NAME

setns − reassociate thread with a namespace

SYNOPSIS

```
#define _GNU_SOURCE         /* See feature_test_macros(7) */
#include <sched.h>

int setns(int fd, int nstype);
```

DESCRIPTION

Given a file descriptor referring to a namespace, reassociate the calling thread with that namespace.

The *fd* argument is a file descriptor referring to one of the namespace entries in a */proc/[pid]/ns/* directory; see **namespaces**(7) for further information on */proc/[pid]/ns/*. The calling thread will be reassociated with the corresponding namespace, subject to any constraints imposed by the *nstype* argument.

The *nstype* argument specifies which type of namespace the calling thread may be reassociated with. This argument can have one of the following values:

0 Allow any type of namespace to be joined.

CLONE_NEWCGROUP (since Linux 4.6)
fd must refer to a cgroup namespace.

CLONE_NEWIPC (since Linux 3.0)
fd must refer to an IPC namespace.

CLONE_NEWNET (since Linux 3.0)
fd must refer to a network namespace.

CLONE_NEWNS (since Linux 3.8)
fd must refer to a mount namespace.

CLONE_NEWPID (since Linux 3.8)
fd must refer to a descendant PID namespace.

CLONE_NEWUSER (since Linux 3.8)
fd must refer to a user namespace.

CLONE_NEWUTS (since Linux 3.0)
fd must refer to a UTS namespace.

Specifying *nstype* as 0 suffices if the caller knows (or does not care) what type of namespace is referred to by *fd*. Specifying a nonzero value for *nstype* is useful if the caller does not know what type of namespace is referred to by *fd* and wants to ensure that the namespace is of a particular type. (The caller might not know the type of the namespace referred to by *fd* if the file descriptor was opened by another process and, for example, passed to the caller via a UNIX domain socket.)

Details for specific namespace types

Note the following details and restrictions when reassociating with specific namespace types:

User namespaces

A process reassociating itself with a user namespace must have the **CAP_SYS_ADMIN** capability in the target user namespace. (This necessarily implies that it is only possible to join a descendant user namespace.) Upon successfully joining a user namespace, a process is granted all capabilities in that namespace, regardless of its user and group IDs.

A multithreaded process may not change user namespace with **setns**().

It is not permitted to use **setns**() to reenter the caller's current user namespace. This prevents a caller that has dropped capabilities from regaining those capabilities via a call to **setns**().

For security reasons, a process can't join a new user namespace if it is sharing filesystem-related attributes (the attributes whose sharing is controlled by the **clone**(2) **CLONE_FS** flag) with another process.

For further details on user namespaces, see **user_namespaces**(7).

Mount namespaces

Changing the mount namespace requires that the caller possess both **CAP_SYS_CHROOT** and **CAP_SYS_ADMIN** capabilities in its own user namespace and **CAP_SYS_ADMIN** in the user namespace that owns the target mount namespace.

A process may not be reassociated with a new mount namespace if it is multithreaded.

See **user_namespaces**(7) for details on the interaction of user namespaces and mount namespaces.

PID namespaces

In order to reassociate itself with a new PID namespace, the caller must have the **CAP_SYS_ADMIN** capability both in its own user namespace and in the user namespace that owns the target PID namespace.

If *fd* refers to a PID namespace, the semantics are somewhat different from other namespace types: reassociating the calling thread with a PID namespace changes only the PID namespace that subsequently created child processes of the caller will be placed in; it does not change the PID namespace of the caller itself.

Reassociating with a PID namespace is allowed only if the PID namespace specified by *fd* is a descendant (child, grandchild, etc.) of the PID namespace of the caller.

For further details on PID namespaces, see **pid_namespaces**(7).

Cgroup namespaces

In order to reassociate itself with a new cgroup namespace, the caller must have the **CAP_SYS_ADMIN** capability both in its own user namespace and in the user namespace that owns the target cgroup namespace.

Using **setns**() to change the caller's cgroup namespace does not change the caller's cgroup memberships.

Network, IPC, and UTS namespaces

In order to reassociate itself with a new network, IPC, or UTS namespace, the caller must have the **CAP_SYS_ADMIN** capability both in its own user namespace and in the user namespace that owns the target namespace.

RETURN VALUE

On success, **setns**() returns 0. On failure, −1 is returned and *errno* is set to indicate the error.

ERRORS

EBADF

fd is not a valid file descriptor.

EINVAL

fd refers to a namespace whose type does not match that specified in *nstype*.

EINVAL

There is problem with reassociating the thread with the specified namespace.

EINVAL

The caller tried to join an ancestor (parent, grandparent, and so on) PID namespace.

EINVAL

The caller attempted to join the user namespace in which it is already a member.

EINVAL

The caller shares filesystem (**CLONE_FS**) state (in particular, the root directory) with other processes and tried to join a new user namespace.

EINVAL

The caller is multithreaded and tried to join a new user namespace.

ENOMEM
Cannot allocate sufficient memory to change the specified namespace.

EPERM
The calling thread did not have the required capability for this operation.

VERSIONS

The **setns**() system call first appeared in Linux in kernel 3.0; library support was added to glibc in version 2.14.

CONFORMING TO

The **setns**() system call is Linux-specific.

NOTES

Not all of the attributes that can be shared when a new thread is created using **clone**(2) can be changed using **setns**().

EXAMPLE

The program below takes two or more arguments. The first argument specifies the pathname of a namespace file in an existing */proc/[pid]/ns/* directory. The remaining arguments specify a command and its arguments. The program opens the namespace file, joins that namespace using **setns**(), and executes the specified command inside that namespace.

The following shell session demonstrates the use of this program (compiled as a binary named *ns_exec*) in conjunction with the **CLONE_NEWUTS** example program in the **clone**(2) man page (complied as a binary named *newuts*).

We begin by executing the example program in **clone**(2) in the background. That program creates a child in a separate UTS namespace. The child changes the hostname in its namespace, and then both processes display the hostnames in their UTS namespaces, so that we can see that they are different.

```
$ su                    # Need privilege for namespace operations
Password:
# ./newuts bizarro &
[1] 3549
clone() returned 3550
uts.nodename in child:  bizarro
uts.nodename in parent: antero
# uname -n              # Verify hostname in the shell
antero
```

We then run the program shown below, using it to execute a shell. Inside that shell, we verify that the hostname is the one set by the child created by the first program:

```
# ./ns_exec /proc/3550/ns/uts /bin/bash
# uname -n              # Executed in shell started by ns_exec
bizarro
```

Program source

```
#define _GNU_SOURCE
#include <fcntl.h>
#include <sched.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

#define errExit(msg)    do { perror(msg); exit(EXIT_FAILURE); \
                        } while (0)

int
main(int argc, char *argv[])
```

```
{
    int fd;

    if (argc < 3) {
        fprintf(stderr, "%s /proc/PID/ns/FILE cmd args...\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    fd = open(argv[1], O_RDONLY); /* Get file descriptor for namespace */
    if (fd == -1)
        errExit("open");

    if (setns(fd, 0) == -1)       /* Join that namespace */
        errExit("setns");

    execvp(argv[2], &argv[2]);    /* Execute a command in namespace */
    errExit("execvp");
}
```

SEE ALSO

nsenter(1), **clone**(2), **fork**(2), **unshare**(2), **vfork**(2), **namespaces**(7), **unix**(7)

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/.