Red Hat Enterprise Linux 7
リソース管理ガイド

Red Hat Enterprise Linux 7 でシステムリソースを管理する

Milan Navrátil
Douglas Silas
Eva Majoršinová
Martin Prpič
Peter Ondrejka
Rüdiger Landmann

Red Hat Enterprise Linux 7 リソース管理ガイド

Red Hat Enterprise Linux 7 でシステムリソースを管理する

Milan Navrátil
Red Hat Customer Content Services
mnavrati@redhat.com

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

Red Hat Enterprise Linux 7 でシステムリソースを管理します。

目次

第1章 コントロールグループについて (cgroup)

1.1. コントロールグループとは

本書で *cgroup* と略称される *コントロールグループ*は、システム上で実行されるプロセスの階層的に順序付けられたグループ間で、CPU 時間、システムメモリー、ネットワーク帯域幅またはこれらのリソースの組み合わせなどのリソースの割り当てを可能にする Linux カーネル機能です。cgroup を使用することにより、システム管理者は、システムリソースの割り当て、優先順位付け、拒否、管理および監視におけるより詳細なレベルの制御を行うことができます。ハードウェアリソースはアプリケーションおよびユーザー間でスマートに分割することができ、これにより全体の効率が向上します。

コントロールグループは、プロセスを階層的にグループ化し、ラベル付けを行う方法や、リソース制限をそれらのプロセスに適用する方法を提供します。これまでは、管理者がプロセスの **niceness** 値で調節できる同様の量のシステムリソースがすべてのプロセスに割り当てられていました。このアプローチでは、アプリケーションの相対的な重要度を問わず、数多くのプロセスが使用されるアプリケーションに対して、より少ないプロセスを使用するアプリケーションよりも多くのリソースが割り当てられました。

Red Hat Enterprise Linux 7 では、cpgroup 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。そのため、システムリソースの管理は、**systemctl** コマンドを使用するか、または systemd ユニットファイルを変更することにより実行できます。詳細は、*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、システム管理者は *libcgroup* パッケージの **cgconfig** コマンドを使用してカスタム cgroup 階層を作成しました。しかし、このパッケージを使用するとデフォルトの cgroup 階層との競合が容易に生じるため、現在は非推奨になっています。ただし、*libcgroup* はとくに **net-prio** サブシステムを使用する場合など**systemd** がまだ適用されない特定のケースに対応するために依然として使用することができます。*3章libcgroup ツールの使用* を参照してください。

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る) と対話するための高レベルなインターフェースを提供します。リソース管理を行うための主な cgroup コントローラーは **cpu**、**memory** および **blkio** です。デフォルトで有効にされるコントローラーの一覧は、Red Hat Enterprise Linux 7 で利用可能なコントローラー を参照してください。リソースコントローラーおよびそれらの設定可能なパラメーターの詳細は、コントローラー固有のカーネルについてのドキュメントを参照してください。

1.2. デフォルトの cgroup 階層

デフォルトで、**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作成し、cgroup ツリーの統一された構造を提供します。**systemctl** コマンドを使用すると、「コントロールグループの作成」 で説明されているように、カスタムスライスを作成してこの構造をさらに変更することができます。また、**systemd** は重要なカーネルリソースコントローラー (Red Hat Enterprise Linux 7 で利用可能なコントローラー を参照) の階層を **/sys/fs/cgroup/** ディレクトリーに自動マウントします。

警告

まだ **systemd** でサポートされていないコントローラー (とくに**net-prio** コントローラー) の階層をマウントし、これを処理するには、**libcgroup** パッケージの非推奨の **cgconfig** ツールを使用することができます。**systemd** でマウントされているデフォルト階層を変更するために**libcgropup** ツールを使用すると、予期しない動作が発生する可能性があるので、その場合はこのツールは使用しないでください。**libcgroup** ライブラリーは Red Hat Enterprise Linux の今後のバージョンでは削除されます。**cgconfig** の使用方法の詳細については、*3章libcgroup ツールの使用* を参照してください。

systemd ユニットタイプ

システム上で実行され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、リソースを制御する目的で使用される 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詳細一覧については、『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**」の章を参照してください)。

- **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です。サービスは指定されたプロセスをカプセル化し、これらのプロセスが 1 つのセットとして起動したり、停止したりできるようにします。サービスの名前は以下のように指定されます。

  ```
  name.service
  ```

  ここで、*name* はサービスの名前を表します。

- **スコープ**: 外部で作成されたプロセスのグループです。スコープは**fork()** 機能により任意のプロセスで起動し、停止された後に、ランタイム時に **systemd** によって登録されたプロセスをカプセル化します。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* はスコープの名前を表します。

- **スライス**: 階層的に編成されたユニットのグループです。スライスにはプロセスが含まれず、スコープやサービスが置かれる階層を編成します。実際のプロセスはスコープやサービスに含まれます。この階層ツリーでは、スライスユニットのすべての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、*parent-name*.**slice** というスライスは*parent*.**slice** 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-name2*.**slice** などの独自のサブスライスを持たせることができます。

  以下のように表示される 1 つの root スライスがあります。

  ```
  -.slice
  ```

サービス、スコープおよびスライスユニットは cgroup ツリー内のオブジェクトに直接マップされます。これらのユニットがアクティブになると、それぞれがユニット名から作成される cgroup パスに直接マップされます。たとえば、**test-waldo.slice** にある **ex.service** は cgroup の **test.slice/test-waldo.slice/ex.service/** にマップされます。

サービス、スコープおよびスライスはシステム管理者が手動で作成することも、またはプログラムによって動的に作成されることもあります。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数の組み込みサービスを定義します。さらに、以下の 4 種類のスライスがデフォルトで作成されます。

- **-.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の場所
- **user.slice**: すべてのユーザーセッションのデフォルトの場所
- **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の場所

すべてのユーザーセッションが、仮想マシンやコンテナープロセスと共に分離したスコープユニットに自動的に置かれることに注意してください。さらにすべてのユーザーには暗黙的なサブスライスが割り当てられます。上記のデフォルト設定に加え、システム管理者は新規スライスを定義し、サービスおよびスコープをこれらのスライスに割り当てることができます。

以下は、cgroup ツリーの単純化したサンプルツリーです。この出力は「コントロールグループに関する情報の取得」で説明されているように**systemd-cgls** コマンドで生成されています。

```
├─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-located.service
  │ └─5328 /usr/lib/systemd/systemd-located
  ├─colord.service
  │ └─5001 /usr/libexec/colord
  ├─sshd.service
  │ └─1191 /usr/sbin/sshd -D
  │
  ...
```

上記のように、サービスおよびスコープにはプロセスが含まれ、それらは独自のプロセスを所有しないスライスに置かれています。唯一の例外となるのは、特殊な **systemd.slice** にある PID 1 です。さらに**-.slice** はツリー全体の root で暗黙的に識別されるため、ここには表示されていません。

サービスおよびスライスユニットは、「ユニットファイルの変更」) で説明されているように永続的なユニットファイルで設定することも、PID 1 への API 呼び出しでランタイム時に動的に作成することもできます (「オンラインのドキュメント」で API の参照情報をご覧ください)。スコープユニットは前者の方法でのみ作成できます。API 呼び出しで動的に作成されるユニットは一時的であり、ランタイム時にのみ存在します。一時的なユニットは、終了するとすぐに自動的にリリースされ、非アクティブにされるか、またはシステムの再起動が行われます。

1.3. Linux カーネルにおける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一定範囲のリソースコントローラーを提供します。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は、**/proc/cgroups** で確認するか、または **lssubsys** モニタリングツールを使用して確認できます。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力アクセスまたはブロックデバイスからの出力アクセスの制限を設定します。
- **cpu**: cgroup タスクに CPU へのアクセスを提供するために CPU スケジューラーを使用します。これは同一のマウント時に **cpuacct** コントローラーと共にマウントされます。
- **cpuacct**: cgroup 内のタスクで使用される CPU リソースについての自動レポートを作成します。これは同一のマウント時に **cpu** コントローラーと共にマウントされます。
- **cpuset**: 個別の CPU (マルチコアシステム上) およびメモリーノードを cgroup 内のタスクに割り当てます。
- **devices**: cgroup 内のタスクについてデバイスへのアクセスを許可または拒否します。
- **freezer**: cgroup 内のタスクを一時停止または再開します。
- **memory**: cgroup 内のタスクによって使用されるメモリーに対する制限を設定し、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**: Linux トラフィックコントローラー (**tc** コマンド) が特定の cgroup タスクから発信されるパケットを識別することを可能にするクラス識別子 (**classid**) を使用して、ネットワークパケットにタグを付けます。**net_cls** のサブシステム **net_filter** (iptables) もこのタグを使用してパケットなどに対してアクションを実行できます。**net_filter** は、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るパケット (skb->sk) を識別することを可能にするファイアウォール識別子 (**fwid**) を使用して、ネットワークソケットにタグを付けます。
- **perf_event**: **perf** ツールを使用した cgroup のモニタリングを可能にします。
- **hugetlb**: サイズの大きい仮想メモリーページの使用を許可し、これらのページへの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幅広い範囲の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については、カーネル関連のドキュメント (コントローラー固有のカーネルについてのドキュメント にある参照情報の一覧) を参照してください。

1.4. その他のリソース

systemd 下のリソース制御、ユニット階層およびカーネルリソースコントローラーについての詳細情報は、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 関連の Systemd ドキュメント

以下の man ページには、**systemd** 下の統一された cgroup 階層についての一般的な情報が含まれます。

- **systemd.resource-control**(5): システムユニットによって共有されるリソース制御についての各種設定オプションについて説明しています。
- **systemd.unit**(5): すべてのユニット設定ファイルの共通オプションについて説明しています。
- **systemd.slice**(5): **.slice** ユニットについての一般的な情報を提供します。
- **systemd.scope**(5): **.scope** ユニットについての一般的な情報を提供します。
- **systemd.service**(5): **.service** ユニットについての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についてのドキュメント

kernel-doc パッケージはすべてのリソースコントローラーの詳細ドキュメントを提供します。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ています。Optional チャンネルをサブスクライブする前に、Optional ソフトウェアについて対象範囲の詳細 を参照してください。次に Red Hat カスタマーポータル上の 証明書ベースの管理を使用して、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 というタイトルの記事に記載されている手順を実行してください。Optional チャンネルから *kernel-doc* をインストールするには、**root** として以下を入力します。

```
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** サブシステム:**blkio-controller.txt**
- **cpuacct** サブシステム:**cpuacct.txt**
- **cpuset** サブシステム:**cpusets.txt**
- **devices** サブシステム:**devices.txt**
- **freezer** サブシステム:**freezer-subsystem.txt**
- **memory** サブシステム:**memory.txt**
- **net_cls** サブシステム:**net_cls.txt**

また、**cpu** サブシステムについての詳しい情報は、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**

オンラインのドキュメント

- Red Hat Enterprise Linux 7 システム管理者のガイド: **システム管理者のガイド** は Red Hat Enterprise Linux 7 の導入、設定および管理に関する情報を記載しています。このガイドには、**systemd** の概念と **systemd** を使用したシステムの管理方法についての詳細が記載されています。
- The D-Bus API of systemd: **systemd** と対話するために使用される D-Bus API コマンドについての参考資料です。

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および管理に関連するタスクの概要を示します。本書は、cgroup 管理の方法として推奨され、今後サポートされる予定の **systemd** によって提供されるユーティリティーに焦点を当てています。以前のバージョンの Red Hat Enterprise Linux では、cgroup の作成および管理の目的で *libcgroup* パッケージを使用しています。このパッケージは後方互換性を確保するために依然として使用できますが (警告 を参照)、Red Hat Enterprise Linux の今後のバージョンではサポートされません。

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し、systemd のコマンドラインユーティリティーで管理できるシステムユニットにバインドされます。アプリケーションのタイプによっては、リソース管理設定を **一時的な** 設定にすることも、**永続的な** 設定にすることもできます。

サービスの **一時的な cgroup** を作成するには、サービスを **`systemd-run`** コマンドで起動します。これにより、ランタイム時にサービスによって消費されるリソースに制限を設定することができます。各種の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 cgroup を動的に作成できます。API の参考情報については「オンラインのドキュメント」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の再起動後も保存されるため、この設定を使用して自動的に起動するサービスを管理することができます。スコープユニットはこの方法では作成できないことに注意してください。

2.1.1. systemd-run を使用した一時的な cgroup 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** ユニットを作成および起動し、このユニットでカスタムコマンドを実行するために使用されます。サービスユニットで実行されるコマンドはバックグランドで非同期に開始し、それらのコマンドは **systemd** プロセスから起動します。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、呼び出し側の実行環境を継承します。この場合は同期的に実行されます。

指定された cgroup でコマンドを実行するには、**`root`** として以下を実行します。

```
systemd-run --unit=name --scope --slice=slice_name command
```

- *name* は、このユニットを認識するのに使用する名前を表します。**`--unit`** が指定されていない場合、ユニット名は自動的に生成されます。この名前は **`systemctl`** 出力のユニットを表すため、ユニットを説明するような名前を選択することをお勧めします。この名前はユニットのランタイム時に固有のもの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、新たに作成した **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、または固有の名前を渡すことによって新規の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`** 構文の末尾に置き、このコマンドのパラメーターが **`systemd-run`** のパラメーターと混同しないようにします。

上記のオプション以外にも、**systemd-run** で利用可能ないくつかのパラメーターがあります。たとえば、**--description** はユニットの記述を作成し、**--remain-after-exit** はサービスのプロセスを停止した後のランタイム情報を収集することを許可します。**--machine** オプションは限定されたコンテナーでコマンドを実行します。さらに詳しくは、**systemd-run**(1) の man ページを参照してください。

> **例2.1 systemd-run を使用した新規サービスの起動**
>
> 以下のコマンドを使用して、**test** という新規スライスのサービスユニットで**top** ユーティリティーを実行します。**root** として以下を実行します。
>
> ```
> ~]# systemd-run --unit=toptest --slice=test top -b
> ```
>
> 以下のメッセージが、サービスを正常に起動したことを確認するために表示されます。
>
> ```
> Running as unit toptest.service
> ```
>
> ここで、**toptest.service** という名前は **systemctl** コマンドで cgroup をモニタリングまたは変更するために使用できます。

2.1.2. 永続的な cgroup 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 システム管理者のガイドの **systemd によるサービス管理** の章を参照してください)。このコマンドを実行すると、ユニットファイルが **/usr/lib/systemd/system/** ディレクトリーに自動的に作成されます。cgroup に永続的な変更を加えるには、そのユニットファイルの設定パラメーターを追加または変更します。詳細は、「ユニットファイルの変更」を参照してください。

2.2. コントロールグループの削除

一時的な cgroup は、それらに含まれるプロセスが終了するとすぐに自動的にリリースされます。**--remain-after-exit** オプションを **systemd-run** に渡すことで、ユニットのプロセスがランタイム情報の収集を終了した後もユニットを実行状態にすることができます。この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。1 つ以上のユニットのプロセスを終了するには、**root** として以下を入力します。

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

name を、**httpd.service** などのユニットの名前に置き換えます。**--kill-who** を使用して、終了する cgroup のプロセスを選択します。複数のプロセスを同時に kill するには、PID のコンマ区切りの一覧を渡します。*signal* を指定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** の man ページを参照してください。

永続的な cgroup は、以下を実行してユニットを無効にし、その設定ファイルを削除する際にリリースされます。

```
systemctl disable name.service
```

ここで、*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 によって監視されるそれぞれの永続的なユニットには、**/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。これは手動で実行することも、**systemctl set-property** コマンドを使用してコマンドラインインターフェースで実行することもできます。

2.3.1. コマンドラインインターフェースでのパラメーターの設定

systemctl set-property コマンドを実行すると、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することができます。これを実行するには、**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、*parameter* を変更するパラメーターの名前に、*value* をこのパラメーターに割り当てる新規の値に置き換えます。

すべてのパラメーターをランタイム時に変更できる訳ではありませんが、リソース制御に関連するパラメーターのほとんどは変更することができます。詳細の一覧については、「ユニットファイルの変更」を参照してください。**systemctl set-property** を実行すると、複数のプロパティーを一度に変更できることに注意してください。この方法は、プロパティーを個別に設定する方法よりも推奨されます。

変更はユニットファイルに即時に適用され、書き込まれるため、再起動後もそれらの変更は保持されます。設定を一時的な設定にする **--runtime** オプションを渡すことでこの動作を変更できます。

```
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 CPU および **httpd.service** のメモリー使用を制限するには、以下を入力します。

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

この一時的な変更を行うには、**--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 サービスのユニットファイルは、リソース管理に役立つハイレベルの設定パラメーターを数多く提供します。これらのパラメーターは、カーネルで有効にしておく必要のある Linux cgroup コントローラーと通信します。これらのパラメーターを使用すると、CPU、メモリー消費、ブロック IO およびさらに詳細に設定されたユニットプロパティーを管理することができます。

CPU の管理

cpu コントローラーはデフォルトでカーネルで有効にされるため、すべてのシステムサービスはこれに含ま

れるプロセスの数にかかわらず、同じ量の CPU 時間を受け取ります。このデフォルト動作は、**/etc/systemd/system.conf** 設定ファイルの **DefaultControllers** パラメーターで変更することができます。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指示文を使用します。

CPUShares=*value*

value を CPU 共有の数に置き換えます。デフォルト値は 1024 ですが、この数を増やすことにより、さらに多くの CPU 時間をユニットに割り当てることができます。このパラメーターは、ユニットファイルで **CPUAccounting** がオンにされていることを示します。

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 関連のコントロールパラメーターを確認するには、コントローラー固有のカーネルのついてのドキュメントの **cpu** 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 消費の制限

Apache サービスにデフォルトの 1024 ではなく、1500 の CPU 共有を割り当てるには、**/usr/lib/systemd/system/httpd.service** ユニットファイルで **CPUShares** 設定を変更します。

```
[Service]
CPUShares=1500
```

変更を適用するには、systemd の設定を再度読み込み、Apache を再起動することで、変更されたサービスファイルが考慮されるようにします。

```
~]# 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についての制限を施行するには、ユニット設定ファイルの **[Service]** セクションで以下の指示文を使用します。

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値に置き換えます。測定単位としてキロバイト、メガバイト、ギガバイト、またはテラバイトを指定するためにサフィックスの **K**、**M**、**G**、または **T** を使用します。さらに、**MemoryAccounting** パラメーターを同じユニットについて有効にしておく必要があります。

MemoryLimit 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、コントローラー固有のカーネルのついてのドキュメントにある **memory** コントローラーの説明を参照してください。

例2.4 ユニットのメモリー消費の制限

Apache サービスに 1GB メモリー制限を割り当てるには、**/usr/lib/systemd/system/httpd.service** 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 の設定を再度読み込み、Apache を再起動することで、変更されたサービスファイルが考慮されるようにします。

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

ブロック IO の管理

ブロック 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指示文を使用します。以下に記載されている指示文は **BlockIOAccounting** パラメーターが有効にされていることを仮定しています。

BlockIOWeight=*value*

value を、実行されているプロセスの新たなブロック IO 全体の重み (weight) に置き換えます。10 から 1000 の間の単一の値を選択します。デフォルト設定は 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 の重みに置き換えます。*device_name* を名前か、またはデバイスへのパスのいずれかに置き換えます。**BlockIOWeight** の場合のように、10 から 1000 までの単一の加重値を設定することができます。

BlockIOReadBandwidth=*device_name value*

この指示文により、ユニットの特定の帯域幅を制限することができます。*device_name* をデバイスの名前か、またはブロックデバイスノードへのパスに置き換えます。*value* は帯域幅レートを表します。測定単位を指定するには、サフィックスの **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 秒あたりのバイト単位で解釈されます。

BlockIOWriteBandwidth=*device_name value*

指定されたデバイスの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。

上記の指示文はそれぞれ対応する cgroup パラメーターを制御します。コントローラー固有のカーネルのついてのドキュメント の **blkio** コントローラーの説明を参照してください。

注記

現在、**blkio** リソースコントローラーはバッファリングされた書き込み操作をサポートしません。これは主として直接 I/O を対象としているため、バッファリング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。他方、バッファリングされた読み取り操作はサポートされ、**BlockIOReadBandwidth** 制限は直接およびバッファリングされた読み取りの両方に正しく適用されます。

例2.5 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 の重みを下げるには、以下のテキストを **/usr/lib/systemd/system/httpd.service** ユニットファイルに追加します。

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

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

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

変更を適用するには、systemd の設定を再度読み込み、Apache を再起動することで、変更されたサービスファイルが考慮されるようにします。

```
~]# 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指示文がほかにもいくつかあります。

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。ここで、*device_name* は **/proc/devices** で指定されるデバイスノードまたはデバイスグループ名を表しています。**options** を **r**、**w**、および **m** の組み合わせに置き換え、ユニットからデバイスノードの読み取り、書き込み、または作成を実行できるようにします。

DevicePolicy=*value*

ここで、*value* は以下のいずれかになります。**strict** (**DeviceAllow** で明示的に指定されるアクセスのタイプのみを許可します)、**closed** (/dev/null、/dev/zero、/dev/full、/dev/random、および /dev/urandom を含む標準的な擬似デバイスへのアクセスを許可します) または **auto** (明示的な **DeviceAllow** がない場合にすべてのデバイスへのアクセスを許可します。これはデフォルトの動作です)。

Slice=*slice_name*

slice_name を、ユニットを入れ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。スコープユニットはそれらの親スライスに結び付けられているため、スコープユニットをこの方法で編成することができません。

ExecStartPost=*command*

現在、**systemd** は cgroup 機能のサブセットをサポートします。ただし、サービスの swap 使用を防ぐために、**ExecStartPost=** オプションを、***memory.memsw.limit_in_bytes*** パラメーターの設定と共に使用することもできます。**ExecStartPost=** の詳細は、**systemd.service(5)** man ページを参照してください。

例2.6 cgroup オプションの設定

特定の *example* サービスの swap 使用を避けるために、***memory.memsw.limit_in_bytes*** 設定をユニットの **MemoryLimit=** と同じ値に設定する必要があるとします。

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

変更を適用するには、**systemd** の設定を再度読み込み、サービスを再起動することで、変更された設定が反映されるようにします。

```
~]# 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、システムユニットを一覧表示し、それらのステータスを表示します。さらに、コントロールグループの階層を表示するために **systemd-cgls** コマンドが提供され、リアルタイムでリソース消費を監視するために **systemd-cgtop** が提供されています。

2.4.1. ユニットの制限

以下のコマンドを使用して、システム上のすべてのアクティブなユニットを一覧表示します。

```
systemctl list-units
```

list-units オプションが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、以下のみを実行する場合と同じ出力が表示され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
...
```

上記のように表示される出力には 5 つの列が含まれます。

- **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です。「systemd ユニットタイプ」で言及されているように、**スライス**、**スコープ** および **サービス** の 3 つのユニットタイプがリソースコントロールと関連しています。**systemd** のユニットタイプの詳細一覧については、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** の章を参照してください。
- **LOAD**: ユニット設定がファイルが適切に読み込まれたかどうかを示します。ユニットファイルが読み込みに失敗した場合、フィールドには **loaded** ではなく **error** 状態が含まれます。他のユニットの読み込み状態には、**stub**、**merged**、および **masked** があります。
- **ACTIVE**: SUB を一般化したユニットの高レベルのアクティブ化の状態です。
- **SUB**: ユニットの低レベルのアクティブ化の状態です。使用できる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です。

デフォルトで、**systemctl** はアクティブなユニットのみを一覧表示します (ACTIVE フィールドの高レベルのアクティブ化の状態)。**--all** オプションを使用して非アクティブなユニットも表示できます。出力一覧の情報量を制限するには、**サービス** および **スライス** などのコンマ区切りのユニットタイプの一覧が必要な **--type** (**-t**) パラメーター、または **loaded** および **masked** などのユニットの読み込み状態を使用します。

例2.7 systemctl list-units の使用

システム上で使用されるすべての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
```

すべてのアクティブなマスクされたサービスを一覧表示するには、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たすべてのユニットファイルとそれらのステータスを一覧表示するには、以下を入力します。

```
systemctl list-unit-files
```

2.4.2. コントロールグループ階層の表示

上記のコマンドはユニットレベルを超えて cgroup 内で実行されている実際のプロセスを表示することはありません。さらに、**systemctl** の出力はユニットの階層を表示しません。これらはいずれも cgroup に従って実行中のプロセスをグループ化する **systemd-cgls** コマンドを使用して実行できます。システム上に 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に実行されると、cgroup 階層全体が返されます。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、以下のように表示されます。

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

仮想マシンまたはコンテナーを実行中の場合にのみマシンスライスが存在することに注意してください。cgroup ツリーの詳細は、「systemd ユニットタイプ」 を参照してください。

systemd-cgls の出力を減らし、階層の指定された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。

代替方法として、**systemctl status** コマンドを使用してシステムユニットについての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 status name
```

systemctl status の詳細は、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** の章を参照してください。

例2.8 コントロールグループ階層の表示

memory 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は、選択したコントローラーと対話するサービスを一覧表示します。別の方法として、cgroup ツリーの一部を表示して特定のサービス、スライス、またはスコープユニットを確認することができます。

```
~]# 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

...
```

上記のツールのほかにも、**systemd** は Linux コンテナーのモニタリングに特化した**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

上記の **systemctl** コマンドは、より高いレベルのユニット階層のモニタリングを可能にしますが、Linux カーネルのどのリソースコントローラーがどのプロセスで実際に使用されているかは表示しません。この情報は専用のプロセスファイルに保存されます。これを表示するには、**root** として以下を入力します。

```
cat proc/PID/cgroup
```

ここで、*PID* は検査するプロセスの ID を表します。デフォルトで、この一覧は**systemd** で開始されるすべてのユニットの一覧と同様です。**systemd** はすべてのデフォルトコントローラーを自動的にマウントするためです。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:/
```

このファイルを検査することにより、プロセスが systemd ユニットファイルの仕様で定義される必要な cgroup に置かれているかどうかを判別することができます。

2.4.4. リソース消費のモニタリング

systemd-cgls コマンドは cgroup 階層の静的なスナップショットを提供します。リソース使用 (CPU、メモリーおよび IO) 別に順序付けられる現在実行中の cgroup の動的なアカウントを表示するには、以下を使用します。

```
systemd-cgtop
```

systemd-cgtop の動作、指定される統計および制御オプションは**top** ユーティリティーのオプションと同様です。詳細は、**systemd-cgtop**(1) の man ページを参照してください。

2.5. その他のリソース

Red Hat Enterprise Linux 上のシステムリソースを管理するために **systemd**および関連ツールを使用する方法については、以下に記載されている情報源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 関連の Systemd ツールの Man ページ

- **systemd-run**(1): この man ページは、**systemd-run** ユーティリティーのコマンドラインのすべてのオプションを一覧表示します。
- **systemctl**(1): **systemctl** ユーティリティーのこの man ページは、利用可能なオプションおよびコマンドを一覧表示します。
- **systemd-cgls**(1): この man ページは、**systemd-cgls** ユーティリティーのコマンドラインのすべてのオプションを一覧表示します。

- **systemd-cgtop**(1): この man ページには、**systemd-cgtop** ユーティリティーのコマンドラインのすべてのオプションの一覧が含まれています。
- **machinectl**(1): この man ページは、**machinectl** ユーティリティーのコマンドラインのすべてのオプションを一覧表示します。
- **systemd.kill**(5): この man ページは、システムユニットの kill 設定オプションの概要を示します。

コントローラー固有のカーネルのついてのドキュメント

kernel-doc パッケージはすべてのリソースコントローラーの詳細ドキュメントを提供します。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ています。Optional チャンネルをサブスクライブする前に、対象範囲の詳細 を参照してください。次に Red Hat カスタマーポータル上の証明書ベースの管理を使用して、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 というタイトルの記事に記載されている手順を実行してください。Optional チャンネルから *kernel-doc* をインストールするには、**root** として以下を入力します。

```
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** サブシステム:**blkio-controller.txt**
- **cpuacct** サブシステム:**cpuacct.txt**
- **cpuset** サブシステム:**cpusets.txt**
- **devices** サブシステム:**devices.txt**
- **freezer** サブシステム:**freezer-subsystem.txt**
- **memory** サブシステム:**memory.txt**
- **net_cls** subsystem — **net_cls.txt**

さらに、**cpu** サブシステムについての詳しい情報は、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**

オンラインのドキュメント

- Red Hat Enterprise Linux 7 システム管理者のガイド: **システム管理者のガイド** は Red Hat Enterprise Linux 7 の導入、設定および管理についての関連情報を記載しています。このガイドは、システムの基本的な知識のあるシステム管理者を対象にしています。
- The D-Bus API of systemd: **systemd** にアクセスするために使用される D-Bus API コマンドについての参考資料です。

第3章 libcgroup ツールの使用

以前のバージョンの Red Hat Enterprise Linux において cgroup 管理用の主なツールであった *libcgroup* パッケージは現在非推奨になっています。競合を避けるために、現在 **systemd** の排他的なドメインになっているデフォルトリソースコントローラー (Red Hat Enterprise Linux 7 で利用可能なコントローラーに記載) に *libcgroup* ツールは使用しないでください。*libcgroup* ツールを適用するための限られたスペースはありますが、このスペースは **net_prio** などの現在 **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。

以下のセクションでは、デフォルトの階層のシステムとの競合を生じさせず関連するシナリオで *libcgroup* ツールを使用する方法について説明します。

注記

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

```
~]# 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のようにカーネルにコンパイルされません。これは、マウントを試行する前に読み込まれている必要のあるモジュールです。このモジュールを読み込むには、**root** として以下を入力します。

```
modprobe netprio_cgroup
```

3.1. 階層のマウント

自動的にマウントされないカーネルの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** セクションを編集することにより階層を追加したり、分離させたりします。この方法はコントローラーの接続を永続化します。つまり設定がシステムの起動後も保持されることを意味します。代替方法として、**mount** コマンドを使用して、現行セッション用にのみ一時的なマウントを作成することもできます。

cgconfig サービスの使用

libcgroup-tools パッケージと共にインストールされる **cgconfig** サービスは、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を提供します。デフォルトで、このサービスは自動的に起動しません。**cgconfig** を起動すると、これは **/etc/cgconfig.conf** 設定ファイルの設定を適用します。したがって、設定はセッションごとに再作成され、永続化します。**cgconfig** を停止する場合、これがマウントしたすべての階層がアンマウントすることに注意してください。

libcgroup パッケージと共にインストールされるデフォルトの**/etc/cgconfig.conf** ファイルにはいずれの設定内容も記載されておらず、**systemd** が主なリソースコントローラーを自動的にマウントするという情報のみが含まれます。

mount、*group* および *template* の 3 つのタイプのエントリーを**/etc/cgconfig.conf** に作成できます。マウントエントリーは仮想ファイルシステムとして階層を作成し、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割り当てます。Red Hat Enterprise Linux 7 では、デフォルトの階層が **/sys/fs/cgroup/** ディレクトリーに自動的にマウントされるため、**cgconfig** はデフォルト以外のコントローラーを割り当てるためにのみ使用されます。mount 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。例については、例3.1「mount エントリーの作成」 を参照してください。

例3.1 mount 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、設定を適用するために **cgconfig** サービスを再起動します。

```
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。グループエントリーの詳細は、「cgroup パラメーターの設定」 を参照してください。

/etc/cgconfig.conf のテンプレートエントリーは、すべてのプロセスに適用されるグループ定義を作成するために使用できます。

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。これを実行するには、まずマウントポイントを **/sys/fs/cgroup/** ディレクトリーに作成します。このディレクトリーで**systemd** は主なリソースコントローラーをマウントします。**root** として以下を入力します。

```
mkdir /sys/fs/cgroup/name
```

name を新規のマウント先の名前に置き換えます。この名前には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、階層をマウントして、1 つ以上のサブシステムを同時に割り当てます。**root** として以下を入力します。

```
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、マウントされるデバイスを宛先フォルダーと共に指定します。**-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 コントローラーを割り当てる mount コマンドの使用

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、まずマウントポイントを作成します。

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

次に、**net_prio** を直前のステップで作成したマウント先にマウントします。

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

lssubsys コマンドを使用し、現在のマウントポイントと共にすべての利用可能な階層を一覧表示することで、階層を正しく割り当てたかどうかを確認できます (「コントローラーの一覧表示」を参照)。

```
~]# 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、この設定ファイルの**mount** セクションから設定の指示文を単純に削除することにより、階層をアンマウントできます。次に、新規の設定を適用するためにサービスを再起動します。

同様に、**root** として以下のコマンドを実行して階層をアンマウントできます。

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

controller_name を、割り当てを解除するリソースが含まれる階層の名前に置き換えます。

警告

手動でマウントした階層のみを削除するために **umount** を使用するようにしてください。デフォルトコントローラー (Red Hat Enterprise Linux 7 で利用可能なコントローラーに一覧を記載) が含まれる階層の割り当てを解除すると、システム再起動が必要な複雑な状況が発生する可能性があります。

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 cgroup を作成します。**cgcreate** の構文は以下のようになります。

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

ここで、

- **-t** (オプション) は、ユーザー (ユーザー ID、uid) とグループ (グループ ID、gid) を指定して、この cgroup の **tasks** 疑似ファイルを所有するためのオプションです。このユーザーは cgroup にタスクを追加することができます。

  

  注記

  cgroup からタスクを削除するには、異なる cgroup にプロセスを移動するのが唯一の手段である点に注意してください。プロセスを移動するには、ユーザーは **移動先の** cgroup への書き込みアクセスが必要となります。元の cgroup への書き込みアクセスは必要ではありません。

- **-a** (オプション): ユーザー (ユーザー ID、uid) とグループ (グループ ID、gid) を指定して、この cgroup の **tasks** 以外の全疑似ファイルを所有するようにします。このユーザーは cgroup 内のタスクについてのシステムリソースへのアクセスを変更できます。
- **-g**: cgroup が作成されるべき階層を、それらの階層に関連付けられるコンマ区切りの*subsystems* 一覧として指定します。コントローラーの一覧の後には、コロンおよび階層に対して相対的な子グループへの *path* が続きます。このパスには、階層のマウントポイントを入れないでください。

同じ階層内の cgroup はすべて同じコントローラーを持つため、子グループは親グループと同じコントローラーを持つことになります。

代替方法として、cgroup の子を直接作成できます。これを実行するには、**mkdir** コマンドを使用します。

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

例:

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

3.4. コントロールグループの削除

cgroup を **cgcreate** の構文に似た構文を持つ **cgdelete** コマンドに置き換えます。**root** として以下のコマンドを実行します。

```
cgdelete controllers:path
```

ここで、

- *controllers* はコントローラーのコンマ区切りの一覧です。
- *path* は、階層の root に対して相対的な cgroup へのパスです。

例:

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

cgdelete では **-r** オプションが指定されていると、すべてのサブグループを再帰的に解除することもできます。

cgroup を削除すると、そのプロセスはすべて親グループに移動することに注意してください。

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、または **cgset** コマンドを使用することにより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** に加えられる変更は再起動後も保持され、**cgset** は現行セッション用にのみ cgroup パラメーターを変更します。

/etc/cgconfig.conf の変更

/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 の名前に置き換えます。*controller* は変更するコントローラーの名前を表します。**systemd** で自動的にマウントされたデフォルトコントローラーのいずれかではなく、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規の値に置き換えます。**permissions** 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

/etc/cgconfig.conf の変更を有効にするには、**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、設定ファイルに指定される階層が再構築されますが、すべてのマウントされた階層には影響はありません。**systemctl restart** コマンドを実行してサービスを起動することができますが、まず **cgconfig** サービスを停止することをお勧めします。

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

次に、設定ファイルを開いてこれを編集します。変更を保存した後に以下のコマンドを使って **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で **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。このコマンド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
```

ここで、

- *parameter* は設定するパラメーターで、特定の cgroup のディレクトリー内のファイルに対応します。
- *value* はパラメーター用の値です。
- *path_to_cgroup* は、**階層の root に対して相対的な** cgroup へのパスです。

cgset で設定できる値は、特定の階層でより高く設定されている値によって左右される可能性があります。たとえば、**group1** がシステム上の CPU 0 のみを使用するように制限されている場合、**group1/subgroup1** が CPU 0 および 1 を使用するように、または 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 1 つの 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** を使用してパラメーターをコピーするための構文は以下のようになります。

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

ここで、

- *path_to_source_cgroup* は、コピーするパラメーターを持つ cgroup の、その階層の root グループに対して相対的なパスです。
- *path_to_target_cgroup* は、その階層の root グループに対して相対的な、コピー先 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
```

ここで、

- *controllers* は、コンマ区切りのリソースコントローラーの一覧、または利用可能なすべてのサブシステムに関連付けられた階層内のプロセスを起動するために ***** に置き換えます。同じ名前の cgroup が複数存在する場合には、**-g** オプションを指定すると、プロセスがそれらのグループのそれぞれに移動することに注意してください。
- *path_to_cgroup* は、その階層内の cgroup へのパスです
- *pidlist* は、*プロセス識別子* (PID) のスペースで区切られた一覧です。

-g オプションが指定されていない場合、**cgclassify** は **/etc/cgrules.conf** を自動的に検索し、最初に適用できる設定ラインを使用します。このラインに基づいて、**cgclassify** はプロセスの移動先となる階層と cgroup を判別します。正常に移動するには、移動先の階層が存在していなければなりません。**/etc/cgrules.conf** に指定されるサブシステムは、**/etc/cgconfig.conf** の対応する階層に適切に設定されている必要があります。

pid の前に **--sticky** オプションを追加すると、同じ cgroup 内に任意の子プロセスを維持することもできます。このオプションを設定せず、かつ **cgred** サービスが稼働中の場合、子プロセスは **/etc/cgrules.conf** の設定に基づいて cgroup に割り当てられます。そのプロセス自体は、それを起動した 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セットに従ってタスクを cgroup に移動する**cgred** サービス (**cgrulesengd** サービスを起動する) を使用することもできます。手動で接続されたコントローラーを管理するには **cgred** のみを使用します。**/etc/cgrules.conf** ファイル内のエントリーは、次の 2 つの形式のいずれかを取ります。

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

例:

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

このエントリーは、**maria** というユーザーに属するプロセスはいずれも**/usergroup/staff** cgroup 内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ことを指定します。特定のコマンドを特定の cgroup に関連付けるには、以下のようにして *command* パラメーターを追加します。

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

このエントリーにより、**maria** という名前のユーザーが **ftp** コマンドを使用する時には、**devices** サブシステムが入っている階層の **/usergroup/staff/ftp** cgroup へプロセスが自動的に移動するように指定されるようになります。 ただし、このデーモンは、該当する条件が満たされている場合にのみ、プロセスを cgroup に移動する点に注意してください。このため、**ftp** プロセスが、誤ったグループで短時間実行される可能性があります。また、そのプロセスが誤ったグループ内にある間に子プロセスが急速に生成した場合には、それらは移動しない可能性があります。

/etc/cgrules.conf ファイル内のエントリーには、以下のような表記を追加することが可能です。

- **@** : *user* にプレフィックスを付けた場合には、個別のユーザーではなくグループを示します。 たとえば、**@admins** は **admins** グループ内のすべてのユーザーです。

- ***** : 「すべて」を示します。たとえば、**subsystem** フィールド内の ***** はすべてのサブシステムを示します。
- **%** : 上の行の項目と同じ項目であることを示します。以下はその例です。

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

3.7. コントロールグループ内のプロセスの起動

cgexec コマンドを実行して手動で作成される 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
```

ここで、

- *controllers* は、コンマ区切りのリソースコントローラーの一覧、または利用可能なすべてのサブシステムに関連付けられた階層内のプロセスを起動するために ***** に置き換えます。「cgroup パラメーターの設定」 で説明されている **cgset** コマンドの場合のように、同じ名前の cgroup が存在する場合、**-g** オプションを指定すると、それらの各グループにプロセスが作成されることに注意してください。
- *path_to_cgroup* は、階層に対して相対的な 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。

command の前に **-- sticky** オプションを追加すると、同じ cgroup の子プロセスを維持することもできます。このオプションを設定しないで **cgred** サービスが稼働していると、子プロセスは **/etc/cgrules.conf** の設定に基づいて cgroup に割り当てられます。しかし、プロセス自体はそれを起動した cgroup 内に残ります。

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それらのパラメーターについての情報を取得するためにいくつかのユーティリティーが含まれます。

コントローラーの一覧表示

カーネルで使用可能なコントローラーおよびそれらがどのようにして階層にまとめてマウントされているかを確認するには、以下のコマンドを実行します。

```
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確認するには、以下のコマンドを実行します。

```
lssubsys -m controllers
```

ここで、*controllers* は、対象となるサブシステムの一覧を表します。**lssubsys -m** コマンドでは、各階層ごとの最上位のマウントポイントのみが返される点に注意してください。

コントロールグループの確認

システム上に cgroup を一覧表示するには、**root** として以下を実行します。

```
lscgroup
```

出力を特定の階層に制限するには、***controller*:*path*** の形式でコントローラーとパスを指定します。以下はその例です。

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

上記のコマンドは、**cpuset** コントローラーが接続されている階層内の**adminusers** cgroup のサブグループのみを一覧表示します。

コントロールグループのパラメーターの表示

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

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

ここで *parameter* は、コントローラーの値を含む擬似ファイルで、*list_of_cgroups* は cgroup のスペース区切りの一覧です。

実際のパラメーターの名前が不明な場合には、以下のようなコマンドを使用してください。

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

3.9. その他のリソース

cgroup コマンドの確実な参照情報は、*libcgroup* パッケージで提供される man ページに記載されています。

インストールされているドキュメント

libcgroup に関連した man ページ

- **cgclassify**(1): **cgclassify** コマンドは、実行中のタスクを 1 つまたは複数の cgroup に移動するために使用されます。
- **cgclear**(1): **cgclear** コマンドは、1 つの階層内のすべての cgroup を削除するために使用されます。

  cgconfig.conf(5): cgroup は **cgconfig.conf** ファイル内で定義されます。
- **cgconfigparser**(8): **cgconfigparser** コマンドは **cgconfig.conf** ファイルを解析して、階層をマウントします。
- **cgcreate**(1): **cgcreate** コマンドは、階層内に新たな cgroup を作成します。
- **cgdelete**(1): **cgdelete** コマンドは指定された cgroup を削除します。
- **cgexec**(1): **cgexec** コマンドは指定された cgroup 内の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 cgroup に属する場合にこれを判別するためのルールが含まれます。
- **cgrulesengd**(8): **cgrulesengd** サービスは、タスクを cgroup に配分します。
- **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、階層内の cgroup を一覧表示します。
- **lssubsys**(1): **lssubsys** コマンドは、指定された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の適用例

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 の優先度設定

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ことにより、優先度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。次の例を検討してください。システムが 2 台の KVM ゲスト内で 2 つのデータベースを実行しています。一方のデータベースは優先度が高く、もう一方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稼働すると、I/O スループットが低減し、両データベースからの要求に同等に対応します。図4.1「リソース割り当てを使用しない I/O スループット」 は、このシナリオを示しています。— 優先度の低いデータベースが起動すると (時間軸 45 前後)、I/O スループットが両データベースサーバーで同じになります。

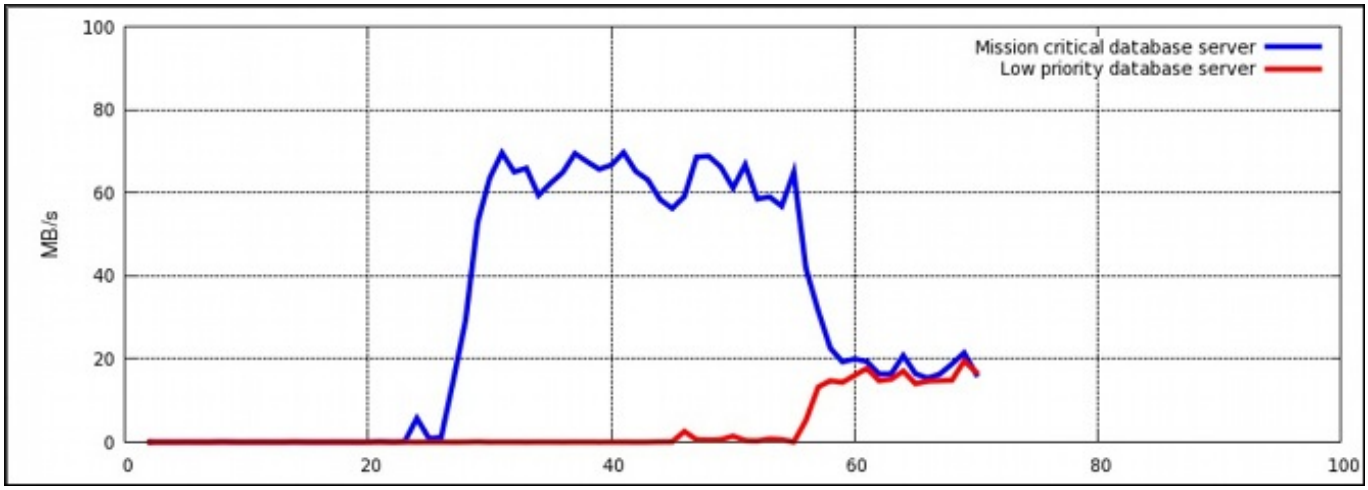

図4.1 リソース割り当てを使用しない I/O スループット

優先度の高いデータベースサーバーを優先するには、予約済みの I/O 操作の高い数値を cgroup に割り当て、優先度の低いデータベースサーバーには予約済み I/O 操作の低い数値を cgroup に割り当てます。この設定は 手順4.1「I/O スループットの優先度設定」 の手順に従って行います。作業はすべてホストシステム上で実行します。

手順4.1 I/O スループットの優先度設定

1. リソースアカウンティングが優先度の高いサービスと優先度の低いサービスの両方に設定されていることを確認します。

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

2. 優先度の高いサービスと優先度の低いサービスの比を 10:1 に設定します。それらのサービスで実行されているプロセスは、それらのサービスで利用可能なリソースのみを使用します。

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

図4.2「I/O スループットとリソース割り当て」 は、優先度の低いデータベースを制限し、優先度の高いデータベースを優先した結果を図示しています。データベースサーバーが適切な cgroup に移動されるとすぐに (時間軸 75 前後)、I/O スループットが 10:1 の比率で両サーバー間で分配されます。

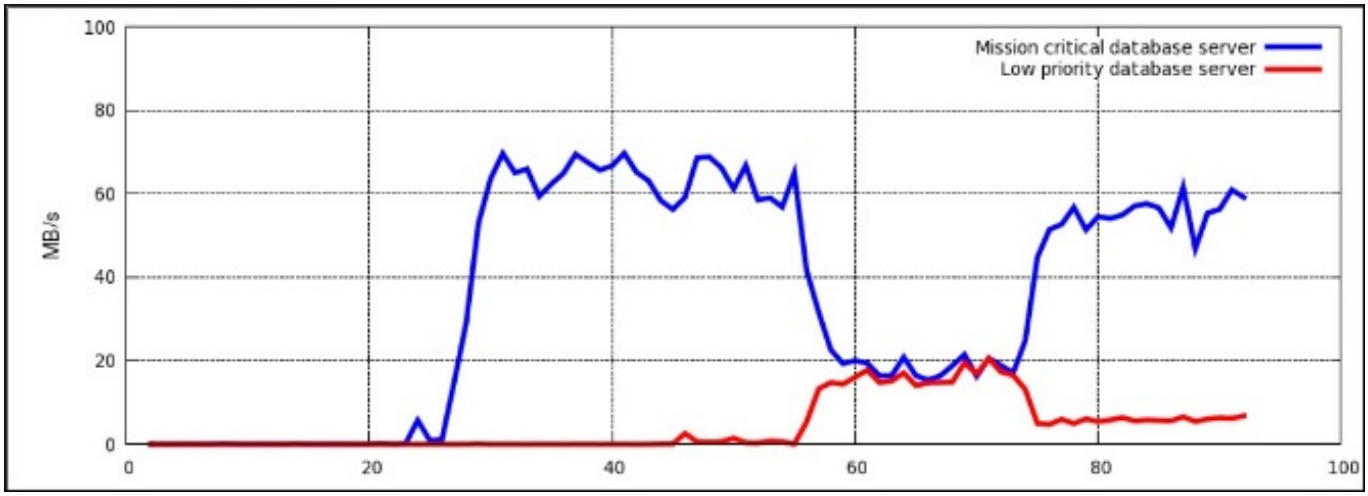

図4.2 I/O スループットとリソース割り当て

別の方法として、ブロックデバイスの I/O スロットリングを使用して、優先度の低いデータベースの読み取り/書き込み操作の回数を制限することができます。詳細は、コントローラー固有のカーネルについてのドキュメント の **blkio** コントローラーに関する説明を参照してください。

4.2. ネットワークトラフィックの優先度設定

単一のサーバーシステムでネットワーク関連サービスを複数実行している場合には、そ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。これらの優先度を定義することにより、特定のサーバーから発信されるパッケージの優先度を、その他のサービスから発信されるパッケージよりも高くすることができます。たとえば、そのような優先度は、サーバーシステムが NFS および Samba サーバーとして同時に機能する場合に役立ちます。NFS のトラフィックは、ユーザーが高スループットを期待するので、優先度を高くする必要があります。Samba のトラフィックは、NFS サーバーのパフォーマンスを向上させるために、優先度を低くすることができます。

net_prio コントローラーを使用して、cgroup 内のプロセスのネットワークの優先順位を設定することができます。次に、これらの優先度が Type of Service (ToS) ビットに変換され、各パケットに埋め込まれます。2 つのファイル共有サービス (NFS および Samba) の優先度を設定するには、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」 の手順に従ってください。

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** コントローラーはカーネルにコンパイルされません。これは手動で読み込む必要のあるモジュールです。これを実行するには、以下を入力します。

```
~]# modprobe netprio_cgroup
```

2. **net_prio** サブシステムを **/cgroup/net_prio** cgroup に接続します。

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

3. サービスごとに 2 つの cgroup を作成します。

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

4. **nfs_high** cgroup に **nfs** サービスを自動的に移動するには、**/etc/sysconfig/nfs** ファイルに

以下の行を追加します。

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

この設定により、**nfs** サービスが起動または再起動する際に**nfs** サービスプロセスが**nfs_high** cgroup に確実に移動します。

5. **smbd** サービスの設定ファイルは **/etc/sysconfig** ディレクトリーにはありません。**smbd** サービスを **samba_low** cgroup に自動的に移動するには、**/etc/cgrules.conf** ファイルに以下の行を追加してください。

```
*:smbd                  net_prio                samba_low
```

このルールにより、**/usr/sbin/smbd** のみではなく、すべての**smbd** サービスが **samba_low** cgroup に移動する点に注意してください。

同様に、**nmbd** および **winbindd** サービスを **samba_low** cgroup に移動させるルールを定義することができます。

6. **cgred** サービスを起動して、前の手順からの設定を読み込みます。

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

7. 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ことを前提とします。各 cgroup にネットワークの優先度を定義します。ここで **1** は優先度が低く、**10** は優先度が高いことを示します。

```
~]# 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
```

8. **nfs** および **smb** サービスを起動し、それらのプロセスが正しい cgroup に移動したことを確認します。

```
~]# 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 から発信されるネットワークトラフィックの優先度が、Samba から発信されるトラフィックよりも高くなりました。

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」と同様に、**net_prio** サブシステムはクライアントアプリケーション (例: Firefox) のネットワーク優先度設定に使用することができます。

付録A 改訂履歴

改訂 0.0-1.7.3	**Mon Jul 3 2017**	**Aiko Sasaki**
翻訳完了		
改訂 0.0-1.7.2	**Mon Feb 27 2017**	**Terry Chuang**
翻訳ファイルを XML ソースバージョン 0.0-1.7 と同期		
改訂 0.0-1.7.1	**Sun Feb 26 2017**	**Terry Chuang**
翻訳ファイルを XML ソースバージョン 0.0-1.7 と同期		
改訂 0.0-1.7	**Mon Oct 17 2016**	**Marie Doleželová**
7.3 GA 公開用バージョン		
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 コンテナーを別のマニュアルに移行。		
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		