NAME

setreuid, setregid − set real and/or effective user or group ID

LIBRARY

Standard C library (*libc*, *−lc*)

SYNOPSIS

```
#include <unistd.h>

int setreuid(uid_t ruid, uid_t euid);
int setregid(gid_t rgid, gid_t egid);
```

Feature Test Macro Requirements for glibc (see **feature_test_macros**(7)):

setreuid(), **setregid**():
_XOPEN_SOURCE >= 500
|| /* Since glibc 2.19: */ _DEFAULT_SOURCE
|| /* glibc <= 2.19: */ _BSD_SOURCE

DESCRIPTION

setreuid() sets real and effective user IDs of the calling process.

Supplying a value of −1 for either the real or effective user ID forces the system to leave that ID unchanged.

Unprivileged processes may only set the effective user ID to the real user ID, the effective user ID, or the saved set-user-ID.

Unprivileged users may only set the real user ID to the real user ID or the effective user ID.

If the real user ID is set (i.e., *ruid* is not −1) or the effective user ID is set to a value not equal to the previous real user ID, the saved set-user-ID will be set to the new effective user ID.

Completely analogously, **setregid**() sets real and effective group ID's of the calling process, and all of the above holds with "group" instead of "user".

RETURN VALUE

On success, zero is returned. On error, −1 is returned, and *errno* is set to indicate the error.

Note: there are cases where **setreuid**() can fail even when the caller is UID 0; it is a grave security error to omit checking for a failure return from **setreuid**().

ERRORS

EAGAIN
The call would change the caller's real UID (i.e., *ruid* does not match the caller's real UID), but there was a temporary failure allocating the necessary kernel data structures.

EAGAIN
ruid does not match the caller's real UID and this call would bring the number of processes belonging to the real user ID *ruid* over the caller's **RLIMIT_NPROC** resource limit. Since Linux 3.1, this error case no longer occurs (but robust applications should check for this error); see the description of **EAGAIN** in **execve**(2).

EINVAL
One or more of the target user or group IDs is not valid in this user namespace.

EPERM
The calling process is not privileged (on Linux, does not have the necessary capability in its user namespace: **CAP_SETUID** in the case of **setreuid**(), or **CAP_SETGID** in the case of **setregid**()) and a change other than (i) swapping the effective user (group) ID with the real user (group) ID, or (ii) setting one to the value of the other or (iii) setting the effective user (group) ID to the value of the saved set-user-ID (saved set-group-ID) was specified.

STANDARDS

POSIX.1-2001, POSIX.1-2008, 4.3BSD (**setreuid**() and **setregid**() first appeared in 4.2BSD).

NOTES

Setting the effective user (group) ID to the saved set-user-ID (saved set-group-ID) is possible since Linux 1.1.37 (1.1.38).

POSIX.1 does not specify all of the UID changes that Linux permits for an unprivileged process. For **setreuid**(), the effective user ID can be made the same as the real user ID or the saved set-user-ID, and it is unspecified whether unprivileged processes may set the real user ID to the real user ID, the effective user ID, or the saved set-user-ID. For **setregid**(), the real group ID can be changed to the value of the saved set-group-ID, and the effective group ID can be changed to the value of the real group ID or the saved set-group-ID. The precise details of what ID changes are permitted vary across implementations.

POSIX.1 makes no specification about the effect of these calls on the saved set-user-ID and saved set-group-ID.

The original Linux **setreuid**() and **setregid**() system calls supported only 16-bit user and group IDs. Subsequently, Linux 2.4 added **setreuid32**() and **setregid32**(), supporting 32-bit IDs. The glibc **setreuid**() and **setregid**() wrapper functions transparently deal with the variations across kernel versions.

C library/kernel differences

At the kernel level, user IDs and group IDs are a per-thread attribute. However, POSIX requires that all threads in a process share the same credentials. The NPTL threading implementation handles the POSIX requirements by providing wrapper functions for the various system calls that change process UIDs and GIDs. These wrapper functions (including those for**setr euid**() and **setregid**()) employ a signal-based technique to ensure that when one thread changes credentials, all of the other threads in the process also change their credentials. For details, see **nptl**(7).

SEE ALSO

getgid(2), **getuid**(2), **seteuid**(2), **setgid**(2), **setresuid**(2), **setuid**(2), **capabilities**(7), **credentials**(7), **user_namespaces**(7)