## NAME

**kqueue_add_filteropts**, **kqueue_del_filteropts**, **kqfd_register**, **knote_fdclose**, **knlist_init**, **knlist_init_mtx**, **knlist_add**, **knlist_remove**, **knlist_remove_inevent**, **knlist_empty**, **knlist_clear**, **knlist_delete**, **knlist_destroy**, **KNOTE_LOCKED**, **KNOTE_UNLOCKED** - event delivery subsystem

## SYNOPSIS

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

*int*
**kqueue_add_filteropts**(*int filt, struct filterops *filtops*);

*int*
**kqueue_del_filteropts**(*int filt*);

*int*
**kqfd_register**(*int fd, struct kevent *kev, struct thread *td, int waitok*);

*void*
**knote_fdclose**(*struct thread *td, int fd*);

*void*
**knlist_init**(*struct knlist *knl, void *lock, void (*kl_lock)(void *), void (*kl_unlock)(void *), int (*kl_locked)(void *)*);

*void*
**knlist_init_mtx**(*struct knlist *knl, struct mtx *lock*);

*void*
**knlist_add**(*struct knlist *knl, struct knote *kn, int islocked*);

*void*
**knlist_remove**(*struct knlist *knl, struct knote *kn, int islocked*);

*void*
**knlist_remove_inevent**(*struct knlist *knl, struct knote *kn*);

*int*
**knlist_empty**(*struct knlist *knl*);

*void*

**knlist_clear**(*struct knlist *knl, int islocked*);

*void*
**knlist_delete**(*struct knlist *knl, struct thread *td, int islocked*);

*void*
**knlist_destroy**(*struct knlist *knl*);

*void*
**KNOTE_LOCKED**(*struct knlist *knl, long hint*);

*void*
**KNOTE_UNLOCKED**(*struct knlist *knl, long hint*);

## DESCRIPTION

The functions **kqueue_add_filteropts**() and **kqueue_del_filteropts**() allow for the addition and removal of a filter type. The filter is statically defined by the EVFILT_* macros. The function **kqueue_add_filteropts**() will make *filt* available. The *struct filterops* has the following members:

*f_isfd* If *f_isfd* is set, *ident* in *struct kevent* is taken to be a file descriptor. In this case, the *knote* passed into *f_attach* will have the *kn_fp* member initialized to the *struct file* * that represents the file descriptor.

*f_attach* The *f_attach* function will be called when attaching a *knote* to the object. The method should call **knlist_add**() to add the *knote* to the list that was initialized with **knlist_init**(). The call to **knlist_add**() is only necessary if the object can have multiple *knotes* associated with it. If there is no *knlist* to call **knlist_add**() with, the function *f_attach* must clear the KN_DETACHED bit of *kn_status* in the *knote*. The function shall return 0 on success, or appropriate error for the failure, such as when the object is being destroyed, or does not exist. During *f_attach*, it is valid to change the *kn_fop* pointer to a different pointer. This will change the *f_event* and *f_detach* functions called when processing the *knote*.

*f_detach*
The *f_detach* function will be called to detach the *knote* if the *knote* has not already been detached by a call to **knlist_remove**(), **knlist_remove_inevent**() or **knlist_delete**(). The list *lock* will not be held when this function is called.

*f_event* The *f_event* function will be called to update the status of the *knote*. If the function returns 0, it will be assumed that the object is not ready (or no longer ready) to be woken up. The *hint* argument will be 0 when scanning *knotes* to see which are triggered. Otherwise, the *hint*

argument will be the value passed to either KNOTE_LOCKED or KNOTE_UNLOCKED. The *kn_data* value should be updated as necessary to reflect the current value, such as number of bytes available for reading, or buffer space available for writing. If the note needs to be removed, **knlist_remove_inevent**() must be called. The function **knlist_remove_inevent**() will remove the note from the list, the *f_detach* function will not be called and the *knote* will not be returned as an event.

Locks *must not* be acquired in *f_event*. If a lock is required in *f_event*, it must be obtained in the *kl_lock* function of the *knlist* that the *knote* was added to.

The function **kqfd_register**() will register the *kevent* on the kqueue file descriptor *fd*. If it is safe to sleep, *waitok* should be set.

The function **knote_fdclose**() is used to delete all *knotes* associated with *fd*. Once returned, there will no longer be any *knotes* associated with the *fd*. The *knotes* removed will never be returned from a kevent(2) call, so if userland uses the *knote* to track resources, they will be leaked. The **FILEDESC_LOCK**() lock must be held over the call to **knote_fdclose**() so that file descriptors cannot be added or removed.

The **knlist_***() family of functions are for managing *knotes* associated with an object. A *knlist* is not required, but is commonly used. If used, the *knlist* must be initialized with either **knlist_init**() or **knlist_init_mtx**(). The *knlist* structure may be embedded into the object structure. The *lock* will be held over *f_event* calls.

For the **knlist_init**() function, if *lock* is NULL, a shared global lock will be used and the remaining arguments must be NULL. The function pointers *kl_lock*, *kl_unlock* and *kl_locked* will be used to manipulate the argument *lock*. If any of the function pointers are NULL, a function operating on MTX_DEF style mutex(9) locks will be used instead.

The function **knlist_init_mtx**() may be used to initialize a *knlist* when *lock* is a MTX_DEF style mutex(9) lock.

The function **knlist_empty**() returns true when there are no *knotes* on the list. The function requires that the *lock* be held when called.

The function **knlist_clear**() removes all *knotes* from the list. The *islocked* argument declares if the *lock* has been acquired. All *knotes* will have EV_ONESHOT set so that the *knote* will be returned and removed during the next scan. The *f_detach* function will be called when the *knote* is deleted during the next scan. This function must not be used when *f_isfd* is set in *struct filterops*, as the *td* argument of **fdrop**() will be NULL.

The function **knlist_delete**() removes and deletes all *knotes* on the list. The function *f_detach* will not be called, and the *knote* will not be returned on the next scan. Using this function could leak userland resources if a process uses the *knote* to track resources.

Both the **knlist_clear**() and **knlist_delete**() functions may sleep. They also may release the *lock* to wait for other *knotes* to drain.

The **knlist_destroy**() function is used to destroy a *knlist*. There must be no *knotes* associated with the *knlist* (**knlist_empty**() returns true) and no more *knotes* may be attached to the object. A *knlist* may be emptied by calling **knlist_clear**() or **knlist_delete**().

The macros **KNOTE_LOCKED**() and **KNOTE_UNLOCKED**() are used to notify *knotes* about events associated with the object. It will iterate over all *knotes* on the list calling the *f_event* function associated with the *knote*. The macro **KNOTE_LOCKED**() must be used if the lock associated with the *knl* is held. The function **KNOTE_UNLOCKED**() will acquire the lock before iterating over the list of *knotes*.

## RETURN VALUES

The function **kqueue_add_filteropts**() will return zero on success, EINVAL in the case of an invalid *filt*, or EEXIST if the filter has already been installed.

The function **kqueue_del_filteropts**() will return zero on success, EINVAL in the case of an invalid *filt*, or EBUSY if the filter is still in use.

The function **kqfd_register**() will return zero on success, EBADF if the file descriptor is not a kqueue, or any of the possible values returned by kevent(2).

## SEE ALSO

kevent(2), kqueue(2)

## AUTHORS

This manual page was written by John-Mark Gurney <*jmg@FreeBSD.org*>.