Red Hat

Red Hat Enterprise Linux 8

Managing, monitoring, and updating the kernel

A guide to managing the Linux kernel on Red Hat Enterprise Linux 8

Last Updated: 2022-12-07

Red Hat Enterprise Linux 8 Managing, monitoring, and updating the kernel

A guide to managing the Linux kernel on Red Hat Enterprise Linux 8

Abstract

This document provides the users and administrators with necessary information about configuring their workstations on the Linux kernel level. Such adjustments bring performance enhancements, easier troubleshooting or optimized system.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright’s message .

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it.

Submitting comments on specific passages

1. View the documentation in the **Multi-page HTML** format and ensure that you see the **Feedback** button in the upper right corner after the page fully loads.
2. Use your cursor to highlight the part of the text that you want to comment on.
3. Click the **Add Feedback** button that appears near the highlighted text.
4. Add your feedback and click **Submit**.

Submitting feedback through Bugzilla (account required)

1. Log in to the Bugzilla website.
2. Select the correct version from the **Version** menu.
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Submit Bug**.

CHAPTER 1. THE LINUX KERNEL

The following sections bring information about the Linux kernel and the Linux kernel RPM package provided and maintained by Red Hat (Red Hat kernel). The integral part of this chapter is instruction on how to keep the Red Hat kernel updated, which ensures the operating system has all the latest bug fixes, performance enhancements, and patches, and is compatible with new hardware.

1.1. WHAT THE KERNEL IS

The kernel is a core part of a Linux operating system that manages the system resources and provides interface between hardware and software applications. The Red Hat kernel is a custom-built kernel based on the upstream Linux mainline kernel that Red Hat engineers further develop and harden with a focus on stability and compatibility with the latest technologies and hardware.

Before Red Hat releases a new kernel version, the kernel needs to pass a set of rigorous quality assurance tests.

The Red Hat kernels are packaged in the RPM format so that they are easily upgraded and verified by the **yum** package manager.

WARNING

Kernels that have not been compiled by Red Hat are **not** supported by Red Hat.

1.2. RPM PACKAGES

An RPM package is a file containing other files and their metadata (information about the files that are needed by the system).

Specifically, an RPM package consists of the **cpio** archive.

The **cpio** archive contains:

- Files
- RPM header (package metadata)
  The **rpm** package manager uses this metadata to determine dependencies, where to install files, and other information.

Types of RPM packages

There are two types of RPM packages. Both types share the file format and tooling, but have different contents and serve different purposes:

- Source RPM (SRPM)
  An SRPM contains source code and a SPEC file, which describes how to build the source code into a binary RPM. Optionally, the patches to source code are included as well.
- Binary RPM
  A binary RPM contains the binaries built from the sources and patches.

1.3. THE LINUX KERNEL RPM PACKAGE OVERVIEW

The **kernel** RPM is a meta package that does not contain any files, but rather ensures that the following required sub-packages are properly installed:

- **kernel-core** - contains the binary image of the kernel, all initramfs-related objects to bootstrap the system, and a minimal number of kernel modules to ensure core functionality. This sub-package alone could be used in virtualized and cloud environments to provide a Red Hat Enterprise Linux 8 kernel with a quick boot time and a small disk size footprint.

- **kernel-modules** - contains the remaining kernel modules that are not present in **kernel-core**.

The small set of **kernel** sub-packages above aims to provide a reduced maintenance surface to system administrators especially in virtualized and cloud environments.

Optional kernel packages are for example:

- **kernel-modules-extra** - contains kernel modules for rare hardware and modules which loading is disabled by default.

- **kernel-debug** — contains a kernel with numerous debugging options enabled for kernel diagnosis, at the expense of reduced performance.

- **kernel-tools** — contains tools for manipulating the Linux kernel and supporting documentation.

- **kernel-devel** — contains the kernel headers and makefiles sufficient to build modules against the **kernel** package.

- **kernel-abi-stablelists** — contains information pertaining to the RHEL kernel ABI, including a list of kernel symbols that are needed by external Linux kernel modules and a **yum** plug-in to aid enforcement.

- **kernel-headers** — includes the C header files that specify the interface between the Linux kernel and user-space libraries and programs. The header files define structures and constants that are needed for building most standard programs.

Additional resources

- *What are the kernel-core, kernel-modules, and kernel-modules-extras packages?*

1.4. DISPLAYING CONTENTS OF THE KERNEL PACKAGE

The following procedure describes how to view the contents of the kernel package and its sub-packages without installing them using the **rpm** command.

Prerequisites

- Obtained **kernel**, **kernel-core**, **kernel-modules**, **kernel-modules-extra** RPM packages for your CPU architecture

Procedure

- List modules for **kernel**:

```
$ rpm -qlp <kernel_rpm>
(contains no files)
…
```

- List modules for **kernel-core**:

```
$ rpm -qlp <kernel-core_rpm>
…
/lib/modules/4.18.0-80.el8.x86_64/kernel/fs/udf/udf.ko.xz
/lib/modules/4.18.0-80.el8.x86_64/kernel/fs/xfs
/lib/modules/4.18.0-80.el8.x86_64/kernel/fs/xfs/xfs.ko.xz
/lib/modules/4.18.0-80.el8.x86_64/kernel/kernel
/lib/modules/4.18.0-80.el8.x86_64/kernel/kernel/trace
/lib/modules/4.18.0-80.el8.x86_64/kernel/kernel/trace/ring_buffer_benchmark.ko.xz
/lib/modules/4.18.0-80.el8.x86_64/kernel/lib
/lib/modules/4.18.0-80.el8.x86_64/kernel/lib/cordic.ko.xz
…
```

- List modules for **kernel-modules**:

```
$ rpm -qlp <kernel-modules_rpm>
…
/lib/modules/4.18.0-80.el8.x86_64/kernel/drivers/infiniband/hw/mlx4/mlx4_ib.ko.xz
/lib/modules/4.18.0-80.el8.x86_64/kernel/drivers/infiniband/hw/mlx5/mlx5_ib.ko.xz
/lib/modules/4.18.0-80.el8.x86_64/kernel/drivers/infiniband/hw/qedr/qedr.ko.xz
/lib/modules/4.18.0-80.el8.x86_64/kernel/drivers/infiniband/hw/usnic/usnic_verbs.ko.xz
/lib/modules/4.18.0-
80.el8.x86_64/kernel/drivers/infiniband/hw/vmw_pvrdma/vmw_pvrdma.ko.xz
…
```

- List modules for **kernel-modules-extra**:

```
$ rpm -qlp <kernel-modules-extra_rpm>
…
/lib/modules/4.18.0-80.el8.x86_64/extra/net/sched/sch_cbq.ko.xz
/lib/modules/4.18.0-80.el8.x86_64/extra/net/sched/sch_choke.ko.xz
/lib/modules/4.18.0-80.el8.x86_64/extra/net/sched/sch_drr.ko.xz
/lib/modules/4.18.0-80.el8.x86_64/extra/net/sched/sch_dsmark.ko.xz
/lib/modules/4.18.0-80.el8.x86_64/extra/net/sched/sch_gred.ko.xz
…
```

Additional resources

- The **rpm(8)** manual page
- RPM packages

1.5. UPDATING THE KERNEL

The following procedure describes how to update the kernel using the **yum** package manager.

Procedure

1. To update the kernel, enter the following command:

   ```
   # yum update kernel
   ```

   This command updates the kernel along with all dependencies to the latest available version.

2. Reboot your system for the changes to take effect.

> **NOTE**
>
> When upgrading from RHEL 7 to RHEL 8, follow relevant sections of the *Upgrading from RHEL 7 to RHEL 8* document.

Additional resources

- Managing software packages

1.6. INSTALLING SPECIFIC KERNEL VERSIONS

The following procedure describes how to install new kernels using the **yum** package manager.

Procedure

- To install a specific kernel version, enter the following command:

  ```
  # yum install kernel-{version}
  ```

Additional resources

- *Red Hat Code Browser*
- *Red Hat Enterprise Linux Release Dates*

CHAPTER 2. MANAGING KERNEL MODULES

The following sections explain what kernel modules are, how to display their information, and how to perform basic administrative tasks with kernel modules.

2.1. INTRODUCTION TO KERNEL MODULES

The Red Hat Enterprise Linux kernel can be extended with optional, additional pieces of functionality, called kernel modules, without having to reboot the system. On Red Hat Enterprise Linux 8, kernel modules are extra kernel code which is built into compressed **<KERNEL_MODULE_NAME>.ko.xz** object files.

The most common functionality enabled by kernel modules are:

- Device driver which adds support for new hardware
- Support for a file system such as **GFS2** or **NFS**
- System calls

On modern systems, kernel modules are automatically loaded when needed. However, in some cases it is necessary to load or unload modules manually.

Like the kernel itself, the modules can take parameters that customize their behavior if needed.

Tooling is provided to inspect which modules are currently running, which modules are available to load into the kernel and which parameters a module accepts. The tooling also provides a mechanism to load and unload kernel modules into the running kernel.

2.2. KERNEL MODULE DEPENDENCIES

Certain kernel modules sometimes depend on one or more other kernel modules. The **/lib/modules/<KERNEL_VERSION>/modules.dep** file contains a complete list of kernel module dependencies for the respective kernel version.

The dependency file is generated by the **depmod** program, which is a part of the **kmod** package. Many of the utilities provided by **kmod** take module dependencies into account when performing operations so that **manual** dependency-tracking is rarely necessary.

WARNING

The code of kernel modules is executed in kernel-space in the unrestricted mode. Because of this, you should be mindful of what modules you are loading.

Additional resources

- **modules.dep(5)** manual page
- **depmod(8)** manual page

2.3. LISTING INSTALLED KERNEL MODULES

The **grubby --info=ALL** command displays an indexed list of installed kernels on **!BLS** and **BLS** installs.

Procedure

- List the installed kernels using the following command:

  ```
  # grubby --info=ALL | grep title
  ```

  The list of all installed kernels is displayed as follows:

  ```
  title=Red Hat Enterprise Linux (4.18.0-20.el8.x86_64) 8.0 (Ootpa)
  title=Red Hat Enterprise Linux (4.18.0-19.el8.x86_64) 8.0 (Ootpa)
  title=Red Hat Enterprise Linux (4.18.0-12.el8.x86_64) 8.0 (Ootpa)
  title=Red Hat Enterprise Linux (4.18.0) 8.0 (Ootpa)
  title=Red Hat Enterprise Linux (0-rescue-2fb13ddde2e24fde9e6a246a942caed1) 8.0 (Ootpa)
  ```

The above example displays the installed kernels list of grubby-8.40-17, from the GRUB menu.

2.4. LISTING CURRENTLY LOADED KERNEL MODULES

The following procedure describes how to view the currently loaded kernel modules.

Prerequisites

- The **kmod** package is installed.

Procedure

- To list all currently loaded kernel modules, execute:

  ```
  $ lsmod

  Module                  Size  Used by
  fuse                  126976  3
  uinput                 20480  1
  xt_CHECKSUM            16384  1
  ipt_MASQUERADE         16384  1
  xt_conntrack           16384  1
  ipt_REJECT             16384  1
  nft_counter            16384  16
  nf_nat_tftp            16384  0
  nf_conntrack_tftp      16384  1 nf_nat_tftp
  tun                    49152  1
  bridge                192512  0
  stp                    16384  1 bridge
  llc                    16384  2 bridge,stp
  nf_tables_set          32768  5
  nft_fib_inet           16384  1
  …
  ```

  In the example above:

- The first column provides the **names** of currently loaded modules.
- The second column displays the amount of **memory** per module in kilobytes.
- The last column shows the number, and optionally the names of modules that are **dependent** on a particular module.

Additional resources

- **/usr/share/doc/kmod/README** file
- **lsmod(8)** manual page

2.5. LISTING ALL INSTALLED KERNELS

The following procedure describes how to use the **grubby** utility to list all installed kernels on your system.

Prerequisites

- You have root permissions.

Procedure

- To list all installed kernels, execute:

```
# grubby --info=ALL | grep ^kernel

kernel="/boot/vmlinuz-4.18.0-305.10.2.el8_4.x86_64"
kernel="/boot/vmlinuz-4.18.0-240.el8.x86_64"
kernel="/boot/vmlinuz-0-rescue-41eb2e172d7244698abda79a51778f1b"
```

The output displays path to all the installed kernels, and displays also their respective versions.

Additional resources

- What grubby is

2.6. SETTING A KERNEL AS DEFAULT

The following procedure describes how to set a specific kernel as default using the **grubby** command-line tool and GRUB.

Procedure

- Setting the kernel as default, using the **grubby** tool
  - Execute the following command to set the kernel as default using the **grubby** tool:

    ```
    # grubby --set-default $kernel_path
    ```

    The command uses a machine ID without the **.conf** suffix as an argument.

NOTE

The machine ID is located in the /**boot**/**loader**/**entries**/ directory.

- Setting the kernel as default, using the **id** argument
    - List the boot entries using the **id** argument and then set an intended kernel as default:

      ```
      # grubby --info ALL | grep id
      # grubby --set-default /boot/vmlinuz-<version>.<architecture>
      ```

NOTE

To list the boot entries using the **title** argument, execute the **# grubby --info=ALL | grep title** command.

- Setting the default kernel for only the next boot
    - Execute the following command to set the default kernel for only the next reboot using the **grub2-reboot** command:

      ```
      # grub2-reboot <index|title|id>
      ```

WARNING

Set the default kernel for only the next boot with care. Installing new kernel RPM's, self-built kernels, and manually adding the entries to the /**boot**/**loader**/**entries**/ directory may change the index values.

2.7. DISPLAYING INFORMATION ABOUT KERNEL MODULES

When working with a kernel module, you may want to see further information about that module. This procedure describes how to display extra information about kernel modules.

Prerequisites

- The **kmod** package is installed.

Procedure

- To display information about any kernel module, execute:

  ```
  $ modinfo <KERNEL_MODULE_NAME>
  ```

  For example:

  ```
  $ modinfo virtio_net
  ```

```
filename:       /lib/modules/4.18.0-94.el8.x86_64/kernel/drivers/net/virtio_net.ko.xz
license:        GPL
description:    Virtio network driver
rhelversion:    8.1
srcversion:     2E9345B281A898A91319773
alias:          virtio:d00000001v*
depends:        net_failover
intree:         Y
name:           virtio_net
vermagic:       4.18.0-94.el8.x86_64 SMP mod_unload modversions
…
parm:           napi_weight:int
parm:           csum:bool
parm:           gso:bool
parm:           napi_tx:bool
```

The **modinfo** command displays some detailed information about the specified kernel module. You can query information about all available modules, regardless of whether they are loaded or not. The **parm** entries show parameters the user is able to set for the module, and what type of value they expect.

NOTE

When entering the name of a kernel module, do not append the **.ko.xz** extension to the end of the name. Kernel module names do not have extensions; their corresponding files do.

Additional resources

- **modinfo(8)** manual page

2.8. LOADING KERNEL MODULES AT SYSTEM RUNTIME

The optimal way to expand the functionality of the Linux kernel is by loading kernel modules. The following procedure describes how to use the **modprobe** command to find and load a kernel module into the currently running kernel.

Prerequisites

- Root permissions
- The **kmod** package is installed.
- The respective kernel module is not loaded. To ensure this is the case, list the loaded kernel modules.

Procedure

1. Select a kernel module you want to load.
   The modules are located in the **/lib/modules/$(uname -r)/kernel/<SUBSYSTEM>/** directory.
2. Load the relevant kernel module:

```
# modprobe <MODULE_NAME>
```

NOTE

When entering the name of a kernel module, do not append the **.ko.xz** extension to the end of the name. Kernel module names do not have extensions; their corresponding files do.

3. Optionally, verify the relevant module was loaded:

```
$ lsmod | grep <MODULE_NAME>
```

If the module was loaded correctly, this command displays the relevant kernel module. For example:

```
$ lsmod | grep serio_raw
serio_raw              16384  0
```

IMPORTANT

The changes described in this procedure **will not persist** after rebooting the system. For information on how to load kernel modules to **persist** across system reboots, see Loading kernel modules automatically at system boot time .

Additional resources

- **modprobe(8)** manual page

2.9. UNLOADING KERNEL MODULES AT SYSTEM RUNTIME

At times, you find that you need to unload certain kernel modules from the running kernel. The following procedure describes how to use the **modprobe** command to find and unload a kernel module at system runtime from the currently loaded kernel.

Prerequisites

- Root permissions
- The **kmod** package is installed.

Procedure

1. Execute the **lsmod** command and select a kernel module you want to unload.
If a kernel module has dependencies, unload those prior to unloading the kernel module. For details on identifying modules with dependencies, see Listing currently loaded kernel modules and Kernel module dependencies.

2. Unload the relevant kernel module:

```
# modprobe -r <MODULE_NAME>
```

When entering the name of a kernel module, do not append the **.ko.xz** extension to the end of the name. Kernel module names do not have extensions; their corresponding files do.

WARNING

Do not unload kernel modules when they are used by the running system. Doing so can lead to an unstable or non-operational system.

3. Optionally, verify the relevant module was unloaded:

```
$ lsmod | grep <MODULE_NAME>
```

If the module was unloaded successfully, this command does not display any output.

IMPORTANT

After finishing this procedure, the kernel modules that are defined to be automatically loaded on boot, **will not stay unloaded** after rebooting the system. For information on how to counter this outcome, see Preventing kernel modules from being automatically loaded at system boot time.

Additional resources

- **modprobe(8)** manual page

2.10. UNLOADING KERNEL MODULES AT EARLY STAGES OF THE BOOT PROCESS

In certain situations it is necessary to unload a kernel module very early in the booting process. For example, when the kernel module contains a code, which causes the system to become unresponsive, and the user is not able to reach the stage to permanently disable the rogue kernel module. In that case it is possible to temporarily block the loading of the kernel module using a bootloader.

IMPORTANT

The changes described in this procedure **will not persist** after the next reboot. For information on how to add a kernel module to a denylist so that it will not be automatically loaded during the boot process, see Preventing kernel modules from being automatically loaded at system boot time.

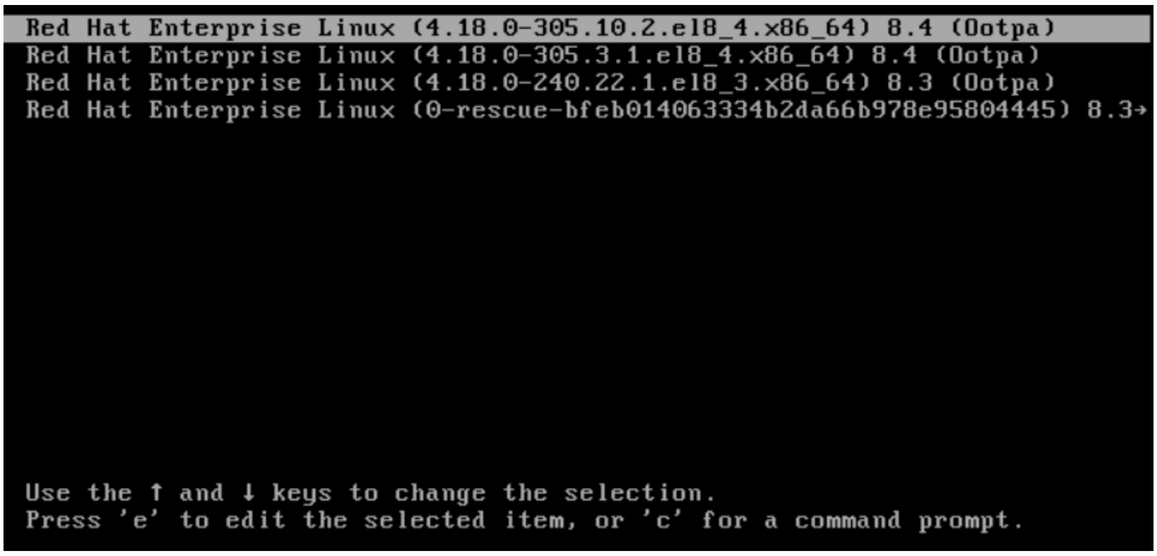
Prerequisites

- You have a loadable kernel module, which you want to prevent from loading for some reason.

Procedure

- Edit the relevant bootloader entry to unload the desired kernel module before the booting sequence continues.
    - Use the cursor keys to highlight the relevant bootloader entry.
    - Press **e** key to edit the entry.

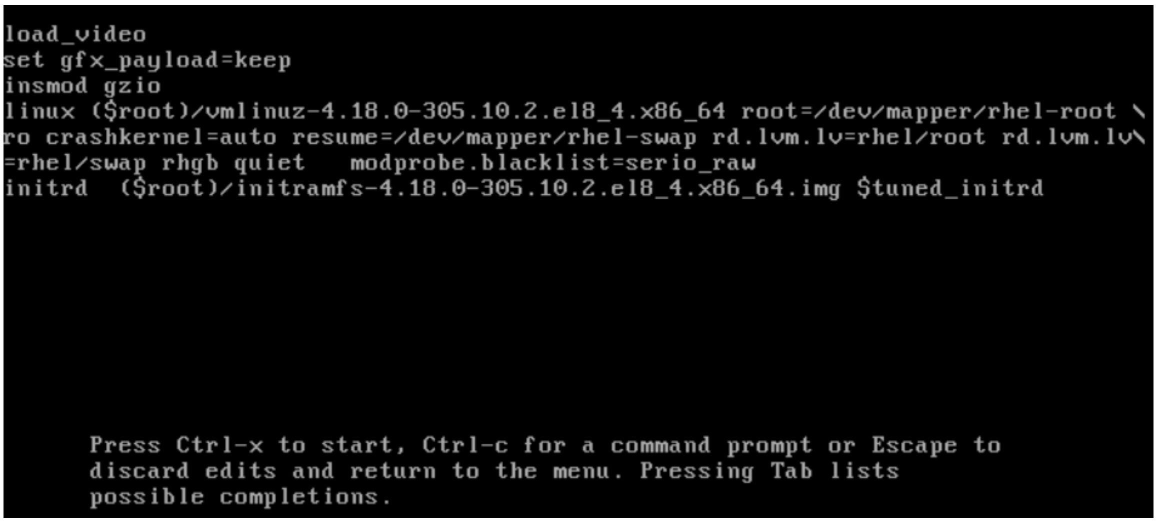
Figure 2.1. Kernel boot menu



```
Red Hat Enterprise Linux (4.18.0-305.10.2.el8_4.x86_64) 8.4 (Ootpa)
Red Hat Enterprise Linux (4.18.0-305.3.1.el8_4.x86_64) 8.4 (Ootpa)
Red Hat Enterprise Linux (4.18.0-240.22.1.el8_3.x86_64) 8.3 (Ootpa)
Red Hat Enterprise Linux (0-rescue-bfeb014063334b2da66b978e95804445) 8.3→

Use the ↑ and ↓ keys to change the selection.
Press 'e' to edit the selected item, or 'c' for a command prompt.
```

- Use the cursor keys to navigate to the line that starts with **linux**.
- Append **modprobe.blacklist=*module_name*** to the end of the line.

Figure 2.2. Kernel boot entry



```
load_video
set gfx_payload=keep
insmod gzio
linux ($root)/vmlinuz-4.18.0-305.10.2.el8_4.x86_64 root=/dev/mapper/rhel-root \
ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv\
=rhel/swap rhgb quiet   modprobe.blacklist=serio_raw
initrd  ($root)/initramfs-4.18.0-305.10.2.el8_4.x86_64.img $tuned_initrd

      Press Ctrl-x to start, Ctrl-c for a command prompt or Escape to
      discard edits and return to the menu. Pressing Tab lists
      possible completions.
```

The **serio_raw** kernel module illustrates a rogue module to be unloaded early in the boot process.

- Press **CTRL+x** keys to boot using the modified configuration.

Verification

- Once the system fully boots, verify that the relevant kernel module is not loaded.

```
# lsmod | grep serio_raw
```

Additional resources

- Managing kernel modules

2.11. LOADING KERNEL MODULES AUTOMATICALLY AT SYSTEM BOOT TIME

The following procedure describes how to configure a kernel module so that it is loaded automatically during the boot process.

Prerequisites

- Root permissions
- The **kmod** package is installed.

Procedure

1. Select a kernel module you want to load during the boot process.
   The modules are located in the **/lib/modules/$(uname -r)/kernel/<SUBSYSTEM>/** directory.
2. Create a configuration file for the module:

   ```
   # echo <MODULE_NAME> > /etc/modules-load.d/<MODULE_NAME>.conf
   ```

   NOTE

   When entering the name of a kernel module, do not append the **.ko.xz** extension to the end of the name. Kernel module names do not have extensions; their corresponding files do.

3. Optionally, after reboot, verify the relevant module was loaded:

   ```
   $ lsmod | grep <MODULE_NAME>
   ```

   The example command above should succeed and display the relevant kernel module.

IMPORTANT

The changes described in this procedure **will persist** after rebooting the system.

Additional resources

- **modules-load.d(5)** manual page

2.12. PREVENTING KERNEL MODULES FROM BEING AUTOMATICALLY LOADED AT SYSTEM BOOT TIME

The following procedure describes how to add a kernel module to a denylist so that it will not be automatically loaded during the boot process.

Prerequisites

- Root permissions
- The **kmod** package is installed.

- Ensure that a kernel module in a denylist is not vital for your current system configuration.

Procedure

1. Select a kernel module that you want to put in a denylist:

```
$ lsmod

Module                  Size  Used by
fuse               126976  3
xt_CHECKSUM           16384  1
ipt_MASQUERADE        16384  1
uinput             20480  1
xt_conntrack          16384  1
…
```

   The **lsmod** command displays a list of modules loaded to the currently running kernel.

   - Alternatively, identify an unloaded kernel module you want to prevent from potentially loading.
     All kernel modules are located in the **/lib/modules/<KERNEL_VERSION>/kernel/<SUBSYSTEM>/** directory.

2. Create a configuration file for a denylist:

```
# vim /etc/modprobe.d/blacklist.conf

 # Blacklists <KERNEL_MODULE_1>
 blacklist <MODULE_NAME_1>
 install <MODULE_NAME_1> /bin/false

 # Blacklists <KERNEL_MODULE_2>
 blacklist <MODULE_NAME_2>
 install <MODULE_NAME_2> /bin/false

 # Blacklists <KERNEL_MODULE_n>
 blacklist <MODULE_NAME_n>
 install <MODULE_NAME_n> /bin/false
 …
```

   The example shows the contents of the **blacklist.conf** file, edited by the **vim** editor. The **blacklist** line ensures that the relevant kernel module will not be automatically loaded during the boot process. The **blacklist** command, however, does not prevent the module from being loaded as a dependency for another kernel module that is not in a denylist. Therefore the **install** line causes the **/bin/false** to run instead of installing a module.

   The lines starting with a hash sign are comments to make the file more readable.

   NOTE

   When entering the name of a kernel module, do not append the **.ko.xz** extension to the end of the name. Kernel module names do not have extensions; their corresponding files do.

3. Create a backup copy of the current initial ramdisk image before rebuilding:

```
# cp /boot/initramfs-$(uname -r).img /boot/initramfs-$(uname -r).bak.$(date +%m-%d-
%H%M%S).img
```

The command above creates a backup **initramfs** image in case the new version has an unexpected problem.

- Alternatively, create a backup copy of other initial ramdisk image which corresponds to the kernel version for which you want to put kernel modules in a denylist:

  ```
  # cp /boot/initramfs-<SOME_VERSION>.img /boot/initramfs-
  <SOME_VERSION>.img.bak.$(date +%m-%d-%H%M%S)
  ```

4. Generate a new initial ramdisk image to reflect the changes:

   ```
   # dracut -f -v
   ```

   - If you are building an initial ramdisk image for a different kernel version than you are currently booted into, specify both target **initramfs** and kernel version:

     ```
     # dracut -f -v /boot/initramfs-<TARGET_VERSION>.img
     <CORRESPONDING_TARGET_KERNEL_VERSION>
     ```

5. Reboot the system:

   ```
   $ reboot
   ```

IMPORTANT

The changes described in this procedure **will take effect and persist**after rebooting the system. If you improperly put a key kernel module in a denylist, you can face an unstable or non-operational system.

Additional resources

- How do I prevent a kernel module from loading automatically?
- **dracut(8)** manual page

2.13. COMPILING CUSTOM KERNEL MODULES

You can build a sampling kernel module as requested by various configurations at hardware and software level.

Prerequisites

- You installed the **kernel-devel**, **gcc**, and **elfutils-libelf-devel** packages.
- You have root permissions.
- You created the **/root/testmodule/** directory where you compile the custom kernel module.

Procedure

1. Create the **/root/testmodule/test.c** file with the following content.

   ```
   #include <linux/module.h>
   #include <linux/kernel.h>

   int init_module(void)
      { printk("Hello World\n This is a test\n"); return 0; }

   void cleanup_module(void)
      { printk("Good Bye World"); }
   ```

   The **test.c** file is a source file that provides the main functionality to the kernel module. The file has been created in a dedicated **/root/testmodule/** directory for organizational purposes. After the module compilation, the **/root/testmodule/** directory will contain multiple files.

   The **test.c** file includes from the system libraries:

   - The **linux/kernel.h** header file is necessary for the **printk()** function in the example code.
   - The **linux/module.h** file contains function declarations and macro definitions to be shared between several source files written in C programming language.
     Next follow the **init_module()** and **cleanup_module()** functions to start and end the kernel logging function **printk()**, which prints text.

2. Create the **/root/testmodule/Makefile** file with the following content.

   ```
   obj-m := test.o
   ```

   The Makefile contains instructions that the compiler has to produce an object file specifically named **test.o**. The **obj-m** directive specifies that the resulting **test.ko** file is going to be compiled as a loadable kernel module. Alternatively, the **obj-y** directive would instruct to build **test.ko** as a built-in kernel module.

3. Compile the kernel module.

   ```
   # make -C /lib/modules/$(uname -r)/build M=/root/testmodule modules
   make: Entering directory '/usr/src/kernels/4.18.0-305.el8.x86_64'
     CC [M]  /root/testmodule/test.o
     Building modules, stage 2.
     MODPOST 1 modules
   WARNING: modpost: missing MODULE_LICENSE() in /root/testmodule/test.o
   see include/linux/module.h for more information
     CC      /root/testmodule/test.mod.o
     LD [M]  /root/testmodule/test.ko
   make: Leaving directory '/usr/src/kernels/4.18.0-305.el8.x86_64'
   ```

   The compiler creates an object file (**test.o**) for each source file (**test.c**) as an intermediate step before linking them together into the final kernel module (**test.ko**).

   After a successful compilation, **/root/testmodule/** contains additional files that relate to the compiled custom kernel module. The compiled module itself is represented by the **test.ko** file.

Verification

1. Optional: check the contents of the **/root/testmodule/** directory:

```
# ls -l /root/testmodule/
total 152
-rw-r—r--. 1 root root    16 Jul 26 08:19 Makefile
-rw-r—r--. 1 root root    25 Jul 26 08:20 modules.order
-rw-r—r--. 1 root root     0 Jul 26 08:20 Module.symvers
-rw-r—r--. 1 root root   224 Jul 26 08:18 test.c
-rw-r—r--. 1 root root 62176 Jul 26 08:20 test.ko
-rw-r—r--. 1 root root    25 Jul 26 08:20 test.mod
-rw-r—r--. 1 root root   849 Jul 26 08:20 test.mod.c
-rw-r—r--. 1 root root 50936 Jul 26 08:20 test.mod.o
-rw-r—r--. 1 root root 12912 Jul 26 08:20 test.o
```

2. Copy the kernel module to the **/lib/modules/$(uname -r)/** directory:

   ```
   # cp /root/testmodule/test.ko /lib/modules/$(uname -r)/
   ```

3. Update the modular dependency list:

   ```
   # depmod -a
   ```

4. Load the kernel module:

   ```
   # modprobe -v test
   insmod /lib/modules/4.18.0-305.el8.x86_64/test.ko
   ```

5. Verify that the kernel module was successfully loaded:

   ```
   # lsmod | grep test
   test                  16384  0
   ```

6. Read the latest messages from the kernel ring buffer:

   ```
   # dmesg
   [74422.545004] Hello World
                  This is a test
   ```

Additional resources

- Managing kernel modules

CHAPTER 3. SIGNING KERNEL MODULES FOR SECURE BOOT

You can enhance the security of your system by using signed kernel modules. The following sections describe how to self-sign privately built kernel modules for use with RHEL 8 on UEFI-based build systems where Secure Boot is enabled. These sections also provide an overview of available options for importing your public key into a target system where you want to deploy your kernel modules.

If Secure Boot is enabled, the UEFI operating system boot loaders, the Red Hat Enterprise Linux kernel, and all kernel modules have to be signed with a private key and authenticated with the corresponding public key. If they are not signed and authenticated, the system will not be allowed to finish the booting process.

The RHEL 8 distribution includes:

- Signed boot loaders
- Signed kernels
- Signed kernel modules

In addition, the signed first-stage boot loader and the signed kernel include embedded Red Hat public keys. These signed executable binaries and embedded keys enable RHEL 8 to install, boot, and run with the Microsoft UEFI Secure Boot Certification Authority keys that are provided by the UEFI firmware on systems that support UEFI Secure Boot. Note that not all UEFI-based systems include support for Secure Boot.

Prerequisites

To be able to sign externally built kernel modules, install the utilities listed in the following table on the build system.

Table 3.1. Required utilities

Utility	Provided by package	Used on	Purpose
openssl	**openssl**	Build system	Generates public and private X.509 key pair
sign-file	**kernel-devel**	Build system	Executable file used to sign a kernel module with the private key
mokutil	**mokutil**	Target system	Optional utility used to manually enroll the public key
keyctl	**keyutils**	Target system	Optional utility used to display public keys in the system keyring

NOTE

The build system, where you build and sign your kernel module, does not need to have UEFI Secure Boot enabled and does not even need to be a UEFI-based system.

3.1. WHAT IS UEFI SECURE BOOT

With the *Unified Extensible Firmware Interface* (UEFI) Secure Boot technology you can ensure that the system boot loader is signed with a cryptographic key. The database of public keys which is contained in the firmware, authorizes the signing key. You can verify a signature in the next-stage boot loader and the kernel. As a result, you can prevent the execution of the kernel space code which has not been signed by a trusted key.

UEFI Secure Boot establishes a chain of trust from the firmware to the signed drivers and kernel modules as follows:

- A UEFI private key signs, and a public key authenticates the **shim** first-stage boot loader. A *certificate authority* (CA) in turn signs the public key. The CA is stored in the firmware database.
- The **shim.efi** contains the Red Hat public key **Red Hat Secure Boot (CA key 1)**to authenticate the GRUB boot loader, and the kernel.
- The kernel in turn contains public keys to authenticate drivers and modules.

Secure Boot is the boot path validation component of the UEFI specification. The specification defines:

- Programming interface for cryptographically protected UEFI variables in non-volatile storage
- Storing the trusted X.509 root certificates in UEFI variables
- Validation of UEFI applications like boot loaders and drivers
- Procedures to revoke known-bad certificates and application hashes

UEFI Secure Boot does not prevent installation or removal of second-stage boot loaders. Also, it does not require explicit user confirmation of such changes. Signatures are verified during booting, not when the boot loader is installed or updated. Therefore, UEFI Secure Boot does not stop boot path manipulations. It helps in the detection of unauthorized changes.

NOTE

The boot loader or the kernel work as long as a system trusted key signs them.

3.2. UEFI SECURE BOOT SUPPORT

RHEL 8 includes support for the Unified Extensible Firmware Interface (UEFI) Secure Boot feature. It means that you can install and run RHEL 8 on machines with enabled UEFI Secure Boot. On these systems, you must sign the all loaded drivers with a trusted key. Otherwise the system will not accept them. Red Hat provides drivers which are signed and authenticated by the relevant Red Hat keys.

If you want to load externally built drivers you must sign them as well.

Restrictions Imposed by UEFI Secure Boot

- The system only runs the kernel mode code after its signature has been properly authenticated.
- GRUB module loading is disabled because there is no infrastructure for signing and verification of GRUB modules. Allowing them to be loaded constitutes execution of untrusted code inside the security perimeter that Secure Boot defines.
- Red Hat provides a signed GRUB binary that contains all the supported modules on RHEL 8.

Additional resources

- Restrictions Imposed by UEFI Secure Boot

3.3. REQUIREMENTS FOR AUTHENTICATING KERNEL MODULES WITH X.509 KEYS

In RHEL 8, when a kernel module is loaded, the kernel checks the signature of the module against the public X.509 keys from the kernel system keyring (**.builtin_trusted_keys**) and the kernel platform keyring (**.platform**). The **.platform** keyring contains keys from third-party platform providers and custom public keys. The keys from the kernel system **.blacklist** keyring are excluded from verification. The following sections provide an overview of sources of keys, keyrings and examples of loaded keys from different sources in the system. Also, you can see how to authenticate a kernel module.

You need to meet certain conditions to load kernel modules on systems with enabled UEFI Secure Boot functionality.

If UEFI Secure Boot is enabled or if the **module.sig_enforce** kernel parameter has been specified:

- You can only load those signed kernel modules whose signatures were authenticated against keys from the system keyring (**.builtin_trusted_keys**) and the platform keyring (**.platform**).
- The public key must not be on the system revoked keys keyring (**.blacklist**).

If UEFI Secure Boot is disabled and the **module.sig_enforce** kernel parameter has not been specified:

- You can load unsigned kernel modules and signed kernel modules without a public key.

If the system is not UEFI-based or if UEFI Secure Boot is disabled:

- Only the keys embedded in the kernel are loaded onto **.builtin_trusted_keys** and **.platform**.
- You have no ability to augment that set of keys without rebuilding the kernel.

Table 3.2. Kernel module authentication requirements for loading

Module signed	Public key found and signature valid	UEFI Secure Boot state	sig_enforce	Module load	Kernel tainted
Unsigned	-	Not enabled	Not enabled	Succeeds	Yes
		Not enabled	Enabled	Fails	-
		Enabled	-	Fails	-
Signed	No	Not enabled	Not enabled	Succeeds	Yes
		Not enabled	Enabled	Fails	-
		Enabled	-	Fails	-
Signed	Yes	Not enabled	Not enabled	Succeeds	No

Module signed	Public key found and signature valid	UEFI Secure Boot state	sig_enforce	Module load	Kernel tainted
		Not enabled	Enabled	Succeeds	No
		Enabled	-	Succeeds	No

3.4. SOURCES FOR PUBLIC KEYS

During boot, the kernel loads X.509 keys from a set of persistent key stores into the following keyrings:

- The system keyring (**.builtin_trusted_keys**)
- The **.platform** keyring
- The system **.blacklist** keyring

Table 3.3. Sources for system keyrings

Source of X.509 keys	User can add keys	UEFI Secure Boot state	Keys loaded during boot
Embedded in kernel	No	-	**.builtin_trusted_keys**
UEFI Secure Boot "db"	Limited	Not enabled	No
		Enabled	**.platform**
Embedded in **shim.efi** boot loader	No	Not enabled	No
		Enabled	**.builtin_trusted_keys**
Machine Owner Key (MOK) list	Yes	Not enabled	No
		Enabled	**.platform**

.builtin_trusted_keys:

- a keyring that is built on boot
- contains trusted public keys
- **root** privileges are needed to view the keys

.platform:

- a keyring that is built on boot
- contains keys from third-party platform providers and custom public keys

- **root** privileges are needed to view the keys

.blacklist

- a keyring with X.509 keys which have been revoked
- a module signed by a key from **.blacklist** will fail authentication even if your public key is in **.builtin_trusted_keys**

UEFI Secure Boot db:

- a signature database
- stores keys (hashes) of UEFI applications, UEFI drivers, and bootloaders
- the keys can be loaded on the machine

UEFI Secure Boot dbx:

- a revoked signature database
- prevents keys from being loaded
- the revoked keys from this database are added to the **.blacklist** keyring

3.5. GENERATING A PUBLIC AND PRIVATE KEY PAIR

You need to generate a public and private X.509 key pair to succeed in your efforts of using kernel modules on a Secure Boot-enabled system. You will later use the private key to sign the kernel module. You will also have to add the corresponding public key to the Machine Owner Key (MOK) for Secure Boot to validate the signed module.

Some of the parameters for this key pair generation are best specified with a configuration file.

Procedure

1. Create a configuration file with parameters for the key pair generation:

```
# cat << EOF > configuration_file.config
[ req ]
default_bits = 4096
distinguished_name = req_distinguished_name
prompt = no
string_mask = utf8only
x509_extensions = myexts

[ req_distinguished_name ]
O = Organization
CN = Organization signing key
emailAddress = E-mail address

[ myexts ]
basicConstraints=critical,CA:FALSE
keyUsage=digitalSignature
```

```
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
EOF
```

2. Create an X.509 public and private key pair as shown in the following example:

```
# openssl req -x509 -new -nodes -utf8 -sha256 -days 36500 \
-batch -config configuration_file.config -outform DER \
-out my_signing_key_pub.der \
-keyout my_signing_key.priv
```

   The public key will be written to the ***my_signing_key_pub*.der** file and the private key will be written to the ***my_signing_key*.priv** file.

   IMPORTANT

   In RHEL 8, the validity dates of the key pair do not matter. The key does not expire, but it is recommended for good security practices that the kernel module be signed within the validity period of its signing key. However, the **sign-file** utility will not warn you and the key will be usable regardless of the validity dates.

3. Optionally, you can review the validity dates of your public keys like in the example below:

```
# openssl x509 -inform der -text -noout -in my_signing_key_pub.der

Validity
        Not Before: Feb 14 16:34:37 2019 GMT
        Not After : Feb 11 16:34:37 2029 GMT
```

4. Enroll your public key on all systems where you want to authenticate and load your kernel module.

WARNING

Apply strong security measures and access policies to guard the contents of your private key. In the wrong hands, the key could be used to compromise any system which is authenticated by the corresponding public key.

Additional resources

- **openssl(1)** manual page
- *RHEL Security Guide*
- Enrolling public key on target system by adding the public key to the MOK list

3.6. EXAMPLE OUTPUT OF SYSTEM KEYRINGS

You can display information about the keys on the system keyrings using the **keyctl** utility from the **keyutils** package.

The following is a shortened example output of **.builtin_trusted_keys**, **.platform**, and **.blacklist** keyrings from a RHEL 8 system where UEFI Secure Boot is enabled.

Prerequisites

- You have root permissions.
- You installed the **keyctl** utility from the **keyutils** package.

```
# keyctl list %:.builtin_trusted_keys
6 keys in keyring:
...asymmetric: Red Hat Enterprise Linux Driver Update Program (key 3): bf57f3e87...
...asymmetric: Red Hat Secure Boot (CA key 1): 4016841644ce3a810408050766e8f8a29...
...asymmetric: Microsoft Corporation UEFI CA 2011: 13adbf4309bd82709c8cd54f316ed...
...asymmetric: Microsoft Windows Production PCA 2011: a92902398e16c49778cd90f99e...
...asymmetric: Red Hat Enterprise Linux kernel signing key: 4249689eefc77e95880b...
...asymmetric: Red Hat Enterprise Linux kpatch signing key: 4d38fd864ebe18c5f0b7...

# keyctl list %:.platform
4 keys in keyring:
...asymmetric: VMware, Inc.: 4ad8da0472073...
...asymmetric: Red Hat Secure Boot CA 5: cc6fafe72...
...asymmetric: Microsoft Windows Production PCA 2011: a929f298e1...
...asymmetric: Microsoft Corporation UEFI CA 2011: 13adbf4e0bd82...

# keyctl list %:.blacklist
4 keys in keyring:
...blacklist: bin:f5ff83a...
...blacklist: bin:0dfdbec...
...blacklist: bin:38f1d22...
...blacklist: bin:51f831f...
```

The **.builtin_trusted_keys** keyring above shows the addition of two keys from the UEFI Secure Boot "db" keys as well as the **Red Hat Secure Boot (CA key 1)**, which is embedded in the **shim.efi** boot loader.

The following example shows the kernel console output. The messages identify the keys with a UEFI Secure Boot related source. These include UEFI Secure Boot db, embedded shim, and MOK list.

```
# dmesg | egrep 'integrity.*cert'
[1.512966] integrity: Loading X.509 certificate: UEFI:db
[1.513027] integrity: Loaded X.509 cert 'Microsoft Windows Production PCA 2011: a929023...
[1.513028] integrity: Loading X.509 certificate: UEFI:db
[1.513057] integrity: Loaded X.509 cert 'Microsoft Corporation UEFI CA 2011: 13adbf4309...
[1.513298] integrity: Loading X.509 certificate: UEFI:MokListRT (MOKvar table)
[1.513549] integrity: Loaded X.509 cert 'Red Hat Secure Boot CA 5: cc6fa5e72868ba494e93...
```

Additional resources

- **keyctl(1)**, **dmesg(1)** manual pages

3.7. ENROLLING PUBLIC KEY ON TARGET SYSTEM BY ADDING THE PUBLIC KEY TO THE MOK LIST

When RHEL 8 boots on a UEFI-based system with Secure Boot enabled, the kernel loads onto the system keyring (**.builtin_trusted_keys**) all public keys that are in the Secure Boot db key database. At the same time the kernel excludes the keys in the dbx database of revoked keys. The sections below describe different ways of importing a public key on a target system so that the system keyring (**.builtin_trusted_keys**) is able to use the public key to authenticate a kernel module.

The Machine Owner Key (MOK) facility feature can be used to expand the UEFI Secure Boot key database. When RHEL 8 boots on a UEFI-enabled system with Secure Boot enabled, the keys on the MOK list are also added to the system keyring (**.builtin_trusted_keys**) in addition to the keys from the key database. The MOK list keys are also stored persistently and securely in the same fashion as the Secure Boot database keys, but these are two separate facilities. The MOK facility is supported by **shim.efi**, **MokManager.efi**, **grubx64.efi**, and the **mokutil** utility.

Enrolling a MOK key requires manual interaction by a user at the UEFI system console on each target system. Nevertheless, the MOK facility provides a convenient method for testing newly generated key pairs and testing kernel modules signed with them.

Procedure

1. Request the addition of your public key to the MOK list:

   ```
   # mokutil --import my_signing_key_pub.der
   ```

   You will be asked to enter and confirm a password for this MOK enrollment request.

2. Reboot the machine.
   The pending MOK key enrollment request will be noticed by **shim.efi** and it will launch **MokManager.efi** to allow you to complete the enrollment from the UEFI console.

3. Choose "Enroll MOK" and enter the password you previously associated with this request when prompted and confirm the enrollment.
   Your public key is added to the MOK list, which is persistent.

Once a key is on the MOK list, it will be automatically propagated to the system keyring on this and subsequent boots when UEFI Secure Boot is enabled.

NOTE

To facilitate authentication of your kernel module on your systems, consider requesting your system vendor to incorporate your public key into the UEFI Secure Boot key database in their factory firmware image.

3.8. SIGNING KERNEL MODULES WITH THE PRIVATE KEY

Users are able to obtain enhanced security benefits on their systems by loading signed kernel modules if the UEFI Secure Boot mechanism is enabled. The following sections describe how to sign kernel modules with the private key.

Prerequisites

- You generated a public and private key pair and know the validity dates of your public keys. For details, see Generating a public and private key pair .
- You enrolled your public key on the target system. For details, see Enrolling public key on target system by adding the public key to the MOK list.
- You have a kernel module in ELF image format available for signing.

Procedure

- Execute the **sign-file** utility with parameters as shown in the example below:

```
# /usr/src/kernels/$(uname -r)/scripts/sign-file sha256 my_signing_key.priv
my_signing_key_pub.der my_module.ko
```

  sign-file computes and appends the signature directly to the ELF image in your kernel module file. The **modinfo** utility can be used to display information about the kernel module's signature, if it is present.

  NOTE

  The appended signature is not contained in an ELF image section and is not a formal part of the ELF image. Therefore, utilities such as **readelf** will not be able to display the signature on your kernel module.

  Your kernel module is now ready for loading. Note that your signed kernel module is also loadable on systems where UEFI Secure Boot is disabled or on a non-UEFI system. That means you do not need to provide both a signed and unsigned version of your kernel module.

  IMPORTANT

  In RHEL 8, the validity dates of the key pair matter. The key does not expire, but the kernel module must be signed within the validity period of its signing key. The **sign-file** utility will not warn you of this. For example, a key that is only valid in 2019 can be used to authenticate a kernel module signed in 2019 with that key. However, users cannot use that key to sign a kernel module in 2020.

Additional resources

- Displaying information about kernel modules

3.9. LOADING SIGNED KERNEL MODULES

Once your public key is enrolled in the system keyring (**.builtin_trusted_keys**) and the MOK list, and after you have signed the respective kernel module with your private key, you can finally load your signed kernel module with the **modprobe** command as described in the following section.

Prerequisites

- You have generated the public and private keypair. For details, see Generating a public and private key pair.
- You have enrolled the public key into the system keyring. For details, see Enrolling public key on target system by adding the public key to the MOK list.

- You have signed a kernel module with the private key. For details, see Signing kernel modules with the private key.

Procedure

1. Verify that your public keys are on the system keyring:

   ```
   # keyctl list %:.platform
   ```

2. Install the **kernel-modules-extra** package which will create the **/lib/modules/$(uname -r)/extra/** directory:

   ```
   # yum -y install kernel-modules-extra
   ```

3. Copy the kernel module into the **/extra/** directory of the kernel you want:

   ```
   # cp my_module.ko /lib/modules/$(uname -r)/extra/
   ```

4. Update the modular dependency list:

   ```
   # depmod -a
   ```

5. Load the kernel module and verify that it was successfully loaded:

   ```
   # modprobe -v my_module
   # lsmod | grep my_module
   ```

   a. Optionally, to load the module on boot, add it to the **/etc/modules-loaded.d/my_module.conf** file:

      ```
      # echo "my_module" > /etc/modules-load.d/my_module.conf
      ```

Additional resources

- Managing kernel modules

CHAPTER 4. CONFIGURING KERNEL COMMAND-LINE PARAMETERS

Kernel command-line parameters are a way to change the behavior of certain aspects of the Red Hat Enterprise Linux kernel at boot time. As a system administrator, you have full control over what options get set at boot. Certain kernel behaviors are only able to be set at boot time, so understanding how to make these changes is a key administration skill.

IMPORTANT

Opting to change the behavior of the system by modifying kernel command-line parameters may have negative effects on your system. You should therefore test changes prior to deploying them in production. For further guidance, contact Red Hat Support.

4.1. UNDERSTANDING KERNEL COMMAND-LINE PARAMETERS

Kernel command-line parameters are used for boot time configuration of:

- The Red Hat Enterprise Linux kernel
- The initial RAM disk
- The user space features

Kernel boot time parameters are often used to overwrite default values and for setting specific hardware settings.

By default, the kernel command-line parameters for systems using the GRUB bootloader are defined in the **kernelopts** variable of the **/boot/grub2/grubenv** file for each kernel boot entry.

NOTE

For IBM Z, the kernel command-line parameters are stored in the boot entry configuration file because the **zipl** bootloader does not support environment variables. Thus, the **kernelopts** environment variable cannot be used.

Additional resources

- **kernel-command-line(7)**, **bootparam(7)** and **dracut.cmdline(7)** manual pages
- How to install and boot custom kernels in Red Hat Enterprise Linux 8

4.2. WHAT GRUBBY IS

grubby is a utility for manipulating boot loader configuration files.

You can also use **grubby** for changing the default boot entry, and for adding or removing arguments from a GRUB2 menu entry.

Additional resources

- The **grubby(8)** manual page

4.3. WHAT BOOT ENTRIES ARE

A boot entry is a collection of options which are stored in a configuration file and tied to a particular kernel version. In practice, you have at least as many boot entries as your system has installed kernels. The boot entry configuration file is located in the **/boot/loader/entries/** directory and can look like this:

```
6f9cc9cb7d7845d49698c9537337cedc-4.18.0-5.el8.x86_64.conf
```

The file name above consists of a machine ID stored in the **/etc/machine-id** file, and a kernel version.

The boot entry configuration file contains information about the kernel version, the initial ramdisk image, and the **kernelopts** environment variable, which contains the kernel command-line parameters. The example contents of a boot entry config can be seen below:

```
title Red Hat Enterprise Linux (4.18.0-74.el8.x86_64) 8.0 (Ootpa)
version 4.18.0-74.el8.x86_64
linux /vmlinuz-4.18.0-74.el8.x86_64
initrd /initramfs-4.18.0-74.el8.x86_64.img $tuned_initrd
options $kernelopts $tuned_params
id rhel-20190227183418-4.18.0-74.el8.x86_64
grub_users $grub_users
grub_arg --unrestricted
grub_class kernel
```

The **kernelopts** environment variable is defined in the **/boot/grub2/grubenv** file.

Additional resources

- How to install and boot custom kernels in Red Hat Enterprise Linux 8

4.4. CHANGING KERNEL COMMAND-LINE PARAMETERS FOR ALL BOOT ENTRIES

This procedure describes how to change kernel command-line parameters for all boot entries on your system.

Prerequisites

- Verify that the **grubby** utility is installed on your system.
- Verify that the **zipl** utility is installed on your IBM Z system.

Procedure

- To add a parameter:

  ```
  # grubby --update-kernel=ALL --args="<NEW_PARAMETER>"
  ```

  For systems that use the GRUB bootloader, the command updates the **/boot/grub2/grubenv** file by adding a new kernel parameter to the **kernelopts** variable in that file.

  On IBM Z that use the zIPL bootloader, the command adds a new kernel parameter to each **/boot/loader/entries/<*ENTRY*>.conf** file.

- On IBM Z, execute the **zipl** command with no options to update the boot menu.

- To remove a parameter:

  ```
  # grubby --update-kernel=ALL --remove-args="<PARAMETER_TO_REMOVE>"
  ```

  - On IBM Z, execute the **zipl** command with no options to update the boot menu.

- After each update of your kernel package, propagate the configured kernel options to the new kernels:

  ```
  # grub2-mkconfig -o /etc/grub2.cfg
  ```

> IMPORTANT
>
> Newly installed kernels do not inherit the kernel command-line parameters from your previously configured kernels. You must run the **grub2-mkconfig** command on the newly installed kernel to propagate the needed parameters to your new kernel.

Additional resources

- Understanding kernel command-line parameters
- **grubby(8)** and **zipl(8)** manual pages
- *grubby tool*

4.5. CHANGING KERNEL COMMAND-LINE PARAMETERS FOR A SINGLE BOOT ENTRY

This procedure describes how to change kernel command-line parameters for a single boot entry on your system.

Prerequisites

- Verify that the **grubby** and **zipl** utilities are installed on your system.

Procedure

- To add a parameter:

  ```
  #  grubby --update-kernel=/boot/vmlinuz-$(uname -r) --args="<NEW_PARAMETER>"
  ```

  - On IBM Z, execute the **zipl** command with no options to update the boot menu.

- To remove a parameter use the following:

  ```
  #  grubby --update-kernel=/boot/vmlinuz-$(uname -r) --remove-args="
  <PARAMETER_TO_REMOVE>"
  ```

  - On IBM Z, execute the **zipl** command with no options to update the boot menu.

NOTE

On systems that use the **grub.cfg** file, there is, by default, the **options** parameter for each kernel boot entry, which is set to the **kernelopts** variable. This variable is defined in the **/boot/grub2/grubenv** configuration file.

IMPORTANT

On GRUB2 systems:

- If the kernel command-line parameters are modified for all boot entries, the **grubby** utility updates the **kernelopts** variable in the **/boot/grub2/grubenv** file.
- If kernel command-line parameters are modified for a single boot entry, the **kernelopts** variable is expanded, the kernel parameters are modified, and the resulting value is stored in the respective boot entry's **/boot/loader/entries/<RELEVANT_KERNEL_BOOT_ENTRY.conf>** file.

On zIPL systems:

- **grubby** modifies and stores the kernel command-line parameters of an individual kernel boot entry in the **/boot/loader/entries/<*ENTRY*>.conf** file.

Additional resources

- Understanding kernel command-line parameters
- **grubby(8)** and **zipl(8)** manual pages
- *grubby tool*

4.6. CHANGING KERNEL COMMAND-LINE PARAMETERS TEMPORARILY AT BOOT TIME

The following procedure allows you to make temporary changes to a Kernel Menu Entry by changing the kernel parameters only during a single boot process.

Procedure

1. Select the kernel you want to start when the GRUB 2 boot menu appears and press the **e** key to edit the kernel parameters.
2. Find the kernel command line by moving the cursor down. The kernel command line starts with **linux** on 64-Bit IBM Power Series and x86-64 BIOS-based systems, or **linuxefi** on UEFI systems.
3. Move the cursor to the end of the line.

   NOTE

   Press **Ctrl**+**a** to jump to the start of the line and **Ctrl**+**e** to jump to the end of the line. On some systems, **Home** and **End** keys might also work.

4. Edit the kernel parameters as required. For example, to run the system in emergency mode, add the *emergency* parameter at the end of the **linux** line:

   ```
   linux   ($root)/vmlinuz-4.18.0-348.12.2.el8_5.x86_64 root=/dev/mapper/rhel-root ro
   crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap
   rhgb quiet emergency
   ```

   To enable the system messages, remove the **rhgb** and **quiet** parameters.

5. Press **Ctrl**+**x** to boot with the selected kernel and the modified command line parameters.

IMPORTANT

Press **Esc** key to leave command line editing and it will drop all the user made changes.

NOTE

This procedure applies only for a single boot and does not persistently make the changes.

4.7. CONFIGURING GRUB SETTINGS TO ENABLE SERIAL CONSOLE CONNECTION

The serial console is beneficial when you need to connect to a headless server or an embedded system and the network is down. Or when you need to avoid security rules and obtain login access on a different system.

You need to configure some default GRUB settings to use the serial console connection.

Prerequisites

- You have root permissions.

Procedure

1. Add the following two lines to the **/etc/default/grub** file:

   ```
   GRUB_TERMINAL="serial"
   GRUB_SERIAL_COMMAND="serial --speed=9600 --unit=0 --word=8 --parity=no --stop=1"
   ```

   The first line disables the graphical terminal. The **GRUB_TERMINAL** key overrides values of **GRUB_TERMINAL_INPUT** and **GRUB_TERMINAL_OUTPUT** keys.

   The second line adjusts the baud rate (**--speed**), parity and other values to fit your environment and hardware. Note that a much higher baud rate, for example 115200, is preferable for tasks such as following log files.

2. Update the GRUB configuration file.

   - On BIOS-based machines:

     ```
     # grub2-mkconfig -o /boot/grub2/grub.cfg
     ```

   - On UEFI-based machines:

```
# grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
```

3. Reboot the system for the changes to take effect.

CHAPTER 5. CONFIGURING KERNEL PARAMETERS AT RUNTIME

As a system administrator, you can modify many facets of the Red Hat Enterprise Linux kernel's behavior at runtime. This section describes how to configure kernel parameters at runtime by using the **sysctl** command and by modifying the configuration files in the **/etc/sysctl.d/** and **/proc/sys/** directories.

5.1. WHAT ARE KERNEL PARAMETERS

Kernel parameters are tunable values which you can adjust while the system is running. There is no requirement to reboot or recompile the kernel for changes to take effect.

It is possible to address the kernel parameters through:

- The **sysctl** command
- The virtual file system mounted at the **/proc/sys/** directory
- The configuration files in the **/etc/sysctl.d/** directory

Tunables are divided into classes by the kernel subsystem. Red Hat Enterprise Linux has the following tunable classes:

Table 5.1. Table of sysctl classes

Tunable class	Subsystem
abi	Execution domains and personalities
crypto	Cryptographic interfaces
debug	Kernel debugging interfaces
dev	Device-specific information
fs	Global and specific file system tunables
kernel	Global kernel tunables
net	Network tunables
sunrpc	Sun Remote Procedure Call (NFS)
user	User Namespace limits
vm	Tuning and management of memory, buffers, and cache

IMPORTANT

Configuring kernel parameters on a production system requires careful planning. Unplanned changes may render the kernel unstable, requiring a system reboot. Verify that you are using valid options before changing any kernel values.

Additional resources

- **sysctl(8)**, and **sysctl.d(5)** manual pages

5.2. CONFIGURING KERNEL PARAMETERS TEMPORARILY WITH SYSCTL

The following procedure describes how to use the **sysctl** command to temporarily set kernel parameters at runtime. The command is also useful for listing and filtering tunables.

Prerequisites

- Root permissions

Procedure

1. To list all parameters and their values, use the following:

```
# sysctl -a
```

NOTE

The **# sysctl -a** command displays kernel parameters, which can be adjusted at runtime and at boot time.

2. To configure a parameter temporarily, use the command as in the following example:

```
# sysctl <TUNABLE_CLASS>.<PARAMETER>=<TARGET_VALUE>
```

The sample command above changes the parameter value while the system is running. The changes take effect immediately, without a need for restart.

NOTE

The changes return back to default after your system reboots.

Additional resources

- **sysctl(8)** manual page
- Configuring kernel parameters permanently with sysctl
- Using configuration files in /etc/sysctl.d/ to adjust kernel parameters

5.3. CONFIGURING KERNEL PARAMETERS PERMANENTLY WITH SYSCTL

The following procedure describes how to use the **sysctl** command to permanently set kernel parameters.

Prerequisites

- Root permissions

Procedure

1. To list all parameters, use the following:

   ```
   # sysctl -a
   ```

   The command displays all kernel parameters that can be configured at runtime.

2. To configure a parameter permanently:

   ```
   # sysctl -w <TUNABLE_CLASS>.<PARAMETER>=<TARGET_VALUE> >> /etc/sysctl.conf
   ```

   The sample command changes the tunable value and writes it to the **/etc/sysctl.conf** file, which overrides the default values of kernel parameters. The changes take effect immediately and persistently, without a need for restart.

NOTE

To permanently modify kernel parameters you can also make manual changes to the configuration files in the **/etc/sysctl.d/** directory.

Additional resources

- **sysctl(8)** and **sysctl.conf(5)** manual pages
- Using configuration files in /etc/sysctl.d/ to adjust kernel parameters

5.4. USING CONFIGURATION FILES IN /ETC/SYSCTL.D/ TO ADJUST KERNEL PARAMETERS

The following procedure describes how to manually modify configuration files in the **/etc/sysctl.d/** directory to permanently set kernel parameters.

Prerequisites

- Root permissions

Procedure

1. Create a new configuration file in **/etc/sysctl.d/**:

   ```
   # vim /etc/sysctl.d/<some_file.conf>
   ```

2. Include kernel parameters, one per line, as follows:

```
<TUNABLE_CLASS>.<PARAMETER>=<TARGET_VALUE>
<TUNABLE_CLASS>.<PARAMETER>=<TARGET_VALUE>
```

3. Save the configuration file.

4. Reboot the machine for the changes to take effect.

   - Alternatively, to apply changes without rebooting, execute:

     ```
     # sysctl -p /etc/sysctl.d/<some_file.conf>
     ```

     The command enables you to read values from the configuration file, which you created earlier.

Additional resources

- **sysctl(8)**, **sysctl.d(5)** manual pages

5.5. CONFIGURING KERNEL PARAMETERS TEMPORARILY THROUGH /PROC/SYS/

The following procedure describes how to set kernel parameters temporarily through the files in the virtual file system **/proc/sys/** directory.

Prerequisites

- Root permissions

Procedure

1. Identify a kernel parameter you want to configure:

   ```
   # ls -l /proc/sys/<TUNABLE_CLASS>/
   ```

   The writable files returned by the command can be used to configure the kernel. The files with read-only permissions provide feedback on the current settings.

2. Assign a target value to the kernel parameter:

   ```
   # echo <TARGET_VALUE> > /proc/sys/<TUNABLE_CLASS>/<PARAMETER>
   ```

   The command makes configuration changes that will disappear once the system is restarted.

3. Optionally, verify the value of the newly set kernel parameter:

   ```
   # cat /proc/sys/<TUNABLE_CLASS>/<PARAMETER>
   ```

Additional resources

- Configuring kernel parameters permanently with sysctl
- Using configuration files in /etc/sysctl.d/ to adjust kernel parameters

CHAPTER 6. MAKING TEMPORARY CHANGES TO THE GRUB MENU

You can modify GRUB menu entries or pass arguments to the kernel, which applies only to the current boot. On a selected menu entry in the boot loader menu, you can:

- display the menu entry editor interface by pressing the **e** key.
- discard any changes and reload the standard menu interface by pressing the **Esc** key.
- load the command-line interface by pressing the **c** key.
- type any relevant GRUB commands and enter them by pressing the **Enter** key.
- complete a command based on context by pressing the **Tab** key.
- move to the beginning of a line by pressing the **Ctrl**+**a** key combination.
- move to the end of a line by pressing the **Ctrl**+**e** key combination.

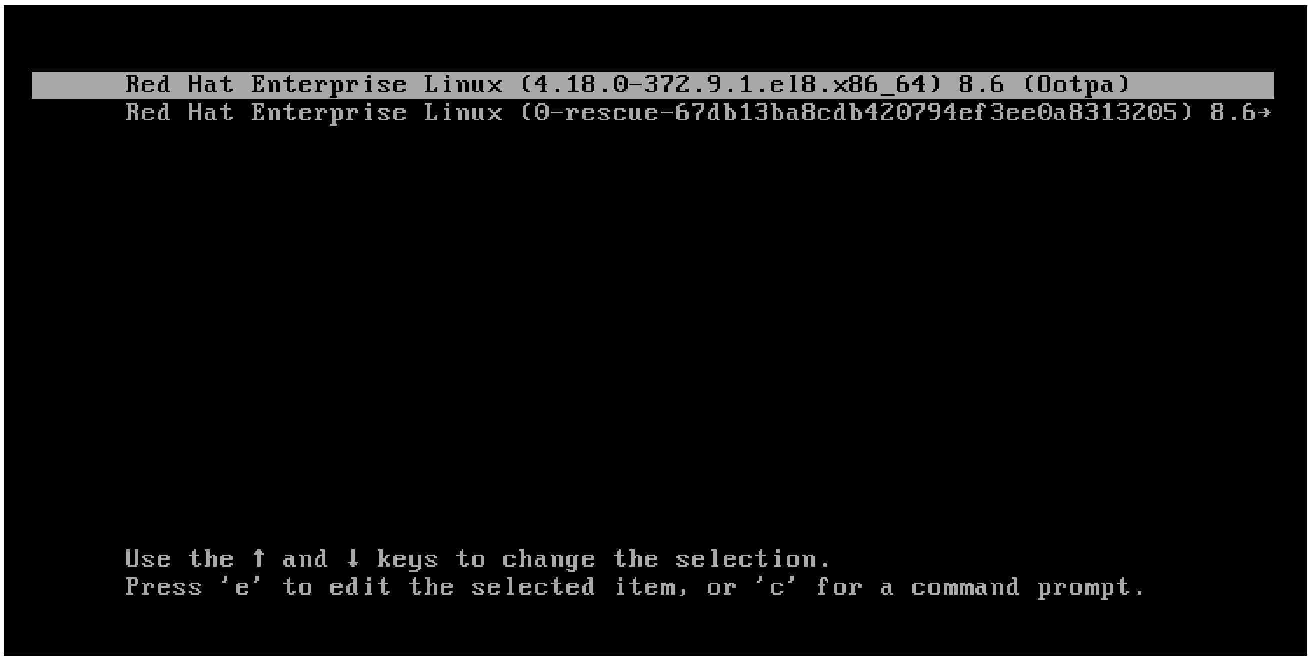
IMPORTANT

The following procedures provide instruction on making changes to a GRUB Menu during a single boot process.

6.1. INTRODUCTION TO GRUB

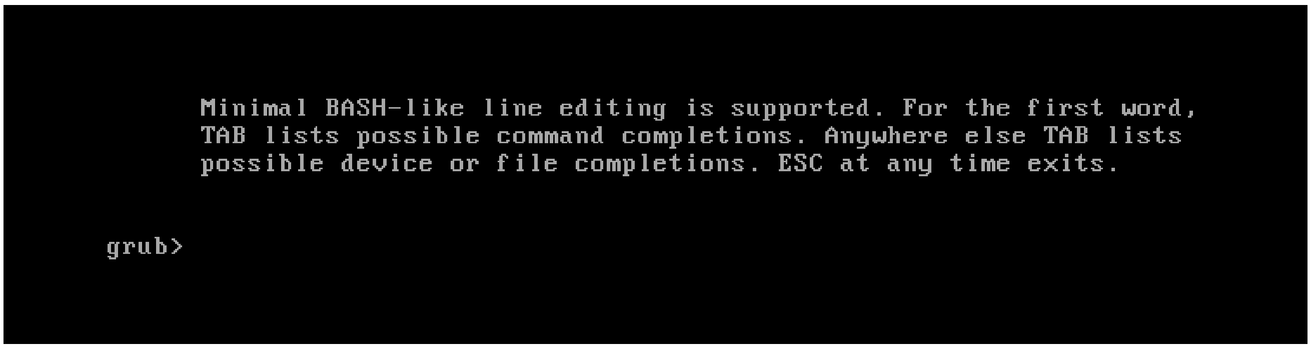
GRUB stands for the **GNU GRand Unified Bootloader**. With GRUB, you can select an operating system or kernel to be loaded at system boot time. Also, you can pass arguments to the kernel.

When booting with GRUB, you can use either a menu interface or a command-line interface (the **GRUB command shell**). When you start the system, the menu interface appears.



```
      Red Hat Enterprise Linux (4.18.0-372.9.1.el8.x86_64) 8.6 (Ootpa)
      Red Hat Enterprise Linux (0-rescue-67db13ba8cdb420794ef3ee0a8313205) 8.6→

      Use the ↑ and ↓ keys to change the selection.
      Press 'e' to edit the selected item, or 'c' for a command prompt.
```

You can switch to the command-line interface by pressing the **c** key.



```
        Minimal BASH-like line editing is supported. For the first word,
        TAB lists possible command completions. Anywhere else TAB lists
        possible device or file completions. ESC at any time exits.


    grub>
```

You can return to the menu interface by typing exit and pressing the Enter key.

GRUB BLS files

The boot loader menu entries are defined as Boot Loader Specification (**BLS**) files. This file format manages boot loader configuration for each boot option in a drop-in directory, without manipulating boot loader configuration files. The **grubby** utility can edit these **BLS** files.

GRUB configuration file

The **/boot/grub2/grub.cfg** configuration file does not define the menu entries.

Additional resources

- Boot menu
- Introduction to bootloader specification

6.2. INTRODUCTION TO BOOTLOADER SPECIFICATION

The BootLoader Specification (BLS) defines a scheme and the file format to manage the bootloader configuration for each boot option in the drop-in directory without the need to manipulate the bootloader configuration files. Unlike earlier approaches, each boot entry is now represented by a separate configuration file in the drop-in directory. The drop-in directory extends its configuration without having the need to edit or regenerate the configuration files. The BLS extends this concept for the boot menu entries.

Using BLS, you can manage the bootloader menu options by adding, removing, or editing individual boot entry files in a directory. This makes the kernel installation process significantly simpler and consistent across the different architectures.

The **grubby** tool is a thin wrapper script around the BLS and it supports the same **grubby** arguments and options. It runs the **dracut** to create an initial ramdisk image. With this setup, the core bootloader configuration files are static and are not modified after kernel installation.

This premise is particularly relevant in RHEL 8, because the same bootloader is not used in all architectures. GRUB is used in most of them such as the 64-bit ARM, but little-endian variants of IBM Power Systems with Open Power Abstraction Layer (OPAL) uses **Petitboot** and the IBM Z architecture uses **zipl**.

Additional Resources

- Section 4.2, “What grubby is”
- Section 4.3, “What boot entries are”

- the **grubby(8)** manual page

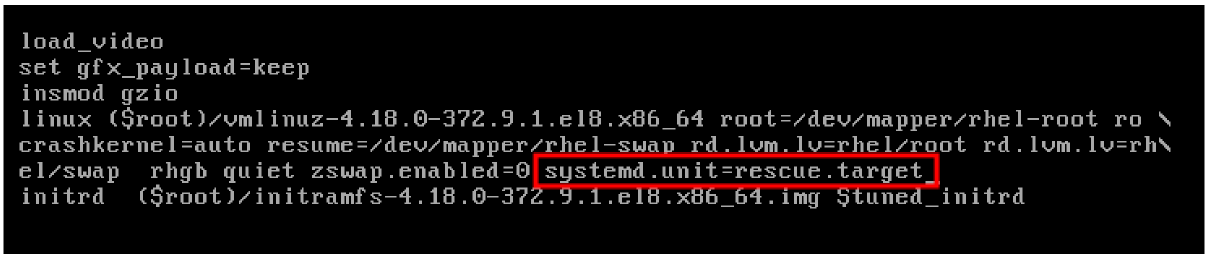
6.3. BOOTING TO RESCUE MODE

Rescue mode provides a convenient single-user environment in which you can repair your system in situations when it is unable to complete a normal booting process. In rescue mode, the system attempts to mount all local file systems and start some important system services. However, it does not activate network interfaces or allow more users to be logged into the system at the same time.

Procedure

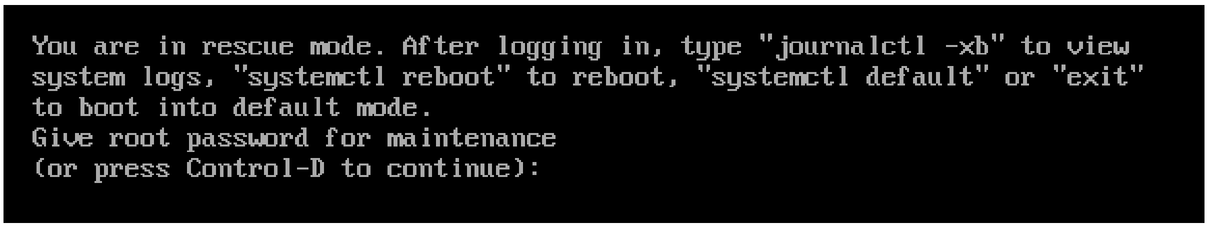
1. On the GRUB boot screen, press the **e** key for edit.
2. Add the following parameter at the end of the **linux** line:

```
systemd.unit=rescue.target
```



```
load_video
set gfx_payload=keep
insmod gzio
linux ($root)/vmlinuz-4.18.0-372.9.1.el8.x86_64 root=/dev/mapper/rhel-root ro \
crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rh\
el/swap  rhgb quiet zswap.enabled=0 systemd.unit=rescue.target_
initrd  ($root)/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd
```

3. Press **Ctrl**+**x** to boot to rescue mode.



```
You are in rescue mode. After logging in, type "journalctl -xb" to view
system logs, "systemctl reboot" to reboot, "systemctl default" or "exit"
to boot into default mode.
Give root password for maintenance
(or press Control-D to continue):
```

6.4. BOOTING TO EMERGENCY MODE

Emergency mode provides the most minimal environment possible in which you can repair your system even in situations when the system is unable to enter rescue mode.

In emergency mode, the system:

- mounts the **root** file system only for reading
- starts a few essential services

However, the system **does not**:

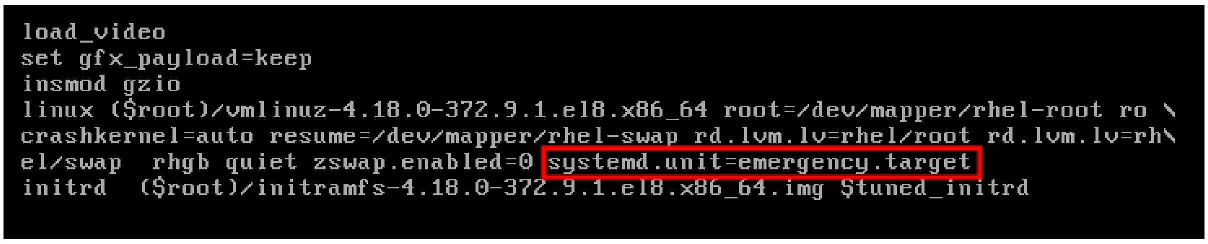
- attempt to mount any other local file systems
- activate network interfaces

Procedure

1. On the GRUB boot screen, press the **e** key for edit.

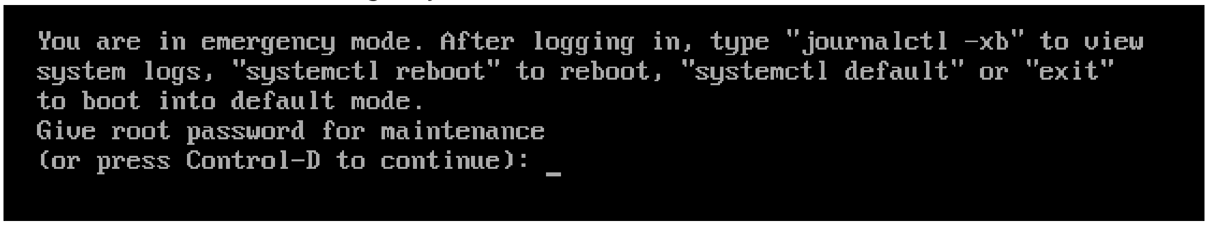
2. Add the following parameter at the end of the **linux** line:

```
systemd.unit=emergency.target
```



```
load_video
set gfx_payload=keep
insmod gzio
linux ($root)/vmlinuz-4.18.0-372.9.1.el8.x86_64 root=/dev/mapper/rhel-root ro \
crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rh\
el/swap  rhgb quiet zswap.enabled=0 systemd.unit=emergency.target
initrd  ($root)/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd
```

3. Press **Ctrl**+**x** to boot to emergency mode.



```
You are in emergency mode. After logging in, type "journalctl -xb" to view
system logs, "systemctl reboot" to reboot, "systemctl default" or "exit"
to boot into default mode.
Give root password for maintenance
(or press Control-D to continue): _
```

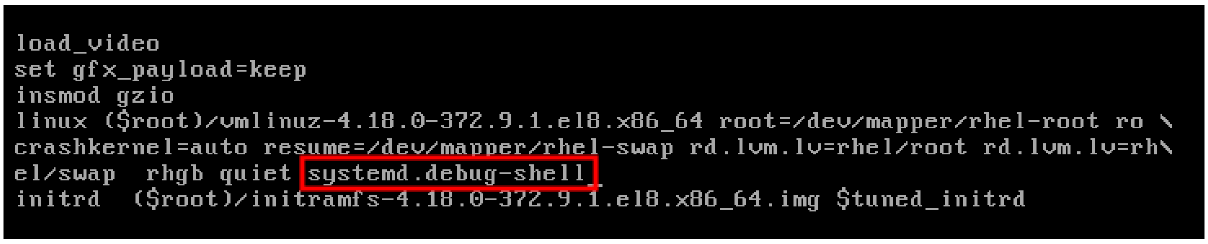
6.5. BOOTING TO THE DEBUG SHELL

The **systemd** debug shell provides a shell very early in the start-up process. Once in the debug shell, you can use the **systemctl** commands, such as **systemctl list-jobs** and **systemctl list-units**, to search for the cause of **systemd** related boot-up problems.

Procedure

1. On the GRUB boot screen, press the **e** key for edit.
2. Add the following parameter at the end of the **linux** line:

```
systemd.debug-shell
```



```
load_video
set gfx_payload=keep
insmod gzio
linux ($root)/vmlinuz-4.18.0-372.9.1.el8.x86_64 root=/dev/mapper/rhel-root ro \
crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rh\
el/swap  rhgb quiet systemd.debug-shell
initrd  ($root)/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd
```

3. Optionally add the **debug** option.

> **NOTE**
>
> Adding the **debug** option to the kernel command line increases the number of log messages. For **systemd**, the kernel command-line option **debug** is now a shortcut for **systemd.log_level=debug**.

4. Press **Ctrl**+**x** to boot to the debug shell.

> **WARNING**
>
> Permanently enabling the debug shell is a security risk because no authentication is required to use it. Disable it when the debugging session has ended.

6.6. CONNECTING TO THE DEBUG SHELL

During the boot process, the **systemd-debug-generator** configures the debug shell on TTY9.

Prerequisites

- You have booted to the debug shell successfully. See Booting to the debug shell.

Procedure

1. Press **Ctrl**+**Alt**+**F9** to connect to the debug shell.
   If you work with a virtual machine, sending this key combination requires support from the virtualization application. For example, if you use **Virtual Machine Manager**, select **Send Key → Ctrl+Alt+F9** from the menu.

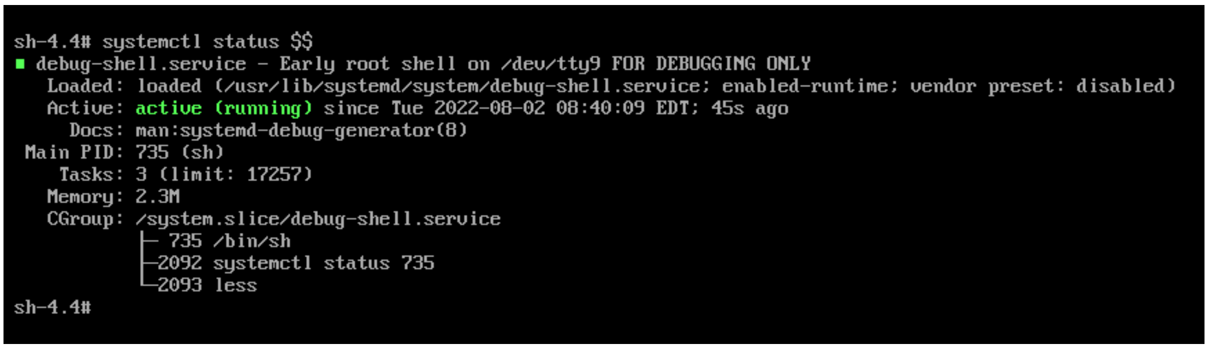
2. The debug shell does not require authentication, therefore you can see a prompt similar to the following on TTY9:

```
sh-4.4#
```

Verification steps

- Enter a command as follows:

  ```
  sh-4.4# systemctl status $$
  ```

  ```
  sh-4.4# systemctl status $$
  ● debug-shell.service - Early root shell on /dev/tty9 FOR DEBUGGING ONLY
     Loaded: loaded (/usr/lib/systemd/system/debug-shell.service; enabled-runtime; vendor preset: disabled)
     Active: active (running) since Tue 2022-08-02 08:40:09 EDT; 45s ago
       Docs: man:systemd-debug-generator(8)
   Main PID: 735 (sh)
      Tasks: 3 (limit: 17257)
     Memory: 2.3M
     CGroup: /system.slice/debug-shell.service
             ├─ 735 /bin/sh
             ├─2092 systemctl status 735
             └─2093 less
  sh-4.4#
  ```

- To return to the default shell, if the boot succeeded, press **Ctrl**+**Alt**+**F1**.

Additional resources

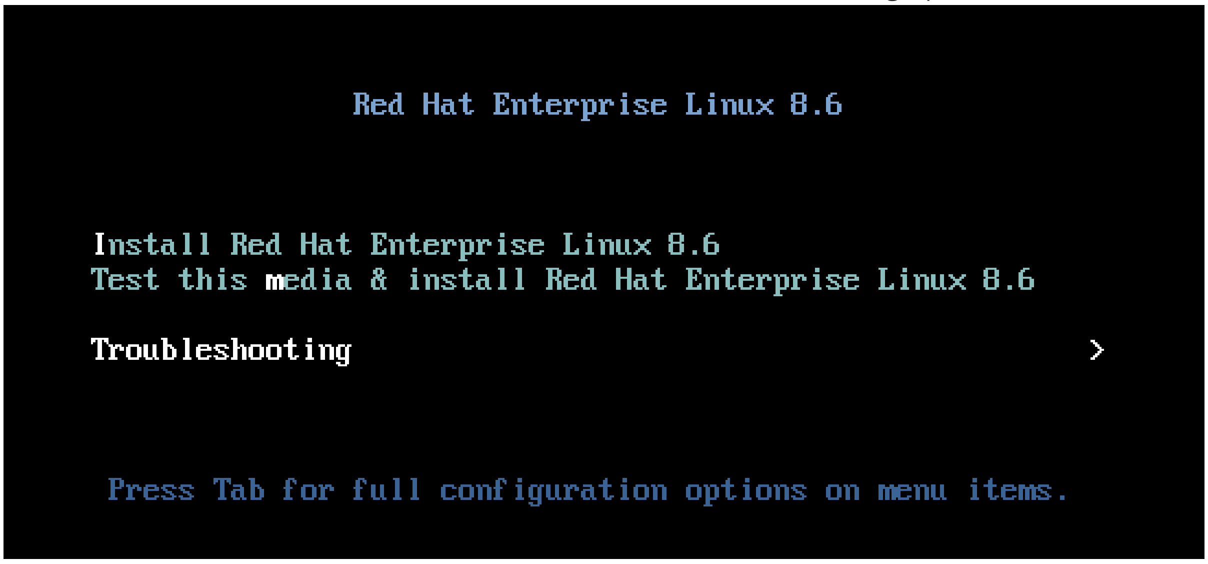
- The **systemd-debug-generator(8)** manual page

6.7. RESETTING THE ROOT PASSWORD USING AN INSTALLATION DISK

In case you forget or lose the **root** password, you can reset it.

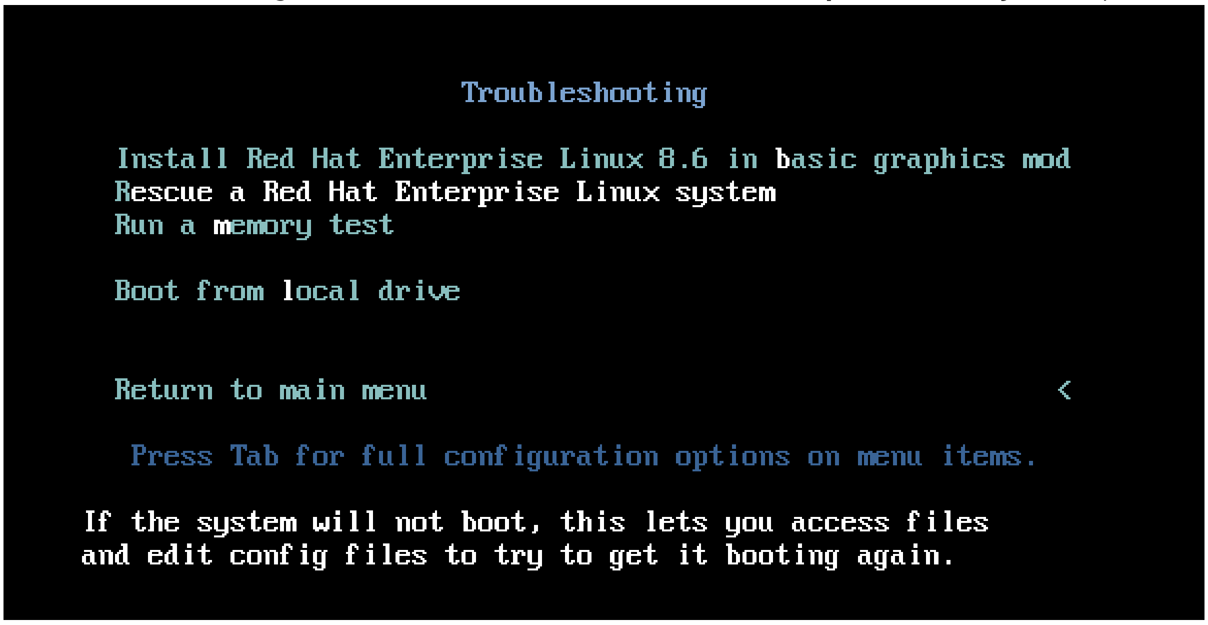
Procedure

1. Boot the host from an installation source.
2. In the boot menu for the installation media, select the **Troubleshooting** option.



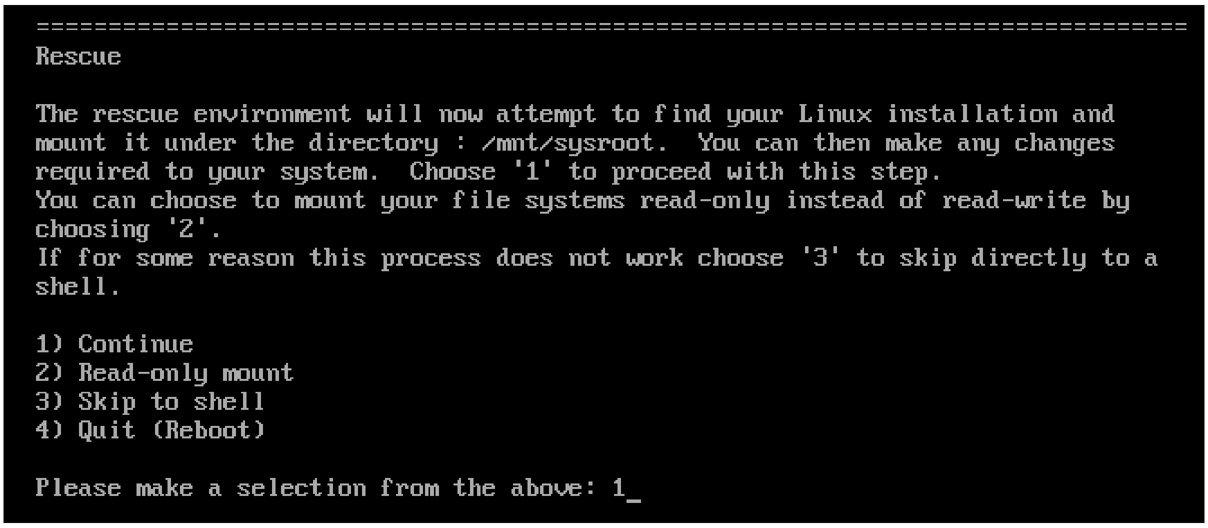
```
Red Hat Enterprise Linux 8.6

Install Red Hat Enterprise Linux 8.6
Test this media & install Red Hat Enterprise Linux 8.6

Troubleshooting                                    >

Press Tab for full configuration options on menu items.
```

3. In the Troubleshooting menu, select the **Rescue a Red Hat Enterprise Linux system** option.



```
Troubleshooting

Install Red Hat Enterprise Linux 8.6 in basic graphics mod
Rescue a Red Hat Enterprise Linux system
Run a memory test

Boot from local drive

Return to main menu                                 <

Press Tab for full configuration options on menu items.

If the system will not boot, this lets you access files
and edit config files to try to get it booting again.
```

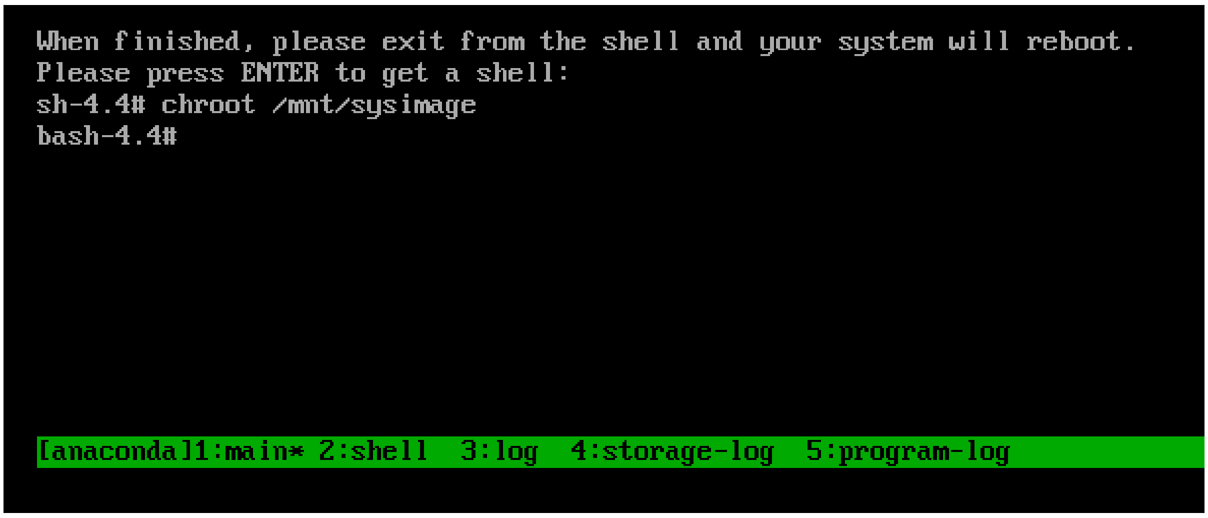
4. At the Rescue menu, select **1** and press the **Enter** key to continue.



```
================================================================================
Rescue

The rescue environment will now attempt to find your Linux installation and
mount it under the directory : /mnt/sysroot.  You can then make any changes
required to your system.  Choose '1' to proceed with this step.
You can choose to mount your file systems read-only instead of read-write by
choosing '2'.
If for some reason this process does not work choose '3' to skip directly to a
shell.

1) Continue
2) Read-only mount
3) Skip to shell
4) Quit (Reboot)

Please make a selection from the above: 1_
```

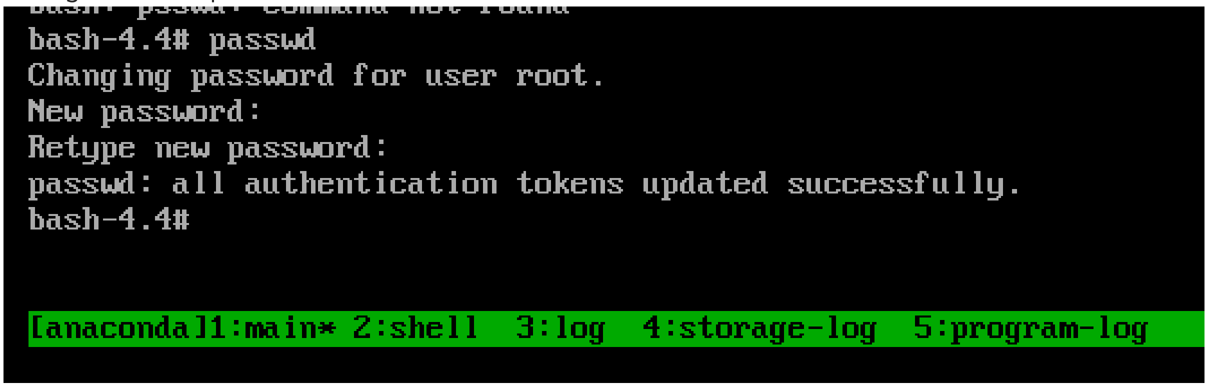
5. Change the file system **root** as follows:

```
sh-4.4# chroot /mnt/sysimage
```



```
When finished, please exit from the shell and your system will reboot.
Please press ENTER to get a shell:
sh-4.4# chroot /mnt/sysimage
bash-4.4#









[anaconda]1:main* 2:shell  3:log  4:storage-log  5:program-log
```

6. Enter the **passwd** command and follow the instructions displayed on the command line to change the **root** password.



```
bash: psswd: command not found
bash-4.4# passwd
Changing password for user root.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
bash-4.4#


[anaconda]1:main* 2:shell  3:log  4:storage-log  5:program-log
```

7. Remove the **autorelable** file to prevent a time consuming SELinux relabel of the disk:

```
sh-4.4# rm -f /.autorelabel
```

8. Enter the **exit** command to exit the **chroot** environment.

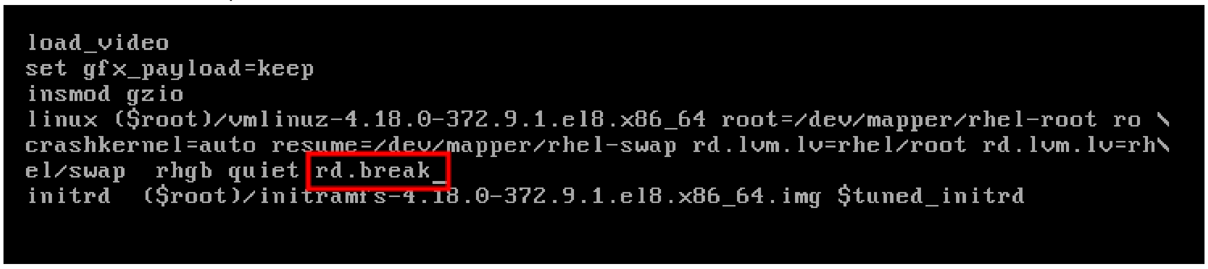
9. Enter the **exit** command again to resume the initialization and finish the system boot.

6.8. RESETTING THE ROOT PASSWORD USING RD.BREAK

In case you forget or lose the **root** password, you can reset it.

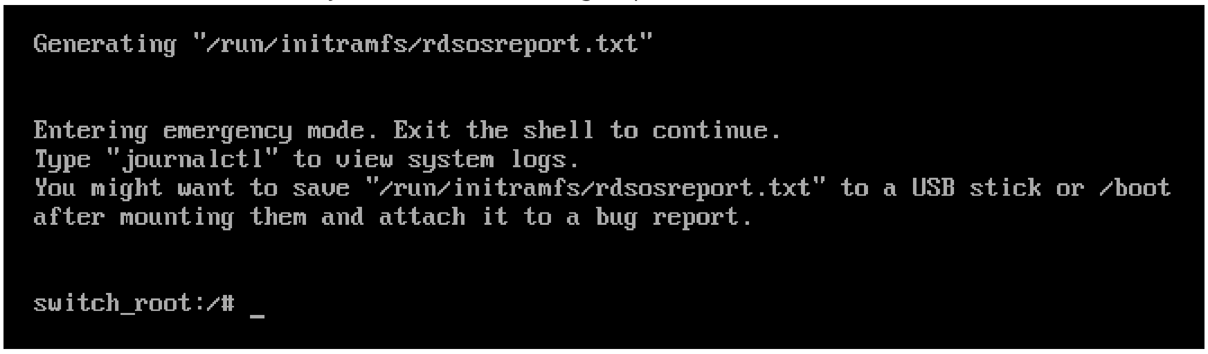
Procedure

1. Start the system and, on the GRUB boot screen, press the **e** key for edit.
2. Add the **rd.break** parameter at the end of the **linux** line:



```
load_video
set gfx_payload=keep
insmod gzio
linux ($root)/vmlinuz-4.18.0-372.9.1.el8.x86_64 root=/dev/mapper/rhel-root ro \
crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rh\
el/swap  rhgb quiet rd.break_
initrd  ($root)/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd
```

3. Press **Ctrl**+**x** to boot the system with the changed parameters.



```
Generating "/run/initramfs/rdsosreport.txt"


Entering emergency mode. Exit the shell to continue.
Type "journalctl" to view system logs.
You might want to save "/run/initramfs/rdsosreport.txt" to a USB stick or /boot
after mounting them and attach it to a bug report.


switch_root:/# _
```

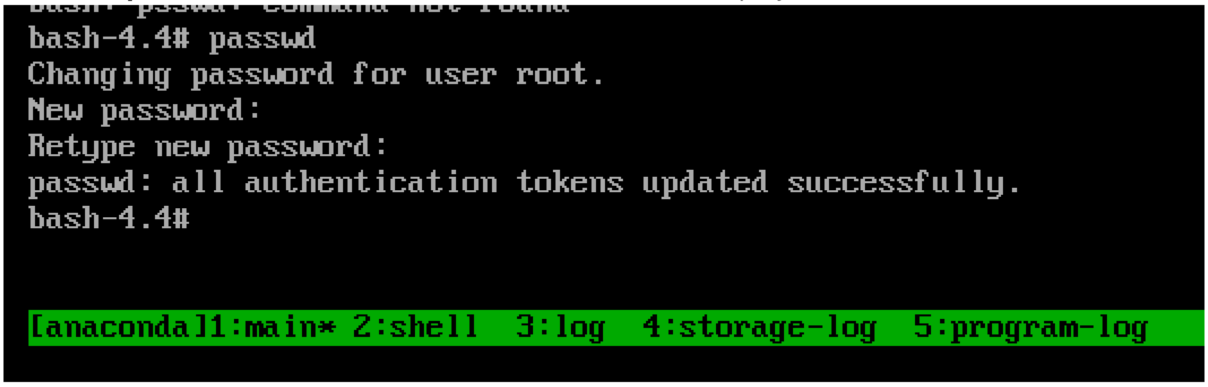
4. Remount the file system as writable.

```
switch_root:/# mount -o remount,rw /sysroot
```

5. Change the file system's **root**.

```
switch_root:/# chroot /sysroot
```

6. Enter the **passwd** command and follow the instructions displayed on the command line.



```
bash-4.4# passwd
Changing password for user root.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
bash-4.4#


[anaconda]1:main* 2:shell  3:log  4:storage-log  5:program-log
```

7. Relabel all files on the next system boot.

```
sh-4.4# touch /.autorelabel
```

8. Remount the file system as **read only**:

```
sh-4.4# mount -o remount,ro /
```

9. Enter the **exit** command to exit the **chroot** environment.

10. Enter the **exit** command again to resume the initialization and finish the system boot.

> **NOTE**
>
> The SELinux relabeling process can take a long time. A system reboot occurs automatically when the process is complete.

TIP

You can omit the time consuming SELinux relabeling process by adding the **enforcing=0** option.

Procedure

1. When adding the **rd.break** parameter at the end of the **linux** line, append **enforcing=0** as well.

```
rd.break enforcing=0
```

2. Restore the **/etc/shadow** file's SELinux security context.

```
# restorecon /etc/shadow
```

3. Turn SELinux policy enforcement back on and verify that it is on.

```
# setenforce 1
# getenforce
Enforcing
```

Note that if you added the **enforcing=0** option in step 3 you can omit entering the **touch /.autorelabel** command in step 8.

6.9. ADDITIONAL RESOURCES

- The **/usr/share/doc/grub2-common** directory.
- The **info grub2** command.

CHAPTER 7. MAKING PERSISTENT CHANGES TO THE GRUB BOOT LOADER

This chapter provides a list of procedures using the **grubby** tool that lead to persistent changes in GRUB.

7.1. PREREQUISITES

- You have successfully installed RHEL on your system.
- You have root permission.

7.2. LISTING THE DEFAULT KERNEL

By listing the default kernel, you can find the file name and the index number of the default kernel to make permanent changes to the GRUB boot loader.

Procedure

- To find out the file name of the default kernel, enter:

```
# grubby --default-kernel
/boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
```

- To find out the index number of the default kernel, enter:

```
# grubby --default-index
0
```

7.3. VIEWING THE GRUB MENU ENTRY FOR A KERNEL

You can list all the kernel menu entries or view the GRUB menu entry for a specific kernel.

Procedure

- To list all kernel menu entries, enter:

```
# grubby --info=ALL
index=0
kernel="/boot/vmlinuz-4.18.0-372.9.1.el8.x86_64"
args="ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap rhgb quiet $tuned_params zswap.enabled=1"
root="/dev/mapper/rhel-root"
initrd="/boot/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd"
title="Red Hat Enterprise Linux (4.18.0-372.9.1.el8.x86_64) 8.6 (Ootpa)"
id="67db13ba8cdb420794ef3ee0a8313205-4.18.0-372.9.1.el8.x86_64"
index=1
kernel="/boot/vmlinuz-0-rescue-67db13ba8cdb420794ef3ee0a8313205"
args="ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap rhgb quiet"
root="/dev/mapper/rhel-root"
initrd="/boot/initramfs-0-rescue-67db13ba8cdb420794ef3ee0a8313205.img"
```

```
title="Red Hat Enterprise Linux (0-rescue-67db13ba8cdb420794ef3ee0a8313205) 8.6
(Ootpa)"
id="67db13ba8cdb420794ef3ee0a8313205-0-rescue"
```

- To view the GRUB menu entry for a specific kernel, enter:

```
# grubby --info /boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
grubby --info /boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
index=0
kernel="/boot/vmlinuz-4.18.0-372.9.1.el8.x86_64"
args="ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap rhgb quiet $tuned_params zswap.enabled=1"
root="/dev/mapper/rhel-root"
initrd="/boot/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd"
title="Red Hat Enterprise Linux (4.18.0-372.9.1.el8.x86_64) 8.6 (Ootpa)"
id="67db13ba8cdb420794ef3ee0a8313205-4.18.0-372.9.1.el8.x86_64"
```

NOTE

Try tab completion to see available kernels within the **/boot** directory.

7.4. ADDING AND REMOVING ARGUMENTS FROM A GRUB MENU ENTRY

You can add, remove, or simultaneously add and remove arguments from the GRUB Menu.

Procedure

- To add arguments to a GRUB menu entry, use the **--update-kernel** option in combination with **--args**. For example, following command adds a serial console:

```
# grubby --args=console=ttyS0,115200 --update-kernel /boot/vmlinuz-4.18.0-
372.9.1.el8.x86_64
```

  The console arguments are attached to the end of the line, the new console will take precedence over any other configured consoles.

- To remove arguments from a GRUB menu entry, use the **--update-kernel** option in combination with **--remove-args**. For example:

```
# grubby --remove-args="rhgb quiet" --update-kernel /boot/vmlinuz-4.18.0-
372.9.1.el8.x86_64
```

  This command removes the Red Hat graphical boot argument and enables log messages, that is verbose mode.

- To add and remove arguments simultaneously, enter:

```
# grubby --remove-args="rhgb quiet" --args=console=ttyS0,115200 --update-kernel
/boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
```

Verification steps

- To review the permanent changes you have made, enter:

```
# grubby --info /boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
index=0
kernel="/boot/vmlinuz-4.18.0-372.9.1.el8.x86_64"
args="ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap $tuned_params zswap.enabled=1 console=ttyS0,115200"
root="/dev/mapper/rhel-root"
initrd="/boot/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd"
title="Red Hat Enterprise Linux (4.18.0-372.9.1.el8.x86_64) 8.6 (Ootpa)"
id="67db13ba8cdb420794ef3ee0a8313205-4.18.0-372.9.1.el8.x86_64"
```

7.5. ADDING A NEW BOOT ENTRY

You can add a new boot entry to the bootloader menu entries.

Procedure

1. Copy all the kernel arguments from your default kernel to this new kernel entry.

```
# grubby --add-kernel=new_kernel --title="entry_title" --initrd="new_initrd" --copy-
default
```

2. Get the list of available boot entries.

```
# ls -l /boot/loader/entries/*
-rw-r--r--. 1 root root 408 May 27 06:18
/boot/loader/entries/67db13ba8cdb420794ef3ee0a8313205-0-rescue.conf
-rw-r--r--. 1 root root 536 Jun 30 07:53
/boot/loader/entries/67db13ba8cdb420794ef3ee0a8313205-4.18.0-372.9.1.el8.x86_64.conf
-rw-r--r--  1 root root 336 Aug 15 15:12
/boot/loader/entries/d88fa2c7ff574ae782ec8c4288de4e85-4.18.0-193.el8.x86_64.conf
```

3. Create a new boot entry. For example, for the *4.18.0-193.el8.x86_64* kernel, issue the command as follows:

```
# grubby --grub2 --add-kernel=/boot/vmlinuz-4.18.0-193.el8.x86_64 --title="Red Hat
Enterprise 8 Test" --initrd=/boot/initramfs-4.18.0-193.el8.x86_64.img --copy-default
```

Verification

- Verify that the newly added boot entry is listed among the available boot entries.

```
# ls -l /boot/loader/entries/*
-rw-r--r--. 1 root root 408 May 27 06:18
/boot/loader/entries/67db13ba8cdb420794ef3ee0a8313205-0-rescue.conf
-rw-r--r--. 1 root root 536 Jun 30 07:53
/boot/loader/entries/67db13ba8cdb420794ef3ee0a8313205-4.18.0-372.9.1.el8.x86_64.conf
-rw-r--r-- 1 root root 287 Aug 16 15:17
/boot/loader/entries/d88fa2c7ff574ae782ec8c4288de4e85-4.18.0-
193.el8.x86_64.0~custom.conf
-rw-r--r--  1 root root 287 Aug 16 15:29
/boot/loader/entries/d88fa2c7ff574ae782ec8c4288de4e85-4.18.0-193.el8.x86_64.conf
```

7.6. CHANGING THE DEFAULT BOOT ENTRY WITH GRUBBY

With the **grubby** tool, you can change the default boot entry.

Procedure

- To make a persistent change in the kernel designated as the default kernel, enter:

```
# grubby --set-default /boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
The default is /boot/loader/entries/67db13ba8cdb420794ef3ee0a8313205-4.18.0-
372.9.1.el8.x86_64.conf with index 0 and kernel /boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
```

7.7. UPDATING ALL KERNEL MENUS WITH THE SAME ARGUMENTS

You can add the same kernel boot arguments to all the kernel menu entries.

Procedure

- To add the same kernel boot arguments to all the kernel menu entries, attach the **--update-kernel=ALL** parameter. For example, this command adds a serial console to all kernels:

  ```
  # grubby --update-kernel=ALL --args=console=ttyS0,115200
  ```

  NOTE

  The **--update-kernel** parameter also accepts **DEFAULT** or a comma-separated list of kernel index numbers.

7.8. ADDITIONAL RESOURCES

- The **/usr/share/doc/grub2-common** directory.
- The **info grub2** command.

CHAPTER 8. BUILDING A CUSTOMIZED BOOT MENU

You can build a boot menu containing specific entries or change the order of the entries. For such a task, you can use GRUB, **grubby**, and Boot Loader Specification (**BLS**) files.

The following sections provide information on using GRUB and **grubby** to do basic customization of the boot menu.

8.1. THE GRUB CONFIGURATION FILE

This section provides details on the boot loader configuration file that is **/boot/grub2/grub.cfg** on BIOS-based machines and **/boot/efi/EFI/redhat/grub.cfg** on UEFI-based machines.

GRUB scripts search the user's computer and build a boot menu based on what operating systems the scripts find. To reflect the latest system boot options, the boot menu is rebuilt automatically when the kernel is updated or a new kernel is added.

GRUB uses a series of scripts to build the menu; these are located in the **/etc/grub.d/** directory. The following files are included:

- **00_header**, which loads GRUB settings from the **/etc/default/grub** file.
- **01_users**, which reads the root password from the **user.cfg** file.
- **10_linux**, which locates kernels in the default partition of Red Hat Enterprise Linux.
- **30_os-prober**, which builds entries for operating systems found on other partitions.
- **40_custom**, a template, which can be used to create additional menu entries.

GRUB reads scripts from the **/etc/grub.d/** directory in alphabetical order and therefore you can rename them to change the boot order of specific menu entries.

8.2. HIDING THE LIST OF BOOTABLE KERNELS

You can prevent GRUB from displaying the list of bootable kernels when the system starts up.

Procedure

1. Set the **GRUB_TIMEOUT_STYLE** option in the **/etc/default/grub** file as follows:

   ```
   GRUB_TIMEOUT_STYLE=hidden
   ```

2. Rebuild the **grub.cfg** file for the changes to take effect.

   - On BIOS-based machines, enter:

     ```
     # grub2-mkconfig -o /boot/grub2/grub.cfg
     ```

   - On UEFI-based machines, enter:

     ```
     # grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
     ```

3. Press the **Esc** key to display the list of bootable kernels when booting.

IMPORTANT

Do not set **GRUB_TIMEOUT** to *0* in the **/etc/default/grub** file to hide the list of bootable kernels. With such a setting, the system always boots immediately on the default menu entry, and if the default kernel fails to boot, it is not possible to boot any previous kernel.

8.3. CHANGING THE DEFAULT BOOT ENTRY WITH THE GRUB CONFIGURATION FILE

You can specify the default kernel package type, and thus change the default boot entry.

Procedure

1. Specify which operating system or kernel should be loaded by default by passing its index to the **grub2-set-default** command. For example:

```
# grubby --set-default-index=1
The default is /boot/loader/entries/d5151aa93c444ac89e78347a1504d6c6-4.18.0-
348.el8.x86_64.conf with index 1 and kernel /boot/vmlinuz-4.18.0-348.el8.x86_64
```

   GRUB supports using a numeric value as the key for the **saved_entry** directive in **/boot/grub2/grubenv** to change the default order in which the operating systems are loaded.

   NOTE

   Index counting starts with zero; therefore, in the previous example, GRUB loads the second entry. With the next installed kernel, the index value will be overwritten.

   NOTE

   You can also use **grubby** to find indices for kernels. For more information, see Viewing the GRUB Menu Entry for a Kernel.

2. *Optional*: Force the system to always use a particular menu entry:

   a. List the available menu entries:

   ```
   # grubby --info=ALL
   ```

   b. Use the menu entry name or the number of the position of a menu entry in the list as the key to the **GRUB_DEFAULT** directive in the **/etc/default/grub** file. For example:

   ```
   GRUB_DEFAULT=example-gnu-linux
   ```

3. Rebuild the **grub.cfg** file for the changes to take effect.

   - On BIOS-based machines, enter:

   ```
   # grub2-mkconfig -o /boot/grub2/grub.cfg
   ```

   - On UEFI-based machines, enter:

```
# grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
```

CHAPTER 9. REINSTALLING GRUB

Reinstalling GRUB bootloader is a convenient way to fix certain problems usually caused by an incorrect installation of GRUB, missing files, or a broken system. You can resolve this issue by restoring the missing files and updating the boot information.

The following are the reasons to reinstall GRUB:

- Upgrading GRUB bootloader.
- Adding the boot information to another drive.
- The user requires the GRUB bootloader to control installed operating systems. However, some operating systems are installed with their own bootloaders and reinstalling GRUB returns control to the desired operating system.

NOTE

GRUB restores files only if they are not corrupted.

9.1. REINSTALLING GRUB ON BIOS-BASED MACHINES

You can reinstall GRUB using the **grub2-install** command. The following procedure updates the boot information and restores the missing files.

IMPORTANT

When you run the **grub2-install** command on an existing boot device, it overrides the existing GRUB to install the new GRUB. Hence, ensure that the system does not cause data corruption or boot crash during the installation before issuing the **grub2-install** command.

Procedure

1. Issue the **grub2-install** command with the device argument. For example, if **sda** is your device:

```
# grub2-install /dev/sda
```

2. Reboot your system for the changes to take effect.

```
# reboot
```

Additional resources

- the **grub-install(1)** man page

9.2. REINSTALLING GRUB ON UEFI-BASED MACHINES

You can reinstall GRUB using the **yum reinstall** command. The following procedure updates the boot information and restores the missing files.

IMPORTANT

Ensure that the system does not cause data corruption or boot crash during the installation before issuing the **yum reinstall** command.

Procedure

1. Issue the **yum reinstall** command with the **grub2-efi** and **shim** bootloader files.

   ```
   # yum reinstall grub2-efi shim
   ```

2. Reboot your system for the changes to take effect.

   ```
   # reboot
   ```

9.3. RESETTING GRUB

Resetting GRUB completely removes all GRUB configuration files and system settings and reinstalls the bootloader. You can reset all the configuration settings to their default values, and thus fix failures caused by corrupted files and incorrect configuration.

IMPORTANT

The following procedure will remove all the customization the user has made.

Procedure

1. Remove the configuration files.

   ```
   # rm /etc/grub.d/*
   # rm /etc/sysconfig/grub
   ```

2. Reinstall packages.

   - On BIOS-based machines, enter:

     ```
     # yum reinstall grub2-tools
     ```

   - On UEFI-based machines, enter:

     ```
     # yum reinstall grub2-efi shim grub2-tools
     ```

3. Rebuild the **grub.cfg** file for the changes to take effect.

   - On BIOS-based machines, enter:

     ```
     # grub2-mkconfig -o /boot/grub2/grub.cfg
     ```

   - On UEFI-based machines, enter:

     ```
     # grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
     ```

4. Follow Reinstalling GRUB procedure to restore GRUB on the **/boot/** partition.

CHAPTER 10. PROTECTING GRUB WITH A PASSWORD

You can protect GRUB with a password in two ways:

- Password is required for modifying menu entries but not for booting existing menu entries.
- Password is required for modifying menu entries as well as for booting existing menu entries.

10.1. SETTING PASSWORD PROTECTION ONLY FOR MODIFYING MENU ENTRIES

You can configure GRUB to support password authentication for modifying GRUB menu entries. This procedure creates a **/boot/grub2/user.cfg** file that contains the password in the hash format.

IMPORTANT

Setting a password using the **grub2-setpassword** command prevents menu entries from unauthorized modification but not from unauthorized booting.

Procedure

1. Issue the **grub2-setpassword** command as root.

```
# grub2-setpassword
```

2. Enter the password for the user and press the **Enter** key to confirm the password.

```
Enter password:
Confirm the password:
```

NOTE

The root user is defined in the **/boot/grub2/grub.cfg** file with the password changes. Therefore, modifying a boot entry during booting requires the root user name and password.

10.2. SETTING PASSWORD PROTECTION FOR MODIFYING AND BOOTING MENU ENTRIES

You can configure GRUB to prevent menu entries from unauthorized modification as well as from unauthorized booting.

WARNING

If you forget the GRUB password, you will not be able to boot the entries you have reconfigured.

Procedure

1. Open the Boot Loader Specification (**BLS**) file for boot entry you want to modify from the **/boot/loader/entries/** directory.

2. Find the line beginning with **grub_users**. This parameter passes extra arguments to **menuentry**.

3. Set the **grub_users** attribute to the user name that is allowed to boot the entry besides the superusers, by default this user is **root**. Here is a sample configuration file:

```
title Red Hat Enterprise Linux (4.18.0-221.el8.x86_64) 8.3
(Ootpa)
version 4.18.0-221.el8.x86_64
linux /vmlinuz-4.18.0-221.el8.x86_64
initrd /initramfs-4.18.0-221.el8.x86_64.img $tuned_initrd
options $kernelopts $tuned_params
id rhel-20200625210904-4.18.0-221.el8.x86_64
grub_users root
grub_arg --unrestricted
grub_class kernel
```

4. Save and close the **BLS** file.

NOTE

If you want to protect all the menu entries from booting, you can directly set the **grub_users** attribute. For example, if root is the user:

```
# grub2-editenv - set grub_users="root"
```

CHAPTER 11. KEEPING KERNEL PANIC PARAMETERS DISABLED IN VIRTUALIZED ENVIRONMENTS

When configuring a virtualized environment in RHEL 8, you should not enable the **softlockup_panic** and **nmi_watchdog** kernel parameters, as the virtualized environment may trigger a spurious soft lockup that should not require a system panic.

The following sections explain the reasons behind this advice by summarizing:

- What causes a soft lockup.
- Describing the kernel parameters that control a system's behavior on a soft lockup.
- Explaining how soft lockups may be triggered in a virtualized environment.

11.1. WHAT IS A SOFT LOCKUP

A soft lockup is a situation usually caused by a bug, when a task is executing in kernel space on a CPU without rescheduling. The task also does not allow any other task to execute on that particular CPU. As a result, a warning is displayed to a user through the system console. This problem is also referred to as the soft lockup firing.

Additional resources

- *What is a CPU soft lockup?*

11.2. PARAMETERS CONTROLLING KERNEL PANIC

The following kernel parameters can be set to control a system's behavior when a soft lockup is detected.

softlockup_panic

Controls whether or not the kernel will panic when a soft lockup is detected.

Type	Value	Effect
Integer	0	kernel does not panic on soft lockup
Integer	1	kernel panics on soft lockup

By default, on RHEL8 this value is 0.

In order to panic, the system needs to detect a hard lockup first. The detection is controlled by the **nmi_watchdog** parameter.

nmi_watchdog

Controls whether lockup detection mechanisms (**watchdogs**) are active or not. This parameter is of integer type.

Value	Effect
0	disables lockup detector
1	enables lockup detector

The hard lockup detector monitors each CPU for its ability to respond to interrupts.

watchdog_thresh

Controls frequency of watchdog **hrtimer**, NMI events, and soft/hard lockup thresholds.

Default threshold	Soft lockup threshold
10 seconds	2 * **watchdog_thresh**

Setting this parameter to zero disables lockup detection altogether.

Additional resources

- *Softlockup detector and hardlockup detector*
- *Kernel sysctl*

11.3. SPURIOUS SOFT LOCKUPS IN VIRTUALIZED ENVIRONMENTS

The soft lockup firing on physical hosts, as described in What is a soft lockup , usually represents a kernel or hardware bug. The same phenomenon happening on guest operating systems in virtualized environments may represent a false warning.

Heavy work-load on a host or high contention over some specific resource such as memory, usually causes a spurious soft lockup firing. This is because the host may schedule out the guest CPU for a period longer than 20 seconds. Then when the guest CPU is again scheduled to run on the host, it experiences a *time jump* which triggers due timers. The timers include also watchdog **hrtimer**, which can consequently report a soft lockup on the guest CPU.

Because a soft lockup in a virtualized environment may be spurious, you should not enable the kernel parameters that would cause a system panic when a soft lockup is reported on a guest CPU.

IMPORTANT

To understand soft lockups in guests, it is essential to know that the host schedules the guest as a task, and the guest then schedules its own tasks.

Additional resources

- What is a soft lockup
- *Virtual machine components and their interaction*

- *Virtual machine reports a "BUG: soft lockup"*

CHAPTER 12. ADJUSTING KERNEL PARAMETERS FOR DATABASE SERVERS

There are different sets of kernel parameters which can affect performance of specific database applications. The following sections explain what kernel parameters to configure to secure efficient operation of database servers and databases.

12.1. INTRODUCTION TO DATABASE SERVERS

A database server is a service that provides features of a database management system (DBMS). DBMS provides utilities for database administration and interacts with end users, applications, and databases.

Red Hat Enterprise Linux 8 provides the following database management systems:

- **MariaDB 10.3**
- **MariaDB 10.5** - available since RHEL 8.4
- **MySQL 8.0**
- **PostgreSQL 10**
- **PostgreSQL 9.6**
- **PostgreSQL 12** - available since RHEL 8.1.1
- **PostgreSQL 13** - available since RHEL 8.4

12.2. PARAMETERS AFFECTING PERFORMANCE OF DATABASE APPLICATIONS

The following kernel parameters affect performance of database applications.

fs.aio-max-nr

Defines the maximum number of asynchronous I/O operations the system can handle on the server.

NOTE

Raising the **fs.aio-max-nr** parameter produces no additional changes beyond increasing the aio limit.

fs.file-max

Defines the maximum number of file handles (temporary file names or IDs assigned to open files) the system supports at any instance.
The kernel dynamically allocates file handles whenever a file handle is requested by an application. The kernel however does not free these file handles when they are released by the application. The kernel recycles these file handles instead. This means that over time the total number of allocated file handles will increase even though the number of currently used file handles may be low.

kernel.shmall

Defines the total number of shared memory pages that can be used system-wide. To use the entire main memory, the value of the **kernel.shmall** parameter should be ≤ total main memory size.

kernel.shmmax

Defines the maximum size in bytes of a single shared memory segment that a Linux process can allocate in its virtual address space.

kernel.shmmni

Defines the maximum number of shared memory segments the database server is able to handle.

net.ipv4.ip_local_port_range

Defines the port range the system can use for programs which want to connect to a database server without a specific port number.

net.core.rmem_default

Defines the default receive socket memory through Transmission Control Protocol (TCP).

net.core.rmem_max

Defines the maximum receive socket memory through Transmission Control Protocol (TCP).

net.core.wmem_default

Defines the default send socket memory through Transmission Control Protocol (TCP).

net.core.wmem_max

Defines the maximum send socket memory through Transmission Control Protocol (TCP).

vm.dirty_bytes / vm.dirty_ratio

Defines a threshold in bytes / in percentage of dirty-able memory at which a process generating dirty data is started in the **write()** function.

NOTE

Either **vm.dirty_bytes** or **vm.dirty_ratio** can be specified at a time.

vm.dirty_background_bytes / vm.dirty_background_ratio

Defines a threshold in bytes / in percentage of dirty-able memory at which the kernel tries to actively write dirty data to hard-disk.

NOTE

Either **vm.dirty_background_bytes** or **vm.dirty_background_ratio** can be specified at a time.

vm.dirty_writeback_centisecs

Defines a time interval between periodic wake-ups of the kernel threads responsible for writing dirty data to hard-disk.
This kernel parameters measures in 100th's of a second.

vm.dirty_expire_centisecs

Defines the time after which dirty data is old enough to be written to hard-disk.
This kernel parameters measures in 100th's of a second.

Additional resources

- *Dirty pagecache writeback and vm.dirty parameters*

CHAPTER 13. GETTING STARTED WITH KERNEL LOGGING

Log files are files that contain messages about the system, including the kernel, services, and applications running on it. The logging system in Red Hat Enterprise Linux is based on the built-in **syslog** protocol. Various utilities use this system to record events and organize them into log files. These files are useful when auditing the operating system or troubleshooting problems.

13.1. WHAT IS THE KERNEL RING BUFFER

During the boot process, the console provides a lot of important information about the initial phase of the system startup. To avoid loss of the early messages the kernel utilizes what is called a ring buffer. This buffer stores all messages, including boot messages, generated by the **printk()** function within the kernel code. The messages from the kernel ring buffer are then read and stored in log files on permanent storage, for example, by the **syslog** service.

The buffer mentioned above is a cyclic data structure which has a fixed size, and is hard-coded into the kernel. Users can display data stored in the kernel ring buffer through the **dmesg** command or the **/var/log/boot.log** file. When the ring buffer is full, the new data overwrites the old.

Additional resources

- **syslog(2)** and **dmesg(1)** manual page

13.2. ROLE OF PRINTK ON LOG-LEVELS AND KERNEL LOGGING

Each message the kernel reports has a log-level associated with it that defines the importance of the message. The kernel ring buffer, as described in What is the kernel ring buffer , collects kernel messages of all log-levels. It is the **kernel.printk** parameter that defines what messages from the buffer are printed to the console.

The log-level values break down in this order:

- 0 – Kernel emergency. The system is unusable.
- 1 – Kernel alert. Action must be taken immediately.
- 2 – Condition of the kernel is considered critical.
- 3 – General kernel error condition.
- 4 – General kernel warning condition.
- 5 – Kernel notice of a normal but significant condition.
- 6 – Kernel informational message.
- 7 – Kernel debug-level messages.

By default, **kernel.printk** in RHEL 8 contains the following four values:

```
# sysctl kernel.printk
kernel.printk = 7 4 1 7
```

The four values define the following:

1. value. Console log-level, defines the lowest priority of messages printed to the console.
2. value. Default log-level for messages without an explicit log-level attached to them.
3. value. Sets the lowest possible log-level configuration for the console log-level.
4. value. Sets default value for the console log-level at boot time.
   Each of these values above defines a different rule for handling error messages.

IMPORTANT

The default 7 4 1 7**printk** value allows for better debugging of kernel activity. However, when coupled with a serial console, this **printk** setting is able to cause intense I/O bursts that could lead to a RHEL system becoming temporarily unresponsive. To avoid these situations, setting a **printk** value of 4 4 1 7typically works, but at the expense of losing the extra debugging information.

Also note that certain kernel command line parameters, such as **quiet** or **debug**, change the default **kernel.printk** values.

Additional resources

- **syslog(2)** manual page

CHAPTER 14. INSTALLING KDUMP

The **kdump** service is installed and activated by default on the new Red Hat Enterprise Linux installations. The following sections explain what **kdump** is and how to install **kdump** when it is not enabled by default.

14.1. WHAT IS KDUMP

kdump is a service which provides a crash dumping mechanism. The service enables you to save the contents of the system memory for analysis. **kdump** uses the **kexec** system call to boot into the second kernel (a *capture kernel*) without rebooting; and then captures the contents of the crashed kernel's memory (a *crash dump* or a *vmcore*) and saves it into a file. The second kernel resides in a reserved part of the system memory.

IMPORTANT

A kernel crash dump can be the only information available in the event of a system failure (a critical bug). Therefore, operational **kdump** is important in mission-critical environments. Red Hat advise that system administrators regularly update and test **kexec-tools** in your normal kernel update cycle. This is especially important when new kernel features are implemented.

You can enable **kdump** for all installed kernels on a machine or only for specified kernels. This is useful when there are multiple kernels used on a machine, some of which are stable enough that there is no concern that they could crash.

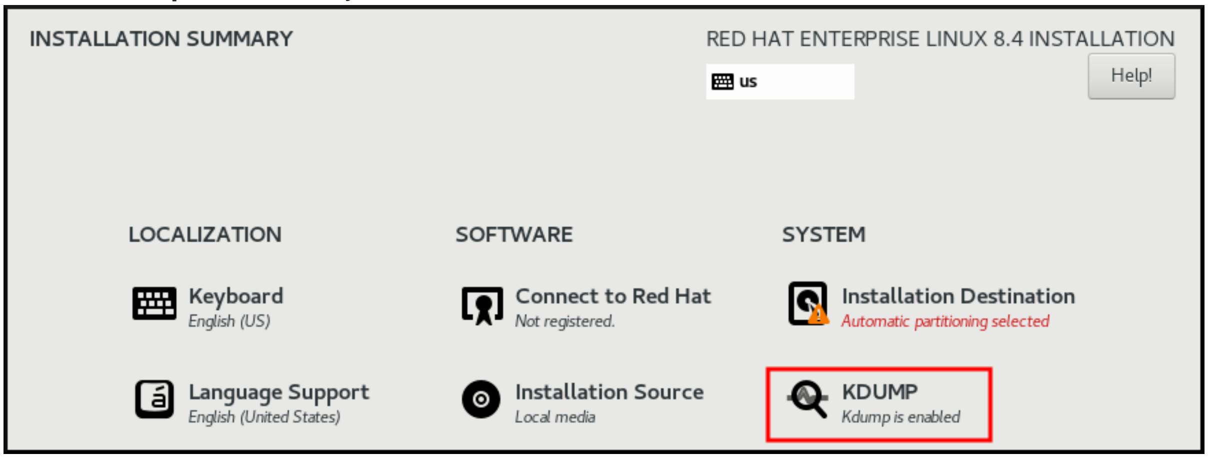
When **kdump** is installed, a default **/etc/kdump.conf** file is created. The file includes the default minimum **kdump** configuration. You can edit this file to customize the **kdump** configuration, but it is not required.

14.2. INSTALLING KDUMP USING ANACONDA

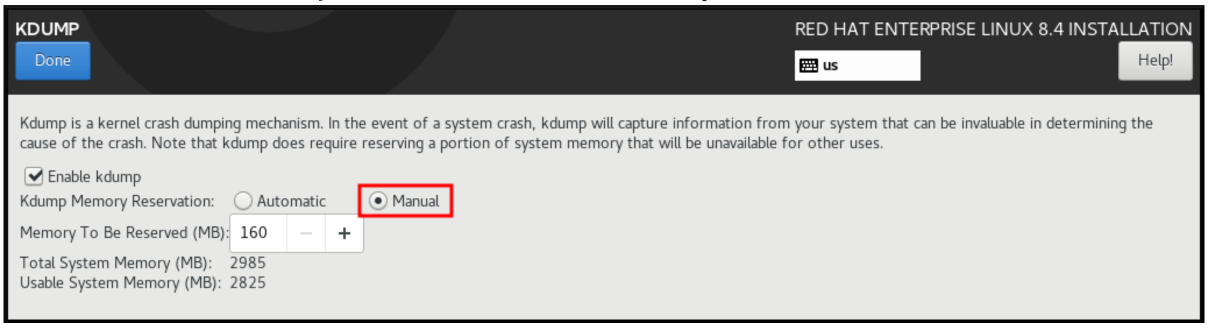
The **Anaconda** installer provides a graphical interface screen for **kdump** configuration during an interactive installation. The installer screen is titled as KDUMP and is available from the main **Installation Summary** screen. You can enable **kdump** and reserve the required amount of memory.

Procedure

1. Go to the **Kdump** field.
2. Enable **kdump** if not already enabled.



3. Define how much memory should be reserved for **kdump**.



Kdump is a kernel crash dumping mechanism. In the event of a system crash, kdump will capture information from your system that can be invaluable in determining the cause of the crash. Note that kdump does require reserving a portion of system memory that will be unavailable for other uses.

[✓] Enable kdump

Kdump Memory Reservation: Automatic [✓ Manual]

Memory To Be Reserved (MB): 160

Total System Memory (MB): 2985

Usable System Memory (MB): 2825

14.3. INSTALLING KDUMP ON THE COMMAND LINE

Some installation options, such as custom **Kickstart** installations, in some cases do **not** install or enable **kdump** by default. If this is your case, follow the procedure below.

Prerequisites

- An active RHEL subscription
- The **kexec-tools** package
- Fulfilled requirements for **kdump** configurations and targets. For details, see Supported kdump configurations and targets.

Procedure

1. Check whether **kdump** is installed on your system:

```
# rpm -q kexec-tools
```

   Output if the package is installed:

```
kexec-tools-2.0.17-11.el8.x86_64
```

   Output if the package is not installed:

```
package kexec-tools is not installed
```

2. Install **kdump** and other necessary packages by:

```
# dnf install kexec-tools
```

IMPORTANT

Starting with kernel-3.10.0-693.el7 the **Intel IOMMU** driver is supported with **kdump**. For prior versions, kernel-3.10.0-514[.XYZ].el7 and earlier, it is advised that **Intel IOMMU** support is disabled, otherwise the capture kernel is likely to become unresponsive.

CHAPTER 15. CONFIGURING KDUMP ON THE COMMAND LINE

The following sections explain how to plan and build your **kdump** environment.

15.1. ESTIMATING THE KDUMP SIZE

When planning and building your **kdump** environment, it is important to know how much space the crash dump file requires.

The **makedumpfile --mem-usage** command estimates how much space the crash dump file requires. It generates a memory usage report. The report helps you determine the dump level and which pages are safe to be excluded.

Procedure

- Execute the following command to generate a memory usage report:

```
# makedumpfile --mem-usage /proc/kcore


TYPE        PAGES     EXCLUDABLE    DESCRIPTION
----------------------------------------------------------------
ZERO          501635      yes        Pages filled with zero
CACHE          51657       yes        Cache pages
CACHE_PRIVATE 5442        yes        Cache pages + private
USER           16301       yes        User process pages
FREE          77738211    yes        Free pages
KERN_DATA      1333192     no         Dumpable kernel data
```

IMPORTANT

The **makedumpfile --mem-usage** command reports required memory in pages. This means that you must calculate the size of memory in use against the kernel page size.

By default the RHEL kernel uses 4 KB sized pages on AMD64 and Intel 64 CPU architectures, and 64 KB sized pages on IBM POWER architectures.

15.2. CONFIGURING KDUMP MEMORY USAGE

The memory for **kdump** is reserved during the system boot. The memory size is set in the system Grand Unified Bootloader (GRUB) configuration. The memory size depends on the value of the **crashkernel=** option specified in the configuration file and the size of the system physical memory.

The **crashkernel=** option can be defined in multiple ways. You can either specify the **crashkernel=** value or configure the **auto** option. The **crashkernel=auto** parameter reserves memory automatically, based on the total amount of physical memory in the system. When configured, the kernel will automatically reserve an appropriate amount of required memory for the capture kernel. This helps to prevent Out-of-Memory (OOM) errors.

NOTE

The automatic memory allocation for **kdump** varies based on system hardware architecture and available memory size.

For example, on AMD64 and Intel 64, the **crashkernel=auto** parameter works only when the available memory is more than 1GB. 64-bit ARM architecture and IBM Power Systems need to have more than 2GB of available memory.

If the system has less than the minimum memory threshold for automatic allocation, you can configure the amount of reserved memory manually.

Prerequisites

- Root permissions.
- Fulfilled requirements for **kdump** configurations and targets. For details, see Supported kdump configurations and targets.

Procedure

1. Prepare the **crashkernel=** option.
   - For example, to reserve 128 MB of memory, use the following:

     ```
     crashkernel=128M
     ```

   - Alternatively, you can set the amount of reserved memory to a variable depending on the total amount of installed memory. The syntax for memory reservation into a variable is **crashkernel=*<range1>*:*<size1>*,*<range2>*:*<size2>***. For example:

     ```
     crashkernel=512M-2G:64M,2G-:128M
     ```

     The above example reserves 64 MB of memory if the total amount of system memory is between 512 MB and 2 GB. If the total amount of memory is more than 2 GB, 128 MB is reserved.

   - Offset the reserved memory.
     Some systems require to reserve memory with a certain fixed offset since **crashkernel** reservation is very early, and it wants to reserve some area for special usage. If the offset is set, the reserved memory begins there. To offset the reserved memory, use the following syntax:

     ```
     crashkernel=128M@16M
     ```

     In this example, **kdump** reserves 128 MB of memory starting at 16 MB (physical address **0x01000000**). If the offset parameter is set to 0 or omitted entirely, **kdump** offsets the reserved memory automatically. You can also use this syntax when setting a variable memory reservation. In that case, the offset is always specified last. For example:

     ```
     crashkernel=512M-2G:64M,2G-:128M@16M
     ```

2. Apply the **crashkernel=** option to your boot loader configuration:

```
# grubby --update-kernel=ALL --args="crashkernel=<value>"
```

Replace ***<value>*** with the value of the the **crashkernel=** option that you prepared in the previous step.

Additional resources

- Memory requirements for kdump
- Configuring kernel command-line parameters
- How to manually modify the boot parameter in grub before the system boots
- How to install and boot custom kernels in Red Hat Enterprise Linux 8
- **grubby(8)** manual page

15.3. CONFIGURING THE KDUMP TARGET

The crash dump is usually stored as a file in a local file system, written directly to a device. Alternatively, you can set up for the crash dump to be sent over a network using the **NFS** or **SSH** protocols. Only one of these options to preserve a crash dump file can be set at a time. The default behavior is to store it in the **/var/crash/** directory of the local file system.

Prerequisites

- **Root** permissions.
- Fulfilled requirements for **kdump** configurations and targets. For details, see Supported kdump configurations and targets.

Procedure

- To store the crash dump file in **/var/crash/** directory of the local file system, edit the **/etc/kdump.conf** file and specify the path:

  ```
  path /var/crash
  ```

  The option **path /var/crash** represents the path to the file system in which **kdump** saves the crash dump file. When you specify a dump target in the **/etc/kdump.conf** file, then the **path** is relative to the specified dump target.

  If you do not specify a dump target in the **/etc/kdump.conf** file, then the **path** represents the absolute path from the root directory. Depending on what is mounted in the current system, the dump target and the adjusted dump path are taken automatically.

WARNING

kdump saves the crash dump file in **/var/crash/var/crash** directory, when the dump target is mounted at **/var/crash** and the option **path** is also set as **/var/crash** in the **/etc/kdump.conf** file. For example, in the following instance, the **ext4** file system is already mounted at **/var/crash** and the **path** are set as **/var/crash**:

```
# grep -v ^# /etc/kdump.conf | grep -v ^$
ext4 /dev/mapper/vg00-varcrashvol
path /var/crash
core_collector makedumpfile -c --message-level 1 -d 31
```

This results in the **/var/crash/var/crash** path. To solve this problem, use the option **path /** instead of **path /var/crash**

- To change the local directory in which the crash dump is to be saved, as **root**, edit the **/etc/kdump.conf** configuration file as described below.
  1. Remove the hash sign ("#") from the beginning of the **#path /var/crash** line.
  2. Replace the value with the intended directory path. For example:

     ```
     path /usr/local/cores
     ```

     IMPORTANT

     In RHEL 8, the directory defined as the kdump target using the **path** directive must exist when the **kdump** systemd service is started - otherwise the service fails. This behavior is different from earlier releases of RHEL, where the directory was being created automatically if it did not exist when starting the service.

- To write the file to a different partition, as **root**, edit the **/etc/kdump.conf** configuration file as described below.
  1. Remove the hash sign ("#") from the beginning of the **#ext4** line, depending on your choice.
     - device name (the **#ext4 /dev/vg/lv_kdump** line)
     - file system label (the **#ext4 LABEL=/boot** line)
     - UUID (the **#ext4 UUID=03138356-5e61-4ab3-b58e-27507ac41937** line)
  2. Change the file system type as well as the device name, label or UUID to the desired values. For example:

     ```
     ext4 UUID=03138356-5e61-4ab3-b58e-27507ac41937
     ```

IMPORTANT

It is recommended to specify storage devices using a **LABEL=** or **UUID=**. Disk device names such as **/dev/sda3** are not guaranteed to be consistent across reboot.

IMPORTANT

When dumping to Direct Access Storage Device (DASD) on IBM Z hardware, it is essential that the dump devices are correctly specified in **/etc/dasd.conf** before proceeding.

- To write the crash dump directly to a device, edit the **/etc/kdump.conf** configuration file:
  1. Remove the hash sign ("#") from the beginning of the **#raw /dev/vg/lv_kdump** line.
  2. Replace the value with the intended device name. For example:

     ```
     raw /dev/sdb1
     ```

- To store the crash dump to a remote machine using the **NFS** protocol, edit the **/etc/kdump.conf** configuration file:
  1. Remove the hash sign ("#") from the beginning of the **#nfs my.server.com:/export/tmp** line.
  2. Replace the value with a valid hostname and directory path. For example:

     ```
     nfs penguin.example.com:/export/cores
     ```

- To store the crash dump to a remote machine using the **SSH** protocol, edit the **/etc/kdump.conf** configuration file:
  1. Remove the hash sign ("#") from the beginning of the **#ssh user@my.server.com** line.
  2. Replace the value with a valid username and hostname.
  3. Include your **SSH** key in the configuration.
     - Remove the hash sign from the beginning of the **#sshkey /root/.ssh/kdump_id_rsa** line.
     - Change the value to the location of a key valid on the server you are trying to dump to. For example:

       ```
       ssh john@penguin.example.com
       sshkey /root/.ssh/mykey
       ```

15.4. CONFIGURING THE KDUMP CORE COLLECTOR

The **kdump** service uses a **core_collector** program to capture the crash dump image. In RHEL, the **makedumpfile** utility is the default core collector. It helps shrink the dump file by:

- Compressing the size of a crash dump file and copying only necessary pages using various dump levels

- Excluding unnecessary crash dump pages
- Filtering the page types to be included in the crash dump.

Syntax

```
core_collector makedumpfile -l --message-level 1 -d 31
```

Options

- **-c**, **-l** or **-p**: specify compress dump file format by each page using either, **zlib** for **-c** option, **lzo** for **-l** option or **snappy** for **-p** option.
- **-d (dump_level)**: excludes pages so that they are not copied to the dump file.
- **--message-level** : specify the message types. You can restrict outputs printed by specifying **message_level** with this option. For example, specifying 7 as **message_level** prints common messages and error messages. The maximum value of **message_level** is 31

Prerequisites

- **Root** permissions
- Fulfilled requirements for **kdump** configurations and targets. For details, see Supported kdump configurations and targets.

Procedure

1. As **root**, edit the **/etc/kdump.conf** configuration file and remove the hash sign ("#") from the beginning of the **#core_collector makedumpfile -l --message-level 1 -d 31**.
2. To enable crash dump file compression, execute:

```
core_collector makedumpfile -l --message-level 1 -d 31
```

The **-l** option specifies the **dump** compressed file format. The **-d** option specifies dump level as 31. The **--message-level** option specifies message level as 1.

Also, consider following examples with the **-c** and **-p** options:

- To compress a crash dump file using **-c**:

```
core_collector makedumpfile -c -d 31 --message-level 1
```

- To compress a crash dump file using **-p**:

```
core_collector makedumpfile -p -d 31 --message-level 1
```

Additional resources

- the **makedumpfile(8)** man page
- The kdump configuration file

15.5. CONFIGURING THE KDUMP DEFAULT FAILURE RESPONSES

By default, when **kdump** fails to create a crash dump file at the configured target location, the system reboots and the dump is lost in the process. To change this behavior, follow the procedure below.

Prerequisites

- Root permissions.
- Fulfilled requirements for **kdump** configurations and targets. For details, see Supported kdump configurations and targets.

Procedure

1. As **root**, remove the hash sign ("#") from the beginning of the **#failure_action** line in the **/etc/kdump.conf** configuration file.
2. Replace the value with a desired action.

   ```
   failure_action poweroff
   ```

Additional resources

- Configuring the kdump target

15.6. CONFIGURATION FILE FOR KDUMP

The configuration file for **kdump** kernel is **/etc/sysconfig/kdump**. This file controls the **kdump** kernel command line parameters.

For most configurations, use the default options. However, in some scenarios you might need to modify certain parameters to control the **kdump** kernel behavior. For example, modifying to append the **kdump** kernel command-line to obtain a detailed debugging output.

This section covers information on modifying the **KDUMP_COMMANDLINE_REMOVE** and **KDUMP_COMMANDLINE_APPEND** options for **kdump**. For information on additional configuration options refer to **Documentation/admin-guide/kernel-parameters.txt** or the **/etc/sysconfig/kdump** file.

- **KDUMP_COMMANDLINE_REMOVE**
  This option removes arguments from the current **kdump** command line. It removes parameters that may cause **kdump** errors or **kdump** kernel boot failures. These parameters may have been parsed from the previous **KDUMP_COMMANDLINE** process or inherited from the **/proc/cmdline** file. When this variable is not configured, it inherits all values from the **/proc/cmdline** file. Configuring this option also provides information that is helpful in debugging an issue.

  Example

  To remove certain arguments, add them to **KDUMP_COMMANDLINE_REMOVE** as follows:

  ```
  KDUMP_COMMANDLINE_REMOVE="hugepages hugepagesz slub_debug quiet
  log_buf_len swiotlb"
  ```

- **KDUMP_COMMANDLINE_APPEND**
  This option appends arguments to the current command line. These arguments may have been parsed by the previous **KDUMP_COMMANDLINE_REMOVE** variable.

  For the **kdump** kernel, disabling certain modules such as **mce**, **cgroup**, **numa**, **hest_disable** can help prevent kernel errors. These modules may consume a significant portion of the kernel memory reserved for **kdump** or cause **kdump** kernel boot failures.

  Example

  To disable memory **cgroups** on the **kdump** kernel command line, run the command as follows:

  ```
  KDUMP_COMMANDLINE_APPEND="cgroup_disable=memory"
  ```

Additional resources

- **Documentation/admin-guide/kernel-parameters.txt** file
- **/etc/sysconfig/kdump** file

15.7. TESTING THE KDUMP CONFIGURATION

You can test that the crash dump process works and is valid before the machine enters production.

WARNING

The commands below cause the kernel to crash. Use caution when following these steps, and never carelessly use them on active production system.

Procedure

1. Reboot the system with **kdump** enabled.
2. Make sure that **kdump** is running:

   ```
   # systemctl is-active kdump
   active
   ```

3. Force the Linux kernel to crash:

   ```
   echo 1 > /proc/sys/kernel/sysrq
   echo c > /proc/sysrq-trigger
   ```

WARNING

The command above crashes the kernel, and a reboot is required.

Once booted again, the ***address-YYYY-MM-DD-HH:MM:SS*/vmcore** file is created at the location you have specified in the **/etc/kdump.conf** file (by default to **/var/crash/**).

NOTE

This action confirms the validity of the configuration. Also it is possible to use this action to record how long it takes for a crash dump to complete with a representative work-load.

Additional resources

- Configuring the kdump target

CHAPTER 16. ENABLING KDUMP

This section provides the information and procedures necessary to enable and start the **kdump** service for all installed kernels or for a specific kernel.

16.1. ENABLING KDUMP FOR ALL INSTALLED KERNELS

You can enable and start the **kdump** service for all kernels installed on the machine.

Prerequisites

- Administrator privileges

Procedure

1. Add the **crashkernel=auto** command-line parameter to all installed kernels:

```
# grubby --update-kernel=ALL --args="crashkernel=auto"
```

2. Enable the **kdump** service.

```
# systemctl enable --now kdump.service
```

Verification

- Check that the **kdump** service is running:

```
# systemctl status kdump.service

○ kdump.service - Crash recovery kernel arming
    Loaded: loaded (/usr/lib/systemd/system/kdump.service; enabled; vendor preset:
disabled)
    Active: active (live)
```

16.2. ENABLING KDUMP FOR A SPECIFIC INSTALLED KERNEL

You can enable the **kdump** service for a specific kernel on the machine.

Prerequisites

- Administrator privileges

Procedure

1. List the kernels installed on the machine.

```
# ls -a /boot/vmlinuz-*
/boot/vmlinuz-0-rescue-2930657cd0dc43c2b75db480e5e5b4a9 /boot/vmlinuz-4.18.0-
330.el8.x86_64 /boot/vmlinuz-4.18.0-330.rt7.111.el8.x86_64
```

2. Add a specific **kdump** kernel to the system's Grand Unified Bootloader (GRUB) configuration file.

For example:

```
# grubby --update-kernel=vmlinuz-4.18.0-330.el8.x86_64 --args="crashkernel=auto"
```

3. Enable the **kdump** service.

```
# systemctl enable --now kdump.service
```

Verification

- Check that the **kdump** service is running:

```
# systemctl status kdump.service

○ kdump.service - Crash recovery kernel arming
    Loaded: loaded (/usr/lib/systemd/system/kdump.service; enabled; vendor preset:
disabled)
    Active: active (live)
```

16.3. DISABLING THE KDUMP SERVICE

To disable the **kdump** service at boot time, follow the procedure below.

Prerequisites

- Fulfilled requirements for **kdump** configurations and targets. For details, see Supported kdump configurations and targets.
- All configurations for installing **kdump** are set up according to your needs. For details, see Installing kdump .

Procedure

1. To stop the **kdump** service in the current session:

```
# systemctl stop kdump.service
```

2. To disable the **kdump** service:

```
# systemctl disable kdump.service
```

WARNING

It is recommended to set **kptr_restrict=1**. In that case, the **kdumpctl** service loads the crash kernel regardless of Kernel Address Space Layout (KASLR) being enabled or not.

Troubleshooting step

When **kptr_restrict** is not set to (1), and if KASLR is enabled, the contents of **/proc/kcore** file are generated as all zeros. Consequently, the **kdumpctl** service fails to access the **/proc/kcore** and load the crash kernel.

To work around this problem, the **/usr/share/doc/kexec-tools/kexec-kdump-howto.txt** file displays a warning message, which recommends the **kptr_restrict=1** setting.

To ensure that **kdumpctl** service loads the crash kernel, verify that **kernel.kptr_restrict = 1** is listed in the **sysctl.conf** file.

Additional resources

- Configuring basic system settings in RHEL

CHAPTER 17. CONFIGURING KDUMP IN THE WEB CONSOLE

Setup and test the **kdump** configuration in the RHEL 8 web console.

The web console is part of a default installation of RHEL 8 and enables or disables the **kdump** service at boot time. Further, the web console enables you to configure the reserved memory for **kdump**; or to select the *vmcore* saving location in an uncompressed or compressed format.

17.1. CONFIGURING KDUMP MEMORY USAGE AND TARGET LOCATION IN WEB CONSOLE

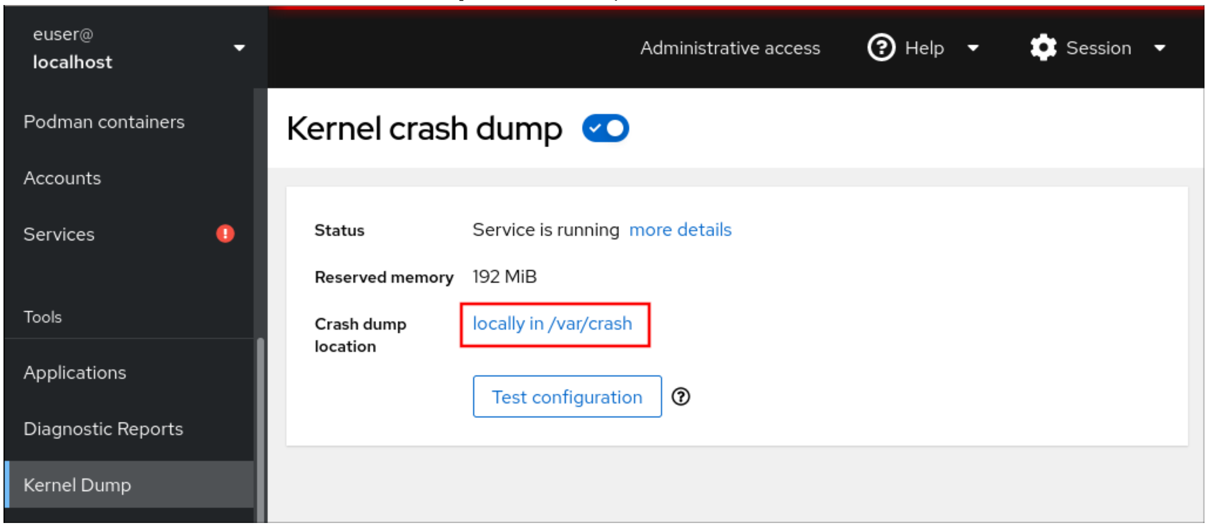
The procedure below shows you how to use the **Kernel Dump** tab in the RHEL web console interface to configure the amount of memory that is reserved for the **kdump** kernel. The procedure also describes how to specify the target location of the **vmcore** dump file and how to test your configuration.

Procedure

1. Open the **Kernel Dump** tab and start the **kdump** service.

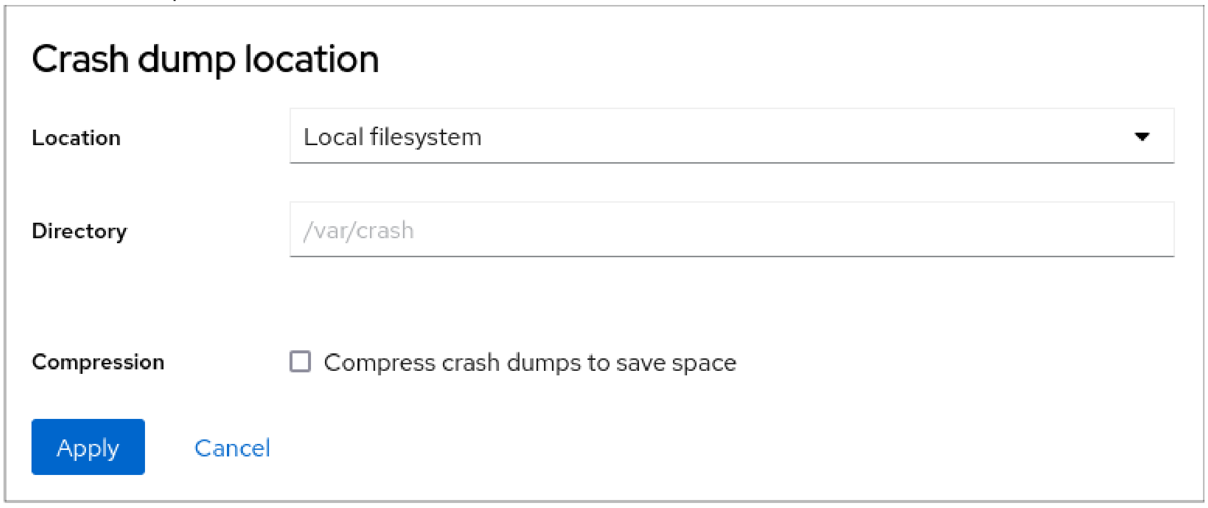
2. Configure the **kdump** memory usage using the command line.

3. Click the link next to the **Crash dump location** option.



Status: Service is running more details
Reserved memory: 192 MiB
Crash dump location: locally in /var/crash

4. Select the **Local Filesystem** option from the drop-down and specify the directory you want to save the dump in.



Crash dump location

Location: Local filesystem
Directory: /var/crash
Compression: ☐ Compress crash dumps to save space

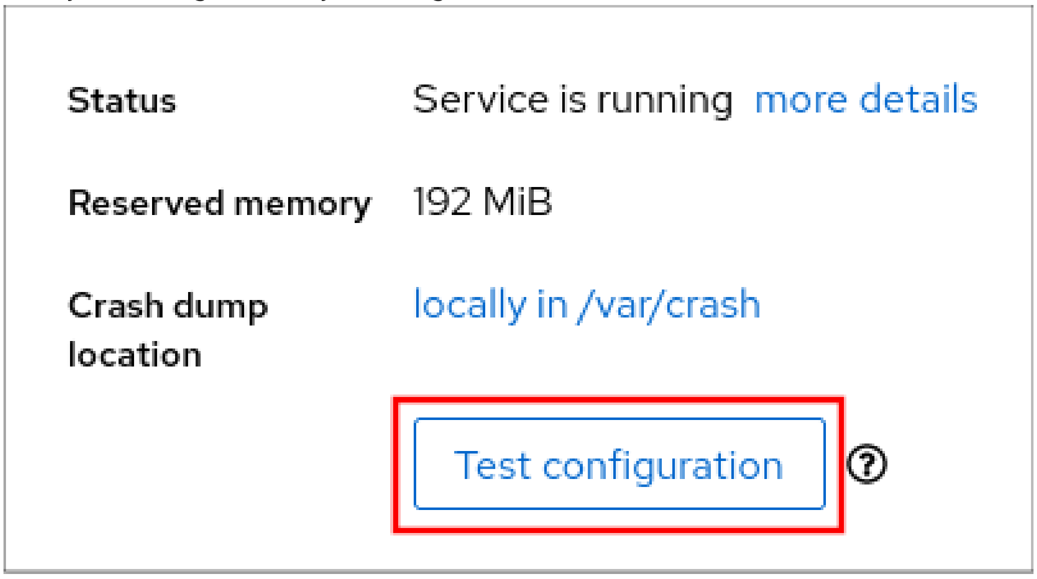
- Alternatively, select the **Remote over SSH** option from the drop-down to send the vmcore to a remote machine using the SSH protocol.
  Fill the **Server**, **ssh key**, and **Directory** fields with the remote machine address, ssh key location, and a target directory.

- Another choice is to select the **Remote over NFS** option from the drop-down and fill the **Mount** field to send the vmcore to a remote machine using the NFS protocol.

NOTE

Tick the **Compression** check box to reduce the size of the vmcore file.

5. Test your configuration by crashing the kernel.



Status — Service is running more details

Reserved memory — 192 MiB

Crash dump location — locally in /var/crash

a. Click **Test configuration**.

b. In the **Test kdump settings** field, click **Crash system**.

WARNING

This step disrupts execution of the kernel and results in a system crash and loss of data.

Additional resources

- Supported kdump targets
- Using secure communications between two systems with OpenSSH

17.2. ADDITIONAL RESOURCES

- Getting started using the RHEL web console

CHAPTER 18. SUPPORTED KDUMP CONFIGURATIONS AND TARGETS

18.1. MEMORY REQUIREMENTS FOR KDUMP

In order for **kdump** to be able to capture a kernel crash dump and save it for further analysis, a part of the system memory has to be permanently reserved for the capture kernel. When reserved, this part of the system memory is not available to the main kernel.

The memory requirements vary based on certain system parameters. One of the major factors is the system's hardware architecture. To find out the exact machine architecture (such as Intel 64 and AMD64, also known as x86_64) and print it to standard output, use the following command:

```
$ uname -m
```

The following table lists the minimum memory requirements to automatically reserve a memory size for **kdump** on the latest available versions. The size changes according to the system's architecture and total available physical memory.

Table 18.1. Minimum Amount of Reserved Memory Required for kdump

Architecture	Available Memory	Minimum Reserved Memory
AMD64 and Intel 64 (**x86_64**)	1 GB to 4 GB	160 MB of RAM.
	4 GB to 64 GB	192 MB of RAM.
	64 GB to 1 TB	256 MB of RAM.
	1 TB and more	512 MB of RAM.
64-bit ARM architecture (**arm64**)	2 GB and more	448 MB of RAM.
IBM Power Systems (**ppc64le**)	2 GB to 4 GB	384 MB of RAM.
	4 GB to 16 GB	512 MB of RAM.
	16 GB to 64 GB	1 GB of RAM.
	64 GB to 128 GB	2 GB of RAM.
	128 GB and more	4 GB of RAM.
IBM Z (**s390x**)	1 GB to 4 GB	160 MB of RAM.
	4 GB to 64 GB	192 MB of RAM.
	64 GB to 1 TB	256 MB of RAM.

Architecture	Available Memory	Minimum Reserved Memory
	1 TB and more	512 MB of RAM.

On many systems, **kdump** is able to estimate the amount of required memory and reserve it automatically. This behavior is enabled by default, but only works on systems that have more than a certain amount of total available memory, which varies based on the system architecture.

IMPORTANT

The automatic configuration of reserved memory based on the total amount of memory in the system is a best effort estimation. The actual required memory may vary due to other factors such as I/O devices. Using not enough of memory might cause that a debug kernel is not able to boot as a capture kernel in case of a kernel panic. To avoid this problem, sufficiently increase the crash kernel memory.

Additional resources

- How has the crashkernel parameter changed between RHEL8 minor releases?
- Technology capabilities and limits tables
- Minimum threshold for automatic memory reservation

18.2. MINIMUM THRESHOLD FOR AUTOMATIC MEMORY RESERVATION

On some systems, it is possible to allocate memory for **kdump** automatically, either by using the **crashkernel=auto** parameter in the boot loader configuration file, or by enabling this option in the graphical configuration utility. For this automatic reservation to work, however, a certain amount of total memory needs to be available in the system. The amount differs based on the system's architecture.

The table below lists the threshold values for automatic memory allocation. If the system has memory less than the specified threshold value, you must configure the memory manually.

Table 18.2. Minimum Amount of Memory Required for Automatic Memory Reservation

Architecture	Required Memory
AMD64 and Intel 64 (**x86_64**)	2 GB
IBM Power Systems (**ppc64le**)	2 GB
IBM Z (**s390x**)	4 GB

18.3. SUPPORTED KDUMP TARGETS

When a kernel crash is captured, the vmcore dump file can be either written directly to a device, stored as a file on a local file system, or sent over a network. The table below contains a complete list of dump targets that are currently supported or explicitly unsupported by **kdump**.

Type	Supported Targets	Unsupported Targets
Raw device	All locally attached raw disks and partitions.	
Local file system	**ext2**, **ext3**, **ext4**, and **xfs** file systems on directly attached disk drives, hardware RAID logical drives, LVM devices, and **mdraid** arrays.	Any local file system not explicitly listed as supported in this table, including the **auto** type (automatic file system detection).
Remote directory	Remote directories accessed using the **NFS** or **SSH** protocol over **IPv4**.	Remote directories on the **rootfs** file system accessed using the **NFS** protocol.
Remote directories accessed using the **iSCSI** protocol over both hardware and software initiators.	Remote directories accessed using the **iSCSI** protocol on **be2iscsi** hardware.	Multipath-based storages.
		Remote directories accessed over **IPv6**.
		Remote directories accessed using the **SMB** or **CIFS** protocol.
		Remote directories accessed using the **FCoE** (*Fibre Channel over Ethernet*) protocol.
		Remote directories accessed using wireless network interfaces.

IMPORTANT

Utilizing firmware assisted dump (**fadump**) to capture a vmcore and store it to a remote machine using SSH or NFS protocol causes renaming of the network interface to **kdump-<interface-name>**. The renaming happens if the **<interface-name>** is generic, for example ***eth#**, **net#**, and so on. This problem occurs because the vmcore capture scripts in the initial RAM disk (**initrd**) add the *kdump-* prefix to the network interface name to secure persistent naming. Since the same **initrd** is used also for a regular boot, the interface name is changed for the production kernel too.

Additional resources

- Configuring kdump target

18.4. SUPPORTED KDUMP FILTERING LEVELS

To reduce the size of the dump file, **kdump** uses the **makedumpfile** core collector to compress the data and optionally to omit unwanted information. The table below contains a complete list of filtering levels that are currently supported by the **makedumpfile** utility.

Option	Description
1	Zero pages
2	Cache pages
4	Cache private
8	User pages
16	Free pages

NOTE

The **makedumpfile** command supports removal of transparent huge pages and hugetlbfs pages. Consider both these types of hugepages User Pages and remove them using the **-8** level.

Additional resources

- Configuring the core collector

18.5. SUPPORTED DEFAULT FAILURE RESPONSES

By default, when **kdump** fails to create a core dump, the operating system reboots. You can, however, configure **kdump** to perform a different operation in case it fails to save the core dump to the primary target. The table below lists all default actions that are currently supported.

Option	Description
dump_to_rootfs	Attempt to save the core dump to the root file system. This option is especially useful in combination with a network target: if the network target is unreachable, this option configures kdump to save the core dump locally. The system is rebooted afterwards.
reboot	Reboot the system, losing the core dump in the process.
halt	Halt the system, losing the core dump in the process.
poweroff	Power off the system, losing the core dump in the process.

Option	Description
shell	Run a shell session from within the initramfs, allowing the user to record the core dump manually.
final_action	Enable additional operations such as **reboot**, **halt**, and **poweroff** actions after a successful**kdump** or when **shell** or **dump_to_rootfs** failure action completes. The default **final_action** option is **reboot**.

Additional resources

- Configuring the kdump default failure responses

18.6. USING FINAL_ACTION PARAMETER

The **final_action** parameter enables you to use certain additional operations such as **reboot**, **halt**, and **poweroff** actions after a successful **kdump** or when an invoked **failure_response** mechanism using **shell** or **dump_to_rootfs** completes. If the **final_action** option is not specified, it defaults to **reboot**.

Procedure

1. Edit the `/**etc/kdump.conf** file and add the **final_action** parameter.

```
final_action <reboot | halt | poweroff>
```

2. Restart the **kdump** service:

```
kdumpctl restart
```

CHAPTER 19. FIRMWARE ASSISTED DUMP MECHANISMS

Firmware assisted dump (fadump) is a dump capturing mechanism, provided as an alternative to the **kdump** mechanism on IBM POWER systems. The **kexec** and **kdump** mechanisms are useful for capturing core dumps on AMD64 and Intel 64 systems. However, some hardware such as mini systems and mainframe computers, leverage the onboard firmware to isolate regions of memory and prevent any accidental overwriting of data that is important to the crash analysis. This section covers **fadump** mechanisms and how they integrate with RHEL. The **fadump** utility is optimized for these expanded dumping features on IBM POWER systems.

19.1. FIRMWARE ASSISTED DUMP ON IBM POWERPC HARDWARE

The **fadump** utility captures the **vmcore** file from a fully-reset system with PCI and I/O devices. This mechanism uses firmware to preserve memory regions during a crash and then reuses the **kdump** userspace scripts to save the **vmcore** file. The memory regions consist of all system memory contents, except the boot memory, system registers, and hardware Page Table Entries (PTEs).

The **fadump** mechanism offers improved reliability over the traditional dump type, by rebooting the partition and using a new kernel to dump the data from the previous kernel crash. The **fadump** requires an IBM POWER6 processor-based or later version hardware platform.

For further details about the **fadump** mechanism, including PowerPC specific methods of resetting hardware, see the **/usr/share/doc/kexec-tools/fadump-howto.txt** file.

NOTE

The area of memory that is not preserved, known as boot memory, is the amount of RAM required to successfully boot the kernel after a crash event. By default, the boot memory size is 256MB or 5% of total system RAM, whichever is larger.

Unlike **kexec-initiated** event, the **fadump** mechanism uses the production kernel to recover a crash dump. When booting after a crash, PowerPC hardware makes the device node **/proc/device-tree/rtas/ibm.kernel-dump** available to the **proc** filesystem (**procfs**). The **fadump-aware kdump** scripts, check for the stored **vmcore**, and then complete the system reboot cleanly.

19.2. ENABLING FIRMWARE ASSISTED DUMP MECHANISM

The crash dumping capabilities of IBM POWER systems can be enhanced by enabling the firmware assisted dump (fadump) mechanism.

Procedure

1. Install and configure **kdump**.

2. Enable the **fadump=on** kernel option:

```
# grubby --update-kernel=ALL --args="fadump=on"
```

3. (Optional) If you want to specify reserved boot memory instead of using the defaults, enable the **crashkernel=*xx*M** option, where ***xx*** is the amount of the memory required in megabytes:

```
# grubby --update-kernel=ALL --args="crashkernel=xxM fadump=on"
```

IMPORTANT

Red Hat recommends to test all boot configuration options before you execute them. If you observe Out of Memory (OOM) errors when booting from the crash kernel, increase the value specified in **crashkernel=** argument until the crash kernel can boot cleanly. Some trial and error may be required in this case.

19.3. FIRMWARE ASSISTED DUMP MECHANISMS ON IBM Z HARDWARE

IBM Z systems support the following firmware assisted dump mechanisms:

- **Stand-alone dump (sadump)**
- **VMDUMP**

The **kdump** infrastructure is supported and utilized on IBM Z systems. However, using one of the firmware assisted dump (fadump) methods for IBM Z can provide various benefits:

- The **sadump** mechanism is initiated and controlled from the system console, and is stored on an **IPL** bootable device.
- The **VMDUMP** mechanism is similar to **sadump**. This tool is also initiated from the system console, but retrieves the resulting dump from hardware and copies it to the system for analysis.
- These methods (similarly to other hardware based dump mechanisms) have the ability to capture the state of a machine in the early boot phase, before the **kdump** service starts.
- Although **VMDUMP** contains a mechanism to receive the dump file into a Red Hat Enterprise Linux system, the configuration and control of **VMDUMP** is managed from the IBM Z Hardware console.

IBM discusses **sadump** in detail in the **Stand-alone dump program** article and **VMDUMP** in **Creating dumps on z/VM with VMDUMP** article.

IBM also has a documentation set for using the dump tools on Red Hat Enterprise Linux 7 in the **Using the Dump Tools on Red Hat Enterprise Linux 7.4** article.

Additional resources

- Stand-alone dump program
- Creating dumps on z/VM with VMDUMP
- Using the Dump Tools on Red Hat Enterprise Linux 7.4

19.4. USING SADUMP ON FUJITSU PRIMEQUEST SYSTEMS

The Fujitsu **sadump** mechanism is designed to provide a **fallback** dump capture in an event when **kdump** is unable to complete successfully. The **sadump** mechanism is invoked manually from the system Management Board (MMB) interface. Using MMB, configure **kdump** like for an Intel 64 or AMD 64 server and then perform the following additional steps to enable **sadump**.

Procedure

1. Add or edit the following lines in the **/etc/sysctl.conf** file to ensure that **kdump** starts as expected for **sadump**:

   ```
   kernel.panic=0
   kernel.unknown_nmi_panic=1
   ```

   WARNING

   In particular, ensure that after **kdump**, the system does not reboot. If the system reboots after **kdump** has fails to save the **vmcore** file, then it is not possible to invoke the **sadump**.

2. Set the **failure_action** parameter in **/etc/kdump.conf** appropriately as **halt** or **shell**.

   ```
   failure_action shell
   ```

Additional resources

- The FUJITSU Server PRIMEQUEST 2000 Series Installation Manual

CHAPTER 20. ANALYZING A CORE DUMP

To determine the cause of the system crash, you can use the **crash** utility, which provides an interactive prompt very similar to the GNU Debugger (GDB). This utility allows you to interactively analyze a core dump created by **kdump**, **netdump**, **diskdump** or **xendump** as well as a running Linux system. Alternatively, you have the option to use Kernel Oops Analyzer or the Kdump Helper tool.

20.1. INSTALLING THE CRASH UTILITY

The following procedure describes how to install the **crash** analyzing tool.

Procedure

1. Enable the relevant repositories:

   ```
   # subscription-manager repos --enable baseos repository
   ```

   ```
   # subscription-manager repos --enable appstream repository
   ```

   ```
   # subscription-manager repos --enable rhel-8-for-x86_64-baseos-debug-rpms
   ```

2. Install the **crash** package:

   ```
   # yum install crash
   ```

3. Install the **kernel-debuginfo** package:

   ```
   # yum install kernel-debuginfo
   ```

   The package corresponds to your running kernel and provides the data necessary for the dump analysis.

Additional resources

- Configuring basic system settings

20.2. RUNNING AND EXITING THE CRASH UTILITY

The following procedure describes how to start the crash utility for analyzing the cause of the system crash.

Prerequisites

- Identify the currently running kernel (for example **4.18.0-5.el8.x86_64**).

Procedure

1. To start the **crash** utility, two necessary parameters need to be passed to the command:

   - The debug-info (a decompressed vmlinuz image), for example **/usr/lib/debug/lib/modules/4.18.0-5.el8.x86_64/vmlinux** provided through a specific **kernel-debuginfo** package.

- The actual vmcore file, for example **/var/crash/127.0.0.1-2018-10-06-14:05:33/vmcore**
  The resulting **crash** command then looks like this:

  ```
  # crash /usr/lib/debug/lib/modules/4.18.0-5.el8.x86_64/vmlinux /var/crash/127.0.0.1-2018-10-06-14:05:33/vmcore
  ```

  Use the same *<kernel>* version that was captured by **kdump**.

  Example 20.1. Running the crash utility

  The following example shows analyzing a core dump created on October 6 2018 at 14:05 PM, using the 4.18.0-5.el8.x86_64 kernel.

  ```
  ...
  WARNING: kernel relocated [202MB]: patching 90160 gdb minimal_symbol values

        KERNEL: /usr/lib/debug/lib/modules/4.18.0-5.el8.x86_64/vmlinux
      DUMPFILE: /var/crash/127.0.0.1-2018-10-06-14:05:33/vmcore  [PARTIAL DUMP]
          CPUS: 2
          DATE: Sat Oct  6 14:05:16 2018
        UPTIME: 01:03:57
  LOAD AVERAGE: 0.00, 0.00, 0.00
         TASKS: 586
      NODENAME: localhost.localdomain
       RELEASE: 4.18.0-5.el8.x86_64
       VERSION: #1 SMP Wed Aug 29 11:51:55 UTC 2018
       MACHINE: x86_64  (2904 Mhz)
        MEMORY: 2.9 GB
         PANIC: "sysrq: SysRq : Trigger a crash"
           PID: 10635
       COMMAND: "bash"
          TASK: ffff8d6c84271800  [THREAD_INFO: ffff8d6c84271800]
           CPU: 1
         STATE: TASK_RUNNING (SYSRQ)

  crash>
  ```

2. To exit the interactive prompt and terminate **crash**, type **exit** or **q**.

   Example 20.2. Exiting the crash utility

   ```
   crash> exit
   ~]#
   ```

NOTE

The **crash** command can also be used as a powerful tool for debugging a live system. However use it with caution so as not to break your system.

Additional resources

- A Guide to Unexpected System Restarts

20.3. DISPLAYING VARIOUS INDICATORS IN THE CRASH UTILITY

The following procedures describe how to use the crash utility and display various indicators, such as a kernel message buffer, a backtrace, a process status, virtual memory information and open files.

Displaying the message buffer

- To display the kernel message buffer, type the **log** command at the interactive prompt as displayed in the example below:

```
crash> log
... several lines omitted ...
EIP: 0060:[<c068124f>] EFLAGS: 00010096 CPU: 2
EIP is at sysrq_handle_crash+0xf/0x20
EAX: 00000063 EBX: 00000063 ECX: c09e1c8c EDX: 00000000
ESI: c0a09ca0 EDI: 00000286 EBP: 00000000 ESP: ef4dbf24
 DS: 007b ES: 007b FS: 00d8 GS: 00e0 SS: 0068
Process bash (pid: 5591, ti=ef4da000 task=f196d560 task.ti=ef4da000)
Stack:
 c068146b c0960891 c0968653 00000003 00000000 00000002 efade5c0 c06814d0
<0> fffffffb c068150f b7776000 f2600c40 c0569ec4 ef4dbf9c 00000002 b7776000
<0> efade5c0 00000002 b7776000 c0569e60 c051de50 ef4dbf9c f196d560 ef4dbfb4
Call Trace:
 [<c068146b>] ? __handle_sysrq+0xfb/0x160
 [<c06814d0>] ? write_sysrq_trigger+0x0/0x50
 [<c068150f>] ? write_sysrq_trigger+0x3f/0x50
 [<c0569ec4>] ? proc_reg_write+0x64/0xa0
 [<c0569e60>] ? proc_reg_write+0x0/0xa0
 [<c051de50>] ? vfs_write+0xa0/0x190
 [<c051e8d1>] ? sys_write+0x41/0x70
 [<c0409adc>] ? syscall_call+0x7/0xb
Code: a0 c0 01 0f b6 41 03 19 d2 f7 d2 83 e2 03 83 e0 cf c1 e2 04 09 d0 88 41 03 f3 c3 90 c7 05
c8 1b 9e c0 01 00 00 00 0f ae f8 89 f6 <c6> 05 00 00 00 00 01 c3 89 f6 8d bc 27 00 00 00 00 8d 50
d0 83
EIP: [<c068124f>] sysrq_handle_crash+0xf/0x20 SS:ESP 0068:ef4dbf24
CR2: 0000000000000000
```

Type **help log** for more information on the command usage.

NOTE

The kernel message buffer includes the most essential information about the system crash and, as such, it is always dumped first in to the **vmcore-dmesg.txt** file. This is useful when an attempt to get the full **vmcore** file failed, for example because of lack of space on the target location. By default, **vmcore-dmesg.txt** is located in the **/var/crash/** directory.

Displaying a backtrace

- To display the kernel stack trace, use the **bt** command.

```
crash> bt
PID: 5591   TASK: f196d560  CPU: 2   COMMAND: "bash"
 #0 [ef4dbdcc] crash_kexec at c0494922
 #1 [ef4dbe20] oops_end at c080e402
 #2 [ef4dbe34] no_context at c043089d
 #3 [ef4dbe58] bad_area at c0430b26
 #4 [ef4dbe6c] do_page_fault at c080fb9b
 #5 [ef4dbee4] error_code (via page_fault) at c080d809
    EAX: 00000063  EBX: 00000063  ECX: c09e1c8c  EDX: 00000000  EBP: 00000000
    DS:  007b      ESI: c0a09ca0  ES:  007b      EDI: 00000286  GS:  00e0
    CS:  0060      EIP: c068124f  ERR: ffffffff  EFLAGS: 00010096
 #6 [ef4dbf18] sysrq_handle_crash at c068124f
 #7 [ef4dbf24] __handle_sysrq at c0681469
 #8 [ef4dbf48] write_sysrq_trigger at c068150a
 #9 [ef4dbf54] proc_reg_write at c0569ec2
#10 [ef4dbf74] vfs_write at c051de4e
#11 [ef4dbf94] sys_write at c051e8cc
#12 [ef4dbfb0] system_call at c0409ad5
    EAX: ffffffda  EBX: 00000001  ECX: b7776000  EDX: 00000002
    DS:  007b      ESI: 00000002  ES:  007b      EDI: b7776000
    SS:  007b      ESP: bfcb2088  EBP: bfcb20b4  GS:  0033
    CS:  0073      EIP: 00edc416  ERR: 00000004  EFLAGS: 00000246
```

Type **bt *<pid>*** to display the backtrace of a specific process or type **help bt** for more information on **bt** usage.

Displaying a process status

- To display the status of processes in the system, use the **ps** command.

```
crash> ps
   PID    PPID  CPU   TASK    ST  %MEM     VSZ    RSS  COMM
>     0      0   0  c09dc560  RU   0.0       0      0  [swapper]
>     0      0   1  f7072030  RU   0.0       0      0  [swapper]
      0      0   2  f70a3a90  RU   0.0       0      0  [swapper]
>     0      0   3  f70ac560  RU   0.0       0      0  [swapper]
      1      0   1  f705ba90  IN   0.0    2828   1424  init
... several lines omitted ...
   5566      1   1  f2592560  IN   0.0   12876    784  auditd
   5567      1   2  ef427560  IN   0.0   12876    784  auditd
   5587   5132   0  f196d030  IN   0.0   11064   3184  sshd
>  5591   5587   2  f196d560  RU   0.0    5084   1648  bash
```

Use **ps *<pid>*** to display the status of a single specific process. Use *help ps* for more information on **ps** usage.

Displaying virtual memory information

- To display basic virtual memory information, type the **vm** command at the interactive prompt.

```
crash> vm
PID: 5591   TASK: f196d560  CPU: 2   COMMAND: "bash"
   MM       PGD      RSS    TOTAL_VM
f19b5900  ef9c6000  1648k    5084k
  VMA       START      END    FLAGS  FILE
f1bb0310    242000    260000 8000875  /lib/ld-2.12.so
f26af0b8    260000    261000 8100871  /lib/ld-2.12.so
efbc275c    261000    262000 8100873  /lib/ld-2.12.so
```

```
efbc2a18    268000    3ed000 8000075  /lib/libc-2.12.so
efbc23d8    3ed000    3ee000 8000070  /lib/libc-2.12.so
efbc2888    3ee000    3f0000 8100071  /lib/libc-2.12.so
efbc2cd4    3f0000    3f1000 8100073  /lib/libc-2.12.so
efbc243c    3f1000    3f4000 100073
efbc28ec    3f6000    3f9000 8000075  /lib/libdl-2.12.so
efbc2568    3f9000    3fa000 8100071  /lib/libdl-2.12.so
efbc2f2c    3fa000    3fb000 8100073  /lib/libdl-2.12.so
f26af888    7e6000    7fc000 8000075  /lib/libtinfo.so.5.7
f26aff2c    7fc000    7ff000 8100073  /lib/libtinfo.so.5.7
efbc211c    d83000    d8f000 8000075  /lib/libnss_files-2.12.so
efbc2504    d8f000    d90000 8100071  /lib/libnss_files-2.12.so
efbc2950    d90000    d91000 8100073  /lib/libnss_files-2.12.so
f26afe00    edc000    edd000 4040075
f1bb0a18   8047000   8118000 8001875  /bin/bash
f1bb01e4   8118000   811d000 8101873  /bin/bash
f1bb0c70   811d000   8122000 100073
f26afae0   9fd9000   9ffa000 100073
... several lines omitted ...
```

Use **vm *<pid>*** to display information on a single specific process, or use **help vm** for more information on **vm** usage.

Displaying open files

- To display information about open files, use the **files** command.

```
crash> files
PID: 5591   TASK: f196d560  CPU: 2   COMMAND: "bash"
ROOT: /    CWD: /root
 FD    FILE     DENTRY    INODE    TYPE  PATH
  0  f734f640  eedc2c6c  eecd6048  CHR   /pts/0
  1  efade5c0  eee14090  f00431d4  REG   /proc/sysrq-trigger
  2  f734f640  eedc2c6c  eecd6048  CHR   /pts/0
 10  f734f640  eedc2c6c  eecd6048  CHR   /pts/0
255  f734f640  eedc2c6c  eecd6048  CHR   /pts/0
```

Use **files *<pid>*** to display files opened by only one selected process, or use **help files** for more information on **files** usage.

20.4. USING KERNEL OOPS ANALYZER

The Kernel Oops Analyzer tool analyzes the crash dump by comparing the oops messages with known issues in the knowledge base.

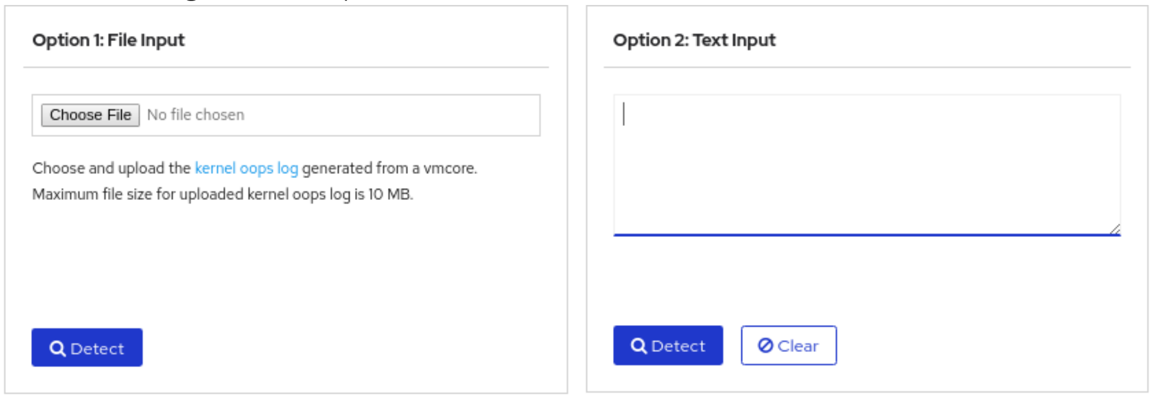
Prerequisites

- Secure an oops message to feed the Kernel Oops Analyzer.

Procedure

1. Access the Kernel Oops Analyzer tool.
2. To diagnose a kernel crash issue, upload a kernel oops log generated in **vmcore**.

- Alternatively you can also diagnose a kernel crash issue by providing a text message or a **vmcore-dmesg.txt** as an input.



Choose and upload the kernel oops log generated from a vmcore.
Maximum file size for uploaded kernel oops log is 10 MB.

3. Click **DETECT** to compare the oops message based on information from the **makedumpfile** against known solutions.

Additional resources

- The Kernel Oops Analyzer article
- A Guide to Unexpected System Restarts

20.5. THE KDUMP HELPER TOOL

The Kdump Helper tool helps to set up the **kdump** using the provided information. Kdump Helper generates a configuration script based on your preferences. Initiating and running the script on your server sets up the **kdump** service.

Additional resources

- Kdump Helper

CHAPTER 21. USING EARLY KDUMP TO CAPTURE BOOT TIME CRASHES

As a system administrator, you can utilize the **early kdump** support of the **kdump** service to capture a vmcore file of the crashing kernel during the early stages of the booting process. This section describes what **early kdump** is, how to configure it, and how to check the status of this mechanism.

21.1. WHAT IS EARLY KDUMP

Kernel crashes during the booting phase occur when the **kdump** service is not yet started, and cannot facilitate capturing and saving the contents of the crashed kernel's memory. Therefore, the vital information for troubleshooting is lost.

To address this problem, RHEL 8 introduced the **early kdump** feature as a part of the **kdump** service.

21.2. ENABLING EARLY KDUMP

This section describes how to enable the **early kdump** feature to eliminate the risk of losing information about the early boot kernel crashes.

Prerequisites

- An active RHEL subscription.
- A repository containing the **kexec-tools** package for your system CPU architecture
- Fulfilled **kdump** configuration and targets requirements.

Procedure

1. Verify that the **kdump** service is enabled and active:

   ```
   # systemctl is-enabled kdump.service && systemctl is-active kdump.service enabled
   active
   ```

   If **kdump** is not enabled and running, set all required configurations and verify that **kdump** service is enabled.

2. Rebuild the **initramfs** image of the booting kernel with the **early kdump** functionality:

   ```
   # dracut -f --add earlykdump
   ```

3. Add the **rd.earlykdump** kernel command line parameter:

   ```
   # grubby --update-kernel=/boot/vmlinuz-$(uname -r) --args="rd.earlykdump"
   ```

4. Reboot the system to reflect the changes

   ```
   # reboot
   ```

Verification step

- Verify that **rd.earlykdump** was successfully added and **early kdump** feature was enabled:

  ```
  # cat /proc/cmdline
  BOOT_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-187.el8.x86_64 root=/dev/mapper/rhel-root ro
  crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap
  rhgb quiet rd.earlykdump

  # journalctl -x | grep early-kdump
  Mar 20 15:44:41 redhat dracut-cmdline[304]: early-kdump is enabled.
  Mar 20 15:44:42 redhat dracut-cmdline[304]: kexec: loaded early-kdump kernel
  ```

Additional resources

- **/usr/share/doc/kexec-tools/early-kdump-howto.txt** file
- What is early kdump support and how do I configure it?
- Enabling kdump

CHAPTER 22. APPLYING PATCHES WITH KERNEL LIVE PATCHING

You can use the Red Hat Enterprise Linux kernel live patching solution to patch a running kernel without rebooting or restarting any processes.

With this solution, system administrators:

- Can immediately apply critical security patches to the kernel.
- Do not have to wait for long-running tasks to complete, for users to log off, or for scheduled downtime.
- Control the system's uptime more and do not sacrifice security or stability.

Note that not every critical or important CVE will be resolved using the kernel live patching solution. Our goal is to reduce the required reboots for security-related patches, not to eliminate them entirely. For more details about the scope of live patching, see the *Customer Portal Solutions article* .

WARNING

Some incompatibilities exist between kernel live patching and other kernel subcomponents. Read the Limitations of kpatch carefully before using kernel live patching.

22.1. LIMITATIONS OF KPATCH

- The **kpatch** feature is not a general-purpose kernel upgrade mechanism. It is used for applying simple security and bug fix updates when rebooting the system is not immediately possible.
- Do not use the **SystemTap** or **kprobe** tools during or after loading a patch. The patch could fail to take effect until after such probes have been removed.

22.2. SUPPORT FOR THIRD-PARTY LIVE PATCHING

The **kpatch** utility is the only kernel live patching utility supported by Red Hat with the RPM modules provided by Red Hat repositories. Red Hat will not support any live patches which were not provided by Red Hat itself.

If you require support for an issue that arises with a third-party live patch, Red Hat recommends that you open a case with the live patching vendor at the outset of any investigation in which a root cause determination is necessary. This allows the source code to be supplied if the vendor allows, and for their support organization to provide assistance in root cause determination prior to escalating the investigation to Red Hat Support.

For any system running with third-party live patches, Red Hat reserves the right to ask for reproduction with Red Hat shipped and supported software. In the event that this is not possible, we require a similar system and workload be deployed on your test environment without live patches applied, to confirm if the same behavior is observed.

For more information about third-party software support policies, see How does Red Hat Global Support Services handle third-party software, drivers, and/or uncertified hardware/hypervisors or guest operating systems?

22.3. ACCESS TO KERNEL LIVE PATCHES

Kernel live patching capability is implemented as a kernel module (**kmod**) that is delivered as an RPM package.

All customers have access to kernel live patches, which are delivered through the usual channels. However, customers who do not subscribe to an extended support offering will lose access to new patches for the current minor release once the next minor release becomes available. For example, customers with standard subscriptions will only be able to live patch RHEL 8.2 kernel until the RHEL 8.3 kernel is released.

22.4. COMPONENTS OF KERNEL LIVE PATCHING

The components of kernel live patching are as follows:

Kernel patch module

- The delivery mechanism for kernel live patches.
- A kernel module which is built specifically for the kernel being patched.
- The patch module contains the code of the desired fixes for the kernel.
- The patch modules register with the **livepatch** kernel subsystem and provide information about original functions to be replaced, with corresponding pointers to the replacement functions. Kernel patch modules are delivered as RPMs.
- The naming convention is **kpatch_<kernel version>_<kpatch version>_<kpatch release>**. The "kernel version" part of the name has *dots* replaced with *underscores*.

The kpatch utility

A command-line utility for managing patch modules.

The kpatch service

A **systemd** service required by **multiuser.target**. This target loads the kernel patch module at boot time.

The kpatch-dnf package

A DNF plugin delivered in the form of an RPM package. This plugin manages automatic subscription to kernel live patches.

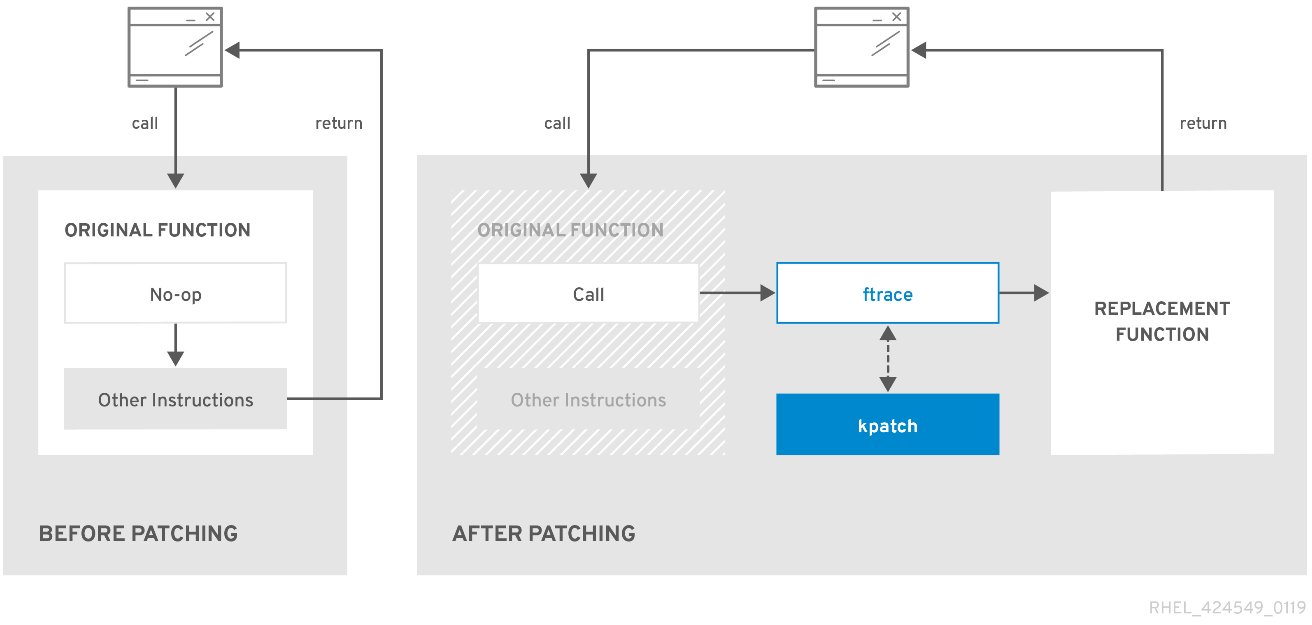
22.5. HOW KERNEL LIVE PATCHING WORKS

The **kpatch** kernel patching solution uses the **livepatch** kernel subsystem to redirect old functions to new ones. When a live kernel patch is applied to a system, the following things happen:

1. The kernel patch module is copied to the **/var/lib/kpatch/** directory and registered for re-application to the kernel by **systemd** on next boot.
2. The kpatch module is loaded into the running kernel and the new functions are registered to the **ftrace** mechanism with a pointer to the location in memory of the new code.

3. When the kernel accesses the patched function, it is redirected by the **ftrace** mechanism which bypasses the original functions and redirects the kernel to patched version of the function.

Figure 22.1. How kernel live patching works



22.6. SUBSCRIBING THE CURRENTLY INSTALLED KERNELS TO THE LIVE PATCHING STREAM

A kernel patch module is delivered in an RPM package, specific to the version of the kernel being patched. Each RPM package will be cumulatively updated over time.

The following procedure explains how to subscribe to all future cumulative live patching updates for a given kernel. Because live patches are cumulative, you cannot select which individual patches are deployed for a given kernel.

WARNING

Red Hat does not support any third party live patches applied to a Red Hat supported system.

Prerequisites

- Root permissions

Procedure

1. Optionally, check your kernel version:

```
# uname -r
4.18.0-94.el8.x86_64
```

2. Search for a live patching package that corresponds to the version of your kernel:

```
# yum search $(uname -r)
```

3. Install the live patching package:

```
# yum install "kpatch-patch = $(uname -r)"
```

The command above installs and applies the latest cumulative live patches for that specific kernel only.

If the version of a live patching package is 1-1 or higher, the package will contain a patch module. In that case the kernel will be automatically patched during the installation of the live patching package.

The kernel patch module is also installed into the **/var/lib/kpatch/** directory to be loaded by the **systemd** system and service manager during the future reboots.

NOTE

An empty live patching package will be installed when there are no live patches available for a given kernel. An empty live patching package will have a *kpatch_version-kpatch_release* of 0-0, for example **kpatch-patch-4_18_0-94-0-0.el8.x86_64.rpm**. The installation of the empty RPM subscribes the system to all future live patches for the given kernel.

4. Optionally, verify that the kernel is patched:

```
# kpatch list
Loaded patch modules:
kpatch_4_18_0_94_1_1 [enabled]

Installed patch modules:
kpatch_4_18_0_94_1_1 (4.18.0-94.el8.x86_64)
…
```

The output shows that the kernel patch module has been loaded into the kernel, which is now patched with the latest fixes from the **kpatch-patch-4_18_0-94-1-1.el8.x86_64.rpm** package.

Additional resources

- **kpatch(1)** manual page
- *Configuring basic system settings* in RHEL

22.7. AUTOMATICALLY SUBSCRIBING ANY FUTURE KERNEL TO THE LIVE PATCHING STREAM

You can use the **kpatch-dnf** YUM plugin to subscribe your system to fixes delivered by the kernel patch module, also known as kernel live patches. The plugin enables **automatic** subscription for any kernel the system currently uses, and also for kernels **to-be-installed in the future**

Prerequisites

- You have root permissions.

Procedure

1. Optionally, check all installed kernels and the kernel you are currently running:

```
# yum list installed | grep kernel
Updating Subscription Management repositories.
Installed Packages
...
kernel-core.x86_64       4.18.0-240.10.1.el8_3        @rhel-8-for-x86_64-baseos-rpms
kernel-core.x86_64       4.18.0-240.15.1.el8_3        @rhel-8-for-x86_64-baseos-rpms
...

# uname -r
4.18.0-240.10.1.el8_3.x86_64
```

2. Install the **kpatch-dnf** plugin:

```
# yum install kpatch-dnf
```

3. Enable automatic subscription to kernel live patches:

```
# yum kpatch auto
Updating Subscription Management repositories.
Last metadata expiration check: 19:10:26 ago on Wed 10 Mar 2021 04:08:06 PM CET.
Dependencies resolved.
==================================================
 Package                      Architecture
==================================================
Installing:
 kpatch-patch-4_18_0-240_10_1       x86_64
 kpatch-patch-4_18_0-240_15_1       x86_64

Transaction Summary
===================================================
Install  2 Packages
…
```

This command subscribes all currently installed kernels to receiving kernel live patches. The command also installs and applies the latest cumulative live patches, if any, for all installed kernels.

In the future, when you update the kernel, live patches will automatically be installed during the new kernel installation process.

The kernel patch module is also installed into the **/var/lib/kpatch/** directory to be loaded by the **systemd** system and service manager during future reboots.

NOTE

An empty live patching package will be installed when there are no live patches available for a given kernel. An empty live patching package will have a *kpatch_version-kpatch_release* of 0-0, for example **kpatch-patch-4_18_0-240-0-0.el8.x86_64.rpm** .The installation of the empty RPM subscribes the system to all future live patches for the given kernel.

Verification step

- Verify that all installed kernels have been patched:

```
# kpatch list
Loaded patch modules:
kpatch_4_18_0_240_10_1_0_1 [enabled]

Installed patch modules:
kpatch_4_18_0_240_10_1_0_1 (4.18.0-240.10.1.el8_3.x86_64)
kpatch_4_18_0_240_15_1_0_2 (4.18.0-240.15.1.el8_3.x86_64)
```

  The output shows that both the kernel you are running, and the other installed kernel have been patched with fixes from **kpatch-patch-4_18_0-240_10_1-0-1.rpm** and **kpatch-patch-4_18_0-240_15_1-0-1.rpm** packages respectively.

Additional resources

- **kpatch(1)** and **dnf-kpatch(8)** manual pages
- *Configuring basic system settings* in RHEL

22.8. DISABLING AUTOMATIC SUBSCRIPTION TO THE LIVE PATCHING STREAM

When you subscribe your system to fixes delivered by the kernel patch module, your subscription is **automatic**. You can disable this feature, and thus disable automatic installation of **kpatch-patch** packages.

Prerequisites

- You have root permissions.

Procedure

1. Optionally, check all installed kernels and the kernel you are currently running:

```
# yum list installed | grep kernel
Updating Subscription Management repositories.
Installed Packages
...
kernel-core.x86_64       4.18.0-240.10.1.el8_3        @rhel-8-for-x86_64-baseos-rpms
kernel-core.x86_64       4.18.0-240.15.1.el8_3        @rhel-8-for-x86_64-baseos-rpms
...
```

```
# uname -r
4.18.0-240.10.1.el8_3.x86_64
```

2. Disable automatic subscription to kernel live patches:

```
# yum kpatch manual
Updating Subscription Management repositories.
```

Verification step

- You can check for the successful outcome:

```
# yum kpatch status
...
Updating Subscription Management repositories.
Last metadata expiration check: 0:30:41 ago on Tue Jun 14 15:59:26 2022.
Kpatch update setting: manual
```

Additional resources

- **kpatch(1)** and **dnf-kpatch(8)** manual pages

22.9. UPDATING KERNEL PATCH MODULES

Since kernel patch modules are delivered and applied through RPM packages, updating a cumulative kernel patch module is like updating any other RPM package.

Prerequisites

- The system is subscribed to the live patching stream, as described in Subscribing the currently installed kernels to the live patching stream.

Procedure

- Update to a new cumulative version for the current kernel:

```
# yum update "kpatch-patch = $(uname -r)"
```

  The command above automatically installs and applies any updates that are available for the currently running kernel. Including any future released cumulative live patches.

- Alternatively, update all installed kernel patch modules:

```
# yum update "kpatch-patch"
```

NOTE

When the system reboots into the same kernel, the kernel is automatically live patched again by the **kpatch.service** systemd service.

Additional resources

- *Configuring basic system settings* in RHEL

22.10. REMOVING THE LIVE PATCHING PACKAGE

The following procedure describes how to disable the Red Hat Enterprise Linux kernel live patching solution by removing the live patching package.

Prerequisites

- Root permissions
- The live patching package is installed.

Procedure

1. Select the live patching package:

   ```
   # yum list installed | grep kpatch-patch
   kpatch-patch-4_18_0-94.x86_64        1-1.el8        @@commandline
   …
   ```

   The example output above lists live patching packages that you installed.

2. Remove the live patching package:

   ```
   # yum remove kpatch-patch-4_18_0-94.x86_64
   ```

   When a live patching package is removed, the kernel remains patched until the next reboot, but the kernel patch module is removed from disk. On future reboot, the corresponding kernel will no longer be patched.

3. Reboot your system.

4. Verify that the live patching package has been removed:

   ```
   # yum list installed | grep kpatch-patch
   ```

   The command displays no output if the package has been successfully removed.

5. Optionally, verify that the kernel live patching solution is disabled:

   ```
   # kpatch list
   Loaded patch modules:
   ```

   The example output shows that the kernel is not patched and the live patching solution is not active because there are no patch modules that are currently loaded.

IMPORTANT

Currently, Red Hat does not support reverting live patches without rebooting your system. In case of any issues, contact our support team.

Additional resources

- **kpatch(1)** manual page
- *Configuring basic system settings* in RHEL

22.11. UNINSTALLING THE KERNEL PATCH MODULE

The following procedure describes how to prevent the Red Hat Enterprise Linux kernel live patching solution from applying a kernel patch module on subsequent boots.

Prerequisites

- Root permissions
- A live patching package is installed.
- A kernel patch module is installed and loaded.

Procedure

1. Select a kernel patch module:

```
# kpatch list
Loaded patch modules:
kpatch_4_18_0_94_1_1 [enabled]

Installed patch modules:
kpatch_4_18_0_94_1_1 (4.18.0-94.el8.x86_64)
…
```

2. Uninstall the selected kernel patch module:

```
# kpatch uninstall kpatch_4_18_0_94_1_1
uninstalling kpatch_4_18_0_94_1_1 (4.18.0-94.el8.x86_64)
```

   - Note that the uninstalled kernel patch module is still loaded:

```
# kpatch list
Loaded patch modules:
kpatch_4_18_0_94_1_1 [enabled]

Installed patch modules:
<NO_RESULT>
```

   When the selected module is uninstalled, the kernel remains patched until the next reboot, but the kernel patch module is removed from disk.

3. Reboot your system.

4. Optionally, verify that the kernel patch module has been uninstalled:

```
# kpatch list
Loaded patch modules:
…
```

The example output above shows no loaded or installed kernel patch modules, therefore the kernel is not patched and the kernel live patching solution is not active.

IMPORTANT

Currently, Red Hat does not support reverting live patches without rebooting your system. In case of any issues, contact our support team.

Additional resources

- **kpatch(1)** manual page

22.12. DISABLING KPATCH.SERVICE

The following procedure describes how to prevent the Red Hat Enterprise Linux kernel live patching solution from applying all kernel patch modules globally on subsequent boots.

Prerequisites

- Root permissions
- A live patching package is installed.
- A kernel patch module is installed and loaded.

Procedure

1. Verify **kpatch.service** is enabled:

```
# systemctl is-enabled kpatch.service
enabled
```

2. Disable **kpatch.service**:

```
# systemctl disable kpatch.service
Removed /etc/systemd/system/multi-user.target.wants/kpatch.service.
```

   - Note that the applied kernel patch module is still loaded:

   ```
   # kpatch list
   Loaded patch modules:
   kpatch_4_18_0_94_1_1 [enabled]

   Installed patch modules:
   kpatch_4_18_0_94_1_1 (4.18.0-94.el8.x86_64)
   ```

3. Reboot your system.

4. Optionally, verify the status of **kpatch.service**:

```
# systemctl status kpatch.service
● kpatch.service - "Apply kpatch kernel patches"
  Loaded: loaded (/usr/lib/systemd/system/kpatch.service; disabled; vendor preset: disabled)
  Active: inactive (dead)
```

The example output testifies that **kpatch.service** has been disabled and is not running. Thereby, the kernel live patching solution is not active.

5. Verify that the kernel patch module has been unloaded:

```
# kpatch list
Loaded patch modules:
<NO_RESULT>

Installed patch modules:
kpatch_4_18_0_94_1_1 (4.18.0-94.el8.x86_64)
```

The example output above shows that a kernel patch module is still installed but the kernel is not patched.

IMPORTANT

Currently, Red Hat does not support reverting live patches without rebooting your system. In case of any issues, contact our support team.

Additional resources

- **kpatch(1)** manual page
- *Configuring basic system settings* in RHEL

CHAPTER 23. IMPROVING SYSTEM PERFORMANCE WITH ZSWAP

You can improve system performance by enabling the **zswap** kernel feature.

23.1. WHAT IS ZSWAP

This section explains what **zswap** is and how it can lead to system performance improvement.

zswap is a kernel feature that provides a compressed RAM cache for swap pages. The mechanism works as follows: **zswap** takes pages that are in the process of being swapped out and attempts to compress them into a dynamically allocated RAM-based memory pool. When the pool becomes full or the RAM becomes exhausted, **zswap** evicts pages from compressed cache on an LRU basis (least recently used) to the backing swap device. After the page has been decompressed into the swap cache, **zswap** frees the compressed version in the pool.

The benefits of zswap

- significant I/O reduction
- significant improvement of workload performance

In Red Hat Enterprise Linux 8, **zswap** is enabled by default.

Additional resources

- What is Zswap?

23.2. ENABLING ZSWAP AT RUNTIME

You can enable the **zswap** feature at system runtime using the **sysfs** interface.

Prerequisites

- You have root permissions.

Procedure

- Enable **zswap**:

```
# echo 1 > /sys/module/zswap/parameters/enabled
```

Verification step

- Verify that **zswap** is enabled:

```
# grep -r . /sys/kernel/debug/zswap

duplicate_entry:0
pool_limit_hit:13422200
pool_total_size:6184960 (pool size in total in pages)
reject_alloc_fail:5
```

```
reject_compress_poor:0
reject_kmemcache_fail:0
reject_reclaim_fail:13422200
stored_pages:4251 (pool size after compression)
written_back_pages:0
```

Additional resources

- How to enable Zswap feature?

23.3. ENABLING ZSWAP PERMANENTLY

You can enable the **zswap** feature permanently by providing the **zswap.enabled=1** kernel command-line parameter.

Prerequisites

- You have root permissions.
- The **grubby** or **zipl** utility is installed on your system.

Procedure

1. Enable **zswap** permanently:

   ```
   # grubby --update-kernel=/boot/vmlinuz-$(uname -r) --args="zswap.enabled=1"
   ```

2. Reboot the system for the changes to take effect.

Verification steps

- Verify that **zswap** is enabled:

  ```
  # cat /proc/cmdline

  BOOT_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-372.9.1.el8.x86_64
  root=/dev/mapper/rhel-root ro crashkernel=auto
  resume=/dev/mapper/rhel-swap
  rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet
  zswap.enabled=1
  ```

Additional resources

- How to enable Zswap feature?
- Configuring kernel command-line parameters

CHAPTER 24. SETTING LIMITS FOR APPLICATIONS

You can use the *control groups* (**cgroups**) kernel functionality to set limits, prioritize or isolate the hardware resources of processes. This allows you to granularly control resource usage of applications to utilize them more efficiently.

24.1. UNDERSTANDING CONTROL GROUPS

Control groups is a Linux kernel feature that enables you to organize processes into hierarchically ordered groups - **cgroups**. The hierarchy (control groups tree) is defined by providing structure to **cgroups** virtual file system, mounted by default on the **/sys/fs/cgroup/** directory. The **systemd** system and service manager utilizes **cgroups** to organize all units and services that it governs. Alternatively, you can manage **cgroups** hierarchies manually by creating and removing sub-directories in the **/sys/fs/cgroup/** directory.

The resource controllers (a kernel component) then modify the behavior of processes in **cgroups** by limiting, prioritizing or allocating system resources, (such as CPU time, memory, network bandwidth, or various combinations) of those processes.

The added value of **cgroups** is process aggregation which enables division of hardware resources among applications and users. Thereby an increase in overall efficiency, stability and security of users' environment can be achieved.

Control groups version 1

Control groups version 1 (**cgroups-v1**) provide a per-resource controller hierarchy. It means that each resource, such as CPU, memory, I/O, and so on, has its own control group hierarchy. It is possible to combine different control group hierarchies in a way that one controller can coordinate with another one in managing their respective resources. However, the two controllers may belong to different process hierarchies, which does not permit their proper coordination.
The **cgroups-v1** controllers were developed across a large time span and as a result, the behavior and naming of their control files is not uniform.

Control groups version 2

The problems with controller coordination, which stemmed from hierarchy flexibility, led to the development of *control groups version 2*.
Control groups version 2 (**cgroups-v2**) provides a single control group hierarchy against which all resource controllers are mounted.

The control file behavior and naming is consistent among different controllers.

NOTE

cgroups-v2 is fully supported in RHEL 8.2 and later versions. For more information, see Control Group v2 is now fully supported in RHEL 8 .

This sub-section was based on a Devconf.cz 2019 presentation.[1]

Additional resources

- What are kernel resource controllers
- **cgroups(7)** manual page

- Role of systemd in control groups

24.2. WHAT ARE KERNEL RESOURCE CONTROLLERS

The functionality of control groups is enabled by kernel resource controllers. RHEL 8 supports various controllers for *control groups version 1* (**cgroups-v1**) and *control groups version 2* (**cgroups-v2**).

A resource controller, also called a control group subsystem, is a kernel subsystem that represents a single resource, such as CPU time, memory, network bandwidth or disk I/O. The Linux kernel provides a range of resource controllers that are mounted automatically by the **systemd** system and service manager. Find a list of currently mounted resource controllers in the **/proc/cgroups** file.

The following controllers are available for **cgroups-v1**:

- **blkio** - can set limits on input/output access to and from block devices.
- **cpu** - can adjust the parameters of the Completely Fair Scheduler (CFS) scheduler for control group's tasks. It is mounted together with the **cpuacct** controller on the same mount.
- **cpuacct** - creates automatic reports on CPU resources used by tasks in a control group. It is mounted together with the **cpu** controller on the same mount.
- **cpuset** - can be used to restrict control group tasks to run only on a specified subset of CPUs and to direct the tasks to use memory only on specified memory nodes.
- **devices** - can control access to devices for tasks in a control group.
- **freezer** - can be used to suspend or resume tasks in a control group.
- **memory** - can be used to set limits on memory use by tasks in a control group and generates automatic reports on memory resources used by those tasks.
- **net_cls** - tags network packets with a class identifier (**classid**) that enables the Linux traffic controller (the **tc** command) to identify packets that originate from a particular control group 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 (through **iptables** command) to identify packets originating from a particular control group task.
- **net_prio** - sets the priority of network traffic.
- **pids** - can set limits for a number of processes and their children in a control group.
- **perf_event** - can group tasks for monitoring by the **perf** performance monitoring and reporting utility.
- **rdma** - can set limits on Remote Direct Memory Access/InfiniBand specific resources in a control group.
- **hugetlb** - can be used to limit the usage of large size virtual memory pages by tasks in a control group.

The following controllers are available for **cgroups-v2**:

- **io** - A follow-up to **blkio** of **cgroups-v1**.
- **memory** - A follow-up to **memory** of **cgroups-v1**.

- **pids** - Same as **pids** in **cgroups-v1**.
- **rdma** - Same as **rdma** in **cgroups-v1**.
- **cpu** - A follow-up to **cpu** and **cpuacct** of **cgroups-v1**.
- **cpuset** - Supports only the core functionality (**cpus{,.effective}**, **mems{,.effective}**) with a new partition feature.
- **perf_event** - Support is inherent, no explicit control file. You can specify a **v2 cgroup** as a parameter to the **perf** command that will profile all the tasks within that **cgroup**.

IMPORTANT

A resource controller can be used either in a **cgroups-v1** hierarchy or a **cgroups-v2** hierarchy, not simultaneously in both.

Additional resources

- **cgroups(7)** manual page
- Documentation in **/usr/share/doc/kernel-doc-<kernel_version>/Documentation/cgroups-v1/** directory (after installing the **kernel-doc** package).

24.3. WHAT ARE NAMESPACES

Namespaces are one of the most important methods for organizing and identifying software objects.

A namespace wraps a global system resource (for example a mount point, a network device, or a hostname) in an abstraction that makes it appear to processes within the namespace that they have their own isolated instance of the global resource. One of the most common technologies that utilize namespaces are containers.

Changes to a particular global resource are visible only to processes in that namespace and do not affect the rest of the system or other namespaces.

To inspect which namespaces a process is a member of, you can check the symbolic links in the **/proc/<*PID*>/ns/** directory.

The following table shows supported namespaces and resources which they isolate:

Namespace	Isolates
Mount	Mount points
UTS	Hostname and NIS domain name
IPC	System V IPC, POSIX message queues
PID	Process IDs
Network	Network devices, stacks, ports, etc

Namespace	Isolates
User	User and group IDs
Control groups	Control group root directory

Additional resources

- **namespaces(7)** and **cgroup_namespaces(7)** manual pages
- Understanding control groups

24.4. SETTING CPU LIMITS TO APPLICATIONS USING CGROUPS-V1

Sometimes an application consumes a lot of CPU time, which may negatively impact the overall health of your environment. Use the /**sys**/**fs**/ virtual file system to configure CPU limits to an application using *control groups version 1* (**cgroups-v1**).

Prerequisites

- You have root permissions.
- You have an application whose CPU consumption you want to restrict.
- You verified that the **cgroups-v1** controllers were mounted:

```
# mount -l | grep cgroup
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,seclabel,mode=755)
cgroup on /sys/fs/cgroup/systemd type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,xattr,release_agent=/usr/lib/systemd/systemd-
cgroups-agent,name=systemd)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,cpu,cpuacct)
cgroup on /sys/fs/cgroup/perf_event type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,perf_event)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,seclabel,pids)
...
```

Procedure

1. Identify the process ID (PID) of the application you want to restrict in CPU consumption:

```
# top
top - 11:34:09 up 11 min,  1 user,  load average: 0.51, 0.27, 0.22
Tasks: 267 total,   3 running, 264 sleeping,   0 stopped,   0 zombie
%Cpu(s): 49.0 us,  3.3 sy,  0.0 ni, 47.5 id,  0.0 wa,  0.2 hi,  0.0 si,  0.0 st
MiB Mem :  1826.8 total,    303.4 free,   1046.8 used,    476.5 buff/cache
MiB Swap:   1536.0 total,   1396.0 free,     140.0 used.    616.4 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 6955 root      20   0  228440   1752   1472 R  99.3   0.1   0:32.71 sha1sum
 5760 jdoe      20   0 3603868 205188  64196 S   3.7  11.0   0:17.19 gnome-shell
 6448 jdoe      20   0  743648  30640  19488 S   0.7   1.6   0:02.73 gnome-terminal-
    1 root      20   0  245300   6568   4116 S   0.3   0.4   0:01.87 systemd
  505 root      20   0       0      0      0 I   0.3   0.0   0:00.75 kworker/u4:4-events_unbound
...
```

The example output of the **top** program reveals that **PID 6955** (illustrative application **sha1sum**) consumes a lot of CPU resources.

2. Create a sub-directory in the **cpu** resource controller directory:

```
# mkdir /sys/fs/cgroup/cpu/Example/
```

The directory above represents a control group, where you can place specific processes and apply certain CPU limits to the processes. At the same time, some **cgroups-v1** interface files and **cpu** controller-specific files will be created in the directory.

3. Optionally, inspect the newly created control group:

```
# ll /sys/fs/cgroup/cpu/Example/
-rw-r—r--. 1 root root 0 Mar 11 11:42 cgroup.clone_children
-rw-r—r--. 1 root root 0 Mar 11 11:42 cgroup.procs
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.stat
-rw-r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_all
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu_sys
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu_user
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_sys
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_user
-rw-r—r--. 1 root root 0 Mar 11 11:42 cpu.cfs_period_us
-rw-r—r--. 1 root root 0 Mar 11 11:42 cpu.cfs_quota_us
-rw-r—r--. 1 root root 0 Mar 11 11:42 cpu.rt_period_us
-rw-r—r--. 1 root root 0 Mar 11 11:42 cpu.rt_runtime_us
-rw-r—r--. 1 root root 0 Mar 11 11:42 cpu.shares
-r—r—r--. 1 root root 0 Mar 11 11:42 cpu.stat
-rw-r—r--. 1 root root 0 Mar 11 11:42 notify_on_release
-rw-r—r--. 1 root root 0 Mar 11 11:42 tasks
```

The example output shows files, such as **cpuacct.usage**, **cpu.cfs._period_us**, that represent specific configurations and/or limits, which can be set for processes in the **Example** control group. Notice that the respective file names are prefixed with the name of the control group controller to which they belong.

By default, the newly created control group inherits access to the system's entire CPU resources without a limit.

4. Configure CPU limits for the control group:

```
# echo "1000000" > /sys/fs/cgroup/cpu/Example/cpu.cfs_period_us
# echo "200000" > /sys/fs/cgroup/cpu/Example/cpu.cfs_quota_us
```

The **cpu.cfs_period_us** file represents a period of time in microseconds (µs, represented here as "us") for how frequently a control group's access to CPU resources should be reallocated. The upper limit is 1 second and the lower limit is 1000 microseconds.

The **cpu.cfs_quota_us** file represents the total amount of time in microseconds for which all processes collectively in a control group can run during one period (as defined by **cpu.cfs_period_us**). As soon as processes in a control group, during a single period, use up all the time specified by the quota, they are throttled for the remainder of the period and not allowed to run until the next period. The lower limit is 1000 microseconds.

The example commands above set the CPU time limits so that all processes collectively in the **Example** control group will be able to run only for 0.2 seconds (defined by **cpu.cfs_quota_us**) out of every 1 second (defined by **cpu.cfs_period_us**).

5. Optionally, verify the limits:

```
# cat /sys/fs/cgroup/cpu/Example/cpu.cfs_period_us
/sys/fs/cgroup/cpu/Example/cpu.cfs_quota_us
1000000
200000
```

6. Add the application's PID to the **Example** control group:

```
# echo "6955" > /sys/fs/cgroup/cpu/Example/cgroup.procs

or

# echo "6955" > /sys/fs/cgroup/cpu/Example/tasks
```

The previous command ensures that a desired application becomes a member of the **Example** control group and hence does not exceed the CPU limits configured for the **Example** control group. The PID should represent an existing process in the system. The **PID 6955** here was assigned to process **sha1sum /dev/zero &**, used to illustrate the use-case of the **cpu** controller.

7. Verify that the application runs in the specified control group:

```
# cat /proc/6955/cgroup
12:cpuset:/
11:hugetlb:/
10:net_cls,net_prio:/
9:memory:/user.slice/user-1000.slice/user@1000.service
8:devices:/user.slice
7:blkio:/
6:freezer:/
5:rdma:/
4:pids:/user.slice/user-1000.slice/user@1000.service
3:perf_event:/
2:cpu,cpuacct:/Example
1:name=systemd:/user.slice/user-1000.slice/user@1000.service/gnome-terminal-
server.service
```

The example output above shows that the process of the desired application runs in the **Example** control group, which applies CPU limits to the application's process.

8. Identify the current CPU consumption of your throttled application:

```
# top
top - 12:28:42 up  1:06,  1 user,  load average: 1.02, 1.02, 1.00
Tasks: 266 total,   6 running, 260 sleeping,   0 stopped,   0 zombie
```

```
%Cpu(s): 11.0 us,  1.2 sy,  0.0 ni, 87.5 id,  0.0 wa,  0.2 hi,  0.0 si,  0.2 st
MiB Mem :   1826.8 total,    287.1 free,   1054.4 used,    485.3 buff/cache
MiB Swap:   1536.0 total,   1396.7 free,    139.2 used.    608.3 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 6955 root      20   0  228440   1752   1472 R  20.6   0.1  47:11.43 sha1sum
 5760 jdoe      20   0 3604956 208832  65316 R   2.3  11.2   0:43.50 gnome-shell
 6448 jdoe      20   0  743836  31736  19488 S   0.7   1.7   0:08.25 gnome-terminal-
  505 root      20   0       0      0      0 I   0.3   0.0   0:03.39 kworker/u4:4-events_unbound
 4217 root      20   0   74192   1612   1320 S   0.3   0.1   0:01.19 spice-vdagentd
...
```

Notice that the CPU consumption of the **PID 6955** has decreased from 99% to 20%.

IMPORTANT

The **cgroups-v2** counterpart for **cpu.cfs_period_us** and **cpu.cfs_quota_us** is the **cpu.max** file. The **cpu.max** file is available through the **cpu** controller.

Additional resources

- Understanding control groups
- What kernel resource controllers are
- **cgroups(7)**, **sysfs(5)** manual pages

[1] Linux Control Group v2 - An Introduction, Devconf.cz 2019 presentation by Waiman Long

CHAPTER 25. USING CGROUPS-V2 TO CONTROL DISTRIBUTION OF CPU TIME FOR APPLICATIONS

Some applications use too much CPU time, which can negatively impact the overall health of your environment. You can put your applications into *control groups version 2* (**cgroups-v2**) and configure CPU limits for those control groups. As a result, you can regulate your applications in CPU consumption.

The user has two methods how to regulate distribution of CPU time allocated to a control group:

- Setting CPU bandwidth (editing the **cpu.max** controller file)
- Setting CPU weight (editing the **cpu.weight** controller file)

25.1. MOUNTING CGROUPS-V2

During the boot process, RHEL 8 mounts the **cgroup-v1** virtual filesystem by default. To utilize **cgroup-v2** functionality in limiting resources for your applications, manually configure the system.

Prerequisites

- You have root permissions.

Procedure

1. Configure the system to mount **cgroups-v2** by default during system boot by the **systemd** system and service manager:

   ```
   # grubby --update-kernel=/boot/vmlinuz-$(uname -r) --
   args="systemd.unified_cgroup_hierarchy=1"
   ```

   This adds the necessary kernel command-line parameter to the current boot entry.

   To add the **systemd.unified_cgroup_hierarchy=1** parameter to all kernel boot entries:

   ```
   # grubby --update-kernel=ALL --args="systemd.unified_cgroup_hierarchy=1"
   ```

2. Reboot the system for the changes to take effect.

Verification steps

1. Optionally, verify that the **cgroups-v2** filesystem was mounted:

   ```
   # mount -l | grep cgroup
   cgroup2 on /sys/fs/cgroup type cgroup2
   (rw,nosuid,nodev,noexec,relatime,seclabel,nsdelegate)
   ```

   The **cgroups-v2** filesystem was successfully mounted on the **/sys/fs/cgroup/** directory.

2. Optionally, inspect the contents of the **/sys/fs/cgroup/** directory:

   ```
   # ll /sys/fs/cgroup/
   -r—r—r--.  1 root root 0 Apr 29 12:03 cgroup.controllers
   -rw-r—r--.  1 root root 0 Apr 29 12:03 cgroup.max.depth
   ```

```
-rw-r—r--.  1 root root 0 Apr 29 12:03 cgroup.max.descendants
-rw-r—r--.  1 root root 0 Apr 29 12:03 cgroup.procs
-r—r—r--.  1 root root 0 Apr 29 12:03 cgroup.stat
-rw-r—r--.  1 root root 0 Apr 29 12:18 cgroup.subtree_control
-rw-r—r--.  1 root root 0 Apr 29 12:03 cgroup.threads
-rw-r—r--.  1 root root 0 Apr 29 12:03 cpu.pressure
-r—r—r--.  1 root root 0 Apr 29 12:03 cpuset.cpus.effective
-r—r—r--.  1 root root 0 Apr 29 12:03 cpuset.mems.effective
-r—r—r--.  1 root root 0 Apr 29 12:03 cpu.stat
drwxr-xr-x.  2 root root 0 Apr 29 12:03 init.scope
-rw-r—r--.  1 root root 0 Apr 29 12:03 io.pressure
-r—r—r--.  1 root root 0 Apr 29 12:03 io.stat
-rw-r—r--.  1 root root 0 Apr 29 12:03 memory.pressure
-r—r—r--.  1 root root 0 Apr 29 12:03 memory.stat
drwxr-xr-x. 69 root root 0 Apr 29 12:03 system.slice
drwxr-xr-x.  3 root root 0 Apr 29 12:18 user.slice
```

The **/sys/fs/cgroup/** directory, also called the *root control group*, by default, contains interface files (starting with **cgroup**) and controller-specific files such as **cpuset.cpus.effective**. In addition, there are some directories related to **systemd**, such as, **/sys/fs/cgroup/init.scope**, **/sys/fs/cgroup/system.slice**, and **/sys/fs/cgroup/user.slice**.

Additional resources

- Understanding control groups
- What kernel resource controllers are
- **cgroups(7)**, **sysfs(5)** manual pages

25.2. PREPARING THE CGROUP FOR DISTRIBUTION OF CPU TIME

To control CPU consumption of your applications, you need to enable specific CPU controllers and create a dedicated control groups. It is recommended to create at least two levels of child control groups inside the **/sys/fs/cgroup/** root control group to maintain better organizational clarity of **cgroup** files.

Prerequisites

- You have identified PIDs of processes that you want to control.
- You have root permissions.
- You have mounted **cgroups-v2** filesystem.

Procedure

1. Identify the process IDs (PIDs) of applications whose CPU consumption you want to constrict:

```
# top
Tasks: 104 total,   3 running, 101 sleeping,   0 stopped,   0 zombie
%Cpu(s): 17.6 us, 81.6 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.8 hi,  0.0 si,  0.0 st
MiB Mem :   3737.4 total,   3312.7 free,    133.3 used,    291.4 buff/cache
MiB Swap:   4060.0 total,   4060.0 free,       0.0 used.   3376.1 avail Mem
```

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
34578 root      20   0   18720   1756   1468 R  99.0   0.0   0:31.09 sha1sum
34579 root      20   0   18720   1772   1480 R  99.0   0.0   0:30.54 sha1sum
    1 root      20   0  186192  13940   9500 S   0.0   0.4   0:01.60 systemd
    2 root      20   0       0      0      0 S   0.0   0.0   0:00.01 kthreadd
    3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
    4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
...
```

The example output reveals that **PID 34578** and **34579** (two illustrative applications of **sha1sum**) consume a lot of resources, namely CPU. Both are example applications used to demonstrate managing the **cgroups-v2** functionality.

2. Verify that the **cpu** and **cpuset** controllers are available in the **/sys/fs/cgroup/cgroup.controllers** file:

```
# cat /sys/fs/cgroup/cgroup.controllers
cpuset cpu io memory hugetlb pids rdma
```

3. Enable CPU-related controllers:

```
# echo "+cpu" >> /sys/fs/cgroup/cgroup.subtree_control
# echo "+cpuset" >> /sys/fs/cgroup/cgroup.subtree_control
```

These commands enable the **cpu** and **cpuset** controllers for the immediate children groups of the **/sys/fs/cgroup/** root control group. A *child group* is where you can specify processes and apply control checks to each of the processes based on your criteria.

Users can read the contents of the **cgroup.subtree_control** file at any level to get an idea of what controllers are going to be available for enablement in the immediate child group.

NOTE

By default, the **/sys/fs/cgroup/cgroup.subtree_control** file in the root control group contains **memory** and **pids** controllers.

4. Create the **/sys/fs/cgroup/Example/** directory:

```
# mkdir /sys/fs/cgroup/Example/
```

The **/sys/fs/cgroup/Example/** directory defines a child group. Also, the previous step enabled the **cpu** and **cpuset** controllers for this child group.

When you create the **/sys/fs/cgroup/Example/** directory, some **cgroups-v2** interface files and **cpu** and **cpuset** controller-specific files are automatically created in the directory. The **/sys/fs/cgroup/Example/** directory contains also controller-specific files for the **memory** and **pids** controllers.

5. Optionally, inspect the newly created child control group:

```
# ll /sys/fs/cgroup/Example/
-r—r—r--. 1 root root 0 Jun  1 10:33 cgroup.controllers
-r—r—r--. 1 root root 0 Jun  1 10:33 cgroup.events
-rw-r—r--. 1 root root 0 Jun  1 10:33 cgroup.freeze
```

```
-rw-r—r--. 1 root root 0 Jun  1 10:33 cgroup.max.depth
-rw-r—r--. 1 root root 0 Jun  1 10:33 cgroup.max.descendants
-rw-r—r--. 1 root root 0 Jun  1 10:33 cgroup.procs
-r—r—r--. 1 root root 0 Jun  1 10:33 cgroup.stat
-rw-r—r--. 1 root root 0 Jun  1 10:33 cgroup.subtree_control
…
-rw-r—r--. 1 root root 0 Jun  1 10:33 cpuset.cpus
-r—r—r--. 1 root root 0 Jun  1 10:33 cpuset.cpus.effective
-rw-r—r--. 1 root root 0 Jun  1 10:33 cpuset.cpus.partition
-rw-r—r--. 1 root root 0 Jun  1 10:33 cpuset.mems
-r—r—r--. 1 root root 0 Jun  1 10:33 cpuset.mems.effective
-r—r—r--. 1 root root 0 Jun  1 10:33 cpu.stat
-rw-r—r--. 1 root root 0 Jun  1 10:33 cpu.weight
-rw-r—r--. 1 root root 0 Jun  1 10:33 cpu.weight.nice
…
-r—r—r--. 1 root root 0 Jun  1 10:33 memory.events.local
-rw-r—r--. 1 root root 0 Jun  1 10:33 memory.high
-rw-r—r--. 1 root root 0 Jun  1 10:33 memory.low
…
-r—r—r--. 1 root root 0 Jun  1 10:33 pids.current
-r—r—r--. 1 root root 0 Jun  1 10:33 pids.events
-rw-r—r--. 1 root root 0 Jun  1 10:33 pids.max
```

The example output shows files such as **cpuset.cpus** and **cpu.max**. These files are specific to the **cpuset** and **cpu** controllers. The **cpuset** and **cpu** controllers are manually enabled for the root's (**/sys/fs/cgroup/**) *direct child control groups* using the **/sys/fs/cgroup/cgroup.subtree_control** file.

The directory also includes general **cgroup** control interface files such as **cgroup.procs** or **cgroup.controllers**, which are common to all control groups, regardless of enabled controllers.

The files such as **memory.high** and **pids.max** relate to the **memory** and **pids** controllers, which are in the root control group (**/sys/fs/cgroup/**), and are always enabled by default.

By default, the newly created child group inherits access to all of the system's CPU and memory resources, without any limits.

6. Enable the CPU-related controllers in **/sys/fs/cgroup/Example/** to obtain controllers that are relevant only to CPU:

   ```
   # echo "+cpu" >> /sys/fs/cgroup/Example/cgroup.subtree_control
   # echo "+cpuset" >> /sys/fs/cgroup/Example/cgroup.subtree_control
   ```

   These commands ensure that the immediate child control group will *only* have controllers relevant to regulate the CPU time distribution - not to **memory** or **pids** controllers.

7. Create the **/sys/fs/cgroup/Example/tasks/** directory:

   ```
   # mkdir /sys/fs/cgroup/Example/tasks/
   ```

   The **/sys/fs/cgroup/Example/tasks/** directory defines a child group with files that relate purely to **cpu** and **cpuset** controllers.

8. Optionally, inspect another child control group:

   ```
   # ll /sys/fs/cgroup/Example/tasks
   ```

```
-r—r—r--. 1 root root 0 Jun  1 11:45 cgroup.controllers
-r—r—r--. 1 root root 0 Jun  1 11:45 cgroup.events
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.freeze
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.max.depth
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.max.descendants
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.procs
-r—r—r--. 1 root root 0 Jun  1 11:45 cgroup.stat
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.subtree_control
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.threads
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.type
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpu.max
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpu.pressure
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpuset.cpus
-r—r—r--. 1 root root 0 Jun  1 11:45 cpuset.cpus.effective
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpuset.cpus.partition
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpuset.mems
-r—r—r--. 1 root root 0 Jun  1 11:45 cpuset.mems.effective
-r—r—r--. 1 root root 0 Jun  1 11:45 cpu.stat
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpu.weight
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpu.weight.nice
-rw-r—r--. 1 root root 0 Jun  1 11:45 io.pressure
-rw-r—r--. 1 root root 0 Jun  1 11:45 memory.pressure
```

9. Ensure the processes that you want to control for CPU time compete on the same CPU:

```
# echo "1" > /sys/fs/cgroup/Example/tasks/cpuset.cpus
```

The previous command ensures that the processes you will place in the **Example/tasks** child control group, compete on the same CPU. This setting is important for the **cpu** controller to activate.

IMPORTANT

The **cpu** controller is only activated if the relevant child control group has at least 2 processes which compete for time on a single CPU.

Verification steps

1. Optional: ensure that the CPU-related controllers are enabled for the immediate children cgroups:

```
# cat /sys/fs/cgroup/cgroup.subtree_control
/sys/fs/cgroup/Example/cgroup.subtree_control
cpuset cpu memory pids
cpuset cpu
```

2. Optional: ensure the processes that you want to control for CPU time compete on the same CPU:

```
# cat /sys/fs/cgroup/Example/tasks/cpuset.cpus
1
```

Additional resources

- Understanding control groups
- What kernel resource controllers are
- Mounting cgroups-v2
- **cgroups(7)**, **sysfs(5)** manual pages

25.3. CONTROLLING DISTRIBUTION OF CPU TIME FOR APPLICATIONS BY ADJUSTING CPU BANDWIDTH

You need to assign values to the relevant files of the **cpu** controller to regulate distribution of the CPU time to applications under the specific cgroup tree.

Prerequisites

- You have root permissions.
- You have at least two applications for which you want to control distribution of CPU time.
- You ensured the relevant applications compete for CPU time on the same CPU as described in Preparing the cgroup for distribution of CPU time .
- You mounted **cgroups-v2** filesystem as described in Mounting cgroups-v2.
- You enabled **cpu** and **cpuset** controllers both in the parent control group and in child control group similarly as described in Preparing the cgroup for distribution of CPU time .
- You created two levels of *child control groups* inside the **/sys/fs/cgroup/** *root control group* as in the example below:

```
…
  ├── Example
  │   ├── tasks
…
```

Procedure

1. Configure CPU bandwidth to achieve resource restrictions within the control group:

   ```
   # echo "200000 1000000" > /sys/fs/cgroup/Example/tasks/cpu.max
   ```

   The first value is the allowed time quota in microseconds for which all processes collectively in a child group can run during one period. The second value specifies the length of the period.

   During a single period, when processes in a control group collectively exhaust the time specified by this quota, they are throttled for the remainder of the period and not allowed to run until the next period.

   This command sets CPU time distribution controls so that all processes collectively in the **/sys/fs/cgroup/Example/tasks** child group can run on the CPU for only 0.2 seconds of every 1 second. That is, one fifth of each second.

2. Optionally, verify the time quotas:

```
# cat /sys/fs/cgroup/Example/tasks/cpu.max
200000 1000000
```

3. Add the applications' PIDs to the **Example/tasks** child group:

```
# echo "34578" > /sys/fs/cgroup/Example/tasks/cgroup.procs
# echo "34579" > /sys/fs/cgroup/Example/tasks/cgroup.procs
```

The example commands ensure that desired applications become members of the **Example/tasks** child group and do not exceed the CPU time distribution configured for this child group.

Verification steps

1. Verify that the applications run in the specified control group:

```
# cat /proc/34578/cgroup /proc/34579/cgroup
0::/Example/tasks
0::/Example/tasks
```

The output above shows the processes of the specified applications that run in the **Example/tasks** child group.

2. Inspect the current CPU consumption of the throttled applications:

```
# top
top - 11:13:53 up 23:10,  1 user,  load average: 0.26, 1.33, 1.66
Tasks: 104 total,   3 running, 101 sleeping,   0 stopped,   0 zombie
%Cpu(s):  3.0 us,  7.0 sy,  0.0 ni, 89.5 id,  0.0 wa,  0.2 hi,  0.2 si,  0.2 st
MiB Mem :   3737.4 total,   3312.6 free,    133.4 used,    291.4 buff/cache
MiB Swap:   4060.0 total,   4060.0 free,      0.0 used.   3376.0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  34578 root      20   0   18720   1756   1468 R  10.0   0.0  37:36.13 sha1sum
  34579 root      20   0   18720   1772   1480 R  10.0   0.0  37:41.22 sha1sum
      1 root      20   0  186192  13940   9500 S   0.0   0.4   0:01.60 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.01 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
...
```

Notice that the CPU consumption for the **PID 34578** and **PID 34579** has decreased to 10%. The **Example/tasks** child group regulates its processes to 20% of the CPU time collectively. Since there are 2 processes in the control group, each can utilize 10% of the CPU time.

Additional resources

- Understanding control groups
- What kernel resource controllers are
- Mounting cgroups-v2
- Preparing the cgroup for distribution of CPU time

- **cgroups(7)**, **sysfs(5)** manual pages

25.4. CONTROLLING DISTRIBUTION OF CPU TIME FOR APPLICATIONS BY ADJUSTING CPU WEIGHT

You need to assign values to the relevant files of the **cpu** controller to regulate distribution of the CPU time to applications under the specific cgroup tree.

Prerequisites

- You have root permissions.
- You have applications for which you want to control distribution of CPU time.
- You ensured the relevant applications compete for CPU time on the same CPU as described in Preparing the cgroup for distribution of CPU time .
- You mounted **cgroups-v2** filesystem as described in Mounting cgroups-v2.
- You created a two level hierarchy of *child control groups* inside the **/sys/fs/cgroup/** *root control group* as in the following example:

```
…
  ├── Example
  │   ├── g1
  │   ├── g2
  │   └── g3
…
```

- You enabled **cpu** and **cpuset** controllers in the parent control group and in child control groups similarly as described in Preparing the cgroup for distribution of CPU time .

Procedure

1. Configure desired CPU weights to achieve resource restrictions within the control groups:

```
# echo "150" > /sys/fs/cgroup/Example/g1/cpu.weight
# echo "100" > /sys/fs/cgroup/Example/g2/cpu.weight
# echo "50" > /sys/fs/cgroup/Example/g3/cpu.weight
```

2. Add the applications' PIDs to the **g1**, **g2**, and **g3** child groups:

```
# echo "33373" > /sys/fs/cgroup/Example/g1/cgroup.procs
# echo "33374" > /sys/fs/cgroup/Example/g2/cgroup.procs
# echo "33377" > /sys/fs/cgroup/Example/g3/cgroup.procs
```

   The example commands ensure that desired applications become members of the **Example/g*/** child cgroups and will get their CPU time distributed as per the configuration of those cgroups.

   The weights of the children cgroups (**g1**, **g2**, **g3**) that have running processes are summed up at the level of the parent cgroup (**Example**). The CPU resource is then distributed proportionally based on the respective weights.

As a result, when all processes run at the same time, the kernel allocates to each of them the proportionate CPU time based on their respective cgroup's **cpu.weight** file:

Child cgroup	**cpu.weight** file	CPU time allocation
g1	150	~50% (150/300)
g2	100	~33% (100/300)
g3	50	~16% (50/300)

The value of the **cpu.weight** controller file is not a percentage.

If one process stopped running, leaving cgroup **g2** with no running processes, the calculation would omit the cgroup **g2** and only account weights of cgroups **g1** and **g3**:

Child cgroup	**cpu.weight** file	CPU time allocation
g1	150	~75% (150/200)
g3	50	~25% (50/200)

IMPORTANT

If a child cgroup had multiple running processes, the CPU time allocated to the respective cgroup would be distributed equally to the member processes of that cgroup.

Verification

1. Verify that the applications run in the specified control groups:

```
# cat /proc/33373/cgroup /proc/33374/cgroup /proc/33377/cgroup
0::/Example/g1
0::/Example/g2
0::/Example/g3
```

The command output shows the processes of the specified applications that run in the **Example/g*/** child cgroups.

2. Inspect the current CPU consumption of the throttled applications:

```
# top
top - 05:17:18 up 1 day, 18:25,  1 user,  load average: 3.03, 3.03, 3.00
Tasks:  95 total,   4 running,  91 sleeping,   0 stopped,   0 zombie
%Cpu(s): 18.1 us, 81.6 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.3 hi,  0.0 si,  0.0 st
MiB Mem :   3737.0 total,   3233.7 free,    132.8 used,    370.5 buff/cache
MiB Swap:   4060.0 total,   4060.0 free,      0.0 used.   3373.1 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  33373 root      20   0   18720   1748   1460 R  49.5   0.0 415:05.87 sha1sum
```

```
 33374 root      20   0   18720   1756   1464 R  32.9   0.0 412:58.33 sha1sum
 33377 root      20   0   18720   1860   1568 R  16.3   0.0 411:03.12 sha1sum
   760 root      20   0  416620  28540  15296 S   0.3   0.7   0:10.23 tuned
     1 root      20   0  186328  14108   9484 S   0.0   0.4   0:02.00 systemd
     2 root      20   0       0      0      0 S   0.0   0.0   0:00.01 kthread
...
```

NOTE

We forced all the example processes to run on a single CPU for clearer illustration. The CPU weight applies the same principles also when used on multiple CPUs.

Notice that the CPU resource for the **PID 33373**, **PID 33374**, and **PID 33377** was allocated based on the weights, 150, 100, 50, you assigned to the respective child cgroups. The weights correspond to around 50%, 33%, and 16% allocation of CPU time for each application.

Additional resources

- Understanding control groups
- What are kernel resource controllers
- Mounting cgroups-v2
- Preparing the cgroup for distribution of CPU time
- Resource Distribution Models
- **cgroups(7)**, **sysfs(5)** manual pages

CHAPTER 26. USING CONTROL GROUPS VERSION 1 WITH SYSTEMD

The following sections provide an overview of tasks related to creation, modification and removal of the control groups (**cgroups**). The utilities provided by the **systemd** system and service manager are the preferred way of the **cgroups** management and will be supported in the future.

26.1. ROLE OF SYSTEMD IN CONTROL GROUPS VERSION 1

RHEL 8 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 the system resources with the **systemctl** command, or by modifying the **systemd** unit files.

By default, the **systemd** system and service manager makes use of the **slice**, the **scope** and the **service** units to organize and structure processes in the control groups. The **systemctl** command enables you to further modify this structure by creating custom **slices**. Also, **systemd** automatically mounts hierarchies for important kernel resource controllers in the **/sys/fs/cgroup/** directory.

Three **systemd** unit types are used for resource control:

- **Service** - A process or a group of processes, which **systemd** started according to 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
  ```

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

  ```
  <name>.scope
  ```

- **Slice** - A group of hierarchically organized units. Slices organize a hierarchy in which scopes and services are placed. The actual processes are contained in scopes or in services. 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 to a slice from the **-.slice** root slice. In the following example:

  ```
  <parent-name>.slice
  ```

  parent-name.slice is a sub-slice of **parent.slice**, which is a sub-slice of the **-.slice** root slice. **parent-name.slice** can have its own sub-slice named **parent-name-name2.slice**, and so on.

The **service**, the **scope**, and the **slice** units directly map to objects in the control group hierarchy. When these units are activated, they map directly to control group paths built from the unit names.

The following is an abbreviated example of a control group hierarchy:

```
Control group /:
-.slice
├─user.slice
│ ├─user-42.slice
│ │ ├─session-c1.scope
```

```
│ │ │ ├─ 967 gdm-session-worker [pam/gdm-launch-environment]
│ │ │ ├─1035 /usr/libexec/gdm-x-session gnome-session --autostart
/usr/share/gdm/greeter/autostart
│ │ │ ├─1054 /usr/libexec/Xorg vt1 -displayfd 3 -auth /run/user/42/gdm/Xauthority -background none
-noreset -keeptty -verbose 3
│ │ │ ├─1212 /usr/libexec/gnome-session-binary --autostart /usr/share/gdm/greeter/autostart
│ │ │ ├─1369 /usr/bin/gnome-shell
│ │ │ ├─1732 ibus-daemon --xim --panel disable
│ │ │ ├─1752 /usr/libexec/ibus-dconf
│ │ │ ├─1762 /usr/libexec/ibus-x11 --kill-daemon
│ │ │ ├─1912 /usr/libexec/gsd-xsettings
│ │ │ ├─1917 /usr/libexec/gsd-a11y-settings
│ │ │ ├─1920 /usr/libexec/gsd-clipboard
…
├─init.scope
│ └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
└─system.slice
  ├─rngd.service
  │ └─800 /sbin/rngd -f
  ├─systemd-udevd.service
  │ └─659 /usr/lib/systemd/systemd-udevd
  ├─chronyd.service
  │ └─823 /usr/sbin/chronyd
  ├─auditd.service
  │ ├─761 /sbin/auditd
  │ └─763 /usr/sbin/sedispatch
  ├─accounts-daemon.service
  │ └─876 /usr/libexec/accounts-daemon
  ├─example.service
  │ ├─ 929 /bin/bash /home/jdoe/example.sh
  │ └─4902 sleep 1
  …
```

The example above shows that services and scopes contain processes and are placed in slices that do not contain processes of their own.

Additional resources

- *Configuring basic system settings* in Red Hat Enterprise Linux
- What are kernel resource controllers
- **systemd.resource-control(5)**, **cgroups(7)**, **fork()**, **fork(2)** manual pages

26.2. CREATING TRANSIENT CONTROL GROUPS

The transient **cgroups** set limits on resources consumed by a unit (service or scope) during its runtime.

Procedure

- To create a transient control group, use the **systemd-run** command in the following format:

  ```
  # systemd-run --unit=<name> --slice=<name>.slice <command>
  ```

This command creates and starts a transient service or a scope unit and runs a custom command in such a unit.

- The **--unit=<name>** option gives a name to the unit. If **--unit** is not specified, the name is generated automatically.
- The **--slice=<*name*>.slice** option makes your service or scope unit a member of a specified slice. Replace **<*name*>.slice** 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 or the scope unit.
  The following message is displayed to confirm that you created and started the service or the scope successfully:

```
# Running as unit <name>.service
```

- Optionally, keep the unit running after its processes finished to collect run-time information:

```
# systemd-run --unit=<name> --slice=<name>.slice --remain-after-exit <command>
```

The command creates and starts a transient service unit and runs a custom command in such a unit. The **--remain-after-exit** option ensures that the service keeps running after its processes have finished.

Additional resources

- Understanding control groups
- Role of systemd in control groups
- *Configuring basic system settings* in RHEL
- the **systemd-run(1)** manual page

26.3. CREATING PERSISTENT CONTROL GROUPS

To assign a persistent control group to a service, it is necessary to 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.

Procedure

- To create a persistent control group, execute:

```
# systemctl enable <name>.service
```

The command above automatically creates a unit configuration file into the **/usr/lib/systemd/system/** directory and by default, it assigns **<*name*>.service** to the **system.slice** unit.

Additional resources

- Understanding control groups
- Role of systemd in control groups
- *Configuring basic system settings* in RHEL
- **systemd-run(1)** manual page

26.4. CONFIGURING MEMORY RESOURCE CONTROL SETTINGS ON THE COMMAND-LINE

Executing commands in the command-line interface is one of the ways how to set limits, prioritize, or control access to hardware resources for groups of processes.

Procedure

- To limit the memory usage of a service, run the following:

  ```
  # systemctl set-property example.service MemoryMax=1500K
  ```

  The command instantly assigns the memory limit of 1,500 KB to processes executed in a control group the **example.service** service belongs to. The **MemoryMax** parameter, in this configuration variant, is defined in the **/etc/systemd/system.control/example.service.d/50-MemoryMax.conf** file and controls the value of the **/sys/fs/cgroup/memory/system.slice/example.service/memory.limit_in_bytes** file.

- Optionally, to temporarily limit the memory usage of a service, run:

  ```
  # systemctl set-property --runtime example.service MemoryMax=1500K
  ```

  The command instantly assigns the memory limit to the **example.service** service. The **MemoryMax** parameter is defined until the next reboot in the **/run/systemd/system.control/example.service.d/50-MemoryMax.conf** file. With a reboot, the whole **/run/systemd/system.control/** directory and **MemoryMax** are removed.

NOTE

The **50-MemoryMax.conf** file stores the memory limit as a multiple of 4096 bytes - one kernel page size specific for AMD64 and Intel 64. The actual number of bytes depends on a CPU architecture.

Additional resources

- Understanding control groups
- What kernel resource controllers are
- **systemd.resource-control(5)** and **cgroups(7)** manual pages
- Role of systemd in control groups

26.5. CONFIGURING MEMORY RESOURCE CONTROL SETTINGS WITH UNIT FILES

Each persistent unit is supervised by the **systemd** system and service manager, and has a unit configuration file in the **/usr/lib/systemd/system/** directory. To change the resource control settings of the persistent units, modify its unit configuration file either manually in a text editor or from the command-line interface.

Manually modifying unit files is one of the ways how to set limits, prioritize, or control access to hardware resources for groups of processes.

Procedure

1. To limit the memory usage of a service, modify the **/usr/lib/systemd/system/example.service** file as follows:

   ```
   …
   [Service]
   MemoryMax=1500K
   …
   ```

   The configuration above places a limit on maximum memory consumption of processes executed in a control group, which **example.service** is part of.

   > **NOTE**
   >
   > Use suffixes K, M, G, or T to identify Kilobyte, Megabyte, Gigabyte, or Terabyte as a unit of measurement.

2. Reload all unit configuration files:

   ```
   # systemctl daemon-reload
   ```

3. Restart the service:

   ```
   # systemctl restart example.service
   ```

4. Reboot the system.

5. Optionally, check that the changes took effect:

   ```
   # cat /sys/fs/cgroup/memory/system.slice/example.service/memory.limit_in_bytes
   1536000
   ```

   The example output shows that the memory consumption was limited at around 1,500 KB.

   > **NOTE**
   >
   > The **memory.limit_in_bytes** file stores the memory limit as a multiple of 4096 bytes - one kernel page size specific for AMD64 and Intel 64. The actual number of bytes depends on a CPU architecture.

Additional resources

- Understanding control groups

- What kernel resource controllers are
- **systemd.resource-control(5)**, **cgroups(7)** manual pages
- *Configuring basic system settings* in RHEL
- Role of systemd in control groups

26.6. REMOVING TRANSIENT CONTROL GROUPS

You can use the **systemd** system and service manager to remove transient control groups (**cgroups**) if you no longer need to limit, prioritize, or control access to hardware resources for groups of processes.

Transient **cgroups** are automatically released once all the processes that a service or a scope unit contains, finish.

Procedure

- To stop the service unit with all its processes, execute:

  ```
  # systemctl stop name.service
  ```

- To terminate one or more of the unit processes, execute:

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

  The command above uses the **--kill-who** option to select process(es) from the control group you wish to terminate. To kill multiple processes at the same time, pass a comma-separated list of PIDs. The **--signal** option determines the type of POSIX signal to be sent to the specified processes. The default signal is *SIGTERM*.

Additional resources

- Understanding control groups
- What are kernel resource controllers
- **systemd.resource-control(5)**, **cgroups(7)** manual pages
- Role of systemd in control groups
- *Configuring basic system settings* in RHEL

26.7. REMOVING PERSISTENT CONTROL GROUPS

You can use the **systemd** system and service manager to remove persistent control groups (**cgroups**) if you no longer need to limit, prioritize, or control access to hardware resources for groups of processes.

Persistent **cgroups** are released when a service or a scope unit is stopped or disabled and its configuration file is deleted.

Procedure

1. Stop the service unit:

```
# systemctl stop <name>.service
```

2. Disable the service unit:

```
# systemctl disable <name>.service
```

3. Remove the relevant unit configuration file:

```
# rm /usr/lib/systemd/system/<name>.service
```

4. Reload all unit configuration files so that changes take effect:

```
# systemctl daemon-reload
```

Additional resources

- Understanding control groups
- What kernel resource controllers are
- **systemd.resource-control(5)**, **cgroups(7)**, and **systemd.kill(5)** manual pages
- Role of systemd in control groups
- *Configuring basic system settings* in RHEL

26.8. LISTING SYSTEMD UNITS

The following procedure describes how to use the **systemd** system and service manager to list its units.

Procedure

- To list all active units on the system, execute the **# systemctl** command and the terminal will return an output similar to the following example:

```
# systemctl
UNIT                              LOAD   ACTIVE SUB      DESCRIPTION
…
init.scope                         loaded active running   System and Service Manager
session-2.scope                        loaded active running   Session 2 of user jdoe
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
…
-.slice                          loaded active active    Root Slice
machine.slice                         loaded active active    Virtual Machine and Container
Slice system-getty.slice                                    loaded active active
system-getty.slice
system-lvm2\x2dpvscan.slice                   loaded active active    system-
lvm2\x2dpvscan.slice
system-sshd\x2dkeygen.slice                   loaded active active    system-
sshd\x2dkeygen.slice
```

```
system-systemd\x2dhibernate\x2dresume.slice        loaded active active    system-
systemd\x2dhibernate\x2dresume>
system-user\x2druntime\x2ddir.slice                loaded active active    system-
user\x2druntime\x2ddir.slice
system.slice                                       loaded active active    System Slice
user-1000.slice                                    loaded active active    User Slice of UID 1000
user-42.slice                                      loaded active active    User Slice of UID 42
user.slice                                         loaded active active    User and Session Slice
…
```

- **UNIT** - a name of a unit that also reflects the unit position in a control group hierarchy. The units relevant for resource control are a *slice*, a *scope*, and a *service*.
- **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 content and functionality.

- To list inactive units, execute:

  ```
  # systemctl --all
  ```

- To limit the amount of information in the output, execute:

  ```
  # systemctl --type service,masked
  ```

  The **--type** option requires a comma-separated list of unit types such as a *service* and a *slice*, or unit load states such as *loaded* and *masked*.

Additional resources

- *Configuring basic system settings* in RHEL
- **systemd.resource-control(5)**, **systemd.exec(5)** manual pages

26.9. VIEWING SYSTEMD CONTROL GROUP HIERARCHY

The following procedure describes how to display control groups (**cgroups**) hierarchy and processes running in specific **cgroups**.

Procedure

- To display the whole **cgroups** hierarchy on your system, execute **# systemd-cgls**:

  ```
  # systemd-cgls
  Control group /:
  -.slice
  ├─user.slice
  │ ├─user-42.slice
  ```

```
│ │ ├─session-c1.scope
│ │ │ ├─ 965 gdm-session-worker [pam/gdm-launch-environment]
│ │ │ ├─1040 /usr/libexec/gdm-x-session gnome-session --autostart
/usr/share/gdm/greeter/autostart
…
├─init.scope
│ └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
└─system.slice
  …
  ├─example.service
  │ ├─6882 /bin/bash /home/jdoe/example.sh
  │ └─6902 sleep 1
  ├─systemd-journald.service
    └─629 /usr/lib/systemd/systemd-journald
  …
```

The example output returns the entire **cgroups** hierarchy, where the highest level is formed by *slices*.

- To display the **cgroups** hierarchy filtered by a resource controller, execute **# systemd-cgls <*resource_controller*>**:

```
# systemd-cgls memory
Controller memory; Control group /:
├─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
├─user.slice
│ ├─user-42.slice
│ │ ├─session-c1.scope
│ │ │ ├─ 965 gdm-session-worker [pam/gdm-launch-environment]
…
└─system.slice
  |
  …
  ├─chronyd.service
  │ └─844 /usr/sbin/chronyd
  ├─example.service
  │ ├─8914 /bin/bash /home/jdoe/example.sh
  │ └─8916 sleep 1
  …
```

The example output of the above command lists the services that interact with the selected controller.

- To display detailed information about a certain unit and its part of the **cgroups** hierarchy, execute **# systemctl status <*system_unit*>**:

```
# systemctl status example.service
● example.service - My example service
   Loaded: loaded (/usr/lib/systemd/system/example.service; enabled; vendor preset:
disabled)
   Active: active (running) since Tue 2019-04-16 12:12:39 CEST; 3s ago
 Main PID: 17737 (bash)
    Tasks: 2 (limit: 11522)
   Memory: 496.0K (limit: 1.5M)
   CGroup: /system.slice/example.service
           ├─17737 /bin/bash /home/jdoe/example.sh
```

```
          └─17743 sleep 1
Apr 16 12:12:39 redhat systemd[1]: Started My example service.
Apr 16 12:12:39 redhat bash[17737]: The current time is Tue Apr 16 12:12:39 CEST 2019
Apr 16 12:12:40 redhat bash[17737]: The current time is Tue Apr 16 12:12:40 CEST 2019
```

Additional resources

- What are kernel resource controllers
- **systemd.resource-control(5)**, **cgroups(7)** manual pages

26.10. VIEWING RESOURCE CONTROLLERS

The following procedure describes how to learn which processes use which resource controllers.

Procedure

1. To view which resource controllers a process interacts with, execute the **# cat proc/<*PID*>/cgroup** command:

   ```
   # cat /proc/11269/cgroup
   12:freezer:/
   11:cpuset:/
   10:devices:/system.slice
   9:memory:/system.slice/example.service
   8:pids:/system.slice/example.service
   7:hugetlb:/
   6:rdma:/
   5:perf_event:/
   4:cpu,cpuacct:/
   3:net_cls,net_prio:/
   2:blkio:/
   1:name=systemd:/system.slice/example.service
   ```

   The example output relates to a process of interest. In this case, it is a process identified by **PID 11269**, which belongs to the **example.service** unit. You can determine whether the process was placed in a correct control group as defined by the **systemd** unit file specifications.

   NOTE

   By default, the items and their ordering in the list of resource controllers is the same for all units started by **systemd**, since it automatically mounts all the default resource controllers.

Additional resources

- **cgroups(7)** manual page
- Documentation in the **/usr/share/doc/kernel-doc-<kernel_version>/Documentation/cgroups-v1/** directory

26.11. MONITORING RESOURCE CONSUMPTION

The following procedure describes how to view a list of currently running control groups (**cgroups**) and their resource consumption in real-time.

Procedure

1. To see a dynamic account of currently running **cgroups**, execute the **# systemd-cgtop** command:

```
# systemd-cgtop
Control Group                            Tasks   %CPU   Memory  Input/s Output/s
/                                      607   29.8     1.5G        -        -
/system.slice                            125      -   428.7M        -        -
/system.slice/ModemManager.service          3      -     8.6M        -        -
/system.slice/NetworkManager.service        3      -    12.8M        -        -
/system.slice/accounts-daemon.service       3      -     1.8M        -        -
/system.slice/boot.mount                    -      -    48.0K        -        -
/system.slice/chronyd.service               1      -     2.0M        -        -
/system.slice/cockpit.socket                -      -     1.3M        -        -
/system.slice/colord.service                3      -     3.5M        -        -
/system.slice/crond.service                 1      -     1.8M        -        -
/system.slice/cups.service                  1      -     3.1M        -        -
/system.slice/dev-hugepages.mount           -      -   244.0K        -        -
/system.slice/dev-mapper-rhel\x2dswap.swap  -      -   912.0K        -        -
/system.slice/dev-mqueue.mount              -      -    48.0K        -        -
/system.slice/example.service               2      -     2.0M        -        -
/system.slice/firewalld.service             2      -    28.8M        -        -
...
```

The example output displays currently running **cgroups** ordered by their resource usage (CPU, memory, disk I/O load). The list refreshes every 1 second by default. Therefore, it offers a dynamic insight into the actual resource usage of each control group.

Additional resources

- **systemd-cgtop(1)** manual page

CHAPTER 27. CONFIGURING RESOURCE MANAGEMENT USING CGROUPS VERSION 2 WITH SYSTEMD

The core of systemd is service management and supervision. systemd ensures that the right services start at the right time and in the correct order during the boot process. When the services are running, they have to run smoothly to use the underlying hardware platform optimally. Therefore, systemd also provides capabilities to define resource management policies and to tune various options, which can improve the performance of the service.

27.1. PREREQUISITES

- Basic knowledge of the Linux cgroup subsystem.

27.2. INTRODUCTION TO RESOURCE DISTRIBUTION MODELS

For resource management, systemd uses the cgroups v2 interface.

Note that RHEL 8 uses cgroups v1 by default. Therefore, you must enable cgroups v2 so that systemd can use the cgroups v2 interface for resource management. For more information on how to enable cgroups v2, see Setting CPU limits to applications using cgroups-v2 .

To modify the distribution of system resources, you can apply one or more of the following resource distribution models:

Weights

The resource is distributed by adding up the weights of all sub-groups and giving each sub-group the fraction matching its ratio against the sum.
For example, if you have 10 cgroups, each with Weight of value 100, the sum is 1000 and each cgroup receives one tenth of the resource.

Weight is usually used to distribute stateless resources. The *CPUWeight=* option is an implementation of this resource distribution model.

Limits

A cgroup can consume up to the configured amount of the resource, but you can also overcommit resources. Therefore, the sum of sub-group limits can exceed the limit of the parent cgroup.
The *MemoryMax=* option is an implementation of this resource distribution model.

Protections

A protected amount of a resource can be set up for a cgroup. If the resource usage is below the protection boundary, the kernel will try not to penalize this cgroup in favor of other cgroups that compete for the same resource. An overcommit is also allowed.
The *MemoryLow=* option is an implementation of this resource distribution model.

Allocations

Exclusive allocations of an absolute amount of a finite resource. An overcommit is not allowed. An example of this resource type in Linux is the real-time budget.

Additional resources

- Managing CPU with systemd

- Allocating memory resources using systemd
- Configuring I/O bandwidth using systemd

27.3. ALLOCATING CPU RESOURCES USING SYSTEMD

On a system managed by systemd, each system service is started in its cgroup. By enabling the support for the CPU cgroup controller, the system uses the service-aware distribution of CPU resources instead of the per-process distribution. In the service-aware distribution, each service receives approximately the same amount of CPU time relative to all other services running on the system, regardless of the number of processes that comprise the service.

If a specific service requires more CPU resources, you can grant them by changing the CPU time allocation policy for the service.

Procedure

To set a CPU time allocation policy option when using systemd:

1. Check the assigned values of the CPU time allocation policy option in the service of your choice:

```
$ systemctl show --property <CPU time allocation policy option> <service name>
```

2. Set the required value of the CPU time allocation policy option as a root:

```
# systemctl set-property <service name> <CPU time allocation policy option>=<value>
```

> **NOTE**
>
> The cgroup properties are applied immediately after they are set. Therefore, the service does not need to be restarted.

The cgroup properties are applied immediately after they are set. Therefore, the service does not need to be restarted.

Verification steps

- To verify whether you successfully changed the required value of the CPU time allocation policy option for your service, run the following command:

```
$ systemctl show --property <CPU time allocation policy option> <service name>
```

Additional resources

- CPU time allocation policy options for systemd
- Introduction to resource distribution models

27.4. CPU TIME ALLOCATION POLICY OPTIONS FOR SYSTEMD

The most frequently used CPU time allocation policy options include:

CPUWeight=

Assigns **higher priority** to a particular service over all other services. You can select a value from the interval 1 – 10,000. The default value is 100.
For example, to give **httpd.service** twice as much CPU as to all other services, set the value to **CPUWeight=200**.

Note that **CPUWeight=** is applied only in cases when the operating system is overloaded.

CPUQuota=

Assigns the **absolute CPU time quota** to a service. The value of this option specifies the maximum percentage of CPU time that a service will receive relative to the total CPU time available, for example **CPUQuota=30%**.
Note that **CPUQuota=** represents the limit value for particular resource distribution models described in Introduction to resource distribution models .

For more information on **CPUQuota=**, see the **systemd.resource-control(5)** man page.

Additional resources

- Introduction to resource distribution models
- Allocating CPU resources using systemd

27.5. ALLOCATING MEMORY RESOURCES USING SYSTEMD

This section describes how to use any of the memory configuration options (**MemoryMin**, **MemoryLow**, **MemoryHigh**, **MemoryMax**, **MemorySwapMax**) to allocate memory resources using systemd.

Procedure

To set a memory allocation configuration option when using systemd:

1. Check the assigned values of the memory allocation configuration option in the service of your choice:

```
$ systemctl show --property <memory allocation configuration option> <service name>
```

2. Set the required value of the memory allocation configuration option as a root:

```
# systemctl set-property <service name> <memory allocation configuration option>=<value>
```

NOTE

The cgroup properties are applied immediately after they are set. Therefore, the service does not need to be restarted.

Verification steps

- To verify whether you successfully changed the required value of the memory allocation configuration option for your service, run the following command:

```
$ systemctl show --property <memory allocation configuration option> <service name>
```

Additional resources

Additional resources

- Memory allocation configuration options for systemd
- Introduction to resource distribution models

27.6. MEMORY ALLOCATION CONFIGURATION OPTIONS FOR SYSTEMD

You can use the following options when using systemd to configure system memory allocation

MemoryMin

Hard memory protection. If the memory usage is below the limit, the cgroup memory will not be reclaimed.

MemoryLow

Soft memory protection. If the memory usage is below the limit, the cgroup memory can be reclaimed only if no memory is reclaimed from unprotected cgroups.

MemoryHigh

Memory throttle limit. If the memory usage goes above the limit, the processes in the cgroup are throttled and put under a heavy reclaim pressure.

MemoryMax

Absolute limit for the memory usage. You can use the kilo (K), mega (M), giga (G), tera (T) suffixes, for example **MemoryMax=1G**.

MemorySwapMax

Hard limit on the swap usage.

NOTE

When you exhaust your memory limit, the Out-of-memory (OOM) killer will stop the running service. To prevent this, lower the **OOMScoreAdjust=** value to increase the memory tolerance.

Additional resources

- Allocating memory resources using systemd
- Introduction to resource distribution models

27.7. CONFIGURING I/O BANDWIDTH USING SYSTEMD

To improve the performance of a specific service in RHEL 8, you can allocate I/O bandwidth resources to that service using systemd.

To do so, you can use the following I/O configuration options:

- IOWeight
- IODeviceWeight
- IOReadBandwidthMax
- IOWriteBandwidthMax

- IOReadIOPSMax
- IOWriteIOPSMax

Procedure

To set a **I/O bandwidth configuration** option using systemd:

1. Check the assigned values of the I/O bandwidth configuration option in the service of your choice:

   ```
   $ systemctl show --property <I/O bandwidth configuration option> <service name>
   ```

2. Set the required value of the I/O bandwidth configuration option as a root:

   ```
   # systemctl set-property <service name> <I/O bandwidth configuration option>=<value>
   ```

The cgroup properties are applied immediately after they are set. Therefore, the service does not need to be restarted.

Verification steps

- To verify whether you successfully changed the required value of the I/O bandwidth configuration option for your service, run the following command:

  ```
  $ systemctl show --property <I/O bandwidth configuration option> <service name>
  ```

Additional resources

- I/O bandwidth configuration options for systemd
- Introduction to resource distribution models

27.8. I/O BANDWIDTH CONFIGURATION OPTIONS FOR SYSTEMD

To manage the block layer I/O policies with systemd, the following configuration options are available:

IOWeight

Sets the default I/O weight. The weight value is used as a basis for the calculation of how much of the real I/O bandwidth the service receives in relation to the other services.

IODeviceWeight

Sets the I/O weight for a specific block device.
For example, **IODeviceWeight=/dev/disk/by-id/dm-name-rhel-root 200**.

IOReadBandwidthMax, IOWriteBandwidthMax

Sets the absolute bandwidth per device or a mount point.
For example, **IOWriteBandwith=/var/log 5M**.

NOTE

Systemd handles the file-system-to-device translation automatically.

IOReadIOPSMax, IOWriteIOPSMax

A similar option to the previous one: sets the absolute bandwidth in Input/Output Operations Per Second (IOPS).

NOTE

Weight-based options are supported only if the block device is using the CFQ I/O scheduler. No option is supported if the device uses the Multi-Queue Block I/O queuing mechanism.

Additional resources

- Configuring I/O bandwidth using systemd
- Introduction to resource distribution models

27.9. CONFIGURING CPUSET CONTROLLER USING SYSTEMD

The **systemd** resource management API allows the user to configure limits on a set of CPUs and NUMA nodes that a service can use. This limit restricts access to system resources utilized by the processes. The requested configuration is written in **cpuset.cpus** and **cpuset.mems**. However, the requested configuration may not be used, as the parent **cgroup** limits either **cpus** or **mems**. To access the current configuration, the **cpuset.cpus.effective** and **cpuset.mems.effective** files are exported to the users.

Procedure

- To set **AllowedCPUs**:

```
# systemctl set-property service_name.service AllowedCPUs=value
```

  For example:

```
# systemctl set-property service_name.service AllowedCPUs=0-5
```

- To set **AllowedMemoryNodes**:

```
# systemctl set-property service_name.service AllowedMemoryNodes=value
```

  For example:

```
# systemctl set-property service_name.service AllowedMemoryNodes=0
```

CHAPTER 28. CONFIGURING CPU AFFINITY AND NUMA POLICIES USING SYSTEMD

The CPU management, memory management, and I/O bandwidth options deal with partitioning available resources.

28.1. CONFIGURING CPU AFFINITY USING SYSTEMD

CPU affinity settings help you restrict the access of a particular process to some CPUs. Effectively, the CPU scheduler never schedules the process to run on the CPU that is not in the affinity mask of the process.

The default CPU affinity mask applies to all services managed by systemd.

To configure CPU affinity mask for a particular systemd service, systemd provides **CPUAffinity=** both as a unit file option and a manager configuration option in the **/etc/systemd/system.conf** file.

The **CPUAffinity= unit file option** sets a list of CPUs or CPU ranges that are merged and used as the affinity mask. The **CPUAffinity option** in the **/etc/systemd/system.conf** file defines an affinity mask for the process identification number (PID) 1 and all processes forked off of PID1. You can then override the **CPUAffinity** on a per-service basis.

NOTE

After configuring CPU affinity mask for a particular systemd service, you must restart the system to apply the changes.

Procedure

To set CPU affinity mask for a particualr systemd service using the **CPUAffinity unit file** option:

1. Check the values of the **CPUAffinity** unit file option in the service of your choice:

```
$ systemctl show --property <CPU affinity configuration option> <service name>
```

2. As a root, set the required value of the **CPUAffinity** unit file option for the CPU ranges used as the affinity mask:

```
# systemctl set-property <service name> CPUAffinity=<value>
```

3. Restart the service to apply the changes.

```
# systemctl restart <service name>
```

To set CPU affinity mask for a particular systemd service using the **manager configuration** option:

1. Edit the **/etc/systemd/system.conf** file:

```
# vi /etc/systemd/system.conf
```

2. Search for the **CPUAffinity=** option and set the CPU numbers

3. Save the edited file and restart the server to apply the changes.

28.2. CONFIGURING NUMA POLICIES USING SYSTEMD

Non-uniform memory access (NUMA) is a computer memory subsystem design, in which the memory access time depends on the physical memory location relative to the processor.

Memory close to the CPU has lower latency (local memory) than memory that is local for a different CPU (foreign memory) or is shared between a set of CPUs.

In terms of the Linux kernel, NUMA policy governs where (for example, on which NUMA nodes) the kernel allocates physical memory pages for the process.

systemd provides unit file options **NUMAPolicy** and **NUMAMask** to control memory allocation policies for services.

Procedure

To set the NUMA memory policy through the **NUMAPolicy unit file** option:

1. Check the values of the **NUMAPolicy** unit file option in the service of your choice:

   ```
   $ systemctl show --property <NUMA policy configuration option> <service name>
   ```

2. As a root, set the required policy type of the **NUMAPolicy** unit file option:

   ```
   # systemctl set-property <service name> NUMAPolicy=<value>
   ```

3. Restart the service to apply the changes.

   ```
   # systemctl restart <service name>
   ```

To set a global **NUMAPolicy** setting through the **manager configuration** option:

1. Search in the **/etc/systemd/system.conf** file for the **NUMAPolicy** option.
2. Edit the policy type and save the file.
3. Reload the **systemd** configuration:

   ```
   # systemd daemon-reload
   ```

4. Reboot the server.

IMPORTANT

When you configure a strict NUMA policy, for example **bind**, make sure that you also appropriately set the **CPUAffinity=** unit file option.

Additional resources

- NUMA policy configuration options for systemd
- **systemd.resource-control(5)**, **systemd.exec(5)**, **set_mempolicy(2)** manual pages.

28.3. NUMA POLICY CONFIGURATION OPTIONS FOR SYSTEMD

Systemd provides the following options to configure the NUMA policy:

NUMAPolicy

Controls the NUMA memory policy of the executed processes. The following policy types are possible:

- default
- preferred
- bind
- interleave
- local

NUMAMask

Controls the NUMA node list which is associated with the selected NUMA policy.
Note that the **NUMAMask** option is not required to be specified for the following policies:

- default
- local

For the preferred policy, the list specifies only a single NUMA node.

Additional resources

- **systemd.resource-control(5)**, **systemd.exec(5)**, and **set_mempolicy(2)** manual pages
- Configuring NUMA using systemd

CHAPTER 29. ANALYZING SYSTEM PERFORMANCE WITH BPF COMPILER COLLECTION

As a system administrator, you can use the BPF Compiler Collection (BCC) library to create tools for analyzing the performance of your Linux operating system and gathering information, which could be difficult to obtain through other interfaces.

29.1. AN INTRODUCTION TO BCC

BPF Compiler Collection (BCC) is a library, which facilitates the creation of the extended Berkeley Packet Filter (eBPF) programs. The main utility of eBPF programs is analyzing OS performance and network performance without experiencing overhead or security issues.

BCC removes the need for users to know deep technical details of eBPF, and provides many out-of-the-box starting points, such as the **bcc-tools** package with pre-created eBPF programs.

NOTE

The eBPF programs are triggered on events, such as disk I/O, TCP connections, and process creations. It is unlikely that the programs should cause the kernel to crash, loop or become unresponsive because they run in a safe virtual machine in the kernel.

29.2. INSTALLING THE BCC-TOOLS PACKAGE

This section describes how to install the **bcc-tools** package, which also installs the BPF Compiler Collection (BCC) library as a dependency.

Procedure

1. Install **bcc-tools**:

   ```
   # yum install bcc-tools
   ```

   The BCC tools are installed in the **/usr/share/bcc/tools/** directory.

2. Optionally, inspect the tools:

   ```
   # ll /usr/share/bcc/tools/
   ...
   -rwxr-xr-x. 1 root root  4198 Dec 14 17:53 dcsnoop
   -rwxr-xr-x. 1 root root  3931 Dec 14 17:53 dcstat
   -rwxr-xr-x. 1 root root 20040 Dec 14 17:53 deadlock_detector
   -rw-r--r--. 1 root root  7105 Dec 14 17:53 deadlock_detector.c
   drwxr-xr-x. 3 root root  8192 Mar 11 10:28 doc
   -rwxr-xr-x. 1 root root  7588 Dec 14 17:53 execsnoop
   -rwxr-xr-x. 1 root root  6373 Dec 14 17:53 ext4dist
   -rwxr-xr-x. 1 root root 10401 Dec 14 17:53 ext4slower
   ...
   ```

   The **doc** directory in the listing above contains documentation for each tool.

29.3. USING SELECTED BCC-TOOLS FOR PERFORMANCE ANALYSES

This section describes how to use certain pre-created programs from the BPF Compiler Collection (BCC) library to efficiently and securely analyze the system performance on the per-event basis. The set of pre-created programs in the BCC library can serve as examples for creation of additional programs.

Prerequisites

- Installed bcc-tools package
- Root permissions

Using execsnoop to examine the system processes

1. Execute the **execsnoop** program in one terminal:

   ```
   # /usr/share/bcc/tools/execsnoop
   ```

2. In another terminal execute for example:

   ```
   $ ls /usr/share/bcc/tools/doc/
   ```

   The above creates a short-lived process of the **ls** command.

3. The terminal running **execsnoop** shows the output similar to the following:

   ```
   PCOMM PID    PPID   RET ARGS
   ls    8382   8287     0 /usr/bin/ls --color=auto /usr/share/bcc/tools/doc/
   ...
   ```

   The **execsnoop** program prints a line of output for each new process, which consumes system resources. It even detects processes of programs that run very shortly, such as **ls**, and most monitoring tools would not register them.

   The **execsnoop** output displays the following fields:

   - PCOMM - The parent process name. (**ls**)
   - PID - The process ID. (**8382**)
   - PPID - The parent process ID. (**8287**)
   - RET - The return value of the **exec()** system call (**0**), which loads program code into new processes.
   - ARGS - The location of the started program with arguments.

To see more details, examples, and options for **execsnoop**, refer to the **/usr/share/bcc/tools/doc/execsnoop_example.txt** file.

For more information about **exec()**, see **exec(3)** manual pages.

Using opensnoop to track what files a command opens

1. Execute the **opensnoop** program in one terminal:

   ```
   # /usr/share/bcc/tools/opensnoop -n uname
   ```

The above prints output for files, which are opened only by the process of the **uname** command.

2. In another terminal execute:

   ```
   $ uname
   ```

   The command above opens certain files, which are captured in the next step.

3. The terminal running **opensnoop** shows the output similar to the following:

   ```
   PID    COMM  FD ERR PATH
   8596   uname  3  0   /etc/ld.so.cache
   8596   uname  3  0   /lib64/libc.so.6
   8596   uname  3  0   /usr/lib/locale/locale-archive
   ...
   ```

   The **opensnoop** program watches the **open()** system call across the whole system, and prints a line of output for each file that **uname** tried to open along the way.

   The **opensnoop** output displays the following fields:

   - **PID** - The process ID. (**8596**)
   - **COMM** - The process name. (**uname**)
   - **FD** - The file descriptor - a value that **open()** returns to refer to the open file. (**3**)
   - **ERR** - Any errors.
   - **PATH** - The location of files that **open()** tried to open.
     If a command tries to read a non-existent file, then the **FD** column returns **-1** and the **ERR** column prints a value corresponding to the relevant error. As a result, **opensnoop** can help you identify an application that does not behave properly.

To see more details, examples, and options for **opensnoop**, refer to the **/usr/share/bcc/tools/doc/opensnoop_example.txt** file.

For more information about **open()**, see **open(2)** manual pages.

Using biotop to examine the I/O operations on the disk

1. Execute the **biotop** program in one terminal:

   ```
   # /usr/share/bcc/tools/biotop 30
   ```

   The command enables you to monitor the top processes, which perform I/O operations on the disk. The argument ensures that the command will produce a 30 second summary.

   NOTE

   When no argument provided, the output screen by default refreshes every 1 second.

2. In another terminal execute for example :

```
# dd if=/dev/vda of=/dev/zero
```

The command above reads the content from the local hard disk device and writes the output to the /**dev**/**zero** file. This step generates certain I/O traffic to illustrate **biotop**.

3. The terminal running **biotop** shows the output similar to the following:

```
PID    COMM             D MAJ MIN DISK       I/O  Kbytes     AVGms
9568   dd               R 252 0   vda      16294 14440636.0  3.69
48     kswapd0          W 252 0   vda       1763 120696.0    1.65
7571   gnome-shell      R 252 0   vda        834 83612.0     0.33
1891   gnome-shell      R 252 0   vda       1379 19792.0     0.15
7515   Xorg             R 252 0   vda        280  9940.0     0.28
7579   llvmpipe-1       R 252 0   vda        228  6928.0     0.19
9515   gnome-control-c  R 252 0   vda         62  6444.0     0.43
8112   gnome-terminal-  R 252 0   vda         67  2572.0     1.54
7807   gnome-software   R 252 0   vda         31  2336.0     0.73
9578   awk              R 252 0   vda         17  2228.0     0.66
7578   llvmpipe-0       R 252 0   vda        156  2204.0     0.07
9581   pgrep            R 252 0   vda         58  1748.0     0.42
7531   InputThread      R 252 0   vda         30  1200.0     0.48
7504   gdbus            R 252 0   vda          3  1164.0     0.30
1983   llvmpipe-1       R 252 0   vda         39   724.0     0.08
1982   llvmpipe-0       R 252 0   vda         36   652.0     0.06
...
```

The **biotop** output displays the following fields:

- **PID** - The process ID. (**9568**)
- **COMM** - The process name. (**dd**)
- **DISK** - The disk performing the read operations. (**vda**)
- **I/O** - The number of read operations performed. (16294)
- **Kbytes** - The amount of Kbytes reached by the read operations. (14,440,636)
- **AVGms** - The average I/O time of read operations. (3.69)

To see more details, examples, and options for **biotop**, refer to the **/usr/share/bcc/tools/doc/biotop_example.txt** file.

For more information about **dd**, see **dd(1)** manual pages.

Using xfsslower to expose unexpectedly slow file system operations

1. Execute the **xfsslower** program in one terminal:

```
# /usr/share/bcc/tools/xfsslower 1
```

The command above measures the time the XFS file system spends in performing read, write, open or sync (**fsync**) operations. The **1** argument ensures that the program shows only the operations that are slower than 1 ms.

NOTE

When no arguments provided, **xfsslower** by default displays operations slower than 10 ms.

2. In another terminal execute, for example, the following:

```
$ vim text
```

   The command above creates a text file in the **vim** editor to initiate certain interaction with the XFS file system.

3. The terminal running **xfsslower** shows something similar upon saving the file from the previous step:

```
TIME     COMM           PID    T BYTES   OFF_KB   LAT(ms) FILENAME
13:07:14 b'bash'        4754   R 256     0           7.11 b'vim'
13:07:14 b'vim'         4754   R 832     0           4.03 b'libgpm.so.2.1.0'
13:07:14 b'vim'         4754   R 32      20          1.04 b'libgpm.so.2.1.0'
13:07:14 b'vim'         4754   R 1982    0           2.30 b'vimrc'
13:07:14 b'vim'         4754   R 1393    0           2.52 b'getscriptPlugin.vim'
13:07:45 b'vim'         4754   S 0       0           6.71 b'text'
13:07:45 b'pool'        2588   R 16      0           5.58 b'text'
...
```

   Each line above represents an operation in the file system, which took more time than a certain threshold. **xfsslower** is good at exposing possible file system problems, which can take form of unexpectedly slow operations.

   The **xfsslower** output displays the following fields:

   - COMM - The process name. (**b'bash'**)
   - T - The operation type. (**R**)
     - Read
     - Write
     - Sync
   - OFF_KB - The file offset in KB. (0)
   - FILENAME - The file being read, written, or synced.

To see more details, examples, and options for **xfsslower**, refer to the **/usr/share/bcc/tools/doc/xfsslower_example.txt** file.

For more information about **fsync**, see **fsync(2)** manual pages.

CHAPTER 30. ENHANCING SECURITY WITH THE KERNEL INTEGRITY SUBSYSTEM

You can increase the protection of your system by utilizing components of the kernel integrity subsystem. The following sections introduce the relevant components and provide guidance on their configuration.

NOTE

You can use the features with cryptographic signatures only for Red Hat products because the kernel keyring system includes only the certificates for Red Hat signature keys. Using other hash features results in incomplete tamperproofing.

30.1. THE KERNEL INTEGRITY SUBSYSTEM

The integrity subsystem is a part of the kernel that is responsible for maintaining the overall system data integrity. This subsystem helps to keep the state of a certain system the same from the time it was built and thus prevents undesired modification on specific system files.

The kernel integrity subsystem consists of two major components:

Integrity Measurement Architecture (IMA)

- Measures file content whenever it is executed or opened by cryptographically hashing or signing with cryptographic keys. The keys are stored in the kernel keyring subsystem.
- Places the measured values within the kernel’s memory space thereby it prevents any modification from the users of the system.
- Allows local and remote parties to verify the measured values.
- Provides local validation of the current content of files against the values previously stored in the measurement list within the kernel memory. This extension forbids to conduct any operation on a specific file in case the current and the previous measures do not match.

Extended Verification Module (EVM)

- Protects extended attributes of files (also known as *xattr*) that are related to the system security, like IMA measurements and SELinux attributes. The component cryptographically hashes their corresponding values or signs them with cryptographic keys. The keys are stored in the kernel keyring subsystem.

The kernel integrity subsystem can harness the Trusted Platform Module (TPM) to harden the system security even more. TPM is a specification by the Trusted Computing Group (TCG) for important cryptographic functions. TPMs are usually built as dedicated hardware that is attached to the platform’s motherboard and prevents software-based attacks by providing cryptographic functions from a protected and tamper-proof area of the hardware chip. Some of the TPM features are:

- Random-number generator
- Generator and secure storage for cryptographic keys
- Hashing generator

- Remote attestation

Additional resources

- *Integrity Measurement Architecture (IMA)*
- *Trusted Computing Group resources*
- Security hardening
- Basic and advanced configuration of Security-Enhanced Linux (SELinux)

30.2. INTEGRITY MEASUREMENT ARCHITECTURE

Integrity Measurement Architecture (IMA) is a component of the kernel integrity subsystem. IMA aims to maintain the contents of local files. Specifically, IMA measures, stores, and appraises file hashes before they are accessed, which prevents the reading and execution of unreliable data. Thereby, usage of IMA with digital signatures enhances the security of the system and prevents from measurement data forgery. The scenario assumes the attacker does not possess the original signature key.

Additional resources

- *Integrity Measurement Architecture (IMA)*
- *Security hardening*
- Basic and advanced configuration of Security-Enhanced Linux (SELinux)

30.3. EXTENDED VERIFICATION MODULE

Extended Verification Module (EVM) is a component of the kernel integrity subsystem that monitors changes in file extended attributes (*xattr*). Many security-oriented technologies store sensitive file information, such as content hashes and signatures, in extended attributes of files. EVM goes one step further and hashes or signs those extended attributes. The resulting data is validated every time the extended attribute is used. For example, when IMA appraises the file.

Additional resources

- *Integrity Measurement Architecture (IMA)*
- *Security hardening*
- Basic and advanced configuration of Security-Enhanced Linux (SELinux)

30.4. TRUSTED AND ENCRYPTED KEYS

The following section introduces trusted and encrypted keys as an important part of enhancing system security.

Trusted and *encrypted keys* are variable-length symmetric keys generated by the kernel that use the kernel keyring service. The integrity of the keys can be verified, which means that they can be used, for example, by the extended verification module (EVM) to verify and confirm the integrity of a running system. User-level programs can only access the keys in the form of encrypted *blobs*.

Trusted keys need a hardware component, the Trusted Platform Module (TPM) chip, that is used to both create and encrypt (seal) the keys. The TPM seals the keys using a key called the primary *storage root key*.

NOTE

Red Hat Enterprise Linux 8 supports both TPM 1.2 and TPM 2.0. For more information, see Is Trusted Platform Module (TPM) supported by Red Hat? .

NOTE

You can verify that a TPM 2.0 chip has been enabled:

```
$ cat /sys/class/tpm/tpm0/tpm_version_major
2
```

You can also enable a TPM 2.0 chip through a setting in the machine firmware and manage the TPM 2.0 device.

In addition to that, the user can seal the trusted keys with a specific set of the TPM's *platform configuration register* (PCR) values. PCR contains a set of integrity-management values that reflect the firmware, boot loader, and operating system. This means that PCR-sealed keys can only be decrypted by the TPM on the same system on which they were encrypted. However, once a PCR-sealed trusted key is loaded (added to a keyring), and thus its associated PCR values are verified, it can be updated with new (or future) PCR values, so that a new kernel, for example, can be booted. A single key can also be saved as multiple blobs, each with different PCR values.

Encrypted keys do not require a TPM, as they use the kernel Advanced Encryption Standard (AES), which makes them faster than trusted keys. Encrypted keys are created using kernel-generated random numbers and encrypted by a *master key* when they are exported into user-space blobs.

The master key is either a trusted key or a user key. If the master key is not trusted, the encrypted key is only as secure as the user key used to encrypt it.

30.5. WORKING WITH TRUSTED KEYS

You can create, export, load or update trusted keys with the **keyctl** utility to improve the system security.

Prerequisites

- For the 64-bit ARM architecture and IBM Z, the **trusted** kernel module needs to be loaded. See Managing kernel modules.
- Trusted Platform Module (TPM) needs to be enabled and active. See The kernel integrity subsystem and Trusted and encrypted keys.

NOTE

Red Hat Enterprise Linux 8 supports both TPM 1.2 and TPM 2.0. If you use TPM 1.2, skip step 1 and use the following syntax in step 2: *# keyctl add trusted <NAME> "new <KEY_LENGTH>" <KEYRING>*.

```
# keyctl add trusted kmk "new 32" @u
```

Procedure

1. Create the 2048-bit RSA with a SHA-256 primary storage key with a persistent handle of, for example, *81000001*.

   a. Using the **tss2** package:

   ```
   # TPM_DEVICE=/dev/tpm0 tsscreateprimary -hi o -st
   Handle 80000000
   # TPM_DEVICE=/dev/tpm0 tssevictcontrol -hi o -ho 80000000 -hp 81000001
   ```

   b. Using the **tpm2-tools** package:

   ```
   # tpm2_createprimary --key-algorithm=rsa2048 --key-context=key.ctxt
   name-alg:
     value: sha256
     raw: 0xb
   …
   sym-keybits: 128
   rsa: xxxxxx…

   # tpm2_evictcontrol -c key.ctxt 0x81000001
   persistentHandle: 0x81000001
   action: persisted
   ```

2. Create a trusted key using a TPM 2.0 with the syntax of *keyctl add trusted <NAME> "new <KEY_LENGTH> keyhandle=<PERSISTENT-HANDLE> [options]" <KEYRING>*. In this example, the persistent handle is *81000001*.

   ```
   # keyctl add trusted kmk "new 32 keyhandle=0x81000001" @u
   642500861
   ```

   The command creates a trusted key called **kmk** with the length of **32** bytes (256 bits) and places it in the user keyring (**@u**). The keys may have a length of 32 to 128 bytes (256 to 1024 bits).

3. List the current structure of the kernel keyrings.

   ```
   # keyctl show
   Session Keyring
          -3 --alswrv    500   500  keyring: ses 97833714 --alswrv 500 -1 \ keyring: uid.1000
   642500861 --alswrv 500 500 \ trusted: kmk
   ```

4. Export the key to a user-space blob.

   ```
   # keyctl pipe 642500861 > kmk.blob
   ```

The command uses the **pipe** subcommand and the serial number of **kmk**.

5. Load the trusted key from the user-space blob, use the **add** subcommand with the blob as an argument.

```
# keyctl add trusted kmk "load `cat kmk.blob`" @u
268728824
```

6. Create secure encrypted keys based on the TPM-sealed trusted key based on the syntax *keyctl add encrypted <NAME> "new [FORMAT] <KEY_TYPE>:<PRIMARY_KEY_NAME> <KEY_LENGTH>" <KEYRING>*.

```
# keyctl add encrypted encr-key "new trusted:kmk 32" @u
159771175
```

The command uses the TPM-sealed trusted key (**kmk**), produced in the previous step, as a *primary key* for generating encrypted keys.

Additional resources

- the **keyctl(1)** manual page
- Trusted and encrypted keys
- Kernel Key Retention Service
- The kernel integrity subsystem

30.6. WORKING WITH ENCRYPTED KEYS

The following section describes managing encrypted keys to improve the system security on systems where a Trusted Platform Module (TPM) is not available.

Prerequisites

- For the 64-bit ARM architecture and IBM Z, the **encrypted-keys** kernel module needs to be loaded. For more information on how to load kernel modules, see Managing kernel modules.

Procedure

1. Use a random sequence of numbers to generate a user key:

```
# keyctl add user kmk-user "$(dd if=/dev/urandom bs=1 count=32 2>/dev/null)" @u
427069434
```

The command generates a user key called **kmk-user** which acts as a *primary key* and is used to seal the actual encrypted keys.

2. Generate an encrypted key using the primary key from the previous step:

```
# keyctl add encrypted encr-key "new user:kmk-user 32" @u
1012412758
```

3. Optionally, list all keys in the specified user keyring:

```
# keyctl list @u
2 keys in keyring:
427069434: --alswrv  1000  1000 user: kmk-user
1012412758: --alswrv  1000  1000 encrypted: encr-key
```

IMPORTANT

Keep in mind that encrypted keys that are not sealed by a trusted primary key are only as secure as the user primary key (random-number key) that was used to encrypt them. Therefore, the primary user key should be loaded as securely as possible and preferably early during the boot process.

Additional resources

- **keyctl(1)** manual page
- Kernel Key Retention Service

30.7. ENABLING INTEGRITY MEASUREMENT ARCHITECTURE AND EXTENDED VERIFICATION MODULE

Integrity measurement architecture (IMA) and extended verification module (EVM) belong to the kernel integrity subsystem and enhance the system security in various ways. The following section describes how to enable and configure IMA and EVM to improve the security of the operating system.

IMPORTANT

This procedure works only when using certificates and keys issued by Red Hat.

Prerequisites

- Verify that the **securityfs** filesystem is mounted on the **/sys/kernel/security/** directory and the **/sys/kernel/security/integrity/ima/** directory exists.

```
# mount
…
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
…
```

- Verify that the **systemd** service manager is already patched to support IMA and EVM on boot time:

```
# dmesg | grep -i -e EVM -e IMA
[    0.000000] Command line: BOOT_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-167.el8.x86_64
root=/dev/mapper/rhel-root ro crashkernel=auto resume=/dev/mapper/rhel-swap
rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet
[    0.000000] kvm-clock: cpu 0, msr 23601001, primary cpu clock
[    0.000000] Using crashkernel=auto, the size chosen is a best effort estimation.
[    0.000000] Kernel command line: BOOT_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-
167.el8.x86_64 root=/dev/mapper/rhel-root ro crashkernel=auto resume=/dev/mapper/rhel-
swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet
[    0.911527] ima: No TPM chip found, activating TPM-bypass!
[    0.911538] ima: Allocated hash algorithm: sha1
[    0.911580] evm: Initialising EVM extended attributes:
[    0.911581] evm: security.selinux
[    0.911581] evm: security.ima
[    0.911582] evm: security.capability
[    0.911582] evm: HMAC attrs: 0x1
[    1.715151] systemd[1]: systemd 239 running in system mode. (+PAM +AUDIT +SELINUX
+IMA -APPARMOR +SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP +GCRYPT
+GNUTLS +ACL +XZ +LZ4 +SECCOMP +BLKID +ELFUTILS +KMOD +IDN2 -IDN +PCRE2
default-hierarchy=legacy)
[    3.824198] fbcon: qxldrmfb (fb0) is primary device
[    4.673457] PM: Image not found (code -22)
[    6.549966] systemd[1]: systemd 239 running in system mode. (+PAM +AUDIT +SELINUX
+IMA -APPARMOR +SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP +GCRYPT
+GNUTLS +ACL +XZ +LZ4 +SECCOMP +BLKID +ELFUTILS +KMOD +IDN2 -IDN +PCRE2
default-hierarchy=legacy)
```

Procedure

1. Add the following kernel command line parameters:

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --args="ima_policy=appraise_tcb
ima_appraise=fix evm=fix"
```

   The command enables IMA and EVM in the *fix* mode for the current boot entry and allows users to gather and update the IMA measurements.

   The **ima_policy=appraise_tcb** kernel command line parameter ensures that the kernel uses the default Trusted Computing Base (TCB) measurement policy and the appraisal step. The appraisal part forbids access to files, whose prior and current measures do not match.

2. Reboot to make the changes come into effect.

3. Optionally, verify that the parameters have been added to the kernel command line:

```
# cat /proc/cmdline
BOOT_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-167.el8.x86_64 root=/dev/mapper/rhel-root ro
crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap
rhgb quiet ima_policy=appraise_tcb ima_appraise=fix evm=fix
```

4. Create a kernel master key to protect the EVM key:

```
# keyctl add user kmk "$(dd if=/dev/urandom bs=1 count=32 2> /dev/null)" @u
748544121
```

   The kernel master key (**kmk**) is kept entirely in the kernel space memory. The 32-byte long value of the kernel master key **kmk** is generated from random bytes from the **/dev/urandom** file and placed in the user (**@u**) keyring. The key serial number is on the second line of the previous output.

5. Create an encrypted EVM key based on the **kmk** key:

```
# keyctl add encrypted evm-key "new user:kmk 64" @u
641780271
```

The command uses **kmk** to generate and encrypt a 64-byte long user key (named **evm-key**) and places it in the user (**@u**) keyring. The key serial number is on the second line of the previous output.

IMPORTANT

It is necessary to name the user key as **evm-key** because that is the name the EVM subsystem is expecting and is working with.

6. Create a directory for exported keys.

```
# mkdir -p /etc/keys/
```

7. Search for the **kmk** key and export its value into a file:

```
# keyctl pipe $(keyctl search @u user kmk) > /etc/keys/kmk
```

The command places the unencrypted value of the kernel master key (**kmk**) into a file of previously defined location (**/etc/keys/**).

8. Search for the **evm-key** user key and export its value into a file:

```
# keyctl pipe $(keyctl search @u encrypted evm-key) > /etc/keys/evm-key
```

The command places the encrypted value of the user **evm-key** key into a file of arbitrary location. The **evm-key** has been encrypted by the kernel master key earlier.

9. Optionally, view the newly created keys:

```
# keyctl show
Session Keyring
974575405   --alswrv     0       0      keyring: ses 299489774 --alswrv 0 65534 \ keyring: uid.0
748544121 --alswrv 0 0 \ user: kmk
641780271   --alswrv     0       0          \_ encrypted: evm-key

# ls -l /etc/keys/
total 8
-rw-r—r--. 1 root root 246 Jun 24 12:44 evm-key
-rw-r—r--. 1 root root  32 Jun 24 12:43 kmk
```

You should be able to see a similar output.

10. Activate EVM:

```
# echo 1 > /sys/kernel/security/evm
```

Verification step

- You can verify that EVM has been initialized by entering:

```
# dmesg | tail -1
[…] evm: key initialized
```

NOTE

If the system is rebooted, the keys are removed from the keyring. In such a case, you can import the already exported **kmk** and **evm-key** keys.

Procedure

1. Add the user **kmk** key (already exported to the **/etc/keys/kmk** file in step 7).

```
# keyctl add user kmk "$(cat /etc/keys/kmk)" @u
451342217

# keyctl show
Session Keyring
  695566911 --alswrv      0     0  keyring: ses 58982213 --alswrv 0 65534 \
keyring: uid.0 451342217 --alswrv 0 0 \ user: kmk
```

2. Import the user **evm-key** key (already exported to the **/etc/keys/evm-key** file in step 8).

```
# keyctl add encrypted evm-key "load $(cat /etc/keys/evm-key)" @u
924537557

# keyctl show
Session Keyring
  695566911 --alswrv      0     0  keyring: ses 58982213 --alswrv 0 65534 \
keyring: uid.0 451342217 --alswrv 0 0 \ user: kmk
  924537557 --alswrv      0     0       \_ encrypted: evm-key
```

Additional resources

- Extended verification module
- Integrity measurement architecture
- The kernel integrity subsystem
- Trusted and encrypted keys.

30.8. COLLECTING FILE HASHES WITH INTEGRITY MEASUREMENT ARCHITECTURE

The first level of operation of integrity measurement architecture (IMA) is the *measurement* phase that allows to create file hashes and store them as extended attributes (*xattrs*) of those files. The following section describes how to create and inspect the files' hashes.

Prerequisites

- Enable integrity measurement architecture (IMA) and extended verification module (EVM) as described in Enabling integrity measurement architecture and extended verification module .
- Verify that the **ima-evm-utils**, **attr**, and **keyutils** packages are already installed:

```
# yum install ima-evm-utils attr keyutils
```

```
Updating Subscription Management repositories.
This system is registered to Red Hat Subscription Management, but is not receiving updates.
You can use subscription-manager to assign subscriptions.
Last metadata expiration check: 0:58:22 ago on Fri 14 Feb 2020 09:58:23 AM CET.
Package ima-evm-utils-1.1-5.el8.x86_64 is already installed.
Package attr-2.4.48-3.el8.x86_64 is already installed.
Package keyutils-1.5.10-7.el8.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
```

Procedure

1. Create a test file:

   ```
   # echo <Test_text> > test_file
   ```

   IMA and EVM ensure that the **test_file** example file has assigned hash values that are stored as its extended attributes.

2. Inspect extended attributes of the file:

   ```
   # getfattr -m . -d test_file
   # file: test_file
   security.evm=0sAnDIy4VPA0HArpPO/EqiutnNyBql
   security.ima=0sAQOEDeuUnWzwwKYk+n66h/vby3eD
   security.selinux="unconfined_u:object_r:admin_home_t:s0"
   ```

   The example output shows extended attributes related to SELinux and the IMA and EVM hash values. EVM adds a **security.evm** extended attribute related to the other attributes. You can use the **evmctl** utility on **security.evm** to generate either an RSA based digital signature or a Hash-based Message Authentication Code (HMAC-SHA1). To successfully conduct the previous operation you need a valid trusted or encrypted key, which is stored in the kernel keyring. This configuration prevents offline tampering attacks on the extended attributes.

Additional resources

- Security hardening
- Extended verification module
- Integrity measurement architecture

CHAPTER 31. USING ANSIBLE ROLES TO PERMANENTLY CONFIGURE KERNEL PARAMETERS

As an experienced user with good knowledge of Red Hat Ansible, you can use the **kernel_settings** role to configure kernel parameters on multiple clients at once. This solution:

- Provides a friendly interface with efficient input setting.
- Keeps all intended kernel parameters in one place.

After you run the **kernel_settings** role from the control machine, the kernel parameters are applied to the managed systems immediately and persist across reboots.

IMPORTANT

Note that RHEL System Role delivered over RHEL channels are available to RHEL customers as an RPM package in the default AppStream repository. RHEL System Role are also available as a collection to customers with Ansible subscriptions over Ansible Automation Hub.

31.1. INTRODUCTION TO THE KERNEL SETTINGS ROLE

RHEL System Roles is a set of roles that provide a consistent configuration interface to remotely manage multiple systems.

RHEL System Roles were introduced for automated configurations of the kernel using the **kernel_settings** System Role. The **rhel-system-roles** package contains this system role, and also the reference documentation.

To apply the kernel parameters on one or more systems in an automated fashion, use the **kernel_settings** role with one or more of its role variables of your choice in a playbook. A playbook is a list of one or more plays that are human-readable, and are written in the YAML format.

You can use an inventory file to define a set of systems that you want Ansible to configure according to the playbook.

With the **kernel_settings** role you can configure:

- The kernel parameters using the **kernel_settings_sysctl** role variable
- Various kernel subsystems, hardware devices, and device drivers using the **kernel_settings_sysfs** role variable
- The CPU affinity for the **systemd** service manager and processes it forks using the **kernel_settings_systemd_cpu_affinity** role variable
- The kernel memory subsystem transparent hugepages using the **kernel_settings_transparent_hugepages** and **kernel_settings_transparent_hugepages_defrag** role variables

Additional resources

- **README.md** and **README.html** files in the **/usr/share/doc/rhel-system-roles/kernel_settings/** directory

- Working with playbooks
- How to build your inventory

31.2. APPLYING SELECTED KERNEL PARAMETERS USING THE KERNEL_SETTINGS ROLE

Follow these steps to prepare and apply an Ansible playbook to remotely configure kernel parameters with persisting effect on multiple managed operating systems.

Prerequisites

- You have **root** permissions.
- Entitled by your RHEL subscription, you installed the **ansible-core** and **rhel-system-roles** packages on the control machine.
- An inventory of managed hosts is present on the control machine and Ansible is able to connect to them.

IMPORTANT

RHEL 8.0 - 8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**; connectors such as **docker** and **podman**; and the entire world of plugins and modules. For information on how to obtain and install Ansible Engine, refer to How do I Download and Install Red Hat Ansible Engine? .

RHEL 8.6 and 9.0 has introduced Ansible Core (provided as **ansible-core** RPM), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. The AppStream repository provides **ansible-core**, which has a limited scope of support. You can learn more by reviewing Scope of support for the ansible-core package included in the RHEL 9 AppStream.

Procedure

1. Optionally, review the **inventory** file for illustration purposes:

```
# cat /home/jdoe/<ansible_project_name>/inventory
[testingservers]
pdoe@192.168.122.98
fdoe@192.168.122.226

[db-servers]
db1.example.com
db2.example.com

[webservers]
web1.example.com
web2.example.com
192.0.2.42
```

   The file defines the **[testingservers]** group and other groups. It allows you to run Ansible more effectively against a specific set of systems.

2. Create a configuration file to set defaults and privilege escalation for Ansible operations.

   a. Create a new YAML file and open it in a text editor, for example:

      ```
      # vi /home/jdoe/<ansible_project_name>/ansible.cfg
      ```

   b. Insert the following content into the file:

      ```
      [defaults]
      inventory = ./inventory

      [privilege_escalation]
      become = true
      become_method = sudo
      become_user = root
      become_ask_pass = true
      ```

      The **[defaults]** section specifies a path to the inventory file of managed hosts. The **[privilege_escalation]** section defines that user privileges be shifted to **root** on the specified managed hosts. This is necessary for successful configuration of kernel parameters. When Ansible playbook is run, you will be prompted for user password. The user automatically switches to **root** by means of **sudo** after connecting to a managed host.

3. Create an Ansible playbook that uses the **kernel_settings** role.

   a. Create a new YAML file and open it in a text editor, for example:

      ```
      # vi /home/jdoe/<ansible_project_name>/kernel-roles.yml
      ```

      This file represents a playbook and usually contains an ordered list of tasks, also called *plays*, that are run against specific managed hosts selected from your **inventory** file.

   b. Insert the following content into the file:

      ```
      ---
      -
        hosts: testingservers
        name: "Configure kernel settings"
        roles:
          - rhel-system-roles.kernel_settings
        vars:
          kernel_settings_sysctl:
            - name: fs.file-max
              value: 400000
            - name: kernel.threads-max
              value: 65536
          kernel_settings_sysfs:
            - name: /sys/class/net/lo/mtu
              value: 65000
          kernel_settings_transparent_hugepages: madvise
      ```

      The **name** key is optional. It associates an arbitrary string with the play as a label and identifies what the play is for. The **hosts** key in the play specifies the hosts against which the play is run. The value or values for this key can be provided as individual names of managed hosts or as groups of hosts as defined in the **inventory** file.

The **vars** section represents a list of variables containing selected kernel parameter names and values to which they have to be set.

The **roles** key specifies what system role is going to configure the parameters and values mentioned in the **vars** section.

NOTE

You can modify the kernel parameters and their values in the playbook to fit your needs.

4. Optionally, verify that the syntax in your play is correct.

```
# ansible-playbook --syntax-check kernel-roles.yml

playbook: kernel-roles.yml
```

   This example shows the successful verification of a playbook.

5. Execute your playbook.

```
# ansible-playbook kernel-roles.yml

...

BECOME password:

PLAY [Configure kernel settings]
**********************************************************************************



PLAY RECAP
********************************************************************************************************
fdoe@192.168.122.226      : ok=10   changed=4    unreachable=0    failed=0    skipped=6
rescued=0    ignored=0
pdoe@192.168.122.98       : ok=10   changed=4    unreachable=0    failed=0    skipped=6
rescued=0    ignored=0
```

   Before Ansible runs your playbook, you are going to be prompted for your password and so that a user on managed hosts can be switched to **root**, which is necessary for configuring kernel parameters.

   The recap section shows that the play finished successfully (**failed=0**) for all managed hosts, and that 4 kernel parameters have been applied (**changed=4**).

6. Restart your managed hosts and check the affected kernel parameters to verify that the changes have been applied and persist across reboots.

Additional resources

- Preparing a control node and managed nodes to use RHEL System Roles
- **README.html** and **README.md** files in the **/usr/share/doc/rhel-system-roles/kernel_settings/** directory

- Build Your Inventory
- Configuring Ansible
- Working With Playbooks
- Using Variables
- Roles

CHAPTER 32. USING ADVANCED ERROR REPORTING

When you use the **Advanced Error Reporting** (**AER**), you receive notifications of error events for **Peripheral Component Interconnect Express** (**PCIe**) devices. RHEL enables this kernel feature by default and collects the reported errors in the kernel logs. Moreover, if you use the **rasdaemon** program, these errors are parsed and stored in its database.

32.1. OVERVIEW OF AER

Advanced Error Reporting (**AER**) is a kernel feature that provides enhanced error reporting for **Peripheral Component Interconnect Express** (**PCIe**) devices. The **AER** kernel driver attaches root ports which support **PCIe AER** capability in order to:

- Gather the comprehensive error information
- Report errors to the users
- Perform error recovery actions

When **AER** captures an error, it sends an *error* message to the console. For a repairable error, the console output is a *warning*.

Example 32.1. Example AER output

```
Feb  5 15:41:33 hostname kernel: pcieport 10003:00:00.0: AER: Corrected error received:
id=ae00
Feb  5 15:41:33 hostname kernel: pcieport 10003:00:00.0: AER: Multiple Corrected error received:
id=ae00
Feb  5 15:41:33 hostname kernel: pcieport 10003:00:00.0: PCIe Bus Error: severity=Corrected,
type=Data Link Layer, id=0000(Receiver ID)
Feb  5 15:41:33 hostname kernel: pcieport 10003:00:00.0:   device [8086:2030] error
status/mask=000000c0/00002000
Feb  5 15:41:33 hostname kernel: pcieport 10003:00:00.0:    [ 6] Bad TLP
Feb  5 15:41:33 hostname kernel: pcieport 10003:00:00.0:    [ 7] Bad DLLP
Feb  5 15:41:33 hostname kernel: pcieport 10003:00:00.0: AER: Multiple Corrected error received:
id=ae00
Feb  5 15:41:33 hostname kernel: pcieport 10003:00:00.0: PCIe Bus Error: severity=Corrected,
type=Data Link Layer, id=0000(Receiver ID)
Feb  5 15:41:33 hostname kernel: pcieport 10003:00:00.0:   device [8086:2030] error
status/mask=00000040/00002000
```

32.2. COLLECTING AND DISPLAYING AER MESSAGES

In order to collect and display AER messages, use the **rasdaemon** program.

Procedure

1. Install the **rasdaemon** package.

   ```
   # yum install rasdaemon
   ```

2. Enable and start the **rasdaemon** service.

```
# systemctl enable --now rasdaemon
Created symlink /etc/systemd/system/multi-user.target.wants/rasdaemon.service →
/usr/lib/systemd/system/rasdaemon.service.
```

3. Issue the **ras-mc-ctl** command.

```
# ras-mc-ctl --summary
# ras-mc-ctl --errors
```

   The command displays a summary of the logged errors (the **--summary** option) or displays the errors stored in the error database (the **--errors** option).

Additional resources

- The **rasdaemon(8)** manual page
- The **ras-mc-ctl(8)** manual page