のプロセスで開始および停止されたプロセスをカプセル化し、ランタイム時に **systemd** により登録されます。たとえば、ユーザーセッション、コンテナー、および仮想マシンはスコープとして処理されます。スコープの名前は以下のように指定されます。*

  > *name.**scope***

  ここで、name はスコープ名を表します。

- ***スライス** - 階層的に編成されたユニットのグループ。スライスにはプロセスが含まれておらず、スコープおよびサービスを配置する階層を編成します。実際のプロセスはスコープまたはサービスに含まれます。階層ツリーでは、スライスユニットの全名前は、階層内の場所へのパスに対応します。ダッシュ("**-**")文字は、パスコンポーネントの区切り文字として機能します。たとえば、スライスの名前は以下のようになります。*

  > *parent-name.**slice***

  *これは、親-name.**slice** が親スライスのサブ **スライスであることを意味します。このスライスには、parent -name -name 2. slice などの独自のサブ スライスを指定できます。***

  以下のように 1 つの root スライスが表示されます。

  > ***-.slice***

サービス、スコープ、スライスユニットは cgroup ツリーのオブジェクトに直接マッピングされます。これらのユニットがアクティブ化されると、ユニット名から構築される cgroup パスに直接マッピングされます。 たとえば、test-waldo.slice に配置された ex.service ress は cgroup test.slice/test-waldo.slice/ex.service/ にマッピングされます。

サービス、スコープ、スライスは、システム管理者、またはプログラムにより動的に作成されます。デフォルトでは、オペレーティングシステムは、システムの実行に必要な多くの組み込みサービスを定義します。また、デフォルトで 4 つのスライスが作成されます。

- **-.slice - root スライス。**
- **system.slice: すべてのシステムサービスのデフォルト場所です。**
- **user.slice - ユーザーセッションのデフォルトの場所です。**
- **machine.slice: すべての仮想マシンおよび Linux コンテナーのデフォルト場所です。**

ユーザーセッションはすべて、自動的にスコープユニットと、仮想マシンおよびコンテナーのプロセスに配置されることに注意してください。さらに、すべてのユーザーが暗黙的なサブスライスと共に割り当てられます。上記のデフォルト設定に加えて、システム管理者は新しいスライスを定義し、サービスやスコープをそれらに割り当てることができます。

以下のツリーは、cgroup ツリーの簡素化された例です。この出力は、で説明されている **systemd-cgls** 「コントロールグループに関する情報の取得」。

```
├─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 20
├─user.slice
│ └─user-1000.slice
│   └─session-1.scope
│     ├─11459 gdm-session-worker [pam/gdm-password]
│     ├─11471 gnome-session --session gnome-classic
│     ├─11479 dbus-launch --sh-syntax --exit-with-session
│     ├─11480 /bin/dbus-daemon --fork --print-pid 4 --print-address 6 --session
│     ...
│
└─system.slice
  ├─systemd-journald.service
  │ └─422 /usr/lib/systemd/systemd-journald
  ├─bluetooth.service
  │ └─11691 /usr/sbin/bluetoothd -n
  ├─systemd-localed.service
  │ └─5328 /usr/lib/systemd/systemd-localed
  ├─colord.service
  │ └─5001 /usr/libexec/colord
  ├─sshd.service
  │ └─1191 /usr/sbin/sshd -D
  │
  ...
```

ご覧のとおり、サービスおよびスコープにはプロセスが含まれており、独自のプロセスを含まないスライスに置かれています。唯一の例外は、**-.slice** とマークされた特別な **systemd** スライスにある **PID**

1 です。また、-.slice はツリー全体のルートディレクトリーで暗黙的に識別されるため、表示されません。

「ユニットファイルの変更」、PID 1 への API 呼び出しによりランタイム時に動的に作成できます（API「オンラインドキュメント」 を参照）。スコープのユニットは、動的に作成できます。API 呼び出しで動的に作成されたユニットは一時的なもので、ランタイム時にのみ存在します。一時的なユニットは終了、無効化されたり、システムが再起動されるとすぐに自動的に解放されます。

1.3. LINUX カーネルのリソースコントローラー

cgroup サブシステムとも呼ばれるリソースコントローラーは、CPU 時間やメモリーなどの単一のリソースを表します。Linux カーネルは、systemd によって自動的にマウントされるリソースコントローラーの範囲を提供します。/proc/cgroups で現在マウントされているリソースコントローラーの一覧を検索するか、lssubsys 監視ツールを使用します。Red Hat Enterprise Linux 7 では、systemd はデフォルトで以下のコントローラーをマウントします。

Red Hat Enterprise Linux 7 で利用可能なコントローラー

- blkio - ブロックデバイスへの入出力アクセスを制限します。
- cpu - CPU スケジューラーを使用して、CPU への cgroup タスクへのアクセスを提供します。これは、同じマウントで cpuacct コントローラーとともにマウントされます。
- cpuacct - cgroup のタスクが使用する CPU リソースに関する自動レポートを作成します。これは、同じマウント上の cpu コントローラーとともにマウントされます。
- cpuset - 個別の CPU（マルチコアシステム）とメモリーノードを cgroup のタスクに割り当てます。
- devices - cgroup のタスクのデバイスへのアクセスを許可または拒否します。
- freezer - cgroup のタスクを一時停止または再開します。
- memory: cgroup のタスクでメモリー使用を設定し、それらのタスクによって使用されるメモリーリソースについての自動レポートを生成します。
-

net_cls - 特定の cgroup タスクから発信されたパケットを識別できるようにするために Linux トラフィックコントローラー （tc コマンド）を許可するクラス識別子(classid)でネットワークパケットをタグ付けします。net_cls のサブシステム net_filter (iptables)はこのタグを使用して、そのようなパケットに対するアクションを実行することもできます。net_filter は、ファイアウォール識別子(fwid)でネットワークソケットをタグ付けします。これにより、Linux ファイアウォール （iptables コマンド）が特定の cgroup タスクから発信されたパケット(skb->sk)を識別できます。

- *perf_event - perf ツールを使用して cgroups の監視を有効にします。*
- *HugeTLB: サイズの大きい仮想メモリーページの使用を許可し、これらのページに対するリソース制限を施行します。*

Linux カーネルは、systemd で設定できるリソースコントローラーのさまざまな調整可能なパラメーターを公開します。これらのパラメーターの詳細な説明は、カーネルドキュメント コントローラー固有のカーネルドキュメント ）を参照してください。

1.4. 関連情報

systemd、ユニット階層、およびカーネルリソースコントローラーでリソース制御の詳細は、以下に記載の資料を参照してください。

インストールされているドキュメント

cgroup-Related Systemd Documentation

以下の man ページには、systemd 下の統一された cgroup 階層に関する一般的な情報が含まれます。

- *systemd.resource-control(5)- では、システムユニットで共有されるリソース制御の設定オプションが説明されています。*
- *systemd.unit(5): すべてのユニット設定ファイルの共通のオプションを説明しています。*
- *systemd.slice(5): ユニットに関する一般的な情報を提供します。*

- *systemd.scope(5):* ユニットに関する一般的な情報を提供します。
- *systemd.service(5)-* ユニットに関する一般的な情報を提供します。

コントローラー固有のカーネルドキュメント

kernel-doc パッケージには、すべてのリソースコントローラーの詳細なドキュメントがあります。このパッケージは Optional サブスクリプションチャンネルに含まれます。Optional チャンネルにサブスクライブする前に、「Optional ソフトウェアに対する 対象範囲の詳細 」を参照して、Red Hat カスタマーポータルの記事「Optional および Supplementary チャンネル、-devel パッケージにアクセスする方法」に記載されている手順を行ってください。Optional チャンネルから kernel-doc をインストールするには、root で以下を入力します。

```
~]# yum install kernel-doc
```

インストール後に、以下のファイルが /usr/share/doc/kernel-doc-<kernel_version>/Documentation/cgroups/ ディレクトリーの下に表示されます。

- *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*
- *net_cls subsystem — net_cls.txt*

また、cpu サブシステムの詳細は以下のファイルを参照してください。

- *リアルタイムスケジューリング - /usr/share/doc/kernel-doc-<kernel_version>/Documentation/scheduler/sched-rt-group.txt*
- *CFS スケジューリング - /usr/share/doc/kernel-doc-<kernel_version>/Documentation/scheduler/sched-bwc.txt*

オンラインドキュメント

- *Red Hat Enterprise Linux 7 システム管理者のガイド： システム管理者ガイドでは、Red Hat Enterprise Linux 7 の導入、設定、および管理に関する関連情報を説明します。本ガイドでは、systemd の概念と、systemd によるサービス管理に関する説明が記載されています。*
- *systemd の D-Bus API - systemd との対話に使用する D-Bus API コマンドの参考資料。*

第2章 コントロールグループの使用

以下のセクションでは、コントロールグループの作成と管理に関連するタスクの概要を説明します。本ガイドでは、systemd が提供するユーティリティーに重点を置いています。これは、cgroup 管理として優先されるため、今後サポートされる予定です。以前のバージョンの Red Hat Enterprise Linux では、cgroups を作成して管理するのに libcgroup パッケージを使用していました。このパッケージは、後方互換性 警告を参照）として引き続き利用できますが、Red Hat Enterprise Linux の今後のバージョンではサポートしていません。

2.1. コントロールグループの作成

systemd の観点では、cgroup はユニットファイルで設定可能なシステムユニットにバインドされ、systemd のコマンドラインユーティリティーで管理できます。アプリケーションのタイプに応じて、リソース管理の設定は一時的なものでも永続 できます。

サービスの一時的な cgroup を作成するには、systemd-run コマンドでサービスを起動します。これにより、ランタイム時にサービスが消費するリソースに制限を設定できます。アプリケーションは、systemd への API 呼び出しを使用して、一時的な cgroups を動的に作成できます。API「オンラインドキュメント」、を参照してください。一時的なユニットは、サービスが停止するとすぐに自動的に削除されます。

永続的な cgroup をサービスに割り当てるには、そのユニット設定ファイルを編集します。この設定はシステム再起動後も保存されるので、これを使用して自動的に起動されたサービスを管理できます。このようにスコープのユニットを作成することができないことに注意してください。

2.1.1. systemd-run でのTransient Cgroups の作成

systemd-run コマンドは、一時的なサービスまたはスコープユニット を作成して起動し、そのユニットでカスタムコマンドを実行します。サービスユニットで実行されるコマンドは、バックグラウンドで非同期に開始され、systemd プロセスから呼び出されます。スコープユニットで実行されるコマンドは、systemd-run プロセスから直接開始されるため、呼び出し元の実行環境を継承します。この場合の実行は同期です。

指定した cgroup でコマンドを実行するには、root で以下を入力します。

```
~]# systemd-run --unit=name --scope --slice=slice_name command
```

- この名前は、ユニットが既知の名前を表します。--unit が指定されていないと、ユニット名が自動的に生成されます。systemctl 出力のユニットを表すため、記述的な名前を選択することが推奨されます。名前は、ユニットのランタイム時に一意である必要があります。

- *オプションの --scope パラメーターを使用して、デフォルトで作成されるサービスユニットの代わりに、一時的なスコープ ユニットを作成します。*

- *--slice オプションを使用すると、新しく作成されたサービスまたはスコープユニット を、指定したスライスのメンバーにすることができます。slice_name を、既存のスライスの名前（systemctl -t sliceの出力に表示）に置き換えるか、または一意の名前を指定して新規スライスを作成します。デフォルトでは、サービスおよびスコープは system.slice のメンバーとして作成されます。*

- *command を、サービスユニットで実行するコマンドに置き換えます。このコマンドは systemd-run 構文を非常に終了しているので、このコマンドのパラメーターに systemd-run のパラメーターに混同しないようにしてください。*

上記のオプションに加えて、systemd-run に利用できるその他のパラメーターがいくつかあります。たとえば、--description はユニットの説明を作成します。--remain-after-exit を使用すると、サービスのプロセス終了後にランタイム情報を収集できます。--machine オプションは、制限のあるコンテナーでコマンドを実行します。詳細は、systemd-run(1)man ページを参照してください。

> ***例2.1 systemd-run で新規サービスの起動***
>
> *テストと呼ばれる新しいスライスで、サービスユニットでトップユーティリティーを実行します。root で以下を入力します。*
>
> ```
> ~]# systemd-run --unit=toptest --slice=test top -b
> ```
>
> *以下のメッセージが表示され、サービスが正常に起動したことを確認します。*
>
> ```
> Running as unit toptest.service
> ```
>
> *これで、name toptest.service を使用して cgroup を systemctl コマンドで監視または変更できるようになりました。*

2.1.2. 永続 Cgroups の作成

システムの起動時にユニットを自動的に起動するように設定するには、systemctl enable コマンドを実行します（『Red Hat Enterprise Linux 7 システム管理者ガイド』の「systemd によるサービスの管理 」の章を参照）。このコマンドを実行すると、/usr/lib/systemd/system/ ディレクトリーにユ

ニットファイルが自動的に作成されます。cgroup を永続的に変更するには、そのユニットファイルに設定パラメーターを追加または変更します。詳細はを参照してください「ユニットファイルの変更」。

2.2. コントロールグループの削除

一時的な cgroups は、プロセスが完了した後すぐに自動的に解放されます。--remain-after-exit オプションを systemd-run に渡すと、ランタイム情報を収集するためにプロセスの終了後にユニットを実行し続けることができます。ユニットを正常に停止するには、以下を入力します。

```
~]# systemctl stop name.service
```

name を、停止するサービスの名前に置き換えます。ユニットのプロセスを 1 つ以上終了するには、root で以下を入力します。

```
~]# systemctl kill name.service --kill-who=PID,... --signal=signal
```

name を、ユニットの名前（httpd.service など）に置き換えます。--kill-who を使用して、終了する cgroup からどのプロセスを選択します。複数のプロセスを同時に強制終了するには、PID のコンマ区切りの一覧を指定します。signal を、指定されたプロセスに送信する POSIX シグナルのタイプに置き換えます。デフォルトは SIGTERM です。詳細は、systemd.kill man ページを参照してください。

永続的な cgroups は、ユニットが無効で、その設定ファイルを実行すると解放されます。

```
~]# systemctl disable name.service
```

name は、無効にするサービスの名前を表します。

2.3. コントロールグループの変更

systemd による永続ユニットのそれぞれには、/usr/lib/systemd/system/ ディレクトリーにユニット設定ファイルがあります。サービスユニットのパラメーターを変更するには、この設定ファイルを変更します。これは、systemctl set-property コマンドを使用して、手動またはコマンドラインインターフェースから実行できます。

2.3.1. コマンドラインインターフェースからのパラメーターの設定

systemctl set-property コマンドを使用すると、アプリケーションのランタイム時にリソース制御の設定を永続的に変更できます。これを行うには、root として以下の構文を使用します。

```
~]# systemctl set-property name parameter=value
```

name を、変更する systemd ユニットの名前に置き換え、parameter を、変更するパラメーターの名前に置き換え、value を、このパラメーターに割り当てる新しい値に置き換えます。

「ユニットファイルの変更」。systemctl set-property では、複数のプロパティーを一度に変更できます。

変更は即座に適用され、ユニットファイルに書き込まれ、再起動後に保持されるようにします。この動作を変更するには、設定を一時的にする --runtime オプションを指定します。

```
~]# systemctl set-property --runtime name property=value
```

例2.2 systemctl set-property の使用

コマンドラインから httpd.service の CPU およびメモリーの使用状況を制限するには、以下を入力します。

```
~]# systemctl set-property httpd.service CPUShares=600 MemoryLimit=500M
```

これを一時的に変更するには、--runtime オプションを追加します。

```
~]# systemctl set-property --runtime httpd.service CPUShares=600 MemoryLimit=500M
```

2.3.2. ユニットファイルの変更

systemd サービスユニットファイルは、リソース管理に役立つ多くの高レベルの設定パラメーターを提供します。これらのパラメーターは Linux cgroup コントローラーと通信し、カーネルで有効にする必要があります。これらのパラメーターを使用して、CPU、メモリー消費、ブロック IO、さらに粒度の細かいユニットプロパティーを管理できます。

CPU の管理

cpu コントローラーは、カーネル内でデフォルトで有効になっており、すべてのシステムサービスは、含まれるプロセスの数にかかわらず、同じ CPU 時間を受け取ります。このデフォルトの動作は、/etc/systemd/system.conf 設定ファイルの DefaultControllers パラメーターで変更できます。CPU 割り当てを管理するには、ユニット設定ファイルの [Service] セクションで以下のディレクティブを使用します。

CPUShares=value

value を、CPU 共有数に置き換えます。デフォルト値は 1024 です。数値を大きくすると、CPU 時間を単位に割り当てます。CPUShares パラメーターの値を設定すると、ユニットファイルで CPUAccounting が自動的に有効になります。したがって、ユーザーは systemd-cgtop コマンドでプロセッサーの使用を監視できます。

CPUShares パラメーターは cpu.shares *コントロールグループパラメーターを制御します。他の CPU コントローラー固有のカーネルドキュメント コントローラーの説明を参照してください。*

> *例2.3 ユニットの CPU 使用率の制限*
>
> *デフォルトの 1024 ではなく Apache サービス 1500 CPU 共有を割り当てるには、以下の内容で新たな /etc/systemd/system/httpd.service.d/cpu.conf 設定ファイルを作成します。*
>
> ```
> [Service]
> CPUShares=1500
> ```
>
> *変更を適用するには、systemd の設定を再読み込みし、変更したサービスファイルを考慮して Apache を再起動します。*
>
> ```
> ~]# systemctl daemon-reload
> ~]# systemctl restart httpd.service
> ```

CPUQuota=value

value を、指定された CPU 時間クォータを実行したプロセスに割り当てる CPU 時間クォータの値に置き換えます。パーセンテージで表される CPUQuota パラメーターの値は、1 つの CPU 時間で利用可能な CPU 時間の合計と、そのユニットが最大で取得する CPU 時間を指定します。

100% より大きい値は、複数の CPU が使用されることを示します。CPUQuota は、統一されたコントロールグループ階層の cpu.max *属性と、レガシー* cpu.cfs_quota_us *属性を制御します。CPUQuota パラメーターの値を設定すると、ユニットファイルで CPUAccounting が自動的に有効になります。したがって、ユーザーは systemd-cgtop コマンドでプロセッサーの使用を監視できます。*

> *例2.4 CPUQuota の使用*
>
> *CPUQuota を 20% に設定すると、実行されたプロセスが 1 つの CPU で 20% を超える CPU 時間を取得できなくなります。*

Apache サービス CPU クォータが 20% のものを割り当てるには、以下の内容を /etc/systemd/system/httpd.service.d/cpu.conf 設定ファイルに追加します。

```
[Service]
CPUQuota=20%
```

変更を適用するには、systemd の設定を再読み込みし、変更したサービスファイルを考慮して Apache を再起動します。

```
~]# systemctl daemon-reload
~]# systemctl restart httpd.service
```

メモリーの管理

ユニットのメモリー消費を制限するには、ユニット設定ファイルの [Service] セクションで以下のディレクティブを使用します。

MemoryLimit=value

value を、cgroup で実行されるプロセスの最大メモリー使用量の制限に置き換えます。測定単位のキロバイト、メガバイト 、ギガバイト、またはテラバイトを指定するには、*接尾辞* K、M、G、または T を使用します。また、ユニット用に MemoryAccounting パラメーターを有効にする必要があります。

TheMemoryLimit パラメーターは、memory.limit_in_bytes コントロールグループパラメーターを制御します。詳細は、メモリーコントローラーの説明を参照してください コントローラー固有のカーネルドキュメント 。

例2.5 ユニットのメモリー消費の制限

1GB のメモリー上限を Apache サービスに割り当てるには、/etc/systemd/system/httpd.service.d/cpu.conf ユニットファイルのMemoryLimit 設定を変更します。

```
[Service]
MemoryLimit=1G
```

変更を適用するには、systemd の設定を再読み込みし、変更したサービスファイルを考慮して Apache を再起動します。

```
~]# systemctl daemon-reload
~]# systemctl restart httpd.service
```

ブロック IO の管理

Block IO を管理するには、ユニット設定ファイルの [Service] セクションで以下のディレクティブを使用します。以下のディレクティブは、BlockIOAccounting パラメーターが有効になっていることを前提としています。

BlockIOWeight=value

実行したプロセスの新しいブロック IO ウェイトに値を置き換えます。10 から 1000 までの値を 1 つ選択します。デフォルト値は 1000 です。

BlockIODeviceWeight=device_name value

value を、device_name で指定されたデバイスのブロック IO ウェイトに置き換えます。device_name は、名前か、デバイスへのパスに置き換えます。BlockIOWeight と同様に、単一の重みの値を 10 から 1000 に設定することができます。

BlockIOReadBandwidth=device_name value

このディレクティブでは、ユニット用の特定の帯域幅を制限できます。device_name を、デバイスの名前またはブロックデバイスノードへのパスに置き換え、value は帯域幅レートを表します。測定単位を指定するには、接尾辞 K、M、G 、または T を使用します。サフィックスのない値は、1 秒ごとにバイトとして解釈されます。

BlockIOWriteBandwidth=device_name value

指定のデバイスに対する書き込み帯域幅を制限します。BlockIOReadBandwidth と同じ引数を受け入れます。

上記のディレクティブはそれぞれ、対応する cgroup パラメーターを制御します。他の CPU 関連の制御パラメーターについては、blkio コントローラーの説明を参照してください コントローラー固有のカーネルドキュメント 。

注記

現在、blkio リソースコントローラーはバッファーされた書き込み操作をサポートしません。これは主に直接 I/O を対象とするため、バッファーされた書き込みを使用するサービスは、BlockIOWriteBandwidth で設定された制限を無視します。一方、バッファーされた読み取り操作がサポートされ、BlockIOReadBandwidth 制限は直接読み取りとバッファーされた読み取りの両方に正常に適用されます。

例2.6 ユニットのブロック IO の制限

/home/jdoe/ ディレクトリーにアクセスする Apache サービスのブロック IO ウェイトを下げるには、以下のテキストを /etc/systemd/system/httpd.service.d/cpu.conf ユニットファイルに追加します。

```
[Service]
BlockIODeviceWeight=/home/jdoe 750
```

Apache 読み取りの最大帯域幅を、1 秒あたり /var/log/ ディレクトリーから 5MB に設定するには、以下の構文を使用します。

```
[Service]
BlockIOReadBandwidth=/var/log 5M
```

変更を適用するには、systemd の設定を再読み込みし、変更したサービスファイルを考慮して Apache を再起動します。

```
~]# systemctl daemon-reload
~]# systemctl restart httpd.service
```

その他のシステムリソースの管理

リソース管理を容易にするために、ユニットファイルで使用できる他のディレクティブがいくつかあります。

DeviceAllow=device_name options

このオプションは、特定のデバイスノードへのアクセスを制御します。device_name は、デバイスノードまたはデバイスグループ名へのパスを /proc/devices で指定します。options を、r、w、および m の組み合わせで、ユニットがデバイスノードの読み取り、書き込み、または作成を可能にします。

DevicePolicy=value

ここで、value は、strict （デバイスAllowで指定されたアクセスの種類のみ）、終了（/dev/null、/dev/zero、/dev/full、/dev/random、および /dev/urandom）などの標準の擬似デバイスへのアクセスを許可します（明示的なデバイス Allow がない場合、デフォルトの動作）。

Slice=slice_name

slice_name を、ユニットを配置するスライスの名前に置き換えます。デフォルトは system.slice です。スコープのユニットは、親スライスと関連付けられているため、このように配置することはできません。

ExecStartPost=command

現在、systemd は cgroup 機能のサブセットのみをサポートします。ただし、回避策として、ExecStartPost= オプションで memory.memsw.limit_in_bytes パラメーターを設定して、サービスにスワップの使用を防ぐために パラメーターを設定します。ExecStartPost= の詳細は、systemd.service(5) man ページを参照してください。

例2.7 Cgroup オプションの設定

memory.memsw.limit_in_bytes 設定をユニットのMemoryLimit= と同じ値に変更し、特定のサンプルサービスにスワップの使用を防ぐことを目的としています。

```
ExecStartPost=/bin/bash -c "echo 1G >
/sys/fs/cgroup/memory/system.slice/example.service/memory.memsw.limit_in_bytes"
```

変更を適用するには、systemd 設定を再読み込みし、変更した設定を考慮してサービスを再起動します。

```
~]# systemctl daemon-reload
~]# systemctl restart example.service
```

2.4. コントロールグループに関する情報の取得

systemctl コマンドを使用してシステムユニットの一覧を表示し、それらのステータスを表示します。また、systemd-cgls コマンドは、コントロールグループの階層と systemd-cgtop がリアルタイムでリソース消費を監視するように提供されています。

2.4.1. ユニットの一覧表示

以下のコマンドを使用して、システム上のアクティブなユニットの一覧を表示します。

```
~]# systemctl list-units
```

list-units オプションは、デフォルトで実行されます。これは、このオプションを省略して実行するときに同じ出力を受け取ることを意味します。

```
~]$systemctl
UNIT                    LOAD   ACTIVE SUB     DESCRIPTION
abrt-ccpp.service       loaded active exited  Install ABRT coredump hook
abrt-oops.service       loaded active running ABRT kernel log watcher
abrt-vmcore.service     loaded active exited  Harvest vmcores for ABRT
abrt-xorg.service       loaded active running ABRT Xorg log watcher
...
```

上記の出力には、5 つのコラムが含まれます。

- **UNIT** - *cgroup* ツリー内のユニットの位置も反映するユニットの名前。「*systemd* のユニットタイプ」、*3* つのユニットタイプは、スライス、スコープ、およびサービスなどのリソース制御 に関連します。*systemd* のユニットタイプの完全なリストは、『Red Hat Enterprise Linux 7 システム管理者ガイド』の「systemd によるサービス管理」の章を参照してください。

- **LOAD** - ユニット設定ファイルが正しく読み込まれたかどうかを示します。ユニットファイルの読み込みに失敗した場合、フィールドの状態が loaded ではなく error になります。ユニットの読み込みの状態は スタブ、マージ、マスクされ ます。

- **ACTIVE** - ユニットのアクティベーションの状態の概要（*SUB* を一般化）

- **SUB** - ユニットのアクティベーションの状態（詳細レベル）。許容値の範囲は、ユニットタイプによって異なります。

- **DESCRIPTION** - ユニットのコンテンツおよび機能の説明。

デフォルトでは、*systemctl* ではアクティブなユニットのみが一覧表示されます（アクティベーションの状態は *ACTIVE* フィールドにあります）。*--all* オプションを使用して、アクティブではないユニットも表示します。出力一覧の情報量を制限するには、サービスや スライスなどのユニットタイプのコ

ンマ区切りの一覧を必要とする *--type (-t)* パラメーターを使用します。あるいは、読み込まれ 、マスクされたユニット負荷の状態を使用します。

例2.8 systemctl list-units の使用

システムで使用される全スライスの一覧を表示するには、以下を入力します。

```
~]$ systemctl -t slice
```

アクティブなマスクされたサービスをすべて一覧表示するには、以下を入力します。

```
~]$ systemctl -t service,masked
```

システムにインストールされているユニットファイルとその状態を一覧表示するには、次のコマンドを実行します。

```
~]$ systemctl list-unit-files
```

2.4.2. コントロールグループ階層の表示

前述の一覧コマンドは、ユニットレベルを超えるものではなく、cgroups で実行している実際のプロセスを表示します。また、systemctl の出力には、ユニットの階層も表示されません。cgroups に応じて実行中のプロセスをグループ化する systemd-cgls コマンドを使用するとも実行できます。システムの cgroup 階層全体を表示するには、以下を入力します。

```
~]$ systemd-cgls
```

systemd-cgls がパラメーターなしで発行されると、cgroup 階層全体を返します。cgroup ツリーの最も高いレベルはスライスによって形成され、以下のようになります。

```
├─system
│ ├─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 20
│ ...
│
├─user
│ ├─user-1000
│ │ └─ ...
│ ├─user-2000
│ │ └─ ...
│ ...
│
```

```
└─machine
  ├─machine-1000
  │ └─ ...
  ...
```

マシンスライスは、仮想マシンまたはコンテナーを実行している場合にのみ存在することに注意してください。cgroup ツリーの詳細は、「systemd のユニットタイプ」。

systemd-cgls の出力を減らし、階層の指定部分を表示するには、以下を実行します。

```
~]$ systemd-cgls name
```

name を、検査するリソースコントローラーの名前に置き換えます。

代わりに、systemctl status コマンドを使用して、システムユニットに関する詳細情報を表示します。cgroup サブツリーは、このコマンドの出力の一部です。

```
~]$ systemctl name
```

systemctl ステータスの詳細は、『Red Hat Enterprise Linux 7 システム管理者ガイド』の「systemd によるサービス管理 」の章を参照してください。

例2.9 コントロールグループ階層の表示

メモリーリソースコントローラーの cgroup ツリーを表示するには、以下を実行します。

```
~]$ systemd-cgls memory
memory:
├─   1 /usr/lib/systemd/systemd --switched-root --system --deserialize 23
├─ 475 /usr/lib/systemd/systemd-journald
...
```

上記のコマンドの出力には、選択したコントローラーと対話するサービスが一覧表示されます。特定のサービス、スライス、またはスコープユニットの cgroup ツリーの一部を表示する方法が異なります。

```
~]# systemctl status httpd.service
httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled)
   Active: active (running) since Sun 2014-03-23 08:01:14 MDT; 33min ago
  Process: 3385 ExecReload=/usr/sbin/httpd $OPTIONS -k graceful (code=exited,
```

```
status=0/SUCCESS)
 Main PID: 1205 (httpd)
   Status: "Total requests: 0; Current requests/sec: 0; Current traffic:   0 B/sec"
   CGroup: /system.slice/httpd.service
           ├─1205 /usr/sbin/httpd -DFOREGROUND
           ├─3387 /usr/sbin/httpd -DFOREGROUND
           ├─3388 /usr/sbin/httpd -DFOREGROUND
           ├─3389 /usr/sbin/httpd -DFOREGROUND
           ├─3390 /usr/sbin/httpd -DFOREGROUND
           └─3391 /usr/sbin/httpd -DFOREGROUND

...
```

systemd は、上記のツールに加えて、Linux コンテナーの監視専用となる machinectl コマンドも提供します。

2.4.3. リソースコントローラーの表示

前述した systemctl コマンドは、高レベルのユニット階層の監視を有効にしますが、Linux カーネルのどのリソースコントローラーが実際にどのプロセスで使用されるかを表示しません。この情報は、専用プロセスファイルに保存され、これを表示するには root で以下を入力します。

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

PID は、検証するプロセスの ID を表します。デフォルトでは、デフォルトのコントローラーがすべて自動的にマウントされるため、systemd が起動するすべてのユニットで同じリストが使用されます。以下の例を参照してください。

```
~]# cat proc/27/cgroup
10:hugetlb:/
9:perf_event:/
8:blkio:/
7:net_cls:/
6:freezer:/
5:devices:/
4:memory:/
3:cpuacct,cpu:/
2:cpuset:/
1:name=systemd:/
```

このファイルを調べると、systemd ユニットファイルの仕様で定義されているように、適切な cgroups にプロセスが置かれているかどうかを判断できます。

2.4.4. リソース消費の監視

systemd-cgls コマンドは、cgroup 階層の静的スナップショットを提供します。現在実行中の cgroups の動的アカウント（CPU、メモリー、および IO）を順番に表示するには、以下を使用します。

```
~]# systemd-cgtop
```

systemd-cgtop の動作、提供された統計、制御オプションは、top ユーティリティーにより表示されます。詳細は、systemd-cgtop(1)man ページを参照してください。

2.5. 関連情報

systemd と関連ツールを使用して Red Hat Enterprise Linux のシステムリソースを管理する方法は、以下の資料を参照してください。

インストールされているドキュメント

Cgroup-Related Systemd ツールの man ページ

- systemd-run(1)- man ページでは、systemd-run ユーティリティーのコマンドラインオプションがすべて表示されます。
- systemctl(1)- 利用可能なオプションとコマンドを一覧表示する systemctl ユーティリティーの man ページです。
- systemd-cgls(1)- この man ページでは、systemd-cgls ユーティリティーのコマンドラインオプションがすべて表示されます。
- systemd-cgtop(1)- man ページには、systemd-cgtop ユーティリティーの全コマンドラインオプションの一覧が含まれます。
- machinectl(1)- この man ページでは、machinectl ユーティリティーのコマンドラインオプションがすべて表示されます。
- systemd.kill(5)- この man ページでは、システムユニットの kill 設定オプションの概要が記載されています。

コントローラー固有のカーネルドキュメント

kernel-doc パッケージでは、すべてのリソースコントローラーの詳細なドキュメントがあります。このパッケージは Optional サブスクリプションチャンネルに含まれます。Optional チャンネルにサブスクライブする前に、「対象範囲の詳細チャネル 」を参照してから、Red Hat カスタマーポータルの「Optional および Supplementary チャンネル、-devel パッケージにアクセスする方法」に記載されている手順を行ってください。Optional チャンネルから kernel-doc をインストールするには、root で以下を入力します。

```
~]# yum install kernel-doc
```

インストール後に、以下のファイルが /usr/share/doc/kernel-doc-<kernel_version>/Documentation/cgroups/ ディレクトリーの下に表示されます。

- *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*
- *net_cls subsystem — net_cls.txt*

また、cpu サブシステムの詳細は以下のファイルを参照してください。

- *リアルタイムスケジューリング - /usr/share/doc/kernel-doc-<kernel_version>/Documentation/scheduler/sched-rt-group.txt*
- *CFS スケジューリング - /usr/share/doc/kernel-*

doc-<kernel_version>/Documentation/scheduler/sched-bwc.txt

オンラインドキュメント

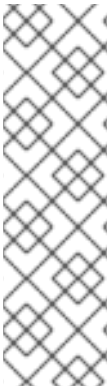
- *Red Hat Enterprise Linux 7 システム管理者ガイド*: システム管理者ガイドでは、*Red Hat Enterprise Linux 7* の導入、設定、および管理に関する関連情報を説明します。本書は、システムに関する基本的な理解があるシステム管理者を対象としています。

- *systemd の D-Bus API - systemd* にアクセスするための *D-Bus API* コマンドへの参照。

第3章 LIBCGROUP ツールの使用

Red Hat Enterprise Linux の以前のバージョンの cgroup 管理用のメインツールである libcgroup パッケージは非推奨になりました。競合を回避するには、systemd Red Hat Enterprise Linux 7 で利用可能なコントローラー libcgroup ツールを使用しないでください。これにより、libcgroup ツールを適用する限定スペースが残されるので、現在 systemd でサポートされていないコントローラーを管理する必要がある場合にのみ使用してください（例： net_prio ）。

以下のセクションでは、階層のデフォルトシステムと競合しずに、関連するシナリオで libcgroup ツールを使用する方法を説明します。



注記

libcgroup ツールを使用するには、まず libcgroup パッケージおよび libcgroup-tools パッケージがシステムにインストールされていることを確認します。これらのツールをインストールするには、root で以下を実行します。

```
~]# yum install libcgroup
~]# yum install libcgroup-tools
```

注記

net_prio コントローラーは、他のコントローラーなどのカーネルでコンパイルされません。マウントを試みる前にロードする必要があるモジュールです。このモジュールを読み込むには、root で以下を入力します。

```
~]# modprobe netprio_cgroup
```

3.1. 階層のマウント

自動的にマウントされていないカーネルリソースコントローラーを使用するには、このコントローラーを含む階層を作成する必要があります。/etc/cgconfig.conf 設定ファイルの mount セクションを編集して、階層を追加または切断します。このメソッドは、コントローラーの割り当てを永続化します。これは、システムの再起動後に設定が保持されます。別の方法として、mount コマンドを使用して、現行セッションにのみ一時的なマウントを作成します。

cgconfig サービスの使用

libcgroup-tools パッケージでインストールされる cgconfig サービスにより、追加のリソースコントローラーの階層をマウントする方法が提供されます。デフォルトでは、このサービスは自動的に開始されません。cgconfig を開始すると、/etc/cgconfig.conf 設定ファイルから設定を適用します。そのた

め、設定はセッションからセッションに再作成され、永続化されます。cgconfig を停止すると、マウントしたすべての階層をアンマウントすることに注意してください。

libcgroup パッケージでインストールされるデフォルトの /etc/cgconfig.conf ファイルには構成設定が含まれておらず、systemd がメインのリソースコントローラーを自動的にマウントする情報のみが含まれます。

3 つのタイプのエントリーは、/etc/cgconfig.conf - mount、group、および template で作成できます。マウントエントリーは、階層を仮想ファイルシステムとして作成およびマウントするために使用され、コントローラーをそれらの階層にアタッチするために使用されます。Red Hat Enterprise Linux 7 では、デフォルトの階層は /sys/fs/cgroup/ ディレクトリーに自動的にマウントされます。したがって、cgconfig はデフォルト以外のコントローラーのみを接続するために使用されます。マウントエントリーは、以下の構文を使用して定義されます。

```
mount {
    controller_name = /sys/fs/cgroup/controller_name;
    …
}
```

controller_name を、階層にマウントするカーネルリソースコントローラーの名前に置き換えます。例3.1「マウントエントリーの作成」 を参照してください。

例3.1 マウントエントリーの作成

net_prio コントローラーをデフォルトの cgroup ツリーに割り当てるには、以下のテキストを /etc/cgconfig.conf 設定ファイルに追加します。

```
mount {
    net_prio = /sys/fs/cgroup/net_prio;
}
```

次に、cgconfig サービスを再起動して設定を適用します。

```
~]# systemctl restart cgconfig.service
```

/etc/cgconfig.conf のグループエントリーは、リソースコントローラーのパラメーターを設定するために使用できます。「Cgroup パラメーターの設定」。

/etc/cgconfig.conf 内のテンプレートエントリーを使用して、すべてのプロセスに適用されるグループ定義を作成できます。

mount コマンドの使用

mount コマンドを使用して、階層を一時的にマウントします。これを行うには、最初に systemd がメインのリソースコントローラーをマウントする /sys/fs/cgroup/ ディレクトリーにマウントポイントを作成します。root で以下を入力します。

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

name を新しいマウント宛先の名前に置き換え、通常コントローラーの名前が使用されます。次に mount コマンドを実行して階層をマウントし、1 つ以上のサブシステムを同時にアタッチします。root で以下を入力します。

```
~]# mount -t cgroup -o controller_name none /sys/fs/cgroup/controller_name
```

controller_name をコントローラーの名前に置き換えて、マウントするデバイスと宛先フォルダーの両方を指定します。-t cgroup パラメーターはマウントのタイプを指定します。

例3.2 mount コマンドでコントローラーをアタッチ

mount コマンドを使用して net_prio コントローラーの階層をマウントするには、最初にマウントポイントを作成します。

```
~]# mkdir /sys/fs/cgroup/net_prio
```

次に、前の手順で作成した宛先に対して net_prio をマウントします。

```
~]# mount -t cgroup -o net_prio none /sys/fs/cgroup/net_prio
```

lssubsys 「コントローラーの一覧表示」。

```
~]# lssubsys -am
cpuset /sys/fs/cgroup/cpuset
cpu,cpuacct /sys/fs/cgroup/cpu,cpuacct
memory /sys/fs/cgroup/memory
devices /sys/fs/cgroup/devices
freezer /sys/fs/cgroup/freezer
net_cls /sys/fs/cgroup/net_cls
blkio /sys/fs/cgroup/blkio
perf_event /sys/fs/cgroup/perf_event
hugetlb /sys/fs/cgroup/hugetlb
net_prio /sys/fs/cgroup/net_prio
```

3.2. 階層のアンマウント

/etc/cgconfig.conf 設定ファイルを編集して階層をマウントした場合は、この設定ファイルの mount セクションから設定ディレクティブを削除するだけで、単にその階層をアンマウントできます。次に、サービスを再起動して新しい設定を適用します。

同様に、root で以下のコマンドを実行すると、階層をアンマウントできます。

```
~]# umount /sys/fs/cgroup/controller_name
```

controller_name を、デタッチするリソースコントローラーが含まれる階層の名前に置き換えます。

警告

umount を使用して、手動でマウントした階層のみを削除してください。デフォルトのコントローラー Red Hat Enterprise Linux 7 で利用可能なコントローラー）を含む階層をデタッチすると、システムの再起動が必要となります。

3.3. コントロールグループの作成

cgcreate コマンドを使用して、独自に作成した階層に一時的な cgroups を作成します。cgcreate の構文は以下のとおりです。

```
cgcreate -t uid:gid -a uid:gid -g controllers:path
```

各オプションについての説明は以下のとおりです。

- -t（オプション）: ユーザー（ユーザー ID、uid 別）およびグループ（グループ ID、gid を使用）を使用して、この cgroup のタスク擬似ファイルを所有する。このユーザーは、タスクを cgroup に追加できます。

注記

cgroup からプロセスを削除する唯一の方法は、これを別の cgroup に移動することです。プロセスを移動できるようにするには、ユーザーは宛先 cgroup への書き込みアクセス権が必要です。ソース cgroup への書き込みアクセスは必要ありません。

- *-A（オプション）: ユーザー（ユーザー ID、uid 別）およびグループ（グループ ID、gid を使用）を指定して、この cgroup のタスク以外のすべての擬似ファイルを所有する。このユーザーは、この cgroup のタスクのシステムリソースへのアクセスを変更できます。*

- *-g: 階層に関連付けられたコントローラーのコンマ区切りリストとして、cgroup を作成する階層を指定します。コントローラーのリストの後に、階層と相対的な子グループへのパスが続きます。パスには階層のマウントポイントを含めないでください。*

同じ階層内の cgroups はすべて同じコントローラーであるため、子グループには親と同じコントローラーがあります。

別の方法として、cgroup の子を直接作成できます。これには、mkdir コマンドを使用します。

```
~]# mkdir /sys/fs/cgroup/controller/name/child_name
```

以下に例を示します。

```
~]# mkdir /sys/fs/cgroup/net_prio/lab1/group1
```

3.4. コントロールグループの削除

cgcreate に類する構文がある cgdelete コマンドを使用して、cgroups を削除します。root で以下のコマンドを実行します。

```
~]# cgdelete controllers:path
```

各オプションについての説明は以下のとおりです。

- *コントローラーは、コントローラーのコンマ区切りリストです。*

- path は、階層のルートに対する相対的な cgroup へのパスです。

以下に例を示します。

```
~]# cgdelete net_prio:/test-subgroup
```

cgdelete は、-r オプションが指定されている場合にすべてのサブグループを再帰的に削除することもできます。

cgroup を削除すると、そのすべてのプロセスが親グループに移動します。

3.5. CGROUP パラメーターの設定

/etc/cgconfig.conf 設定ファイルを編集するか、cg set コマンドを使用して、コントロールグループのパラメーターを変更します。/etc/cgconfig.conf に加えた変更は再起動後に保持されますが、cgset は現行セッションに対してのみ cgroup パラメーターを変更します。

Modifying /etc/cgconfig.conf

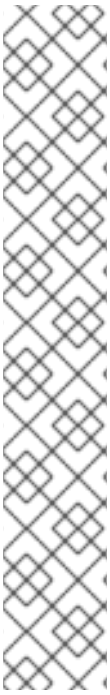
/etc/cgconfig.conf の Groups セクションで、コントローラーパラメーターを設定できます。グループエントリーは、以下の構文を使用して定義されます。

```
group name {
[permissions]
    controller {
        param_name = param_value;
        ...
    }
    ...
}
```

name を、cgroup の名前に、controller は変更するコントローラーの名前を表します。systemd が自動的にマウントされるデフォルトのコントローラーではなく、独自にマウントしたコントローラーのみを変更する必要があります。param_name および param_value を、変更するコントローラーパラメーターとその新しい値に置き換えます。permissions のセクションはオプションであることに注意してください。グループエントリーのパーミッションを定義するには、以下の構文を使用します。

```
perm {
    task {
        uid = task_user;
        gid = task_group;
```

```
    }
    admin {
      uid = admin_name;
      gid = admin_group;
    }
}
```



注記

変更を反映するために、/etc/cgconfig.conf の変更について cgconfig サービスを再起動します。このサービスを再起動すると、設定ファイルで指定された階層が再ビルドされますが、マウントされた階層はすべて影響を受けません。ただし、systemctl restart コマンドを実行すると、サービスを再起動することができます。まず cgconfig サービスを停止することを推奨します。

```
~]# systemctl stop cgconfig
```

次に、設定ファイルを開き、編集します。変更を保存したら、以下のコマンドを使用して cgconfig を再び起動できます。

```
~]# systemctl start cgconfig
```

cgset コマンドの使用

関連する cgroup を変更するパーミッションを持つユーザーアカウントから cgset コマンドを実行して、コントローラーパラメーターを設定します。これは、手動でマウントしたコントローラーにのみ使用します。

cgset の構文は以下のとおりです。

```
cgset -r parameter=value path_to_cgroup
```

各オプションについての説明は以下のとおりです。

- *parameter は設定するパラメーターで、指定の cgroup のディレクトリー内のファイルに対応します。*
- *value は、パラメーターの値です。*

- *path_to_cgroup* は、階層のルートに対する相対的な cgroup へのパスです。

cgset に設定できる値は、特定の階層で高い値に依存する可能性があります。たとえば、*group1* がシステムで *CPU 0* のみを使用するように制限されている場合は、*group1/subgroup1* を *CPU 0* および *1* を使用するように設定したり、*CPU 1* のみを使用するように設定することはできません。

cgset を使用して、ある *cgroup* のパラメーターを別の既存の *cgroup* にコピーすることもできます。*cgset* でパラメーターをコピーする構文は以下のとおりです。

```
cgset --copy-from path_to_source_cgroup path_to_target_cgroup
```

各オプションについての説明は以下のとおりです。

- *path_to_source_cgroup* は、パラメーターを階層のルートグループと相対的にコピーされる *cgroup* へのパスです。

- *path_to_target_cgroup* は、階層のルートグループの相対的な宛先 *cgroup* へのパスです。

3.6. コントロールグループへのプロセスの移動

cgclassify コマンドを実行して、プロセスを *cgroup* に移動します。

```
~]# cgclassify -g controllers:path_to_cgroup pidlist
```

各オプションについての説明は以下のとおりです。

- コントローラーは、リソースコントローラーのカンマ区切りリストまたは */* で、利用可能なすべてのサブシステムに関連する階層でプロセスを起動する */* です。同じ名前の *cgroups* が複数ある場合、*-g* オプションはそれらの各グループ内のプロセスを移動します。

- *path_to_cgroup* は、階層内の *cgroup* へのパスです。

- *pidlist は、プロセス識別子 (PID)のスペース区切りリストです。*

-g オプションが指定されていない場合、cgclassify は自動的に /etc/cgrules.conf を検索し、最初に適用可能な設定行を使用します。この行に応じて、cgclassify が、プロセスを移動させる階層と cgroups を決定します。移行を成功させるには、宛先の階層が必要です。/etc/cgrules.conf で指定されるサブシステムは、/etc/cgconfig.conf の対応する階層に対して適切に設定される必要があります。

pid の前に --sticky オプションを追加して、同じ cgroup に子プロセスを保持することもできます。このオプションを設定しておらず、cgred サービスが実行されている場合には、/etc/cgrules.conf にある設定に基づいて、子プロセスが cgroups に割り当てられます。ただし、プロセス自体は起動した cgroup に残ります。

/etc/cgrules.conf ファイルに設定されているパラメーターに従って、cgred サービス（cgrulesengd サービスの起動）を使用して、タスクを cgroups に移動することもできます。cgred のみを使用して、手動で割り当てられたコントローラーを管理します。/etc/cgrules.conf ファイルのエントリーは、以下の 2 つの形式のいずれかになります。

- *user subsystems control_group;*
- *user:command subsystems control_group.*

以下に例を示します。

```
maria   net_prio  /usergroup/staff
```

このエントリーは、/usergroup/staff cgroup に指定されたパラメーターに従って devices サブシステムにアクセスするように指定します。特定のコマンドと特定の cgroups を関連付けるには、以下のように command パラメーターを追加します。

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

このエントリーは、namedmaria が ftp コマンドを使用する場合に、このプロセスは、devices サブシステムが含まれる階層の /usergroup/staff/ftp cgroup に自動的に移動します。ただし、このデーモンは、適切な条件が満たされた後にのみ、プロセスを cgroup に移動します。したがって、ftp プロセスは、間違ったグループで短時間実行することができます。さらに、プロセスが誤ったグループに子を迅速に生成した場合に、これらの子が移動されない可能性があります。

/etc/cgrules.conf ファイルのエントリーには、以下の追加表記法を含めることができます。

- @ - user のプレフィックスが付けられている場合、個別ユーザーではなくグループを指定します。たとえば、@admins は admins グループのすべてのユーザーです。
- * — represents "all".たとえば、subsystem フィールドの* はすべてのサブシステムを表します。
- %: 上記の行の項目と同じ項目を表します。以下に例を示します。

```
@adminstaff net_prio   /admingroup
@labstaff %   %
```

3.7. コントロールグループでのプロセスの開始

cgexec コマンドを実行して、手動で作成した cgroup でプロセスを起動します。cgexec の構文は以下のとおりです。

```
cgexec -g controllers:path_to_cgroup command arguments
```

各オプションについての説明は以下のとおりです。

- コントローラーは、コントローラーのコンマ区切りリストまたは /* で、利用可能なすべてのサブシステムに関連する階層でプロセスを起動します。に記載されている cgset 「Cgroup パラメーターの設定」、同じ名前の cgroups がある場合には、-g オプションは、これらのグループの各プロセスにプロセスを作成することに注意してください。
- path_to_cgroup は、階層との相対的な cgroup へのパスです。
- command は、cgroup で実行されるコマンドです。
- arguments は、コマンドの引数です。

コマンドの前に --sticky オプションを追加して、同じ cgroup に子プロセスを保持することもできま

す。このオプションを設定しておらず、cgred サービスが実行されている場合には、/etc/cgrules.conf にある設定に基づいて、子プロセスが cgroups に割り当てられます。ただし、プロセス自体は起動した cgroup に残ります。

3.8. コントロールグループに関する情報の取得

libcgroup-tools パッケージには、コントローラー、コントロールグループおよびパラメーターに関する情報を取得する複数のユーティリティーが含まれています。

コントローラーの一覧表示

カーネルで利用可能なコントローラーと、それらのコントローラーがどのように一緒にマウントされるかについての情報を階層に送るには、以下を実行します。

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

または、特定のサブシステムのマウントポイントを見つけるには、以下のコマンドを実行します。

```
~]$ lssubsys -m controllers
```

このコントローラーは、対象のサブシステムのリストを表します。lssubsys -m コマンドは、各階層の最上位のマウントポイントのみを返すことに注意してください。

コントロールグループの検索

システムの cgroups を一覧表示するには、root で以下のコマンドを実行します。

```
~]# lscgroup
```

出力を特定の階層に制限するには、コントローラーとパスをコントローラー:パスで指定します。以下に例を示します。

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

上記のコマンドは、階層内の adminusers cgroup のサブグループのみを一覧表示し、cpuset コントローラーがアタッチされている。

コントロールグループのパラメーターの表示

特定の cgroups のパラメーターを表示するには、次のコマンドを実行します。

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

ここでの parameter は、コントローラーの値を含む擬似ファイルであり、list_of_cgroups はスペースが付いた cgroups で区切られたリストです。

実際のパラメーターの名前が不明な場合は、以下のようなコマンドを実行します。

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

3.9. 関連情報

cgroup コマンドの definitive ドキュメントは、libcgroup パッケージで提供される man ページにあります。

インストールされているドキュメント

libcgroup 関連の man ページ

- cgclassify(1)- cgclassify コマンドは、実行中のタスクを 1 つ以上の cgroups に移動するために使用されます。
- cgclear(1)- cgclear コマンドを使用して、階層内のすべての cgroup を削除します。
- cgconfig.conf(5)- cgroups は cgconfig.conf ファイルで定義されます。
- cgconfigparser(8): cgconfigparser コマンドは cgconfig.conf ファイルを解析し、階層をマウントします。
- cgcreate(1)- cgcreate コマンドは、階層に新しい cgroups を作成します。
- cgdelete(1)- cgdelete コマンドは、指定の cgroups を削除します。

- *cgexec(1)- cgexec コマンドは、指定の cgroups でタスクを実行します。*
- *cgget(1)- cgget コマンドは cgroup パラメーターを表示します。*
- *cgsnapshot(1)- cgsnapshot コマンドは、既存のサブシステムから設定ファイルを生成します。*
- *cgred.conf(5): cgred.conf は、cgred サービスの設定ファイルです。*
- *cgrules.conf(5): cgrules.conf には、タスクが特定の cgroups に属するタイミングを決定するために使用するルールが含まれます。*
- *cgrulesengd(8): cgrulesengd サービスは、タスクを cgroups に配信します。*
- *cgset(1)- cgset コマンドは cgroup のパラメーターを設定します。*
- *lscgroup(1)- lscgroup コマンドは、階層の cgroups を一覧表示します。*
- *lssubsys(1)- lssubsys コマンドは、指定のサブシステムを含む階層を一覧表示します。*

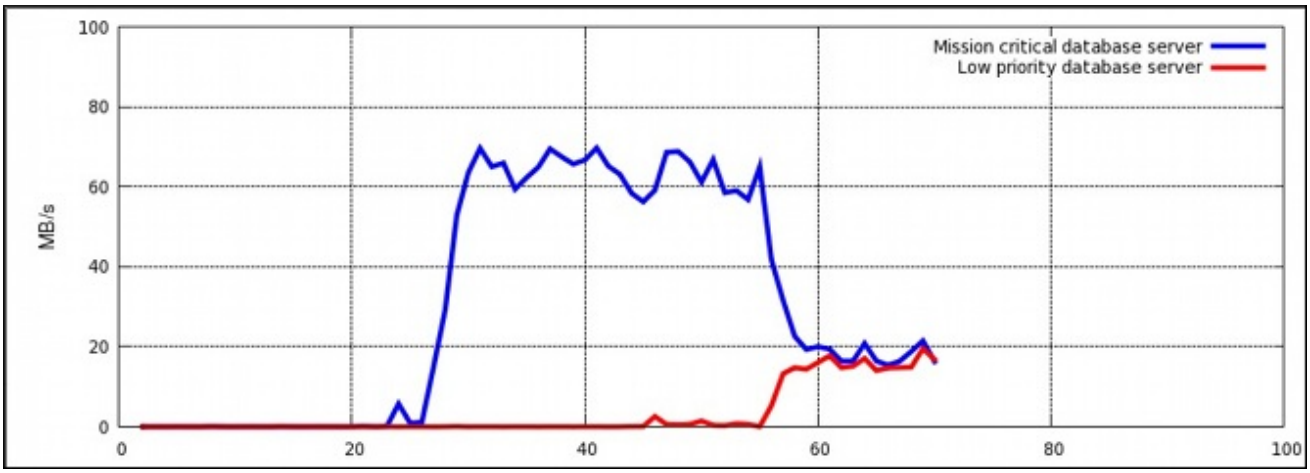
第4章 コントロールグループアプリケーションの例

本章では、cgroup の機能を活用した適用例について説明します。

4.1. データベース I/O の優先順位設定

独自の専用仮想ゲスト内でデータベースサーバーの各インスタンスを実行すると、その優先順位に基づいてデータベースごとにリソースを割り当てることができます。以下の例は、2 つの KVM ゲスト内で 2 つのデータベースサーバーを実行しているとします。データベースの 1 つが優先度の高いデータベースであり、もう 1 つは優先度の低いデータベースです。両方のデータベースサーバーが同時に実行する場合は、I/O 図4.1「リソースの割り当てなしで I/O スループット」、優先度の低いデータベースが起動したら（約 time 45）、I/O スループットは両データベースサーバーで同じです。

図4.1 リソースの割り当てなしで I/O スループット



[D]

優先度の高いデータベースサーバーを優先するには、予約済み I/O 操作が多数ある cgroup に割り当てることができます。また、優先度の低いデータベースサーバーは、予約済み I/O 操作の数が少ない cgroup に割り当てることができます。手順4.1「I/O スループット優先化」、ホストシステム ですべての手順を実行します。

手順4.1 I/O スループット優先化

1. リソースアカウンティングが両方のサービスであることを確認してください。

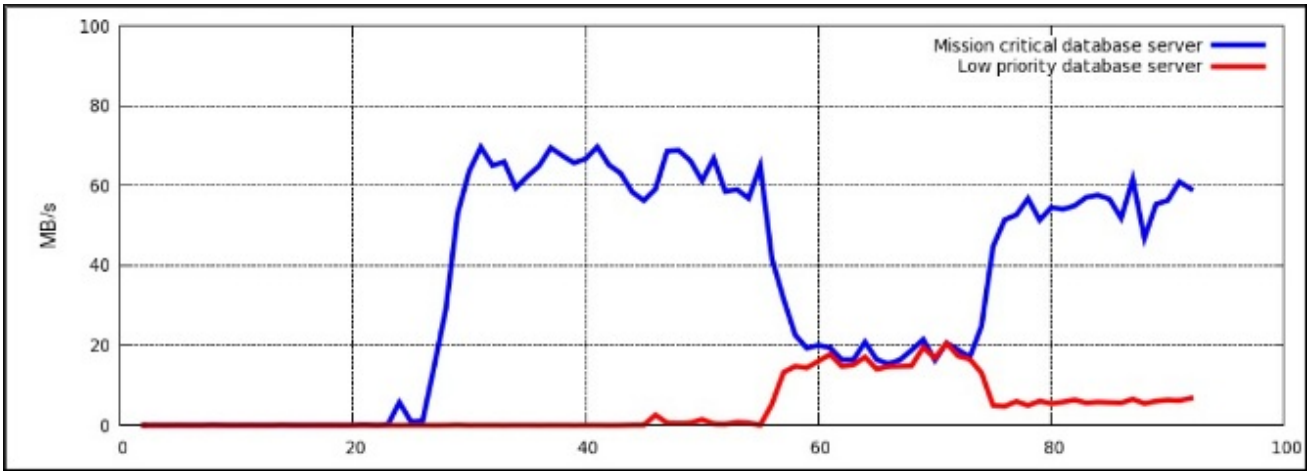
```
~]# systemctl set-property db1.service BlockIOAccounting=true
~]# systemctl set-property db2.service BlockIOAccounting=true
```

2. 優先度が高いサービスに対する 10:1 の比率を設定します。これらのサービスユニットで実行しているプロセスは、

```
~]# systemctl set-property db1.service BlockIOWeight=1000
~]# systemctl set-property db2.service BlockIOWeight=100
```

図4.2「リソース割り当てを使用した I/O スループット」 では、優先順位の低いデータベースの制限と、優先度の高いデータベースの優先順位付けの結果を示しています。データベースサーバーが適切な cgroups（約 75）に移行するとすぐに、I/O スループットは両方のサーバー間で分割され、10:1 の比率で I/O スループットに分割されます。

図4.2 リソース割り当てを使用した I/O スループット



[D]

または、優先度の低いデータベースに対してブロックデバイス I/O スロットリングを使用して、その読み取り操作および書き込み操作の数を制限することもできます。詳細は、blkio コントローラーの説明を参照してください コントローラー固有のカーネルドキュメント 。

4.2. ネットワークトラフィックの設定

1 台のサーバーシステムで複数のネットワーク関連のサービスを実行する場合は、これらのサービス間でネットワークの優先度を定義することが重要です。優先度を定義することで、特定のサービスから発信されるパケットは、他のサービスから送信されるパケットよりも優先度が高くなります。たとえば、このような優先度は、サーバーシステムが同時に NFS および Samba サーバーとして機能する場合に役立ちます。NFS トラフィックは、高スループットを低下させるため、優先度が高くなければなりません。Samba トラフィックでは、NFS サーバーのパフォーマンスを向上させることができます。

net_prio コントローラーを使用して、cgroups のプロセスに対するネットワークの優先度を設定できます。これらの優先度は Service(ToS)フィールドビットに変換され、すべてのパケットに組み込まれます。「NFS と Samba）の、2 つのファイル共有サービス（NFS および Samba） 手順4.2「ファイル共有サービスのネットワーク優先度の設定」 。

手順4.2 ファイル共有サービスのネットワーク優先度の設定

1. **net_prio サブシステムを /cgroup/net_prio cgroup にアタッチします。**

```
~]# mkdir sys/fs/cgroup/net_prio
~]# mount -t cgroup -o net_prio none sys/fs/cgroup/net_prio
```

2. **各サービスに 2 つの cgroups を作成します。**

```
~]# mkdir sys/fs/cgroup/net_prio/nfs_high
~]# mkdir sys/fs/cgroup/net_prio/samba_low
```

3. **nfs サービスを nfs _high cgroup に自動的に移動するには、以下の行を /etc/sysconfig/nfs ファイルに追加します。**

```
CGROUP_DAEMON="net_prio:nfs_high"
```

この設定により、nfs サービスの開始または再起動時に、nfs サービスプロセスが nfs_high cgroup に移動されるようになります。

4. **smbd サービスには、/etc/sysconfig ディレクトリーに設定ファイルがありません。smbd サービスを自動的に samba_low cgroup に移動するには、以下の行を /etc/cgrules.conf ファイルに追加します。**

```
*:smbd          net_prio        samba_low
```

このルールは、/usr/sbin/smbd だけでなく、すべての smbd サービスを samba _low cgroup に移動します。

nmbd サービスおよび winbindd サービスのルールは、同様の方法で samba_low cgroup に移動できます。

5. **cgred サービスを起動し、直前の手順から設定を読み込みます。**

```
~]# systemctl start cgred
Starting CGroup Rules Engine Daemon:                       [  OK  ]
```

6. **この例では、両方のサービスが eth1 ネットワークインターフェースを使用していると仮定します。各 cgroup のネットワーク優先度を定義します。ここで、1 は優先順位が低いため、**

10 は優先度が高いことを意味します。

```
~]# echo "eth1 1" > /sys/fs/cgroup/net_prio/samba_low/net_prio.ifpriomap
~]# echo "eth1 10" > /sys/fs/cgroup/net_prio/nfs_high/net_prio.ifpriomap
```

7. *nfs サービスおよび smb サービスを起動し、そのプロセスが正しい cgroups に移動されたかどうかを確認します。*

```
~]# systemctl start smb
Starting SMB services:                                     [  OK  ]
~]# cat /sys/fs/cgroup/net_prio/samba_low/tasks
16122
16124
~]# systemctl start nfs
Starting NFS services:                                     [  OK  ]
Starting NFS quotas:                                       [  OK  ]
Starting NFS mountd:                                       [  OK  ]
Stopping RPC idmapd:                                       [  OK  ]
Starting RPC idmapd:                                       [  OK  ]
Starting NFS daemon:                                       [  OK  ]
~]# cat sys/fs/cgroup/net_prio/nfs_high/tasks
16321
16325
16376
```

NFS からのネットワークトラフィックは、Samba からのトラフィックよりも優先度が高くなります。

手順4.2「ファイル共有サービスのネットワーク優先度の設定」 サブシステムを使用して、クライアントアプリケーションのネットワークの優先度を設定できます（例： Firefox）。

付録A 改訂履歴

改訂 0.0-1.10 Mon Aug 05 2019 Marie Doleželová
7.7 GA リリース向けバージョン

改訂 0.0-1.6 Wed Nov 11 2015 Jana Heves
7.2 GA リリース向けのバージョン

改訂 0.0-1.4 Thu Feb 19 2015 Radek Bíba
7.1 GA リリース向けバージョンLinux コンテナーは、別のドキュメントに移動しました。

改訂 0.0-1.0 Mon Jul 21 2014 Peter Ondrejka

改訂 0.0-0.14 Mon May 13 2013 Peter Ondrejka
7.0 GA リリース向けバージョン