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 namespace, but are invisible to other processes. One use of namespaces is to implement containers.

Linux provides the following namespaces:

Namespace	**Constant**	**Isolates**
Cgroup	**CLONE_NEWCGROUP**	Cgroup root directory
IPC	**CLONE_NEWIPC**	System V IPC, POSIX message queues
Network	**CLONE_NEWNET**	Network devices, stacks, ports, etc.
Mount	**CLONE_NEWNS**	Mount points
PID	**CLONE_NEWPID**	Process IDs
User	**CLONE_NEWUSER**	User and group IDs
UTS	**CLONE_NEWUTS**	Hostname and NIS domain name

This page describes the various namespaces and the associated */proc* files, and summarizes the APIs for working with namespaces.

The namespaces API

As well as various */proc* files described below, the namespaces API includes 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 namespaces. (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 namespace. User namespaces are the exception: since Linux 3.8, no privilege 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
total 0
lrwxrwxrwx. 1 mtk mtk 0 Apr 28 12:46 cgroup -> cgroup:[4026531835]
lrwxrwxrwx. 1 mtk mtk 0 Apr 28 12:46 ipc -> ipc:[4026531839]
lrwxrwxrwx. 1 mtk mtk 0 Apr 28 12:46 mnt -> mnt:[4026531840]
lrwxrwxrwx. 1 mtk mtk 0 Apr 28 12:46 net -> net:[4026531969]
lrwxrwxrwx. 1 mtk mtk 0 Apr 28 12:46 pid -> pid:[4026531836]
lrwxrwxrwx. 1 mtk mtk 0 Apr 28 12:46 pid_for_children -> pid:[4026531834]
lrwxrwxrwx. 1 mtk mtk 0 Apr 28 12:46 user -> user:[4026531837]
lrwxrwxrwx. 1 mtk mtk 0 Apr 28 12:46 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 corresponding 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/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_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 namespace 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 ensures 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 namespace cannot be used as a means to escape the limits in force in the current user namespace.

Cgroup namespaces (CLONE_NEWCGROUP)

See **cgroup_namespaces**(7).

IPC namespaces (CLONE_NEWIPC)

IPC namespaces isolate certain IPC resources, namely, System V IPC objects (see **sysvipc**(7)) and (since Linux 2.6.30) POSIX message queues (see **mq_overview**(7)). The common characteristic of these IPC mechanisms is that IPC objects are identified by mechanisms other than filesystem pathnames.

Each IPC namespace has its own set of System V IPC identifiers and its own POSIX message queue filesystem. Objects created in an IPC namespace are visible to all other processes that are members of that namespace, but are not visible to processes in other IPC namespaces.

The following */proc* interfaces are distinct in each IPC namespace:

* The POSIX message queue interfaces in */proc/sys/fs/mqueue*.
* The System V IPC interfaces in */proc/sys/kernel*, namely: *msgmax*, *msgmnb*, *msgmni*, *sem*, *shmall*, *shmmax*, *shmmni*, and *shm_rmid_forced*.
* The System V IPC interfaces in */proc/sysvipc*.

When an IPC namespace is destroyed (i.e., when the last process that is a member of the namespace terminates), all IPC objects in the namespace are automatically destroyed.

Use of IPC namespaces requires a kernel that is configured with the **CONFIG_IPC_NS** option.

Network namespaces (CLONE_NEWNET)

See **network_namespaces**(7).

Mount namespaces (CLONE_NEWNS)

See **mount_namespaces**(7).

PID namespaces (CLONE_NEWPID)

See **pid_namespaces**(7).

User namespaces (CLONE_NEWUSER)

See **user_namespaces**(7).

UTS namespaces (CLONE_NEWUTS)

UTS namespaces provide isolation of two system identifiers: the hostname and the NIS domain name. These identifiers are set using **sethostname**(2) and **setdomainname**(2), and can be retrieved using **uname**(2), **gethostname**(2), and **getdomainname**(2).

When a process creates a new UTS namespace using **clone**(2) or **unshare**(2) with the **CLONE_NEWUTS** flag, the hostname and domain of the new UTS namespace are copied from the corresponding values in the caller's UTS namespace.

Use of UTS namespaces requires a kernel that is configured with the **CONFIG_UTS_NS** option.

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 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.

EXAMPLE

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), **network_namespaces**(7), **pid_namespaces**(7), **user_namespaces**(7), **lsns**(8), **pam_namespace**(8), **switch_root**(8)

COLOPHON

This page is part of release 5.02 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/.