## 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.)

Creation of new namespaces using clone(2) and unshare(2) in most cases requires the **CAP_SYS_ADMIN** capability. 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 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 inode numbers of their */proc/[pid]/ns/xxx* symbolic links will be the same; an application can check this using the *stat.st_ino* field 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.

*/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).

## Cgroup namespaces (CLONE_NEWCGROUP)

See cgroup_namespaces(7).

## IPC namespaces (CLONE_NEWIPC)

IPC namespaces isolate certain IPC resources, namely, System V IPC objects (see svipc(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)

Network namespaces provide isolation of the system resources associated with networking: network devices, IPv4 and IPv6 protocol stacks, IP routing tables, firewalls, the */proc/net* directory, the */sys/class/net* directory, port numbers (sockets), and so on. A physical network device can live in exactly one network namespace. A virtual network device ("veth") pair provides a pipe-like abstraction that can be used to create tunnels between network namespaces, and can be used to create a bridge to a physical network device in another namespace.

When a network namespace is freed (i.e., when the last process in the namespace terminates), its physical network devices are moved back to the initial network namespace (not to the parent of the process).

Use of network namespaces requires a kernel that is configured with the **CONFIG_NET_NS** option.

### 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).

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

## EXAMPLE

See 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), pid_namespaces(7), user_namespaces(7), ip-netns(8), lsns(8), switch_root(8)

## COLOPHON

This page is part of release 4.10 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/.