

POSIX Message Queues



POSIX contains the following functions for communication through message queues.

`mq_open()`

establishes the connection between a process and a message queue and returns a queue descriptor. Arguments include a name given to the queue and flags indicating different options.

Options include:

- Open queue for send and/or receive.
- If queue does not exist, create new one with specified max message size and queue length.
- Use non-blocking send and receive

Slide 1

As an example of message passing functions used in existing operating systems, we consider these functions in POSIX standard. We will not cover all the syntax details here, as these can be found easily on-line.

Slide 2

The function `mq_open` (which stands for message queue open) establishes connection with an existing or new message queue. It returns a descriptor used to refer to the queue in other functions. Note that there is no distinction between a one-to-one queue and a mailbox, as both are used in the same way.

Slide 3

Queue can be open for sending, receiving, or both. If queue does not exist, a new one is created, in which case maximum message size and queue length are specified or left to a default value. Process selects to block waiting for a received message or not. Same applies when sending to a full queue.



```
#include <unistd.h>
#include <sys/types.h>
#include <mq.h>
#include <fcntl.h>

main ()
{
    int md;
    int status;

    /* Create message queue */
    md = mq_open ("my_queue", O_CREAT|O_RDWR);

    /*
     * code to close and unlink the message queue goes here
     */
    status = mq_close(md); /* Close message queue */
    status = mq_unlink("my_queue"); /* Unlink message queue */
}
```

Slide 4

In this example, `mq_open` is used to create a new queue with name “my_queue” for read and write. The integer variable `md` becomes the queue descriptor. If time permits, we will consider complete examples on Linux later.

POSIX Message Queues



`Mq_send ()`

Adds a message of specified size and priority to an opened queue. If queue is full it either blocks waiting or returns an error. A version of function allows timing out for more predictable timing.

No function is specified for synchronous send.

Slide 5

The function `mq_send` sends a message to an opened queue. In real-time version of POSIX, it is possible to wait for a full queue for a particular time before timing out and returning an error. POSIX does not have a send function with acknowledgment. If this is needed, user has to build it using normal functions (i.e. send ack message after receiving, and receiving an ack message after sending).

POSIX Message Queues

`Mq_send ()`

Adds a message of specified size and priority to an opened queue. If queue is full it either blocks waiting or returns an error. A version of function allows timing out for more predictable timing.

No function is specified for synchronous send.

`Mq_receive ()`

Reads the oldest of the highest priority messages in the opened queue into a local buffer in task. This message is removed from the queue.

Slide 6



```
char response [20];
struct mq_attr buf;

buf.mq_msgsize = sizeof (response);
buf.mq_maxmsg = 5;
mqd_t handle = mq_open ("/test", O_CREAT | O_RDONLY, 0, &buf);
printf ("return from initialize_message_queue, handle=%08x\n", handle);

int status = mq_receive (handle, (char *) &response, sizeof (response),
0);
printf ("return from mq_receive, status = %08x, errno = %08x, %s\n",
status, errno, strerror (errno) );
```

Slide 7

This is an example of real syntax for creating a queue for reading strings of 20 characters each. Queue can hold up to 5 strings. Mq_receive is then used to receive a string into a local variable.

POSIX Message Queues



If queue is empty it either blocks waiting (with possible time-out) or just returns an error.

If more than one task are waiting for an empty queue, the highest priority task will receive the first arriving message.

Handling of priority inversion is not specified. Implementations should consider this potential problem.

Slide 8

Again, in real-time systems, limiting the waiting time is necessary for predictability.

Slide 10

Use of messages may result in another form of priority inversion. This occurs specifically when a high priority process is blocked waiting for a message from a low priority process. To prevent uncontrollable long delays, sender should inherit the high priority of the receiver. Real-time POSIX standard indicates that this should be done, but does not specify implementation details.