NAME

mq_notify – notify process (or thread) that a message is available on a queue

SYNOPSIS

cc [*flag* ...] *file* ... **–lposix4** [*library* ...]

#include <mqueue.h>

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

```
struct sigevent {
        int             sigev_notify    /* notification type */
        int             sigev_signo;    /* signal number */
        union sigval    sigev_value;    /* signal value */
};
union sigval {
        int             sival_int;      /* integer value */
        void            *sival_ptr;     /* pointer value */
};
```

MT-LEVEL

MT-Safe

DESCRIPTION

mq_notify() provides an asynchronous mechanism for processes to receive notice that messages are available in a message queue, rather than synchronously blocking (waiting) in **mq_receive**(3R).

If *notification* is not **NULL**, this function registers the calling process to be notified of message arrival at an empty message queue associated with the message queue descriptor, *mqdes*. The notification specified by *notification* will be sent to the process when the message queue becomes non-empty. At any time, only one process may be registered for notification by a specific message queue. Also, if the calling process or any other process has already registered for notification of message arrival at the specified message queue, subsequent attempts to register for that message queue will fail.

notification points to a structure that defines both the signal to be generated and how the calling process will be notified upon I/O completion. If *notification*->**sigev_notify** is **SIGEV_NONE**, then no signal will be posted upon I/O completion, but the error status and the return status for the operation will be set appropriately. If *notification*->**sigev_notify** is **SIGEV_SIGNAL**, then the signal specified in *notification*->**sigev_signo** will be sent to the process. If the **SA_SIGINFO** flag is set for that signal number, then the signal will be queued to the process and the value specified in *notification*->**sigev_value** will be the **si_value** component of the generated signal (see **siginfo**(5)).

If *notification* is **NULL** and the process is currently registered for notification by the specified message queue, the existing registration is removed. The message queue is then available for future registration.

When the notification is sent to the registered process, its registration is removed. The message queue is then be available for registration.

If a process has registered for notification of message arrival at a message queue and some processes is blocked in **mq_receive**(3R) waiting to receive a message when a message arrives at the queue, the arriving message will be received by the appropriate **mq_receive**(3R), and no notification will be sent to the registered process. The resulting behavior is as if the message queue remains empty, and this notification will not be sent until the next arrival of a message at this queue.

Any notification registration is removed if the calling process either closes the message queue or exits.

RETURN VALUES

Upon successful completion, **mq_notify( )** returns **0**; otherwise, it returns a value of **-1** and sets **errno** to indicate the error condition.

ERRORS

EBADF	*mqdes* is not a valid message queue descriptor.
EBUSY	A process is already registered for notification by the message queue.
ENOSYS	**mq_notify( )** is not supported by this implementation.

SEE ALSO

mq_close(3R), **mq_open**(3R), **mq_receive**(3R), **mq_send**(3R), **siginfo**(5)

BUGS

In Solaris 2.5, these functions always return –**1** and set **errno** to **ENOSYS,** because this release does not support the Message Passing option. It is our intention to provide support for these interfaces in future releases.