NAME

menab, mdisab, msend, mrecv, mctl − send and receive messages

SYNOPSIS

```
#include <sys/msg.h>

menab (name, flags)
short name;
short flags;

mdisab (disp)
short disp;

msend (&mstr, buf, size)
mrecv (&mstr, buf, size)
struct mstr mstr;
caddr_t buf;
short size;

mctl (&mstr, command, arg, size)
struct mstr mstr;
short command;
caddr_t arg;
short size;
```

DESCRIPTION

Messages are a very fast form of interprocess communication. Messages are stored on named queues. A process may send a message to any queue for which it has permission. A process can attach to one and only one queue at a time to receive messages according to the permissions associated with the queue. (There may, however, be synonyms for the same queue, see below.)

menab(name,flags)

Enable message reception via the queue *name*. If the queue does not already exist, create it, giving it the characteristics specified by *flags*. If the queue already exists, attempt to attach the existing queue. Attaching an existing queue will succeed only if the following conditions are met:

1) The *flags* argument does match the permissions for the queue (see <sys/msg.h>.)

2) The **MXCLUDE** bit is not set for the queue. (This bit is always cleared by the system when the last process disconnects from a queue, hence it is always possible for a process with the proper permissions to attach a queue if no one else is attached.)

3) The **MOTHR** and **MGRPR** permissions in combination with the queue's and process' user and group ids allow the attempt. These permissions are interpreted in the same way as the normal UNIX file permissions: see *access*(2).

The *flags* are as follows:

MNODESTROY Do not destroy the queue when the last process detaches. This is the default action. When either **MNODESTROY** or **MDESTROY** is specified by *menab*() it is used if the process dies or exits without specifically detaching the queue with a *mdisab()*.

MDESTROY Destroy the queue when the last process detaches. All messages remaining on the queue at the time of destruction, which require acknowledgement (the **MACKREQ** flag was set when they were sent), are returned to the sending process if possible, with a type of **MACKTYP**.

MXCLUDE Do not allow any other process to attach to this queue. This remains in force as long as the current process is attached.

MPRIQ Queue messages in order of priority based on *ms_type*. Normally messages are queued in order of arrival, first-in, first-out (FIFO). In a priority queue, messages with larger *ms_stype*'s are stored before messages with lower *ms_stype*'s. (See *mrecv* below)

MGRPR Allow any process with the same group id as the group id of the creating process to read the queue, i.e. attach the queue for receiving.

MGRPW Allow any process with the same group id as the group id of the creating process to write the queue, i.e. send messages to the queue.

MOTHR Allow any process whose user id and group id are different from the creating process' ids to read the queue, i.e. attach the queue for receiving.

MOTHW Allow any process whose user id and group id are different from the creating process' ids to write the queue, i.e. send messages to the queue.

Upon a successfully attaching to a queue, *menab*() returns the number of processes attached to the queue.

mdisab(disp)

Disable message reception and detach the queue. *disp* contains either the **MNODESTROY** or the **MDESTROY** flag, stating what the disposition of the queue is to be if this is the last process releasing the queue. This overrides the disposition specified during the *menab()*.

msend(mstr,buf,size)

Send a message contained in *buf*, which is of *size* bytes to the queue specified by the *mstr* structure. *mstr* should contain the queue name and the system name to which the message is to be sent (in *ms_qname* and *ms_system*). It should also contain the message subtype in *ms_stype* and the message type and flags, specified in *ms_flags*. Message subtypes can take any value from *1* to *127*.

The flags and types are as follows:

MNOBLOCK Do not wait if the message cannot be sent (or received for *mrecv*) immediately, but return with an appropriate error message.

MNOCOPY Do not copy the message out of the user space. Instead adjust the memory mapping so that it is no longer apart of the user's address space. For this feature to work the system must have the feature enabled and the message itself must be in a section of shared memory. Initially shared memory for messages may be gotten using *smget* (see *shmem*(2)). During an *msend*(), if the address of the buffer supplied is not shared memory and the **MNOCOPY** flag is set, then the *msend*() will fail. Messages sent *without* the **MNOCOPY** flag cannot be larger than **MAXMLEN**. Messages sent as **MNOCOPY** are limited only by the amount of shared memory that can be in existence at one time, a system definable parameter. When a process receives a **MNOCOPY** message, the shared memory message space is mapped into the address space of the receiver and *ms_addr* is set to point to the beginning of this shared memory segment. The **MNOCOPY** flag will be on in *ms_flags*. Messages received with the **MNOCOPY** flag set may be sent to other

processes with it set or the shared memory space may be returned to the operating system using *smfree* (see *shmem*(2)). If a process tries to receive a **MNOCOPY** message and it cannot be mapped into the user's address space, as much as possible is copied into the user supplied buffer and the **MNOCOPY** flag is turned off.

MACKREQ An acknowledgement is required for this message. If a message with this type is still on a queue when it is destroyed, the operating system will change its type to **MACKTYP** and attempt to return it to the sender.

MDATATYP Declares that this message is a data type message. This type has no meaning to the operating system and is supplied to be used by users.

MCTLTYP Declares that this message is a control type message. This type has no meaning to the operating system and is supplied to be used by users.

MINTRTYP Declares that this message is an interrupt type message. This type has no meaning to the operating system and is supplied to be used by users.

MACKTYP Declares that this message is an acknowledgement. The operating system will not allow a message to be sent which has **MACKREQ** set and is of type **MACKTYP**. The operating system will change the type of any message being returned to sender to **MACKTYP**. (See **MACKREQ** above.)

Upon successfully sending a message, *msend*() returns the number of bytes of message actually sent.

mrecv(mstr,buf,size)

Receive a message. Normally the message will be placed in *buf*, and truncated to *size* bytes if the message is bigger than the buffer. Messages received with the **MNOCOPY** flag on will not use *buf*. *mstr* should initially contain the subtype (*ms_stype*) and optionally the **MNOBLOCK** flag, if waiting is not desired. The remainder of *mstr* will be filled in by the operating system dependent upon the message actually being received. *ms_qname* and *ms_system* will contain the name of the queue to which the sending process is attached. If the message sender does not have messages enabled, then *ms_qname* will be 0. *ms_rqname* will contain the name of the queue that the message was actually sent to. (See **MAPQ** below.) The subtype and the type of the queue (FIFO or priority) determine which message will be received.

FIFO

ms_type = 0

Return next message of any subtype. The subtype of the message actually received will be placed by the operating system into *mstr*.

ms_type = 1−127

Return only a message of this specific type. If the message queue is full and there isn't a message of the specific type on the queue and someone attempts to send a message of the desired type, the message will be sent and the receiver will wake up. This will not work if there are multiple receivers sleeping on different non-zero types. In this case one of the processes may never wakeup. Receiving a specific message type from a FIFO message queue should be used very carefully.

Priority

ms_type = 0−127
Return the first message whose subtype is greater than or equal to *ms_stype* in the receiver's *mstr*.

mctl(mstr,command,arg,size)
Fetch and change various parameters for queues. The commands are:

GETMSTAT Returns an *mstats* structure containing the number of messages presently on the queue, the maximum number allowable, the owner and group of the queue, the number of processes attached to the queue, and the modes and disposition of the queue.

SETMQLEN Sets the maximum number of messages that a queue can contain to *command.ms_smqlen*. This number cannot be greater than **MAXMSGL** (See **<sys/param.h>**). Only processes with the same user id as the queue or which are super−user can change the maximum queue length.

SETREMQ This allows one queue to be declared as the *remote* queue. All messages destined for systems other than the present system are routed to this queue. The process reading the remote message queue is responsible for actually getting the message to the remote system by whatever means it is programmed to use. *ms_system*, *ms_qname*, and *ms_rqname* have special meanings when a remote queue manager receives and sends messages. When receiving messages *ms_qname* contains the name of a local queue attached to the sending process; *ms_system* continues to contain the name of the remote system to which the message is to be sent; and *ms_rqname* contains the name of the remote system queue to which the message is to be sent. When the process attached to the remote message queue sends a message *ms_qname* always specifies a local queue name. The operating system takes the values of *ms_system* and *ms_rqname* and places them into *ms_system* and *ms_qname* of the final message so that the local receiver of the message sees the message as having arrived from that system and remote queue.

SETSPYQ This is a debugging aid. It specifies that a copy of all messages sent to the queue specified by *mstr* be sent to the queue *arg.ms_spyq*. There can only be one spy queue in the system at a time.

MAPQ This command allows the creation and removal of synonym queue names. A message sent to synonym queue name is sent to the real queue, but with *ms_rqname* set to the synonym queue name to which the message was directed. In this way the receiving process will know where the sender thought the message was going. Note that the synonym queue has all the permissions of the original queue and that the synonym will disappear when the original queue is destroyed. It is illegal to create a synonym which is the same as the original and it is also illegal to attach to a synonym queue. To create or remove a synonym queue the process performing the MAPQ function must have read permission for the real queue. To create a synonym, *mstr* specifies a

real queue and arg.ms_synq is the synonym queue name to be associated with the real queue. If *mstr.ms_qname* is 0 and *arg.ms_synq* specifies a current synonym queue name, then the synonym queue name is removed.

Messages reception remains enabled across *exec*, but not across *fork*.

In creating queue names the following convention is recommended. All system wide permenant queue names should be defined in the header file, **/usr/include/msgqueues.h**. All such permenant queue names should be negative numbers (0100000 to 0177777), thereby leaving the positive numbers available to processes which need a temporary queue for acknowledgements or which are using the old message veneer. (See *msg*(3)). Such processes may therefore create temporary queues with names equal to their *pid* and be assured that these names will not collide with permenant queue names since *pids* are never negative.

The format of **<sys/msg.h>** is as follows:

```
/*              @(#)msg.h       3.1             */
/*
 * Message Control Structures
 */

typedef         short           queue_t;

/*
 * Modes for menab and mdisab. (ST.mq_modes)
 * For mdisab only the MDESTROY flag is meaningful.
 */
#define MNODESTROY      0000    /* Retain queue when unreferenced */
#define MDESTROY        0001    /* Destroy queue when unreferenced */
#define MOTHR           0002    /* Other read permission */
#define MOTHW                   0004        /* Other write permission */
#define MXCLUDE                 0010        /* Only one process may attach */
#define MGRPR           0020    /* Group read permission */
#define MGRPW                   0040        /* Group write permission */
#define MPRIQ           0100    /* Priority type queue */

#define MDEFAULT        (MNODESTROY|MOTHR|MOTHW|MXCLUDE|MGRPR|MGRPW)

/*
 * commands for mctl call
 */
#define GETMSTAT        0       /* get message status */
#define SETMQLEN        1       /* set message queue length */
#define SETREMQ                 2           /* set remote message queue */
#define SETSPYQ                 3           /* set spy parameters */
#define MAPQ            4       /* create/destroy synonym queues */

/*
 * structure of arg for GETMSTAT command of mctl
 */
struct mstats {
                short   mq_cnt;                 /* number in queue */
                short   mq_mslim;       /* maximum queue size */
                short   mq_uid;                 /* "owner" uid */
                short   mq_gid;                 /* "owner" gid */
                char    mq_refc;        /* no. attached to queue */
                char    mq_modes;       /* permissions and disposition */
};

/*
```

```
 * structure of arg for SETMQLEN command
 */
struct setmq {
        short           ms_smqlen;      /* maximum queue length */
};

/*
 * For the SETREMQ command the arg and size arguments to
 * mctl are not used. The queue name specified in the first
 * argument to mctl is the queue which becomes the remote queue.
 * If this queue name is zero, the current remote queue is
 * disconnected.
 */

/*
 * structure of arg for SETSPYQ command
 * The first arg to mctl specifies the queue to be spied upon.
 * This arg specifies the queue to which a copy of the data is
 * to be sent.
 */
struct setspyq {
        queue_t         ms_spyq;
};

/*
 * structure of arg for the MAPQ command
 * The first arg to mctl specifies the existing queue
 * to which the synonym is to be mapped. If it specifies a
 * qname of zero any existing synonym with the name
 * specified in the synq structure is eliminated.
 * To successfully create or remove a queue synonym the
 * user doing the MAPQ command must have read permission
 * for the real queue.
 */
struct synq {
        queue_t         ms_synq;
};

/*
 * structure for sending and receiving messages
 */
struct mstr {
        long            ms_system;      /* system name  */
        queue_t         ms_qname;       /* queue name   */
        char            ms_stype;       /* message sub-type/priority */
        char            ms_flags;
        caddr_t         ms_addr;        /* address for mrecv */
        queue_t         ms_rqname;      /* queue msg was sent to */
        short           ms_uid;                 /* sender's user id */
        short           ms_gid;                 /* sender's group id */
};

/*
 * Flag values for ms_flags
 */
#define MNOBLOCK        001     /* Non-blocking send and recv */
#define MNOCOPY                 002             /* Remap segment-no copy if possible */
#define MACKREQ                 004             /* Ack required */
#define MDATATYP        000     /* Data message */
#define MCTLTYP                 010             /* Control message */
#define MINTRTYP        020     /* Interrupt message */
#define MACKTYP                 030             /* Ack message */
#define MTYPMSK                 030             /* Mask of type bits */
```

```
#ifdef KERNEL

#define MFLGCARE        (MOTHR|MOTHW|MGRPR|MGRPW|MPRIQ)

#define PMSG    PZERO+5                 /* message sleep priority */
#define MSGIN   B_WRITE
#define MSGOUT          B_READ
#define MREAD02
#define MWRITE          04

#define MDISAB          0
#define MENAB 1
#define MSEND 2
#define MRECV 3
#define MSGCTL          4

#define NORMAL_SEND     00000           /* Normal msg - user to user */
#define REM_USR                         00400           /* Remote msg - daemon to user */
#define REM_SEND        01000           /* Remote msg - user to daemon */

/*
 * State bits
 */
#define IP_QWANT        0100            /* msg queue wanted */
#define IP_WANTED       0200            /* resource is desired */

struct msghdr {
        struct msghdr *mq_forw;
        union {
                struct {
                        short           mq_size;
                        queue_t         mq_sender;
                        long            mq_system;
                        paddr_t         mq_addr;
                        queue_t         mq_rqname; /* remote queue name */
                        char            mq_stype;
                        char            mq_flags;
                        short           mq_muid;
                        short           mq_mgid;
                }ms;
                struct {
                        struct msghdr *mq_last;
                        queue_t mq_name;
                        char            mq_twant;       /* Wanted for type */
                        char            mq_state;
                        struct mstats st;
                }qu;
        }UN;
};

/*
 * Shorthand notations for accessing elements of above structure
 */

#define QU      UN.qu
#define MS      UN.ms
#define ST      UN.qu.st

/*
 * Message related measurements
 */
struct M_MEAS {
        short           qinuse;                 /* number of queues in use */
```

```
        short   qtblovr;   /* no. of queue table overflows */
        short   mtblovr;   /* no. of msg table overflows */
        long    msgsent;   /* no. of msgs sent */
        long    msgrecv;   /* no. of msgs received */
        long    msgflsh;   /* no. of msgs flushed */
};
#endif
```

DIAGNOSTICS

A −1 is returned for any one of a number of error conditions. An error occurs when enabling messages if no queue can be allocated or if the process is attempting to connect to a queue that does not have the appropriate permissions; 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 specified message queue or system does not exist, the type or priority specified is not valid, the **MNOCOPY** bit is used incorrectly, or, for conditional sends, the system message buffers are temporarily full or the receiver has an excessive number of messages on its queue. When receiving messages, errors may occur because the process has not enabled message reception, the requested priority is invalid, or, for conditional receives, a message of the requested type is not on the queue. It is also illegal to set the message limit (via *mctl*) to a value larger than defined by **MAXMSGL** or to specify a *mctl* for a queue that the user could not connect to.

FILES

/usr/include/sys/param.h
/usr/include/sys/msg.h

BUGS

It may not be possible to return errors correctly when trying to send messages to remote systems.

SEE ALSO

access(2), shmem(2), msg(3)