NAME

alq, **alq_open_flags**, **alq_open**, **alq_writen**, **alq_write**, **alq_flush**, **alq_close**, **alq_getn**, **alq_get**, **alq_post_flags**, **alq_post** - Asynchronous Logging Queues

SYNOPSIS

#include <sys/alq.h>

int
alq_open_flags(*struct alq **app, const char *file, struct ucred *cred, int cmode, int size, int flags*);

int
alq_open(*struct alq **app, const char *file, struct ucred *cred, int cmode, int size, int count*);

int
alq_writen(*struct alq *alq, void *data, int len, int flags*);

int
alq_write(*struct alq *alq, void *data, int flags*);

void
alq_flush(*struct alq *alq*);

void
alq_close(*struct alq *alq*);

*struct ale **
alq_getn(*struct alq *alq, int len, int flags*);

*struct ale **
alq_get(*struct alq *alq, int flags*);

void
alq_post_flags(*struct alq *alq, struct ale *ale, int flags*);

void
alq_post(*struct alq *alq, struct ale *ale*);

DESCRIPTION

The **alq** facility provides an asynchronous fixed or variable length recording mechanism, known as Asynchronous Logging Queues. It can record to any vnode(9), thus providing the ability to journal logs

to character devices as well as regular files. All functions accept a *struct alq* argument, which is an opaque type that maintains state information for an Asynchronous Logging Queue. The logging facility runs in a separate kernel thread, which services all log entry requests.

An "asynchronous log entry" is defined as *struct ale*, which has the following members:

```
struct ale {
        intptr_t    ae_bytesused;       /* # bytes written to ALE. */
        char                *ae_data; /* Write ptr. */
        int                 ae_pad;             /* Unused, compat. */
};
```

An **alq** can be created in either fixed or variable length mode. A variable length **alq** accommodates writes of varying length using **alq_writen**() and **alq_getn**(). A fixed length **alq** accommodates a fixed number of writes using **alq_write**() and **alq_get**(), each of fixed size (set at queue creation time). Fixed length mode is deprecated in favour of variable length mode.

FUNCTIONS

The **alq_open_flags**() function creates a new variable length asynchronous logging queue. The *file* argument is the name of the file to open for logging. If the file does not yet exist, **alq_open**() will attempt to create it. The *cmode* argument will be passed to **vn_open**() as the requested creation mode, to be used if the file will be created by **alq_open**(). Consumers of this API may wish to pass ALQ_DEFAULT_CMODE, a default creation mode suitable for most applications. The *cred* argument specifies the credentials to use when opening and performing I/O on the file. The *size* argument sets the size (in bytes) of the underlying queue. The ALQ_ORDERED flag may be passed in via *flags* to indicate that the ordering of writer threads waiting for a busy **alq** to free up resources should be preserved.

The deprecated **alq_open**() function is implemented as a wrapper around **alq_open_flags**() to provide backwards compatibility to consumers that have not been updated to utilise the newer **alq_open_flags**() function. It passes all arguments through to **alq_open_flags**() untouched except for *size* and *count*, and sets *flags* to 0. To create a variable length mode **alq**, the *size* argument should be set to the size (in bytes) of the underlying queue and the *count* argument should be set to 0. To create a fixed length mode **alq**, the *size* argument should be set to the size (in bytes) of each write and the *count* argument should be set to the number of *size* byte chunks to reserve capacity for.

The **alq_writen**() function writes *len* bytes from *data* to the designated variable length mode queue *alq*. If **alq_writen**() could not write the entry immediately and ALQ_WAITOK is set in *flags*, the function will be allowed to msleep_spin(9) with the "alqwnord" or "alqwnres" wait message. A write will automatically schedule the queue *alq* to be flushed to disk. This behaviour can be controlled by passing

ALQ_NOACTIVATE via *flags* to indicate that the write should not schedule *alq* to be flushed to disk.

The deprecated **alq_write**() function is implemented as a wrapper around **alq_writen**() to provide backwards compatibility to consumers that have not been updated to utilise variable length mode queues. The function will write *size* bytes of data (where *size* was specified at queue creation time) from the *data* buffer to the *alq*. Note that it is an error to call **alq_write**() on a variable length mode queue.

The **alq_flush**() function is used for flushing *alq* to the log medium that was passed to **alq_open**(). If *alq* has data to flush and is not already in the process of being flushed, the function will block doing IO. Otherwise, the function will return immediately.

The **alq_close**() function will close the asynchronous logging queue *alq* and flush all pending write requests to the log medium. It will free all resources that were previously allocated.

The **alq_getn**() function returns an asynchronous log entry from *alq*, initialised to point at a buffer capable of receiving *len* bytes of data. This function leaves *alq* in a locked state, until a subsequent **alq_post**() or **alq_post_flags**() call is made. If **alq_getn**() could not obtain *len* bytes of buffer immediately and ALQ_WAITOK is set in *flags*, the function will be allowed to msleep_spin(9) with the "alqgnord" or "alqgnres" wait message. The caller can choose to write less than *len* bytes of data to the returned asynchronous log entry by setting the entry's ae_bytesused field to the number of bytes actually written. This must be done prior to calling **alq_post**().

The deprecated **alq_get**() function is implemented as a wrapper around **alq_getn**() to provide backwards compatibility to consumers that have not been updated to utilise variable length mode queues. The asynchronous log entry returned will be initialised to point at a buffer capable of receiving *size* bytes of data (where *size* was specified at queue creation time). Note that it is an error to call **alq_get**() on a variable length mode queue.

The **alq_post_flags**() function schedules the asynchronous log entry *ale* (obtained from **alq_getn**() or **alq_get**()) for writing to *alq*. The ALQ_NOACTIVATE flag may be passed in via *flags* to indicate that the queue should not be immediately scheduled to be flushed to disk. This function leaves *alq* in an unlocked state.

The **alq_post**() function is implemented as a wrapper around **alq_post_flags**() to provide backwards compatibility to consumers that have not been updated to utilise the newer **alq_post_flags**() function. It simply passes all arguments through to **alq_post_flags**() untouched, and sets *flags* to 0.

IMPLEMENTATION NOTES

The **alq_writen**() and **alq_write**() functions both perform a bcopy(3) from the supplied *data* buffer into

the underlying **alq** buffer. Performance critical code paths may wish to consider using **alq_getn**() (variable length queues) or **alq_get**() (fixed length queues) to avoid the extra memory copy. Note that a queue remains locked between calls to **alq_getn**() or **alq_get**() and **alq_post**() or **alq_post_flags**(), so this method of writing to a queue is unsuitable for situations where the time between calls may be substantial.

LOCKING

Each asynchronous logging queue is protected by a spin mutex.

Functions **alq_flush**() and **alq_open**() may attempt to acquire an internal sleep mutex, and should consequently not be used in contexts where sleeping is not allowed.

RETURN VALUES

The **alq_open**() function returns one of the error codes listed in open(2), if it fails to open *file*, or else it returns 0.

The **alq_writen**() and **alq_write**() functions return EWOULDBLOCK if ALQ_NOWAIT was set in *flags* and either the queue is full or the system is shutting down.

The **alq_getn**() and **alq_get**() functions return NULL if ALQ_NOWAIT was set in *flags* and either the queue is full or the system is shutting down.

NOTE: invalid arguments to non-void functions will result in undefined behaviour.

SEE ALSO

syslog(3), kproc(9), ktr(9), msleep_spin(9), vnode(9)

HISTORY

The Asynchronous Logging Queues (ALQ) facility first appeared in FreeBSD 5.0.

AUTHORS

The **alq** facility was written by Jeffrey Roberson <*jeff@FreeBSD.org*> and extended by Lawrence Stewart <*lstewart@freebsd.org*>.

This manual page was written by Hiten Pandya <*hmp@FreeBSD.org*> and revised by Lawrence Stewart <*lstewart@freebsd.org*>.