Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'setuid.3p' command

$ man setuid.3p

SETUID(3P) POSIX Programmer's Manual SETUID(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setuid ? set user ID

SYNOPSIS

#include <unistd.h>

int setuid(uid_t uid);

DESCRIPTION

If the process has appropriate privileges, setuid() shall set the real user ID, effective user ID, and the saved set-user-ID of the calling process to uid.

If the process does not have appropriate privileges, but uid is equal to the real user ID or the saved set-user-ID, setuid() shall set the effective user ID to uid; the real user ID and saved set-user-ID shall remain unchanged.

The setuid() function shall not affect the supplementary group list in any way.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, -1 shall be

returned and errno set to indicate the error.

ERRORS

The setuid() function shall fail, return -1, and set errno to the cor?
responding value if one or more of the following are true:

EINVAL The value of the uid argument is invalid and not supported by the implementation.

EPERM The process does not have appropriate privileges and uid does not match the real user ID or the saved set-user-ID.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The various behaviors of the setuid() and setgid() functions when called by non-privileged processes reflect the behavior of different historical implementations. For portability, it is recommended that new non-privileged applications use the seteuid() and setegid() functions instead.

The saved set-user-ID capability allows a program to regain the effec? tive user ID established at the last exec call. Similarly, the saved set-group-ID capability allows a program to regain the effective group ID established at the last exec call. These capabilities are derived from System V. Without them, a program might have to run as superuser in order to perform the same functions, because superuser can write on the user's files. This is a problem because such a program can write on any user's files, and so must be carefully written to emulate the per? missions of the calling process properly. In System V, these capabili? ties have traditionally been implemented only via the setuid() and set? gid() functions for non-privileged processes. The fact that the behav? ior of those functions was different for privileged processes made them difficult to use. The POSIX.1?1990 standard defined the setuid() func? tion to behave differently for privileged and unprivileged users. When

the caller had appropriate privileges, the function set the real user ID, effective user ID, and saved set-user ID of the calling process on implementations that supported it. When the caller did not have appro? priate privileges, the function set only the effective user ID, subject to permission checks. The former use is generally needed for utilities like login and su, which are not conforming applications and thus out? side the scope of POSIX.1?2008. These utilities wish to change the user ID irrevocably to a new value, generally that of an unprivileged user. The latter use is needed for conforming applications that are installed with the set-user-ID bit and need to perform operations using the real user ID.

POSIX.1?2008 augments the latter functionality with a mandatory feature named _POSIX_SAVED_IDS. This feature permits a set-user-ID application to switch its effective user ID back and forth between the values of its exec-time real user ID and effective user ID. Unfortunately, the POSIX.1?1990 standard did not permit a conforming application using this feature to work properly when it happened to be executed with (im? plementation-defined) appropriate privileges. Furthermore, the applica? tion did not even have a means to tell whether it had this privilege. Since the saved set-user-ID feature is quite desirable for applica? tions, as evidenced by the fact that NIST required it in FIPS 151?2, it has been mandated by POSIX.1?2008. However, there are implementors who have been reluctant to support it given the limitation described above.

The 4.3BSD system handles the problem by supporting separate functions: setuid() (which always sets both the real and effective user IDs, like setuid() in POSIX.1?2008 for privileged users), and seteuid() (which always sets just the effective user ID, like setuid() in POSIX.1?2008 for non-privileged users). This separation of functionality into dis? tinct functions seems desirable. 4.3BSD does not support the saved set-user-ID feature. It supports similar functionality of switching the ef? fective user ID back and forth via setreuid(), which permits reversing the real and effective user IDs. This model seems less desirable than the saved set-user-ID because the real user ID changes as a side-ef?

fect. The current 4.4BSD includes saved effective IDs and uses them for seteuid() and setegid() as described above. The setreuid() and se? tregid() functions will be deprecated or removed.

The solution here is:

* Require that all implementations support the functionality of the saved set-user-ID, which is set by the exec functions and by privi? leged calls to setuid().
* Add the seteuid() and setegid() functions as portable alternatives to setuid() and setgid() for non-privileged and privileged pro? cesses.

Historical systems have provided two mechanisms for a set-user-ID process to change its effective user ID to be the same as its real user ID in such a way that it could return to the original effective user ID: the use of the setuid() function in the presence of a saved set-user-ID, or the use of the BSD setreuid() function, which was able to swap the real and effective user IDs. The changes included in POSIX.1?2008 provide a new mechanism using seteuid() in conjunction with a saved set-user-ID. Thus, all implementations with the new se? teuid() mechanism will have a saved set-user-ID for each process, and most of the behavior controlled by _POSIX_SAVED_IDS has been changed to agree with the case where the option was defined. The kill() function is an exception. Implementors of the new seteuid() mechanism will gen? erally be required to maintain compatibility with the older mechanisms previously supported by their systems. However, compatibility with this use of setreuid() and with the _POSIX_SAVED_IDS behavior of kill() is unfortunately complicated. If an implementation with a saved set-user-ID allows a process to use setreuid() to swap its real and effective user IDs, but were to leave the saved set-user-ID unmodified, the process would then have an effective user ID equal to the original real user ID, and both real and saved set-user-ID would be equal to the original effective user ID. In that state, the real user would be un? able to kill the process, even though the effective user ID of the process matches that of the real user, if the kill() behavior of

_POSIX_SAVED_IDS was used. This is obviously not acceptable. The alter?
native choice, which is used in at least one implementation, is to
change the saved set-user-ID to the effective user ID during most calls
to setreuid(). The standard developers considered that alternative to
be less correct than the retention of the old behavior of kill() in
such systems. Current conforming applications shall accommodate either
behavior from kill(), and there appears to be no strong reason for
kill() to check the saved set-user-ID rather than the effective user
ID.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, getegid(), geteuid(), getgid(), getuid(), setegid(), seteuid(),
setgid(), setregid(), setreuid()

The Base Definitions volume of POSIX.1?2017, <sys_types.h>, <unistd.h>

COPYRIGHT