NAME

mbuf - memory management in the kernel IPC subsystem

SYNOPSIS

#include <sys/param.h>
#include <sys/systm.h>
#include <sys/mbuf.h>

Mbuf allocation macros

MGET(*struct mbuf *mbuf*, *int how*, *short type*);

MGETHDR(*struct mbuf *mbuf*, *int how*, *short type*);

int
MCLGET(*struct mbuf *mbuf*, *int how*);

MEXTADD(*struct mbuf *mbuf*, *char *buf*, *u_int size*, *void (*free)(struct mbuf *)*, *void *opt_arg1*, *void *opt_arg2*, *int flags*, *int type*);

Mbuf utility macros

type
mtod(*struct mbuf *mbuf*, *type*);

*void **
mtodo(*struct mbuf *mbuf*, *offset*);

M_ALIGN(*struct mbuf *mbuf*, *u_int len*);

MH_ALIGN(*struct mbuf *mbuf*, *u_int len*);

int
M_LEADINGSPACE(*struct mbuf *mbuf*);

int
M_TRAILINGSPACE(*struct mbuf *mbuf*);

M_MOVE_PKTHDR(*struct mbuf *to*, *struct mbuf *from*);

M_PREPEND(*struct mbuf *mbuf*, *int len*, *int how*);

MCHTYPE(*struct mbuf *mbuf*, *short type*);

int
M_WRITABLE(*struct mbuf *mbuf*);

Mbuf allocation functions

*struct mbuf **
m_get(*int how*, *short type*);

*struct mbuf **
m_get2(*int size*, *int how*, *short type*, *int flags*);

*struct mbuf **
m_get3(*int size*, *int how*, *short type*, *int flags*);

*struct mbuf **
m_getm(*struct mbuf *orig*, *int len*, *int how*, *short type*);

*struct mbuf **
m_getjcl(*int how*, *short type*, *int flags*, *int size*);

*struct mbuf **
m_getcl(*int how*, *short type*, *int flags*);

*struct mbuf **
m_gethdr(*int how*, *short type*);

*struct mbuf **
m_free(*struct mbuf *mbuf*);

void
m_freem(*struct mbuf *mbuf*);

Mbuf utility functions

void
m_adj(*struct mbuf *mbuf*, *int len*);

void
m_align(*struct mbuf *mbuf*, *int len*);

int
m_append(*struct mbuf *mbuf*, *int len*, *c_caddr_t cp*);

*struct mbuf **
m_prepend(*struct mbuf *mbuf*, *int len*, *int how*);

*struct mbuf **
m_copyup(*struct mbuf *mbuf*, *int len*, *int dstoff*);

*struct mbuf **
m_pullup(*struct mbuf *mbuf*, *int len*);

*struct mbuf **
m_pulldown(*struct mbuf *mbuf*, *int offset*, *int len*, *int *offsetp*);

*struct mbuf **
m_copym(*struct mbuf *mbuf*, *int offset*, *int len*, *int how*);

*struct mbuf **
m_copypacket(*struct mbuf *mbuf*, *int how*);

*struct mbuf **
m_dup(*const struct mbuf *mbuf*, *int how*);

void
m_copydata(*const struct mbuf *mbuf*, *int offset*, *int len*, *caddr_t buf*);

void
m_copyback(*struct mbuf *mbuf*, *int offset*, *int len*, *caddr_t buf*);

*struct mbuf **
m_devget(*char *buf*, *int len*, *int offset*, *struct ifnet *ifp*, *void (*copy)(char *from, caddr_t to, u_int len)*);

void
m_cat(*struct mbuf *m*, *struct mbuf *n*);

void
m_catpkt(*struct mbuf *m*, *struct mbuf *n*);

u_int

m_fixhdr(*struct mbuf *mbuf*);

int
m_dup_pkthdr(*struct mbuf *to*, *const struct mbuf *from*, *int how*);

void
m_move_pkthdr(*struct mbuf *to*, *struct mbuf *from*);

u_int
m_length(*struct mbuf *mbuf*, *struct mbuf **last*);

*struct mbuf **
m_split(*struct mbuf *mbuf*, *int len*, *int how*);

int
m_apply(*struct mbuf *mbuf*, *int off*, *int len*, *int (*f)(void *arg, void *data, u_int len)*, *void *arg*);

*struct mbuf **
m_getptr(*struct mbuf *mbuf*, *int loc*, *int *off*);

*struct mbuf **
m_defrag(*struct mbuf *m0*, *int how*);

*struct mbuf **
m_collapse(*struct mbuf *m0*, *int how*, *int maxfrags*);

*struct mbuf **
m_unshare(*struct mbuf *m0*, *int how*);

DESCRIPTION

An *mbuf* is a basic unit of memory management in the kernel IPC subsystem. Network packets and socket buffers are stored in *mbufs*. A network packet may span multiple *mbufs* arranged into a *mbuf chain* (linked list), which allows adding or trimming network headers with little overhead.

While a developer should not bother with *mbuf* internals without serious reason in order to avoid incompatibilities with future changes, it is useful to understand the general structure of an *mbuf*.

An *mbuf* consists of a variable-sized header and a small internal buffer for data. The total size of an *mbuf*, MSIZE, is a constant defined in <*sys/param.h*>. The *mbuf* header includes:

m_next	(*struct mbuf* *) A pointer to the next *mbuf* in the *mbuf chain*.
m_nextpkt	(*struct mbuf* *) A pointer to the next *mbuf chain* in the queue.
m_data	(*caddr_t*) A pointer to data attached to this *mbuf*.
m_len	(*int*) The length of the data.
m_type	(*short*) The type of the data.
m_flags	(*int*) The *mbuf* flags.

The *mbuf* flag bits are defined as follows:

```
#define    M_EXT              0x00000001 /* has associated external storage */
#define    M_PKTHDR           0x00000002 /* start of record */
#define    M_EOR              0x00000004 /* end of record */
#define    M_RDONLY           0x00000008 /* associated data marked read-only */
#define    M_BCAST                      0x00000010 /* send/received as link-level broadcast */
#define    M_MCAST                      0x00000020 /* send/received as link-level multicast */
#define    M_PROMISC          0x00000040 /* packet was not for us */
#define    M_VLANTAG          0x00000080 /* ether_vtag is valid */
#define    M_EXTPG                      0x00000100 /* has array of unmapped pages and TLS */
#define    M_NOFREE           0x00000200 /* do not free mbuf, embedded in cluster */
#define    M_TSTMP                      0x00000400 /* rcv_tstmp field is valid */
#define    M_TSTMP_HPREC                0x00000800 /* rcv_tstmp is high-prec, typically
                                            hw-stamped on port (useful for IEEE 1588
                                            and 802.1AS) */

#define    M_PROTO1           0x00001000 /* protocol-specific */
#define    M_PROTO2           0x00002000 /* protocol-specific */
#define    M_PROTO3           0x00004000 /* protocol-specific */
#define    M_PROTO4           0x00008000 /* protocol-specific */
#define    M_PROTO5           0x00010000 /* protocol-specific */
#define    M_PROTO6           0x00020000 /* protocol-specific */
#define    M_PROTO7           0x00040000 /* protocol-specific */
#define    M_PROTO8           0x00080000 /* protocol-specific */
#define    M_PROTO9           0x00100000 /* protocol-specific */
#define    M_PROTO10          0x00200000 /* protocol-specific */
#define    M_PROTO11          0x00400000 /* protocol-specific */
```

```
#define  M_PROTO12     0x00800000 /* protocol-specific */
```

The available *mbuf* types are defined as follows:

```
#define  MT_DATA              1         /* dynamic (data) allocation */
#define  MT_HEADER     MT_DATA          /* packet header */

#define  MT_VENDOR1    4        /* for vendor-internal use */
#define  MT_VENDOR2    5        /* for vendor-internal use */
#define  MT_VENDOR3    6        /* for vendor-internal use */
#define  MT_VENDOR4    7        /* for vendor-internal use */

#define  MT_SONAME     8        /* socket name */

#define  MT_EXP1              9         /* for experimental use */
#define  MT_EXP2              10        /* for experimental use */
#define  MT_EXP3              11        /* for experimental use */
#define  MT_EXP4              12        /* for experimental use */

#define  MT_CONTROL    14       /* extra-data protocol message */
#define  MT_EXTCONTROL        15        /* control message with externalized contents */
#define  MT_OOBDATA    16       /* expedited data  */
```

The available external buffer types are defined as follows:

```
#define  EXT_CLUSTER   1        /* mbuf cluster */
#define  EXT_SFBUF     2        /* sendfile(2)'s sf_bufs */
#define  EXT_JUMBOP    3        /* jumbo cluster 4096 bytes */
#define  EXT_JUMBO9    4        /* jumbo cluster 9216 bytes */
#define  EXT_JUMBO16   5        /* jumbo cluster 16184 bytes */
#define  EXT_PACKET    6        /* mbuf+cluster from packet zone */
#define  EXT_MBUF      7        /* external mbuf reference */
#define  EXT_RXRING    8        /* data in NIC receive ring */
#define  EXT_PGS              9         /* array of unmapped pages */

#define  EXT_VENDOR1   224      /* for vendor-internal use */
#define  EXT_VENDOR2   225      /* for vendor-internal use */
#define  EXT_VENDOR3   226      /* for vendor-internal use */
#define  EXT_VENDOR4   227      /* for vendor-internal use */
```

```
#define   EXT_EXP1         244       /* for experimental use */
#define   EXT_EXP2         245       /* for experimental use */
#define   EXT_EXP3         246       /* for experimental use */
#define   EXT_EXP4         247       /* for experimental use */

#define   EXT_NET_DRV      252       /* custom ext_buf provided by net driver(s) */
#define   EXT_MOD_TYPE 253           /* custom module's ext_buf type */
#define   EXT_DISPOSABLE             254       /* can throw this buffer away w/page flipping */
#define   EXT_EXTREF       255       /* has externally maintained ref_cnt ptr */
```

If the M_PKTHDR flag is set, a *struct pkthdr m_pkthdr* is added to the *mbuf* header. It contains a pointer to the interface the packet has been received from (*struct ifnet *rcvif*), and the total packet length (*int len*). Optionally, it may also contain an attached list of packet tags (*struct m_tag*). See mbuf_tags(9) for details. Fields used in offloading checksum calculation to the hardware are kept in *m_pkthdr* as well. See *HARDWARE-ASSISTED CHECKSUM CALCULATION* for details.

If small enough, data is stored in the internal data buffer of an *mbuf*. If the data is sufficiently large, another *mbuf* may be added to the *mbuf chain*, or external storage may be associated with the *mbuf*. MHLEN bytes of data can fit into an *mbuf* with the M_PKTHDR flag set, MLEN bytes can otherwise.

If external storage is being associated with an *mbuf*, the *m_ext* header is added at the cost of losing the internal data buffer. It includes a pointer to external storage, the size of the storage, a pointer to a function used for freeing the storage, a pointer to an optional argument that can be passed to the function, and a pointer to a reference counter. An *mbuf* using external storage has the M_EXT flag set.

The system supplies a macro for allocating the desired external storage buffer, MEXTADD.

The allocation and management of the reference counter is handled by the subsystem.

The system also supplies a default type of external storage buffer called an *mbuf cluster*. *Mbuf clusters* can be allocated and configured with the use of the MCLGET macro. Each *mbuf cluster* is MCLBYTES in size, where MCLBYTES is a machine-dependent constant. The system defines an advisory macro MINCLSIZE, which is the smallest amount of data to put into an *mbuf cluster*. It is equal to MHLEN plus one. It is typically preferable to store data into the data region of an *mbuf*, if size permits, as opposed to allocating a separate *mbuf cluster* to hold the same data.

Macros and Functions

There are numerous predefined macros and functions that provide the developer with common utilities.

mtod(*mbuf*, *type*)

Convert an *mbuf* pointer to a data pointer. The macro expands to the data pointer cast to the specified *type*. **Note**: It is advisable to ensure that there is enough contiguous data in *mbuf*. See **m_pullup**() for details.

mtodo(*mbuf*, *offset*)
Return a data pointer at an offset (in bytes) into the data attached to *mbuf*. Returns a *void* * pointer . **Note**: The caller must ensure that the offset is in bounds of the attached data.

MGET(*mbuf*, *how*, *type*)
Allocate an *mbuf* and initialize it to contain internal data. *mbuf* will point to the allocated *mbuf* on success, or be set to NULL on failure. The *how* argument is to be set to M_WAITOK or M_NOWAIT. It specifies whether the caller is willing to block if necessary. A number of other functions and macros related to *mbufs* have the same argument because they may at some point need to allocate new *mbufs*.

MGETHDR(*mbuf*, *how*, *type*)
Allocate an *mbuf* and initialize it to contain a packet header and internal data. See **MGET**() for details.

MEXTADD(*mbuf*, *buf*, *size*, *free*, *opt_arg1*, *opt_arg2*, *flags*, *type*)
Associate externally managed data with *mbuf*. Any internal data contained in the mbuf will be discarded, and the M_EXT flag will be set. The *buf* and *size* arguments are the address and length, respectively, of the data. The *free* argument points to a function which will be called to free the data when the mbuf is freed; it is only used if *type* is EXT_EXTREF. The *opt_arg1* and *opt_arg2* arguments will be saved in *ext_arg1* and *ext_arg2* fields of the *struct m_ext* of the mbuf. The *flags* argument specifies additional *mbuf* flags; it is not necessary to specify M_EXT. Finally, the *type* argument specifies the type of external data, which controls how it will be disposed of when the *mbuf* is freed. In most cases, the correct value is EXT_EXTREF.

MCLGET(*mbuf*, *how*)
Allocate and attach an *mbuf cluster* to *mbuf*. On success, a non-zero value returned; otherwise, 0. Historically, consumers would check for success by testing the M_EXT flag on the mbuf, but this is now discouraged to avoid unnecessary awareness of the implementation of external storage in protocol stacks and device drivers.

M_ALIGN(*mbuf*, *len*)
Set the pointer *mbuf->m_data* to place an object of the size *len* at the end of the internal data area of *mbuf*, long word aligned. Applicable only if *mbuf* is newly allocated with **MGET**() or **m_get**().

MH_ALIGN(*mbuf*, *len*)

Serves the same purpose as **M_ALIGN**() does, but only for *mbuf* newly allocated with **MGETHDR**() or **m_gethdr**(), or initialized by **m_dup_pkthdr**() or **m_move_pkthdr**().

m_align(*mbuf*, *len*)
Services the same purpose as **M_ALIGN**() but handles any type of mbuf.

M_LEADINGSPACE(*mbuf*)
Returns the number of bytes available before the beginning of data in *mbuf*.

M_TRAILINGSPACE(*mbuf*)
Returns the number of bytes available after the end of data in *mbuf*.

M_PREPEND(*mbuf*, *len*, *how*)
This macro operates on an *mbuf chain*. It is an optimized wrapper for **m_prepend**() that can make use of possible empty space before data (e.g. left after trimming of a link-layer header). The new *mbuf chain* pointer or NULL is in *mbuf* after the call.

M_MOVE_PKTHDR(*to*, *from*)
Using this macro is equivalent to calling **m_move_pkthdr**(*to*, *from*).

M_WRITABLE(*mbuf*)
This macro will evaluate true if *mbuf* is not marked M_RDONLY and if either *mbuf* does not contain external storage or, if it does, then if the reference count of the storage is not greater than 1. The M_RDONLY flag can be set in *mbuf->m_flags*. This can be achieved during setup of the external storage, by passing the M_RDONLY bit as a *flags* argument to the **MEXTADD**() macro, or can be directly set in individual *mbufs*.

MCHTYPE(*mbuf*, *type*)
Change the type of *mbuf* to *type*. This is a relatively expensive operation and should be avoided.

The functions are:

m_get(*how*, *type*)
A function version of **MGET**() for non-critical paths.

m_get2(*size*, *how*, *type*, *flags*)
Allocate an *mbuf* with enough space to hold specified amount of data. If the size is larger than MJUMPAGESIZE, NULL will be returned.

m_get3(*size*, *how*, *type*, *flags*)

Allocate an *mbuf* with enough space to hold specified amount of data. If the size is larger than MJUM16BYTES, NULL will be returned.

m_getm(*orig*, *len*, *how*, *type*)
Allocate *len* bytes worth of *mbufs* and *mbuf clusters* if necessary and append the resulting allocated *mbuf chain* to the *mbuf chain orig*, if it is non-NULL. If the allocation fails at any point, free whatever was allocated and return NULL. If *orig* is non-NULL, it will not be freed. It is possible to use **m_getm**() to either append *len* bytes to an existing *mbuf* or *mbuf chain* (for example, one which may be sitting in a pre-allocated ring) or to simply perform an all-or-nothing *mbuf* and *mbuf cluster* allocation.

m_gethdr(*how*, *type*)
A function version of **MGETHDR**() for non-critical paths.

m_getcl(*how*, *type*, *flags*)
Fetch an *mbuf* with a *mbuf cluster* attached to it. If one of the allocations fails, the entire allocation fails. This routine is the preferred way of fetching both the *mbuf* and *mbuf cluster* together, as it avoids having to unlock/relock between allocations. Returns NULL on failure.

m_getjcl(*how*, *type*, *flags*, *size*)
This is like **m_getcl**() but the specified *size* of the cluster to be allocated must be one of MCLBYTES, MJUMPAGESIZE, MJUM9BYTES, or MJUM16BYTES.

m_free(*mbuf*)
Frees *mbuf*. Returns *m_next* of the freed *mbuf*.

The functions below operate on *mbuf chains*.

m_freem(*mbuf*)
Free an entire *mbuf chain*, including any external storage.

m_adj(*mbuf*, *len*)
Trim *len* bytes from the head of an *mbuf chain* if *len* is positive, from the tail otherwise.

m_append(*mbuf*, *len*, *cp*)
Append *len* bytes of data *cp* to the *mbuf chain*. Extend the mbuf chain if the new data does not fit in existing space.

m_prepend(*mbuf*, *len*, *how*)
Allocate a new *mbuf* and prepend it to the *mbuf chain*, handle M_PKTHDR properly. **Note**: It

does not allocate any *mbuf clusters*, so *len* must be less than MLEN or MHLEN, depending on the M_PKTHDR flag setting.

m_copyup(*mbuf*, *len*, *dstoff*)
Similar to **m_pullup**() but copies *len* bytes of data into a new mbuf at *dstoff* bytes into the mbuf. The *dstoff* argument aligns the data and leaves room for a link layer header. Returns the new *mbuf chain* on success, and frees the *mbuf chain* and returns NULL on failure. **Note**: The function does not allocate *mbuf clusters*, so *len + dstoff* must be less than MHLEN.

m_pullup(*mbuf*, *len*)
Arrange that the first *len* bytes of an *mbuf chain* are contiguous and lay in the data area of *mbuf*, so they are accessible with **mtod**(*mbuf*, *type*). It is important to remember that this may involve reallocating some mbufs and moving data so all pointers referencing data within the old mbuf chain must be recalculated or made invalid. Return the new *mbuf chain* on success, NULL on failure (the *mbuf chain* is freed in this case). **Note**: It does not allocate any *mbuf clusters*, so *len* must be less than or equal to MHLEN.

m_pulldown(*mbuf*, *offset*, *len*, *offsetp*)
Arrange that *len* bytes between *offset* and *offset + len* in the *mbuf chain* are contiguous and lay in the data area of *mbuf*, so they are accessible with **mtod**() or **mtodo**(). *len* must be smaller than, or equal to, the size of an *mbuf cluster*. Return a pointer to an intermediate *mbuf* in the chain containing the requested region; the offset in the data region of the *mbuf chain* to the data contained in the returned mbuf is stored in **offsetp*. If *offsetp* is NULL, the region may be accessed using **mtod**(*mbuf*, *type*) or **mtodo**(*mbuf*, *0*). If *offsetp* is non-NULL, the region may be accessed using **mtodo**(*mbuf*, **offsetp*). The region of the mbuf chain between its beginning and *offset* is not modified, therefore it is safe to hold pointers to data within this region before calling **m_pulldown**().

m_copym(*mbuf*, *offset*, *len*, *how*)
Make a copy of an *mbuf chain* starting *offset* bytes from the beginning, continuing for *len* bytes. If *len* is M_COPYALL, copy to the end of the *mbuf chain*. **Note**: The copy is read-only, because the *mbuf clusters* are not copied, only their reference counts are incremented.

m_copypacket(*mbuf*, *how*)
Copy an entire packet including header, which must be present. This is an optimized version of the common case **m_copym**(*mbuf*, *0*, *M_COPYALL*, *how*). **Note**: the copy is read-only, because the *mbuf clusters* are not copied, only their reference counts are incremented.

m_dup(*mbuf*, *how*)
Copy a packet header *mbuf chain* into a completely new *mbuf chain*, including copying any *mbuf*

clusters. Use this instead of **m_copypacket**() when you need a writable copy of an *mbuf chain*.

m_copydata(*mbuf*, *offset*, *len*, *buf*)
Copy data from an *mbuf chain* starting *off* bytes from the beginning, continuing for *len* bytes, into the indicated buffer *buf*.

m_copyback(*mbuf*, *offset*, *len*, *buf*)
Copy *len* bytes from the buffer *buf* back into the indicated *mbuf chain*, starting at *offset* bytes from the beginning of the *mbuf chain*, extending the *mbuf chain* if necessary. **Note**: It does not allocate any *mbuf clusters*, just adds *mbufs* to the *mbuf chain*. It is safe to set *offset* beyond the current *mbuf chain* end: zeroed *mbufs* will be allocated to fill the space.

m_length(*mbuf*, *last*)
Return the length of the *mbuf chain*, and optionally a pointer to the last *mbuf*.

m_dup_pkthdr(*to*, *from*, *how*)
Upon the function's completion, the *mbuf to* will contain an identical copy of *from->m_pkthdr* and the per-packet attributes found in the *mbuf chain from*. The *mbuf from* must have the flag M_PKTHDR initially set, and *to* must be empty on entry.

m_move_pkthdr(*to*, *from*)
Move *m_pkthdr* and the per-packet attributes from the *mbuf chain from* to the *mbuf to*. The *mbuf from* must have the flag M_PKTHDR initially set, and *to* must be empty on entry. Upon the function's completion, *from* will have the flag M_PKTHDR and the per-packet attributes cleared.

m_fixhdr(*mbuf*)
Set the packet-header length to the length of the *mbuf chain*.

m_devget(*buf*, *len*, *offset*, *ifp*, *copy*)
Copy data from a device local memory pointed to by *buf* to an *mbuf chain*. The copy is done using a specified copy routine *copy*, or **bcopy**() if *copy* is NULL.

m_cat(*m*, *n*)
Concatenate *n* to *m*. Both *mbuf chains* must be of the same type. *n* is not guaranteed to be valid after **m_cat**() returns. **m_cat**() does not update any packet header fields or free mbuf tags.

m_catpkt(*m*, *n*)
A variant of **m_cat**() that operates on packets. Both *m* and *n* must contain packet headers. *n* is not guaranteed to be valid after **m_catpkt**() returns.

m_split(*mbuf*, *len*, *how*)
Partition an *mbuf chain* in two pieces, returning the tail: all but the first *len* bytes. In case of failure, it returns NULL and attempts to restore the *mbuf chain* to its original state.

m_apply(*mbuf*, *off*, *len*, *f*, *arg*)
Apply a function to an *mbuf chain*, at offset *off*, for length *len* bytes. Typically used to avoid calls to **m_pullup**() which would otherwise be unnecessary or undesirable. *arg* is a convenience argument which is passed to the callback function *f*.

Each time **f**() is called, it will be passed *arg*, a pointer to the *data* in the current mbuf, and the length *len* of the data in this mbuf to which the function should be applied.

The function should return zero to indicate success; otherwise, if an error is indicated, then **m_apply**() will return the error and stop iterating through the *mbuf chain*.

m_getptr(*mbuf*, *loc*, *off*)
Return a pointer to the mbuf containing the data located at *loc* bytes from the beginning of the *mbuf chain*. The corresponding offset into the mbuf will be stored in **off*.

m_defrag(*m0*, *how*)
Defragment an mbuf chain, returning the shortest possible chain of mbufs and clusters. If allocation fails and this can not be completed, NULL will be returned and the original chain will be unchanged. Upon success, the original chain will be freed and the new chain will be returned. *how* should be either M_WAITOK or M_NOWAIT, depending on the caller’s preference.

This function is especially useful in network drivers, where certain long mbuf chains must be shortened before being added to TX descriptor lists.

m_collapse(*m0*, *how*, *maxfrags*)
Defragment an mbuf chain, returning a chain of at most *maxfrags* mbufs and clusters. If allocation fails or the chain cannot be collapsed as requested, NULL will be returned, with the original chain possibly modified. As with **m_defrag**(), *how* should be one of M_WAITOK or M_NOWAIT.

m_unshare(*m0*, *how*)
Create a version of the specified mbuf chain whose contents can be safely modified without affecting other users. If allocation fails and this operation can not be completed, NULL will be returned. The original mbuf chain is always reclaimed and the reference count of any shared mbuf clusters is decremented. *how* should be either M_WAITOK or M_NOWAIT, depending on the caller’s preference. As a side-effect of this process the returned mbuf chain may be compacted.

This function is especially useful in the transmit path of network code, when data must be encrypted or otherwise altered prior to transmission.

HARDWARE-ASSISTED CHECKSUM CALCULATION

This section currently applies to TCP/IP only. In order to save the host CPU resources, computing checksums is offloaded to the network interface hardware if possible. The *m_pkthdr* member of the leading *mbuf* of a packet contains two fields used for that purpose, *int csum_flags* and *int csum_data*. The meaning of those fields depends on the direction a packet flows in, and on whether the packet is fragmented. Henceforth, *csum_flags* or *csum_data* of a packet will denote the corresponding field of the *m_pkthdr* member of the leading *mbuf* in the *mbuf chain* containing the packet.

On output, checksum offloading is attempted after the outgoing interface has been determined for a packet. The interface-specific field *ifnet.if_data.ifi_hwassist* (see ifnet(9)) is consulted for the capabilities of the interface to assist in computing checksums. The *csum_flags* field of the packet header is set to indicate which actions the interface is supposed to perform on it. The actions unsupported by the network interface are done in the software prior to passing the packet down to the interface driver; such actions will never be requested through *csum_flags*.

The flags demanding a particular action from an interface are as follows:

CSUM_IP	The IP header checksum is to be computed and stored in the corresponding field of the packet. The hardware is expected to know the format of an IP header to determine the offset of the IP checksum field.
CSUM_TCP	The TCP checksum is to be computed. (See below.)
CSUM_UDP	The UDP checksum is to be computed. (See below.)

Should a TCP or UDP checksum be offloaded to the hardware, the field *csum_data* will contain the byte offset of the checksum field relative to the end of the IP header. In this case, the checksum field will be initially set by the TCP/IP module to the checksum of the pseudo header defined by the TCP and UDP specifications.

On input, an interface indicates the actions it has performed on a packet by setting one or more of the following flags in *csum_flags* associated with the packet:

CSUM_IP_CHECKED	The IP header checksum has been computed.
CSUM_IP_VALID	The IP header has a valid checksum. This flag can appear only in

combination with CSUM_IP_CHECKED.

CSUM_DATA_VALID
The checksum of the data portion of the IP packet has been computed and stored in the field *csum_data* in network byte order.

CSUM_PSEUDO_HDR
Can be set only along with CSUM_DATA_VALID to indicate that the IP data checksum found in *csum_data* allows for the pseudo header defined by the TCP and UDP specifications. Otherwise the checksum of the pseudo header must be calculated by the host CPU and added to *csum_data* to obtain the final checksum to be used for TCP or UDP validation purposes.

If a particular network interface just indicates success or failure of TCP or UDP checksum validation without returning the exact value of the checksum to the host CPU, its driver can mark CSUM_DATA_VALID and CSUM_PSEUDO_HDR in *csum_flags*, and set *csum_data* to 0xFFFF hexadecimal to indicate a valid checksum. It is a peculiarity of the algorithm used that the Internet checksum calculated over any valid packet will be 0xFFFF as long as the original checksum field is included.

STRESS TESTING

When running a kernel compiled with the option MBUF_STRESS_TEST, the following sysctl(8)-controlled options may be used to create various failure/extreme cases for testing of network drivers and other parts of the kernel that rely on *mbufs*.

net.inet.ip.mbuf_frag_size
Causes **ip_output**() to fragment outgoing *mbuf chains* into fragments of the specified size. Setting this variable to 1 is an excellent way to test the long *mbuf chain* handling ability of network drivers.

kern.ipc.m_defragrandomfailures
Causes the function **m_defrag**() to randomly fail, returning NULL. Any piece of code which uses **m_defrag**() should be tested with this feature.

RETURN VALUES

See above.

SEE ALSO

ifnet(9), mbuf_tags(9)

S. J. Leffler, W. N. Joy, R. S. Fabry, and M. J. Karels, "Networking Implementation Notes", *4.4BSD System Manager’s Manual (SMM)*.

HISTORY

Mbufs appeared in an early version of BSD. Besides being used for network packets, they were used to store various dynamic structures, such as routing table entries, interface addresses, protocol control blocks, etc. In more recent FreeBSD use of *mbufs* is almost entirely limited to packet storage, with uma(9) zones being used directly to store other network-related memory.

Historically, the *mbuf* allocator has been a special-purpose memory allocator able to run in interrupt contexts and allocating from a special kernel address space map. As of FreeBSD 5.3, the *mbuf* allocator is a wrapper around uma(9), allowing caching of *mbufs*, clusters, and *mbuf* + cluster pairs in per-CPU caches, as well as bringing other benefits of slab allocation.

AUTHORS

The original **mbuf** manual page was written by Yar Tikhiy. The uma(9) *mbuf* allocator was written by Bosko Milekic.