NAME

capget, capset – set/get capabilities of thread(s)

SYNOPSIS

#include <sys/capability.h>

int capget(cap_user_header_t *hdrp***, cap_user_data_t** *datap***);**

int capset(cap_user_header_t *hdrp***, const cap_user_data_t** *datap***);**

DESCRIPTION

Since Linux 2.2, the power of the superuser (root) has been partitioned into a set of discrete capabilities. Each thread has a set of effective capabilities identifying which capabilities (if any) it may currently exercise. Each thread also has a set of inheritable capabilities that may be passed through an execve(2) call, and a set of permitted capabilities that it can make effective or inheritable.

These two system calls are the raw kernel interface for getting and setting thread capabilities. Not only are these system calls specific to Linux, but the kernel API is likely to change and use of these system calls (in particular the format of the *cap_user_*_t* types) is subject to extension with each kernel revision, but old programs will keep working.

The portable interfaces are **cap_set_proc(3)** and **cap_get_proc(3);** if possible, you should use those interfaces in applications. If you wish to use the Linux extensions in applications, you should use the easier-to-use interfaces **capsetp(3)** and **capgetp(3).**

Current details

Now that you have been warned, some current kernel details. The structures are defined as follows.

```
#define _LINUX_CAPABILITY_VERSION_1  0x19980330
#define _LINUX_CAPABILITY_U32S_1     1

/* V2 added in Linux 2.6.25; deprecated */
#define _LINUX_CAPABILITY_VERSION_2  0x20071026
#define _LINUX_CAPABILITY_U32S_2     2

/* V3 added in Linux 2.6.26 */
#define _LINUX_CAPABILITY_VERSION_3  0x20080522
#define _LINUX_CAPABILITY_U32S_3     2

typedef struct __user_cap_header_struct {
__u32 version;
int pid;
} *cap_user_header_t;

typedef struct __user_cap_data_struct {
__u32 effective;
__u32 permitted;
__u32 inheritable;
} *cap_user_data_t;
```

The *effective*, *permitted*, and *inheritable* fields are bit masks of the capabilities defined in capabilities(7). Note that the **CAP_*** values are bit indexes and need to be bit-shifted before ORing into the bit fields. To define the structures for passing to the system call, you have to use the *struct __user_cap_header_struct* and *struct __user_cap_data_struct* names because the typedefs are only pointers.

Kernels prior to 2.6.25 prefer 32-bit capabilities with version **_LINUX_CAPABILITY_VERSION_1**. Linux 2.6.25 added 64-bit capability sets, with version **_LINUX_CAPABILITY_VERSION_2**. There was, however, an API glitch, and Linux 2.6.26 added **_LINUX_CAPABILITY_VERSION_3** to fix the problem.

Note that 64-bit capabilities use *datap*[0] and *datap*[1], whereas 32-bit capabilities use only *datap*[0].

On kernels that support file capabilities (VFS capabilities support), these system calls behave slightly differently. This support was added as an option in Linux 2.6.24, and became fixed (nonoptional) in Linux

2.6.33.

For **capget**() calls, one can probe the capabilities of any process by specifying its process ID with the *hdrp->pid* field value.

With VFS capabilities support

VFS capabilities employ a file extended attribute (see xattr(7)) to allow capabilities to be attached to executables. This privilege model obsoletes kernel support for one process asynchronously setting the capabilities of another. That is, on kernels that have VFS capabilities support, when calling **capset**(), the only permitted values for *hdrp->pid* are 0 or, equivalently, the value returned by gettid(2).

Without VFS capabilities support

On older kernels that do not provide VFS capabilities support **capset**() can, if the caller has the **CAP_SETPCAP** capability, be used to change not only the caller's own capabilities, but also the capabilities of other threads. The call operates on the capabilities of the thread specified by the *pid* field of *hdrp* when that is nonzero, or on the capabilities of the calling thread if *pid* is 0. If *pid* refers to a single-threaded process, then *pid* can be specified as a traditional process ID; operating on a thread of a multithreaded process requires a thread ID of the type returned by gettid(2). For **capset**(), *pid* can also be: −1, meaning perform the change on all threads except the caller and init(1); or a value less than −1, in which case the change is applied to all members of the process group whose ID is −*pid*.

For details on the data, see capabilities(7).

RETURN VALUE

On success, zero is returned. On error, −1 is returned, and *errno* is set appropriately.

The calls fail with the error **EINVAL**, and set the *version* field of *hdrp* to the kernel preferred value of **_LINUX_CAPABILITY_VERSION_?** when an unsupported *version* value is specified. In this way, one can probe what the current preferred capability revision is.

ERRORS

EFAULT
Bad memory address. *hdrp* must not be NULL. *datap* may be NULL only when the user is trying to determine the preferred capability version format supported by the kernel.

EINVAL
One of the arguments was invalid.

EPERM
An attempt was made to add a capability to the Permitted set, or to set a capability in the Effective or Inheritable sets that is not in the Permitted set.

EPERM
The caller attempted to use **capset**() to modify the capabilities of a thread other than itself, but lacked sufficient privilege. For kernels supporting VFS capabilities, this is never permitted. For kernels lacking VFS support, the **CAP_SETPCAP** capability is required. (A bug in kernels before 2.6.11 meant that this error could also occur if a thread without this capability tried to change its own capabilities by specifying the *pid* field as a nonzero value (i.e., the value returned by getpid(2)) instead of 0.)

ESRCH
No such thread.

CONFORMING TO

These system calls are Linux-specific.

NOTES

The portable interface to the capability querying and setting functions is provided by the *libcap* library and is available here:
Unknown

SEE ALSO

clone(2), gettid(2), capabilities(7)

COLOPHON

This page is part of release 4.16 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/.