

NAME

ftok - generate an IPC key

SYNOPSIS

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

```
key_t ftok(const char *path, int id);
```

DESCRIPTION

The `ftok()` function returns a key based on `_p_a_t_h` and `_i_d` that is usable in subsequent calls to `msgget(2)`, `semget(2)` and `shmget(2)`. The `_p_a_t_h` argument must be the pathname of an existing file that the process is able to `stat(2)`.

The `ftok()` function will return the same key value for all paths that name the same file, when called with the same `_i_d` value, and will return different key values when called with different `_i_d` values.

If the file named by `_p_a_t_h` is removed while still referred to by a key, a call to `ftok()` with the same `_p_a_t_h` and `_i_d` returns an error. If the same file is recreated, then a call to `ftok()` with the same `_p_a_t_h` and `_i_d` is likely to return a different key.

Only the low order 8-bits of `id` are significant. The behavior of `ftok()` is unspecified if these bits are 0.

RETURN VALUES

Upon successful completion, `ftok()` returns a key. Otherwise, `ftok()` returns `(key_t)-1` and sets `errno` to indicate the error.

ERRORS

The `ftok()` function will fail if:

EACCES

Search permission is denied for a component of the path prefix.

ELOOP Too many symbolic links were encountered in resolving `_p_a_t_h`.

ENAMETOOLONG

The length of the `_p_a_t_h` argument exceeds `{PATH_MAX}` or a pathname component is longer than `{NAME_MAX}`.

ENOENT

A component of `_p_a_t_h` does not name an existing file or `_p_a_t_h` is an empty string.

ENOTDIR

A component of the path prefix is not a directory.

The `ftok()` function may fail if:

ENAMETOOLONG

Pathname resolution of a symbolic link produced an intermediate result whose length exceeds `{PATH_MAX}`.

USAGE

For maximum portability, `_i_d` should be a single-byte character.

Another way to compose keys is to include the project ID in the most significant byte and to use the remaining portion as a sequence number. There are many other ways to form keys, but it is necessary for each system to define standards for forming them. If some standard is not adhered to, it will be possible for unrelated processes to unintentionally interfere with each other's operation. It is still possible to interfere intentionally. Therefore, it is strongly suggested that the most significant byte of a key in some sense refer to a project so that keys do not conflict across a given system.

NOTES

Since the `ftok()` function returns a value based on the `_i_d` given and the file serial number of the file named by `_p_a_t_h` in a type that is no longer large enough to hold all file serial numbers, it may return the same key for paths naming different files on large filesystems.

System Calls

`msgget(2)`

NAME

`msgget` - get message queue

SYNOPSIS

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

```
int msgget(key_t key, int msgflg);
```

DESCRIPTION

The `msgget()` argument returns the message queue identifier associated with `_k_e_y`.

A message queue identifier and associated message queue and data structure (see `intro(3)`) are created for `_k_e_y` if one of the following are true:

- + o `_k_e_y` is `IPC_PRIVATE`.
- + o `_k_e_y` does not already have a message queue identifier associated with it, and `(_m_s_g_f_l_g & IPC_CREAT)` is true.

On creation, the data structure associated with the new message queue identifier is initialized as follows:

- + o `msg_perm.cuid`, `msg_perm.uid`, `msg_perm.cgid`, and `msg_perm.gid` are set to the effective user ID and effective group ID, respectively, of the calling process.
- + o The low-order 9 bits of `msg_perm.mode` are set to the low-order 9 bits of `_m_s_g_f_l_g`.
- + o `msg_qnum`, `msg_lspid`, `msg_lrpid`, `msg_stime`, and `msg_rtime` are set to 0.
- + o `msg_ctime` is set to the current time.
- + o `msg_qbytes` is set to the system limit.

RETURN VALUES

Upon successful completion, a non-negative integer representing a message queue identifier is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.

ERRORS

The `msgget()` function will fail if:

EACCES

A message queue identifier exists for `_k_e_y`, but operation permission (see `intro(3)`) as specified by the low-order 9 bits of `_m_s_g_f_l_g` would not be granted.

EEXIST

A message queue identifier exists for `_k_e_y` but `(_m_s_g_f_l_g & IPC_CREAT)` and `(_m_s_g_f_l_g & IPC_EXCL)` are both true.

ENOENT

A message queue identifier does not exist for `_k_e_y` and `(_m_s_g_f_l_g & IPC_CREAT)` is false.

ENOSPC

A message queue identifier is to be created but the system-imposed limit on the maximum number of allowed message queue identifiers system wide would be exceeded.

SEE ALSO

intro(3), msgctl(2), msgrcv(2), msgsnd(2), ftok(3C)

System Calls

msgctl(2)

NAME

msgctl - message control operations

SYNOPSIS

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

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

DESCRIPTION

The msgctl() function provides a variety of message control operations as specified by `_c_m_d`. The following `_c_m_d`s are available:

IPC_STAT

Place the current value of each member of the data structure associated with `_m_s_q_i_d` into the structure pointed to by `_b_u_f`. The contents of this structure are defined in intro(2).

IPC_SET

Set the value of the following members of the data structure associated with `_m_s_q_i_d` to the corresponding value found in the structure pointed to by `_b_u_f`:

```
msg_perm.uid  
msg_perm.gid  
msg_perm.mode /* access permission bits only */  
msg_qbytes
```

This `_c_m_d` can only be executed by a process that has an effective user ID equal to either that of super-user, or to the value of `msg_perm.cuid` or `msg_perm.uid` in the data structure associated with `_m_s_q_i_d`. Only super-user can raise the value of `msg_qbytes`.

IPC_RMID

Remove the message queue identifier specified by `_m_s_q_i_d` from the system and destroy the message queue and data structure associated with it. This `_c_m_d` can only be executed by a process that has an effective user ID equal to either that of super-user, or to the value of `msg_perm.cuid` or `msg_perm.uid` in the data structure associated with `_m_s_q_i_d`. The `_b_u_f` argument is ignored.

RETURN VALUES

Upon successful completion, msgctl() returns 0. Otherwise,

it returns -1 and sets errno to indicate the error.

ERRORS

The msgctl() function will fail if:

EACCESS

The `_c_m_d` argument is `IPC_STAT` and operation permission is denied to the calling process (see intro(2)).

EFAULT

The `_b_u_f` argument points to an illegal address.

EINVAL

The `_m_s_q_i_d` argument is not a valid message queue identifier; or the `_c_m_d` argument is not a valid command or is `IPC_SET` and `msg_perm.uid` or `msg_perm.gid` is not valid.

EPERM The `_c_m_d` argument is `IPC_RMID` or `IPC_SET` and the effective user ID of the calling process is not super-user and is not equal to the value of `msg_perm.cuid` or `msg_perm.uid` in the data structure associated with `_m_s_q_i_d`.

EPERM The `_c_m_d` argument is `IPC_SET`, an attempt is being made to increase to the value of `msg_qbytes`, and the effective user ID of the calling process is not super-user.

EOVERFLOW

The `_c_m_d` argument is `IPC_STAT` and `_u_i_d` or `_g_i_d` is too large to be stored in the structure pointed to by `_b_u_f`.

SEE ALSO

intro(2), msgget(2), msgrcv(2), msgsnd(2)

System Calls `msgsnd(2)`

NAME

msgsnd - message send operation

SYNOPSIS

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

```
int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

DESCRIPTION

The msgsnd() function is used to send a message to the queue associated with the message queue identifier specified by `_m_s_q_i_d`.

The `_m_s_g_p` argument points to a user-defined buffer that must contain first a field of type `long int` that will specify the type of the message, and then a data portion that will hold the data bytes of the message. The structure below is an example of what this user-defined buffer might look like:

```
struct mymsg {
    long mtype; /* message type */
    char mtext[1]; /* message text */
}
```

The `mtype` member is a non-zero positive type `long int` that can be used by the receiving process for message selection.

The `mtext` member is any text of length `_m_s_g_s_z` bytes. The `_m_s_g_s_z` argument can range from 0 to a system-imposed maximum.

The `_m_s_g_f_l_g` argument specifies the action to be taken if one or more of the following are true:

- + o The number of bytes already on the queue is equal to `msg_qbytes`; see `intro(2)`.
- + o The total number of messages on all queues system-wide is equal to the system-imposed limit.

These actions are as follows:

- + o If `(_m_s_g_f_l_g & IPC_NOWAIT)` is non-zero, the message will not be sent and the calling process will return immediately.
- + o If `(_m_s_g_f_l_g & IPC_NOWAIT)` is 0, the calling process will suspend execution until one of the following occurs:
 - + o The condition responsible for the suspension no longer exists, in which case the message is sent.
 - + o The message queue identifier `_m_s_q_i_d` is removed from the system (see `msgctl(2)`); when this occurs, `errno` is set equal to `EIDRM` and `-1` is returned.
 - + o The calling process receives a signal that is to be caught; in this case the message is not sent and the calling process resumes execution in the manner prescribed in `sigaction(2)`.

Upon successful completion, the following actions are taken with respect to the data structure associated with `_m_s_q_i_d` (see `intro(2)`):

- + o `msg_qnum` is incremented by 1.

- + o `msg_lspid` is set equal to the process ID of the calling process.
- + o `msg_stime` is set equal to the current time.

RETURN VALUES

Upon successful completion, 0 is returned. Otherwise, -1 is returned, no message is sent, and `errno` is set to indicate the error.

ERRORS

The `msgsnd()` function will fail if:

EACCES

Operation permission is denied to the calling process.
See [intro\(2\)](#).

EAGAIN

The message cannot be sent for one of the reasons cited above and `(msg_flags & IPC_NOWAIT)` is non-zero.

EIDRM The message queue identifier `msgqid` is removed from the system.

EINTR The `msgsnd()` function was interrupted by a signal.

EINVAL

The value of `msgqid` is not a valid message queue identifier, or the value of `mtype` is less than 1; or the value of `msgsz` is less than 0 or greater than the system-imposed limit.

The `msgsnd()` function may fail if:

EFAULT

The `msgp` argument points to an illegal address.

USAGE

The value passed as the `msgp` argument should be converted to type `void *`.

SEE ALSO

[intro\(2\)](#), [msgctl\(2\)](#), [msgget\(2\)](#), [msgrcv\(2\)](#), [sigaction\(2\)](#)

System Calls

[msgrcv\(2\)](#)

NAME

`msgrcv` - message receive operation

SYNOPSIS

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

```
ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long int
msgtyp, int msgflg);
```

DESCRIPTION

The `msgrcv()` function reads a message from the queue associated with the message queue identifier specified by `_m_s_q_id` and places it in the user-defined buffer pointed to by `_m_s_g_p`.

The `_m_s_g_p` argument points to a user-defined buffer that must contain first a field of type `long int` that will specify the type of the message, and then a data portion that will hold the data bytes of the message. The structure below is an example of what this user-defined buffer might look like:

```
struct mymsg {
    long int  mtype; /* message type */
    char      mtext[1]; /* message text */
}
```

The `mtype` member is the received message's type as specified by the sending process.

The `mtext` member is the text of the message.

The `_m_s_g_s_z` argument specifies the size in bytes of `mtext`. The received message is truncated to `_m_s_g_s_z` bytes if it is larger than `_m_s_g_s_z` and `(_m_s_g_f_l_g & MSG_NOERROR)` is non-zero. The truncated part of the message is lost and no indication of the truncation is given to the calling process.

The `_m_s_g_t_y_p` argument specifies the type of message requested as follows:

- + o If `_m_s_g_t_y_p` is 0, the first message on the queue is received.
- + o If `_m_s_g_t_y_p` is greater than 0, the first message of type `_m_s_g_t_y_p` is received.
- + o If `_m_s_g_t_y_p` is less than 0, the first message of the lowest type that is less than or equal to the absolute value of `_m_s_g_t_y_p` is received.

The `_m_s_g_f_l_g` argument specifies which of the following actions is to be taken if a message of the desired type is not on the queue:

- + o If `(_m_s_g_f_l_g & IPC_NOWAIT)` is non-zero, the calling process will return immediately with a return value of -1 and `errno` set to `ENOMSG`.
- + o If `(_m_s_g_f_l_g & IPC_NOWAIT)` is 0, the calling process will suspend execution until one of the following occurs:
 - + o A message of the desired type is placed on the

queue.

- + o The message queue identifier `_m_s_q_ i_ d` is removed from the system (see `msgctl(2)`); when this occurs, `errno` is set equal to `EIDRM` and `-1` is returned.
- + o The calling process receives a signal that is to be caught; in this case a message is not received and the calling process resumes execution in the manner prescribed in `sigaction(2)`.

Upon successful completion, the following actions are taken with respect to the data structure associated with `_m_s_q_ i_ d` (see `intro(2)`):

- + o `msg_qnum` is decremented by 1.
- + o `msg_lrp_id` is set equal to the process ID of the calling process.
- + o `msg_rtime` is set equal to the current time.

RETURN VALUES

Upon successful completion, `msgrcv()` returns a value equal to the number of bytes actually placed into the buffer `_m_t_e_x_t`. Otherwise, `-1` is returned, no message is received, and `errno` is set to indicate the error.

ERRORS

The `msgrcv()` function will fail if:

E2BIG The value of `mtext` is greater than `_m_s_g_s_z` and `(_m_s_g_f_l_ g&MSG_NOERROR)` is 0.

EACCES

Operation permission is denied to the calling process.
See `intro(2)`.

EIDRM The message queue identifier `_m_s_q_ i_ d` is removed from the system.

EINTR The `msgrcv()` function was interrupted by a signal.

EINVAL

The `_m_s_q_ i_ d` argument is not a valid message queue identifier.

ENOMSG

The queue does not contain a message of the desired type and `(_m_s_g_f_l_ g&IPC_NOWAIT)` is non-zero.

The `msgrcv()` function may fail if:

EFAULT

The `_m_s_g_p` argument points to an illegal address.

USAGE

The value passed as the `_m_s_g_p` argument should be converted to type `void *`.

SEE ALSO

`intro(2)`, `msgctl(2)`, `msgget(2)`, `msgsnd(2)`, `sigaction(2)`

User Commands

`ipcs(1)`

NAME

`ipcs` - report inter-process communication facilities status

SYNOPSIS

```
/usr/bin/ipcs [ -aAbcimopqst ] [ -C_c_o_r_e_f_ i_l_e ] [ -N_n_a_m_e_l_ i_s_t ]
```

```
/usr/xpg4/bin/ipcs [ -aAbcimopqst ] [ -C_c_o_r_e_f_ i_l_e ] [ -N_n_a_m_e_l_ i_s_t ]
```

DESCRIPTION

The utility `ipcs` prints information about active inter-process communication facilities. The information that is displayed is controlled by the options supplied. Without options, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system.

`/usr/xpg4/bin/ipcs`

See NOTES.

OPTIONS

The following options are supported:

- `-m` Prints information about active shared memory segments.
- `-q` Prints information about active message queues.
- `-s` Prints information about active semaphores.

If `-m`, `-q`, or `-s` are specified, information about only those indicated is printed. If none of these three is specified, information about all three is printed subject to these options:

- `-a` Uses all XCU5 print options. (This is a shorthand notation for `-b`, `-c`, `-o`, `-p`, and `-t`.)
- `-A` Uses all print options. (This is a shorthand notation for `-b`, `-c`, `-i`, `-o`, `-p`, and `-t`.)

- b Prints information on biggest allowable size: maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores. See below for meaning of columns in a listing.
- c Prints creator's login name and group name. See below.
- C `_c_o_r_e_f_ i_l_e`
 Uses the file `_c_o_r_e_f_ i_l_e` in place of `/dev/mem` and `/dev/kmem`. Use a core dump obtained from `savecore(1M)` in place of `/dev/mem` and `/dev/kmem`. Without the `-C` option (default), the running system image is used.
- i Prints number of ISM attaches to shared memory segments.
- N `_n_a_m_e_l_ i_s_t`
 Uses the file `_n_a_m_e_l_ i_s_t` in place of `/dev/ksyms`.
- o Prints information on outstanding usage: number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.
- p Prints process number information: process ID of last process to send a message, process ID of last process to receive a message on message queues, process ID of creating process, and process ID of last process to attach or detach on shared memory segments. See below.
- t Prints time information: time of the last control operation that changed the access permissions for all facilities, time of last `msgsnd(2)` and last `msgrcv(2)` on message queues, time of last `shmat(2)` and last `shmdt(2)` on shared memory (see `shmop(2)`), time of last `semop(2)` on semaphores. See below.

The column headings and the meaning of the columns in an `ipcs` listing are given below; the letters in parentheses indicate the options that cause the corresponding heading to appear; "all" means that the heading always appears. Note: These options only determine what information is provided for each facility; they do not determine which facilities are listed.

- T (all)
 Type of the facility:
- q message queue
 - m shared memory segment
 - s semaphore

ID (all)

The identifier for the facility entry.

KEY (all)

The key used as an argument to `msgget(2)`, `semget(2)`,

or `shmget(2)` to create the facility entry. (Note: The key of a shared memory segment is changed to `IPC_PRIVATE` when the segment has been removed until all processes attached to the segment detach it.)

MODE (all)

The facility access modes and flags: The mode consists of 11 characters that are interpreted as follows. The first two characters are:

R A process is waiting on a `msgrcv(2)`.

S A process is waiting on a `msgsnd(2)`.

D The associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it. (Note: If the shared memory segment identifier is removed via an `IPC_RMID` call to `shmctl(2)` before the process has detached from the segment with `shmdt(2)`, the segment is no longer visible to `ipcs` and it will not appear in the `ipcs` output.)

C The associated shared memory segment is to be cleared when the first attach is executed.

- The corresponding special flag is not set.

The next nine characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused.

The permissions are indicated as follows:

r Read permission is granted.

w Write permission is granted.

a Alter permission is granted.

- The indicated permission is not granted.

OWNER (all)
The login name of the owner of the facility entry.

GROUP (all)
The group name of the group of the owner of the facility entry.

CREATOR (a,A,c)
The login name of the creator of the facility entry.

CGROUP (a,A,c)
The group name of the group of the creator of the facility entry.

CBYTES (a,A,o)
The number of bytes in messages currently outstanding on the associated message queue.

QNUM (a,A,o)
The number of messages currently outstanding on the associated message queue.

QBYTES (a,A,b)
The maximum number of bytes allowed in messages outstanding on the associated message queue.

LSPID (a,A,p)
The process ID of the last process to send a message to the associated queue.

LRPID (a,A,p)
The process ID of the last process to receive a message from the associated queue.

STIME (a,A,t)
The time the last message was sent to the associated queue.

RTIME (a,A,t)
The time the last message was received from the associated queue.

CTIME (a,A,t)
The time when the associated entry was created or changed.

ISMATTCH (a,i)
The number of ISM attaches to the associated shared memory segments.

NATTCH (a,A,o)
The number of processes attached to the associated shared memory segment.

SEGSZ (a,A,b)

The size of the associated shared memory segment.

CPID (a,A,p)

The process ID of the creator of the shared memory entry.

LPID (a,A,p)

The process ID of the last process to attach or detach the shared memory segment.

ATIME (a,A,t)

The time the last attach was completed to the associated shared memory segment.

DTIME (a,A,t)

The time the last detach was completed on the associated shared memory segment.

NSEMS (a,A,b)

The number of semaphores in the set associated with the semaphore entry.

NSEMS (a,A,b,t)

(For /usr/xpg4/bin/ipcs) The number of semaphores in the set associated with the semaphore entry.

OTIME (a,A,t)

The time the last semaphore operation was completed on the set associated with the semaphore entry.

ENVIRONMENT VARIABLES

See environ(5) for descriptions of the following environment variables that affect the execution of ipcs: LANG, LC_ALL, LC_CTYPE, LC_MESSAGES, and NLSPATH.

TZ Determine the timezone for the time strings written by ipcs.

FILES

/etc/group
group names

/etc/passwd
user names

/dev/mem
memory

/dev/ksyms
system namelist

SEE ALSO

ipcrm(1), savecore(1M), msgget(2), msgrcv(2), msgsnd(2), semget(2), semop(2), shmctl(2), shmget(2), shmop(2), attributes(5), environ(5)

NOTES

If the user specifies either the -C or -N flag, the real and effective UID/GID is set to the real UID/GID of the user invoking ipcs.

Things can change while ipcs is running; the information it gives is guaranteed to be accurate only when it was retrieved.

When the corresponding facility is not installed or has not been used since the last reboot, /usr/xpg4/bin/ipcs will report

```
"%s facility not in system.\n", _f_a_c_ i_l_ i_t_y
```

while /usr/bin/ipcs will report

```
"%s facility is inactive.\n", _f_a_c_ i_l_ i_t_y
```

where _f_a_c_ i_l_ i_t_y is "Message Queue", "Shared Memory", or "Semaphore", as appropriate.

```
/* Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */  
/* All Rights Reserved */
```

```
/* THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T */  
/* The copyright notice above does not evidence any */  
/* actual or intended publication of such source code. */
```

```
/*  
* Copyright (c) 1999-2000 by Sun Microsystems, Inc.  
* All rights reserved.  
*/
```

```
#ifndef _SYS_MSG_H  
#define _SYS_MSG_H
```

```
#pragma ident "@(#)msg.h 1.32 00/02/14 SMI"
```

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

```
#ifdef __cplusplus  
extern "C" {  
#endif
```

```
/*  
* IPC Message Facility.  
*/
```

```

/*
 * Implementation Constants.
 */

/*
 * Permission Definitions.
 */

#define MSG_R 0400 /* read permission */
#define MSG_W 0200 /* write permission */

/*
 * ipc_perm Mode Definitions.
 */

#define MSG_RWAIT 01000 /* a reader is waiting for a message */
#define MSG_WWAIT 02000 /* a writer is waiting to send */

/*
 * Message Operation Flags.
 */

#define MSG_NOERROR 010000 /* no error if big message */

typedef unsigned long msgqnum_t;
typedef unsigned long msglen_t;

/*
 * There is one msg structure for each message that may be in the system.
 */
struct msg {
    struct msg *msg_next; /* ptr to next message on q */
    long msg_type; /* message type */
    size_t msg_size; /* message text size */
    void *msg_addr; /* message text address */
};

/*
 * There is one msg queue id data structure for each q in the system.
 */

/*
 * Applications that read /dev/mem must be built like the kernel. A new
 * symbol "_KMEMUSER" is defined for this purpose.
 */

#if defined(_KERNEL) || defined(_KMEMUSER)

#include <sys/t_lock.h>

/* expanded msqid_ds structure */

struct msqid_ds {
    struct ipc_perm msg_perm; /* operation permission struct */
    struct msg *msg_first; /* ptr to first message on q */

```

```

        struct msg      *msg_last;      /* ptr to last message on q */
        msglen_t      msg_cbytes;      /* current # bytes on q */
        msgqnum_t     msg_qnum;       /* # of messages on q */
        msglen_t      msg_qbytes;     /* max # of bytes on q */
        pid_t         msg_lspid;      /* pid of last msgsnd */
        pid_t         msg_lrpid;      /* pid of last msgrcv */
#if defined(_LP64)
        time_t        msg_stime;       /* last msgsnd time */
        time_t        msg_rtime;       /* last msgrcv time */
        time_t        msg_ctime;       /* last change time */
#else
        time_t        msg_stime;       /* last msgsnd time */
        int32_t       msg_pad1;        /* reserved for time_t expansion */
        time_t        msg_rtime;       /* last msgrcv time */
        int32_t       msg_pad2;        /* time_t expansion */
        time_t        msg_ctime;       /* last change time */
        int32_t       msg_pad3;        /* time expansion */
#endif
        kcondvar_t    msg_cv;
        kcondvar_t    msg_qnum_cv;
        long          msg_pad4[3];     /* reserve area */
};

#else /* user definition */

struct msqid_ds {
    struct ipc_perm  msg_perm;        /* operation permission struct */
    struct msg      *msg_first;       /* ptr to first message on q */
    struct msg      *msg_last;       /* ptr to last message on q */
    msglen_t      msg_cbytes;        /* current # bytes on q */
    msgqnum_t     msg_qnum;         /* # of messages on q */
    msglen_t      msg_qbytes;       /* max # of bytes on q */
    pid_t         msg_lspid;        /* pid of last msgsnd */
    pid_t         msg_lrpid;        /* pid of last msgrcv */
#if defined(_LP64)
        time_t        msg_stime;       /* last msgsnd time */
        time_t        msg_rtime;       /* last msgrcv time */
        time_t        msg_ctime;       /* last change time */
#else
        time_t        msg_stime;       /* last msgsnd time */
        int32_t       msg_pad1;        /* reserved for time_t expansion */
        time_t        msg_rtime;       /* last msgrcv time */
        int32_t       msg_pad2;        /* time_t expansion */
        time_t        msg_ctime;       /* last change time */
        int32_t       msg_pad3;        /* time_t expansion */
#endif
        short        msg_cv;
        short        msg_qnum_cv;
        long          msg_pad4[3];     /* reserve area */
};

#endif /* _KERNEL */

#if defined(_KERNEL)

/*

```

* Size invariant version of SVR3 structure. Only kept around
* to support old binaries. Perhaps this can go away someday.
*/

```
struct o_msgqid_ds32 {
    struct o_ipc_perm32 msg_perm; /* operation permission struct */
    caddr32_t msg_first; /* ptr to first message on q */
    caddr32_t msg_last; /* ptr to last message on q */
    uint16_t msg_cbytes; /* current # bytes on q */
    uint16_t msg_qnum; /* # of messages on q */
    uint16_t msg_qbytes; /* max # of bytes on q */
    o_pid_t msg_lspid; /* pid of last msgsnd */
    o_pid_t msg_lrpid; /* pid of last msgrcv */
    time32_t msg_stime; /* last msgsnd time */
    time32_t msg_rtime; /* last msgrcv time */
    time32_t msg_ctime; /* last change time */
};
```

```
#endif /* _KERNEL */
```

```
#if defined(_SYSCALL32)
```

```
/* Kernel's view of the user ILP32 msgqid_ds structure */
```

```
struct msgqid_ds32 {
    struct ipc_perm32 msg_perm; /* operation permission struct */
    caddr32_t msg_first; /* ptr to first message on q */
    caddr32_t msg_last; /* ptr to last message on q */
    uint32_t msg_cbytes; /* current # bytes on q */
    uint32_t msg_qnum; /* # of messages on q */
    uint32_t msg_qbytes; /* max # of bytes on q */
    pid32_t msg_lspid; /* pid of last msgsnd */
    pid32_t msg_lrpid; /* pid of last msgrcv */
    time32_t msg_stime; /* last msgsnd time */
    int32_t msg_pad1; /* reserved for time_t expansion */
    time32_t msg_rtime; /* last msgrcv time */
    int32_t msg_pad2; /* time_t expansion */
    time32_t msg_ctime; /* last change time */
    int32_t msg_pad3; /* time expansion */
    int16_t msg_cv;
    int16_t msg_qnum_cv;
    int32_t msg_pad4[3]; /* reserve area */
};
```

```
/* Kernel's view of the user ILP32 msgbuf structure */
```

```
struct ipcmsgbuf32 {
    int32_t mtype; /* message type */
    char mtext[1]; /* message text */
};
```

```
/* Kernel's view of the user ILP32 msgsnap_head and msgsnap_mhead structures */
```

```
struct msgsnap_head32 {
    size32_t msgsnap_size; /* bytes consumed/required in the buffer */
    size32_t msgsnap_nmsg; /* number of messages in the buffer */
};
```

```

struct msgsnap_mhead32 {
    size32_t msgsnap_mlen; /* number of bytes in message that follows */
    int32_t  msgsnap_mtype; /* message type */
};

#endif /* _SYSCALL32 */

/*
 * User message buffer template for msgsnd and msgrcv system calls.
 */

#ifdef _KERNEL
struct ipcmsgbuf {
#else
struct msgbuf {
#endif /* _KERNEL */
#ifdef _XOPEN_SOURCE
    long    _mtype;      /* message type */
    char    _mtext[1];  /* message text */
#else
    long    mtype;      /* message type */
    char    mtext[1];   /* message text */
#endif
};

/*
 * Message information structure.
 */

struct msginfo {
    size_t    msgmax;      /* max message size */
    size_t    msgmnb;     /* max # bytes on queue */
    int       msgmni;     /* # of message queue identifiers */
    int       msgtql;     /* # of system message headers */
};

/*
 * Header and message header structures for msgsnap() system call.
 */
struct msgsnap_head {
    size_t    msgsnap_size; /* bytes consumed/required in the buffer */
    size_t    msgsnap_nmsg; /* number of messages in the buffer */
};

struct msgsnap_mhead {
    size_t    msgsnap_mlen; /* number of bytes in message that follows */
    long      msgsnap_mtype; /* message type */
};

/*
 * We have to be able to lock a message queue since we can
 * sleep during message processing due to a page fault in
 * copyin/copyout or iomove. We cannot add anything to the
 * msqid_ds structure since this is used in user programs
 * and any change would break object file compatibility.

```

```

* Therefore, we allocate a parallel array, msglock, which
* is used to lock a message queue. The array is defined
* in the msg master file. The following macro takes a
* pointer to a message queue and returns a pointer to the
* lock entry. The argument must be a pointer to a msgqid
* structure.
*/

```

```

#define MSGLOCK(X) &msglock[X - msgque]

```

```

#if !defined(_KERNEL)
#if defined(__STDC__)
int msgctl(int, int, struct msqid_ds *);
int msgget(key_t, int);
int msgids(int *, uint_t, uint_t *);
int msgsnap(int, void *, size_t, long);
ssize_t msgrcv(int, void *, size_t, long, int);
int msgsnd(int, const void *, size_t, int);
#else /* __STDC__ */
int msgctl();
int msgget();
int msgids();
int msgsnap();
int msgrcv();
int msgsnd();
#endif /* __STDC__ */
#endif /* !_KERNEL */

```

```

#ifdef _KERNEL

```

```

struct msglock {
    kmutex_t msglock_lock;
};

```

```

/*
* Defined in space.c, allocated/initialized in msg.c
*/

```

```

extern caddr_t      msg;          /* base address of message buffer */
extern struct msg *msggh;        /* message headers */
extern struct msqid_ds *msgque;  /* msg queue headers */
extern struct msglock *msglock;  /* locks for the message queues */
extern struct msg *msgfp;        /* ptr to head of free header list */
extern struct msginfo msginfo; /* message parameters */

```

```

#endif /* _KERNEL */

```

```

#ifdef __cplusplus
}
#endif

```

```

#endif /* _SYS_MSG_H */

```