kill(3) - Linux man page

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

kill - send a signal to a process or a group of processes

Synopsis

#include <signal.h>

int kill(pid_t *pid*, **int** *sig***);**

Description

The *kill*() function shall send a signal to a process or a group of processes specified by *pid*. The signal to be sent is specified by *sig* and is either one from the list given in <*signal.h*> or 0. If *sig* is 0 (the null signal), error checking is performed but no signal is actually sent. The null signal can be used to check the validity of *pid*.

For a process to have permission to send a signal to a process designated by *pid*, unless the sending process has appropriate privileges, the real or effective user ID of the sending process shall match the real or saved set-user-ID of the receiving process.

If *pid* is greater than 0, *sig* shall be sent to the process whose process ID is equal to *pid*.

If *pid* is 0, *sig* shall be sent to all processes (excluding an unspecified set of system processes) whose process group ID is equal to the process group ID of the sender, and for which the process has permission to send a signal.

If *pid* is -1, *sig* shall be sent to all processes (excluding an unspecified set of system processes) for which the process has permission to send that signal.

If *pid* is negative, but not -1, *sig* shall be sent to all processes (excluding an unspecified set of system processes) whose process group ID is equal to the absolute value of *pid*, and for which the process has permission to send a signal.

If the value of *pid* causes *sig* to be generated for the sending process, and if *sig* is not blocked for the calling thread and if no other thread has *sig* unblocked or is waiting in a *sigwait*() function for *sig*, either *sig* or at least one pending unblocked signal shall be delivered to the sending thread before *kill*() returns.

The user ID tests described above shall not be applied when sending SIGCONT to a process that is a member of the same session as the sending process.

An implementation that provides extended security controls may impose further implementation-defined restrictions on the sending of signals, including the null signal. In particular, the system may deny the existence of some or all of the processes specified by *pid*.

The *kill*() function is successful if the process has permission to send *sig* to any of the processes specified by *pid*. If *kill*() fails, no signal shall be sent.

Return Value

Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

Errors

The *kill*() function shall fail if:

EINVAL
: The value of the *sig* argument is an invalid or unsupported signal number.

EPERM
: The process does not have permission to send the signal to any receiving process.

ESRCH
: No process or process group can be found corresponding to that specified by *pid*.

The following sections are informative.

Examples

None.

Application Usage

None.

Rationale

The semantics for permission checking for *kill*() differed between System V and most other implementations, such as Version 7 or 4.3 BSD. The semantics chosen for this volume of IEEE Std 1003.1-2001 agree with System V. Specifically, a set-user-ID process cannot protect itself against signals (or at least not against SIGKILL) unless it changes its real user ID. This choice allows the user who starts an application to send it signals even if it changes its effective user ID. The other semantics give more power to an application that wants to protect itself from the user who ran it.

Some implementations provide semantic extensions to the *kill*() function when the absolute value of *pid* is greater than some maximum, or otherwise special, value. Negative values are a flag to *kill*(). Since most implementations return [ESRCH] in this case, this behavior is not included in this volume of IEEE Std 1003.1-2001, although a conforming implementation could provide such an extension.

The implementation-defined processes to which a signal cannot be sent may include the scheduler or *init*.

There was initially strong sentiment to specify that, if *pid* specifies that a signal be sent to the calling process and that signal is not blocked, that signal would be delivered before *kill*() returns. This would permit a process to call *kill*() and be guaranteed that the call never return. However, historical implementations that provide only the *signal*() function make only the weaker guarantee in this volume of IEEE Std 1003.1-2001, because they only deliver one signal each time a process enters the kernel. Modifications to such implementations to support the *sigaction*() function generally require entry to the kernel following return from a signal-catching function, in order to restore the signal mask. Such modifications have the effect of satisfying the stronger requirement, at least when *sigaction*() is used, but not necessarily when *signal*() is used. The developers of this volume of IEEE Std 1003.1-2001 considered making the stronger requirement except when *signal*() is used, but felt this would be unnecessarily complex. Implementors are encouraged to meet the stronger requirement whenever possible. In practice, the weaker requirement is the same, except in the rare case when two signals arrive during a very short window. This reasoning also applies to a similar requirement for *sigprocmask*().

In 4.2 BSD, the SIGCONT signal can be sent to any descendant process regardless of user-ID security checks. This allows a job control shell to continue a job even if processes in the job have altered their user IDs (as in the *su* command). In keeping with the addition of the concept of sessions, similar functionality is provided by allowing the SIGCONT signal to be sent to any process in the same session regardless of user ID security checks. This is less restrictive than BSD in the sense that ancestor processes (in the same session) can now be the recipient. It is more restrictive than BSD in the sense that descendant processes that form new sessions are now subject to the user ID checks. A similar relaxation of security is not necessary for the other job control signals since those signals are typically sent by the terminal driver in recognition of special characters being typed; the terminal driver bypasses all security checks.

In secure implementations, a process may be restricted from sending a signal to a process having a different security label. In order to prevent the existence or nonexistence of a process from being used as a covert channel, such processes should appear nonexistent to the sender; that is, [ESRCH] should be returned, rather than [EPERM], if *pid* refers only to such processes.

Existing implementations vary on the result of a *kill*() with *pid* indicating an inactive process (a terminated process that has not been waited for by its parent). Some indicate success on such a call (subject to permission checking), while others give an error of [ESRCH]. Since the definition of process lifetime in this volume of IEEE Std 1003.1-2001 covers inactive processes, the [ESRCH] error as described is inappropriate in this case. In particular, this means that an application cannot have a parent process check for termination of a particular child with *kill*(). (Usually this is done with the null signal; this can be done reliably with *waitpid*().)

There is some belief that the name *kill*() is misleading, since the function is not always intended to cause process termination. However, the name is common to all historical implementations, and any change would be in conflict with the goal of minimal changes to existing application code.

Future Directions

None.

See Also

getpid(), *raise*(), *setsid*(), *sigaction*(), *sigqueue*(), the Base Definitions volume of IEEE Std 1003.1-2001, <*signal.h*>, <*sys/types.h*>

Copyright

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1, 2003 Edition, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 6, Copyright Â© 2001-2003 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Referenced By

perlos2(1)