NAME

msg, msgenab, msgdisab, send, sendw, recv, recvw, msgstat, msgctl − old message veneer for sending and receiving messages.

SYNOPSIS

```
# include <sys/ipcomm.h>
msgenab ( )
msgdisab ( )

send (buf, size, topid, type)
sendw (buf, size, topid, type)
char *buf;

recv (buf, size, &mstruct, type)
recvw (buf, size, &mstruct, type)
char *buf;
struct mstruct mstruct;

msgstat (&mstat, sizeof (mstat), pid)
struct mstat mstat;

msgctl (pid, command, arg)
```

DESCRIPTION

The routines described here implement the old message interface. They are now implemented as a veneer using the new message implementation. (See *message*(2)).

A process that has enabled message reception has a message queue on which are placed messages sent to it by other processes. Messages are placed on the queue in the order of arrival. The process actually receives a message by requesting a one from the queue. A process may send a message to any other process that has enabled message reception, as long as the receiver does not have an excessive number of messages pending on its queue.

msgenab ()
Enable message reception by creating a message queue with the name equal to the *process id*.

msgdisab ()
Disable message reception. Any messages still on the queue are destroyed or returned to sender depending upon the *type*.

send (buf, size, topid, type)
Send the message in *buf* and of *size* bytes to the process whose process id is *topid*. The message is stored with the specified *type*, which ranges from *1* to *128*. If the queue for the receiving process is full or if there is no more message space in the operating system, return immediately to the sending process with an appropriate error. When *send* is successful, it returns the number of bytes actually sent in the message.

sendw (buf, size, topid, type)
Send a message in the same fashion as *send*, but if there isn't room for the message, suspend execution until the message can be sent. When *sendw* is successful, it returns the number of bytes actually sent in the message.

recv (buf, size, &mstruct, type)
Receive the first message on the queue if *type* is 0, otherwise receive the first message on the queue whose type matches *type*. Store the message in *buf*, truncating it if the message is larger than *size*. *mstruct* will be filled with the queue name that the sending process has enabled and the actual type of the message. If the sending process did not have messages enabled the queue name in the *mstruct* structure will be 0. If there is no message of the specified type, return immediately with an appropriate error message. Upon a successful

recv, the size of the message received is returned.

recvw (buf, size, &mstruct, type)
Receive a message in the same fashion as *recv*, but if there isn't a message satisfying the requested *type*, suspend execution until one arrives. Upon a successful *recvw*, the size of the message received is returned.

msgstat (&mstat, sizeof(mstat), pid)
Retreive the number of messages currently present on the queue *pid* and the maximum allowable number of messages for this queue. Put the results in the structure *mstat*.

msgctl (pid, command, arg)
Perform the specified command on queue *pid*. The only command currently available is **SETMQLEN**, which allows the maximum number of messages that may queue up for a specific process to be adjusted to *arg*.

The number of bytes actually sent or received is returned by *send*, *sendw*, *recv*, and *recvw*.

The *type* argument is used by a sender to assign a type number (1 to 128) to a message. By convention, types 1 to 63 imply that an acknowledgement message is desired; types 64 to 128 imply no acknowledgement is necessary; type 128 is an acknowledgement message. If a process disables messages (or exits) with any messages still on its queue, those of type 1 to 63 are changed to type 128 and, if possible, returned to the sender; those of type 64 to 128 are discarded.

ipcomm.h is included here for convenience.

```
/*              @(#)ipcomm.h  3.3               */

/*
 * Interprocess Communication Control Structures
 */

#ifdef KERNEL
/*
 * common flags
 */

#define IP_PERM         03                              /* scope permission mask */
#define IP_ANY 0                                /* system scope */
#define IP_UID  01                              /* userid scope */
#define IP_GID  02                              /* groupid scope */
#define IP_QWANT        0100                    /* entry in msg queue wanted */
#define IP_WANTED       0200                    /* resource is desired */

struct ipaword
{               char            ip_flag;
                char            ip_id; };

/*
 * message control
 */

#define PMSG    5                               /* message sleep priority */
#define MSGIO  02                               /* tell iomove() this is msg */
#define MSGIN  0                                /* same as B_WRITE */
#define MSGOUT          01                              /* same as B_READ */

#define MDISAB          0
#define MENAB 1
#define MSEND 2
```

```
#define MSENDW          3
#define MRECV 4
#define MRECVW          5
#define MSTAT 6
#define MSGCTL          7

struct msghdr
{       struct msghdr   *mq_forw;
        int             mq_size;
        int             mq_sender;
        int             mq_type;
};
struct msgqhdr
{       struct msghdr   *mq_forw;       /* note same position as in msghdr */
        struct msghdr   *mq_last;
        int             *mq_procp;
        char            mq_flag;
        char            mq_cnt;
        int             mq_meslim;
};

#endif

/* commands for msgctl call here */
#define SETMQLEN 0                                      /*set mes q length command*/


struct mstat {
        unsigned        ms_cnt;
        unsigned        ms_maxm;
};

struct mstruct {
        int             ms_frompid;
        int             ms_type;
};
```

DIAGNOSTICS

An error occurs when enabling messages if no queue is available for use; it is also erroneous to attempt to disable message reception if it is not enabled. When trying to send messages, errors occur because the message is too long, the receiver has not enabled message reception, the type specified is not valid, the receiver has an excessive number of messages outstanding on its queue, or, for *send*, the system message buffers are temporarily full. When receiving messages, errors may occur because the process has not enabled message reception, the requested type or size are invalid, or, for *recv*, a message of the requested type is not on the queue. It is also illegal to set the message limit (via *msgctl*) to a value larger than defined by MAXMSGDEF in **param.h**.

FILES

/usr/include/sys/ipcomm.h

BUGS

There is one noticable difference between this veneer and the real old messages. The process id of the sender was always given to the message receiving process even if the sender didn't have messages enabled. Now, if the sender doesn't have messages enabled, the receiver gets a 0.

SEE ALSO

message(2)