NAME

bufcall – call a function when a buffer becomes available

SYNOPSIS

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

int bufcall(uint *size,* **int** *pri,* **void (****func***)(***long***), long** *arg***);**

INTERFACE LEVEL

Architecture independent level 1 (DDI/DKI).

ARGUMENTS

size Number of bytes required for the buffer.

pri Priority of the **allocb**(9F) allocation request (not used).

func Function or driver routine to be called when a buffer becomes available.

arg Argument to the function to be called when a buffer becomes available.

DESCRIPTION

bufcall( ) serves as a **timeout**(9F) call of indeterminate length. When a buffer allocation request fails, **bufcall( )** can be used to schedule the routine *func*, to be called with the argument *arg* when a buffer becomes available. *func* may call **allocb( )** or it may do something else.

RETURN VALUES

If successful, **bufcall( )** returns a **bufcall** id that can be used in a call to **unbufcall( )** to cancel the request. If the **bufcall( )** scheduling fails, *func* is never called and **0** is returned.

CONTEXT

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

EXAMPLES

The purpose of this **srv**(9E) service routine is to add a header to all **M_DATA** messages. Service routines must process all messages on their queues before returning, or arrange to be rescheduled.

While there are messages to be processed (line 13), check to see if it is a high priority message or a normal priority message that can be sent on (line 14). Normal priority message that cannot be sent are put back on the message queue (line 34). If the message was a high priority one, or if it was normal priority and **canputnext**(9F) succeeded, then send all but **M_DATA** messages to the next module with **putnext**(9F) (line 16).

For **M_DATA** messages, try to allocate a buffer large enough to hold the header (line 18). If no such buffer is available, the service routine must be rescheduled for a time when a buffer is available. The original message is put back on the queue (line 20) and **bufcall** (line 21) is used to attempt the rescheduling. It will succeed if the rescheduling succeeds, indicating that qenable will be called subsequently with the argument *q* once a buffer of the specified size (**sizeof (struct hdr)**) becomes available. If it does, **qenable**(9F) will put *q* on the list of queues to have their service routines called. If **bufcall( )** fails, **timeout**(9F) (line 22) is used to try again in about a half second.

If the buffer allocation was successful, initialize the header (lines 25–28), make the message type **M_PROTO** (line 29), link the **M_DATA** message to it (line 30), and pass it on (line 31).

Note that this example ignores the bookkeeping needed to handle **bufcall( )** and **timeout**(9F) cancellation for ones that are still outstanding at close time.

```
struct hdr {
    unsigned int h_size;
    int        h_version;
};

void xxxsrv(q)
    queue_t *q;
{
    mblk_t *bp;
    mblk_t *mp;
    struct hdr *hp;

    while ((mp = getq(q)) != NULL) {               /* get next message */
        if (mp->b_datap->db_type >= QPCTL ||       /* if high priority */
            canputnext(q)) {                       /* normal & can be passed */
            if (mp->b_datap->db_type != M_DATA)
                putnext(q, mp);                    /* send all but M_DATA */
            else {
                bp = allocb(sizeof(struct hdr), BPRI_LO);
                if (bp == NULL) {                  /* if unsuccessful */
                    putbq(q, mp);                  /* put it back */
                    if (!bufcall(sizeof(struct hdr), BPRI_LO,
                      qenable, (long)q))           /* try to reschedule */
                        timeout(qenable, (caddr_t)q, drv_usectohz(500000));
                    return (0);
                }
                hp = (struct hdr *)bp->b_wptr;
                hp->h_size = msgdsize(mp);         /* initialize header */
                hp->h_version = 1;
                bp->b_wptr += sizeof(struct hdr);
                bp->b_datap->db_type = M_PROTO;  /* make M_PROTO  */
                bp->b_cont = mp;                   /* link it */
                putnext(q, bp);                    /* pass it on */
            }
        } else {                              /* normal priority, canputnext failed */
            putbq(q, mp);                     /* put back on the message queue */
            return (0);
        }
    }
                                          return (0);
}
```

SEE ALSO

allocb(9F), **esballoc**(9F), **esbbcall**(9F), **testb**(9F), **timeout**(9F), **unbufcall**(9F)

Writing Device Drivers
STREAMS Programming Guide

WARNINGS

Even when *func* is called by **bufcall( ), allocb**(9F) can fail if another module or driver had allocated the memory before *func* was able to call **allocb**(9F).