

**NAME**

ioctl\_ns – ioctl() operations for Linux namespaces

**DESCRIPTION****Discovering namespace relationships**

The following **ioctl(2)** operations are provided to allow discovery of namespace relationships (see **user\_namespaces(7)** and **pid\_namespaces(7)**). The form of the calls is:

```
new_fd = ioctl(fd, request);
```

In each case, *fd* refers to a */proc/pid/ns/\** file. Both operations return a new file descriptor on success.

**NS\_GET\_USERSNS** (since Linux 4.9)

Returns a file descriptor that refers to the owning user namespace for the namespace referred to by *fd*.

**NS\_GET\_PARENT** (since Linux 4.9)

Returns a file descriptor that refers to the parent namespace of the namespace referred to by *fd*. This operation is valid only for hierarchical namespaces (i.e., PID and user namespaces). For user namespaces, **NS\_GET\_PARENT** is synonymous with **NS\_GET\_USERSNS**.

The new file descriptor returned by these operations is opened with the **O\_RDONLY** and **O\_CLOEXEC** (close-on-exec; see **fcntl(2)**) flags.

By applying **fstat(2)** to the returned file descriptor, one obtains a *stat* structure whose *st\_dev* (resident device) and *st\_ino* (inode number) fields together identify the owning/parent namespace. This inode number can be matched with the inode number of another */proc/pid/ns/{pid,user}* file to determine whether that is the owning/parent namespace.

Either of these **ioctl(2)** operations can fail with the following errors:

**EPERM**

The requested namespace is outside of the caller's namespace scope. This error can occur if, for example, the owning user namespace is an ancestor of the caller's current user namespace. It can also occur on attempts to obtain the parent of the initial user or PID namespace.

**ENOTTY**

The operation is not supported by this kernel version.

Additionally, the **NS\_GET\_PARENT** operation can fail with the following error:

**EINVAL**

*fd* refers to a nonhierarchical namespace.

See the **EXAMPLES** section for an example of the use of these operations.

**Discovering the namespace type**

The **NS\_GET\_NSTYPE** operation (available since Linux 4.11) can be used to discover the type of namespace referred to by the file descriptor *fd*:

```
nstype = ioctl(fd, NS_GET_NSTYPE);
```

*fd* refers to a */proc/pid/ns/\** file.

The return value is one of the **CLONE\_NEW\*** values that can be specified to **clone(2)** or **unshare(2)** in order to create a namespace.

**Discovering the owner of a user namespace**

The **NS\_GET\_OWNER\_UID** operation (available since Linux 4.11) can be used to discover the owner user ID of a user namespace (i.e., the effective user ID of the process that created the user namespace). The form of the call is:

```
uid_t uid;
ioctl(fd, NS_GET_OWNER_UID, &uid);
```

*fd* refers to a */proc/pid/ns/user* file.

The owner user ID is returned in the *uid\_t* pointed to by the third argument.

This operation can fail with the following error:

#### **EINVAL**

*fd* does not refer to a user namespace.

### **ERRORS**

Any of the above **ioctl()** operations can return the following errors:

#### **ENOTTY**

*fd* does not refer to a */proc/[pid]/ns/\** file.

### **STANDARDS**

Namespaces and the operations described on this page are a Linux-specific.

### **EXAMPLES**

The example shown below uses the **ioctl(2)** operations described above to perform simple discovery of namespace relationships. The following shell sessions show various examples of the use of this program.

Trying to get the parent of the initial user namespace fails, since it has no parent:

```
$ ./ns_show /proc/self/ns/user p
The parent namespace is outside your namespace scope
```

Create a process running **sleep(1)** that resides in new user and UTS namespaces, and show that the new UTS namespace is associated with the new user namespace:

```
$ unshare -Uu sleep 1000 &
[1] 23235
$ ./ns_show /proc/23235/ns/uts u
Device/Inode of owning user namespace is: [0,3] / 4026532448
$ readlink /proc/23235/ns/user
user:[4026532448]
```

Then show that the parent of the new user namespace in the preceding example is the initial user namespace:

```
$ readlink /proc/self/ns/user
user:[4026531837]
$ ./ns_show /proc/23235/ns/user p
Device/Inode of parent namespace is: [0,3] / 4026531837
```

Start a shell in a new user namespace, and show that from within this shell, the parent user namespace can't be discovered. Similarly, the UTS namespace (which is associated with the initial user namespace) can't be discovered.

```
$ PS1="sh2$ " unshare -U bash
sh2$ ./ns_show /proc/self/ns/user p
The parent namespace is outside your namespace scope
sh2$ ./ns_show /proc/self/ns/uts u
The owning user namespace is outside your namespace scope
```

### **Program source**

```
/* ns_show.c

Licensed under the GNU General Public License v2 or later.
*/
#include <errno.h>
#include <fcntl.h>
#include <linux/nsfs.h>
#include <stdint.h>
```

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/sysmacros.h>
#include <unistd.h>

int
main(int argc, char *argv[])
{
    int          fd, userns_fd, parent_fd;
    struct stat  sb;

    if (argc < 2) {
        fprintf(stderr, "Usage: %s /proc/[pid]/ns/[file] [p|u]\n",
            argv[0]);
        fprintf(stderr, "\nDisplay the result of one or both "
            "of NS_GET_USERNS (u) or NS_GET_PARENT (p)\n"
            "for the specified /proc/[pid]/ns/[file]. If neither "
            "'p' nor 'u' is specified,\n"
            "NS_GET_USERNS is the default.\n");
        exit(EXIT_FAILURE);
    }

    /* Obtain a file descriptor for the 'ns' file specified
       in argv[1]. */

    fd = open(argv[1], O_RDONLY);
    if (fd == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }

    /* Obtain a file descriptor for the owning user namespace and
       then obtain and display the inode number of that namespace. */

    if (argc < 3 || strchr(argv[2], 'u')) {
        userns_fd = ioctl(fd, NS_GET_USERNS);

        if (userns_fd == -1) {
            if (errno == EPERM)
                printf("The owning user namespace is outside "
                    "your namespace scope\n");
            else
                perror("ioctl-NS_GET_USERNS");
            exit(EXIT_FAILURE);
        }

        if (fstat(userns_fd, &sb) == -1) {
            perror("fstat-userns");
            exit(EXIT_FAILURE);
        }

        printf("Device/Inode of owning user namespace is: "

```

```

        "[%x,%x] / %ju\n",
        major(sb.st_dev),
        minor(sb.st_dev),
        (uintmax_t) sb.st_ino);

    close(userns_fd);
}

/* Obtain a file descriptor for the parent namespace and
   then obtain and display the inode number of that namespace. */

if (argc > 2 && strchr(argv[2], 'p')) {
    parent_fd = ioctl(fd, NS_GET_PARENT);

    if (parent_fd == -1) {
        if (errno == EINVAL)
            printf("Can' get parent namespace of a "
                   "nonhierarchical namespace\n");
        else if (errno == EPERM)
            printf("The parent namespace is outside "
                   "your namespace scope\n");
        else
            perror("ioctl-NS_GET_PARENT");
        exit(EXIT_FAILURE);
    }

    if (fstat(parent_fd, &sb) == -1) {
        perror("fstat-parentns");
        exit(EXIT_FAILURE);
    }
    printf("Device/Inode of parent namespace is: [%x,%x] / %ju\n",
           major(sb.st_dev),
           minor(sb.st_dev),
           (uintmax_t) sb.st_ino);

    close(parent_fd);
}

exit(EXIT_SUCCESS);
}

```

**SEE ALSO**

**fstat(2), ioctl(2), proc(5), namespaces(7)**