

**NAME**

sigwaitinfo, sigtimedwait, rt\_sigtimedwait – synchronously wait for queued signals

**LIBRARY**

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

**SYNOPSIS**

```
#include <signal.h>

int sigwaitinfo(const sigset_t *restrict set,
                siginfo_t *_Nullable restrict info);
int sigtimedwait(const sigset_t *restrict set,
                 siginfo_t *_Nullable restrict info,
                 const struct timespec *restrict timeout);
```

Feature Test Macro Requirements for glibc (see **feature\_test\_macros(7)**):

```
sigwaitinfo(), sigtimedwait():
    _POSIX_C_SOURCE >= 199309L
```

**DESCRIPTION**

**sigwaitinfo()** suspends execution of the calling thread until one of the signals in *set* is pending (If one of the signals in *set* is already pending for the calling thread, **sigwaitinfo()** will return immediately.)

**sigwaitinfo()** removes the signal from the set of pending signals and returns the signal number as its function result. If the *info* argument is not NULL, then the buffer that it points to is used to return a structure of type *siginfo\_t* (see **sigaction(2)**) containing information about the signal.

If multiple signals in *set* are pending for the caller, the signal that is retrieved by **sigwaitinfo()** is determined according to the usual ordering rules; see **signal(7)** for further details.

**sigtimedwait()** operates in exactly the same way as **sigwaitinfo()** except that it has an additional argument, *timeout*, which specifies the interval for which the thread is suspended waiting for a signal. (This interval will be rounded up to the system clock granularity, and kernel scheduling delays mean that the interval may overrun by a small amount.) This argument is a **timespec(3)** structure.

If both fields of this structure are specified as 0, a poll is performed: **sigtimedwait()** returns immediately, either with information about a signal that was pending for the caller, or with an error if none of the signals in *set* was pending.

**RETURN VALUE**

On success, both **sigwaitinfo()** and **sigtimedwait()** return a signal number (i.e., a value greater than zero). On failure both calls return *-1*, with *errno* set to indicate the error.

**ERRORS****EAGAIN**

No signal in *set* became pending within the *timeout* period specified to **sigtimedwait()**.

**EINTR**

The wait was interrupted by a signal handler; see **signal(7)**. (This handler was for a signal other than one of those in *set*.)

**EINVAL**

*timeout* was invalid.

**STANDARDS**

POSIX.1-2001, POSIX.1-2008.

**NOTES**

In normal usage, the calling program blocks the signals in *set* via a prior call to **sigprocmask(2)** (so that the default disposition for these signals does not occur if they become pending between successive calls to **sigwaitinfo()** or **sigtimedwait()**) and does not establish handlers for these signals. In a multithreaded program, the signal should be blocked in all threads, in order to prevent the signal being treated according to its default disposition in a thread other than the one calling **sigwaitinfo()** or **sigtimedwait()**.

The set of signals that is pending for a given thread is the union of the set of signals that is pending specifically for that thread and the set of signals that is pending for the process as a whole (see **signal(7)**).

Attempts to wait for **SIGKILL** and **SIGSTOP** are silently ignored.

If multiple threads of a process are blocked waiting for the same signal(s) in **sigwaitinfo()** or **sigtimedwait()**, then exactly one of the threads will actually receive the signal if it becomes pending for the process as a whole; which of the threads receives the signal is indeterminate.

**sigwaitinfo()** or **sigtimedwait()**, can't be used to receive signals that are synchronously generated, such as the **SIGSEGV** signal that results from accessing an invalid memory address or the **SIGFPE** signal that results from an arithmetic error. Such signals can be caught only via signal handler.

POSIX leaves the meaning of a NULL value for the *timeout* argument of **sigtimedwait()** unspecified, permitting the possibility that this has the same meaning as a call to **sigwaitinfo()**, and indeed this is what is done on Linux.

### C library/kernel differences

On Linux, **sigwaitinfo()** is a library function implemented on top of **sigtimedwait()**.

The glibc wrapper functions for **sigwaitinfo()** and **sigtimedwait()** silently ignore attempts to wait for the two real-time signals that are used internally by the NPTL threading implementation. See **nptl(7)** for details.

The original Linux system call was named **sigtimedwait()**. However, with the addition of real-time signals in Linux 2.2, the fixed-size, 32-bit *sigset\_t* type supported by that system call was no longer fit for purpose. Consequently, a new system call, **rt\_sigtimedwait()**, was added to support an enlarged *sigset\_t* type. The new system call takes a fourth argument, *size\_t sigsetsize*, which specifies the size in bytes of the signal set in *set*. This argument is currently required to have the value *sizeof(sigset\_t)* (or the error **EINVAL** results). The glibc **sigtimedwait()** wrapper function hides these details from us, transparently calling **rt\_sigtimedwait()** when the kernel provides it.

### SEE ALSO

**kill(2)**, **sigaction(2)**, **signal(2)**, **signalfd(2)**, **sigpending(2)**, **sigprocmask(2)**, **sigqueue(3)**, **sigsetops(3)**, **sigwait(3)**, **timespec(3)**, **signal(7)**, **time(7)**