proc(5) — Linux manual page

NAME | DESCRIPTION | NOTES | SEE ALSO | COLOPHON

NAME top

proc - process information pseudo-filesystem

DESCRIPTION top

The **proc** filesystem is a pseudo-filesystem which provides an interface to kernel data structures. It is commonly mounted at */proc*. Typically, it is mounted automatically by the system, but it can also be mounted manually using a command such as:

```
mount -t proc proc /proc
```

Most of the files in the **proc** filesystem are read-only, but some files are writable, allowing kernel variables to be changed.

Mount options

The **proc** filesystem supports the following mount options:

hidepid=*n* (since Linux 3.3)
This option controls who can access the information in */proc/[pid]* directories. The argument, *n*, is one of the following values:

0 Everybody may access all */proc/[pid]* directories. This is the traditional behavior, and the default if this mount option is not specified.

1 Users may not access files and subdirectories inside any */proc/[pid]* directories but their own (the */proc/[pid]* directories themselves remain visible). Sensitive files such as */proc/[pid]/cmdline* and */proc/[pid]/status* are now protected against other users. This makes it impossible to learn whether any user is running a specific program (so long as the program doesn't otherwise reveal itself by its behavior).

2 As for mode 1, but in addition the */proc/[pid]* directories belonging to other users become invisible. This means that */proc/[pid]* entries can no longer be used to discover the PIDs on the system. This doesn't hide the fact that a process with a specific PID value exists (it can be learned by other means, for example, by "kill -0 $PID"), but it hides a process's UID and GID, which could otherwise be learned by employing

stat(2) on a */proc/[pid]* directory. This greatly complicates an attacker's task of gathering information about running processes (e.g., discovering whether some daemon is running with elevated privileges, whether another user is running some sensitive program, whether other users are running any program at all, and so on).

gid=*gid* (since Linux 3.3)
Specifies the ID of a group whose members are authorized to learn process information otherwise prohibited by **hidepid** (i.e., users in this group behave as though */proc* was mounted with *hidepid=0*). This group should be used instead of approaches such as putting nonroot users into the sudoers(5) file.

Overview

Underneath */proc*, there are the following general groups of files and subdirectories:

/proc/[pid]
There is a numerical subdirectory for each running process; the subdirectory is named by the process ID. Each */proc/[pid]* subdirectory contains the pseudo-files and directories described below.

The files inside each */proc/[pid]* directory are normally owned by the effective user and effective group ID of the process. However, as a security measure, the ownership is made *root:root* if the process's "dumpable" attribute is set to a value other than 1.

Before Linux 4.11, *root:root* meant the "global" root user ID and group ID (i.e., UID 0 and GID 0 in the initial user namespace). Since Linux 4.11, if the process is in a noninitial user namespace that has a valid mapping for user (group) ID 0 inside the namespace, then the user (group) ownership of the files under */proc/[pid]* is instead made the same as the root user (group) ID of the namespace. This means that inside a container, things work as expected for the container "root" user.

The process's "dumpable" attribute may change for the following reasons:

* The attribute was explicitly set via the prctl(2) **PR_SET_DUMPABLE** operation.

* The attribute was reset to the value in the file */proc/sys/fs/suid_dumpable* (described below), for the reasons described in prctl(2).

Resetting the "dumpable" attribute to 1 reverts the ownership of the */proc/[pid]/** files to the process's effective UID and GID. Note, however, that if the effective UID or GID is subsequently modified, then the "dumpable" attribute may be reset, as described in prctl(2). Therefore, it may be desirable to reset the "dumpable" attribute *after* making any desired changes to the process's effective UID or GID.

/proc/[pid]/mountinfo (since Linux 2.6.26)
This file contains information about mounts in the

process's mount namespace (see mount namespaces(7)). It supplies various information (e.g., propagation state, root of mount for bind mounts, identifier for each mount and its parent) that is missing from the (older) */proc/[pid]/mounts* file, and fixes various other problems with that file (e.g., nonextensibility, failure to distinguish per-mount versus per-superblock options).

The file contains lines of the form:

```
36 35 98:0 /mnt1 /mnt2 rw,noatime master:1 - ext3 /dev/root
rw,errors=continue
(1)(2)(3)   (4)   (5)      (6)      (7)   (8) (9)   (10)          (11)
```

The numbers in parentheses are labels for the descriptions below:

(1) mount ID: a unique ID for the mount (may be reused after umount(2)).

(2) parent ID: the ID of the parent mount (or of self for the root of this mount namespace's mount tree).

If a new mount is stacked on top of a previous existing mount (so that it hides the existing mount) at pathname P, then the parent of the new mount is the previous mount at that location. Thus, when looking at all the mounts stacked at a particular location, the top-most mount is the one that is not the parent of any other mount at the same location. (Note, however, that this top-most mount will be accessible only if the longest path subprefix of P that is a mount point is not itself hidden by a stacked mount.)

If the parent mount lies outside the process's root directory (see chroot(2)), the ID shown here won't have a corresponding record in *mountinfo* whose mount ID (field 1) matches this parent mount ID (because mounts that lie outside the process's root directory are not shown in *mountinfo*). As a special case of this point, the process's root mount may have a parent mount (for the initramfs filesystem) that lies outside the process's root directory, and an entry for that mount will not appear in *mountinfo*.

(3) major:minor: the value of *st_dev* for files on this filesystem (see stat(2)).

(4) root: the pathname of the directory in the filesystem which forms the root of this mount.

(5) mount point: the pathname of the mount point relative to the process's root directory.

(6) mount options: per-mount options (see mount(2)).

(7) optional fields: zero or more fields of the form "tag[:value]"; see below.

(8) separator: the end of the optional fields is marked by a single hyphen.

(9) filesystem type: the filesystem type in the form "type[.subtype]".

(10) mount source: filesystem-specific information or "none".

(11) super options: per-superblock options (see mount(2)).

Currently, the possible optional fields are *shared*, *master*, *propagate_from*, and *unbindable*. See mount_namespaces(7) for a description of these fields. Parsers should ignore all unrecognized optional fields.

For more information on mount propagation see: *Documentation/filesystems/sharedsubtree.txt* in the Linux kernel source tree.

/proc/[pid]/mounts (since Linux 2.4.19)
This file lists all the filesystems currently mounted in the process's mount namespace (see mount_namespaces(7)). The format of this file is documented in fstab(5).

Since kernel version 2.6.15, this file is pollable: after opening the file for reading, a change in this file (i.e., a filesystem mount or unmount) causes select(2) to mark the file descriptor as having an exceptional condition, and poll(2) and epoll_wait(2) mark the file as having a priority event (**POLLPRI**). (Before Linux 2.6.30, a change in this file was indicated by the file descriptor being marked as readable for select(2), and being marked as having an error condition for poll(2) and epoll_wait(2).)

/proc/[pid]/mountstats (since Linux 2.6.17)
This file exports information (statistics, configuration information) about the mounts in the process's mount namespace (see mount_namespaces(7)). Lines in this file have the form:

```
device /dev/sda7 mounted on /home with fstype ext3 [stats]
(       1      )            ( 2 )             (3 ) (  4  )
```

The fields in each line are:

(1) The name of the mounted device (or "nodevice" if there is no corresponding device).

(2) The mount point within the filesystem tree.

(3) The filesystem type.

(4) Optional statistics and configuration information. Currently (as at Linux 2.6.26), only NFS filesystems export information via this field.

This file is readable only by the owner of the process.

/proc/[pid]/ns/ (since Linux 3.0)
This is a subdirectory containing one entry for each namespace that supports being manipulated by setns(2). For more information, see namespaces(7).

/proc/[pid]/root
UNIX and Linux support the idea of a per-process root of

the filesystem, set by the chroot(2) system call. This file is a symbolic link that points to the process's root directory, and behaves in the same way as *exe*, and *fd/**.

Note however that this file is not merely a symbolic link. It provides the same view of the filesystem (including namespaces and the set of per-process mounts) as the process itself. An example illustrates this point. In one terminal, we start a shell in new user and mount namespaces, and in that shell we create some new mounts:

```
$ PS1='sh1# ' unshare -Urnm
sh1# mount -t tmpfs tmpfs /etc  # Mount empty tmpfs at /etc
sh1# mount --bind /usr /dev     # Mount /usr at /dev
sh1# echo $$
27123
```

In a second terminal window, in the initial mount namespace, we look at the contents of the corresponding mounts in the initial and new namespaces:

```
$ PS1='sh2# ' sudo sh
sh2# ls /etc | wc -l                  # In initial NS
309
sh2# ls /proc/27123/root/etc | wc -l  # /etc in other NS
0                                     # The empty tmpfs dir
sh2# ls /dev | wc -l                  # In initial NS
205
sh2# ls /proc/27123/root/dev | wc -l  # /dev in other NS
11                                    # Actually bind
                                      # mounted to /usr
sh2# ls /usr | wc -l                  # /usr in initial NS
11
```

In a multithreaded process, the contents of the */proc/[pid]/root* symbolic link are not available if the main thread has already terminated (typically by calling pthread_exit(3)).

Permission to dereference or read (readlink(2)) this symbolic link is governed by a ptrace access mode **PTRACE_MODE_READ_FSCREDS** check; see ptrace(2).

/proc/mounts
Before kernel 2.4.19, this file was a list of all the filesystems currently mounted on the system. With the introduction of per-process mount namespaces in Linux 2.4.19 (see mount_namespaces(7)), this file became a link to */proc/self/mounts*, which lists the mounts of the process's own mount namespace. The format of this file is documented in fstab(5).

/proc/sys/fs/mount-max (since Linux 4.9)
The value in this file specifies the maximum number of mounts that may exist in a mount namespace. The default value in this file is 100,000.

COLOPHON top

This page is part of release 5.13 of the Linux *man-pages* project.

A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.