NAME

setpgid, getpgid, setpgrp, getpgrp − set/get process group

SYNOPSIS

```
#include <unistd.h>

int setpgid(pid_t pid, pid_t pgid);
pid_t getpgid(pid_t pid);

pid_t getpgrp(void);              /* POSIX.1 version */
pid_t getpgrp(psid_t pid);        /* BSD version */

int setpgrp(void);                /* System V version */
int setpgrp(pid_t pid, pid_t pgid); /* BSD version */
```

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

getpgid(): _XOPEN_SOURCE >= 500
setpgrp() (POSIX.1): _SVID_SOURCE || _XOPEN_SOURCE >= 500

setpgrp() (BSD), **getpgrp**() (BSD): _BSD_SOURCE && ! (_POSIX_SOURCE || _POSIX_C_SOURCE || _XOPEN_SOURCE || _XOPEN_SOURCE_EXTENDED || _GNU_SOURCE || _SVID_SOURCE)

DESCRIPTION

All of these interfaces are available on Linux, and are used for getting and setting the process group ID (PGID) of a process. The preferred, POSIX.1-specified ways of doing this are: **getpgrp**(void), for retrieving the calling process's PGID; and **setpgid**(), for setting a process's PGID.

setpgid() sets the PGID of the process specified by *pid* to *pgid*. If *pid* is zero, then the process ID of the calling process is used. If *pgid* is zero, then the PGID of the process specified by *pid* is made the same as its process ID. If **setpgid**() is used to move a process from one process group to another (as is done by some shells when creating pipelines), both process groups must be part of the same session (see **setsid**(2) and **credentials**(7)). In this case, the *pgid* specifies an existing process group to be joined and the session ID of that group must match the session ID of the joining process.

The POSIX.1 version of **getpgrp**(), which takes no arguments, returns the PGID of the calling process.

getpgid() returns the PGID of the process specified by *pid*. If *pid* is zero, the process ID of the calling process is used. (Retrieving the PGID of a process other than the caller is rarely necessary, and the POSIX.1 **getpgrp**() is preferred for that task.)

The System V-style **setpgrp**(), which takes no arguments, is equivalent to *setpgid(0, 0)*.

The BSD-specific **setpgrp**() call, which takes arguments *pid* and *pgid*, is equivalent to *setpgid(pid, pgid)*.

The BSD-specific **getpgrp**() call, which takes a single *pid* argument, is equivalent to *getpgid(pid)*.

RETURN VALUE

On success, **setpgid**() and **setpgrp**() return zero. On error, −1 is returned, and *errno* is set appropriately.

The POSIX.1 **getpgrp**() always returns the PGID of the caller.

getpgid(), and the BSD-specific **getpgrp**() return a process group on success. On error, −1 is returned, and *errno* is set appropriately.

ERRORS

EACCES
An attempt was made to change the process group ID of one of the children of the calling process and the child had already performed an **execve**(2) (**setpgid**(), **setpgrp**()).

EINVAL
pgid is less than 0 (**setpgid**(), **setpgrp**()).

EPERM
An attempt was made to move a process into a process group in a different session, or to change the process group ID of one of the children of the calling process and the child was in a different session, or to change the process group ID of a session leader (**setpgid**(), **setpgrp**()).

ESRCH
For **getpgid**(): *pid* does not match any process. For **setpgid**(): *pid* is not the calling process and not a child of the calling process.

CONFORMING TO

setpgid() and the version of **getpgrp**() with no arguments conform to POSIX.1-2001.

POSIX.1-2001 also specifies **getpgid**() and the version of **setpgrp**() that takes no arguments. (POSIX.1-2008 marks this **setpgrp**() specification as obsolete.)

The version of **getpgrp**() with one argument and the version of **setpgrp**() that takes two arguments derive from 4.2BSD, and are not specified by POSIX.1.

NOTES

A child created via **fork**(2) inherits its parent's process group ID. The PGID is preserved across an **execve**(2).

Each process group is a member of a session and each process is a member of the session of which its process group is a member.

A session can have a controlling terminal. At any time, one (and only one) of the process groups in the session can be the foreground process group for the terminal; the remaining process groups are in the background. If a signal is generated from the terminal (e.g., typing the interrupt key to generate **SIGINT**), that signal is sent to the foreground process group. (See **termios**(3) for a description of the characters that generate signals.) Only the foreground process group may **read**(2) from the terminal; if a background process group tries to **read**(2) from the terminal, then the group is sent a **SIGTSTP** signal, which suspends it. The **tcgetpgrp**(3) and **tcsetpgrp**(3) functions are used to get/set the foreground process group of the controlling terminal.

The **setpgid**() and **getpgrp**() calls are used by programs such as **bash**(1) to create process groups in order to implement shell job control.

If a session has a controlling terminal, and the **CLOCAL** flag for that terminal is not set, and a terminal hangup occurs, then the session leader is sent a **SIGHUP**. If the session leader exits, then a **SIGHUP** signal will also be sent to each process in the foreground process group of the controlling terminal.

If the exit of the process causes a process group to become orphaned, and if any member of the newly orphaned process group is stopped, then a **SIGHUP** signal followed by a **SIGCONT** signal will be sent to each process in the newly orphaned process group.

SEE ALSO

getuid(2), **setsid**(2), **tcgetpgrp**(3), **tcsetpgrp**(3), **termios**(3), **credentials**(7)

COLOPHON

This page is part of release 3.22 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.