**System V Message Queues**

System V message queues are identified by a message queue identifier

Any process with adequate privileges can place or read a message onto a given queue

For every message queue in the system, the kernel maintains the following structure of information

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

struct msqid\_ds {

struct ipc\_perm msg\_perm; /\* Ownership and permissions \*/

struct msg \*msg\_first; /\* ptr to first message on queue \*/

struct msg \*msg\_last; /\* ptr to last message on queue \*/

msglen\_t msg\_cbytes; /\* current # bytes on queue \*/

time\_t msg\_stime; /\* Time of last msgsnd \*/

time\_t msg\_rtime; /\* Time of last msgrcv \*/

time\_t msg\_ctime; /\* Time of last change \*/

unsigned long \_\_msg\_cbytes;

/\* Current number of bytes in queue (nonstandard) \*/

msgqnum\_t msg\_qnum; /\* Current number of messages in queue \*/

msglen\_t msg\_qbytes; /\* Max number of bytes allowed in queue \*/

pid\_t msg\_lspid; /\* PID of last msgsnd \*/

pid\_t msg\_lrpid; /\* PID of last msgrcv \*/

};

struct ipc\_perm {

key\_t \_\_key; /\* Key supplied to msgget (2) \*/

uid\_t uid; /\* Effective UID of owner \*/

gid\_t gid; /\* Effective GID of owner \*/

uid\_t cuid; /\* Effective UID of creator \*/

gid\_t cgid; /\* Effective GID of creator \*/

unsigned short mode;

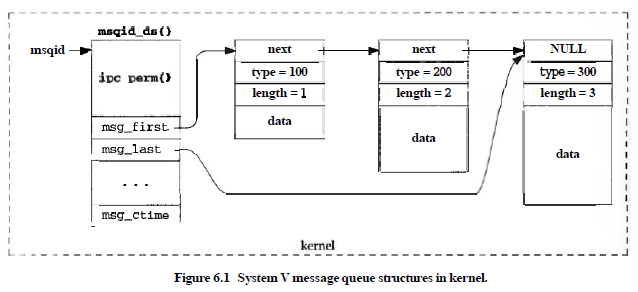
/\* Permissions + SHM\_DEST and SHM\_LOCKED flags \*/

unsigned short \_\_seq; /\* Sequence number \*/

};

Unix 98 does not require the msg\_first, msg\_last, or msg\_cbytes members

Naturally, no requirement exists that the messages on a queue be maintained as a linked list



|  |
| --- |
| Library interface |
| msgget() |
| msgsnd() |
| msgrcv() |
| msgctl() |

# msgget() function

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

**int msgget(key\_t key, int msgflg);**

Return:

On success message queue identifier (a nonnegative integer)

on error -1

* Get a System V message queue identifier
* Returns the System V message queue identifier associated with the value of the key argument
* A new message queue is created if key has the value IPC\_PRIVATE or
* key isn't IPC\_PRIVATE, no message queue with the given key key exists, and IPC\_CREAT is specified in msgflg
* If msgflg specifies both IPC\_CREAT and IPC\_EXCL and a message queue already exists for key, then msgget() fails
* Upon creation, the least significant bits of the argument msgflg define the permissions of the message queue
* If the message queue already exists the permissions are verified, and a check is made to see if it is marked for destruction

If a new message queue is created, structure msqid\_ds is initialized as follows:

msg\_perm.cuid and

msg\_perm.uid = effective user ID of the calling process

msg\_perm.cgid and

msg\_perm.gid = effective group ID of the calling process

msg\_perm.mode = least significant 9 bits of msgflg

msg\_qnum,

msg\_lspid, msg\_lrpid,

msg\_stime, and msg\_rtime = 0

msg\_ctime = current time

msg\_qbytes = system limit MSGMNB

**MSGMNI** - System-wide limit on the number of message queues

On Linux, this limit can be read and modified via /proc/sys/kernel/msgmni

# msgsnd() function

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

**int msgsnd(int msqid, const void \*msgp, size\_t msgsz, int msgflg);**

Return:

On success 0

on error -1

* Send messages to a System V message queue
* Calling process must have write permission on the message queue in order to send a message
* msgp argument is a pointer to a caller-defined structure of the following general form:

**struct msgbuf** {

long mtype; /\* message type, must be > 0 \*/

char mtext[1]; /\* message data \*/

};

* mtype field must have a strictly positive integer value.
* This value can be used by the receiving process for message selection
* mtext field is an array (or other structure) whose size is special fied by msgsz, a nonnegative integer value
* Messages of zero length (i.e., no mtext field) are permitted
* msgsnd() system call appends a copy of the message pointed to by msgp to the message queue
* If sufficient space is available in the queue, msgsnd() succeeds immediately
* queue capacity is governed by the msg\_qbytes field in the associated data structure struct msqid\_ds
* A message queue is considered to be full if:

Adding a new message to the queue would cause the total number of bytes in the queue to exceed the queue's maximum size (the msg\_qbytes field)

Adding another message to the queue would cause the total number of messages in the queue to exceed the queue's maximum size (the msg\_qbytes field)

* If insufficient space is available in the queue, then the default behavior of msgsnd() is to block until space becomes available
* If **IPC\_NOWAIT** is specified in msgflg, then the call instead fails with the error EAGAIN
* A blocked msgsnd() call may also fail if:

The queue is removed, in which case the system call fails with errno set to EIDRM; or

A signal is caught, in which case the system call fails with errno set to EINTR

* Upon success data structure struct msqid\_ds is updated as follows:

msg\_lspid = process ID of the calling process

msg\_qnum = msg\_qnum + 1 (incremented by 1)

msg\_stime = current time

# msgrcv() function

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

**ssize\_t msgrcv(int msqid, void \*msgp, size\_t msgsz, long msgtyp,**

**int msgflg);**

Return:

On success number of bytes actually copied into the mtext array

on error -1

* Receive messages from, a System V message queue
* Calling process must have read permission to receive a message
* msgp argument is a pointer to a caller-defined structure struct msgbuf
* msgrcv() system call removes a message from the queue and places it in the buffer pointed to by msgp
* Argument msgsz specifies the maximum size in bytes for the member mtext of the structure
* If the message text has length > msgsz, then the behavior depends on whether MSG\_NOERROR is specified in msgflg
* If MSG\_NOERROR is specified, then the message text will be truncated (and the truncated part will be lost)
* If MSG\_NOERROR is not specified, then the message isn't removed from the queue and the call fails with error E2BIG

Unless MSG\_COPY is specified in msgflg ,

the msgtyp argument specifies the type of message requested, as follows:

**msgtyp = 0**, then the first message in the queue is read

**msgtyp > 0**, If MSG\_EXCEPT is not specified, then the first message in the queue of type msgtyp is read,

If MSG\_EXCEPT is specified, in which case the first message in the queue of type not equal to msgtyp will be read

**msgtyp < 0**, then the first message in the queue with the lowest type less than or equal to the absolute value of msgtyp will be read

* msgflg argument is a bit mask constructed by ORing together zero or more of the following flags

**IPC\_NOWAIT | MSG\_COPY | MSG\_EXCEPT | MSG\_NOERROR**

* MSG\_COPY and MSG\_EXCEPT may not both be specified in msgflg
* If no message of the requested type is available and IPC\_NOWAIT isn't specified in msgflg, the calling process is blocked until one of the following conditions occurs:
* A message of the desired type is placed in the queue
* The message queue is removed from the system. In this case, the system call fails with errno set to EIDRM
* The calling process catches a signal. In this case, the system call fails with errno set to EINTR

Upon success data structure struct msqid\_ds is updated as follows:

msg\_lrpid = process ID of the calling process

msg\_qnum = msg\_qnum - 1 (decremented by 1)

msg\_rtime = current time.

# msgctl() function

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

**int msgctl(int msqid, int cmd, struct msqid\_ds \*buf);**

Return:

On success

* IPC\_STAT, IPC\_SET, and IPC\_RMID return 0
* IPC\_INFO or MSG\_INFO operation returns the index of the highest used entry in the kernel's internal array recording information about all message queues
* MSG\_STAT operation returns the identifier of the queue whose index was given in msqid

on error -1

* System V message control operations
* Performs the control operation specified by cmd on the System V message queue
* Valid values for cmd are:

IPC\_STAT, IPC\_SET, IPC\_RMID, IPC\_INFO, MSG\_INFO, MSG\_STAT

**IPC\_STAT**

* Copy information from the kernel data structure associated with msqid into the msqid\_ds structure pointed to by buf.
* The caller must have read permission on the message queue.

**IPC\_SET**

* Write the values of some members of the msqid\_ds structure pointed to by buf to the kernel data structure associated with this message queue, updating also its msg\_ctime member.
* The following members of the structure are updated: msg\_qbytes, msg\_perm.uid, msg\_perm.gid, and msg\_perm.mode (the least significant 9 bits).
* The effective UID of the calling process must match the owner (msg\_perm.uid) or creator (msg\_perm.cuid) of the message queue, or the caller must be privileged.
* Appropriate privilege (Linux: the CAP\_SYS\_RESOURCE capability) is required to raise the msg\_qbytes value beyond the system parameter MSGMNB.

**IPC\_RMID**

* Immediately remove the message queue, awakening all waiting reader and writer processes (with an error return and errno set to EIDRM).
* The calling process must have appropriate privileges or its effective user ID must be either that of the creator or owner of the message queue.
* The third argument to msgctl() is ignored in this case.

**IPC\_INFO** (Linux-specific)

Return information about system-wide message queue limits and parameters in the structure pointed to by buf.

This structure is of type msginfo (thus, a cast is required), defined in <sys/msg.h> if the \_GNU\_SOURCE feature test macro is defined:

**struct msginfo** {

int msgpool; /\* Size in kibibytes of buffer pool used to hold

message data; unused within kernel \*/

int msgmap; /\* Maximum number of entries in message map;

unused within kernel \*/

int msgmax; /\* Maximum number of bytes that can be written in

a single message \*/

int msgmnb; /\* Maximum number of bytes that can be written to queue; used to initialize msg\_qbytes during queue creation (msgget(2)) \*/

int msgmni; /\* Maximum number of message queues \*/

int msgssz; /\* Message segment size; unused within kernel \*/

int msgtql; /\* Maximum number of messages on all queues in

system; unused within kernel \*/

unsigned short int msgseg; /\* Maximum number of segments;

unused within kernel \*/

};

The msgmni, msgmax, and msgmnb settings can be changed via /proc files of the same name;

**MSG\_INFO** (Linux-specific)

* Return a msginfo structure containing the same information as for IPC\_INFO, except that the following fields are returned with information about system resources consumed by message queues:

**msgpool** field returns the number of message queues that currently exist on the system;

**msgmap** field returns the total number of messages in all queues on the system; and

**msgtql** field returns the total number of bytes in all messages in all queues on the system.

**MSG\_STAT** (Linux-specific)

* Return a msqid\_ds structure as for IPC\_STAT.

However, the msqid argument is not a queue identifier, but instead an index into the kernel's internal array that maintains information about all message queues on the system.

# Example

# END