Oracle Linux 10

Managing Kernels and System Boot

G14597-06
March 2026

Oracle Linux 10 Managing Kernels and System Boot,

G14597-06

Contents

Preface

Oracle Linux 10: Managing Kernels and System Boot provides information about configuring the Oracle Linux 10 boot loader, managing kernel boot parameters, loading and unloading kernel modules and how the kernel creates virtual file systems as an interface to kernel runtime controls and data.

1

About System Boot

Understanding the Oracle Linux boot process can help you troubleshoot problems when booting a system. The boot process involves several files, and errors in these files are the usual cause of boot problems. Boot processes and configuration differ depending on whether the hardware uses UEFI firmware or legacy BIOS to handle system boot.

An installation of Oracle Linux includes the GRUB 2 boot loader, which is installed into a location on the hard disk that's accessible to the BIOS or UEFI firmware. The GRUB 2 boot loader is used to load a kernel and the initramfs into memory. After the kernel is fully initialized, it starts the systemd process that manages the rest of the operating system.

About UEFI-Based Booting

On a UEFI-based system running the Oracle Linux release, the system boot process uses the following sequence:

1. When the system is powered on, the system performs a power-on self-test (POST) to detect and check the system's core hardware components such as CPU and memory. The UEFI firmware is then initialized.
2. The UEFI firmware detects any other hardware, such as peripheral components including network devices and storage. The UEFI firmware contains its own boot manager, which can directly interact with boot loaders on various storage devices. The boot manager stores a set of variables including the priority of different boot devices and any detected boot loaders.

   UEFI searches for a FAT32 formatted GPT partition with a specific globally unique identifier (GUID) that identifies it as the EFI System Partition (ESP). This partition contains EFI applications such as boot loaders and other configuration files.

   When more than one boot device is present, the UEFI boot manager uses the appropriate ESP based on the order that's defined in the boot manager. With the `efibootmgr` tool, you can define a different order, if you don't want to use the default definition.
3. The UEFI boot manager loads the default boot loader. Oracle Linux uses a 2-stage boot process to handle the Secure Boot validation process. The 2-stage process includes a first stage boot loader called the `shim` boot loader on the ESP, and the second stage boot loader called GRUB 2. If Secure Boot is disabled, the `shim` boot loader directly loads the GRUB 2 boot loader on the ESP, to continue the boot process. Boot loader files are named according to the system architecture, for example the `shim` bootloader is named `shimx64.efi` on x86_64 systems, and `shimaa64.efi` on aarch64 systems.

   Otherwise, if Secure Boot is enabled, the `shim` boot loader is validated against keys stored in the UEFI Secure Boot key database, and in turn, verifies the GRUB 2 boot loader signature against certificates stored in the UEFI Secure Boot key database or the Machine Owner Key (MOK) database. If the GRUB 2 signature is valid, the GRUB 2 boot loader runs and, in turn, validates the kernel that it's configured to load.

   See Oracle Linux: Working With UEFI Secure Boot for more information on Secure Boot.
4. The boot loader loads the `vmlinuz` kernel image file and the `initramfs` image file into memory. The kernel extracts the contents of the `initramfs` image into a temporary,

memory-based file system (`tmpfs`). The `initramfs` contains essential drivers and utilities needed for booting.

5. The boot loader passes control to the kernel and provides pointers to the `initramfs` and any other boot parameters. The kernel continues system initialization, detecting hardware, loading necessary drivers, and mounting the root file system.
6. The kernel searches for the `init` process within `initramfs` and starts the defined process with a process ID of 1 (PID 1). On Oracle Linux, the default `init` process is configured as `systemd`. See Oracle Linux 10: System Management with systemd for more information.
7. `systemd` runs any other processes defined for it.

> **Note**
>
> Specify any other actions to be processed during the boot process by defining `systemd` units. This method is preferred to using the `/etc/rc.local` file.

About BIOS-Based Booting

On a BIOS-based system running the Oracle Linux release, the boot process is as follows:

1. The system's BIOS performs a power-on self-test (POST), and then detects and initializes any peripheral devices and the hard disk.
2. The BIOS reads the Master Boot Record (MBR) into memory from the boot device. The MBR stores information about the organization of partitions on that device, the partition table, and the boot signature which is used for error detection. The MBR also includes the pointer to the boot loader program (GRUB 2), usually on a dedicated `/boot` partition on the same disk device.
3. The boot loader loads the `vmlinuz` kernel image file and the `initramfs` image file into memory. The kernel then extracts the contents of `initramfs` into a temporary, memory-based file system (`tmpfs`).
4. The kernel loads the driver modules from the `initramfs` file system that are needed to access the root file system.
5. The kernel searches for the `init` process within `initramfs` and starts the defined process with a process ID of 1 (PID 1). On Oracle Linux, the default `init` process is configured as `systemd`. See Oracle Linux 10: System Management with systemd for more information.
6. `systemd` runs any other processes defined for it.

> **Note**
>
> Specify any other actions to be processed during the boot process by defining `systemd` units. This method is preferred to using the `/etc/rc.local` file.

About the GRUB 2 Bootloader

Oracle Linux includes version 2 of the GRand Unified Bootloader (GRUB 2), which loads the operating system onto a system at boot time.

In addition to Oracle Linux, GRUB 2 can load and chain-load many proprietary operating systems. GRUB 2 understands the formats of many different file systems and kernel executable files. GRUB 2 requires the full path to the kernel and `initramfs` relative to the boot or root device. You can configure this information by using the GRUB 2 menu or by entering it on the GRUB 2 command line.

The `grub2-mkconfig` command generates the GRUB 2 configuration file using the template scripts in `/etc/grub.d` and menu-configuration settings taken from the configuration file, `/etc/default/grub`.

The generated GRUB 2 files are read during system boot from `/boot`. The main GRUB 2 configuration file is available at `/boot/grub2/grub.cfg`. On UEFI-based systems, an initial configuration file at `/boot/efi/EFI/redhat/grub.cfg` is used to help direct GRUB 2 to the correct device and location of the main GRUB2 configuration file. Each kernel version's boot parameters are stored in independent configuration files in `/boot/loader/entries`. Each kernel configuration is stored with the file name *`machine_id-kernel_version`*`.el10.`*`arch`*`.conf`.

> **Note**
>
> Don't edit the GRUB 2 configuration file in `/boot` directly.

The default menu entry is set by the value of the `GRUB_DEFAULT` parameter in `/etc/default/grub`. If `GRUB_DEFAULT` is set to `saved`, you can use the `grub2-set-default` and `grub2-reboot` commands to specify the default entry. The command `grub2-set-default` sets the default entry for all reboots, while `grub2-reboot` sets the default entry for the next reboot only.

If you specify a numeric value as the value of `GRUB_DEFAULT` or as an argument to either `grub2-reboot` or `grub2-set-default`, GRUB 2 counts the menu entries in the configuration file starting at 0 for the first entry.

To update GRUB 2 boot loader configuration on Oracle Linux, use the `grubby` command to control and manage all boot requirements.

The `grubby` command-line tool helps you manage GRUB 2 configuration and kernel boot parameters. It's fully scriptable and abstracts low level boot loader details, so you don't need to edit GRUB files by hand. See Using grubby to Manage Kernels for more information.

> **Important**
>
> Persistent kernel command-line changes must be made with `grubby`. Regenerating `/boot/grub2/grub.cfg` from `/etc/default/grub` won't apply those changes.

If you need to change some parameters in the configuration at boot time, you can temporarily change kernel boot parameters in the GRUB 2 boot menu. See Changing Kernel Boot Parameters Before Booting.

For more information about using, configuring, and customizing GRUB 2, see the GNU GRUB Manual, which is also installed as `/usr/share/doc/grub2-tools-2.00/grub.html`.

2
About Linux Kernels

The Linux Foundation provides a hub for open source developers to code, manage, and scale different open technology projects. It also manages the Linux Kernel Organization that exists to distribute various versions of the Linux kernel which is at the core of all Linux distributions, including those used by Oracle Linux. The Linux kernel manages the interactions between the computer hardware and user space applications that run on Oracle Linux.

You must install and run one of these Linux kernels with Oracle Linux:

- **Unbreakable Enterprise Kernel** (UEK): UEK is based on a stable kernel branch from the Linux Foundation, with customer-driven additions, and several UEKs can exist for a specific Oracle Linux release. Its focus is performance, stability, and minimal backports by tracking the mainline source code provided by the Linux Kernel Organization, as closely as is practical. UEK is tested and used to run Oracle Engineered Systems, Oracle Cloud Infrastructure (OCI), and large enterprise deployments for Oracle customers.
  UEK includes some packages or package versions that aren't available in RHCK. Some examples are `btrfs-tools`, `rds`, and `rdma` related packages, and some kernel tuning tools.
- **Red Hat Compatible Kernel** (RHCK): RHCK is fully compatible with the Linux kernel that's distributed in a corresponding Red Hat Enterprise Linux (RHEL) release. You can use RHCK to ensure full compatibility with applications that run on Red Hat Enterprise Linux.

Kernel packages are purposely built to avoid dependencies on a particular kernel type. Any kernel that isn't in use can be removed from the system without impact.

For example, to remove RHCK from a system that's running UEK, you can run:

```
sudo dnf remove kernel-core
```

If a system is using RHCK, you can remove UEK by running:

```
sudo dnf remove kernel-uek-core
```

See Checking Available Kernels on the System to see what kernels are installed on the system. See Changing the Default Kernel to learn how to change the default kernel, for example from RHCK to UEK, or from UEK to RHCK.

> **Important**
>
> Linux kernels are critical for running applications in the Oracle Linux user space. Therefore, you must keep the kernel current with the latest bug fixes, enhancements, and security updates provided by Oracle. To do so, implement a continuous update and upgrade strategy. See Oracle Linux: Ksplice User's Guide for information on how to keep the kernel updated without any requirement to reboot the system. See Oracle Linux: Managing Software on Oracle Linux for general information about keeping software on the system up-to-date.

See Unbreakable Enterprise Kernel documentation for more information about UEK.

For more information about available kernels, see Oracle Linux 10 Kernel Reference.

About Kernel Modules

The boot loader loads the kernel into memory. You can add new code to the kernel by including the source files in the kernel source tree and recompiling the kernel. Kernel modules provide device drivers that enable the kernel to access new hardware, support different file system types, and extend its functionality in other ways. The modules can be dynamically loaded and unloaded on demand. To avoid wasting memory on unused device drivers, Oracle Linux supports loadable kernel modules (LKMs), which enable a system to run with only the device drivers and kernel code that are required to be loaded into memory. See Managing Kernel Modules to see more information on how to manage kernel modules on Oracle Linux.

Note

From UEK 8 onward, kernel packaging changes are applied to provide a more streamlined kernel. The minimal number of core kernel modules and supporting files, such as the files generated by depmod, are provided in the `kernel-uek-modules-core` package. Kernel modules that are required for most server configurations are provided in the `kernel-uek-modules` package, while optional kernel modules for hardware less often found in server configurations, such as Bluetooth, Wi-Fi, and video capture cards, can be found in the `kernel-uek-modules-extra` package. Note that both of these packages require the `linux-firmware` package to be installed.

You can view the contents of these packages by running:

```
dnf repoquery -l kernel-uek-modules-core
dnf repoquery -l kernel-uek-modules
dnf repoquery -l kernel-uek-modules-extra
```

To install all available kernel modules, run:

```
sudo dnf install -y kernel-uek-modules-core kernel-uek-modules kernel-
uek-modules-extra linux-firmware
```

See Unbreakable Enterprise Kernel 8: Release Notes (6.12.0-0.20.20).

Kernel modules can be signed to protect the system from running malicious code at boot time. When UEFI Secure Boot is enabled, only kernel modules that contain the correct signature information can be loaded. See Oracle Linux: Working With UEFI Secure Boot for more information.

About Weak Update Modules

External modules, such as drivers that are installed by using a driver update disk or that are installed from an independent package, are typically installed in the `/lib/modules/`*`kernel-version`*`/extra` directory. Modules that are stored in this directory are preferred over any matching modules that are included with the kernel when these modules are being loaded. Installed external drivers and modules can override existing kernel modules to resolve hardware issues. For each kernel update, these external modules must be made available to

each compatible kernel so that potential boot issues resulting from driver incompatibilities with the affected hardware can be avoided.

Because the requirement to load the external module with each compatible kernel update is system critical, a mechanism exists for external modules to be loaded as weak update modules for compatible kernels.

You make weak update modules available by creating symbolic links to compatible modules in the `/lib/modules/`*`kernel-version`*`/weak-updates` directory. The package manager handles this process automatically when it detects driver modules that are installed in the `/lib/modules/`*`kernel-version`*`/extra` directories for any compatible kernels.

For example, if a newer kernel is compatible with a module that was installed for the previous kernel, an external module (such as `kmod-kvdo`) is automatically added as a symbolic link in the `weak-updates` directory as part of the installation process, as shown in the following command output:

```
ls -l /lib/modules/6.12.0-100.28.2.el10.x86_64/weak-updates/kmod-kvdo/uds


lrwxrwxrwx. 1 root root 68 Jul  8 07:57 uds.ko ->
/lib/modules/6.12.0-100.28.2.el10.x86_64/extra/kmod-kvdo/uds/uds.ko


ls -l /lib/modules/6.12.0-100.28.2.el10.x86_64/weak-updates/kmod-kvdo/vdo


lrwxrwxrwx. 1 root root 68 Jul  8 07:57 uds.ko ->
/lib/modules/6.12.0-100.28.2.el10.x86_64/extra/kmod-kvdo/uds/uds.ko
```

The symbolic link enables the external module to load for kernel updates.

Weak updates are beneficial and ensure that no extra work is required to carry an external module through kernel updates. Any potential driver-related boot issues after kernel upgrades are prevented, so this approach provides a more predictable running of a system and its hardware.

You can remove weak update modules if a kernel version provides a superior or preferred driver or module version. See Removing Weak Update Modules for more information.

For more information about external driver modules and driver update disks, see Oracle Linux 10: Installing Oracle Linux.

About Virtual File Systems and System Configuration

After the system completes the boot process, virtual file systems provide an interface to the running kernel and to processes and hardware that are available on the system. Two virtual file systems are available:

- `procfs`: is mounted at `/proc` and provides an interface to kernel data structures, mostly related to processes and hardware.
- `sysfs`: is mounted at `/sys` and provides information about devices, kernel modules, file systems, and other kernel components.

These virtual file systems are used to control and report on the running kernel, so that the system configuration can be monitored and adjusted while the operating system is live.

Although not part of the kernel virtual file system collection, the `/etc/sysconfig` system configuration file path is also important because it provides an interface to many core system configuration variables that are read when the system boots.

Also see Explore System Configuration Files and Kernel Tunables on Oracle Linux for a hands-on tutorial on how to configure system settings as described in this chapter.

About the /etc/sysconfig Files

The `/etc/sysconfig` directory contains some files that control the system's configuration after boot. The contents of this directory depend on the packages that you have installed on the system. The `/etc/sysconfig` directory largely provides a single view of many configuration files that are used by systemd and related components that control system configuration, such as Network Manager. In newer releases of Oracle Linux, the number of configuration files in this directory is diminishing because configuration is better handled by systemd and other configuration units. For more information about systemd, see Oracle Linux 10: System Management with systemd .

Certain files that you might find in the `/etc/sysconfig` directory include the following:

`atd`
Specifies command line arguments for the `atd` daemon.

`crond`
Passes arguments to the `crond` daemon at boot time.

`chronyd`
Passes arguments to the `chronyd` daemon used for NTP services at boot time.

`firewalld`
Passes arguments to the firewall daemon (`firewalld`) at boot time.

`named`
Passes arguments to the name service daemon at boot time. The `named` daemon is a Domain Name System (DNS) server that's part of the Berkeley Internet Name Domain (BIND) distribution. This server maintains a table that associates host names with IP addresses on the network.

`samba`
Passes arguments to the `smbd`, `nmbd`, and `winbindd` daemons at boot time to support file-sharing connectivity for Windows clients, NetBIOS-over-IP naming service, and connection management to domain controllers.

`selinux`
Controls the state of SELinux on the system. This file is a symbolic link to `/etc/selinux/config`.
For more information, see Oracle Linux: Administering SELinux.

`snapper`
Defines a list of btrfs file systems and thinly provisioned LVM volumes whose contents can be recorded as snapshots by the `snapper` utility.

`sysstat`
Configures logging parameters for system activity data collector utilities such as `sar`.

About the /proc Virtual File System

The files in the `/proc` directory hierarchy contain information about the system hardware and the processes that are running on the system. You can change the configuration of the kernel by writing to certain files that have write permission.

Files that are under the `/proc` directory are virtual files that the kernel creates on demand to present a view of the underlying data structures and system information. As such, `/proc` is an example of a virtual file system. Most virtual files are listed as 0 bytes in size, but they contain large amount of information when viewed.

Virtual files such as `/proc/interrupts`, `/proc/meminfo`, `/proc/mounts`, and `/proc/partitions` provide a view of the system's hardware. Other files, such as `/proc/filesystems` and the files under `/proc/sys`, provide information about the system's configuration and through which you can change configurations as needed.

Files that contain information about related topics are grouped into virtual directories. A separate directory exists in the `/proc` directory for each process that's running on the system. The directory's name corresponds to the numeric process ID. For example, `/proc/1` corresponds to the `systemd` process that has a PID of 1.

To examine virtual files, you can use commands such as `cat`, `less`, and `view`, as shown in the following example:

```
cat /proc/cpuinfo
```

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 42
model name      : Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz
stepping        : 7
cpu MHz         : 2393.714
cache size      : 6144 KB
physical id     : 0
siblings        : 2
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 5
wp              : yes
...
```

For files that contain non human-readable content, you can use utilities such as `lspci`, `free`, `top`, and `sysctl` to access information. For example, the `lspci` command lists PCI devices on a system:

```
sudo lspci
```

```
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev 01)
00:02.0 VGA compatible controller: InnoTek Systemberatung GmbH VirtualBox
Graphics Adapter
00:03.0 Ethernet controller: Intel Corporation 82540EM Gigabit Ethernet
Controller (rev 02)
00:04.0 System peripheral: InnoTek Systemberatung GmbH VirtualBox Guest
Service
00:05.0 Multimedia audio controller: Intel Corporation 82801AA AC'97 Audio
Controller (rev 01)
00:06.0 USB controller: Apple Inc. KeyLargo/Intrepid USB
00:07.0 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08)
00:0b.0 USB controller: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family)
USB2 EHCI Controller
00:0d.0 SATA controller: Intel Corporation 82801HM/HEM (ICH8M/ICH8M-E) SATA
Controller [AHCI mode]
        (rev 02)
...
```

See `procfs` Directory Reference for more information about the different directories available under `/proc`. See Managing Kernel Parameters at Runtime for information on how you can view and change kernel parameters in `/proc/sys` to control system runtime behavior.

About the /sys Virtual File System

In addition to the `/proc` file system, the kernel exports information to the `/sys` virtual file system (`sysfs`). Programs such as the dynamic device manager (`udev`), use `/sys` to access device and device driver information. See Oracle Linux 10: Managing Devices with Udev for more information about device management.

> **Note**
>
> `/sys` exposes kernel data structures and control points, which implies that the directory contains circular references, where a directory links to an ancestor directory. Thus, a `find` command used on `/sys` might never stop.

See `sysfs` Directory Reference to see more information about the directories that you can find in `/sys`.

3

Changing Kernel Boot Parameters Before Booting

To make a temporary change to the boot parameters before booting a kernel, follow these steps:

1. Select the kernel in the GRUB boot menu.

   When the GRUB boot menu appears at the beginning of the boot process, use the arrow keys to highlight the required kernel and press the space bar.

2. Press E to edit the boot configuration for the kernel.

3. Move the cursor to the line starting with `linux`.

   Use the arrow keys to bring the cursor to the end of the line that starts with `linux`, which is the boot configuration line for the kernel.

4. Change the boot parameters.

   You can add parameters such as `systemd.target=runlevel1.target`, which instructs the system to boot into the rescue shell.

   See Kernel Boot Parameter Reference for more information about kernel parameters.

5. Press Ctrl+X to boot the system.

The kernel boots with the new parameters that you specified on the kernel command line. As noted, the changes are temporary and are only active for the current boot session. If you reboot the system, the changes are reverted. You might want to update the GRUB configuration to make changes permanent. Use `grubby` to achieve this. See Changing Kernel Command Line Boot Parameters for more information.

4

Using `grubby` to Manage Kernels

Use the `grubby` command to manage the GRUB 2 configuration on the system, including selecting the default boot kernel or configuring extra kernel command line boot parameters to be used at boot.

See the `grubby(8)` manual page for more information.

Checking Available Kernels on the System

Kernels are named to include the upstream version number and the distribution build numbering. The kernel names on Oracle Linux also include indications of whether they're standard RHCK or whether they're UEK based. Also, the names identify their system architecture. For example, the `el10` suffix indicates an RHCK, while `el10uek` indicates a UEK. See About Linux Kernels for more information.

Several methods are available for checking which kernels are available on a system:

- List the kernels in the `/boot` directory.

  ```
  ls -1 /boot/vmlinuz*
  ```

  The command produces an exact list of kernels available on the system.
- Use the `grubby` command on specific kernels or using the ALL option.

  ```
  sudo grubby --info /boot/vmlinuz-6.12.0*
  sudo grubby --info ALL
  ```

  The command provides fuller information about the boot configuration associated with each kernel in the system's `/boot` directory. The details are based on the GRUB title configuration.

Comparing the Default Kernel to the Running Kernel

The running kernel and the kernel configured as the default kernel that GRUB 2 selects to boot into after a timeout period for the boot menu can differ.

If the default kernel version and the running kernel version aren't identical, the underlying reasons might be one of the following:

- A newer kernel is installed, but the system hasn't been rebooted.
- During a system reboot, a different kernel was manually selected to be the operative kernel.
- The default kernel was manually updated but the system hasn't been rebooted after the update.

To compare the default configured kernel to the running kernel, do the following:

1. Check the default kernel configured in GRUB 2.

To check which kernel is already configured as the current default kernel to use at boot, run:

```
sudo grubby --default-kernel
```

2. Check the running kernel version.

   To check which kernel is running on a system, run:

   ```
   uname -r
   ```

Changing the Default Kernel

Use grubby to set the default kernel that GRUB2 boots into after a timeout period is reached when displaying the GRUB2 boot menu.

You might change the default kernel from RHCK to UEK, from UEK to RHCK, or to switch to a specific kernel version.

You can follow one of two options to set the default kernel in GRUB 2, by using the `grubby` command, choose either of the following:

- Use the `grubby --set-default` command to set the default kernel.

  To switch to a different default kernel, run the following command making sure to specify the full path to the selected default kernel:

  ```
  sudo grubby --set-default /boot/vmlinuz-6.12.0-100.28.2.el10uek.x86_64
  ```

  The change takes effect immediately and persists across system reboots.

- Use the `grubby --set-default-index` command to set the default kernel to match the kernel at a particular index point in the kernel boot list. The index values are displayed when you run the `grubby --info` command

  For example, you can use the `--set-default-index=0` option to set the default kernel to the first kernel listed in the kernel boot index by running:

  ```
  sudo grubby --set-default-index=0
  ```

Example 4-1 Switch to the Most Recent Available RHCK or UEK Kernel

By using the naming convention to identify UEK kernels and RHCK kernels that are available in the `/boot` directory, you can easily switch the default kernel to use the most recent version of either kernel type.

- To switch to the most recent version of UEK on the system, run:

  ```
  sudo grubby --set-default $(ls /boot/vmlinuz-* | grep 'uek' | sort -V |
  tail -1)
  ```

- To switch to the most recent version of RHCK on the system, run:

  ```
  sudo grubby --set-default $(ls /boot/vmlinuz-* | grep -v 'uek' | sort -V |
  tail -1)
  ```

Reboot the system after setting the default kernel to switch to that kernel type.

Changing Kernel Command Line Boot Parameters

Sometimes you might need to edit the GRUB 2 configuration to specify particular kernel boot parameters on the kernel command line. Setting parameters in the GRUB 2 configuration means that the parameters are used for the affected kernels at every boot.
You can update the GRUB 2 boot configuration for a specific kernel, or across all kernels that are installed on the system by doing the following:

1. Use the `grubby --update-kernel` command to update a kernel entry with `--args` to add new arguments or to change existing argument values, or `--remove-args` to remove existing arguments.

   Multiple arguments can be specified for each option in a quoted space-separated list. You can add and remove arguments in the same operation. When using the `--args` option, if an argument already exists the new value replaces the old values.

   To update a specific kernel, run the `grubby --update-kernel` command with the full path to the kernel that you want to update. To update all kernel entries to use a specific kernel boot argument, you can use `grubby --update-kernel=ALL`.

   For example, you can update all kernel entries to change the `loglevel` and `LANG` arguments:

   ```
   sudo grubby --update-kernel=ALL --args "loglevel=3,LANG=en_GB.UTF-8"
   ```

   See Kernel Boot Parameter Reference for more information about kernel parameters.

2. Verify that the changes have taken effect and that the command line arguments are correct for the kernel that you updated.

   For example, if you have made a change to all kernels, use the `grubby --info ALL` command to check that the change is implemented across all kernels:

   ```
   sudo grubby --info ALL
   ```

Checking the Kernel Command Line Last Used to Boot The System

The kernel boot parameters that were last used to boot a system are recorded in `/proc/cmdline`.

For more information, see the `kernel-command-line(7)` manual page.

- Check the contents of `/proc/cmdline` to view the kernel command line that was used to boot the running system.

  ```
  cat /proc/cmdline
  ```

  ```
  BOOT_IMAGE=(hd0,gpt2)/vmlinuz-6.12.0-100.28.2.el10.x86_64
  root=/dev/mapper/ol-root ro
  resume=UUID=b2136352-95d6-4eb7-93db-8f0ca4da7238
  ```

```
rd.luks.uuid=luks-a80f8f10-75b6-45de-b63e-64b8b6a3a94b
crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M rhgb quiet
```

5

Managing Kernel Parameters at Runtime

Some virtual files under `/proc`, and especially under `/proc/sys`, are writable. You can adjust settings in the running kernel through these files. For example, to change the hostname, you can revise the `/proc/sys/kernel/hostname` file as follows:

```
echo www.mydomain.com | sudo tee /proc/sys/kernel/hostname
```

Other files take binary or Boolean values, such as the setting of IP forwarding, which is defined in `/proc/sys/net/ipv4/ip_forward`:

```
cat /proc/sys/net/ipv4/ip_forward
```

```
0
```

```
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
cat /proc/sys/net/ipv4/ip_forward
```

```
1
```

Use the `sysctl` command to view or change values under the `/proc/sys` directory.

> **Note**
>
> Even `root` can't bypass the file access permissions of virtual file entries under `/proc`. If you change the value of a read-only entry such as `/proc/partitions`, no kernel code exists to service the `write()` system call.

For more information, see the `sysctl(8)` and `sysctl.d(5)` manual pages.

Listing Configurable Kernel Parameters and Values

Use the `sysctl` command to browse kernel system parameters that are defined in the `/proc/sys` virtual file system. The following methods of viewing kernel parameters and their values by using the `sysctl` command are available:

- Run `sysctl -a` to view all available kernel parameters and their values for the running kernel.

  ```
  sysctl -a
  ```

  ```
  kernel.sched_child_runs_first = 0
  kernel.sched_min_granularity_ns = 2000000
  kernel.sched_latency_ns = 10000000
  kernel.sched_wakeup_granularity_ns = 2000000
  kernel.sched_shares_ratelimit = 500000
  ...
  ```

  > **Note**
  >
  > The delimiter character in the name of a setting is a period (`.`) rather than a slash (`/`) in a path relative to `/proc/sys`, such as `net.ipv4.ip_forward`. This setting represents `net/ipv4/ip_forward`. As another example, `kernel.msgmax` represents `kernel/msgmax`.

- Display an individual setting or a collection of settings by specifying its name as the argument to `sysctl`.

  ```
  sysctl net.ipv4.ip_forward
  ```

  ```
  net.ipv4.ip_forward = 0
  ```

  For a broader collection of settings, you can specify the name of a collection of settings earlier in the naming hierarchy:

  ```
  sysctl net.ipv4.conf.all
  ```

  ```
  net.ipv4.conf.all.accept_local = 0
  net.ipv4.conf.all.accept_redirects = 0
  net.ipv4.conf.all.accept_source_route = 0
  net.ipv4.conf.all.arp_accept = 0
  net.ipv4.conf.all.arp_announce = 0
  net.ipv4.conf.all.arp_filter = 0
  net.ipv4.conf.all.arp_ignore = 0
  net.ipv4.conf.all.arp_notify = 0
  net.ipv4.conf.all.bc_forwarding = 0
  net.ipv4.conf.all.bootp_relay = 0
  net.ipv4.conf.all.disable_policy = 0
  net.ipv4.conf.all.disable_xfrm = 0
  net.ipv4.conf.all.drop_gratuitous_arp = 0
  net.ipv4.conf.all.drop_unicast_in_l2_multicast = 0
  net.ipv4.conf.all.force_igmp_version = 0
  net.ipv4.conf.all.forwarding = 0
  net.ipv4.conf.all.igmpv2_unsolicited_report_interval = 10000
  net.ipv4.conf.all.igmpv3_unsolicited_report_interval = 1000
  net.ipv4.conf.all.ignore_routes_with_linkdown = 0
  ```

```
net.ipv4.conf.all.log_martians = 0
net.ipv4.conf.all.mc_forwarding = 0
net.ipv4.conf.all.medium_id = 0
net.ipv4.conf.all.promote_secondaries = 0
net.ipv4.conf.all.proxy_arp = 0
net.ipv4.conf.all.proxy_arp_pvlan = 0
net.ipv4.conf.all.route_localnet = 0
net.ipv4.conf.all.rp_filter = 0
net.ipv4.conf.all.secure_redirects = 1
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.all.shared_media = 1
net.ipv4.conf.all.src_valid_mark = 0
net.ipv4.conf.all.tag = 0
```

Updating Kernel Parameters

Use the `sysctl` command to update kernel system parameters that are defined in the `/proc/sys` virtual file system.

1. Use the `sysctl -w` command to set the value for a kernel parameter.

   for example, to change the value of the `net.ipv4.ip_forward` setting to enabled, use the following command format:

   ```
   sudo sysctl -w net.ipv4.ip_forward=1
   ```

   Changes that you make in this way remain in force only until the system is rebooted.

2. To make configuration changes persist after the system is rebooted, add them to the `/etc/sysctl.d` directory as a configuration file.

   Any changes that you make to the files in this directory take effect when the system reboots or if you run the `sysctl --system` command, for example:

   ```
   echo 'net.ipv4.ip_forward=1' | sudo tee /etc/sysctl.d/ip_forward.conf
   grep -r ip_forward /etc/sysctl.d
   ```

   ```
   /etc/sysctl.d/ip_forward.conf:net.ipv4.ip_forward=1
   ```

3. To reset the system to use only the values that are configured to load at boot time, use the `sysctl --system` command.

   ```
   sudo sysctl --system
   ```

   ```
   * Applying /usr/lib/sysctl.d/00-system.conf ...
   net.bridge.bridge-nf-call-ip6tables = 0
   net.bridge.bridge-nf-call-iptables = 0
   net.bridge.bridge-nf-call-arptables = 0
   * Applying /usr/lib/sysctl.d/50-default.conf ...
   kernel.sysrq = 16
   kernel.core_uses_pid = 1
   net.ipv4.conf.default.rp_filter = 1
   net.ipv4.conf.all.rp_filter = 1
   ```

```
net.ipv4.conf.default.accept_source_route = 0
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.default.promote_secondaries = 1
net.ipv4.conf.all.promote_secondaries = 1
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
* Applying /etc/sysctl.d/99-sysctl.conf ...
* Applying /etc/sysctl.d/ip_forward.conf ...
net.ipv4.ip_forward = 1
* Applying /etc/sysctl.conf ...
```

Note that any configuration entries that you added to `/etc/sysctl.d` are read by the system and applied.

6

Managing Kernel Modules

Use the `lsmod` command to view which modules are loaded into the running kernel. Use the `modinfo` command to find out information about a kernel module. Use the `modprobe` command to load a module into the running kernel or to change kernel module parameters. You can also create configuration files in `/etc/modprobe.d/` to control parameters that are used when kernel modules are loaded. You can also configure whether modules load at boot time, by editing configuration in `/etc/modules-load.d/`.

Listing Information About Loaded Modules

Use the `lsmod` command to list modules that are loaded into the kernel and use the `modinfo` command to find out more information about each module. For more information, see the `lsmod(5)` and `modinfo(8)` manual pages.

- Run the `lsmod` command to list the modules that are loaded into the kernel.

  ```
  lsmod
  ```

  ```
  Module                  Size  Used by
  udp_diag               16384  0
  ib_core               311296  0
  tcp_diag               16384  0
  inet_diag              24576  2 tcp_diag,udp_diag
  nfsv3                  49152  0
  nfs_acl                16384  1 nfsv3
  ...
  dm_mirror              24576  0
  dm_region_hash         20480  1 dm_mirror
  dm_log                 20480  2 dm_region_hash,dm_mirror
  ...
  ```

  The output shows the module name, the amount of memory it uses, the number of processes using the module and the names of other modules on which it depends. The module `dm_log`, for example, depends on the `dm_region_hash` and `dm_mirror` modules. The example also shows that two processes are using all three modules.

- Use the `modinfo` command to show detailed information about a module.

  ```
  modinfo ahci
  ```

  ```
  filename:       /lib/modules/6.12.0-100.28.2.el10uek.x86_64/kernel/
  drivers/ata/ahci.ko.xz
  version:        3.0
  license:        GPL
  description:    AHCI SATA low-level driver
  author:         Jeff Garzik
  srcversion:     1DC2CDA088C5DC03187A5E0
  ```

```
alias:          pci:v*d*sv*sd*bc01sc06i01*
...
depends:        libata,libahci
intree:         Y
name:           ahci
retpoline:      Y
vermagic:       6.12.0-100.28.2.el10uek.x86_64 SMP preempt mod_unload
modversions
sig_id:         PKCS#7
signer:         Oracle CA Server
sig_key:        7B:38:D7:DC:38:51:E7:C7:F1:61:C5:5D:8D:CC:6B:1C:90:82:4D:05
sig_hashalgo:   sha512
signature:
64:05:FC:CC:B1:D3:88:91:B6:C9:A2:39:A3:A9:BB:8C:95:11:36:20:

62:9C:95:D9:8B:B8:F6:5F:CC:D2:93:4E:7D:59:E1:80:DB:70:FA:4C:

9B:8D:75:E3:98:AB:9D:BD:94:93:A7:72:0B:28:3B:15:4E:96:0D:E3:

9F:FE:24:1A:09:B5:31:27:F2:EE:45:61:C8:4A:D3:4B:82:07:23:66:

A1:06:F4:DF:B9:FF:D2:78:08:1D:AA:EC:DE:3C:E4:17:BD:69:6A:A5:

                ...

64:F0:4F:E2:4E:F3:47:A5:40:E8:F7:07:68:3F:58:25:32:BA:13:E9:
                00:46:7A:2F:30:73:B4:32:48:76:6B:1E
parm:           marvell_enable:Marvell SATA via AHCI (1 = enabled) (int)
parm:           mobile_lpm_policy:Default LPM policy for mobile chipsets
(int)
...
```

The output includes the following information:

`filename`
Absolute path of the kernel object file.

`version`
Version number of the module. Note that the version number might not be updated for patched modules and might be missing or removed in newer kernels.

`license`
License information for the module.

`description`
Short description of the module.

`author`
Author credit for the module.

`srcversion`
Hash of the source code used to create the module.

`alias`
Internal alias names for the module.

depends
Comma-separated list of any modules on which this module depends.

retpoline
A flag indicating that the module is built that includes a mitigation against the Spectre security vulnerability.

name
The name of the module.

intree
A flag indicating that the module is built from the kernel in-tree source and isn't tainted.

vermagic
Kernel version that was used to compile the module, which is checked against the current kernel when the module is loaded.

sig_id
The method used to store signing keys that might have been used to sign a module for Secure Boot, typically PKCS#7

signer
The name of the signing key used to sign a module for Secure Boot.

sig_key
The signature key identifier for the key used to sign the module.

sig_hashalgo
The algorithm used to generate the signature hash for a signed module.

signature
The signature data for a signed module.

parm
Module parameters and descriptions.

- Use the `modinfo -n` command to find the file path to the module on the file system.

  Modules are loaded into the kernel from kernel object files (`/lib/modules/`*`kernel_version`*`/kernel/*ko*`). To display the absolute path of a kernel object file, specify the `-n` option, for example:

  ```
  modinfo -n parport


  /lib/modules/6.12.0-100.28.2.el10uek.x86_64/kernel/drivers/parport/
  parport.ko.xz
  ```

Loading and Unloading Modules

Modules are loaded and unloaded by using the `modprobe` command. For more information, see the `modprobe(8)` and `modules.dep(5)` manual pages.

- Load a kernel module using the `modprobe` command.

The `modprobe` command loads kernel modules, for example:

```
sudo modprobe nfs
sudo lsmod | grep nfs
```

```
nfs                   266415  0
lockd                  66530  1 nfs
fscache                41704  1 nfs
nfs_acl                 2477  1 nfs
auth_rpcgss            38976  1 nfs
sunrpc                204268  5 nfs,lockd,nfs_acl,auth_rpcgss
```

Include the `-v` (verbose) option to show whether any other modules are loaded to resolve dependencies.

```
sudo modprobe -v nfs
```

```
insmod /lib/modules/6.12.0-100.28.2.el10uek.x86_64/kernel/net/sunrpc/
auth_gss/auth_rpcgss.ko
insmod /lib/modules/6.12.0-100.28.2.el10uek.x86_64/kernel/fs/nfs_common/
nfs_acl.ko
insmod /lib/modules/6.12.0-100.28.2.el10uek.x86_64/kernel/fs/fscache/
fscache.ko
...
```

> **Note**
>
> The `modprobe` command doesn't reload modules that are already loaded. You must first unload a module before you can load it again.

- Unload a module by using the `modprobe -r` command.

  Use the `-r` option to unload kernel modules:

  ```
  sudo modprobe -rv nfs
  ```

  ```
  rmmod /lib/modules/6.12.0-100.28.2.el10uek.x86_64/kernel/fs/nfs/nfs.ko
  rmmod /lib/modules/6.12.0-100.28.2.el10uek.x86_64/kernel/fs/lockd/lockd.ko
  rmmod /lib/modules/6.12.0-100.28.2.el10uek.x86_64/kernel/fs/fscache/
  fscache.ko
  ...
  ```

  Modules are unloaded in reverse order in which they were first loaded. Modules aren't unloaded if a process or another loaded module requires them.

Changing Kernel Module Parameters

Kernel modules, such as hardware drivers, often have custom parameters that can be set to change the behavior of the driver or module. Several mechanisms are available to update module parameters.

- Use sysfs to update module parameters immediately.

  You can change the values of some parameters for loaded modules and built-in drivers by writing the new value to a file under `/sys/module/`*`module_name`*`/parameters`, for example:

  ```
  echo 0 | sudo tee /sys/module/ahci/parameters/skip_host_reset
  ```

  See About the /sys Virtual File System and `sysfs` Directory Reference.

  Note that changes aren't persistent and don't apply automatically after reboot.

- Use the `modprobe` command to change the running configuration for a module.

  To change a module's behavior, specify parameters for the module in the `modprobe` command:

  ```
  sudo modprobe module_name parameter=value ...
  ```

  Separate parameter and value pairs with spaces. Array values are represented by a comma-separated list, for example:

  ```
  sudo modprobe foo parm=bar arrayparm=1,2,3,4
  ```

- Update the `modprobe` configuration for more permanent module configuration changes.

  Configuration files (`/etc/modprobe.d/*.conf`) specify module options, create module aliases, and override the usual behavior of `modprobe` for modules with special requirements. The `/etc/modprobe.conf` file that was used with earlier versions of `modprobe` is also valid if it exists. Entries in the `/etc/modprobe.conf` and `/etc/modprobe.d/*.conf` files use the same syntax. See Modprobe Configuration Reference for more information.

Specifying Modules To Be Loaded at Boot Time

The system loads most modules automatically at boot time. You can also add modules to be loaded by creating a configuration file for the module in the `/etc/modules-load.d` directory. The file name must have the extension `.conf`.

Changes to the `/etc/modules-load.d` directory persist across reboots.

1. To force a module to load at boot time, create a configuration file in `/etc/modules-load.d` for the module.

   For example to force the `bnxt_en.conf` to load at boot time, run the following command:

   ```
   echo bnxt_en | sudo tee /etc/modules-load.d/bnxt_en.conf
   ```

2. Verify that the file exists and contains the module name.

   ```
   cat /etc/modules-load.d/bnxt_en.conf
   ```

   If the module isn't already loaded, you can load it manually by using the `modprobe` command, or you can reboot the system and it loads automatically using the configuration that you have provided.

Preventing Modules From Loading at Boot Time

You can prevent modules from loading at boot time by adding a deny rule in a configuration file in the `/etc/modprobe.d` directory and then rebuilding the initial ramdisk used to load the kernel at boot time.

> **Warning**
>
> Disabling modules can have unintended consequences and can prevent a system from booting or from being fully functional after boot. As a best practice, create a backup ramdisk image before making changes and ensure that the configuration is correct.

1. Create a configuration file to prevent the module from loading.

   For example:

   ```
   sudo tee /etc/modprobe.d/bnxt_en-deny.conf <<'EOF'
   #DENY bnxt_en
   blacklist bnxt_en
   install bnxt_en /bin/false
   EOF
   ```

2. Rebuild the initial ramdisk image.

   ```
   sudo dracut -f -v
   ```

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

   ```
   sudo reboot
   ```

Removing Weak Update Modules

In certain cases, you might remove weak update modules in place of a newer kernel, for example, in the case where an issue with a shipped driver has been resolved in a newer kernel. In this case, you might prefer to use the new driver rather than the external module that you installed as part of a driver update. See About Weak Update Modules for more information.

Two different approaches can be used to remove a weak update module.

1. Remove the symbolic link manually.

Because weak update modules are symbolically linked for each kernel version on the system, you can remove the symbolic link for the module from each kernel where you don't want it to apply. For example:

```
sudo rm -rf /lib/modules/6.12.0-100.28.2.el10uek.x86_64/weak-updates/kmod-kvdo
```

In this example, the weak update module, *kmod-kvdo*, is removed from the kernel, 6.12.0-100.28.2.el10uek.x86_64.

2. Use the `weak-modules` command to remove the module.

   You can use the `weak-modules` command to remove a specified weak update module for all compatible kernels or use the command to remove the weak update module for the current kernel. You can also use the `weak-modules` command similarly to add weak update modules. For more information on this command, run:

   ```
   weak-modules -h
   ```

   You can also use the `weak-modules` command with the `dry-run` option to test the results without making actual changes, for example:

   ```
   weak-modules --remove-kernel --dry-run --verbose
   ```

7

Managing Resources Using Control Groups

Control groups, referred to as `cgroups`, are an Oracle Linux kernel feature that organizes `systemd` services, and if required, individual processes (`PIDs`), into hierarchical groups for allocating system resources, such as CPU, memory, and I/O.

For example, if you have identified three processes that need to be allocated CPU time in a ratio of 150:100:50, you can create three `cgroups`, each with a CPU weight corresponding to one of the three values in the ratio, and assign the appropriate process to each `cgroup`.

> **Important**
>
> **Use `systemd` to configure `cgroups`.**
>
> Manual creation of `cgroup` directories in the `/sys/fs/cgroup` virtual file system (as discussed in this topic) can be helpful for illustrating underlying concepts. However, this approach should only be used for specific scenarios, such as temporary debugging or testing. For most use cases, we recommend using `systemd` to configure `cgroups` to ensure correct and persistent resource management.

By default, `systemd` creates a `cgroup` for the following:

- Each `systemd` service set up on the host.

  For example, a server might have control group `NetworkManager.service` to group processes owned by the `NetworkManager` service, and control group `firewalld.service` to group processes owned by the `firewalld` service, and so on.

- Each user (`UID`) on the host.

The `cgroup` functionality is mounted as a virtual file system under `/sys/fs/cgroup`. Each cgroup has a corresponding directory within `/sys/fs/cgroup` file system. For example, the `cgroups` created by `systemd` for the services it manages can be seen by running the command `ls -l /sys/fs/cgroup/system.slice | grep ".service"` as shown in the following sample code block:

```
ls -l /sys/fs/cgroup/system.slice | grep ".service"
            ...root root 0 Mar 22 10:47 atd.service
            ...root root 0 Mar 22 10:47 auditd.service
            ...root root 0 Mar 22 10:47 chronyd.service
            ...root root 0 Mar 22 10:47 crond.service
            ...root root 0 Mar 22 10:47 dbus-broker.service
            ...root root 0 Mar 22 10:47 dtprobed.service
            ...root root 0 Mar 22 10:47 firewalld.service
            ...root root 0 Mar 22 10:47 httpd.service
            ...
```

You can also create custom `cgroups` by creating directories under the `/sys/fs/cgroup` virtual file system and assigning process IDs (`PIDs`) to different `cgroups` according to system

requirements. However, the recommended practice is to use `systemd` to configure `cgroups` instead of creating the `cgroups` manually under `/sys/fs/cgroup`. See Oracle Linux 10: System Management with systemd for the recommended method of managing `cgroups` through `systemd`.

Oracle Linux 10 provides control groups version 2 (`cgroups v2`). These groups provide a single control group hierarchy against which all resource controllers are mounted. In this hierarchy, you can obtain better proper coordination of resource uses across different resource controllers. This version is an improvement over `cgroups v1` whose over flexibility prevented proper coordination of resource use among the system consumers.

Note that `cgroups v1` is deprecated and isn't available on Oracle Linux 10. The `cgroups v2` functionality is enabled and mounted by default.

For more information about control groups, see the `cgroups(7)` and `sysfs(5)` manual pages.

Verifying cgroups v2

At boot time, Oracle Linux 10 mounts `cgroups v2` by default.

1. Verify that `cgroups v2` is enabled and mounted on the system.

   ```
   sudo mount -l | grep cgroup
   ```

   ```
   cgroup2 on /sys/fs/cgroup type cgroup2
   (rw,nosuid,nodev,noexec,relatime,seclabel,nsdelegate,memory_recursiveprot)
   ```

2. Optionally, check the contents of `/sys/fs/cgroup` directory, which is also called the root control group.

   ```
   ls -l /sys/fs/cgroup/
   ```

   For `cgroups v2`, the files in the directory should have prefixes to their file names, for example, `cgroup.*`, `cpu.*`, `memory.*`, and so on. See About the Control Group File System.

About Kernel Resource Controllers

Control groups manage resource use through *kernel resource controllers*. A kernel resource controller represents a single resource, such as CPU time, memory, network bandwidth, or disk I/O.

To identify mounted resource controllers in the system, check the contents of the `/procs/cgroups` file, for example:

```
less /proc/cgroups
```

```
#subsys_name    hierarchy       num_cgroups     enabled
cpuset  0       103     1
cpu     0       103     1
cpuacct 0       103     1
blkio   0       103     1
memory  0       103     1
devices 0       103     1
```

```
freezer 0       103     1
net_cls 0       103     1
perf_event      0       103     1
net_prio        0       103     1
hugetlb 0       103     1
pids    0       103     1
rdma    0       103     1
misc    0       103     1
```

For a detailed explanation of the kernel resource controllers of `cgroups`, see the `cgroups(7)` manual page.

About the Control Group File System

`cgroup` functionality is mounted as a hierarchical file system in `/sys/fs/cgroup`.

The directory `/sys/fs/cgroup` is also called the root control group. The directory contents might look as follows::

```
ls /sys/fs/cgroup
```

```
cgroup.controllers      cpuset.mems.effective  memory.stat
cgroup.max.depth        cpu.stat               misc.capacity
cgroup.max.descendants  dev-hugepages.mount    sys-fs-fuse-connections.mount
cgroup.procs            dev-mqueue.mount       sys-kernel-config.mount
cgroup.stat             init.scope             sys-kernel-debug.mount
cgroup.subtree_control  io.pressure            sys-kernel-tracing.mount
cgroup.threads          io.stat                system.slice
cpu.pressure            memory.numa_stat       user.slice
cpuset.cpus.effective   memory.pressure
```

You can use the `mkdir` command to create `cgroup` subdirectories within the root control group. For example, you might create the following `cgroup` subdirectories:

- `/sys/fs/cgroup/MyGroups/`
- `/sys/fs/cgroup/MyGroups/cgroup1`
- `/sys/fs/cgroup/MyGroups/cgroup2`

> **Note**
>
> Best design practice is for child `cgroups` to be at least 2 levels deep inside the `/sys/fs/cgroup`. The examples in the preceding list follow this practice by using the first child group, `MyGroups`, as a parent that contains the different `cgroups` needed for the system.

Each `cgroup` in the hierarchy contains the following files:

`cgroup.controllers`
This read-only file lists the controllers available in the current `cgroup`. The contents of this file match the contents of the `cgroup.subtree_control` file in the parent `cgroup`.

cgroup.subtree_control
This file contains those controllers in the `cgroup.controllers` file that are enabled for the current `cgroup`'s immediate child `cgroups`.
When a controller (for example, `pids`) is present in the `cgroup.subtree_control` file, the corresponding controller-interface files (for example, `pids.max`) are automatically created in the immediate children of the current `cgroup`.

For a sample procedure that creates child groups where you can implement resource management for an application, see Setting CPU Weight to Regulate Distribution of CPU Time.

To remove a `cgroup`, ensure that the `cgroup` doesn't contain other child groups, and then remove the directory. For example, to remove child group `/sys/fs/cgroup/MyGroups/cgroup1` you can run the following command:.

```
sudo rmdir /sys/fs/cgroup/MyGroups/cgroup1
```

About Resource Distribution Models

The following distribution models provide you ways of implementing control or regulation in distributing resources for use by `cgroups v2`:

Weights
In this model, the weights of all the control groups are totaled. Each group receives a fraction of the resource based on the ratio of the group's weight against the total weight.
Consider 10 control groups, each with a weight of 100 for a combined total of 1000. In this case, each group can use a tenth of a specified resource.
Weight is typically used to distribute stateless resources. To apply this resource, the `CPUWeight` option is used.

Limits
In this model, a group can use up to the configured amount of a resource. If a resource such as memory usage for a process exceeds the limit, the kernel might stop the process with an out-of-memory (oom) message.
You can also overcommit resources so that the sum of the subgroups limits can exceed the limit of the parent group. Overcommitment assumes that resources in all subgroups aren't likely to all reach their limits at the same time.
To implement this distribution model, the `MemoryMax` option is often used.

Protections
In this model, a group is assigned a *protected boundary*. If the group's resource usage remains within the protected amount, the kernel can't deprive the group of the use of the resource in favor of other groups that are competing for the same resource. In this model, an overcommitment of resources is allowed.
To implement this model, the `MemoryLow` option is often used.

Allocations
In this model, a specific absolute amount is allocated for the use of finite type of resources, such as real-time budget.

Managing `cgroups v2` Using `sysfs`

This section shows you how to create and configure `cgroups` to manage the distribution of resources amongst processes running on the system by using the `sysfs` interface.

> **Important**
>
> We recommend that you use systemd to handle all resource management. See Oracle Linux 10: System Management with systemd for more information. The examples provided here provide the context for actions that systemd performs on a system and show the functionality outside of systemd. The information provided can be helpful when debugging issues with `cgroups`.

The example procedure involves allocating CPU time between `cgroups` that each have different application PIDs assigned to them. The CPU time and application PID values are set in each group's `cpu.weight` and `cgroup.procs` files.

The example also includes the steps required to ensure the `cpu` controller and its associated files, including the `cpu.weight` file, are available in the `cgroups` you need to create under `/sys/fs/cgroup`.

Preparing the Control Group for Distribution of CPU Time

This procedure describes how to manually prepare a control group to manage the distribution of CPU time. Note that the recommended approach to configuring control groups is to use `systemd`.

1. Verify that the `cpu` controller is available at the top of the hierarchy, in the root control group.

   Printing the contents of the `/sys/fs/cgroup/cgroup.controllers` file on the screen:

   ```
   sudo cat /sys/fs/cgroup/cgroup.controllers
   ```

   ```
   cpuset cpu io memory hugetlb pids rdma misc
   ```

   You can add any controllers listed in the `cgroup.controllers` file to the `cgroup.subtree_control` file in the same directory to make them available to the group's immediate child `cgroups`.

2. Add the `cpu` controller to the `cgroup.subtree_control` file to make it available to immediate child `cgroups` of the root.

   By default, only the `memory` and `pids` controllers are in the file. To add the `cpu` controller, type:

   ```
   echo "+cpu" | sudo tee /sys/fs/cgroup/cgroup.subtree_control
   ```

3. Optionally, verify that the `cpu` controller has been added as expected.

   ```
   sudo cat /sys/fs/cgroup/cgroup.subtree_control
   ```

   ```
   cpu memory pids
   ```

4. Create a child group under the root control group to become the new control group for managing CPU resources on applications.

```
sudo mkdir /sys/fs/cgroup/MyGroups
```

5. Optionally, list the contents of the new subdirectory, or child group, and confirm that the **cpu** controller is present as expected.

```
ls -l /sys/fs/cgroup/MyGroups
```

```
-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
…
-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
```

6. Enable the `cpu` controller in `cgroup.subtree_control` file in the `MyGroups` directory to make it available to its immediate child `cgroups`.

```
echo "+cpu" | sudo tee /sys/fs/cgroup/MyGroups/cgroup.subtree_control
```

7. Optionally, verify that the `cpu` controller is enabled for child groups under `MyGroups`.

```
sudo cat /sys/fs/cgroup/MyGroups/cgroup.subtree_control
```

```
cpu
```

Setting CPU Weight to Regulate Distribution of CPU Time

This procedure describes how to set CPU weight for three different processes by using a control group to manage the distribution of CPU time. Note that the recommended approach to configuring control groups is to use `systemd`.

This procedure is based on the following assumptions:

- The application that's consuming CPU resources excessively is `sha1sum`, as shown in the following sample output of the `top` command:

```
sudo top
```

```
...
PID   USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+
COMMAND
33301 root      20   0   18720   1756   1468 R  99.0   0.0   0:31.09
sha1sum
33302 root      20   0   18720   1772   1480 R  99.0   0.0   0:30.54
sha1sum
33303 root      20   0   18720   1772   1480 R  99.0   0.0   0:30.54
sha1sum
1 root      20   0  109724  17196  11032 S   0.0   0.1   0:03.28
systemd
2 root      20   0       0      0      0 S   0.0   0.0   0:00.00
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 `sha1sum` processes have PIDs 33301, 33302, and 33303, as listed in the preceding sample output.

> **Important**
>
> As a prerequisite to the following procedure, you must complete the preparations of `cgroup-v2` as described in Preparing the Control Group for Distribution of CPU Time. If you skipped those preparations, you can't complete this procedure.

1. Create 3 child groups in the `MyGroups` subdirectory.

```
sudo mkdir /sys/fs/cgroup/MyGroups/g1
sudo mkdir /sys/fs/cgroup/MyGroups/g2
sudo mkdir /sys/fs/cgroup/MyGroups/g3
```

2. Configure the CPU weight for each child group.

```
echo "150" | sudo tee /sys/fs/cgroup/MyGroups/g1/cpu.weight
echo "100" | sudo tee /sys/fs/cgroup/MyGroups/g2/cpu.weight
echo "50" | sudo tee /sys/fs/cgroup/MyGroups/g3/cpu.weight
```

3. Apply the application PIDs to their corresponding child groups.

```
echo "33301" | sudo tee /sys/fs/cgroup/MyGroups/g1/cgroup.procs
echo "33302" | sudo tee /sys/fs/cgroup/MyGroups/g2/cgroup.procs
echo "33303" | sudo /sys/fs/cgroup/MyGroups/g3/cgroup.procs
```

These commands set the selected applications to become members of the `MyGroups/g*/` control groups. The CPU time for each `sha1sum` process depends on the CPU time distribution as configured for each group.

The weights of the `g1`, `g2`, and `g3` groups that have running processes are summed up at the level of `MyGroups`, which is the parent control group.

With this configuration, when all processes run at the same time, the kernel allocates to each of the `sha1sum` processes the proportionate CPU time based on their respective `cgroup`'s `cpu.weight` file, as follows:

Child group	`cpu.weight` setting	Percent of CPU time allocation
g1	150	~50% (150/300)
g2	100	~33% (100/300)
g3	50	~16% (50/300)

If one child group has no running processes, then the CPU time allocation for running processes is recalculated based on the total weight of the remaining child groups with running processes. For example, if the `g2` child group doesn't have any running processes, then the total weight becomes 200, which is the weight of `g1+g3`. In this case, the CPU time for `g1` becomes 150/200 (~75%) and for `g3`, 50/200 (~25%)

4. Check that the applications are running in the specified control groups.

```
sudo cat /proc/33301/cgroup /proc/33302/cgroup /proc/33303/cgroup
```

```
0::/MyGroups/g1
0::/MyGroups/g2
0::/MyGroups/g3
```

5. Check the current CPU consumption after you have set the CPU weights.

```
top
```

```
...
PID   USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+
COMMAND
33301 root      20   0   18720   1748   1460 R  49.5   0.0 415:05.87
sha1sum
33302 root      20   0   18720   1756   1464 R  32.9   0.0 412:58.33
sha1sum
33303 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
...
```

8
Configuring the Watchdog Service

Watchdog is an Oracle Linux service that runs in the background to monitor host availability and processes and reports back to the kernel. If the Watchdog service fails to notify the kernel that the system is healthy, the kernel typically automatically reboots the system.

To install the Watchdog package, run:

```
sudo dnf install watchdog
```

To configure the Watchdog service, edit the `/etc/watchdog.conf` file. The `watchdog.conf` file includes all Watchdog configuration properties. For information on how to edit this file, see the `watchdog.conf(5)` manual page.

To enable and start the Watchdog service, run:

```
sudo systemctl enable --now watchdog
```

The Watchdog service immediately starts and runs in the background.

> **Note**
>
> The Watchdog service starts and runs immediately after a power reset.

9

Working With Kernel Dumps

The Kdump feature provides a kernel crash information dumping mechanism in Oracle Linux. The `kdump` service saves the contents of the system's memory for later analysis. Kdump includes a second kernel that resides in a reserved part of the system memory, so that Kdump can capture information about a stopped kernel.

Kdump uses the `kexec` system call to boot into the second kernel, called a *capture kernel*, without the need to reboot the system, and then captures the contents of the stopped kernel's memory as a kernel crash dump (`vmcore`) and saves it. The `vmcore` kernel crash dump can help with finding the cause of the malfunction.

Enabling the Kdump feature is highly recommended because a crash dump might be the only information source that's available if a system failure occurs. Kdump is vital in many mission-critical environments.

Before enabling Kdump, ensure that the system meets all the memory requirements for using Kdump. To capture a kernel crash dump and save it for further analysis, reserve part of the system's memory permanently for that purpose. When you do so, that part of the system's memory is no longer available to the main kernel.

For information about configuring Kdump by using the Cockpit web console, see Oracle Linux: Using the Cockpit Web Console

Kdump System Memory Requirements

The following table lists the minimum amount of reserved memory that's required to use Kdump, based on the system's architecture and the amount of available memory.

Table 9-1 Kdump Memory Requirements

Architecture	Available Memory	Minimum Reserved Memory
x86_64	1 GB to 64 GB	448 MB of RAM
	64 GB and more	512 MB of RAM
Arm (aarch64)	2 GB to 8 GB	256 MB of RAM
	8 GB and more	1 GB of RAM

Installing Kdump

During an Oracle Linux interactive installation with the graphical installer, you have the option to enable Kdump and specify how much system memory is reserved for Kdump. The installer screen is titled **Kdump** and is available from the main Installation Summary screen of the installer. If you don't enable Kdump at installation time, or it's not enabled by default during an installation, as in the case of a custom kickstart installation, you can install and enable the feature by using the command line.

Before you install and configure Kdump by using the command line, ensure that:

- The system meets all the necessary memory specifications. For details, see Kdump System Memory Requirements.
- You understand how Kdump reserves memory and have made the appropriate memory reservation for the system. For details, see #unique_48

To install and enable Kdump, follow these steps:

1. Install the `kdump` package.

   Install Kdump by executing the following command:

   ```
   sudo dnf install kexec-tools
   ```

2. Configure the Kdump output location.

   For instructions, see Configuring the Kdump Output Location.

3. Configure where Kdump saves the dump data.

   For instructions, see Configuring the Default Kdump Failure State.

4. Start and enable the `kdump` service.

   Run the following command to start the `kdump` service, and enable it to start automatically on system boot:

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

Configuring Kdump

When you install and configure Kdump, the following files are changed:

- `/boot/grub2/grub.cfg`: Appends the `crashkernel` option to the kernel line to specify the amount of reserved memory and any offset value.
- `/etc/kdump.conf`: Sets the location in which the dump file can be written, the filtering level for the `makedumpfile` command, and the default behavior to take if the dump fails. See the comments in the file for information about the enabled parameters.

When you edit these files, you must reboot the system for the changes to take effect.

For more information, see the `kdump.conf(5)` manual page.

Configuring the Kdump Output Location

After installing and enabling Kdump, you can define the location in which the resulting output is saved. For Oracle Linux, Kdump files are stored in the `/var/crash` directory by default, or in the `/var/oled/crash` directory on Oracle Cloud Infrastructure compute instances.

> **Important**
>
> Make sure that the Kdump output path is at a location that has sufficient disk space to store the kernel crash dump file.
>
> On systems with memory of 1 TB or more, such as the larger bare metal shapes on Oracle Cloud Infrastructure, you should cater to around 20 GB disk space at minimum.

1. Edit the configuration file at `/etc/kdump.conf` file and remove the # comment character at the beginning of each line that you want to enable.

   For example, to add a new directory location, prefix it with the `path` keyword:

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

   Use `raw` to output directly to a specific device in the `/dev` directory.

   You can also manually specify the output file system for a particular device by using its label, name, or UUID. For example:

   ```
   ext4 UUID=5b065be6-9ce0-4154-8bf3-b7c4c7dc7365
   ```

   Kernel crash dump files can also be transferred over a secure shell connection, as shown in the following example:

   ```
   ssh user@example.com
   sshkey /root/.ssh/mykey
   ```

   You can also set the Kernel crash dump files to be exported to a compatible network share:

   ```
   nfs example.com:/output
   ```

   See the `kdump.conf(5)` manual page for more information.

2. Restart the Kdump service.

   When you have finished configuring the output location for Kdump, restart the `kdump` service.

   ```
   sudo systemctl restart kdump.service
   ```

Configuring the Default Kdump Failure State

By default, if `kdump` fails to send its result to the configured output locations, it reboots the server. This action deletes any data that has been collected for the dump. To prevent this outcome, change the Kdump configuration.

1. Edit `/etc/kdump.conf` to uncomment and change the `default` value in the file as follows:

   ```
   default dump_to_rootfs
   ```

   The `dump_to_rootfs` option tries to save the result to a local directory, which can be useful if a network share is unreachable. You can use `shell` instead to copy the data manually from the command line.

> **Note**
>
> The `poweroff`, `restart`, and `halt` options are also valid for the default `kdump` failure state. However, performing these actions causes you to lose the collected data if those actions are performed.

2. Restart the Kdump service.

   When you have finished configuring the output location for Kdump, restart the `kdump` service.

   ```
   sudo systemctl restart kdump.service
   ```

Analyzing Kdump Output

You can use the `crash` utility to analyze the contents of `kdump` core dumps in a shell prompt, which is useful when troubleshooting problems. For more detailed information about using the `crash` utility, see the `crash(8)` manual page.

1. Install the `crash` package:

   ```
   sudo dnf install crash
   ```

2. Provide the location of the kernel `debuginfo` module and the location of the core dump as parameters to the `crash` utility, for example:

   ```
   sudo crash /usr/lib/debug/lib/modules/$(uname -r)/vmlinux \
    /var/crash/127.0.0.1-2025-05-03-12:38:25/vmcore
   ```

   In the previous command, we use `$(uname -r)` to identify the running kernel version within the command and `127.0.0.1-2025-05-03-12:38:25` represents the *ipaddress-timestamp*.

3. Use the `crash` shell to get more information about the core dump.

   Inside the `crash` shell, use the `help log` command for information about how to use the `log` command, which displays the kernel `log_buf` contents in chronological order.

   You can also use the `bt`, `ps`, `vm`, and `files` commands to get more information about the core dump:

   bt
   Displays a task's kernel-stack backtrace.

   ps
   Displays process status for the specified, or all, processes in the system.

   vm
   Displays basic virtual memory information of a context.

   files
   Displays information about open files in a context.

4. When you have finished analyzing the core dump, exit the shell.

You can either type the `exit` command or use the q command as shorthand.

Using Early Kdump

Early Kdump loads the crash kernel and `initramfs` early enough to capture `vmcore` information for early malfunctions.

Because the `kdump` service starts too late, early malfunctions don't trigger the `kdump` kernel to boot, which prevents the capture of diagnostic information. To address that problem, you can enable early Kdump by adding a `dracut` module so that the crash kernel and `initramfs` are loaded as early as possible.

> **Note**
>
> The following limitations apply for early Kdump:
>
> - The feature doesn't work with `Fadump`.
> - Early Kdump becomes active the moment the system's `initramfs` begins to be processed. Any malfunction that occurs before that moment isn't captured even if early Kdump is enabled.

For more information about configuring early Kdump, see the step-by-step instructions in the `/usr/share/doc/kexec-tools/early-kdump-howto.txt` file.

10

Oracle Linux 10 Kernel Reference

The information provided in this reference can help you to understand the different kernel versions that are available for Oracle Linux 10.

Oracle Linux 10 Kernel Version Matrix

The following tables provide an overview of kernel availability on Oracle Linux 10 releases. You can use these tables to identify the initial UEK and RHCK kernel package versions for each release and to see which UEK releases are available for each update level. Never pin a system to an update level or a particular kernel version for an indefinite period, or the system becomes insecure as it doesn't receive the latest patches and security updates.

> **Note**
>
> Oracle Linux 10 releases on aarch64 platforms only include UEK kernel packages. RHCK isn't provided for this platform architecture.

Table 10-1 Kernel Availability on Oracle Linux 10 Releases for x86_64

Oracle Linux 10 Update Level	Initial UEK Version	Initial RHCK Version	UEK 8
10.1	kernel-uek-6.12.0-105.51.5	kernel-6.12.0-124.8.1	Yes, default
10.0	kernel-uek-6.12.0-100.28.2	kernel-6.12.0-55.9.1.0.1	Yes, default

Table 10-2 Kernel Availability on Oracle Linux 10 Releases for aarch64

Oracle Linux 10 Update Level	Initial UEK Version
10.1	kernel-uek-6.12.0-105.51.5
10.0	kernel-uek-6.12.0-100.28.2

UEK Version Mappings

The following table provides the mapping between UEK releases and the kernel versions that they're associated with. The table is useful when trying to identify a particular UEK update level against the kernel version reported by the system. Note that to keep a system secure with the latest patches, always run the latest update level for any UEK release.

Table 10-3 UEK Update Levels and Kernel Versions

UEK Update Level	Kernel Major Release	Kernel Release Date
UEK 8 U2	6.12.0-200*	March, 2026
UEK 8 U1	6.12.0-100*	June, 2025
UEK 8	6.12.0-0*	April, 2025

11

Kernel Boot Parameter Reference

The following table describes some commonly used kernel boot parameters.

Option	Description
`0`, `1`, `2`, `3`, `4`, `5`, or `6`, or `systemd.unit=runlevel`*`N`*`.target`	Specifies the nearest `systemd`-equivalent system-state target to match a legacy SysV run level. *N* can take an integer value between 0 and 6. Systemd maps system-state targets to mimic the legacy SysV init system. For a description of system-state targets, see Oracle Linux 10: System Management with systemd .
`1`, `s`, `S`, `single`, or `systemd.unit=rescue.target`	Specifies the rescue shell. The system boots to single-user mode prompts for the `root` password.
`3` or `systemd.unit=multi-user.target`	Specifies the `systemd` target for multiuser, nongraphical login.
`5` or `systemd.unit=graphical.target`	Specifies the `systemd` target for multiuser, graphical login.
`-b`, `emergency`, or `systemd.unit=emergency.target`	Specifies emergency mode. The system boots to single-user mode and prompts for the `root` password. Fewer services are started than when in rescue mode.
`KEYBOARDTYPE=`*`kbtype`*	Specifies the keyboard type, which is written to `/etc/sysconfig/keyboard` in the `initramfs`.
`KEYTABLE=`*`kbtype`*	Specifies the keyboard layout, which is written to `/etc/sysconfig/keyboard` in the `initramfs`.
`LANG=`*`language_territory.codeset`*	Specifies the system language and code set, which is written to `/etc/sysconfig/i18n` in the `initramfs`.
`max_loop=`*`N`*	Specifies the number of loop devices (`/dev/loop*`) that are available for accessing files as block devices. The default and maximum values of *N* are 8 and 255.
`nouptrack`	Disables Ksplice Uptrack updates from being applied to the kernel.
`quiet`	Reduces debugging output.
`rd_LUKS_UUID=`*`UUID`*	Activates an encrypted Linux Unified Key Setup (LUKS) partition with the specified UUID.
`rd_LVM_VG=`*`vg`*`/lv_`*`vol`*	Specifies an LVM volume group and volume to be activated.

Option	Description
`rd_NO_LUKS`	Disables detection of an encrypted LUKS partition.
`rhgb`	Specifies to use the Red Hat graphical boot display to indicate the progress of booting.
`rn_NO_DM`	Disables Device-Mapper (DM) RAID detection.
`rn_NO_MD`	Disables Multiple Device (MD) RAID detection.
`ro root=/dev/mapper/`*`vg`*`-lv_root`	Specifies that the root file system is to be mounted read-only, and specifies the root file system by the device path of its LVM volume (where *vg* is the name of the volume group).
`rw root=UUID=`*`UUID`*	Specifies that the root (`/`) file system is to be mounted read-writable at boot time, and specifies the root partition by its UUID.
`selinux=0`	Disables SELinux and touches the `/.autorelabel` file so that SELinux file contexts are automatically relabeled the next time you boot with SELinux enabled. Don't disable SELinux in production environments. Rather, set SELinux to permissive mode.
`enforcing=0`	Sets SELinux to permissive mode until next rebooted. In permissive mode, file contexts are automatically labeled and denials are logged, but applications can continue to function. Use SELinux permissive mode to debug SELinux issues.
`SYSFONT=`*`font`*	Specifies the console font, which is written to `/etc/sysconfig/i18n` in the `initramfs`.

Parameters That Control System Performance

The following parameters control various aspects of system performance:

Parameter	Description
`fs.file-max`	Specifies the maximum number of open files for all processes. Increase the value of this parameter if you see messages about running out of file handles.

Parameter	Description
kernel.io_uring_disabled	Specifies the disabled setting for creating io_uring instances. io_uring provides an interface to handle asynchronous I/O operations that can improve performance for storage and networking. io_uring is supported with UEK and is enabled by default when running UEK on Oracle Linux. You can set the following values for the io_uring parameter: • kernel.io_uring_disabled=0 (default). This setting specifies all processes can create io_uring instances. • kernel.io_uring_disabled=1. This setting specifies only processes with CAP_SYS_ADMIN privileges can create io_uring instances. • kernel.io_uring_disabled=2. This setting specifies that io_uring instance creation is disabled for all users.
net.core.netdev_max_backlog	Specifies the size of the receiver backlog queue, which is used if an interface receives packets faster than the kernel can process them. If this queue is too small, packets are lost at the receiver, rather than on the network.
net.core.rmem_max	Specifies the maximum read socket buffer size. To minimize network packet loss, this buffer must be large enough to handle incoming network packets.
net.core.wmem_max	Specifies the maximum write socket buffer size. To minimize network packet loss, this buffer must be large enough to handle outgoing network packets.
net.ipv4.tcp_available_congestion_control	Displays the TCP congestion avoidance algorithms that are available for use. Use the modprobe command if you need to load additional modules such as tcp_htcp to implement the htcp algorithm.
net.ipv4.tcp_congestion_control	Specifies which TCP congestion avoidance algorithm is used.
net.ipv4.tcp_max_syn_backlog	Specifies the number of outstanding SYN requests that are allowed. Increase the value of this parameter if you see synflood warnings in the logs that are caused by the server being overloaded by legitimate connection attempts.
net.ipv4.tcp_rmem	Specifies minimum, default, and maximum receive buffer sizes that are used for a TCP socket. The maximum value can't be larger than net.core.rmem_max.
net.ipv4.tcp_wmem	Specifies minimum, default, and maximum send buffer sizes that are used for a TCP socket. The maximum value can't be larger than net.core.wmem_max.

Parameter	Description
`vm.swappiness`	Specifies how likely the kernel is to write loaded pages to swap rather than drop pages from the system page cache. When set to 0, swapping only occurs to avoid an out of memory condition. When set to 100, the kernel swaps aggressively. For a desktop system, setting a lower value can improve system responsiveness by decreasing latency. The default value is 60. **⚠ Caution** This parameter is intended for use with laptop computers to reduce power consumption by the hard disk. Don't adjust this value on server systems.

Parameters That Control Kernel Panics

The following parameters control the circumstances under which a kernel panic can occur.

Parameter	Description
`kernel.hung_task_panic`	If set to 1, the kernel panics if any kernel or user thread sleeps in the `TASK_UNINTERRUPTIBLE` state (*D state*) for more than `kernel.hung_task_timeout_secs` seconds. A process remains in D state while waiting for I/O to complete. You can't stop or interrupt a process in this state. The default value is 0, which disables the panic. **✓ Tip** To diagnose a hung thread, you can examine `/proc/`*`PID`*`/stack`, which displays the kernel stack for both kernel and user threads.

Parameter	Description
`kernel.hung_task_timeout_secs`	Specifies how long a user or kernel thread can remain in D state before a warning message is generated or the kernel panics, if the value of `kernel.hung_task_panic` is 1. The default value is 120 seconds. A value of 0 disables the timeout.
`kernel.nmi_watchdog`	If set to 1 (default), enables the nonmaskable interrupt (NMI) watchdog thread in the kernel. To use the NMI switch or the OProfile system profiler to generate an undefined NMI, set the value of `kernel.nmi_watchdog` to 0.
`kernel.panic`	Specifies the number of seconds after a panic before a system automatically resets itself. If the value is 0, which is the default value, the system becomes suspended, and you can collect detailed information about the panic for troubleshooting. To enable automatic reset, set a nonzero value. If you require a memory image (`vmcore`), leave enough time for Kdump to create this image. The suggested value is 30 seconds, although large systems require a longer time.
`kernel.panic_on_io_nmi`	If set to 0 (default), the system tries to continue operations if the kernel detects an I/O channel check (IOCHK) NMI that typically indicates a uncorrectable hardware error. If set to 1, the system panics.
`kernel.panic_on_oops`	If set to 0, the system tries to continue operations if the kernel detects an `oops` or BUG condition. If set to 1 (default), the system delays a few seconds to give the kernel log daemon, `klogd`, time to record the oops output before the panic occurs. In an OCFS2 cluster. set the value to 1 to specify that a system must panic if a kernel oops occurs. If a kernel thread required for cluster operation fails, the system must reset itself. Otherwise, another node might not detect whether a node is slow to respond or unable to respond, causing cluster operations to halt.
`kernel.panic_on_unrecovered_nmi`	If set to 0 (default), the system tries to continue operations if the kernel detects an NMI that might indicate an uncorrectable parity or ECC memory error. If set to 1, the system panics.
`kernel.softlockup_panic`	If set to 0 (default), the system tries to continue operations if the kernel detects a *soft-lockup* error that causes the NMI watchdog thread to fail to update its timestamp for more than twice the value of `kernel.watchdog_thresh` seconds. If set to 1, the system panics.

Parameter	Description
`kernel.unknown_nmi_panic`	If set to 1, the system panics if the kernel detects an undefined NMI. You can generate an undefined NMI by manually pressing an NMI switch. As the NMI watchdog thread also uses the undefined NMI, set the value of `kernel.unknown_nmi_panic` to 0 if you set `kernel.nmi_watchdog` to 1.
`kernel.watchdog_thresh`	Specifies the interval between generating an NMI performance monitoring interrupt that the kernel uses to check for *hard-lockup* and *soft-lockup* errors. A hard-lockup error is assumed if a CPU is unresponsive to the interrupt for more than `kernel.watchdog_thresh` seconds. The default value is 10 seconds. A value of 0 disables the detection of lockup errors.
`vm.panic_on_oom`	If set to 0 (default), the kernel's OOM-killer scans through the entire task list and stops a memory-hogging process to avoid a panic. If set to 1, the kernel panics but can survive under certain conditions. If a process limits allocations to certain nodes by using memory policies or cpusets, and those nodes reach memory exhaustion status, the OOM-killer can stop one process. No panic occurs in this case because other nodes' memory might be free and the system as a whole might not yet be out of memory. If set to 2, the kernel always panics when an OOM condition occurs. Settings of 1 and 2 are for intended for use with clusters, depending on the defined failover policy.

12

Modprobe Configuration Reference

The following are commonly used commands in `modprobe` configuration files:

`alias`
Creates an alternative name for a module. The alias can include shell wildcards. To create an alias for the `sd-mod` module:

```
alias block-major-8-* sd_mod
```

`blacklist`
Ignore a module's internal alias that's displayed by the `modinfo` command. This command is typically used in the following conditions:

- The associated hardware isn't required.
- Two or more modules both support the same devices.
- A module invalidly claims to support a device.

For example, to demote the alias for the frame-buffer driver `cirrusfb`, type:

```
blacklist cirrusfb
```

The `/etc/modprobe.d/blacklist.conf` file prevents hotplug scripts from loading a module so that a different driver binds the module instead regardless of which driver happens to be probed first. If it doesn't already exist, you must create it.

`install`
Runs a shell command instead of loading a module into the kernel. For example, load the module `snd-emu10k1-synth` instead of `snd-emu10k1`:

```
install snd-emu10k1 /sbin/modprobe --ignore-install snd-emu10k1 && /sbin/
modprobe snd-emu10k1-synth
```

`options`
Defines options for a module. For example, to define the `nohwcrypt` and `qos` options for the `b43` module, type:

```
options b43 nohwcrypt=1 qos=0
```

`remove`
Runs a shell command instead of unloading a module. To unmount `/proc/fs/nfsd` before unloading the `nfsd` module, type:

```
remove nfsd { /bin/umount /proc/fs/nfsd > /dev/null 2>&1 || :; } ;
/sbin/modprobe -r --first-time --ignore-remove nfsd
```

For more information, see the `modprobe.conf(5)` manual page.

13

sysfs Directory Reference

The following table describes some useful virtual directories under the /sys directory hierarchy.

For more information, see https://www.kernel.org/doc/Documentation/filesystems/sysfs.txt.

Table 13-1 Virtual Directories Under /sys

Virtual Directory	Description
block	Contains subdirectories for block devices. For example: /sys/block/sda.
bus	Contains subdirectories for each physical bus type, such as pci, pcmcia, scsi, or usb. Under each bus type, the devices directory lists discovered devices, and the drivers directory contains directories for each device driver.
class	Contains subdirectories for every class of device that's registered with the kernel.
dev	Contains the char/ and block/ directories. Inside these two directories are symbolic links named *major*:*minor*. These symbolic links point to the sysfs directory for the particular device. The /sys/dev directory provides a quick way to look up the sysfs interface for a device from the result of the stat(2) operation.
devices	Contains the global device hierarchy of all devices on the system. The platform directory contains peripheral devices such as device controllers that are specific to a particular platform. The system directory contains non peripheral devices such as CPUs and APICs. The virtual directory contains virtual and pseudo devices. See Oracle Linux 10: Managing Devices with Udev for more information about device management.
firmware	Contains subdirectories for firmware objects.
fs	Contains subdirectories for file system objects.
kernel	Contains subdirectories for other kernel objects
module	Contains subdirectories for each module loaded into the kernel. You can alter some parameter values for loaded modules. See Modprobe Configuration Reference.
power	Contains attributes that control the system's power state.

14

procfs Directory Reference

The following table describes the most useful virtual files and directories under the /proc directory hierarchy. For more information, see the proc(5) manual page.

Table 14-1 Useful Virtual Files and Directories Under the /proc Directory

Virtual File or Directory	Description
PID (Directory)	Provides information about the process with the process ID (*PID*). The directory's owner and group is same as the process's. Useful files under the directory include: **cmdline** Command path. **cwd** Symbolic link to the process's current working directory. **environ** Environment variables. **exe** Symbolic link to the command executable. **fd/*N*** File descriptors. **maps** Memory maps to executable and library files. **root** Symbolic link to the effective root directory for the process. **stack** The contents of the kernel stack. **status** Run state and memory usage.
buddyinfo	Provides information for diagnosing memory fragmentation.
bus (directory)	Contains information about the various buses (such as pci and usb) that are available on the system. You can use commands such as lspci, lspcmcia, and lsusb to display information for such devices.
cgroups	Provides information about the resource control groups that are in use on the system.

Table 14-1 (Cont.) Useful Virtual Files and Directories Under the /proc Directory

Virtual File or Directory	Description
`cmdline`	Lists parameters passed to the kernel at boot time.
`cpuinfo`	Provides information about the system's CPUs.
`crypto`	Provides information about all installed cryptographic cyphers.
`devices`	Lists the names and major device numbers of all currently configured characters and block devices.
`dma`	Lists the direct memory access (DMA) channels that are currently in use.
`driver` (directory)	Contains information about drivers used by the kernel, such as those for nonvolatile RAM (`nvram`), the real-time clock (`rtc`), and memory allocation for sound (`snd-page-alloc`).
`execdomains`	Lists the execution domains for binaries that the Oracle Linux kernel provides.
`filesystems`	Lists the file system types that the kernel provides. Entries marked with `nodev` aren't in use.
`fs` (directory)	Contains information about mounted file systems, organized by file system type.
`interrupts`	Records the number of interrupts per interrupt request queue (IRQ) for each CPU after system startup.
`iomem`	Lists the system memory map for each physical device.
`ioports`	Lists the range of I/O port addresses that the kernel uses with devices.
`irq` (directory)	Contains information about each IRQ. You can configure the affinity between each IRQ and the system CPUs.
`kcore`	Presents the system's physical memory in `core` file format that you can examine using a debugger such as `crash` or `gdb`. This file isn't human-readable.
`kmsg`	Records kernel-generated messages, which are picked up by programs such as `dmesg`.
`loadavg`	Displays the system load averages (number of queued processes) for the past 1, 5, and 15 minutes, the number of running processes, the total number of processes, and the PID of the process that's running.

Table 14-1 (Cont.) Useful Virtual Files and Directories Under the /proc Directory

Virtual File or Directory	Description
`locks`	Displays information about the file locks that the kernel is currently holding on behalf of processes. The information provided includes: • lock class (`FLOCK` or `POSIX`) • lock type (`ADVISORY` or `MANDATORY`) • access type (`READ` or `WRITE`) • process ID • major device, minor device, and inode numbers • bounds of the locked region
`mdstat`	Lists information about multiple-disk RAID devices.
`meminfo`	Reports the system's usage of memory in more detail than is available using the `free` or `top` commands.
`modules`	Displays information about the modules that are currently loaded into the kernel. The `lsmod` command formats and displays the same information, excluding the kernel memory offset of a module.
`mounts`	Lists information about all mounted file systems.
`net` (directory)	Provides information about networking protocol, parameters, and statistics. Each directory and virtual file describes aspects of the configuration of the system's network.
`partitions`	Lists the major and minor device numbers, number of blocks, and name of partitions mounted by the system.
`scsi/device_info`	Provides information about SCSI devices.
`scsi/scsi` and `scsi/sg/*`	Provide information about configured SCSI devices, including vendor, model, channel, ID, and LUN data.
`self`	Symbolic link to the process that's examining `/proc`.
`slabinfo`	Provides detailed information about slab memory usage.
`softirqs`	Displays information about software interrupts (`softirqs`). A `softirq` is similar to a hardware interrupt (`hardirq`) and configures the kernel to perform asynchronous processing that would take too long during a hardware interrupt.

Table 14-1 (Cont.) Useful Virtual Files and Directories Under the /proc Directory

Virtual File or Directory	Description
`stat`	Records information about the system from when it was started, including: **`cpu`** Total CPU time (measured in `jiffies`) spent in user mode, low-priority user mode, system mode, idle, waiting for I/O, handling hardirq events, and handling softirq events. **`cpuN`** Times for CPU *N*.
`swaps`	Provides information about swap devices. The units of size and usage are in kilobytes.
`sys` (directory)	Provides information about the system and also enables you to enable, disable, or modify kernel features. You can write new settings to any file that has write permission. See Managing Kernel Parameters at Runtime. The following subdirectory hierarchies of `/proc/sys` contain virtual files, some of whose values you can alter: **`dev`** Device parameters. **`fs`** File system parameters. **`kernel`** Kernel configuration parameters. **`net`** Networking parameters.
`sysvipc` (directory)	Provides information about the usage of System V Interprocess Communication (IPC) resources for messages (`msg`), semaphores (`sem`), and shared memory (`shm`).
`tty` (directory)	Provides information about the available and currently used terminal devices on the system. The `drivers` virtual file lists the devices that are currently configured.
`vmstat`	Provides information about virtual memory usage.