NAME

mq_notify − register for notification when a message is available

SYNOPSIS

#include <mqueue.h>

mqd_t mq_notify(mqd_t *mqdes***, const struct sigevent ****notification***);**

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 *notification* argument is a pointer to a *sigevent* structure that is defined something like the following:

```
union sigval {          /* Data passed with notification */
    int     sival_int;         /* Integer value */
    void   *sival_ptr;         /* Pointer value */
};

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 for thread
                                  notification */
    void        *sigev_notify_attributes;
                              /* Thread function attributes */
};
```

If *notification* is a non-NULL pointer, then **mq_notify**() registers the calling process to receive message notification. The *sigev_notify* field of the *sigevent* to which *notification* 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*. If the signal is caught with a signal handler that was registered using the **sigaction**(2) **SA_SIGINFO** flag, then the following fields are set in the *siginfo_t* structure that is passed as the second argument of the handler: *si_code* is set to **SI_MESGQ**; *si_signo* is set to the signal number; *si_value* is set to the value specified in *notification−>sigev_value*; *si_pid* is set to the PID of the process that sent the message; and *si_uid* is set to the real user ID of the sending process. The same information is available if the signal is accepted using **sigwaitinfo**(2).

SIGEV_THREAD
Deliver notification by invoking *notification−>sigev_notify_function* as the start function of a new thread. The function is invoked with *notification−>sigev_value* as its sole argument. If *notification−>sigev_notify_attributes* is not NULL, then it should point to a *pthread_attr_t* structure that defines attributes for the thread (see **pthread_attr_init**(3)).

Only one process can be registered to receive notification from a message queue.

If *notification* 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 only occurs 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 only occur 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 non-blocking 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
notification−>sigev_notify is not one of the permitted values; or *notification−>sigev_notify* is **SIGEV_SIGNAL** and *notification−>sigev_signo* is not a valid signal number.

ENOMEM
Insufficient memory.

POSIX.1-2008 says that an implementation *may* generate an **EINVAL** error if *notification* 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.

```
#include <pthread.h>
#include <mqueue.h>
#include <assert.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 %ld bytes from MQ\n", (long) nr);
    free(buf);
    exit(EXIT_SUCCESS);         /* Terminate the process */
}

int
main(int argc, char *argv[])
{
    mqd_t mqdes;
    struct sigevent not;

    assert(argc == 2);

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

    not.sigev_notify = SIGEV_THREAD;
    not.sigev_notify_function = tfunc;
    not.sigev_notify_attributes = NULL;
    not.sigev_value.sival_ptr = &mqdes;   /* Arg. to thread func. */
    if (mq_notify(mqdes, &not) == −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)

COLOPHON

This page is part of release 3.22 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.