Oracle Linux 9

Managing Core System Configuration

F56701-15
May 2024

Oracle Linux 9 Managing Core System Configuration,

F56701-15

Contents

Preface

Oracle Linux 9: Managing Core System Configuration provides information about configuring Oracle Linux 9 systems, including the boot loader configuration and processes, system devices, services and settings, as well as kernel parameters.

Documentation License

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1 Managing Kernels and System Boot

This chapter describes the Oracle Linux boot process and how to configure and use the GRand Unified Bootloader (GRUB) version 2 and boot-related kernel parameters.

> **Tip:**
>
> See Manage the Boot Kernel for Oracle Linux for a hands-on tutorial and video demonstrations on configuring the boot kernel in Oracle Linux.

About the Boot Process

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.

About UEFI-Based Booting

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

1. The system's UEFI firmware performs a power-on self-test (POST) and then detects and initializes peripheral devices and the hard disk.
2. UEFI searches for a 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. In case of the presence of multiple boot devices, 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 checks whether Secure Boot is enabled. If Secure Boot is disabled, the boot manager runs the GRUB 2 bootloader on the ESP.

   Otherwise, the boot manager requests a certificate from the boot loader and validates this against keys stored in the UEFI Secure Boot key database. To handle the certificate validation process, the environment is configured to perform a 2-stage boot process and the `shim.efi` application that's responsible for certification is loaded first before loading the GRUB 2 bootloader. If the certificate is valid, the 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 into memory and extracts the contents of the `initramfs` image file into a temporary, memory-based file system (`tmpfs`).
5. The kernel loads the driver modules from the `initramfs` file system that are needed to access the root file system.

6. The kernel starts the `systemd` process with a process ID of 1 (PID 1). See About the systemd Service Manager.
7. `systemd` runs any additional processes defined for it.

> **Note:**
>
> Specify any other actions to be processed during the boot process by defining your own `systemd` unit. This method is the preferred approach than 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). The boot program itself can be on the same device or on another device.
3. The boot loader loads the `vmlinuz` kernel image file into memory and extracts the contents of the `initramfs` image file 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 starts the `systemd` process with a process ID of 1 (PID 1). See About the systemd Service Manager for more information.
6. `systemd` runs any additional processes defined for it.

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

About the GRUB 2 Bootloader

In addition to Oracle Linux, GRUB 2 can load and chain-load many proprietary operating systems. GRUB 2 understands the formats of file systems and kernel executable files. Therefore, it can load an arbitrary OS without needing to know the exact location of the kernel on the boot device. GRUB 2 requires only the file name and drive partitions to load a kernel. You can configure this information by using the GRUB 2 menu or by entering it on the command line.

GRUB 2 behavior is based on configuration files. On BIOS-based systems, the configuration file is `/boot/grub2/grub.cfg`. On UEFI-based systems, the configuration file is `/boot/efi/EFI/redhat/grub.cfg`. 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*.el8.*arch*.conf`.

Note:

Don't edit the GRUB 2 configuration file directly.

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

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 subsequent 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.

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`.

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 multiple 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 Oracles 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.

> **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.

Managing Kernels in GRUB 2 Using `grubby`

You can use the `grubby` command to view and manage kernels.

Use the following command to display the kernels that are installed and configured on the system:

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

To configure a specific kernel as the default boot kernel, run:

```
sudo grubby --set-default /boot/vmlinuz-4.18.0-80.el8.x86_64
```

You can also use the `grubby` command to update a kernel configuration entry to add or remove kernel boot arguments, for example:

```
sudo grubby --remove-args="rhgb quiet" --
args=rd_LUKS_UUID=luks-39fec799-6a6c-4ac1-ac7c-1d68f2e6b1a4 \
--update-kernel /boot/vmlinuz-4.18.0-80.el8.x86_64
```

For more information about the `grubby` command, see the `grubby(8)` manual page.

For a hands-on tutorial on the use of `grubby` to manage kernels, see Manage the Boot Kernel for Oracle Linux.

Important:

Security scanners on the system might report CVEs for any kernel on the system that's not used as the running or default kernel. As a good practice and to avoid unnecessary noise and false positives that are being reported by the scanner, remove unused kernels after you switch kernels.

For example, if you switch to RHCK from UEK, follow these steps to ensure a proper transition:

1. Set the default kernel to RHCK with the appropriate `grubby` command.

   See the preceding examples for the correct command syntax.
2. Reboot the system to ensure that you're now running RHCK.
3. Remove the UEK kernel.

   ```
   sudo dnf remove kernel-uek
   ```
4. Disable the UEK repositories by running the following command:

   ```
   for uek_repo in $(dnf repolist enabled|grep UEK|awk '{print $1}');
   do sudo dnf config-manager disable $uek_repo; done
   ```
5. Downgrade kernel plumbing packages and remove orphan packages.

   ```
   sudo dnf downgrade $(sudo package-cleanup --orphans)
   ```

Likewise, if you choose to use UEK as the standard kernel, consider removing RHCK. For instructions to remove RHCK, see Remove the Red Hat Compatible Kernel With the kernel-transition Package.

Kernel Boot Parameters

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 About System-State Targets.
`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.

Option	Description
`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.
`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.
`SYSFONT=`*`font`*	Specifies the console font, which is written to `/etc/sysconfig/i18n` in the `initramfs`.

The kernel boot parameters that were last used to boot a system are recorded in `/proc/cmdline`, for example:

```
sudo cat /proc/cmdline
```

```
BOOT_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-80.el8.x86_64 root=/dev/mapper/ol-root
ro \
crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M resume=/dev/mapper/ol-swap
rd.lvm.lv=ol/root \
rd.lvm.lv=ol/swap rhgb quiet
```

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

Modifying Kernel Boot Parameters Before Booting

To modify boot parameters before booting a kernel, follow these steps:

1. 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. 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. Modify the boot parameters.

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

Modifying GRUB 2 Default Kernel Boot Parameters

To modify the boot parameters for the GRUB 2 configuration so that these parameters are applied by default at every reboot, follow these steps:

1. Edit `/etc/default/grub` and add parameter settings to the `GRUB_CMDLINE_LINUX` definition, for example:

   ```
   GRUB_CMDLINE_LINUX="vconsole.font=latarcyrheb-sun16 vconsole.keymap=uk
   crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M  rd.lvm.lv=ol/swap
   rd.lvm.lv=ol/root biosdevname=0
   rhgb quiet systemd.unit=runlevel3.target"
   ```

   This example adds the parameter `systemd.unit=runlevel3.target` so that the system boots into multiuser, nongraphical mode by default.
2. Rebuild `/boot/grub2/grub.cfg`:

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

   The change takes effect at the next system reboot of all configured kernels.

Note:

For systems that boot with UEFI, the `grub.cfg` file is located in the `/boot/efi/EFI/redhat` directory because the boot configuration is stored on a dedicated FAT32-formatted partition.

After the system has successfully booted, the `EFI` folder on that partition is mounted inside the `/boot/efi` directory on the root file system for Oracle Linux.

2 Managing System Services With systemd

The systemd daemon is the system initialization and service manager in Oracle Linux. This chapter describes how to use systemd to manage system processes, services and systemd targets.

Tip:

See Use systemd on Oracle Linux for a hands-on tutorial and video demonstrations on working with systemd in Oracle Linux.

About the systemd Service Manager

The systemd daemon is the first process that starts after a system boots and is the final process that's running when the system shuts down. systemd controls the final stages of booting and prepares the system for use. It also speeds up booting by loading services concurrently.

systemd reads its configuration from files in the /etc/systemd directory. For example, the /etc/systemd/system.conf file controls how systemd handles system initialization.

The systemd daemon starts services during the boot process by reading the symbolic link /etc/systemd/system/default.target. The following example shows the value of /etc/systemd/system/default.target on a system configured to boot to a multiuser mode without a graphical user interface, a target called multi-user.target:

```
sudo ls -l /etc/systemd/system/default.target


 /etc/systemd/system/default.target -> /usr/lib/systemd/system/multi-
user.target
```

Note:

You can use a kernel boot parameter to override the default system target. See Kernel Boot Parameters.

systemd Units

systemd organizes the different types of resources it manages into units. Most units are configured in unit configuration files that enable you to configure these units according to system needs. In addition to the files, you can also use systemd runtime commands to configure the units.

The following list describes some system units that you can manage on an Oracle Linux system by using `systemd`:

Services
Service unit configuration files have the filename format *service_name*.`service`, for example `sshd.service`, `crond.service`, and `httpd.service`.
Service units start and control daemons and the processes of which the daemons consist.
The following example shows how you might start the `systemd` service unit for the Apache HTTP server, `httpd.service`:

```
sudo systemctl start httpd.service
```

Targets
Target unit configuration files have the filename format *target_name*.`target`, for example `graphical.target`.
Targets are similar to runlevels. A system reaches different targets during the boot process as resources get configured. For example, a system reaches `network-pre.target` before it reaches the target `network-online.target`.
Many target units have dependencies. For example, the activation of `graphical.target` (for a graphical session) fails unless `multi-user.target` (for multiuser system) is also active.

File System Mount Points
Mount unit configuration files have the filename format *mount_point_name*.`mount`.
Mount units enable you to mount filesystems at boot time. For example, you can run the following command to mount the temporary file system (`tmpfs`) on `/tmp` at boot time:

```
sudo systemctl enable tmp.mount
```

Devices
Device unit configuration files have the filename format *device_unit_name*.`device`.
Device units are named after the `/sys` and `/dev` paths they control. For example, the device `/dev/sda5` is exposed in systemd as `dev-sda5.device`.
Device units enable you to implement device-based activation.

Sockets
Socket unit configuration files have the filename format *socket_unit_name*.`socket`.
Each "*.`socket`" file needs a corresponding "*.`service`" file to configure the service to start on incoming traffic on the socket.
Socket units enable you to implement socket-based activation.

Timers
Timer unit configuration files have the filename format *timer_unit_name*.`timer`.
Each "*.`timer`" file needs a corresponding "*.`service`" file to configure the service to start at a configured timer event. A `Unit` configuration entry can be used to specify a service that's named differently to the timer unit, if required.
Timer units can control when service units are run and can act as an alternative to using the cron daemon. Timer units can be configured for calendar time events, monotonic time events, and can be run asynchronously.

Paths to `systemd` unit configuration files vary depending on their purpose and whether `systemd` is running in 'user' or 'system' mode. For example, configuration for units that are installed from packages might be available in `/usr/lib/systemd/system` or in `/usr/local/lib/systemd/system`, while a user mode configuration unit is likely to be stored in `$HOME/.config/systemd/user`. See the `systemd.unit(5)` manual page for more information.

See About System-State Targets.

About System-State Targets

By using system-state targets, you can control `systemd` so that it starts only the services that are required for a specific purpose. For example, you set the default target to `multi-user.target` on a production server so that the graphical user interface isn't used when the system boots. In a case where you need to troubleshoot or perform diagnostics, you might consider setting the target to `rescue.target`, where only `root` logs onto the system to run the minimum number of services.

Each run level defines the services that `systemd` stops or starts. As an example, `systemd` starts network services for `multi-user.target` and the X Window System for `graphical.target`, and stops both services for `rescue.target`.

Table 2-1 shows the commonly used system-state targets and the equivalent runlevel targets.

Table 2-1 System-State Targets and Equivalent Runlevel Targets

System-State Targets	Equivalent Runlevel Targets	Description
`graphical.target`	`runlevel5.target`	Set up a multiuser system with networking and display manager.

`multi-user.target`	`runlevel2.target`
	`runlevel3.target`
	`runlevel4.target`
		Set up a nongraphical multiuser system with networking.

`poweroff.target`	`runlevel0.target`	Shut down and power off the system.
`reboot.target`	`runlevel6.target`	Shut down and reboot the system.
`rescue.target`	`runlevel1.target`	Set up a rescue shell.

Note that `runlevel*` targets are implemented as symbolic links.

For more information, see the `systemd.target(5)` manual page.

Displaying Default and Active System-State Targets

To display the default system-state target, use the `systemctl get-default` command:

```
sudo systemctl get-default
```

```
graphical.target
```

To display the active targets on a system, use the `systemctl list-units --type target` command:

```
sudo systemctl list-units --type target [--all]
```

```
UNIT                       LOAD   ACTIVE SUB    DESCRIPTION
basic.target               loaded active active Basic System
cryptsetup.target          loaded active active Local Encrypted Volumes
getty.target               loaded active active Login Prompts
graphical.target           loaded active active Graphical Interface
local-fs-pre.target        loaded active active Local File Systems (Pre)
local-fs.target            loaded active active Local File Systems
multi-user.target          loaded active active Multi-User System
network-online.target      loaded active active Network is Online
network-pre.target         loaded active active Network (Pre)
network.target             loaded active active Network
nfs-client.target          loaded active active NFS client services
nss-user-lookup.target     loaded active active User and Group Name Lookups
paths.target               loaded active active Paths
remote-fs-pre.target       loaded active active Remote File Systems (Pre)
remote-fs.target           loaded active active Remote File Systems
rpc_pipefs.target          loaded active active rpc_pipefs.target
rpcbind.target             loaded active active RPC Port Mapper
slices.target              loaded active active Slices
sockets.target             loaded active active Sockets
sound.target               loaded active active Sound Card
sshd-keygen.target         loaded active active sshd-keygen.target
swap.target                loaded active active Swap
sysinit.target             loaded active active System Initialization
timers.target              loaded active active Timers

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.

24 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
```

The output for a system with the `graphical` target active shows that this target depends on other active targets, including `network` and `sound` to support networking and sound.

Use the `--all` option to include inactive targets in the list.

For more information, see the `systemctl(1)` and `systemd.target(5)` manual pages.

Note:

Target is only one of `systemd` types of units. To display all the types of units, use the following command:

```
sudo systemctl -t help
```

```
Available unit types:
service
mount
swap
socket
target
device
automount
timer
path
slice
scope
```

Changing Default and Active System-State Targets

Use the `systemctl set-default` command to change the default system-state target:

```
sudo systemctl set-default multi-user.target
```

```
Removed /etc/systemd/system/default.target.
Created symlink /etc/systemd/system/default.target → /usr/lib/systemd/system/
multi-user.target
```

Note:

This command changes the target to which the default target is linked, but doesn't change the state of the system.

To change the current active system target, use the `systemctl isolate` command, for example:

```
sudo systemctl isolate multi-user.target
```

For more information, see the `systemctl(1)` manual page.

Shutting Down, Suspending, and Rebooting the System

Table 2-2 systemctl Commands for Shutting Down, Suspending, and Rebooting a System

systemctl Command	Description
`systemctl halt`	Halt the system.
`systemctl hibernate`	Put the system into hibernation.
`systemctl hybrid-sleep`	Put the system into hibernation and suspend its operation.
`systemctl poweroff`	Halt and power off the system.
`systemctl reboot`	Reboot the system.
`systemctl suspend`	Suspend the system.

For more information, see the `systemctl(1)` manual page.

Managing Services

Services in an Oracle Linux system are managed by the `systemctl` *`subcommand`* command.

Examples of subcommands are `enable`, `disable`, `stop`, `start`, `restart`, reload, and `status`.

For more information, see the `systemctl(1)` manual page.

Starting and Stopping Services

To start a service, use the `systemctl start` command:

```
sudo systemctl start sshd
```

To stop a service, use the `systemctl stop` command:

```
sudo systemctl stop sshd
```

Changing the state of a service only lasts while the system remains at the same state. If you stop a service and then change the system-state target to one in which the service is configured to run (for example, by rebooting the system), the service restarts. Similarly, starting a service doesn't enable the service to start following a reboot. See Enabling and Disabling Services.

Enabling and Disabling Services

You can use the `systemctl` command to enable or disable a service from starting when the system boots, for example:

```
sudo systemctl enable httpd
```

```
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service
→ /usr/lib/systemd/system/httpd.service.
```

The `enable` command activates a service by creating a symbolic link for the lowest-level system-state target at which the service should start. In the previous example, the command creates the symbolic link `httpd.service` for the `multi-user` target.

Disabling a service removes the symbolic link:

```
sudo systemctl disable httpd
```

```
Removed /etc/systemd/system/multi-user.target.wants/httpd.service.
```

To check whether a service is enabled, use `is-enabled` subcommand as shown in the following examples:

```
sudo systemctl is-enabled httpd
```

```
disabled
```

```
sudo systemctl is-enabled sshd
```

```
enabled
```

After running the `systemctl disable` command, the service can still be started or stopped by user accounts, scripts, and other processes. However, if you need to ensure that the service might be started inadvertently, for example, by a conflicting service, then use the `systemctl mask` command as follows:

```
sudo systemctl mask httpd
```

```
Created symlink from '/etc/systemd/system/multi-user.target.wants/
httpd.service' to '/dev/null'
```

The `mask` command sets the service reference to `/dev/null`. If you try to start a service that has been masked, you will receive an error as shown in the following example:

```
sudo systemctl start httpd


Failed to start httpd.service: Unit is masked.
```

To relink the service reference back to the matching service unit configuration file, use the `systemctl unmask` command:

```
sudo systemctl unmask httpd
```

For more information, see the `systemctl(1)` manual page.

Displaying the Status of Services

To check whether a service is running, use the `is-active` subcommand. The output would either be *active*) or *inactive*, as shown in the following examples:

```
sudo systemctl is-active httpd


active


systemctl is-active sshd


inactive
```

The `status` subcommand provides a detailed summary of the status of a service, including a tree that displays the tasks in the control group (`CGroup`) that the service implements:

```
sudo systemctl status httpd


httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor 
preset: disabled)
   Active: active (running) since ...
     Docs: man:httpd.service(8)
 Main PID: 11832 (httpd)
   Status: "Started, listening on: port 80"
    Tasks: 213 (limit: 26213)
   Memory: 32.5M
   CGroup: /system.slice/httpd.service
           ├─11832 /usr/sbin/httpd -DFOREGROUND
           ├─11833 /usr/sbin/httpd -DFOREGROUND
           ├─11834 /usr/sbin/httpd -DFOREGROUND
           ├─11835 /usr/sbin/httpd -DFOREGROUND
           └─11836 /usr/sbin/httpd -DFOREGROUND
```

```
Jul 17 00:14:32 Unknown systemd[1]: Starting The Apache HTTP Server...
Jul 17 00:14:32 Unknown httpd[11832]: Server configured, listening on: port 80
Jul 17 00:14:32 Unknown systemd[1]: Started The Apache HTTP Server.
```

A `cgroup` is a collection of processes that are bound together so that you can control their access to system resources. In the example, the `cgroup` for the `httpd` service is `httpd.service`, which is in the `system` slice.

Slices divide the `cgroups` on a system into different categories. To display the slice and `cgroup` hierarchy, use the `systemd-cgls` command:

```
sudo systemd-cgls
```

```
Control group /:
-.slice
├─user.slice
│ └─user-1000.slice
│   ├─user@1000.service
│   │ └─init.scope
│   │   ├─6488 /usr/lib/systemd/systemd --user
│   │   └─6492 (sd-pam)
│   └─session-7.scope
│     ├─6484 sshd: root [priv]
│     ├─6498 sshd: root@pts/0
│     ├─6499 -bash
│     ├─6524 sudo systemd-cgls
│     ├─6526 systemd-cgls
│     └─6527 less
├─init.scope
│ └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 16
└─system.slice
  ├─rngd.service
  │ └─1266 /sbin/rngd -f --fill-watermark=0
  ├─irqbalance.service
  │ └─1247 /usr/sbin/irqbalance --foreground
  ├─libstoragemgmt.service
  │ └─1201 /usr/bin/lsmd -d
  ├─systemd-udevd.service
  │ └─1060 /usr/lib/systemd/systemd-udevd
  ├─polkit.service
  │ └─1241 /usr/lib/polkit-1/polkitd --no-debug
  ├─chronyd.service
  │ └─1249 /usr/sbin/chronyd
  ├─auditd.service
  │ ├─1152 /sbin/auditd
  │ └─1154 /usr/sbin/sedispatch
  ├─tuned.service
  │ └─1382 /usr/libexec/platform-python -Es /usr/sbin/tuned -l -P
  ├─systemd-journald.service
  │ └─1027 /usr/lib/systemd/systemd-journald
  ├─atd.service
  │ └─1812 /usr/sbin/atd -f
  ├─sshd.service
  │ └─1781 /usr/sbin/sshd
```

The `system.slice` contains services and other system processes. `user.slice` contains user processes, which run within transient cgroups called *scopes*. In the example, the processes for the user with ID 1000 are running in the scope `session-7.scope` under the slice `/user.slice/user-1000.slice`.

You can use the `systemctl` command to limit the CPU, I/O, memory, and other resources that are available to the processes in service and scope cgroups. See Controlling Access to System Resources.

For more information, see the `systemctl(1)` and `systemd-cgls(1)` manual pages.

Controlling Access to System Resources

Use the `systemctl` command to control a cgroup's access to system resources, for example:

```
sudo systemctl [--runtime] set-property httpd CPUShares=512 MemoryLimit=1G
```

`CPUShare` controls access to CPU resources. As the default value is 1024, a value of 512 halves the access to CPU time that the processes in the `cgroup` have. Similarly, `MemoryLimit` controls the maximum amount of memory that the `cgroup` can use.

Note:

You don't need to specify the `.service` extension to the name of a service.

If you specify the `--runtime` option, the setting doesn't persist across system reboots.

Alternatively, you can change the resource settings for a service under the `[Service]` heading in the service's configuration file in `/usr/lib/systemd/system`. After editing the file, make `systemd` reload its configuration files and then restart the service:

```
sudo systemctl daemon-reload
sudo systemctl restart service
```

You can run general commands within scopes and use `systemctl` to control the access that these transient cgroups have to system resources. To run a command within in a scope, use the `systemd-run` command:

```
sudo systemd-run --scope --unit=group_name [--slice=slice_name]
```

If you don't want to create the group under the default `system` slice, you can specify another slice or the name of a new slice. The following example runs a command named `mymonitor` in `mymon.scope` under `myslice.slice`:

```
sudo systemd-run --scope --unit=mymon --slice=myslice mymonitor


Running as unit mymon.scope.
```

Note:

If you don't specify the `--scope` option, the control group is a created as a service rather than as a scope.

You can then use `systemctl` to control the access that a scope has to system resources in the same way as for a service. However, unlike a service, you must specify the `.scope` extension, for example:

```
sudo systemctl --runtime set-property mymon.scope CPUShares=256
```

For more information see the `systemctl(1)`, `systemd-cgls(1)`, and `systemd.resource-control(5)` manual pages.

Running systemctl on a Remote System

If the `sshd` service is running on a remote Oracle Linux system, specify the `-H` option with the `systemctl` command to control the system remotely, for example:

```
sudo systemctl -H root@10.0.0.2 status sshd
```

```
root@10.0.0.2's password: password
sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled)
   Active: active (running) since ...
  Process: 1498 ExecStartPre=/usr/sbin/sshd-keygen (code=exited, status=0/
SUCCESS)
 Main PID: 1524 (sshd)
   CGroup: /system.slice/sshd.service
```

For more information see the `systemctl(1)` manual page.

Modifying systemd Service Unit Files

To change the configuration of `systemd` services, copy the files with `.service`, `.target`, `.mount` and `.socket` extensions from `/usr/lib/systemd/system` to `/etc/systemd/system`.

After you have copied the files, you can edit the versions in `/etc/systemd/system`. The files in `/etc/systemd/system` take precedence over the versions in `/usr/lib/systemd/system`. Files in `/etc/systemd/system` aren't overwritten when you update a package that touches files in `/usr/lib/systemd/system`.

To revert to the default `systemd` configuration for a particular service, you can either rename or delete the copies in `/etc/systemd/system`.

The following sections describe the different parts of a service unit file that you can edit and customize for a system.

About Service Unit Files

Services run based on their corresponding service unit files. A service unit file typically contains the following sections, with each section having its respective defined options that determine how a specific service runs:

`[Unit]`
Contains information about the service.

`[`*`UnitType`*`]:`
Contains options that are specific to the unit type of the file. For example, in a service unit file this section is titled `[Service]` and contains options that are specific to units of the service type, such as `ExecStart` or `StandardOutput`.
Only those unit types that offer options specific to their type have such a section.

`[Install]`
Contains installation information for the specific unit. The information in this section is used by the `systemctl enable` and `systemctl disable` commands.

A service unit file might contain the following configurations for a service.

```
[Unit]
Description=A test service used to develop a service unit file template

[Service]
Type=simple
StandardOutput=journal
ExecStart=/usr/lib/systemd/helloworld.sh

[Install]
WantedBy=default.target
```

Configurable Options in Service Unit Files describes some commonly used configured options available under each section. A complete list is also available in the `systemd.service(5)` and `systemd.unit(5)` manual pages.

Configurable Options in Service Unit Files

Each of the following lists deals with a separate section of the service unit file.

Description of Options Under [Unit] Section

The following list provides a general overview of the commonly used configurable options available in the `[Unit]` section of service unit file:

`Description`
Provides information about the service. The information is displayed when you run the `systemctl status` command on the unit.

`Documentation`
Contains a space-separated list of URIs referencing documentation for this unit or its configuration.

`After`
Configures the unit to only run after the units listed in the option finish starting up.
In the following example, if the file *var3*`.service` has the following entry, then it's only started after units *`var1`*`.service` and *`var2`*`.service` have started:

```
After=var1.service var2.service
```

`Requires`
Configures a unit to have requirement dependencies on other units. If a unit is activated, those listed in its `Requires` option are also activated.

`Wants`
A less stringent version of the `Requires` option. For example, a specific unit can be activated even if one of those listed in its `Wants` option fails to start.

Description of Options Under [Service] Section

This following list gives a general overview of the commonly used configurable options available in the `[Service]` section of a service unit file.

`Type`
Configures the process start-up type for the service unit.
By default, this parameter's value is `simple`, which indicates that the service's main process is that which is started by the `ExecStart` parameter.
Typically, if a service's type is `simple`, then the definition can be omitted from the file.

`StandardOutput`
Configures the how the service's events are logged. For example, consider a service unit file has the following entry:

```
StandardOutput=journal
```

In the example, the value `journal` indicates that the events are recorded in the journal, which can be viewed by using the `journalctl` command.

`ExecStart`
Specifies the full path and command that starts the service, for example, `/usr/bin/npm start`.

`ExecStop`
Specifies the commands to run to stop the service started through `ExecStart`.

`ExecReload`
Specifies the commands to run to trigger a configuration reload in the service.

`Restart`
Configures whether the service is to be restarted when the service process exits, is stopped, or when a timeout is reached.

> **Note:**
>
> This option doesn't apply when the process is stopped cleanly by a `systemd` operation, for example a `systemctl stop` or `systemctl restart`. In these cases, the service isn't restarted by this configuration option.

`RemainAfterExit`
A Boolean value that configures whether the service is to be considered active even when all of its processes have exited. The default value is `no`.

Description of Options Under [Install] Section

This following list gives a general overview of the commonly used configurable options available in the `[Install]` section of service unit file.

`Alias`
A space-separated list of names for a unit.
At installation time, `systemctl enable` creates symlinks from these names to the unit filename.
Aliases are only effective when the unit is enabled.

`RequiredBy`
Configures the service to be required by other units.
For example, consider a unit file *`var1`*`.service` that has the following configuration added to it:

```
RequiredBy=var2.service var3.service
```

When *`var1`*`.service` is enabled, both *`var2`*`.service` and *`var3`*`.service` are granted a `Requires` dependency upon *`var1`*`.service`. This dependency is defined by a symbolic link that's created in the `.requires` folder of each dependent service (*`var2`*`.service` and `var3.service`) that points to the *`var1`*`.service` system unit file.

`WantedBy`
Specifies a list of units that are to be granted a `wants` dependency upon the service whose file you're editing.
For example, consider a unit file *`var1`*`.service` that has the following configuration added to it:

```
WantedBy=var2.service var3.service
```

When *`var1`*`.service` is enabled, both *`var2`*`.service` and *`var3`*`.service` are granted a `Wants` dependency upon *`var1`*`.service`. This dependency is defined by a symbolic link that's created in the "`.wants`" folder of each dependent service (*`var2`*`.service` and `var3.service`) that points to the system unit file for *`var1`*`.service` .

`Also`
Lists additional units to install or remove when the unit is installed or removed.

`DefaultInstance`
The `DefaultInstance` option applies to template unit files only.
Template unit files enable the creation of multiple units from a single configuration file. The `DefaultInstance` option specifies the instance for which the unit is enabled if the template is enabled without any explicitly set instance.

Creating a User-Based systemd Service

In addition to the system-wide `systemd` files, `systemd` enables you to create user-based services that you can run from a user level without requiring root access and privileges. These user-based services are under user control and are configurable independent of system services.

The following are some distinguishing features of user-based `systemd` services:

- User-based `systemd` services are linked with a specific user account.
- They're created under the associated user's home directory in `$HOME/.config/systemd/user/`.
- After these services are enabled, they start when the associated user logs in. This behavior differs from that of enabled `systemd` services which start when the system boots.

This feature is useful when creating podman container services. For more information about podman, see Oracle Linux: Podman User's Guide.

To create a user based service:

1. Create the service's unit file in the `~/.config/systemd/user` directory, for example:

   ```
   touch ~/.config/systemd/user/myservice.service
   ```

2. Open the unit file and specify the values to the options you want to use, such as `Description`, `ExecStart`, `WantedBy`, and so on.

   For reference, see Configurable Options in Service Unit Files and the `systemd.service(5)` and `systemd.unit(5)` manual pages.

3. Enable the service to start automatically when you log in.

   ```
   sudo systemctl --user enable myservice.service
   ```

   Note:

   When you log out, the service is stopped unless the root user has enabled processes to continue to run for the user.

   See Use systemd on Oracle Linux for more information.

4. Start the service.

   ```
   sudo systemctl --user start myservice.service
   ```

5. Verify that the service is running.

   ```
   sudo systemctl --user status myservice.service
   ```

Using Timer Units to Control Service Unit Runtime

Timer units can be configured to control when service units run. You can use timer units instead of configuring the `cron` daemon for time-based events. Timer units can be more complicated to configure than creating a crontab entry. However, timer units are more configurable and the services that they control can be configured for better logging and deeper integration with `systemd` architecture.

Timer units are started, enabled, and stopped similarly to service units. For example, to enable and start a timer unit immediately, type:

```
sudo systemctl enable --now myscript.timer
```

To list all existing timers on the system, to see when they last ran, and when they're next configured to run, type:

```
systemctl list-timers
```

For more information about system timers, see the `systemd.timer(5)` and `systemd.time(7)` manual pages.

Configuring a Realtime Timer Unit

Realtime timers activate on a calendar event, similar to events in a crontab. The option `OnCalendar` specifies when the timer runs a service.

- If needed, create a `.service` file that defines the service to be triggered by the timer unit. In the following procedure, the sample service is `/etc/systemd/system/update.service` which is a service unit that runs an update script.

  For more information about creating service units, see Creating a User-Based systemd Service.
- Decide the time and frequency for running the service. In this procedure, the timer is configured to run the service every 2 hours from Monday to Friday.

This task shows you how to create a system timer to trigger a service to run based on a calendar event. The definition of the calendar event is similar to entries that you put in a cron job.

1. Create the `/etc/systemd/system/update.timer` with the following content:

   ```
   [Unit]
   Description="Run the update.service every two hours from Mon to Fri."

   [Timer]
   OnCalendar=Mon..Fri 00/2
   Unit=update.service

   [Install]
   WantedBy=multi-user.target
   ```

Defining `OnCalendar` can vary from a simple wetting such as `OnCalendar=weekly` definitions that are more detailed. However, the format for defining settings is constant, as follows:

```
DayofWeek Year-Month-Day Hour:Minute:Second
```

The following definition means "the first 4 days of each month at 12:00 o'clock noon, but only if that day is either a Monday or a Tuesday":

```
OnCalendar=Mon,Tue *-*-01..04 12:00:00
```

For other ways to define `OnCalendar` and for more timer options that you can configure in the system timer file, see the `systemd.timer(5)` and `systemd.time(7)` manual pages.

2. Check that all the files related to this timer are configured correctly.

   ```
   systemd-analyze verify /etc/systemd/system/update.*
   ```

   Any detected errors are reported on the screen.

3. Start the timer.

   ```
   sudo systemctl start update.timer
   ```

   This command starts the timer for the current session only.

4. Ensure that the timer starts when the system is booted.

   ```
   sudo systemctl enable update.timer
   ```

Configuring a Monotonic Timer Unit

Monotonic timers that activate after a time span relative to a varying starting point, such as a boot event, or when a particular `systemd` unit becomes active. These timer units stop if the computer is temporarily suspended or shut down. Monotonic timers are configured by using the `On`*`Type`*`Sec` option, where *Type* is the name of the event to which the timer is related. Common monotonic timers include `OnBootSec` and `OnUnitActiveSec`.

- If needed, create a `.service` file that defines the service to be triggered by the timer unit. In the following procedure, the sample service is `/etc/systemd/system/update.service` which is a service unit that runs an update script.

  For more information about creating service units, see Creating a User-Based systemd Service.

- Decide the time and frequency for running the service. In this procedure, the timer is configured to run the service 10 minutes after a system boot, and every 2 hours from when the service is last activated.

This task shows you how to create a system timer to trigger a service to run at specific events, which are when the system boots or after 2 hours have lapsed from the timer's activation.

1. Create the `/etc/systemd/system/update.timer` with the following content:

   ```
   [Unit]
   Description="Run the update.service every two hours from Mon to Fri."

   [Timer]
   OnBootSec=10min
   OnUnitActiveSec=2h
   Unit=update.service

   [Install]
   WantedBy=multi-user.target
   ```

   For more timer options that you can configure in the system timer, see the `systemd.timer(5)` and `systemd.time(7)` manual pages.
2. Check that all the files related to this timer are configured correctly.

   ```
   systemd-analyze verify /etc/systemd/system/update.*
   ```

   Any detected errors are reported on the screen.
3. Start the timer.

   ```
   sudo systemctl start update.timer
   ```

   This command starts the timer for the current session only.
4. Ensure that the timer starts when the system is booted.

   ```
   sudo systemctl enable update.timer
   ```

Running a Transient Timer Unit

Transient timers are temporary timers that are valid only for the current session. These timers can be created to run a program or script directly without requiring service or timer units to be configured within `systemd`. These units are generated by using the `systemd-run` command. See the `systemd-run(1)` manual page for more information.

The parameter options that you would add to the `*unit-file*.timer` file also serve as arguments when you use `systemd-run` command to run a transient timer unit.

The following examples show how to use `systemd-run` to activate transient timers.

- Run `update.service` after 2 hours have elapsed.

  ```
  sudo systemd-run --on-active="2h" --unit update.service
  ```

- Create `~/tmp/myfile` after 1 hour.

  ```
  sudo systemd-run --on-active="1h" /bin/touch ~/tmp/myfile
  ```

- Run `~/myscripts/update.sh` 5 minutes after the service manager is started. Use this syntax to run a service after the service manager has started at user login.

  ```
  sudo systemd-run --on-startup="5m" ~/myscripts/update.sh
  ```

- Run `myjob.service` 10 minutes after system boot.

  ```
  sudo systemd-run --on-boot="10m" --unit myjob.service
  ```

- Run `report.service` at the end of the day.

  ```
  sudo systemd-run --on-calendar="17:00:00"
  ```

3 Configuring System Settings

This chapter describes the files and virtual file systems that you can use to change the configuration settings for the system.

Also see Configure System Settings 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 files that control the system's configuration. The contents of this directory depend on the packages that you have installed on the system.

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.
For more information, see Oracle Linux 9: Managing Local File Systems.

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

For more information, see /usr/share/doc/initscripts*/sysconfig.txt.

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 browsable 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 nonhuman-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)
...
```

Virtual Files and Directories Under /proc

The following table describes the most useful virtual files and directories under the `/proc` directory hierarchy.

Table 3-1 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.
`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.

Table 3-1 (Cont.) Useful Virtual Files and Directories Under the /proc Directory

Virtual File or Directory	Description
`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.
`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.

Table 3-1 (Cont.) Useful Virtual Files and Directories Under the /proc Directory

Virtual File or Directory	Description
`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.
`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. **cpu*N*** Times for CPU *N*.
`swaps`	Provides information about swap devices. The units of size and usage are in kilobytes.

Table 3-1 (Cont.) Useful Virtual Files and Directories Under the /proc Directory

Virtual File or Directory	Description
`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 Modifying Kernel Parameters. 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.

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

Modifying Kernel Parameters

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

```
echo www.mydomain.com > /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 > /proc/sys/net/ipv4/ip_forward
cat /proc/sys/net/ipv4/ip_forward


1
```

You can use the `sysctl` command to view or modify 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.

To display the current kernel settings, use the following command:

```
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`.

To display an individual setting, specify its name as the argument to `sysctl`:

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

To change the value of a setting, use the following command format:

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

Changes that you make in this way remain in force only until the system is rebooted. To make configuration changes persist after the system is rebooted, you must 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' > /etc/sysctl.d/ip_forward.conf
grep -r ip_forward /etc/sysctl.d


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


sysctl net.ipv4.ip_forward


net.ipv4.ip_forward = 0


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 ...


sysctl net.ipv4.ip_forward


net.ipv4.ip_forward = 1
```

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

Parameters That Control System Performance

The following parameters control various aspects of system performance:

`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.

`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 significantly 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`.

`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. Do not adjust this value on server systems.

Parameters That Control Kernel Panics

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

`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.

`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 bcomes 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 usually indicates 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.

kernel.unknown_nmi_panic
If set to `1`, the system panics if the kernel detects an undefined NMI. You would usually 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.

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.

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.

Virtual Directories Under the /sys Directory

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

Table 3-2 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 symlinks named *major*:*minor*. These symlinks 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 Managing System Devices.
`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 About Module Parameters.
`power`	Contains attributes that control the system's power state.

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

Configuring System Language (Locale) and Keyboard Settings

System-wide preferences for language and keyboard are stored in the locale configuration file (`/etc/locale.conf`). You can query and change these settings as needed using `localectl` command. Note that the `systemd` process reads the locale configuration file at boot and applies these settings to every system-wide service, user interface, and user profile, unless they're overridden by other programs or users. For more information about configuring these system-wide settings, see:

- Changing the Language Setting
- Changing the Keyboard Layout

Note:

System-wide preferences for language and keyboard are also configurable during installation. For details on how to configure these settings at installation, see Oracle Linux 9: Installing Oracle Linux.

Changing the Language Setting

The system locale language setting defines the language in which text appears in the Linux user interfaces (text-based and graphical).

To query and change the language setting on the system, follow these steps:

1. To check the current language locale set on the system, type:

   ```
   localectl status
   ```

   For example, the following system language locale output indicates: English (en) as the language, US as the country code, and UTF-8 as the codeset.

   ```
    System Locale: LANG=en_US.UTF-8
   ```

2. To list all possible language locales available on the system, type:

   ```
   localectl list-locales
   ```

   To search the output for a specific language locale, use the `grep` command. For example, to list all possible English locales available for configuration, type:

   ```
   localectl list-locales | grep en
   ```

3. To list all language packs already installed on your system and all language packs available on the `ol8_appstream` repository, type:

   ```
   sudo dnf list langpacks-*
   ```

   For example, the following shows that this system has Spanish, French, Japanese, and Russian language packs installed followed by a truncated list of language packs available on `ol9_appstream`.

   ```
   sudo dnf list langpacks-*
   Last metadata expiration check: 0:00:35 ago on Wed 08 May 2024 04:04:39 PM
   GMT.
   ```

```
Installed Packages
langpacks-core-en.noarch              3.0-16.el9
@ol9_appstream
langpacks-core-font-en.noarch         3.0-16.el9
@ol9_appstream
langpacks-en.noarch                   3.0-16.el9
@ol9_appstream
Available Packages
langpacks-af.noarch                   3.0-16.el9
ol9_appstream
langpacks-am.noarch                   3.0-16.el9
ol9_appstream
langpacks-ar.noarch                   3.0-16.el9
ol9_appstream
langpacks-as.noarch                   3.0-16.el9
ol9_appstream
    ...
```

4. Use dnf to install a language pack. For example, the following installs the Japanese language pack:

   ```
   sudo dnf install langpacks-ja.noarch
   ```

5. To set the default language locale on the system, type:

   ```
   sudo localectl set-locale LANG=locale_name
   ```

   Where:

   - *locale_name* is replaced with the name retrieved earlier from the `list-locales` output.

   For example, to set British English as the system language locale, type:

   ```
   sudo localectl set-locale LANG=en_GB.utf8
   ```

   Note:

   Locale options are typically listed in the following format: `LANGUAGE_COUNTRY.CODESET[@MODIFIERS]`. The `LANGUAGE` is an ISO 639 language code, for example, `en` for English and `COUNTRY` is an ISO 3166 country code. The two letter country code in this example is `GB` for Great Britain and the United Kingdom. The `CODESET` is the character set or encoding, for example, `utf-8`.

   For more information on how to configure language locale options on the system, see the `locale` manual page.

Installing Language Locales Individually

To reduce storage space required for languages, you can choose to install individual glibc locale langpack packages (glibc-langpack-<locale_code>).

To list all installed and all available glibc Langpack packages, run the following command:

```
sudo dnf list glibc-langpack*
```

To install a language pack, run the following command:

```
sudo dnf install glibc-langpack-language_code
```

In the previous command, *language_code* is the language code you want to install. For example, the following example installs Japanese.

```
sudo dnf install glibc-langpack-ja.x86_64
```

Changing the Keyboard Layout

The keyboard layout settings enable you to specify a keymap locale for the Linux user interfaces (text-based and graphical).

To query and change the keyboard layout settings on the system, follow these steps:

1. To check the current keyboard layout configuration on the system, type:

   ```
   localectl status
   ```

   For example, the following keyboard layout output indicates a US country code for the virtual console keymap and a US country code for the X11 layout.

   ```
    System Locale: LANG=en_US.UTF-8
          VC Keymap: us
         X11 Layout: us
   ```

2. To list all possible keyboard layout configurations available, type:

   ```
   localectl list-keymaps
   ```

   To search the output for a specific keymap name, use the `grep` command. For example, to list British compatible keyboard layouts, type:

   ```
   localectl list-keymaps | grep gb
   ```

3. To set the default keyboard layout on the system, type:

   ```
   sudo localectl set-keymap keymap_name
   ```

   Where:

   - *keymap_name* is replaced with the name of the keymap retrieved earlier from the `list-keymaps` output.

   Note that the keymap name change applies to both the virtual console and the x11 layout settings. If you want the X11 layout to differ from the virtual console keymap, use the `--no-convert` option, for example:

   ```
   sudo localectl --no-convert set-x11-keymap keymap_name
   ```

   The *no-convert* option retains the previous x11 keyboard layout setting.

   For more information on how to use the `localectl` command line utility to change keyboard system settings, see the`localectl` manual page.

Configuring System Date and Time Settings

System time is based on the POSIX time standard, where time is measured as the number of seconds that have elapsed from 00:00:00 Coordinated Universal Time (UTC), Thursday, January 1, 1970. A day is defined as 86400 seconds and leap seconds are subtracted automatically.

Date and time representation on a system can be set to match a specific timezone. To list the available timezones, run:

```
timedatectl list-timezones
```

To set the system timezone to match a value returned from the available timezones, you can run:

```
timedatectl set-timezone America/Los_Angeles
```

Substitute *America/Los_Angeles* with a valid timezone entry.

This command sets a symbolic link from `/etc/localtime` to point to the appropriate zone information file in `/usr/share/zoneinfo/`. The setting takes effect immediately. Some long running processes that use `/etc/localtime` to detect the current system timezone might not detect a change in system timezone until the process is restarted.

Note that timezones are largely used for display purposes or to handle user input. Changing timezone doesn't change the time for the system clock. You can change the presentation for system time in any console by setting the `TZ` environment variable. For example, to see the current time in Tokyo, you can run:

```
TZ="Asia/Tokyo" date
```

You can check the system's current date and time configuration by running the `timedatectl` command on its own:

```
timedatectl


               Local time: Wed 2021-07-17 00:50:58
EDT


           Universal time: Wed 2021-07-17 04:50:58
UTC


                 RTC time: Wed 2021-07-17
04:50:55


                Time zone: America/New_York (EDT,
-0400)


System clock synchronized:
yes


              NTP service:
active
```

```
          RTC in local TZ: no
```

To set system time manually, use the `timedatectl set-time` command:

```
timedatectl set-time "2021-07-17 01:59:59"
```

This command sets the current system time based on the time specified assuming the currently set system timezone. The command also updates the system Real Time Clock (RTC).

Tip:

See Learn How to Localize Your Installation on Oracle Linux for a hands-on tutorial that describes how to use tools to configure system parameters such as date, time, and locale.

Consider configuring the system to use network time synchronization for more accurate time-keeping. Using network time synchronization is important especially when setting up high-availability or when using network-based file systems.

For more information about configuring the network time services that use NTP, see Oracle Linux 9: Setting Up Networking.

Tip:

See Configure Chrony on Oracle Linux for a hands-on tutorial on setting up and configuring the `chronyd` service.

If you configure an NTP service, enable NTP by running the following command:

```
timedatectl set-ntp true
```

This command enables and starts the `chronyd` service, if available.

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.

4 Managing System Devices

This chapter describes how the system uses device files and how the Udev device manager dynamically creates or removes device node files.

About Device Files

The `/dev` directory contains *device files* or *device nodes* that provide access to peripheral devices such as hard disks, to resources on peripheral devices such as disk partitions, and pseudo devices such as a random number generator.

The `/dev` directory has several subdirectory hierarchies, each of which holds device files that relate to a certain type of device. However, the contents of these subdirectories are implemented as symbolic links to corresponding files in `/dev`. Thus, the files can be accessed either through the linked file in `/dev` or the corresponding file in the subdirectory.

Using the `ls -l /dev` command lists files, some of which are flagged as being either type `b` (for *block*) or type `c` (for *character*). These devices have an associated pair of numbers that identify the device to the system.

```
ls -l /dev
```

```
total 0
crw-r--r--. 1 root root     10, 235 Aug 20 08:36 autofs
drwxr-xr-x. 2 root root         240 Sep 20 07:37 block
drwxr-xr-x. 2 root root         100 Aug 20 08:36 bsg
drwxr-xr-x. 3 root root          60 Nov  4  2019 bus
lrwxrwxrwx. 1 root root           3 Aug 20 08:36 cdrom -> sr0
drwxr-xr-x. 2 root root        2720 Sep 20 07:37 char
crw-------. 1 root root      5,   1 Aug 20 08:36 console
lrwxrwxrwx. 1 root root          11 Aug 20 08:36 core -> /proc/kcore
drwxr-xr-x. 3 root root          60 Nov  4  2019 cpu
crw-------. 1 root root     10,  62 Aug 20 08:36 cpu_dma_latency
drwxr-xr-x. 7 root root         140 Aug 20 08:36 disk
brw-rw----. 1 root disk    253,   0 Aug 20 08:36 dm-0
brw-rw----. 1 root disk    253,   1 Aug 20 08:36 dm-1
brw-rw----. 1 root disk    253,   2 Aug 20 08:36 dm-2
lrwxrwxrwx. 1 root root          13 Aug 20 08:36 fd -> /proc/self/fd
crw-rw-rw-. 1 root root      1,   7 Aug 20 08:36 full
crw-rw-rw-. 1 root root     10, 229 Aug 20 08:36 fuse
crw-------. 1 root root     10, 228 Aug 20 08:36 hpet
drwxr-xr-x. 2 root root           0 Aug 20 08:36 hugepages
crw-------. 1 root root     10, 183 Aug 20 08:36 hwrng
lrwxrwxrwx. 1 root root          12 Aug 20 08:36 initctl -> /run/initctl
drwxr-xr-x. 3 root root         220 Aug 20 08:36 input
crw-r--r--. 1 root root      1,  11 Aug 20 08:36 kmsg
lrwxrwxrwx. 1 root root          28 Aug 20 08:36 log -> /run/systemd/journal/
dev-log
brw-rw----. 1 root disk      7,   0 Sep 23 01:28 loop0
```

```
crw-rw----. 1 root disk     10, 237 Sep 20 07:37 loop-control
drwxr-xr-x. 2 root root         120 Aug 20 08:36 mapper
crw-------. 1 root root     10, 227 Aug 20 08:36 mcelog
crw-r-----. 1 root kmem      1,   1 Aug 20 08:36 mem
crw-------. 1 root root     10,  59 Aug 20 08:36 memory_bandwidth
drwxrwxrwt. 2 root root          40 Nov  4  2019 mqueue
drwxr-xr-x. 2 root root          60 Aug 20 08:36 net
crw-------. 1 root root     10,  61 Aug 20 08:36 network_latency
crw-------. 1 root root     10,  60 Aug 20 08:36 network_throughput
crw-rw-rw-. 1 root root      1,   3 Aug 20 08:36 null
crw-------. 1 root root     10, 144 Aug 20 08:36 nvram
drwxr-xr-x. 2 root root         100 Aug 20 08:36 ol_ca-virtdoc-oltest1
crw-r-----. 1 root kmem      1,   4 Aug 20 08:36 port
crw-------. 1 root root    108,   0 Aug 20 08:36 ppp
crw-rw-rw-. 1 root tty       5,   2 Oct  7 08:10 ptmx
drwxr-xr-x. 2 root root           0 Aug 20 08:36 pts
crw-rw-rw-. 1 root root      1,   8 Aug 20 08:36 random
drwxr-xr-x. 2 root root          60 Nov  4  2019 raw
lrwxrwxrwx. 1 root root           4 Aug 20 08:36 rtc -> rtc0
crw-------. 1 root root    251,   0 Aug 20 08:36 rtc0
brw-rw----. 1 root disk      8,   0 Aug 20 08:36 sda
brw-rw----. 1 root disk      8,   1 Aug 20 08:36 sda1
brw-rw----. 1 root disk      8,   2 Aug 20 08:36 sda2
brw-rw----. 1 root disk      8,  16 Aug 20 08:36 sdb
brw-rw----. 1 root disk      8,  17 Aug 20 08:36 sdb1
crw-rw----. 1 root cdrom    21,   0 Aug 20 08:36 sg0
```

Block devices support random access to data, seeking media for data, and typically buffers data while data is being written or read. Examples of block devices include hard disks, CD-ROM drives, flash memory, and other addressable memory devices.

Character devices support the streaming of data to or from a device. The data isn't typically buffered nor is random access granted to data on a device. The kernel writes data to or reads data from a character device 1 byte at a time. Examples of character devices include keyboards, mice, terminals, pseudo terminals, and tape drives. `tty0` and `tty1` are character device files that correspond to terminal devices so users can log in from serial terminals or terminal emulators.

Pseudo terminals secondary devices emulate real terminal devices to interact with software. For example, a user might log in to a terminal device such as `/dev/tty1`, which then uses the pseudo terminal primary device, `/dev/pts/ptmx`, to interact with an underlying pseudo terminal device. The character device files for pseudo terminal secondary and primary devices are located in the `/dev/pts` directory, as shown in the following example:

```
ls -l /dev/pts


total 0
crw--w----. 1 guest tty  136, 0 Mar 17 10:11 0
crw--w----. 1 guest tty  136, 1 Mar 17 10:53 1
crw--w----. 1 guest tty  136, 2 Mar 17 10:11 2
c---------. 1 root  root   5, 2 Mar 17 08:16 ptmx
```

Some device entries, such as `stdin` for the standard input, are symbolically linked through the `self` subdirectory of the `proc` file system. The pseudo-terminal device file to which they actually point depends on the context of the process.

```
ls -l /proc/self/fd/[012]
```

```
lrwx------. 1 root root 64 Oct  7 08:23 /proc/self/fd/0 -> /dev/pts/0
lrwx------. 1 root root 64 Oct  7 08:23 /proc/self/fd/1 -> /dev/pts/0
lrwx------. 1 root root 64 Oct  7 08:23 /proc/self/fd/2 -> /dev/pts/0
```

Character devices, such as `null`, `random`, `urandom`, and `zero` are examples of pseudo devices that provide access to virtual functionality implemented in software rather than to physical hardware.

`/dev/null` is a data sink. Data that you write to `/dev/null` effectively disappears but the write operation succeeds. Reading from `/dev/null` returns `EOF` (end-of-file).

`/dev/zero` is a data source of an unlimited number of 0-value bytes.

`/dev/random` and `/dev/urandom` are data sources of streams of pseudo random bytes. To maintain high-entropy output, `/dev/random` blocks if its entropy pool doesn't contain sufficient bits of noise. `/dev/urandom` doesn't block and, thereforem, the entropy of its output might not be as consistently high as that of `/dev/random`. However, neither `/dev/random` nor `/dev/urandom` are considered to be truly random enough for the purposes of secure cryptography such as military-grade encryption.

You can find out the size of the entropy pool and the entropy value for `/dev/random` from virtual files under `/proc/sys/kernel/random`:

```
cat /proc/sys/kernel/random/poolsize
```

```
4096
```

```
cat /proc/sys/kernel/random/entropy_avail
```

```
3467
```

For more information, see the `null(4)`, `pts(4)`, and `random(4)` manual pages.

About the Udev Device Manager

The Udev device manager dynamically creates or removes device node files at boot time . When creating a device node, `udev` reads the device’s `/sys` directory for attributes such as the label, serial number, and bus device number.

Udev can use persistent device names to guarantee consistent naming of devices across reboots, regardless of their order of discovery. Persistent device names are especially important when using external storage devices.

The configuration file for udev is /etc/udev/udev.conf, in which you can define the udev_log logging priority, which can be set to err, info and debug. Note that the default value is err.

For more information, see the udev(7) manual page.

About Udev Rules

Udev uses rules files to determine how to identify devices and create device names. The udev service (systemd-udevd) reads the rules files at system start-up and stores the rules in memory. If the kernel discovers a new device or an existing device goes offline, the kernel sends an event action (*uevent*) notification to udev, which matches the in-memory rules against the device attributes in the /sys directory to identify the device.

Multiple rules files exist in different directories. However, you only need to know about /etc/udev/rules.d/*.rules files because these are the only rules files that you can modify. See Modifying Udev Rules.

Udev processes the rules files in lexical order, regardless of the directory of the rule files. Rules files in /etc/udev/rules.d override rules files of the same name in other locations.

The following rules are extracted from the file /lib/udev/rules.d/50-udev-default.rules and illustrate the syntax of udev rules:

```
# do not edit this file, it will be overwritten on update

SUBSYSTEM=="block", SYMLINK{unique}+="block/%M:%m"
SUBSYSTEM!="block", SYMLINK{unique}+="char/%M:%m"

KERNEL=="pty[pqrstuvwxyzabcdef][0123456789abcdef]", GROUP="tty", MODE="0660"
KERNEL=="tty[pqrstuvwxyzabcdef][0123456789abcdef]", GROUP="tty", MODE="0660"
...
# mem
KERNEL=="null|zero|full|random|urandom", MODE="0666"
KERNEL=="mem|kmem|port|nvram",  GROUP="kmem", MODE="0640"
...
# block
SUBSYSTEM=="block", GROUP="disk"
...
# network
KERNEL=="tun",                  MODE="0666"
KERNEL=="rfkill",               MODE="0644"

# CPU
KERNEL=="cpu[0-9]*",            MODE="0444"
...
# do not delete static device nodes
ACTION=="remove", NAME=="", TEST=="/lib/udev/devices/%k", \
    OPTIONS+="ignore_remove"
ACTION=="remove", NAME=="?*", TEST=="/lib/udev/devices/$name", \
    OPTIONS+="ignore_remove"
```

A rule either assigns a value to a key or it tries to find a match for a key by comparing its current value with the specified value. The following table shows the assignment and comparison operators that you can use.

Operator	Description
`=`	Assign a value to a key, overwriting any previous value.
`+=`	Assign a value by appending it to the key's current list of values.
`:=`	Assign a value to a key. This value cannot be changed by any further rules.
`==`	Match the key's current value against the specified value for equality.
`!=`	Match the key's current value against the specified value for equality.

You can use the following shell-style pattern-matching characters in values.

Character	Description
`?`	Matches a single character.
`*`	Matches any number of characters, including zero.
`[]`	Matches any single character or character from a range of characters specified within the brackets. For example, `tty[sS][0-9]` would match `ttys7` or `ttyS7`.

The following table describes commonly used match keys in rules.

Match Key	Description
`ACTION`	Matches the name of the action that led to an event. For example, `ACTION="add"` or `ACTION="remove"`.
`ENV{`*`key`*`}`	Matches a value for the device property *key*. For example, `ENV{DEVTYPE}=="disk"`.
`KERNEL`	Matches the name of the device that is affected by an event. For example, `KERNEL=="dm-*"` for disk media.
`NAME`	Matches the name of a device file or network interface. For example, `NAME="?*"` for any name that consists of one or more characters.
`SUBSYSTEM`	Matches the subsystem of the device that is affected by an event. For example, `SUBSYSTEM=="tty"`.
`TEST`	Tests wheter the specified file or path exists; for example, `TEST=="/lib/udev/devices/$name"`, where `$name` is the name of the currently matched device file.

Other match keys include `ATTR{`*`filename`*`}`, `ATTRS{`*`filename`*`}`, `DEVPATH`, `DRIVER`, `DRIVERS`, `KERNELS`, `PROGRAM`, `RESULT`, `SUBSYSTEMS`, and `SYMLINK`.

The following table describes commonly used assignment keys in rules.

Assignment Key	Description
`ENV{`*key*`}`	Specifies a value for the device property *key*, such as `GROUP="disk"`.
`GROUP`	Specifies the group for a device file, such as `GROUP="disk"`.
`IMPORT{`*type*`}`	Specifies a set of variables for the device property, depending on *type*: **`cmdline`** Import a single property from the boot `kernel` command line. For simple flags, `udev` sets the value of the property to 1. For example, `IMPORT{cmdline}="nodmraid"`. **`db`** Interpret the specified value as an index into the device database and import a single property, which must have already been set by an earlier event. For example, `IMPORT{db}="DM_UDEV_LOW_PRIORITY_FLAG"`. **`file`** Interpret the specified value as the name of a text file and import its contents, which must be in environmental key format. For example, `IMPORT{file}="keyfile"`. **`parent`** Interpret the specified value as a key-name filter and import the stored keys from the database entry for the parent device. For example `IMPORT{parent}="ID_*"`. **`program`** Run the specified value as an external program and imports its result, which must be in environmental key format. For example `IMPORT{program}="usb_id --export %p"`.
`MODE`	Specifies the permissions for a device file, such as `MODE="0640"`.
`NAME`	Specifies the name of a device file, such as `NAME="em1"`.
`OPTIONS`	Specifies rule and device options, such as `OPTIONS+="ignore_remove"`, which means that the device file isn't removed if the device is removed.
`OWNER`	Specifies the owner for a device file, such as `GROUP="root"`.
`RUN`	Specifies a command to be run after the device file has been created, such as `RUN+="/usr/bin/eject $kernel"`, where `$kernel` is the kernel name of the device.

Assignment Key	Description
`SYMLINK`	Specifies the name of a symbolic link to a device file, such as `SYMLINK+="disk/by-uuid/$env{ID_FS_UUID_ENC}"`, where `$env{}` is substituted with the specified device property.

Other assignment keys include `ATTR{key}`, `GOTO`, `LABEL`, `RUN`, and `WAIT_FOR`.

The following table describes the string substitutions that are commonly used with the `GROUP`, `MODE`, `NAME`, `OWNER`, `PROGRAM`, `RUN`, and `SYMLINK` keys.

String Substitution	Description
`$attr{file}` or `%s{file}`	Specifies the value of a device attribute from a file under `/sys`, such as `ENV{MATCHADDR}="$attr{address}"`.
`$devpath` or `%p`	The device path of the device in the `sysfs` file system under `/sys`, such as `RUN+="keyboard-force-release.sh $devpath common-volume-keys"`.
`$env{key}` or `%E{key}`	Specifies the value of a device property, such as `SYMLINK+="disk/by-id/md-name-$env{MD_NAME}-part%n"`.
`$kernel` or `%k`	Specifies the kernel name for the device.
`$major` or `%M`	Specifies the major number of a device, such as `IMPORT{program}="udisks-dm-export %M %m"`.
`$minor` or `%m`	Specifies the minor number of a device, such as `RUN+="$env{LVM_SBIN_PATH}/lvm pvscan --cache --major $major --minor $minor"`.
`$name`	Specifies the device file of the current device, such as `TEST=="/lib/udev/devices/$name"`.

Udev expands the strings specified for `RUN` immediately before its program is run, which is after udev has finished processing all other rules for the device. For the other keys, `udev` expands the strings while it's processing the rules.

For more information, see the `udev(7)` manual page.

Querying Udev and Sysfs

You can use the `udevadm` command to query the `udev` database and `sysfs`.

To query the `sysfs` device path relative to `/sys` that corresponds to the device file `/dev/sda`:

```
udevadm info --query=path --name=/dev/sda


/devices/pci0000:00/0000:00:0d.0/host0/target0:0:0/0:0:0:0/block/sda
```

To query the symbolic links that point to `/dev/sda`, use the following command:

```
udevadm info --query=symlink --name=/dev/sda


block/8:0
disk/by-id/ata-VBOX_HARDDISK_VB6ad0115d-356e4c09
disk/by-id/scsi-SATA_VBOX_HARDDISK_VB6ad0115d-356e4c09
disk/by-path/pci-0000:00:0d.0-scsi-0:0:0:0
```

To query the properties of `/dev/sda`, use the following command:

```
udevadm info --query=property --name=/dev/sda


UDEV_LOG=3
DEVPATH=/devices/pci0000:00/0000:00:0d.0/host0/target0:0:0/0:0:0:0/block/sda
MAJOR=8
MINOR=0
DEVNAME=/dev/sda
DEVTYPE=disk
SUBSYSTEM=block
ID_ATA=1
ID_TYPE=disk
ID_BUS=ata
ID_MODEL=VBOX_HARDDISK
ID_MODEL_ENC=VBOX\x20HARDDISK\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20.
..
ID_REVISION=1.0
ID_SERIAL=VBOX_HARDDISK_VB579a85b0-bf6debae
ID_SERIAL_SHORT=VB579a85b0-bf6debae
ID_ATA_WRITE_CACHE=1
ID_ATA_WRITE_CACHE_ENABLED=1
ID_ATA_FEATURE_SET_PM=1
ID_ATA_FEATURE_SET_PM_ENABLED=1
ID_ATA_SATA=1
ID_ATA_SATA_SIGNAL_RATE_GEN2=1
ID_SCSI_COMPAT=SATA_VBOX_HARDDISK_VB579a85b0-bf6debae
ID_PATH=pci-0000:00:0d.0-scsi-0:0:0:0
ID_PART_TABLE_TYPE=dos
LVM_SBIN_PATH=/sbin
UDISKS_PRESENTATION_NOPOLICY=0
UDISKS_PARTITION_TABLE=1
UDISKS_PARTITION_TABLE_SCHEME=mbr
UDISKS_PARTITION_TABLE_COUNT=2
UDISKS_ATA_SMART_IS_AVAILABLE=0
```

```
DEVLINKS=/dev/block/8:0 /dev/disk/by-id/ata-VBOX_HARDDISK_VB579a85b0-
bf6debae ...
```

To query the entire information for `/dev/sda`, use the following command:

```
udevadm info --query=all --name=/dev/sda


P: /devices/pci0000:00/0000:00:0d.0/host0/target0:0:0/0:0:0:0/block/sda
N: sda
W: 37
S: block/8:0
S: disk/by-id/ata-VBOX_HARDDISK_VB579a85b0-bf6debae
S: disk/by-id/scsi-SATA_VBOX_HARDDISK_VB579a85b0-bf6debae
S: disk/by-path/pci-0000:00:0d.0-scsi-0:0:0:0
E: UDEV_LOG=3
E: DEVPATH=/devices/pci0000:00/0000:00:0d.0/host0/target0:0:0/0:0:0:0/
block/sda
E: MAJOR=8
E: MINOR=0
E: DEVNAME=/dev/sda
E: DEVTYPE=disk
E: SUBSYSTEM=block
E: ID_ATA=1
E: ID_TYPE=disk
E: ID_BUS=ata
E: ID_MODEL=VBOX_HARDDISK
E:
ID_MODEL_ENC=VBOX\x20HARDDISK\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20.
..
E: ID_SERIAL=VBOX_HARDDISK_VB579a85b0-bf6debae
E: ID_SERIAL_SHORT=VB579a85b0-bf6debae
E: ID_ATA_WRITE_CACHE=1
E: ID_ATA_WRITE_CACHE_ENABLED=1
E: ID_ATA_FEATURE_SET_PM=1
E: ID_ATA_FEATURE_SET_PM_ENABLED=1
E: ID_ATA_SATA=1
E: ID_ATA_SATA_SIGNAL_RATE_GEN2=1
E: ID_SCSI_COMPAT=SATA_VBOX_HARDDISK_VB579a85b0-bf6debae
E: ID_PATH=pci-0000:00:0d.0-scsi-0:0:0:0
E: ID_PART_TABLE_TYPE=dos
E: LVM_SBIN_PATH=/sbin
E: UDISKS_PRESENTATION_NOPOLICY=0
E: UDISKS_PARTITION_TABLE=1
E: UDISKS_PARTITION_TABLE_SCHEME=mbr
E: UDISKS_PARTITION_TABLE_COUNT=2
E: UDISKS_ATA_SMART_IS_AVAILABLE=0
E: DEVLINKS=/dev/block/8:0 /dev/disk/by-id/ata-VBOX_HARDDISK_VB579a85b0-
bf6debae ...
```

To display all of the properties of `/dev/sda`, as well as the parent devices that `udev` has found in `/sys`, use the following command:

```
udevadm info --attribute-walk --name=/dev/sda
```

```
...
  looking at device '/devices/pci0000:00/0000:00:0d.0/host0/
target0:0:0/0:0:0:0/block/sda':
    KERNEL=="sda"
    SUBSYSTEM=="block"
    DRIVER==""
    ATTR{range}=="16"
    ATTR{ext_range}=="256"
    ATTR{removable}=="0"
    ATTR{ro}=="0"
    ATTR{size}=="83886080"
    ATTR{alignment_offset}=="0"
    ATTR{capability}=="52"
    ATTR{stat}=="   20884    15437  1254282   338919     5743     8644
103994   109005 ...
    ATTR{inflight}=="       0        0"

  looking at parent device '/devices/pci0000:00/0000:00:0d.0/host0/
target0:0:0/0:0:0:0':
    KERNELS=="0:0:0:0"
    SUBSYSTEMS=="scsi"
    DRIVERS=="sd"
    ATTRS{device_blocked}=="0"
    ATTRS{type}=="0"
    ATTRS{scsi_level}=="6"
    ATTRS{vendor}=="ATA     "
    ATTRS{model}=="VBOX HARDDISK   "
    ATTRS{rev}=="1.0 "
    ATTRS{state}=="running"
    ATTRS{timeout}=="30"
    ATTRS{iocounterbits}=="32"
    ATTRS{iorequest_cnt}=="0x6830"
    ATTRS{iodone_cnt}=="0x6826"
    ATTRS{ioerr_cnt}=="0x3"
    ATTRS{modalias}=="scsi:t-0x00"
    ATTRS{evt_media_change}=="0"
    ATTRS{dh_state}=="detached"
    ATTRS{queue_depth}=="31"
    ATTRS{queue_ramp_up_period}=="120000"
    ATTRS{queue_type}=="simple"

  looking at parent device '/devices/pci0000:00/0000:00:0d.0/host0/
target0:0:0':
    KERNELS=="target0:0:0"
    SUBSYSTEMS=="scsi"
    DRIVERS==""

  looking at parent device '/devices/pci0000:00/0000:00:0d.0/host0':
    KERNELS=="host0"
    SUBSYSTEMS=="scsi"
    DRIVERS==""

  looking at parent device '/devices/pci0000:00/0000:00:0d.0':
    KERNELS=="0000:00:0d.0"
    SUBSYSTEMS=="pci"
    DRIVERS=="ahci"
    ATTRS{vendor}=="0x8086"
    ATTRS{device}=="0x2829"
    ATTRS{subsystem_vendor}=="0x0000"
    ATTRS{subsystem_device}=="0x0000"
    ATTRS{class}=="0x010601"
    ATTRS{irq}=="21"

ATTRS{local_cpus}=="00000000,00000000,00000000,00000000,00000000,00000000,0000
0000,00000003"
    ATTRS{local_cpulist}=="0-1"
    ATTRS{modalias}=="pci:v00008086d00002829sv00000000sd00000000bc01sc06i01"
    ATTRS{numa_node}=="-1"
    ATTRS{enable}=="1"
    ATTRS{broken_parity_status}=="0"
    ATTRS{msi_bus}==""
    ATTRS{msi_irqs}==""

  looking at parent device '/devices/pci0000:00':
    KERNELS=="pci0000:00"
    SUBSYSTEMS==""
    DRIVERS==""
```

The command starts at the device that's specified by the device path and walks the chain of parent devices. For every device that the command finds, the command displays the possible attributes for the device and its parent devices by using the match key format for `udev` rules.

For more information, see the `udevadm(8)` manual page.

Modifying Udev Rules

The order in which rules are evaluated is important. Udev processes rules in lexical order. If you want to add custom rules, you need `udev` to locate and evaluate these rules before the default rules.

The following example illustrates how to implement a `udev` rules file that adds a symbolic link to the disk device `/dev/sdb`.

1. Create a rule file under `/etc/udev/rules.d` with a file name such as `10-local.rules` that udev reads before any other rules file.

   The following rule in `10-local.rules` creates the symbolic link `/dev/my_disk`, which points to `/dev/sdb`:

   ```
   KERNEL=="sdb", ACTION=="add", SYMLINK="my_disk"
   ```

Listing the device files in `/dev` shows that `udev` hasn't yet applied the rule:

```
ls /dev/sd* /dev/my_disk
```

```
ls: cannot access /dev/my_disk: No such file or directory
/dev/sda  /dev/sda1  /dev/sda2  /dev/sdb
```

2. To simulate how `udev` applies its rules to create a device, you can use the `udevadm test` command with the device path of `sdb` listed under the `/sys/class/block` hierarchy, for example:

   ```
   udevadm test /sys/class/block/sdb
   ```

   ```
   calling: test
   version ...
   This program is for debugging only, it does not run any program
   specified by a RUN key. It may show incorrect results, because
   some values may be different, or not available at a simulation run.
   ...
   LINK 'my_disk' /etc/udev/rules.d/10-local.rules:1
   ...
   creating link '/dev/my_disk' to '/dev/sdb'
   creating symlink '/dev/my_disk' to 'sdb
   ...
   ACTION=add
   DEVLINKS=/dev/disk/by-id/ata-VBOX_HARDDISK_VB186e4ce2-f80f170d
     /dev/disk/by-uuid/a7dc508d-5bcc-4112-b96e-f40b19e369fe
     /dev/my_disk
   ...
   ```

3. Restart the `systemd-udevd` service:

   ```
   sudo systemctl restart systemd-udevd
   ```

   After `udev` processes the rules files, the symbolic link `/dev/my_disk` has been added:

   ```
   ls -F /dev/sd* /dev/my_disk
   ```

   ```
   /dev/my_disk@  /dev/sda  /dev/sda1  /dev/sda2  /dev/sdb
   ```

4. (Optional) To undo the changes, remove `/etc/udev/rules.d/10-local.rules` and `/dev/my_disk`, then run `systemctl restart systemd-udevd` again.

5

Managing Kernel Modules

This chapter describes how to load, unload, and modify the behavior of kernel modules.

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.

Note:

From UEK R7 onward, kernel packaging changes are applied to provide a more streamlined kernel. 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 `kernel-uek-firmware` package to be installed.

You can view the contents of these packages by running:

```
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 kernel-uek-modules-extra linux-
firmware
```

See Unbreakable Enterprise Kernel Release 7: Release Notes (5.15.0-0.30).

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.

Listing Information About Loaded Modules

The `lsmod` command lists 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.

Show detailed information about a module by using the `modinfo` command:

```
modinfo ahci
```

```
filename:       /lib/modules/5.4.17-2136.306.1.3.el8uek.x86_64/kernel/
drivers/ata/ahci.ko.xz
version:        3.0
license:        GPL
description:    AHCI SATA low-level driver
author:         Jeff Garzik
srcversion:     3F4E4F52FD2D5F8BBD5F972
alias:          pci:v*d*sv*sd*bc01sc06i01*
alias:          pci:v00001C44d00008000sv*sd*bc*sc*i*
...
depends:        libahci,libata
retpoline:      Y
intree:         Y
name:           ahci
vermagic:       5.4.17-2136.306.1.3.el8uek.x86_64 SMP mod_unload modversions
sig_id:         PKCS#7
signer:         Oracle CA Server
sig_key:        22:07:CB:47:59:F3:50:A0:A2:FA:24:CE:B4:00:53:4E:C5:1D:C6:2A
sig_hashalgo:   sha512
signature:      2F:AE:AF:6D:56:92:69:C4:77:AB:E1:3D:41:09:AF:A6:FC:1D:3B:A2:
                9C:23:79:6F:17:82:D5:A3:9B:61:64:32:72:9B:98:C9:8C:89:73:FB:
                A4:86:4F:B5:7D:DF:84:8E:05:26:4F:22:CB:02:41:38:7B:7C:CB:C2:
                ...
                9F:FD:94:8F:35:9B:2A:89:3E:E1:17:40:49:79:30:8B:92:4D:3A:9A:
```

```
                F4:C7:82:8D:26:BE:6D:FB:71:C6:E5:FD
parm:           marvell_enable:Marvell SATA via AHCI (1 = enabled) (int)
parm:           mobile_lpm_policy:Default LPM policy for mobile chipsets (int)

...
```

The output would include 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.

`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.

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/5.4.17-2136.306.1.3.el8uek.x86_64/kernel/drivers/parport/
parport.ko.xz
```

For more information, see the `lsmod(5)` and `modinfo(8)` manual pages.

Loading and Unloading Modules

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 additional modules are loaded to resolve dependencies.

```
sudo modprobe -v nfs
```

```
insmod /lib/modules/4.18.0-80.el8.x86_64/kernel/net/sunrpc/auth_gss/
auth_rpcgss.ko
insmod /lib/modules/4.18.0-80.el8.x86_64/kernel/fs/nfs_common/nfs_acl.ko
insmod /lib/modules/4.18.0-80.el8.x86_64/kernel/fs/fscache/fscache.ko
...
```

Note:

The `modprobe` command does not reload modules that are already loaded. You must first unload a module before you can load it again.

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

```
sudo modprobe -rv nfs
```

```
rmmod /lib/modules/4.18.0-80.el8.x86_64/kernel/fs/nfs/nfs.ko
rmmod /lib/modules/4.18.0-80.el8.x86_64/kernel/fs/lockd/lockd.ko
rmmod /lib/modules/4.18.0-80.el8.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.

For more information, see the `modprobe(8)` and `modules.dep(5)` manual pages.

About Module Parameters

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

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

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

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

Alternatively, 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
```

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.

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.

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`.

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
```

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

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.

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
   ```

> **WARNING:**
>
> Disabling modules can have unintended consequences and can prevent a system from booting properly 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.

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/4.18.0-80.el8.x86_64/weak-updates/kmod-kvdo/uds


lrwxrwxrwx. 1 root root 68 Oct  8 07:57 uds.ko ->
/lib/modules/4.18.0-80.0.0.0.1.el8.x86_64/extra/kmod-kvdo/uds/uds.ko


ls -l /lib/modules/4.18.0-80.el8.x86_64/weak-updates/kmod-kvdo/vdo


lrwxrwxrwx. 1 root root 68 Oct  8 07:57 uds.ko ->
/lib/modules/4.18.0-80.0.0.0.1.el8.x86_64/extra/kmod-kvdo/uds/uds.ko
```

The symbolic link enables the external module to loaded 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, thus provides a more predictable running of a system and its hardware.

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.

To remove weak update modules, use the following command:

```
rm -rf /lib/modules/4.18.0-80.el8.x86_64/weak-updates/kmod-kvdo/
```

Running this command manually removes the symbolic links for each kernel.

Alternatively, you can use the `weak-modules` command, which safely removes the specified weak update module for the compatible kernels or the command removes the weak update modules for the current kernel. You can use the `weak-modules` command similarly to add weak update modules.

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
rm -rf kmod-kvdo
```

For more information about external driver modules and driver update disks, see Oracle Linux 9: Installing Oracle Linux.

6 Configuring Huge Pages

In Oracle Linux, physical memory is managed in fixed-size blocks called pages. In the x86_64 architecture, the default size of each page is 4 KB.

The kernel stores virtual to physical address mappings for the pages in a data structure known as the page table. However, page table lookups are resource-intensive, so the most recently used addresses are cached in the CPU's Translation Lookaside Buffer (TLB) for faster retrieval. When the CPU needs to fulfill a request for an address-mapping, the CPU first searches the TLB cache. A TLB hit describes the CPU finding the address in the TLB cache. A TLB miss describes the CPU unable to find the requested address-mapping in the cache, in which case the system would perform a resource intensive lookup on the page table to retrieve the address information.

The default page size of 4 KB is suitable for most applications. However, for applications that work with large amounts of memory, the number of 4 KB pages required can be large and can lead to a high number of TLB misses and a performance overhead. Oracle Linux provides huge page features so that applications requiring more memory can have their requirements fulfilled with fewer pages.

Available Huge Page Features

The following huge page features are included in Oracle Linux:

HugeTLB Pages

HugeTLB pages are also called static huge pages.

With HugeTLB pages feature, you can reserve pools of huge pages, each of a specified quantity, for each huge page size. The available huge page size options on x86_64 platforms are 2 MB and 1 GB.

For more information about configuring HugeTLB pages, see Configuring HugeTLB Pages

Note:

Best Practices:

- Make requests to the kernel for static huge pages as close to boot time as possible, when the occurrence of memory fragmentation is at a minimum.
- Huge pages can reduce the amount of memory available to the system. Therefore, when requesting a reserved huge page pool, ensure the pool isn't oversized and that the system's access to memory isn't impacted.

Transparent HugePages

The Transparent HugePages (THP) feature is enabled by default in Oracle Linux. With THP, the kernel automatically assigns huge pages to processes. With THP, you can assign only 2 MB pages on x86_64 platforms.

THP can run in the following modes:

- `system-wide` (default): The kernel assigns huge pages to processes that use large contiguous virtual memory areas whenever it's possible to do so.
- `per-process:` The kernel only assigns huge pages to application processes that explicitly request huge pages through the `madvise()` system call.

For more information about configuring THP, see Configuring Transparent HugePages

Configuring HugeTLB Pages

You can configure HugeTLB pages by using the following types of parameters:

- Kernel boot parameters
- File-based configuration parameters

The following sections discuss the parameters in greater detail.

Kernel Boot Parameters for HugeTLB Pages

The kernel boot options enable you to specify values such as the size and the number of pages to be reserved in the kernel's pool. Using the kernel boot parameters is the most reliable method of requesting huge pages.

The following table describes the kernel boot parameters available for HugeTLB page setup.

Table 6-1 The Kernel Boot Command Line Parameters for Requesting HugeTLB Pages

Parameters	Purpose	Accepted Value Option on x86_64 Architecture
`default_hugepagesz`	Defines the default size of persistent huge pages configured in the kernel at boot time.	`2M`(default), `1G`

Table 6-1 (Cont.) The Kernel Boot Command Line Parameters for Requesting HugeTLB Pages

Parameters	Purpose	Accepted Value Option on x86_64 Architecture

`hugepagesz` and `hugepages`	Size parameter `hugepagesz` is used with quantity parameter `hugepages` to reserve a pool of a specified page size and quantity. For example, to request a pool of 1500 pages of size 2 MB, the command line options would be as follows:
	`hugepagesz=2M`
	`hugepages=1500`
	If multiple huge page sizes are supported, the "`hugepagesz=<size> hugepages=<qty>`" pair can be specified multiple times, once for each page size. For example, you can use the following command line options to request one pool of four pages of 1 GB size, and a second pool of 1500 pages of 2 MB size:
	`hugepagesz=1G`
	`hugepages=4`
	`hugepagesz=2M`
	`hugepages=1500`
		`Hugepagesz`: `2M`, `1G`
		`hugepages`: `0` or greater

Note:

In a NUMA system, pages reserved with kernel command line options, as shown in the previous table, are divided equally between the NUMA nodes.
If the requirement is to have a different number of pages on each node, you can use the file-based HugeTLB parameters in the `sysfs` file system. See File-Based Configuration Parameters for HugeTLB Pages and Requesting HugeTLB Pages Using NUMA Node Specific Parameters Early in the Boot Process.

File-Based Configuration Parameters for HugeTLB Pages

The file-based configuration parameters provide runtime access to the configuration settings.

Note:

In addition to accessing the settings at runtime, you can also initialize the parameters early in the boot process, for example, by creating a start-up bash script or by setting the parameters up in a local rc init script.

Multiple instances of each file-based parameter can be configured on a system. For example, on a system that can handle both 2 MB and 1 GB HugeTLB page sizes, several `nr_hugepages` settings can exist. This parameter defines the number of pages in a pool, including the following:

- File `/sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages` for the number of pages in the pool of 2 MB pages.
- File `/sys/kernel/mm/hugepages/hugepages-1048576kB/nr_hugepages` for the number of pages in the pool of 1 GB pages.

The following table outlines commonly used HugeTLB configuration parameters and the multiple file instances that you might find for each parameter.

Table 6-2 Commonly Used File-Based HugeTLB Parameters

Parameter	Purpose	File Paths for Different Instances
`nr_hugepages`	• Each instance of `nr_hugepages` defines the current number of huge pages in the pool associated with that instance. • Can be modified at runtime. • Example command: `echo 20 | sudo tee /proc/sys/vm/nr_hugepages` • Default value is `0`.	The file path formats for different instances of `nr_hugepages` are as follows: • File location: `/proc/sys/vm/nr_hugepages` (present on all systems). • File location: `/sys/kernel/mm/hugepages/hugepages-<SIZE>kB/nr_hugepages` (present on systems that support more than one huge page size). • File location: `/sys/devices/system/node/node{0,1,2…n}/hugepages/hugepages-<SIZE>kB/nr_hugepages` (present on NUMA systems only). **Note:** Use the NUMA node specific path format if you need to request different quantities of pages of different sizes to be supported on specific NUMA nodes. If you use any other path format (for example, `/proc/sys/vm/nr_hugepages`) to request HugeTLB pages. The pages are divided equally between the NUMA nodes.

Table 6-2 (Cont.) Commonly Used File-Based HugeTLB Parameters

Parameter	Purpose	File Paths for Different Instances
`nr_overcommit_hugepages`	• Each instance of `nr_overcommit_hugepages` defines the *additional* number of huge pages that's higher then the quantity specified by `nr_hugepages`. It can be created by the system at runtime through overcommitting memory. • As these additional huge pages become unused, they're freed and returned to the kernel's normal page pool. • Example command: `echo 20 | sudo tee /proc/sys/vm/nr_overcommit_hugepages`	The file path formats for different instances of `nr_overcommit_hugepages` are as follows: • File location `/proc/sys/vm/nr_overcommit_hugepages` (present on all systems). • File location: `/sys/kernel/mm/hugepages/`**`hugepages-<`*`SIZE`*`>`**`kB/nr_overcommit_hugepages` (present on systems that support more than one huge page size). **Note:** The `nr_overcommit_hugepages` parameter isn't defined at the individual node level, so no nod

Table 6-2 (Cont.) Commonly Used File-Based HugeTLB Parameters

Parameter	Purpose	File Paths for Different Instances
		e specific file exists for this setting.
`free_hugepages`	• Read-only parameter. • Each instance of `free_hugepages` returns the number of huge pages in its associated page pool that have yet to be allocated.	The file path formats for different instances of `free_hugepages` are as follows: • File location: `/sys/kernel/mm/hugepages/hugepages-<`*`SIZE`*`>kB/free_hugepages` (present on systems that support more than one huge page size). • File location: `/sys/devices/system/node/node{`*`0,1,2…n`*`}/hugepages/hugepages-<SIZE>kB/free_hugepages` (present on NUMA systems only).

Table 6-2 (Cont.) Commonly Used File-Based HugeTLB Parameters

Parameter	Purpose	File Paths for Different Instances
`surplus_hugepages`	• Read-only parameter. • Each instance of `surplus_hugepages` returns the number of huge pages that have been overcommitted from its associated page pool.	The file path formats for different instances of `surplus_hugepages` are as follows: • File location: `/sys/kernel/mm/hugepages/`**`hugepages-<SIZE>kB`**`/surplus_hugepages` (present on systems that support more than one huge page size). • File location: `/sys/devices/system/node/`**`node{0,1,2…n}`**`/hugepages/`**`hugepages-<SIZE>kB`**`/surplus_hugepages` (present on NUMA systems only).

The following sections show file branches under which different instances of the HugeTLB parameters are stored:

/proc/sys/vm

All systems that support static huge pages contain HugeTLB parameter files under `/proc/sys/vm`.

Note:

On many systems, including many Oracle database servers, the `procfs` file system is the main parameter-set used.

The `sysctl` parameter `vm.nr_hugepages` that's commonly initialized in scripts that request huge pages also writes to the `procfs` file `/proc/sys/vm/nr_hugepages`.

The following are example folders under branch `/proc/sys/vm`:

```
├── ...
├── ...
├── nr_hugepages
├── ...
├── nr_overcommit_hugepages
├── ...
├── ...
```

/sys/kernel/mm/hugepages/

Systems that support multiple size pools contain HugeTLB parameter files in size-specific folders under `/sys/kernel/mm/hugepages/`.

The following are example folders under branch `/sys/kernel/mm/hugepages/`:

```
└── hugepages-2048kB
    ├── free_hugepages
    ├── nr_hugepages
    ├── ...
    ├── nr_overcommit_hugepages
    ├── ...
    └── surplus_hugepages

└── hugepages-1048576kB
    ├── free_hugepages
    ├── nr_hugepages
    ├── ...
    ├── nr_overcommit_hugepages
    ├── ...
    └── surplus_hugepages
```

/sys/devices/system/node/

Only NUMA systems contain HugeTLB parameter files under /`sys/devices/system/node/`.

The following are example folders under branch `/sys/devices/system/node`:

```
├─  ...
├── node0
│   ├── ...
│   ├──hugepages
│          hugepages-2048kB
│              ├── free_hugepages
│              ├── nr_hugepages
│              └── surplus_hugepages
│
│          hugepages-1048576kB
│              ├── free_hugepages
│              ├── nr_hugepages
│              └── surplus_hugepages
├── node1
    ├── ...
    ├──hugepages
           hugepages-2048kB
               ├── free_hugepages
               ├── nr_hugepages
               └── surplus_hugepages

           hugepages-1048576kB
               ├── free_hugepages
               ├── nr_hugepages
               └── surplus_hugepages
```

Configuring HugeTLB Pages at Boot Time

The precise way to request huge pages at boot time depends upon the system's requirements. The following example procedures provide possible starting points.

Requesting HugeTLB Pages by Using Kernel Parameters at Boot Time

The following procedure shows how to use kernel command line options to specify two pools of HugeTLB pages and a default page size on a system that handles multiple huge page sizes. In this procedure, the following are requested:

- A default page size of 1 GB.
- One pool with four HugeTLB pages of 1 GB size.
- One pool of 1500 HugeTLB pages of 2 MB size.

Before beginning the following procedure, ensure that you have the administrative privileges required.

1. Specify 1 GB size for kernel boot parameter `default_hugepagesz` and 2 pairs of "`hugepagesz=<`*`Size_num`*`>G hugepages=`*`Qty_num`*" parameters for the two huge page pools. Append the following line to the kernel command line options in `/etc/default/grub`

   ```
   default_hugepagesz=1G hugepagesz=1G hugepages=4 hugepagesz=2M
   hugepages=1500
   ```

2. Regenerate the `GRUB2` configuration file:
   a. If the system uses BIOS firmware, run the following command:

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

   b. If the system uses UEFI framework, run the following command:

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

3. The next time the system boots, the two huge page pools are requested.

Requesting HugeTLB Pages Using NUMA Node Specific Parameters Early in the Boot Process

Huge Pages requested by using the kernel boot-time parameters, as shown in the previous example, are divided equally between the NUMA nodes.

However, you might need to request a different number of huge pages for specific nodes by setting the configuration values in a node specific file path. The file path is defined as follows:.

```
/sys/devices/system/node/node{0,1,2…n}/hugepages/hugepages-<SIZE>kB/
```

The following procedure describes how to reserve 299 pages of 2 MB size on node 0, and 300 pages of 2 MB size on node 1 on a NUMA system.

Before beginning the following procedure, ensure that you have the administrative privileges required for all of the steps.

1. Create a script file called `hugetlb-reserve-pages.sh` in the `/usr/lib/systemd/` directory and add the following content.

   ```
   #!/bin/sh

   nodes_path=/sys/devices/system/node/
   if [ ! -d $nodes_path ]; then
       echo "ERROR: $nodes_path does not exist"
       exit 1
   fi

   #######################################################
   #                                                     #
   #      FUNCTION                                       #
   #            reserve_pages <number_of_pages> <node_id> #
   #                                                     #
   #######################################################

   reserve_pages()
   {
       echo $1 > $nodes_path/$2/hugepages/hugepages-2048kB/nr_hugepages
   }

   reserve_pages 299 node0
   reserve_pages 300 node1
   ```

2. Make the script executable:

   ```
   sudo chmod +x /usr/lib/systemd/hugetlb-reserve-pages.sh
   ```

3. Create a service file called `hugetlb-gigantic-pages.service` in the `/usr/lib/systemd/system/` directory and add the following content to it.

   ```
   [Unit]
   Description=HugeTLB Gigantic Pages Reservation
   DefaultDependencies=no
   Before=dev-hugepages.mount
   ConditionPathExists=/sys/devices/system/node

   [Service]
   Type=oneshot
   RemainAfterExit=yes
   ExecStart=/usr/lib/systemd/hugetlb-reserve-pages.sh

   [Install]
   WantedBy=sysinit.target
   ```

4. Enable the service file.

   ```
   sudo systemctl enable hugetlb-gigantic-pages
   ```

Configuring HugeTLB at Runtime

In certain cases, you might need make a request for huge pages at runtime.

Configuring HugeTLB Pages for a Specific NUMA Node at Runtime

The following procedure shows how to request 20 HugeTLB pages of size 2048 kB for node2 at runtime.

Before starting, you must ensure you have the required administrative privileges required for all the steps.

1. Run the numastat command to show memory statistics relating to the NUMA nodes:

   ```
   numastat -cm | egrep 'Node|Huge'| grep -v AnonHugePages


                    Node 0 Node 1 Node 2 Node 3  Total add
   HugePages_Total       0      0      0      0      0
   HugePages_Free        0      0      0      0      0
   HugePages_Surp        0      0      0      0      0
   ```

2. Add the required number of huge pages of a specified size to the selected node, for example 20 pages of 2 MB size on node 2:

   ```
   echo 20 | sudo tee /sys/devices/system/node/node2/hugepages/
   hugepages-2048kB/nr_hugepages
   ```

3. Run the `numastat` command again to ensure the request was successful and that the requested memory (in our example 20 x 2 MB pages = 40 MB) has been added `HugePages_Total` for `node2`:

   ```
   numastat -cm | egrep 'Node|Huge'| grep -v AnonHugePages


                    Node 0 Node 1 Node 2 Node 3  Total
   HugePages_Total       0      0     40      0     40
   HugePages_Free        0      0     40      0     40
   HugePages_Surp        0      0      0      0      0
   ```

Configuring Transparent HugePages

The Transparent HugePages (THP) feature is enabled by default in Oracle Linux. However, you might still need to access and configure THP according to the system's needs.

The following sections look at various THP parameters and examples of how they can be configured.

Parameters Used to Configure Transparent HugePages

The following table describes selected parameter settings that can be used when configuring Transparent HugePages (THP).

Table 6-3   Commonly Used THP Parameters

Parameter	File Location	Value Options
`enabled`	`/sys/kernel/mm/transparent_hugepage/enabled`	Sets THP and its mode, which is one of the following: • `always` (default): THP is enabled in `system-wide` mode. In this setting, the kernel, whenever possible, assigns huge pages to processes using large contiguous virtual memory areas. • `madvise`: THP is enabled in `per-process` mode. In this setting the kernel only assigns huge pages to application processes that explicitly request huge pages through the `madvise()` system call. • `disabled`: THP is disabled.

Table 6-3 (Cont.) Commonly Used THP Parameters

Parameter	File Location	Value Options
`defrag`	`/sys/kernel/mm/transparent_hugepage/defrag`	Determines how aggressively an application can reclaim pages and defrag memory when THP is unavailable. The following list explains the available options: • `always`: An application requesting THP stalls on allocation failure and directly reclaims pages and compact memory to obtain a THP immediately. • `defer`: An application doesn't stall but continues using small pages. The application requests the kernel daemons `kswapd` and `kcompactd` to reclaim pages and compact memory so that THP is available later. • `defer+madvise`: Regions using the `madvise(MADV_HUGEPAGE)` call stall on allocation failure and directly reclaim pages and compact memory to obtain a THP immediately However, all other regions request the kernel daemons `kswapd` and `kcompactd` to reclaim pages and compact memory so that THP is available later. • `madvise` (default): Regions using the `madvise(MADV_HUGEPAGE)` call stall on allocation failure and directly reclaim pages and compact memory to obtain a THP immediately.

Configuring Transparent HugePages at Runtime

The following sections show you examples of how you can configure THP at runtime by accessing the THP parameters in the `sysfs` virtual file system.

Retrieving the Current Status of Transparent HugePages

To see the current setting of THP you can read the `/sys/kernel/mm/transparent_hugepage/enabled` parameter as shown in the following code sample:

```
sudo cat /sys/kernel/mm/transparent_hugepage/enabled


[always] madvise never
```

The value inside the square brackets represents the current setting.

Changing the Current Status of Transparent HugePages

To change the current status of THP, you need to write the preferred settings to `/sys/kernel/mm/transparent_hugepage/enabled`. The following example shows you how to set the status to `always`:

1. Check the current status of THP by reading the `enabled` parameter.

   ```
   sudo cat /sys/kernel/mm/transparent_hugepage/enabled


   always madvise [never]
   ```

   The value inside the square brackets represents the current setting.
2. Set THP mode to `always`.

   ```
   echo always | sudo tee /sys/kernel/mm/transparent_hugepage/enabled
   ```

3. Confirm the change has been successful by reading the `enabled` parameter.

   ```
   sudo cat /sys/kernel/mm/transparent_hugepage/enabled


   [always] madvise never
   ```

Note:

Virtual file systems such as `sysfs` provide a file system interface to items that aren't necessarily stored as files on disk. The `sysfs` files therefore don't always interact with file commands in the same way that regular physical files on disk would. In the previous example, the `echo` command used doesn't overwrite `/sys/kernel/mm/transparent_hugepage/enabled`, as it would if used with a regular file, but instead changes the selected option:

```
sudo cat /sys/kernel/mm/transparent_hugepage/enabled

always madvise [never]

echo always | sudo tee /sys/kernel/mm/transparent_hugepage/enabled

always

sudo cat /sys/kernel/mm/transparent_hugepage/enabled

[always] madvise never
```

.

Changing the defrag Setting of Transparent HugePages

To change the THP defrag setting you need to write the setting of your choice to `/sys/kernel/mm/transparent_hugepage/defrag`

Note:

The best `defrag` setting varies from system to system. Reclaiming pages and memory compaction can increase the number of THP pages available. However, the process also uses CPU time. Therefore, you need to find the correct balance for a specific system.

The following example shows you how to set the `defrag` setting to `madvise`.

1. Check the current value of the `defrag` parameter:

   ```
   sudo cat /sys/kernel/mm/transparent_hugepage/defrag

   [always] defer defer+madvise madvise never
   ```

   The value inside square brackets represents the current setting.

2. Set the `/sys/kernel/mm/transparent_hugepage/defrag` parameter to `madvise`:

   ```
   echo madvise | sudo tee /sys/kernel/mm/transparent_hugepage/defrag
   ```

3. Confirm the change has worked by reading the `defrag` parameter.

   ```
   sudo cat /sys/kernel/mm/transparent_hugepage/defrag
   ```

   ```
   always defer defer+madvise [madvise] never
   ```

7

Managing Resources

This chapter describes how to manage the use of resources in an Oracle Linux system.

About Control Groups

Control groups, usually referred to as `cgroups`, are an Oracle Linux kernel feature that enables processes (`PIDs`) to be organized into hierarchical groups for the purpose of resource allocation. For example, if you have identified 3 sets of processes that need to be allocated CPU time in a ratio of 150:100:50, you can create 3 `cgroups`, each with a CPU weight corresponding to one of the 3 values in your ratio, and then assign the appropriate processes to each `cgroup`.

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 folder 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 `cgroups` of your own by creating your own folders under the `/sys/fs/cgroup` virtual file system and assigning process IDs (`PIDs`) to different `cgroups` according to your system needs. However, the recommended practice is to use `systemd` to configure `cgroups` instead of creating the `cgroups` manually under `/sys/fs/cgroup`. See Using systemd to Manage cgroups v2 for the recommended method of managing `cgroups` through `systemd`.

Note:

Use `systemd` to configure `cgroups`.

Although the recommended method for configuring using `systemd` to manage cgroups, this topic also covers the manual creation of `cgroup` folders in the `/sys/fs/cgroup` file system. However, this coverage is mainly to provide background knowledge of the kernel `cgroup` feature to which `systemd` provides access.

Oracle Linux provides two types of control groups:

Control groups version 1 (`cgroups v1`)
These groups provide a per-resource controller hierarchy. Each resource, such as CPU, memory, I/O, and so on, has its own control group hierarchy. A disadvantage of this group is the difficulty of establishing proper coordination of resource use among groups that might belong to different process hierarchies.

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.

Both versions are present in Oracle Linux. However, by default, the `cgroups v2` functionality is enabled and mounted on Oracle Linux 9 systems.

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

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 both `cgroups v1` and `cgroups v2`, see the `cgroups(7)` manual page.

About the Control Group File System

This section describes how `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 contents of the root control group directory differ depending on which `cgroup` version is mounted on the system. For `cgroups v2`, the directory contents are 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 your own `cgroup` subdirectories below 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 practice is to create child `cgroups` 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 Control Groups and systemd

Control groups can be used by the `systemd` system and service manager for resource management. `Systemd` uses these groups to organize units and services that consume resources. For more information about `systemd`, see About the systemd Service Manager.

`Systemd` provides different unit types, three of which are for resource control purposes:

- **Service**: A process or a group of processes whose settings are based on a unit configuration file. Services encompass specified processes in a "collection" so that `systemd` can start or stop the processes as one set. Service names follow the format *`name`*`.service`.
- **Scope**: A group of externally created processes, such as user sessions, containers, virtual machines, and so on. Similar to services, scopes encapsulate these created processes and are started or stopped by the arbitrary processes and then registered by `systemd` at runtime. Scope names follow the format *`name`*`.scope`.
- **Slice**: A group of hierarchically organized units in which services and scopes are located. Thus, slices themselves don't contain processes. Rather, the scopes and services in a slice define the processes. Every name of a slice unit corresponds to the path to a location in the hierarchy. Root slices, typically `user.slice` for all user-based processes and `system.slice` for system-based processes, are automatically created in the hierarchy. Parent slices exist immediately below the root slice and follow the format *`parent-name`*`.slice`. These root slices can then have subslices on multiple levels.

The service, the scope, and the slice units directly map to objects in the control group hierarchy. When these units are activated, they map directly to control group paths that are built from the unit names. To display the mapping between the `systemd` resource unit types and control groups, type:

```
sudo systemd-cgls
```

```
Working directory /sys/fs/cgroup:
├─user.slice (#1243)
│ → trusted.invocation_id: 50ce3909b2644f919ee420adc39edb4b
│ ├─user-1001.slice (#4167)
│ │ → trusted.invocation_id: 02e80a960d4549a7a9c69ce0fb546c26
│ │ ├─session-2.scope (#4405)
│ │ │ ├─2417 sshd: alice [priv]
│ │ │ ├─2430 sshd: alice@pts/0
│ │ │ ├─2431 -bash
│ │ │ ├─2689 sudo systemd-cgls
│ │ │ ├─2691 systemd-cgls
│ │ │ └─2692 less
...
│   └─user@984.service … (#3827)
│     → trusted.delegate: 1
│     → trusted.invocation_id: 09b47ce9f3124239b75814114353f3f2
│     └─init.scope (#3861)
│       ├─2058 /usr/lib/systemd/systemd --user
│       └─2099 (sd-pam)
├─init.scope (#19)
│ └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 17
└─system.slice (#53)
...
  ├─chronyd.service (#2467)
  │ → trusted.invocation_id: c0f77aaa9c7844e6bef6a6898ae4dd56
  │ └─1358 /usr/sbin/chronyd -F 2
  ├─auditd.service (#2331)
  │ → trusted.invocation_id: 756808add6a348609316c9e8c1801846
  │ └─1310 /sbin/auditd
  ├─tuned.service (#3079)
  │ → trusted.invocation_id: 2c358135fc46464d862b05550338d4f4
  │ └─1415 /usr/bin/python3 -Es /usr/sbin/tuned -l -P
  ├─systemd-journald.service (#1651)
  │ → trusted.invocation_id: 7cb7ccb14e044a899aadf47bbb583ada
  │ └─977 /usr/lib/systemd/systemd-journald
  ├─atd.service (#3623)
  │ → trusted.invocation_id: 597a7a4e5646468db407801b8562d869
  │ └─1915 /usr/sbin/atd -f
  ├─sshd.service (#3419)
  │ → trusted.invocation_id: 490504a683fc4311ab0fbeb0864a1a34
  │ └─1871 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
...
```

For an example of how to use `systemd` commands such as `systemctl` to manage resources, see Controlling Access to System Resources. For further technical details, see the `systemctl(1)`, `systemd-cgls(1)`, and `systemd.resource-control(5)` manual pages.

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
A control group can use the configured amount of a resource. However, you can also overcommit resources. Therefore, the sum of the subgroups limits can exceed the limit of the parent group.
To implement this distribution model, the `MemoryMax` option is 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 used.

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

Using cgroups v2 to Manage Resources for Applications

This section shows you how to enable the `cgroups v2` feature so you can create and configure `cgroups` to manage the distribution of resources amongst processes running on your system.

The sample procedure included in this section 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 section 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` when following the sample procedure.

Enabling cgroups v2

At boot time, Oracle Linux 9 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.

   ```
   ll /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.

Preparing the Control Group for Distribution of CPU Time

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

   ```
   ll /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 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/Example/g1/cgroup.procs
   echo "33302" | sudo tee /sys/fs/cgroup/Example/g2/cgroup.procs
   echo "33303" | sudo /sys/fs/cgroup/Example/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
   ...
   ```

Using systemd to Manage cgroups v2

The preferred method of managing resource allocation with `cgroups v2` is to use the control group functionality provided by `systemd`.

Note:

For information on enabling `cgroups v2` functionality on your system, see Enabling cgroups v2

By default, `systemd` creates a `cgroup` folder for each `systemd` service set up on the host. `systemd` names these folders using the format *servicename*`.service`, where *servicename* is the name of the service associated with the folder.

To see a list of the `cgroup` folders `systemd` creates for the services, run the `ls` command on the **`system.slice`** branch of the `cgroup` file system as shown in the following sample code block:

```
ls /sys/fs/cgroup/system.slice/
...                     ...                           ...
app_service1.service    cgroup.subtree_control        httpd.service
app_service2.service    chronyd.service               ...
```

```
...                     crond.service               ...
cgroup.controllers      dbus-broker.service         ...
cgroup.events           dtprobed.service            ...
cgroup.freeze           firewalld.service           ...
...                     gssproxy.service            ...
...                     ...                         ...
```

In the preceding command block:

- The folders *app_service1*.service and *app_service2*.service represent custom application services you might have on your system.

In addition to service control groups, systemd also creates a cgroup folder for each user on the host. To see the cgroups created for each user you can run the ls command on the **user.slice** branch of the cgroup file system as shown in the following sample code block:

```
ls /sys/fs/cgroup/user.slice/
cgroup.controllers      cgroup.subtree_control      user-1001.slice
cgroup.events           cgroup.threads              user-982.slice
cgroup.freeze           cgroup.type                 ...
...                     ...                         ...
...                     ...                         ...
...                     ...                         ...
```

In the preceding code block:

- Each user cgroup folder is named using the format user-*UID*.slice. So, control group user-1001.slice is for a user whose UID is 1001, for example.

systemd provides high-level access to the cgroups and kernel resource controller features so you do not have to access the file system directly. For example, to set the CPU weight of a service called *app_service1*.service, you might choose to run the systemctl set-property command as follows:

```
sudo systemctl set-property app_service1.service CPUWeight=150
```

Thus, systemd enables you to manage resource distribution at an application level, rather than the process PID level used when configuring cgroups without using systemd functionality.

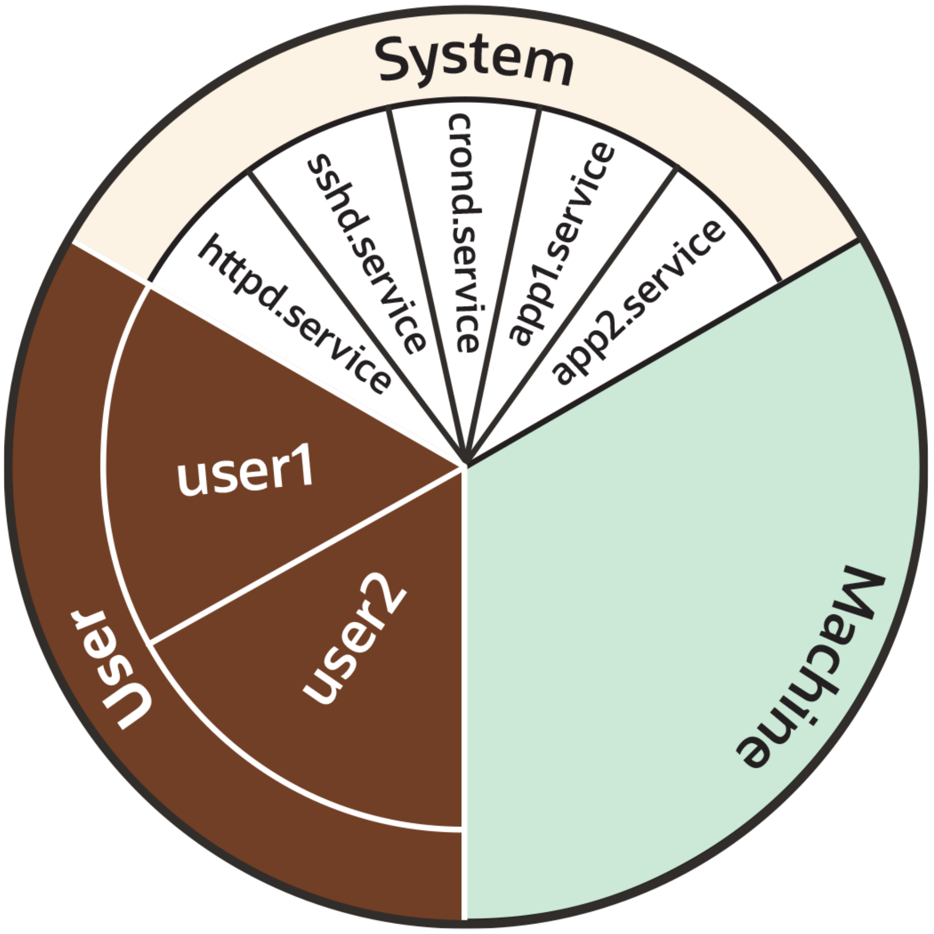
About Slices and Resource Allocation in systemd

This section looks at the way systemd initially divides each of the default kernel controllers, for example CPU, memory and blkio, into portions called "slices" as illustrated by the following example pie chart:

Note:

You can also create your own custom slices for resource distribution, as shown in section Setting Resource Controller Options and Creating Custom Slices.

Figure 7-1 Pie chart illustrating distribution in a resource controller, such as CPU or Memory



As the preceding pie chart shows, by default each resource controller is divided equally between the following 3 slices:

- System (`system.slice`).
- User (`user.slice`).
- Machine (`machine.slice`).

The following list looks at each slice more closely. For the purposes of discussion, the examples in the list focus on the CPU controller.

System (`system.slice`)
This resource slice is used for managing resource allocation amongst daemons and service units.
As shown in the preceding example pie chart, the system slice is divided into further sub-slices. For example, in the case of CPU resources, we might have sub-slice allocations within the system slice that include the following:

- `httpd.service` (`CPUWeight`=100)

- `sshd.service` (`CPUWeight` =100)
- `crond.service` (`CPUWeight` =100)
- *app1*.`service` (`CPUWeight` =100)
- *app2*.`service` (`CPUWeight` =100)

In the preceding list, *app1*.`service` and *app2*.`service` represent custom application services you might have running on your system.

User (`user.slice`)
This resource slice is used for managing resource allocation amongst user sessions. A single slice is created for each `UID` irrespective of how many logins the associated user has active on the server. Continuing with our pie chart example, the sub-slices might be as follows:

- *user1* (`CPUWeight`=100, `UID`=982)
- *user2* (`CPUWeight`=100, `UID`=1001)

Machine (`machine.slice`)
This slice of the resource is used for managing resource allocation amongst hosted virtual machines, such as KVM guests, and Linux Containers. The machine slice is only present on a server if the server is hosting virtual machines or Linux Containers.

Note:

Share allocations do not set a maximum limit for a resource.

For instance, in the preceding examples, the slice `user.slice` has 2 users: *user1* and *user2*. Each user is allocated an equal share of the CPU resource available to the parent `user.slice`. However, if the processes associated with *user1* are idle, and do not require any CPU resource, then its CPU share is available for allocation to *user2* if needed. In such a situation, *user2* might even be allocated the entire CPU resource apportioned to the parent `user.slice` if it is required by other users.

To cap CPU resource, you would need to set the `CPUQuota` property to the required percentage.

Slices, Services, and Scopes in the cgroup Hierarchy

The pie chart analogy used in the preceding sections is a helpful way to conceptualize the division of resources into slices. However, in terms of structural organization, the control groups are arranged in a hierarchy. You can view the `systemd` control group hierarchy on your system by running the `systemd-cgls` command as follows:

Tip:

To see the entire `cgroup` hierarchy, starting from the root slice `-.slice`, as in the following example, ensure you run `systemd-cgls` from outside of the control group mount point `/sys/fs/cgroup/`. Otherwise, If you run the command from within `/sys/fs/cgroup/`, the output starts from the `cgroup` location from which the command was run. See `systemd-cgls(1)` for more information.

```
systemd-cgls
```

```
Control group /:
-.slice
...
├─user.slice (#1429)
│ → user.invocation_id: 604cf5ef07fa4bb4bb86993bb5ec15e0
│ ├─user-982.slice (#4131)
│ │ → user.invocation_id: 9d0d94d7b8a54bcea2498048911136c8
│ │ ├─session-c1.scope (#4437)
│ │ │ ├─2416 /usr/bin/sudo -u ocarun /usr/libexec/oracle-cloud-agent/plugins/
runcommand/runcommand
│ │ │ └─2494 /usr/libexec/oracle-cloud-agent/plugins/runcommand/runcommand
│ │ └─user@982.service … (#4199)
│ │   → user.delegate: 1
│ │   → user.invocation_id: 37c7aed7aa6e4874980b79616acf0c82
│ │   └─init.scope (#4233)
│ │     ├─2437 /usr/lib/systemd/systemd --user
│ │     └─2445 (sd-pam)
│ └─user-1001.slice (#7225)
│   → user.invocation_id: ce93ad5f5299407e9477964494df63b7
│   ├─session-2.scope (#7463)
│   │ ├─20304 sshd: oracle [priv]
│   │ ├─20404 sshd: oracle@pts/0
│   │ ├─20405 -bash
│   │ ├─20441 systemd-cgls
│   │ └─20442 less
│   └─user@1001.service … (#7293)
│     → user.delegate: 1
│     → user.invocation_id: 70284db060c1476db5f3633e5fda7fba
│     └─init.scope (#7327)
│       ├─20395 /usr/lib/systemd/systemd --user
│       └─20397 (sd-pam)
├─init.scope (#19)
│ └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 28
└─system.slice (#53)
 ...
  ├─dbus-broker.service (#2737)
  │ → user.invocation_id: 2bbe054a2c4d49809b16cb9c6552d5a6
  │ ├─1450 /usr/bin/dbus-broker-launch --scope system --audit
  │ └─1457 dbus-broker --log 4 --controller 9 --machine-id
852951209c274cfea35a953ad2964622 --max-bytes 536870912 --max-fds 4096 --max-
matches 131072 --audit
 ...
```

```
├─chronyd.service (#2805)
│ → user.invocation_id: e264f67ad6114ad5afbe7929142faa4b
│ └─1482 /usr/sbin/chronyd -F 2
├─auditd.service (#2601)
│ → user.invocation_id: f7a8286921734949b73849b4642e3277
│ ├─1421 /sbin/auditd
│ └─1423 /usr/sbin/sedispatch
├─tuned.service (#3349)
│ → user.invocation_id: fec7f73678754ed687e3910017886c5e
│ └─1564 /usr/bin/python3 -Es /usr/sbin/tuned -l -P
├─systemd-journald.service (#1837)
│ → user.invocation_id: bf7fb22ba12f44afab3054aab661aedb
│ └─1068 /usr/lib/systemd/systemd-journald
├─atd.service (#3961)
│ → user.invocation_id: 1c59679265ab492482bfdc9c02f5eec5
│ └─2146 /usr/sbin/atd -f
├─sshd.service (#3757)
│ → user.invocation_id: 57e195491341431298db233e998fb180
│ └─2097 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
├─crond.service (#3995)
│ → user.invocation_id: 4f5b380a53db4de5adcf23f35d638ff5
│ └─2150 /usr/sbin/crond -n
...
```

The preceding sample output shows how all "`*.slice`" control groups reside under the root slice `-.slice`. Beneath the root slice you can see the `user.slice` and `system.slice` control groups, each with their own child `cgroup` sub-slices.

Examining the `systemd-cgls` command output you can see how, with the exception of root `-.slice` , all processes are on leaf nodes. This arrangement is enforced by `cgroups` v2, in a rule called the "no internal processes" rule. See `cgroups (7)` for more information about the "no internal processes" rule.

The output in the preceding `systemd-cgls` command example also shows how slices can have descendent child control groups that are `systemd` scopes. `systemd` scopes are reviewed in the following section.

systemd Scopes

`systemd` scope is a `systemd` unit type that groups together system service worker processes that have been launched independently of `systemd`. The scope units are transient `cgroups` created programmatically using the bus interfaces of `systemd`.

For example, in the following sample code, the user with `UID` 1001 has run the `systemd-cgls` command, and the output shows `session-2.scope` has been created for processes the user has spawned independently of `systemd` (including the process for the command itself , `21380 sudo systemd-cgls`):

Note:

In the following example, the command has been run from within the control group mount point `/sys/fs/cgroup/`. Hence, instead of the root slice, the output starts from the `cgroup` location from which the command was run.

```
sudo systemd-cgls
```

```
Working directory /sys/fs/cgroup:
...
├─user.slice (#1429)
│ → user.invocation_id: 604cf5ef07fa4bb4bb86993bb5ec15e0
│ → trusted.invocation_id: 604cf5ef07fa4bb4bb86993bb5ec15e0
...
│ └─user-1001.slice (#7225)
│   → user.invocation_id: ce93ad5f5299407e9477964494df63b7
│   → trusted.invocation_id: ce93ad5f5299407e9477964494df63b7
│   ├─session-2.scope (#7463)
│   │ ├─20304 sshd: oracle [priv]
│   │ ├─20404 sshd: oracle@pts/0
│   │ ├─20405 -bash
│   │ ├─21380 sudo systemd-cgls
│   │ ├─21382 systemd-cgls
│   │ └─21383 less
│   └─user@1001.service … (#7293)
│     → user.delegate: 1
│     → trusted.delegate: 1
│     → user.invocation_id: 70284db060c1476db5f3633e5fda7fba
│     → trusted.invocation_id: 70284db060c1476db5f3633e5fda7fba
│     └─init.scope (#7327)
│       ├─20395 /usr/lib/systemd/systemd --user
│       └─20397 (sd-pam)
```

Setting Resource Controller Options and Creating Custom Slices

`systemd` provides the following methods for setting resource controller options, such as `CPUWeight`, `CPUQuota`, and so on, to customize resource allocation on your system:

- Using service unit files.
- Using drop-in files.
- Using the `systemctl set-property` command.

The following sections provide example procedures for using each of these methods to configure resources and slices in your system.

Using Service Unit Files

To set options in a service unit file, perform the following steps:

1. Create file `/etc/systemd/system/myservice1.service` with the following content:

   ```
   [Service]
   Type=oneshot
   ExecStart=/usr/lib/systemd/generate_load.sh
   TimeoutSec=0
   StandardOutput=tty
   RemainAfterExit=yes

   [Install]
   WantedBy=multi-user.target
   ```

2. The service created in the preceding step requires a bash script `/usr/lib/systemd/generate_load.sh`. Create the file with the following content:

   ```
   #!/bin/bash
   for i in {1..4};do while : ; do : ; done & done
   ```

3. Make the script runnable:

   ```
   sudo chmod +x /usr/lib/systemd/generate_load.sh
   ```

4. Enable and start the service:

   ```
   sudo systemctl enable myservice1 --now
   ```

5. Run the `systemd-cgls` command and confirm the service `myservice1` is running under `system.slice`:

   ```
   systemd-cgls
   ```

   ```
   Control group /:
   -.slice
   ...
   ├─user.slice (#1429)
   ...
   └─system.slice (#53)
     ...
     ├─myservice1.service (#7939)
     │ → user.invocation_id: e227f8f288444fed92a976d391e6a897
     │ ├─22325 /bin/bash /usr/lib/systemd/generate_load.sh
     │ ├─22326 /bin/bash /usr/lib/systemd/generate_load.sh
     │ ├─22327 /bin/bash /usr/lib/systemd/generate_load.sh
     │ └─22328 /bin/bash /usr/lib/systemd/generate_load.sh
     ├─pmie.service (#4369)
     │ → user.invocation_id: 68fcd40071594481936edf0f1d7a8e12
      ...
   ```

6. Create a custom slice for the service.

Add the line `Slice=my_custom_slice.slice` to the `[Service]` section in the `myservice1.service` file, created in a previous step, as shown in the following code block:

```
[Service]
Slice=my_custom_slice.slice
Type=oneshot
ExecStart=/usr/lib/systemd/generate_load.sh
TimeoutSec=0
StandardOutput=tty
RemainAfterExit=yes

[Install]
WantedBy=multi-user.target
```

NOT_SUPPORTED:

Use underscores instead of dashes to separate terms in slice names.

In `systemd`, a dash in a slice name is a special character: in `systemd`, dashes in slice names are used to describe the full `cgroup` path to the slice (starting from the root slice). For example, if you specify a slice name as "`my-custom-slice.slice`", instead of creating a slice of that name, `systemd` creates the following `cgroups` path underneath the root slice: `my.slice/my-custom.slice/my-custom-slice.slice`.

7. After editing the file, ensure `systemd` reloads its configuration files and then restart the service:

```
sudo systemctl daemon-reload
sudo systemctl restart myservice1
```

8. Run the `systemd-cgls` command and confirm the service `myservice1` is now running under custom slice `my_custom_slice`:

```
systemd-cgls
```

```
Control group /:
-.slice
...
├─user.slice (#1429)
...
├─my_custom_slice.slice (#7973)
│ → user.invocation_id: a8a493a8db1342be85e2cdf1e80255f8
│ └─myservice1.service (#8007)
│   → user.invocation_id: 9a4a6171f2844e479d4a0f347aac38ce
│   ├─22385 /bin/bash /usr/lib/systemd/generate_load.sh
│   ├─22386 /bin/bash /usr/lib/systemd/generate_load.sh
│   ├─22387 /bin/bash /usr/lib/systemd/generate_load.sh
│   └─22388 /bin/bash /usr/lib/systemd/generate_load.sh
├─init.scope (#19)
│ └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 28
└─system.slice (#53)
```

```
├─irqbalance.service (#2907)
│ → user.invocation_id: 00d64c9b9d224f179496a83536dd60bb
│ └─1464 /usr/sbin/irqbalance --foreground
...
```

Using Drop-in Files

To use a drop-in file to configure resources, perform the following steps:

1. Create the directory for your service drop-in file.

   Tip:

   The "drop-in" directory for drop-in files for a service is located at `/etc/systemd/system/`*`service_name`*`.service.d` where *service_name* is the name of the service.

   Continuing with our example with service `myservice1`, we would run the following command:

   ```
   sudo mkdir -p /etc/systemd/system/myservice1.service.d/
   ```

2. Create 2 drop-in files called `00-slice.conf` and `10-CPUSettings.conf` in the `myservice1.service.d` directory created in the preceding step.

   Note:

   - Multiple drop-in files with different names are applied in **lexicographic** order.
   - These drop-in files take precedence over the service unit file.

3. a. Add the following contents to `00-slice.conf`

      ```
      [Service]
      Slice=my_custom_slice2.slice
      MemoryAccounting=yes
      CPUAccounting=yes
      ```

   b. And add the following contents to `10-CPUSettings.conf`

      ```
      [Service]
      CPUWeight=200
      ```

4. Create a second service (`myservice2`) and assign it a different `CPUWeight` to that assigned to `myservice1`:

   a. Create file `/etc/systemd/system/myservice2.service` with the following contents:

      ```
      [Service]
      Slice=my_custom_slice2.slice
      Type=oneshot
      ```

```
ExecStart=/usr/lib/systemd/generate_load2.sh
TimeoutSec=0
StandardOutput=tty
RemainAfterExit=yes

[Install]
WantedBy=multi-user.target
```

b. The service created in the preceding step requires a bash script `/usr/lib/systemd/generate_load2.sh`. Create the file with the following content:

```
#!/bin/bash
for i in {1..4};do while : ; do : ; done & done
```

c. Make the script runnable:

```
sudo chmod +x /usr/lib/systemd/generate_load2.sh
```

d. Create a drop in file `/etc/systemd/system/myservice2.service.d/10-CPUSettings.conf` for myservice2 with the following contents:

```
[Service]
CPUWeight=400
```

5. Ensure `systemd` reloads its configuration files, and restart `myservice1`, and also enable and start `myservices2`:

```
sudo systemctl daemon-reload
sudo systemctl restart myservice1
sudo systemctl enable myservice2 --now
```

6. Run the `systemd-cgtop` command to display control groups ordered by their resource usage. You can see from the following sample output how, in addition to the resource usage of each slice, the systemd-`cgtop` command displays resource usage breakdown within each slice, so you can use it to confirm your CPU weight has been divided as expected.

```
systemd-cgtop


Control Group                                      Tasks   %CPU   Memory
Input/s Output/s
/                                                    228  198.8
712.5M        -        -
my_custom_slice2.slice                                 8  198.5
1.8M        -        -
my_custom_slice2.slice/myservice2.service              4  132.8
944.0K        -        -
my_custom_slice2.slice/myservice1.service              4   65.6
976.0K        -        -
user.slice                                            18    0.9
43.9M        -        -
user.slice/user-1001.slice                             6    0.9
13.7M        -        -
user.slice/user-1001.slice/session-2.scope             4    0.9
```

```
9.4M         -         -
system.slice                                      60     0.0
690.8M         -         -
```

Using systemctl set-property

The `systemctl set-property` command places the configuration files under the following location:

```
/etc/systemd/system.control
```

Caution:

You must not manually edit the files systemctl set-property command creates.

Note:

The `systemctl set-property` command does not recognize every resource-control property used in the system-unit and drop-in files covered earlier in this topic.

The following procedure demonstrates how you can use the `systemctl set-property` command to configure resource allocation:

1. Continuing with our example, create another service file at location `/etc/systemd/system/myservice3.service` with the following content:

   ```
   [Service]
   Type=oneshot
   ExecStart=/usr/lib/systemd/generate_load3.sh
   TimeoutSec=0
   StandardOutput=tty
   RemainAfterExit=yes
   [Install]
   WantedBy=multi-user.target
   ```

2. Set the slice for the service to be `my_custom_slice2` (the same slice used by the services created in from earlier steps) by adding the following line to the `[Service]` section in the `myservice3.service` file:

   ```
   Slice=my_custom_slice2.slice
   ```

   Note:

   The slice must be set in the service-unit file because the `systemctl set-property` command does not recognize the `Slice` property.

3. The service created in the preceding step requires a bash script `/usr/lib/systemd/generate_load3.sh`. Create the file with the following content:

   ```
   #!/bin/bash
   for i in {1..4};do while : ; do : ; done & done
   ```

4. Make the script runnable:

   ```
   sudo chmod +x /usr/lib/systemd/generate_load3.sh
   ```

5. Ensure systemd reloads its configuration files, and then enable and start the service:

   ```
   sudo systemctl daemon-reload
   sudo systemctl enable myservice3 --now
   ```

6. Optionally run the to `systemd-cgtop` confirm all 3 services, , `myservice1`, `myservice2`, and `myservice3`, are all running in the same slice.
7. Use `systemctl set-property` command to set the `CPUWeight` for `myservice3` to 800:

   ```
   sudo systemctl set-property myservice3.service CPUWeight=800
   ```

8. You can optionally confirm that a drop-in file has been created for you under `/etc/systemd/system.control/myservice3.service.d`. However, you must not edit the file:

   ```
   cat /etc/systemd/system.control/myservice3.service.d/50-CPUWeight.conf


   # This is a drop-in unit file extension, created via "systemctl set-
   property"
   # or an equivalent operation. Do not edit.
   [Service]
   CPUWeight=800
   ```

   Ensure `systemd` reloads its configuration files, and restart all the services:

   ```
   sudo systemctl daemon-reload
   sudo systemctl restart myservice1
   sudo systemctl restart myservice2
   sudo systemctl restart myservice3
   ```

9. Run the `systemd-cgtop` command to confirm your CPU weight has been divided as expected:

   ```
   systemd-cgtop


   Control Group                                         Tasks   %CPU
   Memory  Input/s Output/s
   /                                                       235  200.0
   706.1M        -        -
   my_custom_slice2.slice                                   12  198.4
   2.9M        -        -
   ```

```
my_custom_slice2.slice/myservice3.service                 4  112.7
976.0K        -        -
my_custom_slice2.slice/myservice2.service                 4   56.9
996.0K        -        -
my_custom_slice2.slice/myservice1.service                 4   28.8
988.0K        -        -
user.slice                                               18    0.9
44.1M        -        -
user.slice/user-1001.slice                                6    0.9
13.9M        -        -
user.slice/user-1001.slice/session-2.scope                4    0.9
9.5M        -        -
```

Using cgroups v2 to Manage Resources for Users

The previous sample procedures describe how to manage applications' use of system resources. You can also manage resource use by directly implementing resource filters to users who log in to the system.

Run Control Groups Version 2 on Oracle Linux is a tutorial that provides examples on how to control users' use of system resources. Further, the tutorial offers a lab environment where you can perform steps in real time to regulate resource consumption by users.