NAME

epoll_wait, epoll_pwait, epoll_pwait2 − wait for an I/O event on an epoll file descriptor

LIBRARY

Standard C library (*libc*, *−lc*)

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 *_Nullable sigmask);
int epoll_pwait2(int epfd, struct epoll_event *events,
               int maxevents, const struct timespec *_Nullable timeout,
               const sigset_t *_Nullable sigmask);
```

DESCRIPTION

The **epoll_wait**() system call waits for events on the **epoll**(7) instance referred to by the file descriptor *epfd*. The buffer pointed to by *events* is used to return information from the ready list about file descriptors in the interest list that have some events available. Up to*maxe vents* are returned by **epoll_wait**(). The*maxe vents* argument must be greater than zero.

The *timeout* argument specifies the number of milliseconds that **epoll_wait**() will block. Time is measured against the **CLOCK_MONOTONIC** clock.

A call to **epoll_wait**() 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 described in **epoll_event**(3type).

The *data* field of each returned *epoll_event* structure contains the same data as was specified in the most recent call to **epoll_ctl**(2) (**EPOLL_CTL_ADD**, **EPOLL_CTL_MOD**) for the corresponding open file descriptor.

The *events* field is a bit mask that indicates the events that have occurred for the corresponding open file description. See**epoll_ctl**(2) for a list of the bits that may appear in this mask.

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;

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

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

epoll_pwait2()

The **epoll_pwait2**() system call is equivalent to **epoll_pwait**() except for the *timeout* argument. It takes an argument of type *timespec* to be able to specify nanosecond resolution timeout. This argument functions the same as in **pselect**(2) and **ppoll**(2). If*timeout* is NULL, then **epoll_pwait2**() can block indefinitely .

RETURN VALUE

On success, **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. On failure, **epoll_wait**() returns −1 and *errno* is set to indicate the error.

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 in Linux 2.6. Library support is provided in glibc 2.3.2.

epoll_pwait() was added in Linux 2.6.19. Library support is provided in glibc 2.6.

epoll_pwait2() was added in Linux 5.11.

STANDARDS

epoll_wait(), **epoll_pwait**(), and **epoll_pwait2**() are Linux-specific.

NOTES

While one thread is blocked in a call to **epoll_wait**(), 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.

If more than *maxevents* file descriptors are ready when **epoll_wait**() is called, then successive **epoll_wait**() calls will round robin through the set of ready file descriptors. This behavior helps avoid starvation scenarios, where a process fails to notice that additional file descriptors are ready because it focuses on a set of file descriptors that are already known to be ready.

Note that it is possible to call **epoll_wait**() on an **epoll** instance whose interest list is currently empty (or whose interest list becomes empty because file descriptors are closed or removed from the interest in another thread). The call will block until some file descriptor is later added to the interest list (in another thread) and that file descriptor becomes ready.

C library/kernel differences

The raw **epoll_pwait**() and **epoll_pwait2**() system calls have 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)*).

BUGS

Before Linux 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 *sizeof(long)* is 4 and the kernel *HZ* value is 1000, this means that timeouts greater than 35.79 minutes are treated as infinity.

SEE ALSO

epoll_create(2), **epoll_ctl**(2), **epoll**(7)