NAME

setresuid, setresgid − set real, effective, and saved user or group ID

LIBRARY

Standard C library (*libc*, −*lc*)

SYNOPSIS

```
#define _GNU_SOURCE         /* See feature_test_macros(7) */
#include <unistd.h>

int setresuid(uid_t ruid, uid_t euid, uid_t suid);
int setresgid(gid_t rgid, gid_t egid, gid_t sgid);
```

DESCRIPTION

setresuid() sets the real user ID, the effective user ID, and the saved set-user-ID of the calling process.

An unprivileged process may change its real UID, effective UID, and saved set-user-ID, each to one of: the current real UID, the current effective UID, or the current saved set-user-ID.

A privileged process (on Linux, one having the **CAP_SETUID** capability) may set its real UID, effective UID, and saved set-user-ID to arbitrary values.

If one of the arguments equals −1, the corresponding value is not changed.

Regardless of what changes are made to the real UID, effective UID, and saved set-user-ID, the filesystem UID is always set to the same value as the (possibly new) effective UID.

Completely analogously, **setresgid**() sets the real GID, effective GID, and saved set-group-ID of the calling process (and always modifies the filesystem GID to be the same as the effective GID), with the same restrictions for unprivileged processes.

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 **setresuid**() can fail even when the caller is UID 0; it is a grave security error to omit checking for a failure return from **setresuid**().

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 (did not have the necessary capability in its user namespace) and tried to change the IDs to values that are not permitted. For **setresuid**(), the necessary capability is **CAP_SETUID**; for **setresgid**(), it is **CAP_SETGID**.

VERSIONS

These calls are available under Linux since Linux 2.1.44.

STANDARDS

These calls are nonstandard; they also appear on HP-UX and some of the BSDs.

NOTES

Under HP-UX and FreeBSD, the prototype is found in <*unistd.h*>. Under Linux, the prototype is provided since glibc 2.3.2.

The original Linux **setresuid**() and **setresgid**() system calls supported only 16-bit user and group IDs. Subsequently, Linux 2.4 added **setresuid32**() and **setresgid32**(), supporting 32-bit IDs. The glibc **setresuid**() and **setresgid**() 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 esuid**() and **setresgid**()) 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

getresuid(2), **getuid**(2), **setfsgid**(2), **setfsuid**(2), **setreuid**(2), **setuid**(2), **capabilities**(7), **credentials**(7), **user_namespaces**(7)