NAME

allocb – allocate a message block

SYNOPSIS

#include <sys/stream.h>

mblk_t *allocb(int *size*, **uint** *pri***);**

INTERFACE LEVEL

Architecture independent level 1 (DDI/DKI).

ARGUMENTS

size The number of bytes in the message block.

pri Priority of the request (no longer used).

DESCRIPTION

allocb( ) tries to allocate a STREAMS message block. Buffer allocation fails only when the system is out of memory. If no buffer is available, the **bufcall**(9F) function can help a module recover from an allocation failure.

A STREAMS message block is composed of three structures. The first structure is a message block (**mblk_t**). See **msgb**(9S). The **mblk_t** structure points to a data block structure (**dblk_t**). See **datab**(9S). Together these two structures describe the message type (if applicable) and the size and location of the third structure, the data buffer. The data buffer contains the data for this message block.

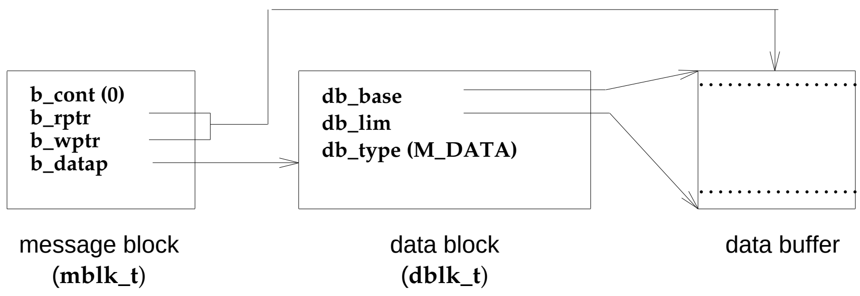
The fields in the **mblk_t** structure are initialized as follows:

b_cont	set to **NULL**
b_rptr	points to the beginning of the data buffer
b_wptr	points to the beginning of the data buffer
b_datap	points to the **dblk_t** structure

The fields in the **dblk_t** structure are initialized as follows:

db_base	points to the first byte of the data buffer
db_lim	points to the last byte + 1 of the buffer
db_type	set to **M_DATA**

The following figure identifies the data structure members that are affected when a message block is allocated.



RETURN VALUES

A pointer to the allocated message block of type **M_DATA** on success.

A **NULL** pointer on failure.

CONTEXT

allocb( ) can be called from user or interrupt context.

EXAMPLES

Given a pointer to a queue (*q*) and an error number (*err*), the **send_error( )** routine sends an **M_ERROR** type message to the stream head.

If a message cannot be allocated, **NULL** is returned, indicating an allocation failure (line 8). Otherwise, the message type is set to **M_ERROR** (line 10). Line 11 increments the write pointer (**bp->b_wptr**) by the size (one byte) of the data in the message.

A message must be sent up the read side of the stream to arrive at the stream head. To determine whether *q* points to a read queue or to a write queue, the **q->q_flag** member is tested to see if **QREADR** is set (line 13). If it is not set, *q* points to a write queue, and in line 14 the **RD**(9F) function is used to find the corresponding read queue. In line 15, the **putnext**(9F) function is used to send the message upstream, returning **1** if successful.

```
 1  send_error(q,err)
 2          queue_t *q;
 3          unsigned char err;
 4  {
 5          mblk_t *bp;
 6
 7          if ((bp = allocb(1, BPRI_HI)) == NULL)      /* allocate msg. block */
 8                  return(0);
 9
10          bp->b_datap->db_type = M_ERROR;             /* set msg type to M_ERROR */
11          *bp->b_wptr++ = err;                        /* increment write pointer */
12
13          if (!(q->q_flag & QREADR))                  /* if not read queue     */
14                  q = RD(q);                          /*    get read queue     */
15          putnext(q,bp);                              /* send message upstream */
16          return(1);
17  }
```

SEE ALSO

bufcall(9F), **esballoc**(9F), **esbbcall**(9F), **putnext**(9F), **RD**(9F), **testb**(9F), **datab**(9S), **msgb**(9S)

Writing Device Drivers
STREAMS Programming Guide

NOTES

The *pri* argument is no longer used, but is retained for compatibility with existing drivers.