Rocky Enterprise Linux 9.2 Manual Pages on command 'credentials.7'

$ man credentials.7

CREDENTIALS(7) Linux Programmer's Manual CREDENTIALS(7)

NAME

credentials - process identifiers

DESCRIPTION

Process ID (PID)

Each process has a unique nonnegative integer identifier that is assigned when the process is created using fork(2). A process can obtain its PID using getpid(2). A PID is repre? sented using the type pid_t (defined in <sys/types.h>).

PIDs are used in a range of system calls to identify the process affected by the call, for example: kill(2), ptrace(2), setpriority(2) setpgid(2), setsid(2), sigqueue(3), and wait? pid(2).

A process's PID is preserved across an execve(2).

Parent process ID (PPID)

A process's parent process ID identifies the process that created this process using fork(2). A process can obtain its PPID using getppid(2). A PPID is represented using the type pid_t.

A process's PPID is preserved across an execve(2).

Process group ID and session ID

Each process has a session ID and a process group ID, both represented using the type pid_t. A process can obtain its session ID using getsid(2), and its process group ID us? ing getpgrp(2).

A child created by fork(2) inherits its parent's session ID and process group ID. A process's session ID and process group ID are preserved across an execve(2).

Sessions and process groups are abstractions devised to support shell job control. A process group (sometimes called a "job") is a collection of processes that share the same process group ID; the shell creates a new process group for the process(es) used to exe? cute single command or pipeline (e.g., the two processes created to execute the command "ls | wc" are placed in the same process group). A process's group membership can be set using setpgid(2). The process whose process ID is the same as its process group ID is the process group leader for that group.

A session is a collection of processes that share the same session ID. All of the members of a process group also have the same session ID (i.e., all of the members of a process group always belong to the same session, so that sessions and process groups form a strict two-level hierarchy of processes.) A new session is created when a process calls set? sid(2), which creates a new session whose session ID is the same as the PID of the process that called setsid(2). The creator of the session is called the session leader.

All of the processes in a session share a controlling terminal. The controlling terminal is established when the session leader first opens a terminal (unless the O_NOCTTY flag is specified when calling open(2)). A terminal may be the controlling terminal of at most one session.

At most one of the jobs in a session may be the foreground job; other jobs in the session are background jobs. Only the foreground job may read from the terminal; when a process in the background attempts to read from the terminal, its process group is sent a SIGTTIN signal, which suspends the job. If the TOSTOP flag has been set for the terminal (see termios(3)), then only the foreground job may write to the terminal; writes from back? ground job cause a SIGTTOU signal to be generated, which suspends the job. When terminal keys that generate a signal (such as the interrupt key, normally control-C) are pressed, the signal is sent to the processes in the foreground job.

Various system calls and library functions may operate on all members of a process group, including kill(2), killpg(3), getpriority(2), setpriority(2), ioprio_get(2), io? prio_set(2), waitid(2), and waitpid(2). See also the discussion of the F_GETOWN, F_GETOWN_EX, F_SETOWN, and F_SETOWN_EX operations in fcntl(2).

User and group identifiers

Each process has various associated user and group IDs. These IDs are integers, respec? tively represented using the types uid_t and gid_t (defined in <sys/types.h>).

On Linux, each process has the following user and group identifiers:

* Real user ID and real group ID. These IDs determine who owns the process. A process can obtain its real user (group) ID using getuid(2) (getgid(2)).

* Effective user ID and effective group ID. These IDs are used by the kernel to deter? mine the permissions that the process will have when accessing shared resources such as message queues, shared memory, and semaphores. On most UNIX systems, these IDs also determine the permissions when accessing files. However, Linux uses the filesystem IDs described below for this task. A process can obtain its effective user (group) ID us? ing geteuid(2) (getegid(2)).

* Saved set-user-ID and saved set-group-ID. These IDs are used in set-user-ID and set-group-ID programs to save a copy of the corresponding effective IDs that were set when the program was executed (see execve(2)). A set-user-ID program can assume and drop privileges by switching its effective user ID back and forth between the values in its real user ID and saved set-user-ID. This switching is done via calls to seteuid(2), setreuid(2), or setresuid(2). A set-group-ID program performs the analogous tasks us? ing setegid(2), setregid(2), or setresgid(2). A process can obtain its saved set-user-ID (set-group-ID) using getresuid(2) (getresgid(2)).

* Filesystem user ID and filesystem group ID (Linux-specific). These IDs, in conjunction with the supplementary group IDs described below, are used to determine permissions for accessing files; see path_resolution(7) for details. Whenever a process's effective user (group) ID is changed, the kernel also automatically changes the filesystem user (group) ID to the same value. Consequently, the filesystem IDs normally have the same values as the corresponding effective ID, and the semantics for file-permission checks are thus the same on Linux as on other UNIX systems. The filesystem IDs can be made to differ from the effective IDs by calling setfsuid(2) and setfsgid(2).

* Supplementary group IDs. This is a set of additional group IDs that are used for per? mission checks when accessing files and other shared resources. On Linux kernels be? fore 2.6.4, a process can be a member of up to 32 supplementary groups; since kernel 2.6.4, a process can be a member of up to 65536 supplementary groups. The call sysconf(_SC_NGROUPS_MAX) can be used to determine the number of supplementary groups of which a process may be a member. A process can obtain its set of supplementary group IDs using getgroups(2).

A child process created by fork(2) inherits copies of its parent's user and groups IDs. During an execve(2), a process's real user and group ID and supplementary group IDs are preserved; the effective and saved set IDs may be changed, as described in execve(2).

Aside from the purposes noted above, a process's user IDs are also employed in a number of other contexts:

* when determining the permissions for sending signals (see kill(2));
* when determining the permissions for setting process-scheduling parameters (nice value, real time scheduling policy and priority, CPU affinity, I/O priority) using setprior? ity(2), sched_setaffinity(2), sched_setscheduler(2), sched_setparam(2), sched_se? tattr(2), and ioprio_set(2);
* when checking resource limits (see getrlimit(2));
* when checking the limit on the number of inotify instances that the process may create (see inotify(7)).

Modifying process user and group IDs

Subject to rules described in the relevant manual pages, a process can use the following APIs to modify its user and group IDs:

setuid(2) (setgid(2))

Modify the process's real (and possibly effective and saved-set) user (group) IDs.

seteuid(2) (setegid(2))

Modify the process's effective user (group) ID.

setfsuid(2) (setfsgid(2))

Modify the process's filesystem user (group) ID.

setreuid(2) (setregid(2))

Modify the process's real and effective (and possibly saved-set) user (group) IDs.

setresuid(2) (setresgid(2))

Modify the process's real, effective, and saved-set user (group) IDs.

setgroups(2)

Modify the process's supplementary group list.

Any changes to a process's effective user (group) ID are automatically carried over to the process's filesystem user (group) ID. Changes to a process's effective user or group ID can also affect the process "dumpable" attribute, as described in prctl(2).

Changes to process user and group IDs can affect the capabilities of the process, as de? scribed in capabilities(7).

CONFORMING TO

Process IDs, parent process IDs, process group IDs, and session IDs are specified in

POSIX.1. The real, effective, and saved set user and groups IDs, and the supplementary group IDs, are specified in POSIX.1. The filesystem user and group IDs are a Linux exten? sion.

NOTES

Various fields in the /proc/[pid]/status file show the process credentials described above. See proc(5) for further information.

The POSIX threads specification requires that credentials are shared by all of the threads in a process. However, at the kernel level, Linux maintains separate user and group cre? dentials for each thread. The NPTL threading implementation does some work to ensure that any change to user or group credentials (e.g., calls to setuid(2), setresuid(2)) is car? ried through to all of the POSIX threads in a process. See nptl(7) for further details.

SEE ALSO

bash(1), csh(1), groups(1), id(1), newgrp(1), ps(1), runuser(1), setpriv(1), sg(1), su(1), access(2), execve(2), faccessat(2), fork(2), getgroups(2), getpgrp(2), getpid(2), getp? pid(2), getsid(2), kill(2), setegid(2), seteuid(2), setfsgid(2), setfsuid(2), setgid(2), setgroups(2), setpgid(2), setresgid(2), setresuid(2), setsid(2), setuid(2), waitpid(2), euidaccess(3), initgroups(3), killpg(3), tcgetpgrp(3), tcgetsid(3), tcsetpgrp(3), group(5), passwd(5), shadow(5), capabilities(7), namespaces(7), path_resolution(7), pid_namespaces(7), pthreads(7), signal(7), system_data_types(7), unix(7), user_name? spaces(7), sudo(8)

COLOPHON

This page is part of release 5.10 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.