

**NAME**

`mq_notify` - register for notification when a message is available

**SYNOPSIS**

```
#include <mqqueue.h>
```

```
int mq_notify(mqd_t mqdes, const struct sigevent *sevp);
```

Link with `-lrt`.

**DESCRIPTION**

`mq_notify()` allows the calling process to register or unregister for delivery of an asynchronous notification when a new message arrives on the empty message queue referred to by the descriptor `mqdes`.

The `sevp` argument is a pointer to a `sigevent` structure. For the definition and general details of this structure, see [sigevent\(7\)](#).

If `sevp` is a non-null pointer, then `mq_notify()` registers the calling process to receive message notification. The `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**

A null notification: 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`. See [sigevent\(7\)](#) for general details. The `si_code` field of the `siginfo_t` structure will be set to **SI\_MESGQ**. In addition, `si_pid` will be set to the PID of the process that sent the message, and `si_uid` will be set to the 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. See [sigevent\(7\)](#) for details.

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 the queue was not empty at the time `mq_notify()` was called, then a notification will occur only after the queue is emptied and a new message arrives.

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.)

**RETURN VALUE**

On success `mq_notify()` returns 0; on error, -1 is returned, with `errno` set to indicate the error.

**ERRORS**

**EBADF**

The descriptor specified in *mqdes* is invalid.

**EBUSY**

Another process has already registered to receive notification for this message queue.

**EINVAL**

*sevp->sigev\_notify* is not one of the permitted values; or *sevp->sigev\_notify* is **SIGEV\_SIGNAL** and *sevp->sigev\_signo* is not a valid signal number.

**ENOMEM**

Insufficient memory.

POSIX.1-2008 says that an implementation *may* generate an **EINVAL** error if *sevp* is NULL, and the caller is not currently registered to receive notifications for the queue *mqdes*.

**CONFORMING TO**

POSIX.1-2001.

**EXAMPLE**

The following program registers a notification request for the message queue named in its command-line argument. Notification is performed by creating a thread. The thread executes a function which reads one message from the queue and then terminates the process.

**Program source**

```
#include <pthread.h>
#include <mqqueue.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define handle_error(msg)
do { perror(msg); exit(EXIT_FAILURE); } while (0)

static void /* Thread start function */
tfunc(union sigval sv)
{
    struct mq_attr attr;
    ssize_t nr;
    void *buf;
    mqd_t mqdes = *((mqd_t *) sv.sival_ptr);

    /* Determine max. msg size; allocate buffer to receive msg */
    if (mq_getattr(mqdes, &attr) == -1)
        handle_error(mq_getattr);
    buf = malloc(attr.mq_msgsize);
    if (buf == NULL)
        handle_error(malloc);

    nr = mq_receive(mqdes, buf, attr.mq_msgsize, NULL);
    if (nr == -1)
        handle_error(mq_receive);

    printf("Read %zd bytes from MQn, nr);
    free(buf);
    exit(EXIT_SUCCESS); /* Terminate the process */
}

int
main(int argc, char *argv[])
{
```

```
mqd_t mqdes;
struct sigevent sev;

if (argc != 2) {
    fprintf(stderr, Usage: %s <mq-name>n, argv[0]);
    exit(EXIT_FAILURE);
}

mqdes = mq_open(argv[1], O_RDONLY);
if (mqdes == (mqd_t) -1)
    handle_error(mq_open);

sev.sigev_notify = SIGEV_THREAD;
sev.sigev_notify_function = tfunc;
sev.sigev_notify_attributes = NULL;
sev.sigev_value.sival_ptr = &mqdes; /* Arg. to thread func. */
if (mq_notify(mqdes, &sev) == -1)
    handle_error(mq_notify);

pause(); /* Process will be terminated by thread function */
}
```

**SEE ALSO**

[mq\\_close\(3\)](#), [mq\\_getattr\(3\)](#), [mq\\_open\(3\)](#), [mq\\_receive\(3\)](#), [mq\\_send\(3\)](#), [mq\\_unlink\(3\)](#), [mq\\_overview\(7\)](#), [sigevent\(7\)](#)

**COLOPHON**

This page is part of release 3.74 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at <http://www.kernel.org/doc/man-pages/>.