Red Hat Enterprise Linux 7

リソース管理ガイド

cgroup を使用して RHEL 7 でシステムリソースを管理する

Last Updated: 2023-03-25

Red Hat Enterprise Linux 7 リソース管理ガイド

cgroup を使用して RHEL 7 でシステムリソースを管理する

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

このガイドでは、Red Hat Enterprise Linux 7 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が可能な Linux カーネル機能、コントロールグループ (cgroups) について説明します。

目次

第1章 コントロールグループ (CGROUPS) の概要

1.1. コントロールグループとは

コントロールグループ (このガイドでの略称 *cgroup*) は、Linux カーネルの機能のことで、システムで実行されているプロセスのグループを階層的に順序付けて、そのグループ間で CPU 時間、システムメモリー、ネットワーク帯域幅、またはこのようなリソースの組み合わせなど、リソースの割り当てが可能になります。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、および監視をきめ細かく制御できます。ハードウェアリソースをアプリケーションとユーザー間でスマートに分割できるため、全体的な効率が向上します。

制御グループは、プロセスを階層的にグループ化してラベル付けして、そのようなグループにリソース制限を適用する方法を提供します。従来は、全プロセスは、同様のシステムリソースを受け取っていました。このリソースは、管理者がプロセスの **nice** 値で調節できていました。このアプローチでは、これらのアプリケーションの相対的な重要性に関係なく、プロセス数が多いアプリケーションは、少数のプロセスを含むアプリケーションよりも多くのリソースを受け取っていました。

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、**systemctl** コマンドを使用するか、systemd ユニットファイルを変更することで、システムリソースを管理できます。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、システム管理者が libcgroup パッケージの **cgconfig** コマンドを使用してカスタム cgroup 階層を構築していました。このパッケージは非推奨になり、デフォルトの cgroup 階層と簡単に競合する可能性があるため、使用はお勧めしません。ただし、libcgroup は、**net-prio** サブシステムを使用する場合など、**systemd** がまだ適用されていない特定のケースに対応するために引き続き利用できます。*3章libcgroup ツールの使用*を参照してください。

前述のツールは、Linux カーネルの cgroup コントローラー (サブシステムとも呼ばれます) とやり取りするための高レベルのインターフェイスを提供します。リソース管理用の主要な 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なっているコントローラーのリストについては、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とそ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

デフォルトでは、**systemd** は *slice*、*scope*、および *service* ユニットの階層を自動的に作成して、cgroup ツリーに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」 の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できます。また、**systemd** は、**/sys/fs/cgroup/** ディレクトリーの重要なカーネルリソースコントローラーの階層を自動的にマウントします (Red Hat Enterprise Linux 7 で利用可能なコントローラー を参照)。

警告

libcgroup パッケージの非推奨の **cgconfig** ツールを使用して、**systemd** でまだサポートされていないコントローラー (特に **net-prio** コントローラー) の階層をマウントおよび処理できます。予期しない動作が発生する可能性があるので、**libcgroupup** ツールを使用して、**systemd** によってマウントされたデフォルトの階層を変更しないでください。**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る予定です。**cgconfig** の使用方法の詳細は、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

システムで実行され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 サービスの管理** という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** が開始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。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** で登録します。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。スライスにはプロセスは含まれず、スコープとサービスが配置され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。この階層ツリーでは、スライスユニットのすべての名前が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、スライスの名前が次のようになっているとします。

  parent-name.**slice**

  これは、*parent-name*.**slice** 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含めることができます。

  次のような 1 つのルート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、ユニット名から構築される cgroup パスを制御するよ

うに直接マッピングされます。たとえば、**test-waldo.slice** にある **ex.service** は、cgroup **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、システム管理者によって手動で作成されるか、プログラムによって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数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**: ルートスライス。
- **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と Linux コンテナーの既定の場所。

すべてのユーザーセッションは、仮想マシンとコンテナープロセスと同様に、別のスコープユニットに自動的に配置されることに注意してください。さらに、すべてのユーザーに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て、システム管理者は新しいスライスを定義し、それらにサービスとスコープを割り当てることができます。

次のツリーは、cgroup ツリーの単純化された例です。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 **systemd-cgls** コマンドで生成されました。

```
├─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
  │
 ...
```

ここにあるように、サービスと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配置されています。唯一の例外は、**-.slice** として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で暗黙的に識別されるため、表示されないことに注意してください。

「ユニットファイルの変更」 で説明されているように、サービスおよびスライスユニットは、永続ユニットファイルを使用して設定できます。または PID 1 への API 呼び出しによって実行時に動的に作成されます (API リファレンスについては 「オンラインドキュメント」 を参照)。スコープユニットは動

的にしか作成できません。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あり、実行時にのみ存在します。一時的なユニットは、終了するか、非アクティブになるか、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、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一連のリソースコントローラーを提供します。**/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リストを検索するか、**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、cgroup タスクが CPU にアクセスできるようにします。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 内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**: マルチコアシステムとメモリーノードで個別の CPU を cgroup のタスクに割り当てます。
- **devices**: cgroup 内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 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) を有効にするクラス識別子 (**classic**) 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) でも、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ができます。**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、特定の制御グループタスクから発信されたパケットを識別できるようになります。
- **perf_event**: perf ツールを使用して cgroup 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用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、カーネルのドキュメント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 のリファレンス一覧)。

1.4. 関連情報

systemd でのリソース制御、ユニット階層、およびカーネルリソースコントローラーの詳細は、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 関連の Systemd ドキュメント

次のマニュアルページには、**systemd** の下の統合された cgroup 階層に関する一般的な情報が含まれています。

- **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の設定オプションについて説明します。
- **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ます。
- **systemd.slice** (5): **.slice** ユニットに関する一般的な情報を提供します。
- **systemd.scope** (5): **.scope** ユニットに関する一般的な情報を提供します。
- **systemd.service** (5): **.service** 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含まれています。このパッケージは、オプションのサブスクリプションチャネルに含まれています。Optional チャネルにサブスクライブする前に、Optional ソフトウェアの 対象範囲の詳細 を参照してから、Red Hat Customer Portal の How to access Optional and Supplementary channels, and -devel packages using Red Hat Subscription Manager (RHSM)? という記事に記載されている手順に従ってください。Optional チャネルから kernel-doc をインストールするには、**root** で次のように入力します。

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

インストール後、次のファイルが **/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**
- **デバイス** サブシステム: **devices.txt**
- **冷凍庫** サブシステム: **freezer-subsystem.txt**
- **メモリー** サブシステム: **memory.txt**
- **net_cls** サブシステム: **net_cls.txt**

さらに、**cpu** サブシステムの詳細は、次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 - **システム管理者のガイド** では、Red Hat Enterprise Linux 7 のデプロイメント、設定、および管理に関する情報を説明しています。このガイドには、**systemd** の概念の詳細な説明と、**systemd** を使用したサービス管理の手順が含ま

れています。

- The D-Bus API of systemd: **systemd** とのやり取りに使用される 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について説明します。このガイドでは、cgroup 管理の方法として推奨されていて、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。Red Hat Enterprise Linux の以前のバージョンでは、cgroup の作成と管理に libcgroup パッケージを使用していました。このパッケージは、下位互換性を確保するために引き続き利用できますが (警告)、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 の観点からは、cgroup は、ユニットファイルで設定可能で、systemd のコマンドラインユーティリティーで管理可能な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応じて、リソース管理設定は **transient** または **persistent** を使用できます。

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開始します。このようにして、実行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 cgroup を動的に作成できます。API リファレンスについては 「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。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 **service** または **scope** ユニットを作成して起動し、そのユニットでカスタムのコマンドを実行するために使用します。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。スコープ単位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的です。

指定した cgroup でコマンドを実行するには、**root** で、次のように入力します。

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

- *name* は、ユニットに付ける名前を表します。**--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。**systemctl** 出力でユニットを表すため、わかりやすい名前を選択することをお勧めします。名前は、ユニットの実行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* は、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* は、サービスユニットで実行するコマンドに置き換えます。このコマンドのパラメーターが **systemd-run** のパラメーターと混同されないように、このコマンドを **systemd-run** 構文の最後に配置します。

上記のオプションに加えて、**systemd-run** で使用できるパラメーターが他にもいくつかあります。たと

えば、**--description** はユニットの説明を作成し、**--remain-after-exit** はサービスのプロセスを終了した後にランタイム情報を収集できるようにします。**--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。詳細は、**systemd-run** (1) マニュアルページを参照してください。

例2.1 systemd-run での新規サービスの開始

次のコマンドを使用して、**test** という新しい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** で次のコマンドを入力します。

```
~]# systemd-run --unit=toptest --slice=test top -b
```

サービスが正常に開始されたことを確認する次のメッセージが表示されます。

```
Running as unit toptest.service
```

これで、**toptest.service** という名前を使用して、**systemctl** コマンドで cgroup を監視または変更できるようになりました。

2.1.2. 永続的な cgroup の作成

システム起動時にユニットが自動的に開始され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** という章を参照)。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で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、含まれるプロセスが終了するとすぐに自動的に解放されます。**systemd-run** に **--remain-after-exit** オプションを指定すると、プロセスが終了した後もユニットを実行し続けてランタイム情報を収集できます。ユニットを正常に停止するには、次のように入力します。

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

name は、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** で、次のように入力します。

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

name は、ユニットの名前に置き換えます (例 : **httpd.service**)。**--kill-who** を使用して、cgroup から終了する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 は、指定し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** マニュアルページを参照してください。

永続的な cgroup は、ユニットが無効になると解放され、次のコマンドを実行してその設定ファイルが削除されます。

```
~]# 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 で監視される各永続ユニットには、**/usr/lib/systemd/system/** ディレクトリーにある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、または **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イスから行うことができます。

2.3.1. コマンドラインインターフェイ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実行中にリソース制御設定を永続的に変更できます。これを行うには、**root** として次の構文を使用します。

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

name は、変更する systemd ユニットの名前に、*parameter* は、変更するパラメーターの名前に、*value* は、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。

すべてのユニットパラメーターを実行時に変更できるわけではありませんが、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できます。完全なリストについては 「ユニットファイルの変更」 を参照してください。**systemctl set-property** を使用すると、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。これは、個別に設定する場合に比べ推奨の方法です。

変更は即座に適用され、再起動後も保持されるようにユニットファイルに書き込まれます。設定を一時的にする **--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 **httpd.service** の CPU とメモリーの使用量を制限するには、次のように入力します。

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

これを一時的な変更にするには、**--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高レベルの設定パラメーターを多数提供します。これらのパラメーターは、カーネルで有効にする必要がある 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、さらに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でデフォルトで有効になっているため、含まれるプロセスの数に関係なく、すべてのシステムサービスは同じ量の 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** セクションで次のディレクティブを使用します。

CPUShare = *value*

value は、CPU シェアの数に置き換えます。デフォルト値は 1024 です。数値を増やすと、より多くの CPU 時間がユニットに割り当てられます。**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** が自動的にオンになります。したがって、ユーザーは **systemd-cgtop** コマンドを使用してプロセッサーの使用状況を監視できます。

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 関連の制御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

デフォルトの 1024 ではなく 1500 の CPU シェアを Apache サービスに割り当てるには、次の内容で新しい **/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、変更されたサービスファイルが考慮されるように、systemd の設定をリロードして Apache を再起動します。

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

CPUQuota = *value*

value は、 CPU 時間クォータの値に置き換えて、指定された CPU 時間クォータを実行されるプロセスに割り当てます。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 で使用可能な合計 CPU 時間に対して、ユニットが最大で取得する CPU 時間を指定します。

100% を超える値は、複数の CPU が使用されていることを示します。**CPUQuota** は、統合されたコントロールグループ階層の **cpu.max** 属性と、従来の **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** が自動的にオンになります。したがって、ユーザーは **systemd-cgtop** コマンドを使用してプロセッサーの使用状況を監視できます。

例2.4 CPUQuota の使用

CPUQuota を 20% に設定すると、実行プロセスに対して、単一の CPU で 20% 以上の CPU 時間が渡されないようにします。

20% の Apache サービス CPU クォータを割り当てるには、次の内容を **/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、変更されたサービスファイルが考慮されるように、systemd の設定をリロードして 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量を制限するには、ユニット設定ファイルの **Service** セクションで次のディレクティブを使用します。

MemoryLimit=*value*

value は、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、ユニットに対して **MemoryAccounting** パラメーターを有効にする必要があります。

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ドキュメント の **メモリー** コントローラーの説明を参照してください。

例2.5 ユニットのメモリー消費の制限

Apache サービスに 1GB のメモリー制限を割り当てるには、**/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、変更されたサービスファイルが考慮されるように、systemd の設定をリロードして Apache を再起動します。

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

ブロック IO の管理

ブロック IO を管理するには、ユニット設定ファイルの **Service** セクションで次のディレクティブを使用します。以下にリストされているディレクティブは、**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*

value は、実行されたプロセスの新しい全体的なブロック IO の重みに置き換えます。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 は、*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* は、名前またはデバイスへのパスに置き換えます。**BlockIOWeight** と同様に、10 から 1000 までの単一の重み値を設定できます。

BlockIOReadBandwidth=*device_name value*

このディレクティブを使用すると、ユニットの特定の帯域幅を制限できます。*device_name* は、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。接尾辞 **K**、**M**、**G**、または **T** を使用して、測定単位を指定します。接尾辞のない値は、1 秒あたりのバイト数として解釈されます。

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。

前述の各ディレクティブは、対応する cgroup パラメーターを制御します。その他の CPU 関連の制御パラメーターについては、コントローラー固有のカーネルドキュメント 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、バッファーリングされた書き込み操作をサポートしていません。これは主にダイレクト I/O を対象としているため、バッファー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。一方、バッファリングされた読み取り操作はサポートされており、**BlockIOReadBandwidth** の制限は、直接読み取りと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2.6 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 の重みを下げるには、次のテキストを **/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
```

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

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

変更を適用するには、変更されたサービスファイルが考慮されるように、systemd の設定をリロードして 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その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。ここで、*device_name* は **/proc/devices** で指定されているデバイスノードまたはデバイスグループ名へのパスを表します。**オプション** を **r**、**w**、および **m** の組み合わせに置き換えて、ユニットがデバイスノードを読み取り、書き込み、または作成できるようにします。

DevicePolicy=*value*

ここで、*value* は、**strict** (**DeviceAllow** で明示的に指定されたタイプのアクセスのみを許可する)、**closed** (/dev/null、/dev/zero、/dev/full、/dev/random などの標準の疑似デバイスへのアクセ

スを許可する) および/dev/urandom) または **auto** (デフォルトの動作である明示的な **DeviceAllow** が存在しない場合、すべてのデバイスへのアクセスを許可します) のいずれかです。

Slice=*slice_name*

slice_name は、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。スコープユニットは、親スライスに関連付けられているため、この方法で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ています。ただし、回避策として、**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して、サービスのスワップの使用を防ぐことができます。**ExecStartPost=** の詳細は、**systemd.service (5)** の 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

特定の *example* サービスのスワップの使用を防ぐために、***memory.memsw.limit_in_bytes*** 設定をユニットの **MemoryLimit** = と同じ値に変更すると想定します。

```
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、変更された設定が考慮されるように、**systemd** 設定をリロード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、システムユニットを一覧表示し、そのステータスを表示します。また、制御グループの階層を表示する **systemd-cgls** コマンドと、それらのリソース消費をリアルタイムで監視する **systemd-cgtop** が提供されています。

2.4.1. ユニットのリスト

次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 オプションはデフォルトで実行されるので、このオプションを省略して以下を実行すると、同じ出力が得られます。

```
~]$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、次の 5 つの列が含まれています。

- **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 ユニットタイプ」 で説明したように、リソース制御には、**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関連しています。**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 システム管理者ガイド の **systemd での歯kステムの管理** という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には、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他に **stub**, **merged**, and **masked** などがあります。
- **ACTIVE**: ユニットのアクティベーションの状態 (概要レベル)。こちらは SUB の概要レベルの状態です。
- **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** 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ベーション状態)。**--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。出力リストの情報量を制限するには、**--type** (**-t**) パラメーターを使用します。このパラメーターには、**service** や **slice** などのユニットタイプ、または **loaded** や **masked** などのユニットの負荷状態のコンマ区切りリストが必要です。

> **例2.8 systemctl リスト単位の使用**
>
> システムで使用されているすべてのスライスのリストを表示するには、次のように入力します。
>
> ```
> ~]$ 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、次のように入力します。
>
> ```
> ~]$ systemctl -t service,masked
> ```

システムにインストールされているすべてのユニットファイルとそのステータスを一覧表示するには、次のように入力します。

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

2.4.2. コントロールグループ階層の表示

前述の一覧表示コマンドは、ユニットレベルを超えて cgroup で実行されている実際のプロセスを表示するものではありません。また、**systemctl** の出力には、ユニットの階層が表示されません。cgroup に従って実行中のプロセスをグループ化する **systemd-cgls** コマンドを使用することで、両方を実現できます。システムの cgroup 階層全体を表示するには、次のように入力します。

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

パラメーターなしで **systemd-cgls** を発行すると、cgroup 階層全体が返されます。cgroup ツリーの最上位レベルはスライスによって形成され、次のようになります。

■

```
├─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 ツリーの詳細については、次を参照してください。「systemd ユニットタイプ」 .

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、次を実行します。

```
~]$ 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。

別の方法として、**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 の詳細については、Red Hat Enterprise Linux 7 システム管理者ガイド の **systemd によるサービスの管理** という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、次のコマンドを実行します。

```
~]$ 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例では、選択したコントローラーと対話するサービスの一覧を表示します。別のアプローチは、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ことです。

```
~]# 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

...
```

前述のツールに加えて、**systemd** は Linux コンテナーの監視専用の **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

前述の **systemctl** コマンドを使用すると、上位レベルのユニット階層を監視できますが、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されません。この情報は専用のプロセスファイルに保存されます。表示するには、**root** として次のように入力します。

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

PID は、調べたいプロセスの ID を表します。デフォルトでは、リストは **systemd** によって開始されたすべてのユニットで同じです。これは、すべてのデフォルトコントローラーが自動的にマウントされるためです。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:/
```

このファイルを調べることで、systemd ユニットファイルの仕様で定義されているように、プロセスが正しい cgroup に配置さ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 cgroup の動的アカウントをリソース使用量 (CPU、メモリー、および IO) 順に並べて表示するには、次を使用します。

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

systemd-cgtop の動作、提供される統計、および制御オプションは、**top** ユーティリティーのものと似ています。詳細については 、**systemd-cgtop** (1) のマニュアルページを参照してください。

2.5. 関連情報

systemd および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の詳細については、以下のソース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 ツールのマニュアルページ

- **systemd-run** (1) – man ページには、 **systemd-run** ユーティリティーのすべてのコマンドラインオプションがリストされています。
- **systemctl** (1) – 使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マニュアルページ。
- **systemd-cgls** (1) – このマニュアルページには、 **systemd-cgls** ユーティリティーのすべてのコマンドラインオプションが一覧表示されています。
- **systemd-cgtop** (1) – man ページには、 **systemd-cgtop** ユーティリティーのすべてのコマンドラインオプションのリストが含まれています。
- **machinectl** (1) – このマニュアルページには、 **machinectl** ユーティリティーのすべてのコマンドラインオプションがリストされています。
- **systemd.kill** (5) – このマニュアルページでは、システムユニットの kill 設定オプションの概要を説明します。

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含まれています。このパッケージは、オプションのサブスクリプションチャネルに含まれています。Optional チャネルにサブスクライブする前に、Optional ソフトウェアの 対象範囲の詳細 を参照してから、Red Hat Customer Portal の How to access Optional and Supplementary channels, and -devel packages using Red Hat Subscription Manager (RHSM)? という記事に記載されている手順に従ってください。Optional チャネルから kernel-doc をインストールするには、**root** で次のように入力します。

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

インストール後、次のファイルが **/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**
- **デバイス** サブシステム: **devices.txt**
- **冷凍庫** サブシステム: **freezer-subsystem.txt**
- **メモリー** サブシステム: **memory.txt**
- **net_cls** サブシステム: **net_cls.txt**

さらに、**cpu** サブシステムの詳細は、次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 - **システム管理者のガイド** では、Red Hat Enterprise Linux 7 のデプロイメント、設定、および管理に関する情報を説明しています。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。
- The D-Bus API of systemd – **systemd** にアクセスするための D-Bus API コマンドのリファレンス。

第3章 LIBCGROUP ツールの使用

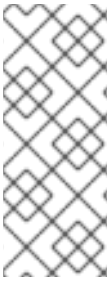
以前のバージョンの Red Hat Enterprise Linux で cgroup 管理用の主要なツールであった libcgroup パッケージは非推奨になりました。競合を 避ける ために、デフォルトのリソースコントローラー (Red Hat Enterprise Linux 7 で利用可能なコントローラー) が **systemd** の専用ドメインになりました。これにより、libcgroup ツールを適用するための限られたスペースが残されます。**net_prio** など、現在 **systemd** でサポートされていないコントローラーを管理する必要がある場合にのみ使用してください。

以下のセクションでは、デフォルトの階層システムと競合することなく、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にインストールされていることを確認してください。それらをインストールするには、**root** として実行します:

```
~]# 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でコンパイルされません。マウントする前にロードする必要があるモジュールです。このモジュールをロードするには、**root** として次のように入力します。

```
~]# 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 **マウント** セクションを編集して、階層を追加または分離します。この方法では、コントローラーの接続が永続的になります。つまり、システムの再起動後も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、現在のセッションのみの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。デフォルトでは、このサービスは自動的に開始されません。**cgconfig** を起動すると、**/etc/cgconfig.conf** 設定ファイルの設定が適用されます。したがって、設定はセッションごとに再作成され、永続的になります。**cgconfig** を停止すると、マウントしたすべての階層がアンマウントされ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** ファイルには設定設定は含まれず、**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。

/etc/cgconfig.conf には、*mount*、*group*、および *template* の 3 種類のエントリーを作成できます。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し、コントローラーをそれらの階層に接続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接続するためにのみ使用されます。マウントエントリーは、次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。例は、 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 ツリーに接続するには、次のテキストを **/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を使用して、リソースコントローラーのパラメーターを設定できます。見る「cgroup パラメーターの設定」グループエントリーの詳細については、

/etc/cgconfig.conf 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 コマンドの使い方

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、まず **/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。ここで、**systemd** はメインのリソースコントローラーをマウントします。**root** で次のコマンドを入力します。

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

name を新しいマウント先の名前に置き換えます。通常は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、同時に 1 つ以上のサブシステムを接続します。**root** で次のコマンドを入力します。

```
~]# 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 コマンドを使用してコントローラーを接続する

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、前の手順で作成した宛先に **net_prio** をマウントします。

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

lssubsys コマンド (「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、この設定ファイルの **マウント** セクションから設定ディレクティブを削除するだけでマウントを解除できます。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** として次のコマンドを実行することで、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を含む階層の名前に置き換えます。

警告

umount を使用して、自分で手動でマウントした階層のみを削除してください。デフォルトコントローラーを含む階層のデタッチ (Red Hat Enterprise Linux 7 で利用可能なコントローラー) は、システムの再起動を必要とする複雑な問題につながる可能性が最も高いでしょう。

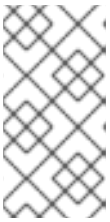
3.3. コントロールグループの作成

cgcreate コマンドを使用して、自分で作成した階層に一時的な cgroup を作成します。**cgcreate** の構文は次のとおりです。

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

ここでは、以下のようになります。

- **-t** (オプション) – この cgroup の **タスク** 疑似ファイルを所有するユーザー (ユーザー ID、uid) とグループ (グループ ID、gid) を指定します。このユーザーは、cgroup にタスクを追加できます。



注記

cgroup からプロセスを削除する唯一の方法は、プロセスを別の 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、ユーザーは移動 **先** の cgroup への書き込みアクセス権を持っている必要があります。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) – この cgroup の **タスク** 以外のすべての疑似ファイルを所有するユーザー (ユーザー ID、uid による) およびグループ (グループ ID、gid による) を指定します。このユーザーは、この cgroup 内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* のコンマ区切りのリストとして、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は、コロンと、階層に関連する子グループへの *パス* が続きます。パスに階層マウントポイントを含めないでください。

同じ階層内のすべての cgroup は同じコントローラーを持っているため、子グループはその親と同じコントローラーを持っています。

別の方法として、cgroup の子を直接作成することもできます。これを行うには、**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 と同様の構文を持つ **cgdelete** コマンドで cgroup を削除します。 **root** で以下のコマンドを入力します。

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

ここでは、以下のようになります。

- *controllers* は、コンマ区切りのコントローラーのリストです。

- *path* は、階層のルートに相対的な cgroup へのパスです。

以下に例を示します。

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

-r オプションが指定されている場合、**cgdelete** は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その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、**cgset** コマンドを使用して、制御グループのパラメーターを変更します。**/etc/cgconfig.conf** に加えられた変更は再起動後も保持されますが、**cgset** は現在のセッションの cgroup パラメーターのみを変更します。

/etc/cgconfig.conf の変更

/etc/cgconfig.conf の **Groups** セクションでコントローラーのパラメーターを設定できます。グループエントリーは、次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** によって自動的にマウントされたデフォルトのコントローラーではなく、自分で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* と *param_value* を、変更したい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。**権限** 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権限を定義するには、次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ために **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構築されますが、マウントされたすべての階層には影響しません。**systemctl restart** コマンドを実行してサービスを再起動できますが、最初に **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編集します。変更を保存したら、次のコマンドで **cgconfig** を再度開始できます。

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

cgset コマンドの使用

関連する cgroup を変更する権限を持つユーザーアカウントから **cgset** コマンドを実行して、コントローラーの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てください。

cgset の構文は次のとおりです。

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

ここでは、以下のようになります。

- *parameter* は設定するパラメーターで、指定された 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、特定の階層の上位に設定された値に依存する場合があります。たとえば、**グループ** 1 がシステム上で CPU 0 のみを使用するように制限されている場合、グループ 1 /**サブグループ** 1 が CPU 0 と 1 を使用したり、CPU 1 のみを使用したり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次のとおりです。

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

ここでは、以下のようになります。

- *path_to_source_cgroup* は、階層のルートグループに相対的な、パラメーターがコピーされる cgroup へのパスです。
- *path_to_target_cgroup* は、階層のルートグループに相対的な宛先 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
```

ここでは、以下のようになります。

- *controllers* は、リソースコントローラーのコンマ区切りリスト、または /* を使用して、使用可能なすべてのサブシステムに関連付けられた階層でプロセスを起動します。同じ名前の cgroup が複数ある場合、**-g** オプションはそれらの各グループのプロセスを移動することに注意してください。
- *path_to_cgroup* は、階層内の cgroup へのパスです。
- *pidlist* は、 *プロセス ID* (PID) のスペース区切りのリストです。

-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。この行に従って、**cgclassify** はプロセスを移動する階層と cgroup を決定します。移動を成功させるには、宛先階層が存在している必要があることに注意してください。**/etc/cgrules.conf** で指定されたサブシステムも、**/etc/cgconfig.conf** の対応する階層に対して適切に設定する必要があります。

--sticky オプションを *pid* の前に追加して、子プロセスを同じ cgroup に保持することもできます。このオプションを設定せず、**cgred** サービスが実行中の場合、子プロセスは **/etc/cgrules.conf** にある設定に基づいて cgroup に割り当てられます。ただし、プロセス自体は、それを開始した cgroup に残りま

す。

/etc/cgrules.conf ファイルに設定されたパラメーターに従ってタスクを cgroup に移動する **cgred** サービス (**cgrulesengd** サービスを開始する) を使用することもできます。**cgred** は、手動で接続されたコントローラーを管理する場合にのみ使用してください。**/etc/cgrules.conf** ファイルのエントリーは、次の 2 つの形式のいずれかを取ることができます。

- *ユーザーサブシステムcontrol_group* ;
- *ユーザー*: *コマンドサブシステムcontrol_group*。

以下に例を示します。

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

このエントリーは、**maria** という名前のユーザーに属するすべてのプロセスが、**/usergroup/staff** cgroup で指定されたパラメーターに従って **デバイス** 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、次のように *コマンド*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、**maria** という名前のユーザーが **ftp** コマンドを使用すると、**デバイス** サブシステムを含む階層内の **/usergroup/staff/ftp** cgroup にプロセスが自動的に移動することを指定するようになりました。ただし、適切な条件が満たされた後にのみ、デーモンがプロセスを cgroup に移動することに注意してください。したがって、**ftp** プロセスは不適切なグループで短時間実行される可能性があります。さらに、間違ったグループにいるときにプロセスが子をすぐに生成する場合、これらの子は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、次の追加の表記を含めることができます。

- **@** – *user* の前に付けると、個々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。
- * – すべてを表します。たとえば、**サブシステム** フィールドの * は、すべてのサブシステムを表します。
- **%** – 上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** の構文は次のとおりです。

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

ここでは、以下のようになります。

- *controllers* はコンマ区切りのコントローラーのリスト、または /* を使用して、使用可能なすべてのサブシステムに関連付けられた階層でプロセスを起動します。で説明されている **cgset** コマンドと同様に、「cgroup パラメーターの設定」、同じ名前の cgroup が存在する場合、**-g** オプションはそれらのグループのそれぞれにプロセスを作成します。

- *path_to_cgroup* は、階層に相対的な cgroup へのパスです。
- *command* は、cgroup で実行されるコマンドです。
- *引数* は、コマンドの任意の引数です。

コマンド の前に **--sticky** オプションを追加して、子プロセスを同じ cgroup に保持することもできます。このオプションを設定せず、**cgred** サービスが実行中の場合、子プロセスは **/etc/cgrules.conf** にある設定に基づいて cgroup に割り当てられます。ただし、プロセス自体は、それを開始した 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、制御グループ、およびそれらのパラメーターに関する情報を取得するためのいくつか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

カーネルで使用可能なコントローラーと、それらが階層に一緒にマウントされる方法に関する情報を見つけるには、次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見つけるには、次のコマンドを実行します。

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

ここで *controllers* は、関心のあるサブシステムのリストを表します。**lssubsys -m** コマンドは、各階層ごとに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

システム上の cgroup を一覧表示するには、**root** として実行します。

```
~]# 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*** の形式で指定します。以下に例を示します。

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

上記のコマンドは、**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、次を実行します。

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

ここで、*パラメーター* はコントローラーの値を含む疑似ファイルであり、*list_of_cgroups* はスペースで区切られた cgroup のリストです。

実際のパラメーターの名前がわからない場合は、次のようなコマンドを使用します。

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

3.9. 関連情報

cgroup コマンドの完全なドキュメントは、libcgroup パッケージで提供されるマニュアルページにあります。

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 cgroup に移動するために使用されます。
- **cgclear** (1) – **cgclear** コマンドは、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、階層内に新しい 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 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 cgroup にいつ属するかを決定するために使用されるルールが含まれています。
- **cgrulesengd** (8) – **cgrulesengd** サービスはタスクを cgroup に分散します。
- **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リストします。

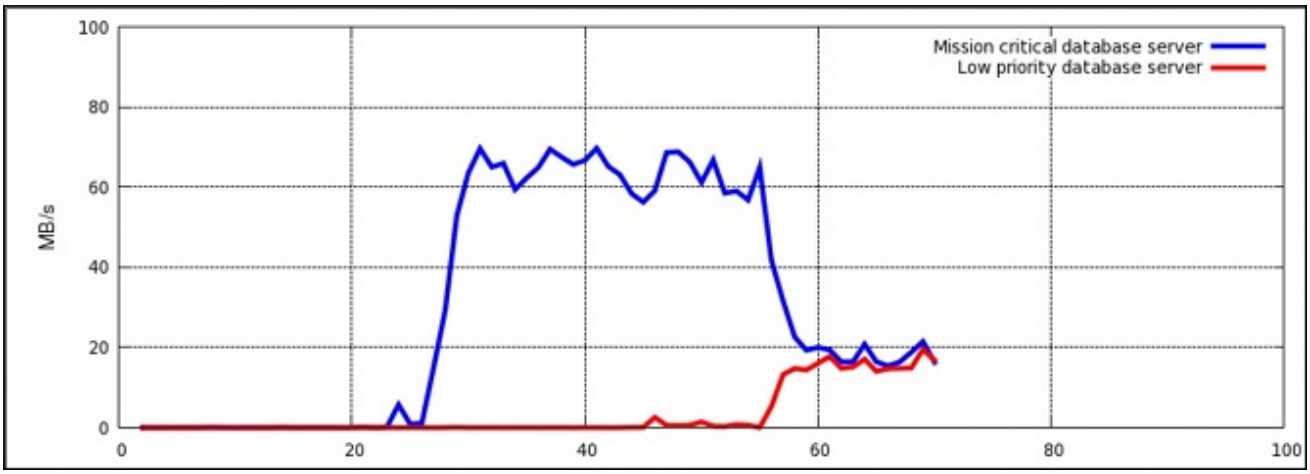
第4章 コントロールグループの適用例

この章では、cgroup 機能を利用するアプリケーションの例を示します。

4.1. データベース I/O の優先順位付け

専用の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、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。次の例を考えてみましょう。システムは、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ます。データベースの 1 つは優先度の高いデータベースで、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を同時に実行すると、I/O スループットが低下し、両方のデータベースからの要求を均等に処理できます。図 4.1「リソース割り当てなしの 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開始されると (時間 45 頃)、I/O スループットは両方のデータベースサーバーで同じになります。

図4.1 リソース割り当てなしの 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に優先順位を付けるには、予約済み I/O 操作の数が多い cgroup に割り当てることができます。一方、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。これを行うには、次の手順に従います。手順4.1「I/O スループットの優先順位付け」、これらはすべてホストシステムで実行されます。

手順4.1 I/O スループットの優先順位付け

1. 両方のサービスでリソースアカウンティングがオンになっていることを確認します。

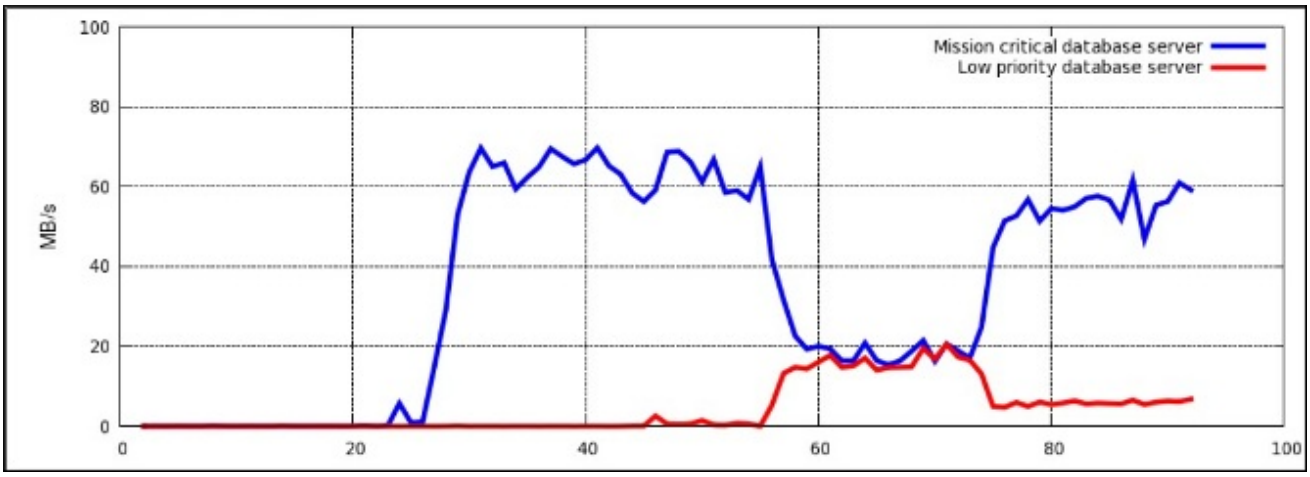
```
~]# systemctl set-property db1.service BlockIOAccounting=true
~]# systemctl set-property db2.service BlockIOAccounting=true
```

2. 高優先度サービスと低優先度サービスの比率を 10:1 に設定します。これらのサービスユニットで実行されているプロセスは、使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「リソース割り当てによる I/O スループット」 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 75 前後)、I/O スループットは両方のサーバー間で 10:1 の比率で分割されます。

図4.2 リソース割り当てによる I/O スループット



[D]

または、ブロックデバイス I/O スロットリングを優先度の低いデータベースに使用して、読み取り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 .

4.2. ネットワークトラフィックの優先順位付け

単一のサーバーシステムで複数のネットワーク関連サービスを実行する場合、これらのサービス間でネットワークの優先順位を定義することが重要です。優先度を定義すると、特定のサービスから発信されたパケットが、他のサービスから発信されたパケットよりも高い優先度を持つようになります。たとえば、このような優先度は、サーバーシステムが 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。ユーザーは高いスループットを期待するため、NFS トラフィックの優先順位を高くする必要があります。NFS サーバーのパフォーマンスを向上させるために、Samba トラフィックの優先順位を下げることができます。

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優先度を設定できます。次に、これらの優先順位は Type of Service (ToS) フィールドビットに変換され、すべてのパケットに埋め込まれます。の手順に従います。手順4.2「ファイル共有サービスのネットワーク優先順位の設定」 2 つのファイル共有サービス (NFS と Samba) の優先順位を設定します。

手順4.2 ファイル共有サービスのネットワーク優先順位の設定

1. **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
```

2. サービスごとに 1 つずつ、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
```

3. **nfs** サービスを **nfs_high** cgroup に自動的に移動するには、次の行を **/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** サービスが開始または再起動されたときに、**nfs** サービスプロセスが確実に **nfs_high** cgroup に移動されます。

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 **samba_low** cgroup に自動的に移動するには、次の行を **/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 に移動することに注意してください。

同様の方法で、**nmbd** サービスと **winbindd** サービスを **samba_low** cgroup に移動するためのルールを定義できます。

5. **cgred** サービスを開始して、前の手順の設定をロードします。

```
~]# 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イスを使用すると仮定します。各 cgroup のネットワーク優先度を定義します。**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
```

7. **nfs** および **smb** サービスを開始し、それらのプロセスが正しい cgroup に移動されている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 から発信されたネットワークトラフィックは、Samba から発信されたトラフィックよりも優先度が高くなりました。

に似ている手順4.2「ファイル共有サービスのネットワーク優先順位の設定」、**net_prio** サブシステムを使用して、Firefox などのクライアントアプリケーションのネットワーク優先度を設定できます。

付録A 更新履歴

改訂 0.0-1.10 Mon Aug 05 2019 Marie Doleželová
7.0 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 コンテナーは別の本に移動しました。

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