NAME

g_new_bio, **g_clone_bio**, **g_destroy_bio**, **g_format_bio**, **g_print_bio**, **g_reset_bio** - GEOM bio controlling functions

SYNOPSIS

#include <sys/bio.h>
#include <geom/geom.h>

*struct bio **
g_new_bio(*void*);

*struct bio **
g_alloc_bio(*void*);

*struct bio **
g_clone_bio(*struct bio *bp*);

*struct bio **
g_duplicate_bio(*struct bio *bp*);

void
g_destroy_bio(*struct bio *bp*);

void
g_format_bio(*struct sbuf *sb*, *const struct bio *bp*);

void
g_print_bio(*struct sbuf *sb*, *const char *prefix*, *const struct bio *bp*, *const char *fmtsuffix*, ...);

void
g_reset_bio(*struct bio *bp*);

DESCRIPTION

A *struct bio* is used by GEOM to describe I/O requests, its most important fields are described below:

bio_cmd I/O request command. There are five I/O requests available in GEOM:

BIO_READ A read request.

BIO_WRITE A write request.

BIO_DELETE Indicates that a certain range of data is no longer used and that it can be erased or freed as the underlying technology supports. Technologies like flash adaptation layers can arrange to erase the relevant blocks before they will become reassigned and cryptographic devices may want to fill random bits into the range to reduce the amount of data available for attack.

BIO_GETATTR Inspect and manipulate out-of-band attributes on a particular provider or path. Attributes are named by ascii strings and are stored in the *bio_attribute* field.

BIO_FLUSH Tells underlying providers to flush their write caches.

bio_flags Available flags:

BIO_ERROR Request failed (error value is stored in *bio_error* field).

BIO_DONE Request finished.

bio_cflags Private use by the consumer.

bio_pflags Private use by the provider.

bio_offset Offset into provider.

bio_data Pointer to data buffer.

bio_error Error value when BIO_ERROR is set.

bio_done Pointer to function which will be called when the request is finished.

bio_driver1 Private use by the provider.

bio_driver2 Private use by the provider.

bio_caller1 Private use by the consumer.

bio_caller2 Private use by the consumer.

bio_attribute Attribute string for BIO_GETATTR request.

bio_from Consumer to use for request (attached to provider stored in *bio_to* field) (typically read-only for a class).

bio_to Destination provider (typically read-only for a class).

bio_length Request length in bytes.

bio_completed
Number of bytes completed, but they may not be completed from the front of the request.

bio_children Number of *bio* clones (typically read-only for a class).

bio_inbed Number of finished *bio* clones.

bio_parent Pointer to parent *bio*.

The **g_new_bio**() function allocates a new, empty *bio* structure.

g_alloc_bio() - same as **g_new_bio**(), but always succeeds (allocates bio with the M_WAITOK malloc flag).

The **g_clone_bio**() function allocates a new *bio* structure and copies the following fields from the *bio* given as an argument to clone: *bio_cmd*, *bio_length*, *bio_offset*, *bio_data*, *bio_attribute*. The field *bio_parent* in the clone points to the passed *bio* and the field *bio_children* in the passed *bio* is incremented.

This function should be used for every request which enters through the provider of a particular geom and needs to be scheduled down. Proper order is:

1. Clone the received *struct bio*.
2. Modify the clone.
3. Schedule the clone on its own consumer.

g_duplicate_bio() - same as **g_clone_bio**(), but always succeeds (allocates bio with the M_WAITOK malloc flag).

The **g_destroy_bio**() function deallocates and destroys the given *bio* structure.

The **g_format_bio**() function prints information about the given *bio* structure into the provided *sbuf*.

The **g_print_bio**() function is a convenience wrapper around **g_format_bio**() that can be used for debugging purposes. It prints a provided *prefix* string, followed by the formatted *bio*, followed by a *fmtsuffix* in the style of printf(9). Any of the prefix or suffix strings may be the empty string. **g_print_bio**() always prints a newline character at the end of the line.

The **g_reset_bio**() function resets the given *bio* structure back to its initial state. **g_reset_bio**() preserves internal data structures, while setting all user visible fields to their initial values. When reusing a *bio* obtained from **g_new_bio**(), **g_alloc_bio**(), **g_clone_bio**(), or **g_duplicate_bio**() for multiple transactions, **g_reset_bio**() must be called between the transactions in lieu of **bzero**(). While not strictly required for a *bio* structure created by other means, **g_reset_bio**() should be used to initialize it and between transactions.

RETURN VALUES

The **g_new_bio**() and **g_clone_bio**() functions return a pointer to the allocated *bio*, or NULL if an error occurred.

EXAMPLES

Implementation of "NULL-transformation", meaning that an I/O request is cloned and scheduled down without any modifications. Let us assume that field *ex_consumer* in structure *example_softc* contains a consumer attached to the provider we want to operate on.

```
void
example_start(struct bio *bp)
{
        struct example_softc *sc;
        struct bio *cbp;

        g_print_bio("Request received: ", bp, "");

        sc = bp->bio_to->geom->softc;
        if (sc == NULL) {
                g_io_deliver(bp, ENXIO);
                return;
        }

        /* Let's clone our bio request. */
        cbp = g_clone_bio(bp);
        if (cbp == NULL) {
                g_io_deliver(bp, ENOMEM);
                return;
```

```
                }
                cbp->bio_done = g_std_done;             /* Standard 'done' function. */

                /* Ok, schedule it down. */
                /*
                 * The consumer can be obtained from
                 * LIST_FIRST(&bp->bio_to->geom->consumer) as well,
                 * if there is only one in our geom.
                 */
                g_io_request(cbp, sc->ex_consumer);
        }
```

SEE ALSO

geom(4), DECLARE_GEOM_CLASS(9), g_access(9), g_attach(9), g_consumer(9), g_data(9), g_event(9), g_geom(9), g_provider(9), g_provider_by_name(9), g_wither_geom(9)

AUTHORS

This manual page was written by Pawel Jakub Dawidek <*pjd@FreeBSD.org*>.