Rocky Enterprise Linux 9.2 Manual Pages on command 'namespaces.7'

$ man namespaces.7

NAMESPACES(7) Linux Programmer's Manual NAMESPACES(7)

NAME

namespaces - overview of Linux namespaces

DESCRIPTION

A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource. Changes to the global resource are visible to other processes that are members of the name? space, but are invisible to other processes. One use of namespaces is to implement containers.

This page provides pointers to information on the various namespace types, describes the associated /proc files, and summarizes the APIs for working with namespaces.

Namespace types

The following table shows the namespace types available on Linux. The second column of the table shows the flag value that is used to specify the namespace type in various APIs. The third column identifies the manual page that provides details on the namespace type. The last col?

umn is a summary of the resources that are isolated by the namespace type.

Namespace	Flag	Page	Isolates
Cgroup	CLONE_NEWCGROUP	cgroup_namespaces(7)	Cgroup root directory
IPC	CLONE_NEWIPC	ipc_namespaces(7)	System V IPC, POSIX message queues
Network	CLONE_NEWNET	network_namespaces(7)	Network devices, stacks, ports, etc.
Mount	CLONE_NEWNS	mount_namespaces(7)	Mount points
PID	CLONE_NEWPID	pid_namespaces(7)	Process IDs
Time	CLONE_NEWTIME	time_namespaces(7)	Boot and monotonic clocks
User	CLONE_NEWUSER	user_namespaces(7)	User and group IDs
UTS	CLONE_NEWUTS	uts_namespaces(7)	Hostname and NIS domain name

The namespaces API

As well as various /proc files described below, the namespaces API in? cludes the following system calls:

clone(2)

The clone(2) system call creates a new process. If the flags argument of the call specifies one or more of the CLONE_NEW* flags listed below, then new namespaces are created for each flag, and the child process is made a member of those name? spaces. (This system call also implements a number of features unrelated to namespaces.)

setns(2)

The setns(2) system call allows the calling process to join an existing namespace. The namespace to join is specified via a file descriptor that refers to one of the /proc/[pid]/ns files described below.

unshare(2)

The unshare(2) system call moves the calling process to a new namespace. If the flags argument of the call specifies one or

more of the CLONE_NEW* flags listed below, then new namespaces are created for each flag, and the calling process is made a member of those namespaces. (This system call also implements a number of features unrelated to namespaces.)

ioctl(2)

Various ioctl(2) operations can be used to discover information about namespaces. These operations are described in ioctl_ns(2).

Creation of new namespaces using clone(2) and unshare(2) in most cases requires the CAP_SYS_ADMIN capability, since, in the new namespace, the creator will have the power to change global resources that are visible to other processes that are subsequently created in, or join the name? space. User namespaces are the exception: since Linux 3.8, no privi? lege is required to create a user namespace.

The /proc/[pid]/ns/ directory

Each process has a /proc/[pid]/ns/ subdirectory containing one entry for each namespace that supports being manipulated by setns(2):

```
$ ls -l /proc/$$/ns | awk '{print $1, $9, $10, $11}'
total 0
lrwxrwxrwx. cgroup -> cgroup:[4026531835]
lrwxrwxrwx. ipc -> ipc:[4026531839]
lrwxrwxrwx. mnt -> mnt:[4026531840]
lrwxrwxrwx. net -> net:[4026531969]
lrwxrwxrwx. pid -> pid:[4026531836]
lrwxrwxrwx. pid_for_children -> pid:[4026531834]
lrwxrwxrwx. time -> time:[4026531834]
lrwxrwxrwx. time_for_children -> time:[4026531834]
lrwxrwxrwx. user -> user:[4026531837]
lrwxrwxrwx. uts -> uts:[4026531838]
```

Bind mounting (see mount(2)) one of the files in this directory to somewhere else in the filesystem keeps the corresponding namespace of the process specified by pid alive even if all processes currently in the namespace terminate.

Opening one of the files in this directory (or a file that is bind mounted to one of these files) returns a file handle for the corre? sponding namespace of the process specified by pid. As long as this file descriptor remains open, the namespace will remain alive, even if all processes in the namespace terminate. The file descriptor can be passed to setns(2).

In Linux 3.7 and earlier, these files were visible as hard links. Since Linux 3.8, they appear as symbolic links. If two processes are in the same namespace, then the device IDs and inode numbers of their /proc/[pid]/ns/xxx symbolic links will be the same; an application can check this using the stat.st_dev and stat.st_ino fields returned by stat(2). The content of this symbolic link is a string containing the namespace type and inode number as in the following example:

```
$ readlink /proc/$$/ns/uts
uts:[4026531838]
```

The symbolic links in this subdirectory are as follows:

/proc/[pid]/ns/cgroup (since Linux 4.6)

This file is a handle for the cgroup namespace of the process.

/proc/[pid]/ns/ipc (since Linux 3.0)

This file is a handle for the IPC namespace of the process.

/proc/[pid]/ns/mnt (since Linux 3.8)

This file is a handle for the mount namespace of the process.

/proc/[pid]/ns/net (since Linux 3.0)

This file is a handle for the network namespace of the process.

/proc/[pid]/ns/pid (since Linux 3.8)

This file is a handle for the PID namespace of the process. This handle is permanent for the lifetime of the process (i.e., a process's PID namespace membership never changes).

/proc/[pid]/ns/pid_for_children (since Linux 4.12)

This file is a handle for the PID namespace of child processes created by this process. This can change as a consequence of calls to unshare(2) and setns(2) (see pid_namespaces(7)), so the file may differ from /proc/[pid]/ns/pid. The symbolic link

gains a value only after the first child process is created in the namespace. (Beforehand, readlink(2) of the symbolic link will return an empty buffer.)

/proc/[pid]/ns/time (since Linux 5.6)

This file is a handle for the time namespace of the process.

/proc/[pid]/ns/time_for_children (since Linux 5.6)

This file is a handle for the time namespace of child processes created by this process. This can change as a consequence of calls to unshare(2) and setns(2) (see time_namespaces(7)), so the file may differ from /proc/[pid]/ns/time.

/proc/[pid]/ns/user (since Linux 3.8)

This file is a handle for the user namespace of the process.

/proc/[pid]/ns/uts (since Linux 3.0)

This file is a handle for the UTS namespace of the process.

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

The /proc/sys/user directory

The files in the /proc/sys/user directory (which is present since Linux 4.9) expose limits on the number of namespaces of various types that can be created. The files are as follows:

max_cgroup_namespaces

The value in this file defines a per-user limit on the number of cgroup namespaces that may be created in the user namespace.

max_ipc_namespaces

The value in this file defines a per-user limit on the number of ipc namespaces that may be created in the user namespace.

max_mnt_namespaces

The value in this file defines a per-user limit on the number of mount namespaces that may be created in the user namespace.

max_net_namespaces

The value in this file defines a per-user limit on the number of network namespaces that may be created in the user namespace.

max_pid_namespaces

The value in this file defines a per-user limit on the number of PID namespaces that may be created in the user namespace.

max_time_namespaces (since Linux 5.7)

The value in this file defines a per-user limit on the number of time namespaces that may be created in the user namespace.

max_user_namespaces

The value in this file defines a per-user limit on the number of user namespaces that may be created in the user namespace.

max_uts_namespaces

The value in this file defines a per-user limit on the number of uts namespaces that may be created in the user namespace.

Note the following details about these files:

* The values in these files are modifiable by privileged processes.
* The values exposed by these files are the limits for the user name? space in which the opening process resides.
* The limits are per-user. Each user in the same user namespace can create namespaces up to the defined limit.
* The limits apply to all users, including UID 0.
* These limits apply in addition to any other per-namespace limits (such as those for PID and user namespaces) that may be enforced.
* Upon encountering these limits, clone(2) and unshare(2) fail with the error ENOSPC.
* For the initial user namespace, the default value in each of these files is half the limit on the number of threads that may be created (/proc/sys/kernel/threads-max). In all descendant user namespaces, the default value in each file is MAXINT.
* When a namespace is created, the object is also accounted against ancestor namespaces. More precisely:
  + Each user namespace has a creator UID.
  + When a namespace is created, it is accounted against the creator UIDs in each of the ancestor user namespaces, and the kernel en? sures that the corresponding namespace limit for the creator UID

in the ancestor namespace is not exceeded.

+ The aforementioned point ensures that creating a new user name? space cannot be used as a means to escape the limits in force in the current user namespace.

Namespace lifetime

Absent any other factors, a namespace is automatically torn down when the last process in the namespace terminates or leaves the namespace. However, there are a number of other factors that may pin a namespace into existence even though it has no member processes. These factors include the following:

* An open file descriptor or a bind mount exists for the corresponding /proc/[pid]/ns/* file.
* The namespace is hierarchical (i.e., a PID or user namespace), and has a child namespace.
* It is a user namespace that owns one or more nonuser namespaces.
* It is a PID namespace, and there is a process that refers to the namespace via a /proc/[pid]/ns/pid_for_children symbolic link.
* It is a time namespace, and there is a process that refers to the namespace via a /proc/[pid]/ns/time_for_children symbolic link.
* It is an IPC namespace, and a corresponding mount of an mqueue filesystem (see mq_overview(7)) refers to this namespace.
* It is a PID namespace, and a corresponding mount of a proc(5) filesystem refers to this namespace.

EXAMPLES

See clone(2) and user_namespaces(7).

SEE ALSO

nsenter(1), readlink(1), unshare(1), clone(2), ioctl_ns(2), setns(2), unshare(2), proc(5), capabilities(7), cgroup_namespaces(7), cgroups(7), credentials(7), ipc_namespaces(7), network_namespaces(7), pid_name? spaces(7), user_namespaces(7), uts_namespaces(7), lsns(8), pam_name? space(8), switch_root(8)

COLOPHON

This page is part of release 5.10 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/.