NAME

aio - asynchronous I/O

DESCRIPTION

The **aio** facility provides system calls for asynchronous I/O. Asynchronous I/O operations are not completed synchronously by the calling thread. Instead, the calling thread invokes one system call to request an asynchronous I/O operation. The status of a completed request is retrieved later via a separate system call.

Asynchronous I/O operations on some file descriptor types may block an AIO daemon indefinitely resulting in process and/or system hangs. Operations on these file descriptor types are considered "unsafe" and disabled by default. They can be enabled by setting the *vfs.aio.enable_unsafe* sysctl node to a non-zero value.

Asynchronous I/O operations on sockets, raw disk devices, and regular files on local filesystems do not block indefinitely and are always enabled.

The **aio** facility uses kernel processes (also known as AIO daemons) to service most asynchronous I/O requests. These processes are grouped into pools containing a variable number of processes. Each pool will add or remove processes to the pool based on load. Pools can be configured by sysctl nodes that define the minimum and maximum number of processes as well as the amount of time an idle process will wait before exiting.

One pool of AIO daemons is used to service asynchronous I/O requests for sockets. These processes are named "soaiod<N>". The following sysctl nodes are used with this pool:

kern.ipc.aio.num_procs
: The current number of processes in the pool.

kern.ipc.aio.target_procs
: The minimum number of processes that should be present in the pool.

kern.ipc.aio.max_procs
: The maximum number of processes permitted in the pool.

kern.ipc.aio.lifetime
: The amount of time a process is permitted to idle in clock ticks. If a process is idle for this amount of time and there are more processes in the pool than the target minimum, the process will exit.

A second pool of AIO daemons is used to service all other asynchronous I/O requests except for I/O requests to raw disks. These processes are named "aiod<N>". The following sysctl nodes are used with this pool:

vfs.aio.num_aio_procs
The current number of processes in the pool.

vfs.aio.target_aio_procs
The minimum number of processes that should be present in the pool.

vfs.aio.max_aio_procs
The maximum number of processes permitted in the pool.

vfs.aio.aiod_lifetime
The amount of time a process is permitted to idle in clock ticks. If a process is idle for this amount of time and there are more processes in the pool than the target minimum, the process will exit.

Asynchronous I/O requests for raw disks are queued directly to the disk device layer after temporarily wiring the user pages associated with the request. These requests are not serviced by any of the AIO daemon pools.

Several limits on the number of asynchronous I/O requests are imposed both system-wide and per-process. These limits are configured via the following sysctls:

vfs.aio.max_buf_aio
The maximum number of queued asynchronous I/O requests for raw disks permitted for a single process. Asynchronous I/O requests that have completed but whose status has not been retrieved via aio_return(2) or aio_waitcomplete(2) are not counted against this limit.

vfs.aio.num_buf_aio
The number of queued asynchronous I/O requests for raw disks system-wide.

vfs.aio.max_aio_queue_per_proc
The maximum number of asynchronous I/O requests for a single process serviced concurrently by the default AIO daemon pool.

vfs.aio.max_aio_per_proc
The maximum number of outstanding asynchronous I/O requests permitted for a single process. This includes requests that have not been serviced, requests currently being serviced, and

requests that have completed but whose status has not been retrieved via aio_return(2) or aio_waitcomplete(2).

vfs.aio.num_queue_count
The number of outstanding asynchronous I/O requests system-wide.

vfs.aio.max_aio_queue
The maximum number of outstanding asynchronous I/O requests permitted system-wide.

Asynchronous I/O control buffers should be zeroed before initializing individual fields. This ensures all fields are initialized.

All asynchronous I/O control buffers contain a *sigevent* structure in the *aio_sigevent* field which can be used to request notification when an operation completes.

For SIGEV_KEVENT notifications, the *sigevent*'s *sigev_notify_kqueue* field should contain the descriptor of the kqueue that the event should be attached to, its *sigev_notify_kevent_flags* field may contain EV_ONESHOT, EV_CLEAR, and/or EV_DISPATCH, and its *sigev_notify* field should be set to SIGEV_KEVENT. The posted kevent will contain:

Member	**Value**
ident	asynchronous I/O control buffer pointer
filter	EVFILT_AIO
flags	EV_EOF
udata	value stored in *aio_sigevent.sigev_value*

For SIGEV_SIGNO and SIGEV_THREAD_ID notifications, the information for the queued signal will include SI_ASYNCIO in the *si_code* field and the value stored in *sigevent.sigev_value* in the *si_value* field.

For SIGEV_THREAD notifications, the value stored in *aio_sigevent.sigev_value* is passed to the *aio_sigevent.sigev_notify_function* as described in sigevent(3).

SEE ALSO

aio_cancel(2), aio_error(2), aio_read(2), aio_readv(2), aio_return(2), aio_suspend(2), aio_waitcomplete(2), aio_write(2), aio_writev(2), lio_listio(2), sigevent(3), sysctl(8)

HISTORY

The **aio** facility appeared as a kernel option in FreeBSD 3.0. The **aio** kernel module appeared in FreeBSD 5.0. The **aio** facility was integrated into all kernels in FreeBSD 11.0.