NAME

sglist, **sglist_alloc**, **sglist_append**, **sglist_append_bio**, **sglist_append_mbuf**, **sglist_append_mbuf_epg**, **sglist_append_phys**, **sglist_append_sglist**, **sglist_append_single_mbuf**, **sglist_append_uio**, **sglist_append_user**, **sglist_append_vmpages**, **sglist_build**, **sglist_clone**, **sglist_consume_uio**, **sglist_count**, **sglist_count_mbuf_epg**, **sglist_count_vmpages**, **sglist_free**, **sglist_hold**, **sglist_init**, **sglist_join**, **sglist_length**, **sglist_reset**, **sglist_slice**, **sglist_split** - manage a scatter/gather list of physical memory addresses

SYNOPSIS

#include <sys/types.h>
#include <sys/sglist.h>

*struct sglist **
sglist_alloc(*int nsegs*, *int mflags*);

int
sglist_append(*struct sglist *sg*, *void *buf*, *size_t len*);

int
sglist_append_bio(*struct sglist *sg*, *struct bio *bp*);

int
sglist_append_mbuf_epg(*struct sglist *sg*, *struct mbuf *m*, *size_t offset*, *size_t len*);

int
sglist_append_mbuf(*struct sglist *sg*, *struct mbuf *m*);

int
sglist_append_phys(*struct sglist *sg*, *vm_paddr_t paddr*, *size_t len*);

int
sglist_append_sglist(*struct sglist *sg*, *struct sglist *source*, *size_t offset*, *size_t len*);

int
sglist_append_single_mbuf(*struct sglist *sg*, *struct mbuf *m*);

int
sglist_append_uio(*struct sglist *sg*, *struct uio *uio*);

int

sglist_append_user(*struct sglist *sg, void *buf, size_t len, struct thread *td*);

int
sglist_append_vmpages(*struct sglist *sg, vm_page_t *m, size_t pgoff, size_t len*);

*struct sglist **
sglist_build(*void *buf, size_t len, int mflags*);

*struct sglist **
sglist_clone(*struct sglist *sg, int mflags*);

int
sglist_consume_uio(*struct sglist *sg, struct uio *uio, size_t resid*);

int
sglist_count(*void *buf, size_t len*);

int
sglist_count_mbuf_epg(*struct mbuf *m, size_t offset, size_t len*);

int
sglist_count_vmpages(*vm_page_t *m, size_t pgoff, size_t len*);

void
sglist_free(*struct sglist *sg*);

*struct sglist **
sglist_hold(*struct sglist *sg*);

void
sglist_init(*struct sglist *sg, int maxsegs, struct sglist_seg *segs*);

int
sglist_join(*struct sglist *first, struct sglist *second*);

size_t
sglist_length(*struct sglist *sg*);

void
sglist_reset(*struct sglist *sg*);

int
sglist_slice(*struct sglist *original, struct sglist **slice, size_t offset, size_t length, int mflags*);

int
sglist_split(*struct sglist *original, struct sglist **head, size_t length, int mflags*);

DESCRIPTION

The **sglist** API manages physical address ranges. Each list contains one or more elements. Each element contains a starting physical address and a length. Scatter/gather lists are read-only while they are shared. If one wishes to alter an existing scatter/gather list and does not hold the sole reference to the list, then one should create a new list instead of modifying the existing list.

Each scatter/gather list object contains a reference count. New lists are created with a single reference. New references are obtained by calling **sglist_hold** and are released by calling **sglist_free**.

Allocating and Initializing Lists

Each **sglist** object consists of a header structure and a variable-length array of scatter/gather list elements. The **sglist_alloc** function allocates a new list that contains a header and *nsegs* scatter/gather list elements. The *mflags* argument can be set to either M_NOWAIT or M_WAITOK.

The **sglist_count** function returns the number of scatter/gather list elements needed to describe the physical address ranges mapped by a single kernel virtual address range. The kernel virtual address range starts at *buf* and is *len* bytes long.

The **sglist_count_mbuf_epg** function returns the number of scatter/gather list elements needed to describe the external multipage mbuf buffer *m*. The ranges start at an offset of *offset* relative to the start of the buffer and is *len* bytes long.

The **sglist_count_vmpages** function returns the number of scatter/gather list elements needed to describe the physical address ranges of a buffer backed by an array of virtual memory pages *m*. The buffer starts at an offset of *pgoff* bytes relative to the first page and is *len* bytes long.

The **sglist_build** function allocates a new scatter/gather list object that describes the physical address ranges mapped by a single kernel virtual address range. The kernel virtual address range starts at *buf* and is *len* bytes long. The *mflags* argument can be set to either M_NOWAIT or M_WAITOK.

The **sglist_clone** function returns a copy of an existing scatter/gather list object *sg*. The *mflags* argument can be set to either M_NOWAIT or M_WAITOK. This can be used to obtain a private copy of a scatter/gather list before modifying it.

The **sglist_init** function initializes a scatter/gather list header. The header is pointed to by *sg* and is initialized to manage an array of *maxsegs* scatter/gather list elements pointed to by *segs*. This can be used to initialize a scatter/gather list header whose storage is not provided by **sglist_alloc**. In that case, the caller should not call **sglist_free** to release its own reference and is responsible for ensuring all other references to the list are dropped before it releases the storage for *sg* and *segs*.

Constructing Scatter/Gather Lists

The **sglist** API provides several routines for building a scatter/gather list to describe one or more objects. Specifically, the **sglist_append** family of routines can be used to append the physical address ranges described by an object to the end of a scatter/gather list. All of these routines return 0 on success or an error on failure. If a request to append an address range to a scatter/gather list fails, the scatter/gather list will remain unchanged.

The **sglist_append** function appends the physical address ranges described by a single kernel virtual address range to the scatter/gather list *sg*. The kernel virtual address range starts at *buf* and is *len* bytes long.

The **sglist_append_bio** function appends the physical address ranges described by a single bio *bp* to the scatter/gather list *sg*.

The **sglist_append_mbuf_epg** function appends the physical address ranges described by the external multipage mbuf(9) buffer *ext_pgs* to the scatter/gather list *sg*. The physical address ranges start at offset *offset* within *ext_pgs* and continue for *len* bytes. Note that unlike **sglist_append_mbuf**, **sglist_append_mbuf_epg** only adds ranges for a single mbuf, not an entire mbuf chain.

The **sglist_append_mbuf** function appends the physical address ranges described by an entire mbuf chain *m* to the scatter/gather list *sg*.

The **sglist_append_mbuf** function appends the physical address ranges described by a single mbuf *m* to the scatter/gather list *sg*.

The **sglist_append_phys** function appends a single physical address range to the scatter/gather list *sg*. The physical address range starts at *paddr* and is *len* bytes long.

The **sglist_append_sglist** function appends physical address ranges described by the scatter/gather list *source* to the scatter/gather list *sg*. The physical address ranges start at offset *offset* within *source* and continue for *len* bytes.

The **sglist_append_uio** function appends the physical address ranges described by a uio(9) object to the scatter/gather list *sg*. Note that it is the caller's responsibility to ensure that the pages backing the I/O

request are wired for the lifetime of *sg*. Note also that this routine does not modify *uio*.

The **sglist_append_user** function appends the physical address ranges described by a single user virtual address range to the scatter/gather list *sg*. The user virtual address range is relative to the address space of the thread *td*. It starts at *buf* and is *len* bytes long. Note that it is the caller's responsibility to ensure that the pages backing the user buffer are wired for the lifetime of *sg*.

The **sglist_append_vmpages** function appends the physical address ranges of a buffer backed by an array of virtual memory pages *m*. The buffer starts at an offset of *pgoff* bytes relative to the first page and is *len* bytes long.

The **sglist_consume_uio** function is a variation of **sglist_append_uio**. As with **sglist_append_uio**, it appends the physical address ranges described by *uio* to the scatter/gather list *sg*. Unlike **sglist_append_uio**, however, **sglist_consume_uio** modifies the I/O request to indicate that the appended address ranges have been processed similar to calling uiomove(9). This routine will only append ranges that describe up to *resid* total bytes in length. If the available segments in the scatter/gather list are exhausted before *resid* bytes are processed, then the *uio* structure will be updated to reflect the actual number of bytes processed, and **sglist_consume_io** will return zero to indicate success. In effect, this function will perform partial reads or writes. The caller can compare the *uio_resid* member of *uio* before and after calling **sglist_consume_uio** to determine the actual number of bytes processed.

Manipulating Scatter/Gather Lists

The **sglist_join** function appends physical address ranges from the scatter/gather list *second* onto *first* and then resets *second* to an empty list. It returns zero on success or an error on failure.

The **sglist_split** function splits an existing scatter/gather list into two lists. The first *length* bytes described by the list *original* are moved to a new list **head*. If *original* describes a total address range that is smaller than *length* bytes, then all of the address ranges will be moved to the new list at **head* and *original* will be an empty list. The caller may supply an existing scatter/gather list in **head*. If so, the list must be empty. Otherwise, the caller may set **head* to NULL in which case a new scatter/gather list will be allocated. In that case, *mflags* may be set to either M_NOWAIT or M_WAITOK. Note that since the *original* list is modified by this call, it must be a private list with no other references. The **sglist_split** function returns zero on success or an error on failure.

The **sglist_slice** function generates a new scatter/gather list from a sub-range of an existing scatter/gather list *original*. The sub-range to extract is specified by the *offset* and *length* parameters. The new scatter/gather list is stored in **slice*. As with *head* for **sglist_join**, the caller may either provide an empty scatter/gather list, or it may set **slice* to NULL in which case **sglist_slice** will allocate a new list subject to *mflags*. Unlike **sglist_split**, **sglist_slice** does not modify *original* and does not require it to be a private list. The **sglist_split** function returns zero on success or an error on failure.

Miscellaneous Routines

The **sglist_reset** function clears the scatter/gather list *sg* so that it no longer maps any address ranges. This can allow reuse of a single scatter/gather list object for multiple requests.

The **sglist_length** function returns the total length of the physical address ranges described by the scatter/gather list *sg*.

RETURN VALUES

The **sglist_alloc**, **sglist_build**, and **sglist_clone** functions return a new scatter/gather list on success or NULL on failure.

The **sglist_append** family of functions and the **sglist_consume_uio**, **sglist_join**, **sglist_slice**, and **sglist_split** functions return zero on success or an error on failure.

The **sglist_count** family of functions return a count of scatter/gather list elements.

The **sglist_length** function returns a count of address space described by a scatter/gather list in bytes.

ERRORS

The **sglist_append** functions return the following errors on failure:

[EINVAL] The scatter/gather list has zero segments.

[EFBIG] There are not enough available segments in the scatter/gather list to append the specified physical address ranges.

The **sglist_consume_uio** function returns the following error on failure:

[EINVAL] The scatter/gather list has zero segments.

The **sglist_join** function returns the following error on failure:

[EFBIG] There are not enough available segments in the scatter/gather list *first* to append the physical address ranges from *second*.

The **sglist_slice** function returns the following errors on failure:

[EINVAL] The *original* scatter/gather list does not describe enough address space to cover the requested sub-range.

[EINVAL] The caller-supplied scatter/gather list in **slice* is not empty.

[ENOMEM] An attempt to allocate a new scatter/gather list with M_NOWAIT set in *mflags* failed.

[EFBIG] There are not enough available segments in the caller-supplied scatter/gather list in **slice* to describe the requested physical address ranges.

The **sglist_split** function returns the following errors on failure:

[EDOOFUS] The *original* scatter/gather list has more than one reference.

[EINVAL] The caller-supplied scatter/gather list in **head* is not empty.

[ENOMEM] An attempt to allocate a new scatter/gather list with M_NOWAIT set in *mflags* failed.

[EFBIG] There are not enough available segments in the caller-supplied scatter/gather list in **head* to describe the requested physical address ranges.

SEE ALSO

g_bio(9), malloc(9), mbuf(9), uio(9)

HISTORY

This API was first introduced in FreeBSD 8.0.