Red Hat Enterprise Linux 7.3 Beta Resource Management Guide

Managing system resources on Red Hat Enterprise Linux 7

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

Red Hat Enterprise Linux 7.3 Beta Resource Management Guide

Managing system resources on Red Hat Enterprise Linux 7

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

Legal Notice

Abstract

Managing system resources on Red Hat Enterprise Linux 7. Note: This document is under development, is subject to substantial change, and is provided only as a preview. The included information and instructions should not be considered complete, and should be used with caution.

Table of Contents

Chapter 1. Introduction to Control Groups (Cgroups)

1.1. What are Control Groups

The *control groups*, abbreviated as *cgroups* in this guide, are a Linux kernel feature that allows you to allocate resources — such as CPU time, system memory, network bandwidth, or combinations of these resources — among hierarchically ordered groups of processes running on a system. By using cgroups, system administrators gain fine-grained control over allocating, prioritizing, denying, managing, and monitoring system resources. Hardware resources can be smartly divided up among applications and users, increasing overall efficiency.

Control Groups provide a way to hierarchically group and label processes, and to apply resource limits to them. Traditionally, all processes received similar amounts of system resources that the administrator could modulate with the process *niceness* value. With this approach, applications that involved a large number of processes received more resources than applications with few processes, regardless of the relative importance of these applications.

Red Hat Enterprise Linux 7 moves the resource management settings from the process level to the application level by binding the system of cgroup hierarchies with the systemd unit tree. Therefore, you can manage system resources with **systemctl** commands, or by modifying systemd unit files. See Chapter 2, *Using Control Groups* for details.

In previous versions of Red Hat Enterprise Linux, system administrators built custom cgroup hierarchies with the use of the **cgconfig** command from the *libcgroup* package. This package is now deprecated, and it is not recommended to use it since it can easily create conflicts with the default cgroup hierarchy. However, *libcgroup* is still available to cover for certain specific cases, where **systemd** is not yet applicable, most notably for using the *net-prio* subsystem. See Chapter 3, *Using libcgroup Tools*.

The aforementioned tools provide a high-level interface to interact with cgroup controllers (also known as subsystems) in Linux kernel. The main cgroup controllers for resource management are *cpu*, *memory*, and *blkio*, see Available Controllers in Red Hat Enterprise Linux 7 for the list of controllers enabled by default. For detailed description of resource controllers and their configurable parameters, refer to Controller-Specific Kernel Documentation.

1.2. Default Cgroup Hierarchies

By default, **systemd** automatically creates a hierarchy of *slice*, *scope* and *service* units to provide a unified structure for the cgroup tree. With the **systemctl** command, you can further modify this structure by creating custom slices, as shown in Section 2.1, "Creating Control Groups". Also, **systemd** automatically mounts hierarchies for important kernel resource controllers (see Available Controllers in Red Hat Enterprise Linux 7) in the **/sys/fs/cgroup/** directory.

Warning

The deprecated **cgconfig** tool from the **libcgroup** package is available to mount and handle hierarchies for controllers not yet supported by **systemd** (most notably the **net-prio** controller). Never use **libcgropup** tools to modify the default hierarchies mounted by **systemd** since it would lead to unexpected behavior. The **libcgroup** library will be removed in future versions of Red Hat Enterprise Linux. For more information on how to use **cgconfig**, see Chapter 3, *Using libcgroup Tools*.

Systemd Unit Types

All processes running on the system are child processes of the **systemd** init process. Systemd provides three unit types that are used for the purpose of resource control (for a complete list of **systemd**'s unit types, see the chapter called *Managing Services with systemd* in Red Hat Enterprise Linux 7 System Administrator's Guide):

- **Service** — A process or a group of processes, which **systemd** started based on a unit configuration file. Services encapsulate the specified processes so that they can be started and stopped as one set. Services are named in the following way:

  ```
  name.service
  ```

  Where *name* stands for the name of the service.

- **Scope** — A group of externally created processes. Scopes encapsulate processes that are started and stopped by arbitrary processes via the **fork()** function and then registered by **systemd** at runtime. For instance, user sessions, containers, and virtual machines are treated as scopes. Scopes are named as follows:

  ```
  name.scope
  ```

  Here, *name* stands for the name of the scope.

- **Slice** — A group of hierarchically organized units. Slices do not contain processes, they organize a hierarchy in which scopes and services are placed. The actual processes are contained in scopes or in services. In this hierarchical tree, every name of a slice unit corresponds to the path to a location in the hierarchy. The dash ("**-**") character acts as a separator of the path components. For example, if the name of a slice looks as follows:

  ```
  parent-name.slice
  ```

  it means that a slice called *parent-name*.**slice** is a subslice of the *parent*.**slice**. This slice can have its own subslice named *parent-name-name2*.**slice**, and so on.

  There is one root slice denoted as:

  ```
  -.slice
  ```

Service, scope, and slice units directly map to objects in the cgroup tree. When these units are activated, they map directly to cgroup paths built from the unit names. For example, the *ex.service* residing in the *test-waldo.slice* is mapped to the cgroup **test.slice/test-waldo.slice/ex.service/**.

Services, scopes, and slices are created manually by the system administrator or dynamically by programs. By default, the operating system defines a number of built-in services that are necessary to run the system. Also, there are four slices created by default:

- **-.slice** — the root slice;
- **system.slice** — the default place for all system services;
- **user.slice** — the default place for all user sessions;
- **machine.slice** — the default place for all virtual machines and Linux containers.

Note that all user sessions are automatically placed in a separated scope unit, as well as virtual machines and container processes. Furthermore, all users are assigned with an implicit subslice. Besides the above default configuration, the system administrator can define new slices and assign services and scopes to them.

The following tree is a simplified example of a cgroup tree. This output was generated with the **systemd-cgls** command described in Section 2.4, “Obtaining Information about Control Groups”:

```
├─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
  │
  ...
```

As you can see, services and scopes contain processes and are placed in slices that do not contain processes of their own. The only exception is PID 1 that is located in the special **systemd.slice**. Also note that **-.slice** is not shown as it is implicitly identified with the root of the entire tree.

Service and slice units can be configured with persistent unit files as described in Section 2.3.2, “Modifying Unit Files”, or created dynamically at runtime via API calls to PID 1 (see Section 1.4, “Online Documentation” for API reference). Scope units can be created only by the first method. Units created dynamically with API calls are transient and exist only during runtime. Transient units are released automatically as soon as they finish, get deactivated, or the system is rebooted.

1.3. Resource Controllers in Linux Kernel

A resource controller, also called a cgroup subsystem, represents a single resource, such as CPU time or memory. The Linux kernel provides a range of resource controllers, that are mounted automatically by **systemd**. Find the list of currently mounted resource controllers in **/proc/cgroups**, or use the **lssubsys** monitoring tool. In Red Hat Enterprise Linux 7, **systemd** mounts the following controllers by default:

Available Controllers in Red Hat Enterprise Linux 7

- **blkio** — sets limits on input/output access to and from block devices;

- **`cpu`** — uses the CPU scheduler to provide cgroup tasks access to the CPU. It is mounted together with the **`cpuacct`** controller on the same mount;
- **`cpuacct`** — creates automatic reports on CPU resources used by tasks in a cgroup. It is mounted together with the **`cpu`** controller on the same mount;
- **`cpuset`** — assigns individual CPUs (on a multicore system) and memory nodes to tasks in a cgroup;
- **`devices`** — allows or denies access to devices for tasks in a cgroup;
- **`freezer`** — suspends or resumes tasks in a cgroup;
- **`memory`** — sets limits on memory use by tasks in a cgroup and generates automatic reports on memory resources used by those tasks;
- **`net_cls`** — tags network packets with a class identifier (**`classid`**) that allows the Linux traffic controller (the **`tc`** command) to identify packets originating from a particular cgroup task. A subsystem of **`net_cls`**, the **`net_filter`** (iptables) can also use this tag to perform actions on such packets. The **`net_filter`** tags network sockets with a firewall identifier (**`fwid`**) that allows the Linux firewall (the **`iptables`** command) to identify packets (skb->sk) originating from a particular cgroup task;
- **`perf_event`** — enables monitoring cgroups with the **`perf`** tool;
- **`hugetlb`** — allows to use virtual memory pages of large sizes and to enforce resource limits on these pages.

The Linux kernel exposes a wide range of tunable parameters for resource controllers that can be configured with **`systemd`**. See the kernel documentation (list of references in the Controller-Specific Kernel Documentation section) for detailed description of these parameters.

1.4. Additional Resources

To find more information about resource control under **`systemd`**, the unit hierarchy, as well as the kernel resource controllers, refer to the materials listed below:

Installed Documentation

Cgroup-Related Systemd Documentation

The following manual pages contain general information on unified cgroup hierarchy under **`systemd`**:

- **`systemd.resource-control`**(5) — describes the configuration options for resource control shared by system units.
- **`systemd.unit`**(5) — describes common options of all unit configuration files.
- **`systemd.slice`**(5) — provides general information about *.slice* units.
- **`systemd.scope`**(5) — provides general information about *.scope* units.
- **`systemd.service`**(5) — provides general information about *.service* units.

Controller-Specific Kernel Documentation

The *kernel-doc* package provides a detailed documentation of all resource controllers. This package is included in the Optional subscription channel. Before subscribing to the Optional channel, please see the Scope of Coverage Details for Optional software, then follow the steps documented in the article called How to access Optional and Supplementary channels, and -devel packages using Red Hat Subscription Manager (RHSM)? on Red Hat Customer Portal. To install *kernel-doc* from the Optional channel, type as **root**:

```
yum install kernel-doc
```

After the installation, the following files will appear under the **/usr/share/doc/kernel-doc-*<kernel_version>*/Documentation/cgroups/** directory:

- **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**

Additionally, refer to the following files on further information about the **cpu** subsystem:

- Real-Time scheduling — **/usr/share/doc/kernel-doc-*<kernel_version>*/Documentation/scheduler/sched-rt-group.txt**
- CFS scheduling — **/usr/share/doc/kernel-doc-*<kernel_version>*/Documentation/scheduler/sched-bwc.txt**

Online Documentation

- Red Hat Enterprise Linux 7 System Administrator's Guide — The *System Administrator's Guide* documents relevant information regarding the deployment, configuration, and administration of Red Hat Enterprise Linux 7. This guide contains a detailed explanation of the **systemd** concepts as well as instructions for service management with **systemd**.
- The D-Bus API of systemd — The reference material for D-Bus API commands used to interact with **systemd**.

Chapter 2. Using Control Groups

The following sections provide an overview of tasks related to creation and management of control groups. This guide focuses on utilities provided by **systemd** that are preferred as a way of cgroup management and will be supported in the future. Previous versions of Red Hat Enterprise Linux used the *libcgroup* package for creating and managing cgroups. This package is still available to assure backward compatibility (see Warning), but it will not be supported in future versions of Red Hat Enterprise Linux.

2.1. Creating Control Groups

From the **systemd**'s perspective, a cgroup is bound to a system unit configurable with a unit file and manageable with systemd's command-line utilities. Depending on the type of application, your resource management settings can be *transient* or *persistent*.

To create a **transient cgroup** for a service, start the service with the **systemd-run** command. This way, it is possible to set limits on resources consumed by the service during its runtime. Applications can create transient cgroups dynamically by using API calls to **systemd**. See Section 2.5, "Online Documentation" for API reference. Transient unit is removed automatically as soon as the service is stopped.

To assign a **persistent cgroup** to a service, edit its unit configuration file. The configuration is preserved after the system reboot, so it can be used to manage services that are started automatically. Note that scope units can not be created in this way.

2.1.1. Creating Transient Cgroups with systemd-run

The **systemd-run** command is used to create and start a transient *service* or *scope* unit and run a custom command in the unit. Commands executed in service units are started asynchronously in the background, where they are invoked from the **systemd** process. Commands run in scope units are started directly from the **systemd-run** process and thus inherit the execution environment of the caller. Execution in this case is synchronous.

To run a command in a specified cgroup, type as **root**:

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

- The *name* stands for the name you want the unit to be known under. If **--unit** is not specified, a unit name will be generated automatically. It is recommended to choose a descriptive name, since it will represent the unit in the **systemctl** output. The name has to be unique during runtime of the unit.
- Use the optional **--scope** parameter to create a transient *scope* unit instead of *service* unit that is created by default.
- With the **--slice** option, you can make your newly created *service* or *scope* unit a member of a specified slice. Replace *slice_name* with the name of an existing slice (as shown in the output of **systemctl -t slice**), or create a new slice by passing a unique name. By default, services and scopes are created as members of the **system.slice**.
- Replace *command* with the command you wish to execute in the service unit. Place this command at the very end of the **systemd-run** syntax, so that the parameters of this command are not confused for parameters of **systemd-run**.

Besides the above options, there are several other parameters available for **systemd-run**. For example, **--description** creates a description of the unit, **--remain-after-exit** allows to collect runtime information after terminating the service's process. The **--machine** option executes the command in a confined container. See the **systemd-run**(1) manual page to learn more.

> **Example 2.1. Starting a New Service with systemd-run**
>
> Use the following command to run the **top** utility in a service unit in a new slice called **test**. Type as **root**:
>
> ```
> ~]# systemd-run --unit=toptest --slice=test top -b
> ```
>
> The following message is displayed to confirm that you started the service successfully:
>
> ```
> Running as unit toptest.service
> ```
>
> Now, the name *toptest.service* can be used to monitor or to modify the cgroup with **systemctl** commands.

2.1.2. Creating Persistent Cgroups

To configure a unit to be started automatically on system boot, execute the **systemctl enable** command (see the chapter called *Managing Services with systemd* in Red Hat Enterprise Linux 7 System Administrators Guide). Running this command automatically creates a unit file in the **/usr/lib/systemd/system/** directory. To make persistent changes to the cgroup, add or modify configuration parameters in its unit file. For more information, see Section 2.3.2, “Modifying Unit Files”.

2.2. Removing Control Groups

Transient cgroups are released automatically as soon as the processes they contain finish. By passing the **--remain-after-exit** option to **systemd-run** you can keep the unit running after its processes finished to collect runtime information. To stop the unit gracefully, type:

```
systemctl stop name.service
```

Replace *name* with the name of the service you wish to stop. To terminate one or more of the unit's processes, type as **root**:

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

Replace *name* with a name of the unit, for example *httpd.service*. Use **--kill-who** to select which processes from the cgroup you wish to terminate. To kill multiple processes at the same time, pass a comma-separated list of PIDs. Replace *signal* with the type of POSIX signal you wish to send to specified processes. Default is *SIGTERM*. For more information, see the **systemd.kill** manual page.

Persistent cgroups are released when the unit is disabled and its configuration file is deleted by running:

```
systemctl disable name.service
```

where *name* stands for the name of the service to be disabled.

2.3. Modifying Control Groups

Each persistent unit supervised by **`systemd`** has a unit configuration file in the **`/usr/lib/systemd/system/`** directory. To change parameters of a service unit, modify this configuration file. This can be done either manually or from the command-line interface by using the **`systemctl set-property`** command.

2.3.1. Setting Parameters from the Command-Line Interface

The **`systemctl set-property`** command allows you to persistently change resource control settings during the application runtime. To do so, use the following syntax as **`root`**:

```
systemctl set-property name parameter=value
```

Replace *name* with the name of the systemd unit you wish to modify, *parameter* with a name of the parameter to be changed, and *value* with a new value you want to assign to this parameter.

Not all unit parameters can be changed at runtime, but most of those related to resource control may, see Section 2.3.2, “Modifying Unit Files” for a complete list. Note that **`systemctl set-property`** allows you to change multiple properties at once, which is preferable over setting them individually.

The changes are applied instantly, and written into the unit file so that they are preserved after reboot. You can change this behavior by passing the **`--runtime`** option that makes your settings transient:

```
systemctl set-property --runtime name property=value
```

Example 2.2. Using systemctl set-property

To limit the CPU and memory usage of *httpd.service* from the command line, type:

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

To make this a temporary change, add the **`--runtime`** option:

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

2.3.2. Modifying Unit Files

Systemd service unit files provide a number of high-level configuration parameters useful for resource management. These parameters communicate with Linux cgroup controllers, that have to be enabled in the kernel. With these parameters, you can manage CPU, memory consumption, block IO, as well as some more fine-grained unit properties.

Managing CPU

The *cpu* controller is enabled by default in the kernel, and consequently every system service receives the same amount of CPU time, regardless of how many processes it contains. This default behavior can be changed with the **`DefaultControllers`** parameter in the **`/etc/systemd/system.conf`** configuration file. To manage CPU allocation, use the following directive in the **`[Service]`** section of the unit configuration file:

`CPUShares=`*value*

Replace *value* with a number of CPU shares. The default value is 1024. By increasing the number, you assign more CPU time to the unit. This parameter implies that **`CPUAccounting`** is turned on in the unit file.

The **`CPUShares`** parameter controls the *cpu.shares* control group parameter. See the description of the **`cpu`** controller in Controller-Specific Kernel Documentation to see other CPU-related control parameters.

Example 2.3. Limiting CPU Consumption of a Unit

To assign the Apache service 1500 CPU shares instead of the default 1024, modify the **`CPUShares`** setting in the **`/usr/lib/systemd/system/httpd.service`** unit file:

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

To apply the changes, reload systemd's configuration and restart Apache so that the modified service file is taken into account:

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

Managing Memory

To enforce limits on the unit's memory consumption, use the following directives in the **`[Service]`** section of the unit configuration file:

`MemoryLimit=`*value*

Replace *value* with a limit on maximum memory usage of the processes executed in the cgroup. Use suffixes *K*, *M*, *G*, or *T* to identify Kilobyte, Megabyte, Gigabyte, or Terabyte as the unit of measurement. Also, the **`MemoryAccounting`** parameter has to be enabled for the unit.

The **`MemoryLimit`** parameter controls the *memory.limit_in_bytes* control group parameter. For more information, see the description of the **`memory`** controller in Controller-Specific Kernel Documentation.

Example 2.4. Limiting Memory Consumption of a Unit

To assign a 1GB memory limit to the Apache service, modify the **`MemoryLimit`** setting in the **`/usr/lib/systemd/system/httpd.service`** unit file:

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

To apply the changes, reload systemd's configuration and restart Apache so that the modified service file is taken into account:

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

Managing Block IO

To manage the Block IO, use the following directives in the **`[Service]`** section of the unit configuration file. Directives listed below assume that the **`BlockIOAccounting`** parameter is enabled:

`BlockIOWeight=`*value*

Replace *value* with a new overall block IO weight for the executed processes. Choose a single value between 10 and 1000, the default setting is 1000.

`BlockIODeviceWeight=`*device_name value*

Replace *value* with a block IO weight for a device specified with *device_name*. Replace *device_name* either with a name or with a path to a device. As with **`BlockIOWeight`**, it is possible to set a single weight value between 10 and 1000.

`BlockIOReadBandwidth=`*device_name value*

This directive allows you to limit a specific bandwidth for a unit. Replace *device_name* with the name of a device or with a path to a block device node, *value* stands for a bandwidth rate. Use suffixes *K*, *M*, *G*, or *T* to specify units of measurement. A value with no suffix is interpreted as bytes per second.

`BlockIOWriteBandwidth=`*device_name value*

Limits the write bandwidth for a specified device. Accepts the same arguments as **`BlockIOReadBandwidth`**.

Each of the aforementioned directives controls a corresponding cgroup parameter. See the description of the **`blkio`** controller in Controller-Specific Kernel Documentation.

Note

Currently, the **`blkio`** resource controller does not support buffered write operations. It is primarily targeted at direct I/O, so the services that use buffered write will ignore the limits set with **`BlockIOWriteBandwidth`**. On the other hand, buffered read operations are supported, and **`BlockIOReadBandwidth`** limits will be applied correctly both on direct and buffered read.

Example 2.5. Limiting Block IO of a Unit

To lower the block IO weight for the Apache service accessing the **`/home/jdoe/`** directory, add the following text into the **`/usr/lib/systemd/system/httpd.service`** unit file:

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

To set the maximum bandwidth for Apache reading from the **/var/log/** directory to 5MB per second, use the following syntax:

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

To apply your changes, reload systemd's configuration and restart Apache so that the modified service file is taken into account:

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

Managing Other System Resources

There are several other directives that can be used in the unit file to facilitate resource management:

DeviceAllow=*device_name options*

This option controls access to specific device nodes. Here, *device_name* stands for a path to a device node or a device group name as specified in **/proc/devices**. Replace **options** with a combination of **r**, **w**, and **m** to allow the unit to read, write, or create device nodes.

DevicePolicy=*value*

Here, *value* is one of: *strict* (only allows the types of access explicitly specified with **DeviceAllow**), *closed* (allows access to standard pseudo devices including /dev/null, /dev/zero, /dev/full, /dev/random, and /dev/urandom) or *auto* (allows access to all devices if no explicit **DeviceAllow** is present, which is the default behavior)

Slice=*slice_name*

Replace *slice_name* with the name of the slice to place the unit in. The default is *system.slice*. Scope units can not be arranged in this way, since they are tied to their parent slices.

ExecStartPost=*command*

Currently, **systemd** supports only a subset of cgroup features. However, as a workaround, you can use the **ExecStartPost=** option along with setting the ***memory.memsw.limit_in_bytes*** parameter in order to prevent any swap usage for a service. For more information on **ExecStartPost=**, see the **systemd.service(5)** man page.

Example 2.6. Configuring Cgroup Options

Imagine that you wish to change the ***memory.memsw.limit_in_bytes*** setting to the same value as the unit's **MemoryLimit=** in order to prevent any swap usage for a given *example* service.

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

To apply the change, reload **systemd** configuration and restart the service so that the modified setting is taken into account:

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

2.4. Obtaining Information about Control Groups

Use the **systemctl** command to list system units and to view their status. Also, the **systemd-cgls** command is provided to view the hierarchy of control groups and **systemd-cgtop** to monitor their resource consumption in real time.

2.4.1. Listing Units

Use the following command to list all active units on the system:

```
systemctl list-units
```

The **list-units** option is executed by default, which means that you will receive the same output when you omit this option and execute just:

```
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
...
```

The output displayed above contains five columns:

- *UNIT* — the name of the unit that also reflects the unit's position in the cgroup tree. As mentioned in Section 1.2, “Systemd Unit Types”, three unit types are relevant for resource control: *slice*, *scope*, and *service*. For a complete list of **systemd**'s unit types, see the chapter called *Managing Services with systemd* in Red Hat Enterprise Linux 7 System Administrators Guide.
- *LOAD* — indicates whether the unit configuration file was properly loaded. If the unit file failed to load, the field contains the state *error* instead of *loaded*. Other unit load states are: *stub*, *merged*, and *masked*.
- *ACTIVE* — the high-level unit activation state, which is a generalization of SUB.
- *SUB* — the low-level unit activation state. The range of possible values depends on the unit type.
- *DESCRIPTION* — the description of the unit's content and functionality.

By default, **systemctl** lists only active units (in terms of high-level activations state in the ACTIVE field). Use the **--all** option to see inactive units too. To limit the amount of information in the output list, use the **--type** (**-t**) parameter that requires a comma-separated list of unit types such as *service* and *slice*, or unit load states such as *loaded* and *masked*.

Example 2.7. Using systemctl list-units

To view a list of all slices used on the system, type:

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

To list all active masked services, type:

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

To list all unit files installed on your system and their status, type:

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

2.4.2. Viewing the Control Group Hierarchy

The aforementioned listing commands do not go beyond the unit level to show the actual processes running in cgroups. Also, the output of **`systemctl`** does not show the hierarchy of units. You can achieve both by using the **`systemd-cgls`** command that groups the running process according to cgroups. To display the whole cgroup hierarchy on your system, type:

```
systemd-cgls
```

When **`systemd-cgls`** is issued without parameters, it returns the entire cgroup hierarchy. The highest level of the cgroup tree is formed by slices and can look as follows:

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

Note that machine slice is present only if you are running a virtual machine or a container. For more information on the cgroup tree, see Section 1.2, “Systemd Unit Types”.

To reduce the output of **`systemd-cgls`**, and to view a specified part of the hierarchy, execute:

```
systemd-cgls name
```

Replace *name* with a name of the resource controller you want to inspect.

As an alternative, use the **`systemctl status`** command to display detailed information about a system unit. A cgroup subtree is a part of the output of this command.

```
systemctl status name
```

To learn more about **systemctl status**, see the chapter called *Managing Services with systemd* in Red Hat Enterprise Linux 7 System Administrators Guide.

Example 2.8. Viewing the Control Group Hierarchy

To see a cgroup tree of the **memory** resource controller, execute:

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

The output of the above command lists the services that interact with the selected controller. A different approach is to view a part of the cgroup tree for a certain service, slice, or scope unit:

```
~]# 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

...
```

Besides the aforementioned tools, **systemd** also provides the **machinectl** command dedicated to monitoring Linux containers.

2.4.3. Viewing Resource Controllers

The aforementioned **systemctl** commands enable monitoring the higher-level unit hierarchy, but do not show which resource controllers in Linux kernel are actually used by which processes. This information is stored in dedicated process files, to view it, type as **root**:

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

Where *PID* stands for the ID of the process you wish to examine. By default, the list is the same for all units started by **systemd**, since it automatically mounts all default controllers. See the following example:

```
~]# 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:/
```

By examining this file, you can determine if the process has been placed in the desired cgroups as defined by the systemd unit file specifications.

2.4.4. Monitoring Resource Consumption

The **systemd-cgls** command provides a static snapshot of the cgroup hierarchy. To see a dynamic account of currently running cgroups ordered by their resource usage (CPU, Memory, and IO), use:

```
systemd-cgtop
```

The behavior, provided statistics, and control options of **systemd-cgtop** are akin of those of the **top** utility. See **systemd-cgtop**(1) manual page for more information.

2.5. Additional Resources

For more information on how to use **systemd** and related tools to manage system resources on Red Hat Enterprise Linux, refer to the sources listed below:

Installed Documentation

Man Pages of Cgroup-Related Systemd Tools

- **systemd-run**(1) — The manual page lists all command-line options of the **systemd-run** utility.
- **systemctl**(1) — The manual page of the **systemctl** utility that lists available options and commands.
- **systemd-cgls**(1) — This manual page lists all command-line options of the **systemd-cgls** utility.
- **systemd-cgtop**(1) — The manual page contains the list of all command-line options of the **systemd-cgtop** utility.
- **machinectl**(1) — This manual page lists all command-line options of the **machinectl** utility.
- **systemd.kill**(5) — This manual page provides an overview of kill configuration options for system units.

Controller-Specific Kernel Documentation

The *kernel-doc* package provides detailed documentation of all resource controllers. This package is included in the Optional subscription channel. Before subscribing to the Optional channel, please see the Scope of Coverage Details channel, then follow the steps documented in the article called How to access Optional and Supplementary channels, and -devel packages using Red Hat Subscription Manager (RHSM)? on Red Hat Customer Portal. To install *kernel-doc* from the Optional channel, type as **`root`**:

```
yum install kernel-doc
```

After the installation, the following files will appear under the **`/usr/share/doc/kernel-doc-<kernel_version>/Documentation/cgroups/`** directory:

- **`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`**

Additionally, refer to the following files on further information about the **`cpu`** subsystem:

- Real-Time scheduling — **`/usr/share/doc/kernel-doc-<kernel_version>/Documentation/scheduler/sched-rt-group.txt`**
- CFS scheduling — **`/usr/share/doc/kernel-doc-<kernel_version>/Documentation/scheduler/sched-bwc.txt`**

Online Documentation

- Red Hat Enterprise Linux 7 System Administrators Guide — The *System Administrator's Guide* documents relevant information regarding the deployment, configuration and administration of Red Hat Enterprise Linux 7. It is oriented towards system administrators with a basic understanding of the system.
- The D-Bus API of systemd — The reference for D-Bus API commands for accessing **`systemd`**.

Chapter 3. Using libcgroup Tools

The *libcgroup* package, which was the main tool for cgroup management in previous versions of Red Hat Enterprise Linux, is now deprecated. To avoid conflicts, do not use *libcgroup* tools for default resource controllers (listed in Available Controllers in Red Hat Enterprise Linux 7) that are now an exclusive domain of **systemd**. This leaves a limited space for applying *libcgroup* tools, use it only when you need to manage controllers not currently supported by **systemd**, such as *net_prio*.

The following sections describe how to use *libcgroup* tools in relevant scenarios without conflicting with the default system of hierarchy.

Note

In order to use *libcgroup* tools, first ensure the *libcgroup* and *libcgroup-tools* packages are installed on your system. To install them, run as **root**:

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

Note

The **net_prio** controller is not compiled in the kernel like the rest of the controllers, rather it is a module that has to be loaded before attempting to mount it. To load this module, type as **root**:

```
modprobe netprio_cgroup
```

3.1. Mounting a Hierarchy

To use a kernel resource controller that is not mounted automatically, you have to create a hierarchy that will contain this controller. Add or detach the hierarchy by editing the **mount** section of the **/etc/cgconfig.conf** configuration file. This method makes the controller attachment persistent, which means your settings will be preserved after system reboot. As an alternative, use the **mount** command to create a transient mount only for the current session.

Using the cgconfig Service

The **cgconfig** service installed with the *libcgroup-tools* package provides a way to mount hierarchies for additional resource controllers. By default, this service is not started automatically. When you start **cgconfig**, it applies the settings from the **/etc/cgconfig.conf** configuration file. The configuration is therefore recreated from session to session and becomes persistent. Note that if you stop **cgconfig**, it unmounts all the hierarchies that it mounted.

The default **/etc/cgconfig.conf** file installed with the *libcgroup* package does not contain any configuration settings, only information that **systemd** mounts the main resource controllers automatically.

Entries of three types can be created in **/etc/cgconfig.conf** — *mount*, *group*, and *template*. Mount entries are used to create and mount hierarchies as virtual file systems, and attach controllers to those hierarchies. In Red Hat Enterprise Linux 7, default hierarchies are mounted automatically to the **/sys/fs/cgroup/** directory, **cgconfig** is therefore used solely to attach non-default controllers. Mount entries are defined using the following syntax:

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

Replace *controller_name* with a name of the kernel resource controller you wish to mount to the hierarchy. See Example 3.1, “Creating a mount entry” for an example.

Example 3.1. Creating a mount entry

To attach the **net_prio** controller to the default cgroup tree, add the following text to the **/etc/cgconfig.conf** configuration file:

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

Then restart the **cgconfig** service to apply the setting:

```
systemctl restart cgconfig.service
```

Group entries in **/etc/cgconfig.conf** can be used to set the parameters of resource controllers. See Section 3.5, “Setting Cgroup Parameters” for more information about group entries.

Template entries in **/etc/cgconfig.conf** can be used to create a group definition applied to all processes.

Using the mount Command

Use the **mount** command to temporarily mount a hierarchy. To do so, first create a mount point in the **/sys/fs/cgroup/** directory where **systemd** mounts the main resource controllers. Type as **root**:

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

Replace *name* with a name of the new mount destination, usually the name of the controller is used. Next, execute the **mount** command to mount the hierarchy and simultaneously attach one or more subsystems. Type as **root**:

```
mount -t cgroup -o controller_name none /sys/fs/cgroup/controller_name
```

Replace *controller_name* with a name of the controller to specify both the device to be mounted as well as the destination folder. The **-t cgroup** parameter specifies the type of mount.

Example 3.2. Using the mount command to attach controllers

To mount a hierarchy for the **net_prio** controller with use of the **mount** command, first create the mount point:

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

Then mount **net_prio** to the destination you created in the previous step:

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

You can verify whether you attached the hierarchy correctly by listing all available hierarchies along with their current mount points using the **lssubsys** command (see Section 3.8, “Listing Controllers”):

```
~]# 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. Unmounting a Hierarchy

If you mounted a hierarchy by editing the **/etc/cgconfig.conf** configuration file, you can unmount it simply by removing the configuration directive from the *mount* section of this configuration file. Then restart the service to apply the new configuration.

Similarly, you can unmount a hierarchy by executing the following command as **root**:

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

Replace *controller_name* with the name of the hierarchy that contains the resource controller you wish to detach.

Warning

Make sure that you use **umount** to remove only hierarchies that you mounted yourself manually. Detaching a hierarchy that contains a default controller (listed in Available Controllers in Red Hat Enterprise Linux 7) will most probably lead to complications requiring a system reboot.

3.3. Creating Control Groups

Use the **cgcreate** command to create transient cgroups in hierarchies you created yourself. The syntax for **cgcreate** is:

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

where:

- **`-t`** (optional) — specifies a user (by user ID, uid) and a group (by group ID, gid) to own the **`tasks`** pseudo-file for this cgroup. This user can add tasks to the cgroup.

  Note

  Note that the only way to remove a process from a cgroup is to move it to a different cgroup. To be able to move a process, the user has to have write access to the *destination* cgroup; write access to the source cgroup is not necessary.

- **`-a`** (optional) — specifies a user (by user ID, uid) and a group (by group ID, gid) to own all pseudo-files other than **`tasks`** for this cgroup. This user can modify the access to system resources for tasks in this cgroup.

- **`-g`** — specifies the hierarchy in which the cgroup should be created, as a comma-separated list of the *controllers* associated with hierarchies. The list of controllers is followed by a colon and the *path* to the child group relative to the hierarchy. Do not include the hierarchy mount point in the path.

Because all cgroups in the same hierarchy have the same controllers, the child group has the same controllers as its parent.

As an alternative, you can create a child of the cgroup directly. To do so, use the **`mkdir`** command:

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

For example:

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

3.4. Removing Control Groups

Remove cgroups with the **`cgdelete`** command that has syntax similar to that of **`cgcreate`**. Run the following command as **`root`**:

```
cgdelete controllers:path
```

where:

- *controllers* is a comma-separated list of controllers.
- *path* is the path to the cgroup relative to the root of the hierarchy.

For example:

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

`cgdelete` can also recursively remove all subgroups when the **`-r`** option is specified.

Note that when you delete a cgroup, all its processes move to its parent group.

3.5. Setting Cgroup Parameters

Modify the parameters of the control groups by editing the **`/etc/cgconfig.conf`** configuration file, or by using the **`cgset`** command. Changes made to **`/etc/cgconfig.conf`** are preserved after reboot, while **`cgset`** changes the cgroup parameters only for the current session.

Modifying /etc/cgconfig.conf

You can set the controller parameters in the *Groups* section of **`/etc/cgconfig.conf`**. Group entries are defined using the following syntax:

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

Replace *name* with the name of your cgroup, *controller* stands for the name of the controller you wish to modify. You should modify only controllers you mounted yourself, not any of the default controllers mounted automatically by **`systemd`**. Replace *param_name* and *param_value* with the controller parameter you wish to change and its new value. Note that the **`permissions`** section is optional. To define permissions for a group entry, use the following syntax:

```
perm {
    task {
        uid = task_user;
        gid = task_group;
    }
    admin {
       uid = admin_name;
       gid = admin_group;
    }
}
```

Note

Restart the **`cgconfig`** service for the changes in the **`/etc/cgconfig.conf`** to take effect. Restarting this service rebuilds hierarchies specified in the configuration file but does not affect all mounted hierarchies. You can restart a service by executing the **`systemctl restart`** command, however, it is recommended to first stop the **`cgconfig`** service:

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

Then open and edit the configuration file. After saving your changes, you can start **`cgconfig`** again with the following command:

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

Using the cgset Command

Set controller parameters by running the **`cgset`** command from a user account with permission to modify the relevant cgroup. Use this only for controllers you mounted manually.

The syntax for **`cgset`** is:

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

where:

- *parameter* is the parameter to be set, which corresponds to the file in the directory of the given cgroup;
- *value* is the value for the parameter;
- *path_to_cgroup* is the path to the cgroup *relative to the root of the hierarchy*.

The values that can be set with **`cgset`** might depend on values set higher in a particular hierarchy. For example, if **`group1`** is limited to use only CPU 0 on a system, you cannot set **`group1/subgroup1`** to use CPUs 0 and 1, or to use only CPU 1.

It is also possible use **`cgset`** to copy the parameters of one cgroup into another, existing cgroup. The syntax to copy parameters with **`cgset`** is:

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

where:

- *path_to_source_cgroup* is the path to the cgroup whose parameters are to be copied, relative to the root group of the hierarchy;
- *path_to_target_cgroup* is the path to the destination cgroup, relative to the root group of the hierarchy.

3.6. Moving a Process to a Control Group

Move a process into a cgroup by running the **`cgclassify`** command:

```
cgclassify -g controllers:path_to_cgroup pidlist
```

where:

- *controllers* is a comma-separated list of resource controllers, or `*` to launch the process in the hierarchies associated with all available subsystems. Note that if there are multiple cgroups of the same name, the **`-g`** option moves the processes in each of those groups.
- *path_to_cgroup* is the path to the cgroup within the hierarchy;
- *pidlist* is a space-separated list of *process identifier* (PIDs).

If the **`-g`** option is not specified, **`cgclassify`** automatically searches **`/etc/cgrules.conf`** and uses the first applicable configuration line. According to this line, **`cgclassify`** determines the hierarchies and cgroups to move the process under. Note that for the move to be successful, the destination hierarchies have to exist. The subsystems specified in **`/etc/cgrules.conf`** has to be also properly configured for the corresponding hierarchy in **`/etc/cgconfig.conf`**.

You can also add the **`--sticky`** option before the *pid* to keep any child processes in the same cgroup. If you do not set this option and the **`cgred`** service is running, child processes will be allocated to cgroups based on the settings found in **`/etc/cgrules.conf`**. The process itself, however, will remain in the cgroup in which you started it.

It is also possible to use the **`cgred`** service (which starts the **`cgrulesengd`** service) that moves tasks into cgroups according to parameters set in the **`/etc/cgrules.conf`** file. Use **`cgred`** only to manage manually attached controllers. Entries in the **`/etc/cgrules.conf`** file can take one of the two forms:

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

For example:

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

This entry specifies that any processes that belong to the user named **`maria`** access the **`devices`** subsystem according to the parameters specified in the **`/usergroup/staff`** cgroup. To associate particular commands with particular cgroups, add the *command* parameter, as follows:

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

The entry now specifies that when the user named **`maria`** uses the **`ftp`** command, the process is automatically moved to the **`/usergroup/staff/ftp`** cgroup in the hierarchy that contains the **`devices`** subsystem. Note, however, that the daemon moves the process to the cgroup only after the appropriate condition is fulfilled. Therefore, the **`ftp`** process can run for a short time in an incorrect group. Furthermore, if the process quickly spawns children while in the incorrect group, these children might not be moved.

Entries in the **`/etc/cgrules.conf`** file can include the following extra notation:

- `@` — when prefixed to *user*, indicates a group instead of an individual user. For example, **`@admins`** are all users in the **`admins`** group.
- `*` — represents "all". For example, `*` in the **`subsystem`** field represents all subsystems.
- `%` — represents an item the same as the item on the line above. For example:

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

3.7. Starting a Process in a Control Group

Launch processes in a manually created cgroup by running the **`cgexec`** command. The syntax for **`cgexec`** is:

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

where:

- *controllers* is a comma-separated list of controllers, or **`*`** to launch the process in the hierarchies associated with all available subsystems. Note that, as with the **`cgset`** command described in Section 3.5, “Setting Cgroup Parameters”, if cgroups of the same name exist, the **`-g`** option creates processes in each of those groups.
- *path_to_cgroup* is the path to the cgroup relative to the hierarchy;
- *command* is the command to be executed in the cgroup;
- *arguments* are any arguments for the command.

It is also possible to add the **`--sticky`** option before the *command* to keep any child processes in the same cgroup. If you do not set this option and the **`cgred`** service is running, child processes will be allocated to cgroups based on the settings found in **`/etc/cgrules.conf`**. The process itself, however, will remain in the cgroup in which you started it.

3.8. Obtaining Information about Control Groups

The *libcgroup-tools* package contains several utilities for obtaining information about controllers, control groups, and their parameters.

Listing Controllers

To find the controllers that are available in your kernel and information on how they are mounted together to hierarchies, execute:

```
cat /proc/cgroups
```

Alternatively, to find the mount points of particular subsystems, execute the following command:

```
lssubsys -m controllers
```

Here *controllers* stands for a list of the subsystems in which you are interested. Note that the **`lssubsys -m`** command returns only the top-level mount point per each hierarchy.

Finding Control Groups

To list the cgroups on a system, execute as **`root`**:

```
lscgroup
```

To restrict the output to a specific hierarchy, specify a controller and a path in the format ***controller*:*path***. For example:

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

The above command lists only subgroups of the **adminusers** cgroup in the hierarchy to which the **cpuset** controller is attached.

Displaying Parameters of Control Groups

To display the parameters of specific cgroups, run:

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

where *parameter* is a pseudo-file that contains values for a controller, and *list_of_cgroups* is a list of cgroups separated with spaces.

If you do not know the names of the actual parameters, use a command similar to:

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

3.9. Additional Resources

The definitive documentation for cgroup commands can be found in the manual pages provided with the *libcgroup* package.

Installed Documentation

The libcgroup-related Man Pages

- **cgclassify**(1) — the **cgclassify** command is used to move running tasks to one or more cgroups.
- **cgclear**(1) — the **cgclear** command is used to delete all cgroups in a hierarchy.

  cgconfig.conf(5) — cgroups are defined in the **cgconfig.conf** file.
- **cgconfigparser**(8) — the **cgconfigparser** command parses the **cgconfig.conf** file and mounts hierarchies.
- **cgcreate**(1) — the **cgcreate** command creates new cgroups in hierarchies.
- **cgdelete**(1) — the **cgdelete** command removes specified cgroups.
- **cgexec**(1) — the **cgexec** command runs tasks in specified cgroups.
- **cgget**(1) — the **cgget** command displays cgroup parameters.
- **cgsnapshot**(1) — the **cgsnapshot** command generates a configuration file from existing subsystems.
- **cgred.conf**(5) — **cgred.conf** is the configuration file for the **cgred** service.

- **`cgrules.conf`**(5) — **`cgrules.conf`** contains the rules used for determining when tasks belong to certain cgroups.
- **`cgrulesengd`**(8) — the **`cgrulesengd`** service distributes tasks to cgroups.
- **`cgset`**(1) — the **`cgset`** command sets parameters for a cgroup.
- **`lscgroup`**(1) — the **`lscgroup`** command lists the cgroups in a hierarchy.
- **`lssubsys`**(1) — the **`lssubsys`** command lists the hierarchies containing the specified subsystems.

Chapter 4. Control Group Application Examples

This chapter provides application examples that take advantage of the cgroup functionality.

4.1. Prioritizing Database I/O

Running each instance of a database server inside its own dedicated virtual guest allows you to allocate resources per database based on their priority. Consider the following example: a system is running two database servers inside two KVM guests. One of the databases is a high priority database and the other one a low priority database. When both database servers are run simultaneously, the I/O throughput is decreased to accommodate requests from both databases equally; Figure 4.1, “I/O throughput without resource allocation” indicates this scenario — once the low priority database is started (around time 45), I/O throughput is the same for both database servers.

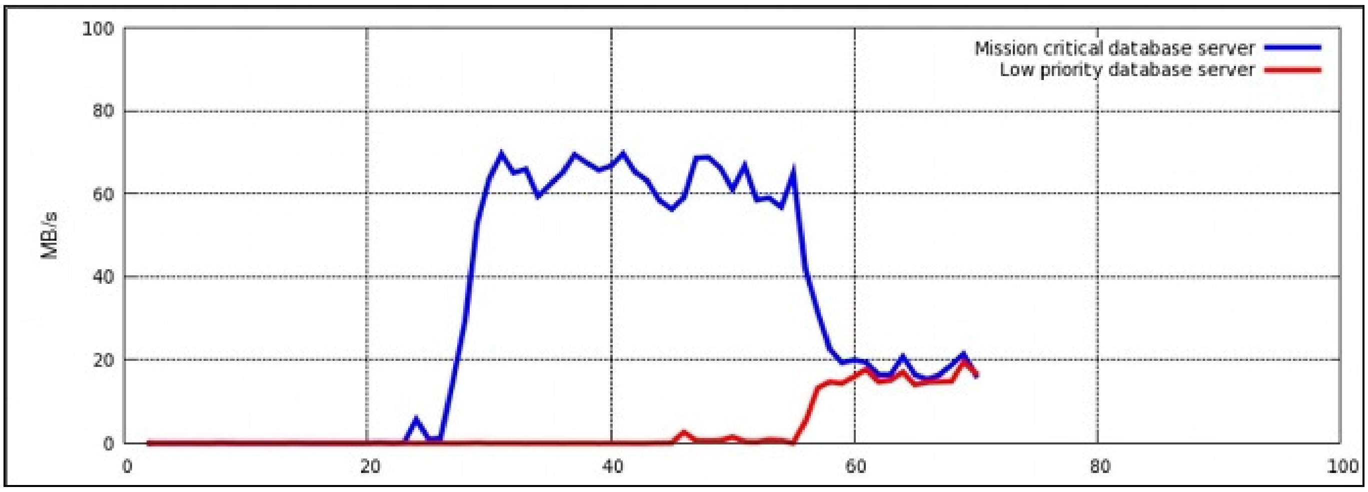

Figure 4.1. I/O throughput without resource allocation

To prioritize the high priority database server, it can be assigned to a cgroup with a high number of reserved I/O operations, whereas the low priority database server can be assigned to a cgroup with a low number of reserved I/O operations. To achieve this, follow the steps in Procedure 4.1, “I/O Throughput Prioritization”, all of which are performed on the host system.

Procedure 4.1. I/O Throughput Prioritization

1. Make sure resource accounting is on for both services:

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

2. Set a ratio of 10:1 for the high and low priority services. Processes running in those service units will use only the resources made available to them

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

Figure 4.2, “I/O throughput with resource allocation” illustrates the outcome of limiting the low priority database and prioritizing the high priority database. As soon as the database servers are moved to their appropriate cgroups (around time 75), I/O throughput is divided between both servers with the ratio of 10:1.

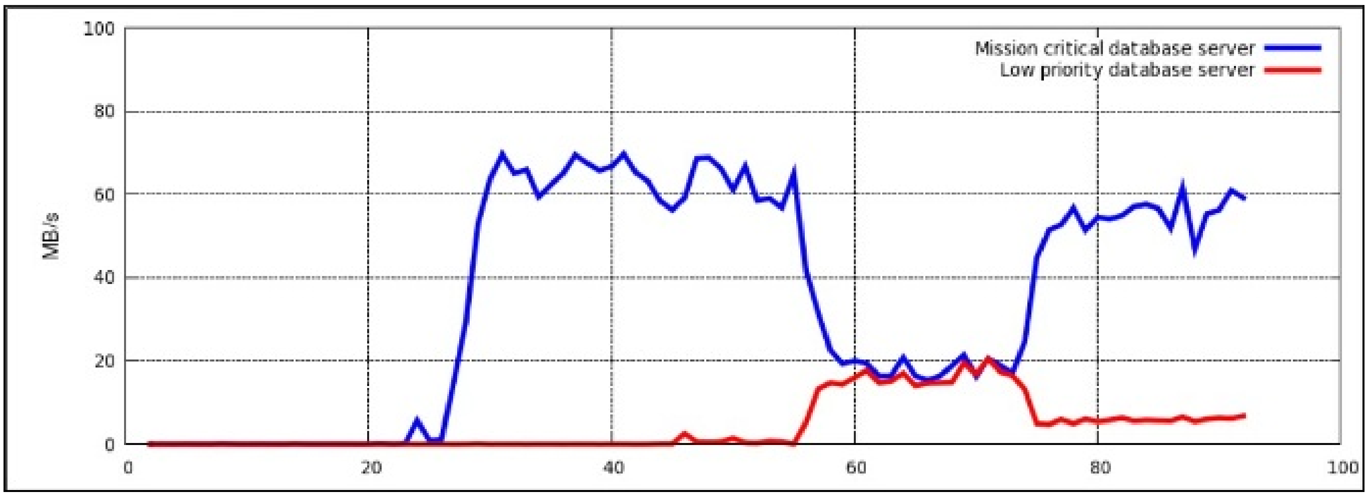

Figure 4.2. I/O throughput with resource allocation

Alternatively, block device I/O throttling can be used for the low priority database to limit its number of read and write operations. For more information, refer to the description of the **`blkio`** controller in Controller-Specific Kernel Documentation.

4.2. Prioritizing Network Traffic

When running multiple network-related services on a single server system, it is important to define network priorities among these services. Defining the priorities ensures that packets originating from certain services have a higher priority than packets originating from other services. For example, such priorities are useful when a server system simultaneously functions as an NFS and Samba server. The NFS traffic has to be of high priority as users expect high throughput. The Samba traffic can be deprioritized to allow better performance of the NFS server.

The **`net_prio`** controller can be used to set network priorities for processes in cgroups. These priorities are then translated into Type of Service (ToS) field bits and embedded into every packet. Follow the steps in Procedure 4.2, “Setting Network Priorities for File Sharing Services” to configure prioritization of two file sharing services (NFS and Samba).

Procedure 4.2. Setting Network Priorities for File Sharing Services

1. The **`net_prio`** controller is not compiled in the kernel, it is a module that has to be loaded manually. To do so, type:

```
~]# modprobe netprio_cgroup
```

2. Attach the **`net_prio`** subsystem to the **`/cgroup/net_prio`** cgroup:

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

3. Create two cgroups, one for each service:

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

4. To automatically move the **nfs** services to the **nfs_high** cgroup, add the following line to the **/etc/sysconfig/nfs** file:

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

   This configuration ensures that **nfs** service processes are moved to the **nfs_high** cgroup when the **nfs** service is started or restarted.

5. The **smbd** service does not have a configuration file in the **/etc/sysconfig** directory. To automatically move the **smbd** service to the **samba_low** cgroup, add the following line to the **/etc/cgrules.conf** file:

   ```
   *:smbd                      net_prio                        samba_low
   ```

   Note that this rule moves every **smbd** service, not only **/usr/sbin/smbd**, into the **samba_low** cgroup.

   You can define rules for the **nmbd** and **winbindd** services to be moved to the **samba_low** cgroup in a similar way.

6. Start the **cgred** service to load the configuration from the previous step:

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

7. For the purposes of this example, let us assume both services use the **eth1** network interface. Define network priorities for each cgroup, where **1** denotes low priority and **10** denotes high priority:

   ```
   ~]# 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. Start the **nfs** and **smb** services and check whether their processes have been moved into the correct cgroups:

   ```
   ~]# 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
```

Network traffic originating from NFS now has higher priority than traffic originating from Samba.

Similar to Procedure 4.2, “Setting Network Priorities for File Sharing Services”, the **net_prio** subsystem can be used to set network priorities for client applications, for example, Firefox.

Appendix A. Revision History

Revision 0.0-1.7	**Thu Aug 18 2016**	**Milan Navrátil**
Preparing document for 7.3 Beta publication.		
Revision 0.0-1.6	**Wed Nov 11 2015**	**Jana Heves**
Version for 7.2 GA release.		
Revision 0.0-1.4	**Thu Feb 19 2015**	**Radek Bíba**
Version for 7.1 GA release. Linux Containers moved to a separate book.		
Revision 0.0-1.0	**Mon Jul 21 2014**	**Peter Ondrejka**
Revision 0.0-0.14	**Mon May 13 2013**	**Peter Ondrejka**
Version for 7.0 GA release		