NAME

cap_get_file, cap_set_file, cap_get_fd, cap_set_fd, cap_get_nsowner, cap_set_nsowner − capability manipulation on files

SYNOPSIS

```
#include <sys/capability.h>

cap_t cap_get_file(const char *path_p);
int cap_set_file(const char *path_p, cap_t cap_p);
cap_t cap_get_fd(int fd);
int cap_set_fd(int fd, cap_t caps);
uid_t cap_get_nsowner(cap_t caps);
int cap_set_nsowner(cap_t caps, uid_t rootuid);
```

Link with −*lcap*.

DESCRIPTION

cap_get_file() and **cap_get_fd**() allocate a capability state in working storage and set it to represent the capability state of the pathname pointed to by *path_p* or the file open on descriptor *fd*. These functions return a pointer to the newly created capability state. The effects of reading the capability state from any file other than a regular file is undefined. The caller should free any releasable memory, when the capability state in working storage is no longer required, by calling **cap_free**() with the used *cap_t* as an argument.

cap_set_file() and **cap_set_fd**() set the values for all capability flags for all capabilities for the pathname pointed to by *path_p* or the file open on descriptor *fd*, with the capability state identified by *cap_p*. The new capability state of the file is completely determined by the contents of *cap_p*. A NULL value for *cap_p* is used to indicate that capabilities for the file should be deleted. For these functions to succeed, the calling process must have the **CAP_SETFCAP** capability in its effective set and either the effective user ID of the process must match the file owner or the calling process must have the **CAP_FOWNER** capability in its effective capability set. The effects of writing the capability state to any file type other than a regular file are undefined.

A capability set held in memory can be associated with the root user ID in use in a specific user namespace. It is possible to get and set this value (in the memory copy) with **cap_get_nsowner**() and **cap_set_nsowner**() respectively. The root user ID is ignored by the libcap library in all cases other than when the capability is written to a file. Only if the value is non-zero will the library attempt to include it in the written file capability set.

RETURN VALUE

cap_get_file() and **cap_get_fd**() return a non-NULL value on success, and NULL on failure.

cap_set_file() and **cap_set_fd**() return zero on success, and −1 on failure.

On failure, *errno* is set to **EACCES**, **EBADFD**, **ENAMETOOLONG**, **ENOENT**, **ENOMEM**, **ENOTDIR**, **EPERM**, or **EROFS**.

CONFORMING TO

These functions are specified by withdrawn POSIX.1e draft specification.

NOTES

Support for file capabilities is provided on Linux since version 2.6.24.

On Linux, the file Effective set is a single bit. If it is enabled, then all Permitted capabilities are enabled in the Effective set of the calling process when the file is executed; otherwise, no capabilities are enabled in the process's Effective set following an **execve**(2). Because the file Effective set is a single bit, if any capability is enabled in the Effective set of the *cap_t* given to **cap_set_file**() or **cap_set_fd**(), then all capabilities whose Permitted or Inheritable flag is enabled must also have the Effective flag enabled. Conversely, if the Effective bit is enabled on a file, then the *cap_t* returned by **cap_get_file()** and **cap_get_fd()** will have the Effective flag enabled for each capability that has the Permitted or Inheritable flag enabled.

SEE ALSO

libcap(3), **cap_clear**(3), **cap_copy_ext**(3), **cap_from_text**(3), **cap_get_proc**(3), **cap_init**(3), **capabilities**(7), **user_namespaces**(7)