# Posix Message Queues

* message queue - can be thought of as a linked list of messages
* Threads with adequate permission can put and remove messages from the queue
* Each message is a record and is assigned a priority by the sender
* **message queues have kernel persistence**

|  |  |
| --- | --- |
| Posix message queues | System V message queues |
| read returns the oldest message of the highest priority | read return a message of any desired priority |
| allow the generation of a signal  or the initiation of a thread  when a message is placed onto an empty queue |  |

* Every message on a queue has the following attributes:

an unsigned integer priority (Posix) or a long integer type (System V),

the length of the data portion of the message (which can be O), and

the data itself (if the length is greater than 0)

# 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 \*/

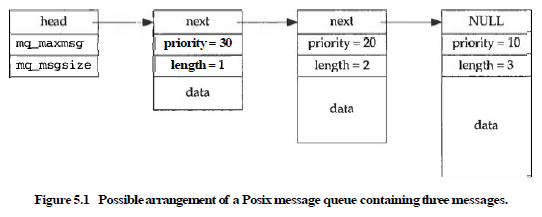
time\_t msg\_stime; /\* Time of last msgsnd \*/

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; /\* Maximum 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

**struct mq\_attr** {

~~long mq\_flags;~~  /\* Flags \*/

long mq\_maxmsg; /\* msgs per queue \*/

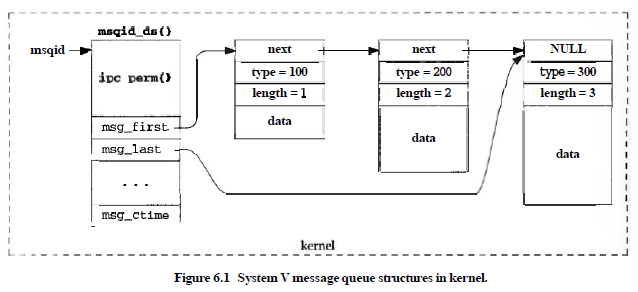
long mq\_msgsize; /\* bytes per msg \*/

~~long mq\_curmsgs;~~ /\* # of messages currently in queue\*/

};

~~ignored for mq\_open()~~

|  |  |
| --- | --- |
| **Library interface** | **System call** |
| mq\_open(3) | mq\_open(2) |
| mq\_close(3) | close(2) |
| mq\_unlink(3) | mq\_unlink(2) |
|  |  |
| mq\_getattr(3) | mq\_getsetattr(2) |
| mq\_setattr(3) | mq\_getsetattr(2) |
|  |  |
| mq\_send(3) | mq\_timedsend(2) |
| mq\_receive(3) | mq\_timedreceive(2) |
|  |  |
| mq\_timedsend(3) | mq\_timedsend(2) |
| mq\_timedreceive(3) | mq\_timedreceive(2) |
|  |  |
| mq\_notify(3) | mq\_notify(2) |



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

# mq\_open() function

#include <fcntl.h> /\* For O\_\* constants \*/

#include <sys/stat.h> /\* For mode constants \*/

#include <mqueue.h>

**mqd\_t mq\_open(const char \*name, int oflag);**

**mqd\_t mq\_open(const char \*name, int oflag, mode\_t mode,**

**struct mq\_attr \*attr);**

Return:

* On success, mq\_open() returns a message queue descriptor for use by other message queue functions
* On error, mq\_open() returns (mqd\_t) -1
* creates a new POSIX message queue or opens an existing queue.
* The queue is identified by name.
* oflag argument specifies flags that control the operation of the call

O\_RDONLY | O\_WRONLY | O\_RDWR | O\_CLOEXEC | O\_CREAT | O\_EXCL | O\_NONBLOCK

* If O\_CREAT is specified in oflag, then two additional arguments must be supplied
* The mode argument (flags S\_I\*) specifies the permissions to be placed on the new queue

# 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
* Structure mq\_attr is defined as follows:

**struct mq\_attr** {

~~long mq\_flags;~~  /\* Flags \*/

long mq\_maxmsg; /\* msgs per queue \*/

long mq\_msgsize; /\* bytes per msg \*/

~~long mq\_curmsgs;~~ /\* # of messages currently in queue\*/

};

~~ignored for mq\_open()~~

* Only the mq\_maxmsg and mq\_msgsize fields are employed when calling mq\_open(); the values in the remaining fields are ignored
* The fields of the struct mq\_attr pointed to attr specify the maximum number of messages and the maximum size of messages that the queue will allow.
* If attr is NULL, then the queue is created with implementation-defined default attributes

|  |  |  |
| --- | --- | --- |
| **Interface** | **Attribute** | **Value** |
| mq\_open() | Thread safety | MT-Safe |

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

# mq\_close() function

#include <mqueue.h>

**int mq\_close(mqd\_t mqdes);**

Return:

On success 0

on error -1

* closes the message queue descriptor mqdes.
* If the calling process has attached a notification request ( (mq\_notify() ) to this message queue via mqdes, then this request is removed, and another process can now attach a notification request.
* the calling process can no longer use the descriptor, but the message queue is not removed from the system.
* If the process terminates, all open message queues are closed, as if mq\_close were called.

|  |  |  |
| --- | --- | --- |
| **Interface** | **Attribute** | **Value** |
| mq\_close () | Thread safety | MT-Safe |

# 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

# mq\_unlink() function

#include <mqueue.h>

**int mq\_unlink(const char \*name);**

Return:

On success 0

on error -1

* removes the specified message queue name.
* The message queue name is removed immediately.
* Message queues have a reference count of how many times they are currently open
* the name can be removed from the system while its reference count is greater than 0,

but the destruction of the queue (versus removing its name from the system) does not take place until the last mq\_close() occurs (count reach 0)

|  |  |  |
| --- | --- | --- |
| **Interface** | **Attribute** | **Value** |
| mq\_unlink() | Thread safety | MT-Safe |

* 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

# mq\_getattr() function

#include <mqueue.h>

**int mq\_getattr(mqd\_t mqdes, struct mq\_attr \*attr);**

Return:

On success 0

on error -1

On Linux, **mq\_getattr()** and **mq\_setattr()** are library functions layered on top of the **mq\_getsetattr(2)** system call

* get message queue attributes
* retrieve attributes of the message queue referred to by the message queue descriptor mqdes
* returns an structure mq\_attr in the buffer pointed by attr
* mq\_flags: contains flags associated with the open message queue description 0 or O\_NONBLOCK
* mq\_maxmsg: (> 0) max number of messages that may be placed on the queue (messages per queue) using mq\_send()
* mq\_msgsize: (> 0) upper limit on the size of messages (bytes per message) that may be placed on the queue
* mq\_curmsgs: number of messages currently held in the queue

Since Linux 3.5, the following /proc files can be used to control the defaults

/proc/sys/fs/mqueue/msg\_default

/proc/sys/fs/mqueue/msgsize\_default

|  |  |  |
| --- | --- | --- |
| **Interface** | **Attribute** | **Value** |
| mq\_getattr() | Thread safety | MT-Safe |

* 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

# mq\_setattr() function

#include <mqueue.h>

**int mq\_setattr(mqd\_t mqdes, const struct mq\_attr \*newattr,**

**struct mq\_attr \*oldattr);**

Return:

On success 0

on error -1

On Linux, **mq\_getattr()** and **mq\_setattr()** are library functions layered on top of the **mq\_getsetattr(2)** system call

* set message queue attributes
* modify attributes of the message queue referred to by the message queue descriptor mqdes
* sets message queue attributes using information supplied in the structure mq\_attr pointed to by newattr
* **only attribute that can be modified is the setting of the O\_NONBLOCK flag in mq\_flags**
* **other fields in newattr are ignored**
* If the oldattr field is not NULL,
* then the buffer that it points to is used to return an mq\_attr structure that contains the same information that is returned by mq\_getattr()

|  |  |  |
| --- | --- | --- |
| **Interface** | **Attribute** | **Value** |
| mq\_setattr() | Thread safety | MT-Safe |

# 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 mes sage 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

# mq\_send() and mq\_timedsend() function

#include <mqueue.h>

**int mq\_send(mqd\_t mqdes, const char \*msg\_ptr, size\_t msg\_len,**

**unsigned int msg\_prio);**

#include <time.h>

#include <mqueue.h>

**int mq\_timedsend(mqd\_t mqdes, const char \*msg\_ptr, size\_t**

**msg\_len, unsigned int msg\_prio,**

**const struct timespec \*abs\_timeout);**

Return:

On success 0

on error -1

On Linux, **mq\_timedsend**() is a system call, and

**mq\_send**() is a library function layered on top of that system call

* send a message to a message queue
* adds the message pointed to by msg\_ptr to the message queue
* msg\_len argument specifies the length of the message pointed to by msg\_ptr
* **length <= queue's mq\_msgsize**
* Zero-length messages are allowed

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

* msg\_prio argument is a non -ve integer that specifies the priority of this message
* Messages are placed on the queue in decreasing order of priority (High P -> Low P)
* newer messages of the same priority being placed after older messages with the same priority (P1 old -> P1 new)
* Message priorities range from

0 (low) to sysconf(\_SC\_MQ\_PRIO\_MAX) - 1 (high)

* On Linux, sysconf(\_SC\_MQ\_PRIO\_MAX) returns 32768
* but POSIX.1 requires only that an implementation support at least priorities in the range 0 to 31; some implementations provide only this range
* If the message queue is already full mq\_send() blocks until

sufficient space becomes available or

until the call is interrupted by a signal handler

* If the O\_NONBLOCK flag is enabled for the message queue description, then the call instead fails immediately with the error EAGAIN
* **mq\_timedsend()** behaves just like mq\_send(), except that if the queue is full and the O\_NONBLOCK flag is not enabled then abs\_timeout points to a structure which specifies how long the call will block
* If the message queue is full, and the timeout has already expired by the time of the call, mq\_timedsend() returns immediately

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.

**struct timespec** {

time\_t tv\_sec; /\* seconds \*/

long tv\_nsec; /\* nanoseconds \*/

};

|  |  |  |
| --- | --- | --- |
| **Interface** | **Attribute** | **Value** |
| mq\_send(),mq\_timedsend() | Thread safety | MT-Safe |

# 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.

# mq\_receive() and mq\_timedreceive() function

#include <mqueue.h>

**ssize\_t mq\_receive(mqd\_t mqdes, char \*msg\_ptr, size\_t msg\_len,**

**unsigned int \*msg\_prio);**

#include <time.h>

#include <mqueue.h>

**ssize\_t mq\_timedreceive(mqd\_t mqdes, char \*msg\_ptr, size\_t**

**msg\_len, unsigned int \*msg\_prio,**

**const struct timespec \*abs\_timeout);**

Return:

On success number of bytes in the received message

on error -1

On Linux, **mq\_timedreceive**() is a system call, and

**mq\_receive**() is a library function layered on top of that system call

* receive a message from a message queue
* removes the oldest message with the highest priority from the message queue
* and places it in the buffer pointed to by msg\_ptr
* msg\_len argument specifies the size of the buffer pointed to by msg\_ptr
* **msg\_len must be >= mq\_msgsize attribute of the queue**
* If msg\_prio is not NULL, then the buffer to which it points is used to return the priority associated with the received message

**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.
* If the queue is empty, then, by default, mq\_receive() blocks until

a message becomes available, or

the call is interrupted by a signal handler

* If the O\_NONBLOCK flag is enabled for the message queue description, then the call instead fails immediately with the error EAGAIN
* **mq\_timedreceive()** behaves just like mq\_receive(), except that if the queue is empty and the O\_NONBLOCK flag is not enabled for the message queue description, then abs\_timeout points to a structure which specifies how long the call will block
* If no message is available, and the timeout has already expired by the time of the call, mq\_timedreceive() returns immediately

**struct timespec** {

time\_t tv\_sec; /\* seconds \*/

long tv\_nsec; /\* nanoseconds \*/

};

|  |  |  |
| --- | --- | --- |
| **Interface** | **Attribute** | **Value** |
| mq\_receive(),mq\_timedreceive() | Thread safety | MT-Safe |

**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;

# mq\_notify() function

#include <mqueue.h>

**int mq\_notify(mqd\_t mqdes, const struct sigevent \*sevp);**

Return:

On success 0

on error -1

* register for notification when a message is available allows the calling process to register or unregister for delivery of an asynchronous notification when a new message arrives on the empty message queue
* sevp argument is a pointer to a sigevent structure
* **If sevp is a non-null pointer,**
* then mq\_notify() registers the calling process to receive message notification
* sigev\_notify field of the sigevent structure to which sevp points specifies how notification is to be performed
* This field has one of the following values:

**SIGEV\_NONE**: the calling process is registered as the target for notification, but when a message arrives, no notification is sent

**SIGEV\_SIGNAL**: Notify the process by sending the signal specified in sigev\_signo

**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.

the siginfo\_t structure will be set to

si\_code = SI\_MESGQ

si\_pid = PID of the process that sent the message

si\_uid = real user ID of the sending process

**SIGEV\_THREAD**: Upon message delivery, invoke sigev\_notify\_function as if it were the start function of a new thread

* Only one process can be registered to receive notification from a message queue
* **If sevp is NULL**,
* and the calling process is currently registered to receive notifications for this message queue,

then the registration is removed;

another process can then register to receive a message notification for this queue

* Message notification occurs only when a new message arrives and the queue was previously empty
* If another process or thread is waiting to read a message from an empty queue using mq\_receive(3),

then any message notification registration is ignored:

the message is delivered to the process or thread calling mq\_receive(3),

and the message notification registration remains in effect

* Notification occurs once: after a notification is delivered, the notification registration is removed, and another process can register for message notification.
* If the notified process wishes to receive the next notification, it can use mq\_notify() to request a further notification.

This should be done before emptying all unread messages from the queue.

(Placing the queue in nonblocking mode is useful for emptying the queue of messages without blocking once it is empty.)

|  |  |  |
| --- | --- | --- |
| **Interface** | **Attribute** | **Value** |
| mq\_notify() | Thread safety | MT-Safe |

**struct sigevent** {

int sigev\_notify; /\* Notification method \*/

int sigev\_signo; /\* Notification signal \*/

union sigval sigev\_value; /\* Data passed with notification \*/

void (\*sigev\_notify\_function) (union sigval);

/\* Function used for thread notification (SIGEV\_THREAD) \*/

void \*sigev\_notify\_attributes;

/\* Attributes for notification thread (SIGEV\_THREAD) \*/

pid\_t sigev\_notify\_thread\_id;

/\* ID of thread to signal (SIGEV\_THREAD\_ID) \*/

};