**NAME**

dupb – duplicate a message block descriptor

**SYNOPSIS**

**#include <sys/stream.h>**

**mblk_t *dupb(mblk_t ***bp***);**

**INTERFACE LEVEL**

Architecture independent level 1 (DDI/DKI).

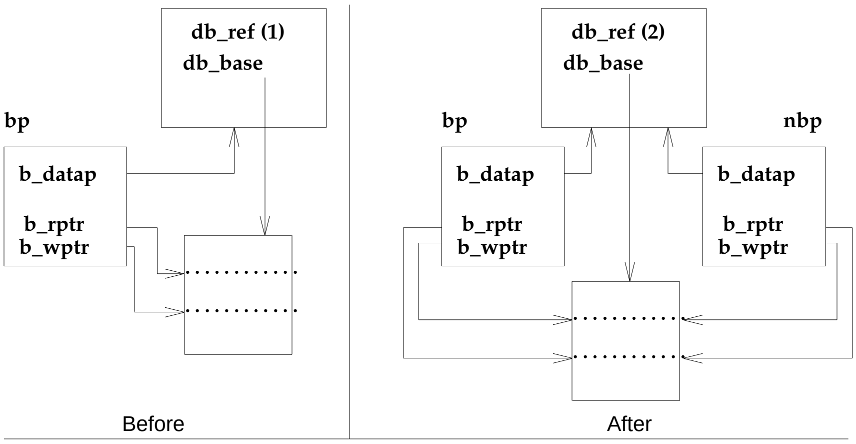
**ARGUMENTS**

*bp* Pointer to the message block to be duplicated. **mblk_t** is an instance of the **msgb**(9S) structure.

**DESCRIPTION**

**dupb( )** creates a new **mblk_t** structure (see **msgb**(9S)) to reference the message block pointed to by *bp*.

Unlike **copyb**(9F), **dupb**( ) does not copy the information in the **dblk_t** structure (see **datab**(9S)), but creates a new **mblk_t** structure to point to it. The reference count in the **dblk_t** structure (**db_ref**) is incremented. The new **mblk_t** structure contains the same information as the original. Note that **b_rptr** and **b_wptr** are copied from the *bp*.




**nbp=dupb(bp);**

**RETURN VALUES**

If successful, **dupb( )** returns a pointer to the new message block. A **NULL** pointer is returned if **dupb( )** cannot allocate a new message block descriptor or if the **db_ref** field of the data block structure (see **datab**(9S)) has reached a maximum value (**255**).

**CONTEXT**

**dupb( )** can be called from user, kernel, or interrupt context.

**EXAMPLES**

This **srv**(9E) (service) routine adds a header to all **M_DATA** messages before passing them along. **dupb** is used instead of **copyb**(9F) because the contents of the header block are not changed.

For each message on the queue, if it is a priority message, pass it along immediately (lines 10–11). Otherwise, if it is anything other than an **M_DATA** message (line 12), and if it can be sent along (line 13), then do so (line 14). Otherwise, put the message back on the queue and return (lines 16–17). For all **M_DATA** messages, first check to see if the stream is flow-controlled (line 20). If it is, put the message back on the queue and return (lines 37–38). If it is not, the header block is duplicated (line 21).

**dupb( )** can fail either due to lack of resources or because the message block has already been duplicated 255 times. In order to handle the latter case, the example calls **copyb**(9F) (line 22). If **copyb**(9F) fails, it is due to buffer allocation failure. In this case, **qbufcall**(9F) is used to initiate a callback (lines 30-31) if one is not already pending (lines 26-27).

The callback function, **xxxcallback( )**, clears the recorded **qbufcall**(9F) callback id and schedules the service procedure (lines 49-50). Note that the close routine, **xxxclose( )**, must cancel any outstanding **qbufcall**(9F) callback requests (lines 58-59).

If **dupb( )** or **copyb**(9F) succeed, link the **M_DATA** message to the new message block (line 34) and pass it along (line 35).

```
 1  xxxsrv(q)
 2      queue_t *q;
 3  {
 4          struct xx *xx = (struct xx *)q->q_ptr;
 5          mblk_t *mp;
 6          mblk_t *bp;
 7          extern mblk_t *hdr;
 8
 9          while ((mp = getq(q)) != NULL) {
10                  if (mp->b_datap->db_type >= QPCTL) {
11                          putnext(q, mp);
12                  } else if (mp->b_datap->db_type != M_DATA) {
13                          if (canputnext(q))
14                                  putnext(q, mp);
15                          else {
16                                  putbq(q, mp);
17                                  return;
18                          }
19                  } else {  /* M_DATA */
20                          if (canputnext(q)) {
21                                  if ((bp = dupb(hdr)) == NULL)
22                                          bp = copyb(hdr);
23                                  if (bp == NULL) {
24                                          uint size = msgdsize(mp);
25                                          putbq(q, mp);
26                                          if (xx->xx_qbufcall_id) {
27                                                  /* qbufcall pending */
28                                                  return;
29                                          }
```

```
                                        xx->xx_qbufcall_id = qbufcall(q, size,
                                                BPRI_MED, xxxcallback, (long)q);
                                        return;
                                }
                                linkb(bp, mp);
                                putnext(q, bp);
                        } else {
                                putbq(q, mp);
                                return;
                        }
                }
        }
}

        void
        xxxcallback(q)
                queue_t *q;
        {
                struct xx *xx = (struct xx *)q->q_ptr;

                xx->xx_qbufcall_id = 0;
                qenable(q);
        }

        xxxclose(q, cflag, crp)
                queue_t *q;
                int  cflag;
                cred_t *crp;
        {
                struct xx *xx = (struct xx *)q->q_ptr;
                . . .
                if (xx->xx_qbufcall_id)
                        qunbufcall(q, xx->xx_qbufcall_id);
                . . .
        }
```

**SEE ALSO**

**copyb**(9F), **qbufcall**(9F), **datab**(9S), **msgb**(9S)

*Writing Device Drivers*
*STREAMS Programming Guide*