

NAME

`epoll_wait`, `epoll_pwait` - wait for an I/O event on an epoll file descriptor

SYNOPSIS

```
#include <sys/epoll.h>
```

```
int epoll_wait(int epfd, struct epoll_event *events,
               int maxevents, int timeout);
int epoll_pwait(int epfd, struct epoll_event *events,
                int maxevents, int timeout,
                const sigset_t *sigmask);
```

DESCRIPTION

The `epoll_wait()` system call waits for events on the [epoll\(7\)](#) instance referred to by the file descriptor *epfd*. The memory area pointed to by *events* will contain the events that will be available for the caller. Up to *maxevents* are returned by `epoll_wait()`. The *maxevents* argument must be greater than zero.

The *timeout* argument specifies the number of milliseconds that `epoll_wait()` will block. The call will block until either:

- * a file descriptor delivers an event;
- * the call is interrupted by a signal handler; or
- * the timeout expires.

Note that the *timeout* interval will be rounded up to the system clock granularity, and kernel scheduling delays mean that the blocking interval may overrun by a small amount. Specifying a *timeout* of -1 causes `epoll_wait()` to block indefinitely, while specifying a *timeout* equal to zero cause `epoll_wait()` to return immediately, even if no events are available.

The *struct epoll_event* is defined as :

```
typedef union epoll_data {
    void *ptr;
    int fd;
    uint32_t u32;
    uint64_t u64;
} epoll_data_t;

struct epoll_event {
    uint32_t events; /* Epoll events */
    epoll_data_t data; /* User data variable */
};
```

The *data* of each returned structure will contain the same data the user set with an [epoll_ctl\(2\)](#) (`EPOLL_CTL_ADD`, `EPOLL_CTL_MOD`) while the *events* member will contain the returned event bit field.

epoll_pwait()

The relationship between `epoll_wait()` and `epoll_pwait()` is analogous to the relationship between [select\(2\)](#) and [pselect\(2\)](#): like [pselect\(2\)](#), `epoll_pwait()` allows an application to safely wait until either a file descriptor becomes ready or until a signal is caught.

The following `epoll_pwait()` call:

```
ready = epoll_pwait(epfd, &events, maxevents, timeout, &sigmask);
```

is equivalent to *atomically* executing the following calls:

```
sigset_t origmask;
```

```
sigprocmask(SIG_SETMASK, &sigmask, &origmask);
```

```
ready = epoll_wait(epfd, &events, maxevents, timeout);
sigprocmask(SIG_SETMASK, &origmask, NULL);
```

The *sigmask* argument may be specified as NULL, in which case **epoll_pwait()** is equivalent to **epoll_wait()**.

RETURN VALUE

When successful, **epoll_wait()** returns the number of file descriptors ready for the requested I/O, or zero if no file descriptor became ready during the requested *timeout* milliseconds. When an error occurs, **epoll_wait()** returns -1 and *errno* is set appropriately.

ERRORS

EBADF

epfd is not a valid file descriptor.

EFAULT

The memory area pointed to by *events* is not accessible with write permissions.

EINTR

The call was interrupted by a signal handler before either (1) any of the requested events occurred or (2) the *timeout* expired; see [signal\(7\)](#).

EINVAL

epfd is not an **epoll** file descriptor, or *maxevents* is less than or equal to zero.

VERSIONS

epoll_wait() was added to the kernel in version 2.6. Library support is provided in glibc starting with version 2.3.2.

epoll_pwait() was added to Linux in kernel 2.6.19. Library support is provided in glibc starting with version 2.6.

CONFORMING TO

epoll_wait() is Linux-specific.

NOTES

While one thread is blocked in a call to **epoll_pwait()**, it is possible for another thread to add a file descriptor to the waited-upon **epoll** instance. If the new file descriptor becomes ready, it will cause the **epoll_wait()** call to unblock.

For a discussion of what may happen if a file descriptor in an **epoll** instance being monitored by **epoll_wait()** is closed in another thread, see [select\(2\)](#).

BUGS

In kernels before 2.6.37, a *timeout* value larger than approximately $LONG_MAX / HZ$ milliseconds is treated as -1 (i.e., infinity). Thus, for example, on a system where the *sizeof(long)* is 4 and the kernel *HZ* value is 1000, this means that timeouts greater than 35.79 minutes are treated as infinity.

C library/kernel ABI differences

The raw **epoll_pwait()** system call has a sixth argument, *size_t sigsetsize*, which specifies the size in bytes of the *sigmask* argument. The glibc **epoll_pwait()** wrapper function specifies this argument as a fixed value (equal to *sizeof(sigset_t)*).

SEE ALSO

[epoll_create\(2\)](#), [epoll_ctl\(2\)](#), [epoll\(7\)](#)

COLOPHON

This page is part of release 3.74 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 <http://www.kernel.org/doc/man-pages/>.