NAME

setfsuid − set user identity used for filesystem checks

LIBRARY

Standard C library (*libc*, *−lc*)

SYNOPSIS

#include <sys/fsuid.h>

int setfsuid(uid_t *fsuid***);**

DESCRIPTION

On Linux, a process has both a filesystem user ID and an effective user ID. The (Linux-specific) filesystem user ID is used for permissions checking when accessing filesystem objects, while the effective user ID is used for various other kinds of permissions checks (see **credentials**(7)).

Normally, the value of the process's filesystem user ID is the same as the value of its effective user ID. This is so, because whenever a process's effective user ID is changed, the kernel also changes the filesystem user ID to be the same as the new value of the effective user ID. A process can cause the value of its filesystem user ID to diverge from its effective user ID by using **setfsuid**() to change its filesystem user ID to the value given in *fsuid*.

Explicit calls to **setfsuid**() and **setfsgid**(2) are (were) usually used only by programs such as the Linux NFS server that need to change what user and group ID is used for file access without a corresponding change in the real and effective user and group IDs. A change in the normal user IDs for a program such as the NFS server is (was) a security hole that can expose it to unwanted signals. (However, this issue is historical; see below.)

setfsuid() will succeed only if the caller is the superuser or if *fsuid* matches either the caller' s real user ID, effective user ID, saved set-user-ID, or current filesystem user ID.

RETURN VALUE

On both success and failure, this call returns the previous filesystem user ID of the caller.

VERSIONS

This system call is present since Linux 1.2.

STANDARDS

setfsuid() is Linux-specific and should not be used in programs intended to be portable.

NOTES

At the time when this system call was introduced, one process could send a signal to another process with the same effective user ID. This meant that if a privileged process changed its effective user ID for the purpose of file permission checking, then it could become vulnerable to receiving signals sent by another (unprivileged) process with the same user ID. The filesystem user ID attribute was thus added to allow a process to change its user ID for the purposes of file permission checking without at the same time becoming vulnerable to receiving unwanted signals. Since Linux 2.0, signal permission handling is different (see **kill**(2)), with the result that a process can change its effective user ID without being vulnerable to receiving signals from unwanted processes. Thus, **setfsuid**() is nowadays unneeded and should be avoided in new applications (likewise for **setfsgid**(2)).

The original Linux **setfsuid**() system call supported only 16-bit user IDs. Subsequently, Linux 2.4 added **setfsuid32**() supporting 32-bit IDs. The glibc **setfsuid**() wrapper function transparently deals with the variation across kernel versions.

C library/kernel differences

In glibc 2.15 and earlier, when the wrapper for this system call determines that the argument can't be passed to the kernel without integer truncation (because the kernel is old and does not support 32-bit user IDs), it will return −1 and set *errno* to **EINVAL** without attempting the system call.

BUGS

No error indications of any kind are returned to the caller, and the fact that both successful and unsuccessful calls return the same value makes it impossible to directly determine whether the call succeeded or failed.

Instead, the caller must resort to looking at the return value from a further call such as *setfsuid(−1)* (which will always fail), in order to determine if a preceding call to **setfsuid**() changed the filesystem user ID. At the very least, **EPERM** should be returned when the call fails (because the caller lacks the **CAP_SETUID** capability).

SEE ALSO

kill(2), **setfsgid**(2), **capabilities**(7), **credentials**(7)