NAME

mq_open - open a message queue (REALTIME)

LIBRARY

POSIX Real-time Library (librt, -lrt)

SYNOPSIS

#include <mqueue.h>

mqd_t
mq_open(*const char *name, int oflag, ...*);

DESCRIPTION

The **mq_open**() system call establishes the connection between a process and a message queue with a message queue descriptor. It creates an open message queue description that refers to the message queue, and a message queue descriptor that refers to that open message queue description. The message queue descriptor is used by other functions to refer to that message queue. The *name* argument points to a string naming a message queue. The *name* argument should conform to the construction rules for a pathname. The *name* should begin with a slash character. Processes calling **mq_open**() with the same value of *name* refers to the same message queue object, as long as that name has not been removed. If the *name* argument is not the name of an existing message queue and creation is not requested, **mq_open**() will fail and return an error.

The *oflag* argument requests the desired receive and/or send access to the message queue. The requested access permission to receive messages or send messages would be granted if the calling process would be granted read or write access, respectively, to an equivalently protected file.

The value of *oflag* is the bitwise-inclusive OR of values from the following list. Applications should specify exactly one of the first three values (access modes) below in the value of *oflag*:

O_RDONLY Open the message queue for receiving messages. The process can use the returned message queue descriptor with **mq_receive**(), but not **mq_send**(). A message queue may be open multiple times in the same or different processes for receiving messages.

O_WRONLY Open the queue for sending messages. The process can use the returned message queue descriptor with **mq_send**() but not **mq_receive**(). A message queue may be open multiple times in the same or different processes for sending messages.

O_RDWR Open the queue for both receiving and sending messages. The process can use any of the functions allowed for O_RDONLY and O_WRONLY. A message queue may be

open multiple times in the same or different processes for sending messages.

Any combination of the remaining flags may be specified in the value of *oflag*:

O_CREAT Create a message queue. It requires two additional arguments: *mode*, which is of type *mode_t*, and *attr*, which is a pointer to an *mq_attr* structure. If the pathname *name* has already been used to create a message queue that still exists, then this flag has no effect, except as noted under O_EXCL. Otherwise, a message queue will be created without any messages in it. The user ID of the message queue will be set to the effective user ID of the process, and the group ID of the message queue will be set to the effective group ID of the process. The permission bits of the message queue will be set to the value of the *mode* argument, except those set in the file mode creation mask of the process. When bits in *mode* other than the file permission bits are specified, the effect is unspecified. If *attr* is NULL, the message queue is created with implementation-defined default message queue attributes. If attr is non-NULL and the calling process has the appropriate privilege on name, the message queue *mq_maxmsg* and *mq_msgsize* attributes will be set to the values of the corresponding members in the *mq_attr* structure referred to by *attr*. If *attr* is non-NULL, but the calling process does not have the appropriate privilege on name, the **mq_open**() function will fail and return an error without creating the message queue.

O_EXCL If O_EXCL and O_CREAT are set, **mq_open**() will fail if the message queue name exists.

O_NONBLOCK Determines whether an **mq_send**() or **mq_receive**() waits for resources or messages that are not currently available, or fails with *errno* set to EAGAIN; see mq_send(2) and mq_receive(2) for details.

The **mq_open**() system call does not add or remove messages from the queue.

NOTES

FreeBSD implements message queue based on file descriptor. The descriptor is inherited by child after fork(2). The descriptor is closed in a new image after exec(3). The select(2) and kevent(2) system calls are supported for message queue descriptor.

Please see the mqueuefs(5) man page for instructions on loading the module or compiling the service into the kernel.

RETURN VALUES

Upon successful completion, the function returns a message queue descriptor; otherwise, the function

returns (*mqd_t*)-1 and sets the global variable *errno* to indicate the error.

ERRORS

The **mq_open**() system call will fail if:

[EACCES] The message queue exists and the permissions specified by *oflag* are denied, or the message queue does not exist and permission to create the message queue is denied.

[EEXIST] O_CREAT and O_EXCL are set and the named message queue already exists.

[EINTR] The **mq_open**() function was interrupted by a signal.

[EINVAL] The **mq_open**() function is not supported for the given name.

[EINVAL] O_CREAT was specified in *oflag*, the value of *attr* is not NULL, and either *mq_maxmsg* or *mq_msgsize* was less than or equal to zero.

[EMFILE] Too many message queue descriptors or file descriptors are currently in use by this process.

[ENAMETOOLONG] The length of the *name* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENFILE] Too many message queues are currently open in the system.

[ENOENT] O_CREAT is not set and the named message queue does not exist.

[ENOSPC] There is insufficient space for the creation of the new message queue.

SEE ALSO

mq_close(2), mq_getattr(2), mq_receive(2), mq_send(2), mq_setattr(2), mq_unlink(2), mq_timedreceive(3), mq_timedsend(3), mqueuefs(5)

STANDARDS

The **mq_open**() system call conforms to IEEE Std 1003.1-2004 ("POSIX.1").

HISTORY

Support for POSIX message queues first appeared in FreeBSD 7.0.

BUGS

This implementation places strict requirements on the value of *name*: it must begin with a slash (‘/’) and contain no other slash characters.

The *mode* and *attr* arguments are variadic and may result in different calling conventions than might otherwise be expected.

COPYRIGHT