UNIX System Programming
Lecture 16: Message Queues

• Outline
  – POSIX Message Queues

• Reference
  – BLP: Chapter 14
  – man pages: mq_open, mq_close, mq_send, mq_receive, mq_unlink.
Lecture 16: Message Queues

Introduction

- Message queues allow messages (data packets) to be passed from one process to another. There can be multiple writers to the queue as well as multiple readers.

- Message queues are often used for passing small messages between processes. They can also be used for process **synchronization**.
Lecture 16: Message Queues

POSIX Message Queues

• To create or open a POSIX message queue:

```c
mqd_t mq_open(
    const char *name,
    int flags,
    mode_t perms,
    struct mq_attr *attr);
```

• For portability the **name** should begin with a **slash** and contain no other slashes.
The `flags` argument should contain one of `O_RDONLY`, `O_WRONLY` or `O_RDWR`, depending on whether the process wants to receive or send or both.

`flags` may also be or’ed with `O_CREAT` to create the queue along with `O_EXCL` if you want the call to fail if the queue already exists. Use `O_NONBLOCK` if you do not want `mq_send()` or `mq_receive()` to block.
• **perms** and **attr** are only necessary when **flags** contains O_CREAT.

• **perms** is similar to the permissions on files (0600 for example). Read and write permission mean the ability to receive and send messages and **execute** permission is meaningless.
Lecture 16: Message Queues

POSIX Message Queues

- The `mq_maxmsg` and `mq_msgsize` fields of the `attr` structure set the maximum number of messages and the maximum message size respectively.

- A message queue descriptor is returned on success. On error, -1 is returned and `errno` is set appropriately.
Lecture 16: Message Queues

POSIX Message Queues

• To close a queue use `mq_close()`:
  ```c
  int mq_close(mgq_t mqd);
  ```

• To remove a queue use `mq_unlink()`:
  ```c
  int mq_unlink(const char *name);
  ```

• The name disappears immediately, but the queue is not removed until all open queue descriptors have been closed.
Lecture 16: Message Queues
POSIX Message Queues

• To send messages use `mq_send()`

```c
int mq_send(
    mqd_t mqd,
    const char *msg,
    size_t msgsize,
    unsigned priority);
```

• The `priority` must be greater than or equal to zero. Messages are placed in the queue in decreasing priority order.
• If the queue is full `mq_send()` will block until there is space. If `O_NONBLOCK` flag is set `ms_send()` returns -1 with error `EAGAIN`.

• `mq_timedsend()` takes an additional argument that specifies a time. In blocking mode this routine will return if the queue is full when the time is reached. Then -1 is returned with error `ETIMEDOUT`. 
Lecture 16: Message Queues
POSIX Message Queues

• To receive a message:

```c
ssize_t mq_receive(
    mqd_t mqd,
    char *msg,
    size_t msgsize,
    unsigned *priorityp);
```

• `msgsize` is the size of the `msg` buffer. It must be at least as big as the queue `mq_msgsize` attribute or the call will fail.
The oldest message with the highest priority is received. If the `priorityp` pointer is non-NULL then the message priority is returned at that address.

On Linux priorities values are in the range 0 (low) to 32767 (high). POSIX specifies a range of at least 0 to 31.
If the queue is empty `mq_receive()` will block until there is a message. If O_NONBLOCK flag is set `ms_receive()` returns -1 with error `EAGAIN`.

`mq_timedreceive()` takes an additional argument that specifies a time. In blocking mode this routine will return if the queue is still empty when the time is reached. Then -1 is returned with error `ETIMEDOUT`.
The `mq_getattr()` routine can be used to retrieve queue attributes: blocking mode, queue size, maximum message size and the current number of messages in the queue.

The `mq_setattr()` routine can be used to change the blocking mode of the queue.
You can use `mq_notify()` to request notification when a message arrives in an empty queue. The notification can be in the form of a signal or by invoking a specified function as a thread.
• On Linux, message queues are created in a virtual file system. This system is usually mounted over `/dev/mqueue`.

• Message queues have kernel persistence, unless removed via `mq_unlink()` they remain until a system reboot.
In Class Exercise

• Download the example programs. Modify the client so that it communicates with the server via a pair of message queues. The queues are created by the server.