**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

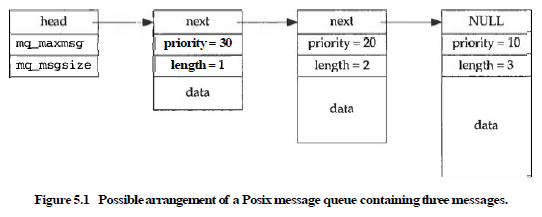
Each message is a record and is assigned a priority by the sender

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



**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) |

# 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.

O flag 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

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 |

# 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 |

# 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 |

# 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 |

# 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 |

# 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
* 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

**struct timespec** {

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

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

};

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

# 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
* 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 |

# 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

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) \*/

};

union sigval {

int sival\_int; /\* integer value \*/

void \*sival\_ptr; /\* pointer value \*/

};

## Example

#include "header.h"

mqd-t mqd;

void \*buff;

struct mq-attr attr;

struct sigevent sigev;

static void sig-usrl(int);

int main(int argc, char \*\*arm) {

if(argc != 2)

err\_quit("usage: mqnotifysigl <name>");

/\* open queue, get attributes, allocate read buffer \*/

mqd = mq\_open(argv[1], O\_RDONLY) ;

mq\_getattr(mqd, &attr);

buff = malloc(attr.mq\_msgsize);

/\* establish signal handler, enable notification \*/

Signal (SIGUSR1, sig-usrl) ;

sigev.sigev\_notify = SIGEV\_SIGNAL;

sigev.sigev\_signo = SIGUSR1:

mq\_notify(mqd, &sigev):

for ( ; ; )

pause ( ) ;

exit (0) ;

}

/\* signal handler does everything \*/

static void sig-usrl(int signo) {

ssize\_t n;

mq\_notify(mqd, &sigev); /\* re-register first \*/

n = mq\_receive(mqd, buff, attr.mq\_msgsize, NULL);

printf("SIGUSR1 received, read %Id bytes\n", (long) n);

return;

}

# Message Queue Limits

Two limits for any given queue, both of which are established when the queue is created:

**mq-maxmsg** The maximum number of messages on the queue, and

**mq-msgsize** The maximum size of a given message.

Two other limits are defined by the implementation:

**MQ\_OPEN\_MAX** The maximum number of message queues that a process can have open at once (Posix requires that this be at least 8)

**MQ\_PRIO\_MAX** The maximum value plus one for the priority of any message (Posix equires that this be at least 32).

These two constants are often defined in the <unistd.h> header and can also be obtained at run time by calling the sysconf function.

# Example



# END