
KQUEUE(2) System Calls Manual KQUEUE(2)

NAME

kqueue, kevent -- kernel event notification mechanism

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

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

int
kqueue(void);

int
kqueuex(u_int flags);

int
kevent(int kq,      const struct kevent *changelist,      int nchanges,
    struct kevent *eventlist,                             int nevents,
    const struct timespec *timeout);

EV_SET(kev, ident, filter, flags, fflags, data, udata);
```

DESCRIPTION

The **kqueue**() system call provides a generic method of notifying the user when an event happens or a condition holds, based on the results of small pieces of kernel code termed filters. A kevent is identified by the (ident, filter) pair; there may only be one unique kevent per kqueue.

The filter is executed upon the initial registration of a kevent in order to detect whether a preexisting condition is present, and is also executed whenever an event is passed to the filter for evaluation. If the filter determines that the condition should be reported, then the kevent is placed on the kqueue for the user to retrieve.

The filter is also run when the user attempts to retrieve the kevent from the kqueue. If the filter indicates that the condition that triggered the event no longer holds, the kevent is removed from the kqueue and is not returned.

Multiple events which trigger the filter do not result in multiple kevents being placed on the kqueue; instead, the filter will aggregate the events into a single struct kevent. Calling **close**() on a file de-

scriptor will remove any kevents that reference the descriptor.

The **kqueue**() system call creates a new kernel event queue and returns a descriptor. The queue is not inherited by a child created with *fork*(2). However, if *rfork*(2) is called without the RFFDG flag, then the descriptor table is shared, which will allow sharing of the kqueue between two processes.

The **kqueuex**() system call also creates a new kernel event queue, and additionally takes the *flags* argument, which is a bitwise-inclusive OR of the following flags:

KQUEUE_CLOEXEC The returned file descriptor is automatically closed on *execve*(2)

The `fd = kqueue()' call is equivalent to `fd = kqueuex(0)'.

For compatibility with NetBSD, the **kqueue1**() function is provided, which accepts the O_CLOEXEC flag with the expected semantic.

The **kevent**() system call is used to register events with the queue, and return any pending events to the user. The *changelist* argument is a pointer to an array of *kevent* structures, as defined in *<sys/event.h>*. All changes contained in the *changelist* are applied before any pending events are read from the queue. The *nchanges* argument gives the size of *changelist*. The *eventlist* argument is a pointer to an array of kevent structures. The *nevents* argument determines the size of *eventlist*. When *nevents* is zero, **kevent**() will return immediately even if there is a *timeout* specified unlike *select*(2). If *timeout* is a non-NULL pointer, it specifies a maximum interval to wait for an event, which will be interpreted as a struct timespec. If *timeout* is a NULL pointer, **kevent**() waits indefinitely. To effect a poll, the *timeout* argument should be non-NULL, pointing to a zero-valued *timespec* structure. The same array may be used for the *changelist* and *eventlist*.

The **EV_SET**() macro is provided for ease of initializing a kevent structure.

The *kevent* structure is defined as:

```
struct kevent {
        uintptr_t  ident;       /* identifier for this event */
        short      filter;      /* filter for event */
        u_short    flags;       /* action flags for kqueue */
        u_int      fflags;      /* filter flag value */
        int64_t    data;        /* filter data value */
        void       *udata;      /* opaque user data identifier */
        uint64_t   ext[4];      /* extensions */
};
```

The fields of *struct kevent* are:

ident Value used to identify this event. The exact interpretation is determined by the attached filter, but often is a file descriptor.

filter Identifies the kernel filter used to process this event. The pre-defined system filters are described below.

flags Actions to perform on the event.

fflags Filter-specific flags.

data Filter-specific data value.

udata Opaque user-defined value passed through the kernel unchanged.

ext Extended data passed to and from kernel. The *ext[0]* and *ext[1]* members use is defined by the filter. If the filter

does not use them, the members are copied unchanged. The *ext[2]* and *ext[3]* members are always passed through the kernel as-is, making additional context available to application.

The *flags* field can contain the following values:

EV_ADD Adds the event to the kqueue. Re-adding an existing event will modify the parameters of the original event, and not result in a duplicate entry. Adding an event automatically enables it, unless overridden by the EV_DISABLE flag.

EV_ENABLE Permit **kevent**() to return the event if it is triggered.

EV_DISABLE Disable the event so **kevent**() will not return it. The filter itself is not disabled.

EV_DISPATCH Disable the event source immediately after delivery of an event. See EV_DISABLE above.

EV_DELETE Removes the event from the kqueue. Events which are attached to file descriptors are automatically deleted on the last close of the descriptor.

EV_RECEIPT This flag is useful for making bulk changes to a kqueue without draining any pending events. When passed as input, it forces EV_ERROR to always be returned. When a filter is successfully added the *data* field will be zero. Note that if this flag is encountered and there is no remaining space in *eventlist* to hold the EV_ERROR event, then subsequent changes will not get processed.

EV_ONESHOT Causes the event to return only the first occurrence of the filter being triggered. After the user retrieves the event from the kqueue, it is deleted.

EV_CLEAR After the event is retrieved by the user, its state is reset. This is useful for filters which report state transitions instead of the current state. Note that some filters may automatically set this flag internally.

EV_EOF Filters may set this flag to indicate filter-specific EOF condition.

EV_ERROR See "RETURN VALUES" below.

EV_KEEPUDATA
Causes **kevent**() to leave unchanged any *udata* associated with an existing event. This allows other aspects of the event to be modified without requiring the caller to know the *udata* value presently associated. This is especially useful with NOTE_TRIGGER or flags like EV_ENABLE. This flag may not be used with EV_ADD.

The predefined system filters are listed below. Arguments may be passed to and from the filter via the *fflags* and *data* fields in the kevent structure.

EVFILT_READ Takes a descriptor as the identifier, and returns whenever there is data available to read. The behavior of the filter is slightly different depending on the descriptor type.

Sockets
Sockets which have previously been passed to *listen*(2) return when there is an incoming connection pending. *data* contains the size of the listen backlog.

Other socket descriptors return when there is data to be read, subject to the SO_RCVLOWAT value of the socket buffer. This may be overridden with a per-filter low water mark at the time the filter is added by setting the NOTE_LOWAT flag in *fflags*, and specifying the new low water mark in *data*. On return, *data* contains the number of bytes of protocol data available to read.

If the read direction of the socket has shutdown, then the filter also sets EV_EOF in *flags*, and returns the socket error (if any) in *fflags*. It is possible for EOF to be returned (indicating the connection is gone) while there is still data pending in the socket buffer.

Vnodes
Returns when the file pointer is not at the end of file. *data* contains the offset from current position to end of file, and may be negative.

This behavior is different from *poll*(2), where read events are triggered for regular files unconditionally. This event can be triggered unconditionally by setting the NOTE_FILE_POLL flag in *fflags*.

Fifos, Pipes
Returns when the there is data to read; *data* contains the number of bytes available.

When the last writer disconnects, the filter will set EV_EOF in *flags*. This will be cleared by the filter when a new writer connects, at which point the filter will resume waiting for data to become available before returning.

BPF devices
Returns when the BPF buffer is full, the BPF timeout has expired, or when the BPF has "immediate mode" enabled and there is any data to read; *data* contains the number of bytes available.

Eventfds
Returns when the counter is greater than 0; *data* contains the counter value, which must be cast to *uint64_t*.

Kqueues
Returns when pending events are present on the queue; *data* contains the number of events available.

EVFILT_WRITE Takes a descriptor as the identifier, and returns whenever it is possible to write to the descriptor. For sockets, pipes and fifos, *data* will contain the amount of space remaining in the write buffer. The filter will set EV_EOF when the reader disconnects, and for the fifo case, this will be cleared when a new reader connects. Note that this filter is not supported for vnodes.

For sockets, the low water mark and socket error handling is identical to the EVFILT_READ case.

For eventfds, *data* will contain the maximum value

that can be added to the counter without blocking.

For BPF devices, when the descriptor is attached to an interface the filter always indicates that it is possible to write and *data* will contain the MTU size of the underlying interface.

EVFILT_EMPTY — Takes a descriptor as the identifier, and returns whenever there is no remaining data in the write buffer.

EVFILT_AIO — Events for this filter are not registered with **kevent**() directly but are registered via the *aio_sigevent* member of an asynchronous I/O request when it is scheduled via an asynchronous I/O system call such as **aio_read**(). The filter returns under the same conditions as **aio_error**(). For more details on this filter see *sigevent*(3) and *aio*(4).

EVFILT_VNODE — Takes a file descriptor as the identifier and the events to watch for in *fflags*, and returns when one or more of the requested events occurs on the descriptor. The events to monitor are:

- NOTE_ATTRIB — The file referenced by the descriptor had its attributes changed.
- NOTE_CLOSE — A file descriptor referencing the monitored file, was closed. The closed file descriptor did not have write access.
- NOTE_CLOSE_WRITE — A file descriptor referencing the monitored file, was closed. The closed file descriptor had write access.

  This note, as well as NOTE_CLOSE, are not activated when files are closed forcibly by *unmount*(2) or *revoke*(2). Instead, NOTE_REVOKE is sent for such events.
- NOTE_DELETE — The **unlink**() system call was called on the file referenced by the descriptor.
- NOTE_EXTEND — For regular file, the file referenced by the descriptor was extended.

  For directory, reports that a directory entry was added or removed, as the result of rename operation. The NOTE_EXTEND event is not reported when a name is changed inside the directory.
- NOTE_LINK — The link count on the file changed. In particular, the NOTE_LINK event is reported if a subdirectory was created or deleted inside the directory referenced by the descriptor.

NOTE_OPEN The file referenced by the descriptor was opened.

NOTE_READ A read occurred on the file referenced by the descriptor.

NOTE_RENAME The file referenced by the descriptor was renamed.

NOTE_REVOKE Access to the file was revoked via *revoke*(2) or the underlying file system was unmounted.

NOTE_WRITE A write occurred on the file referenced by the descriptor.

On return, *fflags* contains the events which triggered the filter.

EVFILT_PROC Takes the process ID to monitor as the identifier and the events to watch for in *fflags*, and returns when the process performs one or more of the requested events. If a process can normally see another process, it can attach an event to it. The events to monitor are:

NOTE_EXIT The process has exited. The exit status will be stored in *data* in the same format as the status returned by *wait*(2).

NOTE_FORK The process has called **fork**().

NOTE_EXEC The process has executed a new process via *execve*(2) or a similar call.

NOTE_TRACK Follow a process across **fork**() calls. The parent process registers a new kevent to monitor the child process using the same *fflags* as the original event. The child process will signal an event with NOTE_CHILD set in *fflags* and the parent PID in *data*.

If the parent process fails to register a new kevent (usually due to resource limitations), it will signal an event with NOTE_TRACKERR set in *fflags*, and the child process will not signal a NOTE_CHILD event.

On return, *fflags* contains the events which triggered the filter.

EVFILT_PROCDESC Takes the process descriptor created by *pdfork*(2) to monitor as the identifier and the events to watch for in *fflags*, and returns when the associated process performs one or more of the requested events. The events to monitor are:

NOTE_EXIT The process has exited. The exit status will be stored in *data*.

On return, *fflags* contains the events which triggered the filter.

EVFILT_SIGNAL Takes the signal number to monitor as the identifier and returns when the given signal is delivered to the process. This coexists with the **signal**() and **sigaction**() facilities, and has a lower precedence. The filter will record all attempts to deliver a signal to a process, even if the signal has been marked as SIG_IGN, except for the SIGCHLD signal, which, if ignored, will not be recorded by the filter. Event notification happens after normal signal delivery processing. *data* returns the number of times the signal has occurred since the last call to **kevent**(). This filter automatically sets the EV_CLEAR flag internally.

EVFILT_TIMER Establishes an arbitrary timer identified by *ident*. When adding a timer, *data* specifies the moment to fire the timer (for NOTE_ABSTIME) or the timeout period. The timer will be periodic unless EV_ONESHOT or NOTE_ABSTIME is specified. On return, *data* contains the number of times the timeout has expired since the last call to **kevent**(). For non-monotonic timers, this filter automatically sets the EV_CLEAR flag internally.

The filter accepts the following flags in the *fflags* argument:

NOTE_SECONDS *data* is in seconds.

NOTE_MSECONDS *data* is in milliseconds.

NOTE_USECONDS *data* is in microseconds.

NOTE_NSECONDS *data* is in nanoseconds.

NOTE_ABSTIME The specified expiration time is absolute.

If *fflags* is not set, the default is milliseconds. On return, *fflags* contains the events which triggered the filter.

Periodic timers with a specified timeout of 0 will be silently adjusted to timeout after 1 of the time units specified by the requested precision in *fflags*. If an absolute time is specified that has already passed, then it is treated as if the current time were specified and the event will fire as soon as possible.

If an existing timer is re-added, the existing timer will be effectively canceled (throwing away any undelivered record of previous timer expiration) and re-started using the new parameters contained in *data* and *fflags*.

There is a system wide limit on the number of timers which is controlled by the *kern.kq_calloutmax* sysctl.

EVFILT_USER Establishes a user event identified by *ident* which is not associated with any kernel mechanism but is triggered by user level code. The lower 24 bits of the *fflags* may be used for user defined flags and manipulated using the following:

NOTE_FFNOP Ignore the input *fflags*.

NOTE_FFAND	Bitwise AND *fflags*.
NOTE_FFOR	Bitwise OR *fflags*.
NOTE_FFCOPY	Copy *fflags*.
NOTE_FFCTRLMASK	Control mask for *fflags*.
NOTE_FFLAGSMASK	User defined flag mask for *fflags*.

A user event is triggered for output with the following:

NOTE_TRIGGER	Cause the event to be triggered.

On return, *fflags* contains the users defined flags in the lower 24 bits.

CANCELLATION BEHAVIOUR

If *nevents* is non-zero, i.e., the function is potentially blocking, the call is a cancellation point. Otherwise, i.e., if *nevents* is zero, the call is not cancellable. Cancellation can only occur before any changes are made to the kqueue, or when the call was blocked and no changes to the queue were requested.

RETURN VALUES

The **kqueue**() system call creates a new kernel event queue and returns a file descriptor. If there was an error creating the kernel event queue, a value of -1 is returned and errno set.

The **kevent**() system call returns the number of events placed in the *eventlist*, up to the value given by *nevents*. If an error occurs while processing an element of the *changelist* and there is enough room in the *eventlist*, then the event will be placed in the *eventlist* with EV_ERROR set in *flags* and the system error in *data*. Otherwise, -1 will be returned, and errno will be set to indicate the error condition. If the time limit expires, then **kevent**() returns 0.

EXAMPLES

```
#include <sys/event.h>
#include <err.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int
main(int argc, char **argv)
{
    struct kevent event;    /* Event we want to monitor */
    struct kevent tevent;   /* Event triggered */
    int kq, fd, ret;

    if (argc != 2)
        err(EXIT_FAILURE, "Usage: %s path\n", argv[0]);
    fd = open(argv[1], O_RDONLY);
    if (fd == -1)
        err(EXIT_FAILURE, "Failed to open '%s'", argv[1]);

    /* Create kqueue. */
    kq = kqueue();
    if (kq == -1)
        err(EXIT_FAILURE, "kqueue() failed");

    /* Initialize kevent structure. */
    EV_SET(&event, fd, EVFILT_VNODE, EV_ADD | EV_CLEAR, NOTE_WRITE,
        0, NULL);
```

```
        /* Attach event to the kqueue. */
        ret = kevent(kq, &event, 1, NULL, 0, NULL);
        if (ret == -1)
            err(EXIT_FAILURE, "kevent register");

        for (;;) {
            /* Sleep until something happens. */
            ret = kevent(kq, NULL, 0, &tevent, 1, NULL);
            if (ret == -1) {
                err(EXIT_FAILURE, "kevent wait");
            } else if (ret > 0) {
                if (tevent.flags & EV_ERROR)
                    errx(EXIT_FAILURE, "Event error: %s", strerror(event.data));
                else
                    printf("Something was written in '%s'\n", argv[1]);
            }
        }

        /* kqueues are destroyed upon close() */
        (void)close(kq);
        (void)close(fd);
    }
```

ERRORS

The **kqueue**() system call fails if:

[ENOMEM] The kernel failed to allocate enough memory for the kernel queue.

[ENOMEM] The RLIMIT_KQUEUES rlimit (see *getrlimit(2)*) for the current user would be exceeded.

[EMFILE] The per-process descriptor table is full.

[ENFILE] The system file table is full.

The **kevent**() system call fails if:

[EACCES] The process does not have permission to register a filter.

[EFAULT] There was an error reading or writing the *kevent* structure.

[EBADF] The specified descriptor is invalid.

[EINTR] A signal was delivered before the timeout expired and before any events were placed on the kqueue for return.

[EINTR] A cancellation request was delivered to the thread, but not yet handled.

[EINVAL] The specified time limit or filter is invalid.

[EINVAL] The specified length of the event or change lists is negative.

[ENOENT] The event could not be found to be modified or deleted.

[ENOMEM] No memory was available to register the event or, in the special case of a timer, the maximum number of timers has been exceeded. This maximum is configurable via the *kern.kq_calloutmax* sysctl.

[ESRCH] The specified process to attach to does not exist.

When **kevent**() call fails with EINTR error, all changes in the

changelist have been applied.

SEE ALSO

aio_error(2), *aio_read*(2), *aio_return*(2), *poll*(2), *read*(2), *select*(2), *sigaction*(2), *write*(2), *pthread_setcancelstate*(3), *signal*(3)

Jonathan Lemon, "Kqueue: A Generic and Scalable Event Notification Facility", *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, *USENIX Association*, June 25-30, 2001.

HISTORY

The **kqueue**() and **kevent**() system calls first appeared in FreeBSD 4.1.

AUTHORS

The **kqueue**() system and this manual page were written by Jonathan Lemon <*jlemon@FreeBSD.org*>.

BUGS

In versions older than FreeBSD 12.0, <*sys/event.h*> failed to parse without including <*sys/types.h*> manually.

FreeBSD 13.2 March 26, 2023 KQUEUE(2)

NAME | LIBRARY | SYNOPSIS | DESCRIPTION | CANCELLATION BEHAVIOUR | RETURN VALUES | EXAMPLES | ERRORS | SEE ALSO | HISTORY | AUTHORS | BUGS

Want to link to this manual page? Use this URL:
<https://man.freebsd.org/cgi/man.cgi?query=kqueue&manpath=FreeBSD+14.1-RELEASE+and+Ports>

home | help
