

NAME

pidfd_send_signal – send a signal to a process specified by a file descriptor

LIBRARY

Standard C library (*libc*, *-lc*)

SYNOPSIS

```
#include <linux/signal.h> /* Definition of SIG* constants */
#include <signal.h>        /* Definition of SI_* constants */
#include <sys/syscall.h>   /* Definition of SYS_* constants */
#include <unistd.h>
```

```
int syscall(SYS_pidfd_send_signal, int pidfd, int sig,
            siginfo_t * _Nullable info, unsigned int flags);
```

Note: glibc provides no wrapper for **pidfd_send_signal()**, necessitating the use of **syscall(2)**.

DESCRIPTION

The **pidfd_send_signal()** system call sends the signal *sig* to the target process referred to by *pidfd*, a PID file descriptor that refers to a process.

If the *info* argument points to a *siginfo_t* buffer, that buffer should be populated as described in **rt_sigqueueinfo(2)**.

If the *info* argument is a NULL pointer, this is equivalent to specifying a pointer to a *siginfo_t* buffer whose fields match the values that are implicitly supplied when a signal is sent using **kill(2)**:

- *si_signo* is set to the signal number;
- *si_errno* is set to 0;
- *si_code* is set to **SI_USER**;
- *si_pid* is set to the caller's PID; and
- *si_uid* is set to the caller's real user ID.

The calling process must either be in the same PID namespace as the process referred to by *pidfd*, or be in an ancestor of that namespace.

The *flags* argument is reserved for future use; currently, this argument must be specified as 0.

RETURN VALUE

On success, **pidfd_send_signal()** returns 0. On error, *-1* is returned and *errno* is set to indicate the error.

ERRORS**EBADF**

pidfd is not a valid PID file descriptor.

EINVAL

sig is not a valid signal.

EINVAL

The calling process is not in a PID namespace from which it can send a signal to the target process.

EINVAL

flags is not 0.

EPERM

The calling process does not have permission to send the signal to the target process.

EPERM

pidfd doesn't refer to the calling process, and *info.si_code* is invalid (see **rt_sigqueueinfo(2)**).

ESRCH

The target process does not exist (i.e., it has terminated and been waited on).

VERSIONS

pidfd_send_signal() first appeared in Linux 5.1.

STANDARDS

pidfd_send_signal() is Linux specific.

NOTES

PID file descriptors

The *pidfd* argument is a PID file descriptor, a file descriptor that refers to process. Such a file descriptor can be obtained in any of the following ways:

- by opening a */proc/pid* directory;
- using **pidfd_open(2)**; or
- via the PID file descriptor that is returned by a call to **clone(2)** or **clone3(2)** that specifies the **CLONE_PIDFD** flag.

The **pidfd_send_signal()** system call allows the avoidance of race conditions that occur when using traditional interfaces (such as **kill(2)**) to signal a process. The problem is that the traditional interfaces specify the target process via a process ID (PID), with the result that the sender may accidentally send a signal to the wrong process if the originally intended target process has terminated and its PID has been recycled for another process. By contrast, a PID file descriptor is a stable reference to a specific process; if that process terminates, **pidfd_send_signal()** fails with the error **ESRCH**.

EXAMPLES

```
#define _GNU_SOURCE
#include <fcntl.h>
#include <limits.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/syscall.h>
#include <unistd.h>

static int
pidfd_send_signal(int pidfd, int sig, siginfo_t *info,
                  unsigned int flags)
{
    return syscall(SYS_pidfd_send_signal, pidfd, sig, info, flags);
}

int
main(int argc, char *argv[])
{
    int      pidfd, sig;
    char     path[PATH_MAX];
    siginfo_t info;

    if (argc != 3) {
        fprintf(stderr, "Usage: %s <pid> <signal>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    sig = atoi(argv[2]);

    /* Obtain a PID file descriptor by opening the /proc/PID directory
       of the target process. */
```

```
    snprintf(path, sizeof(path), "/proc/%s", argv[1]);

    pidfd = open(path, O_RDONLY);
    if (pidfd == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }

    /* Populate a 'siginfo_t' structure for use with
       pidfd_send_signal(). */

    memset(&info, 0, sizeof(info));
    info.si_code = SI_QUEUE;
    info.si_signo = sig;
    info.si_errno = 0;
    info.si_uid = getuid();
    info.si_pid = getpid();
    info.si_value.sival_int = 1234;

    /* Send the signal. */

    if (pidfd_send_signal(pidfd, sig, &info, 0) == -1) {
        perror("pidfd_send_signal");
        exit(EXIT_FAILURE);
    }

    exit(EXIT_SUCCESS);
}
```

SEE ALSO

clone(2), kill(2), pidfd_open(2), rt_sigqueueinfo(2), sigaction(2), pid_namespaces(7), signal(7)