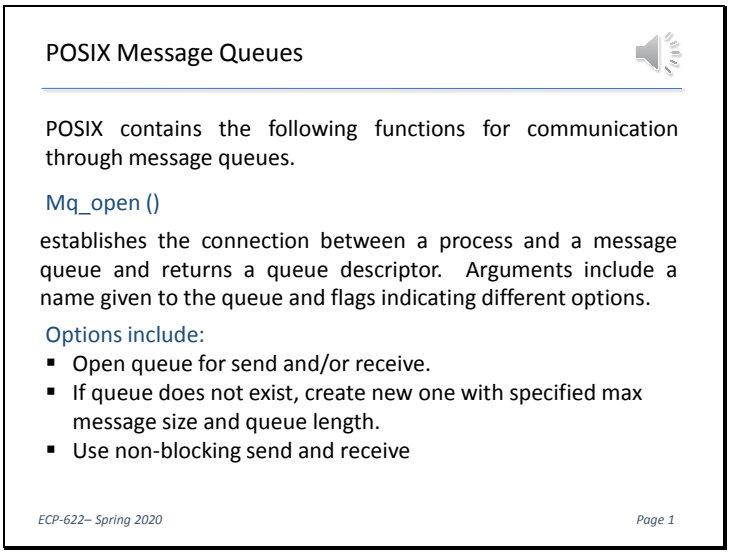


Slide 1

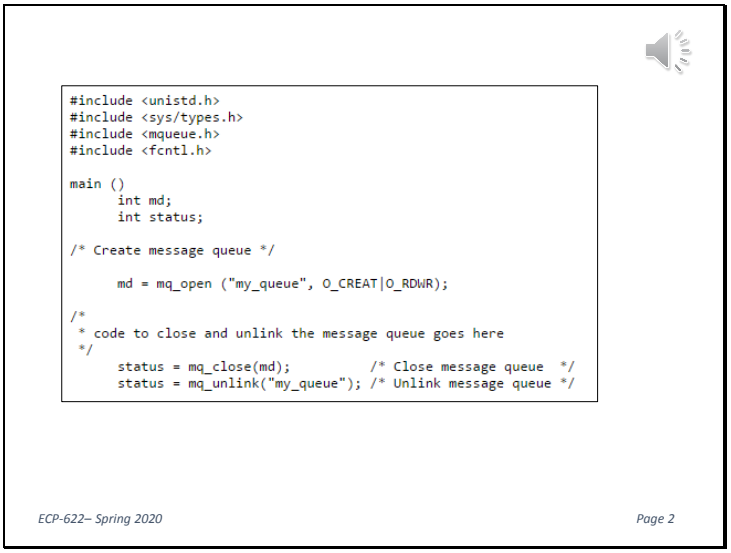
As an example of message passing functions used in existing operating systems, we consider these functions in POSIX standard. We will not cover all the syntax details here, as these can be found easily on-line.

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.

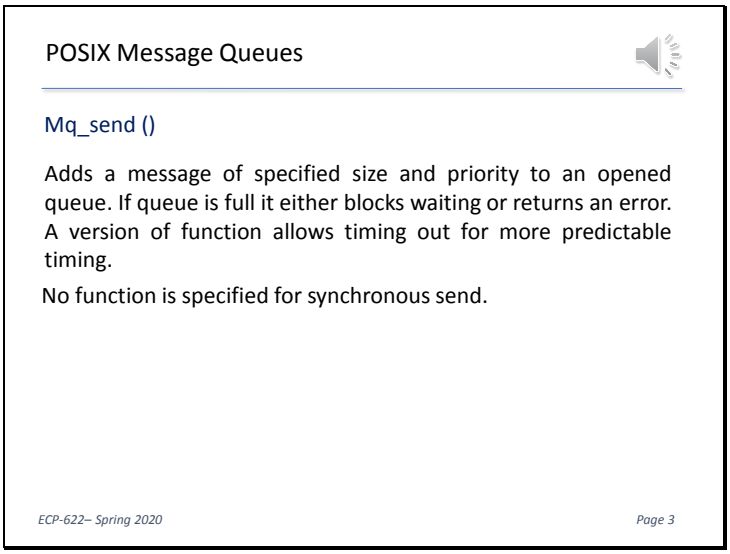
Slide 3

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.



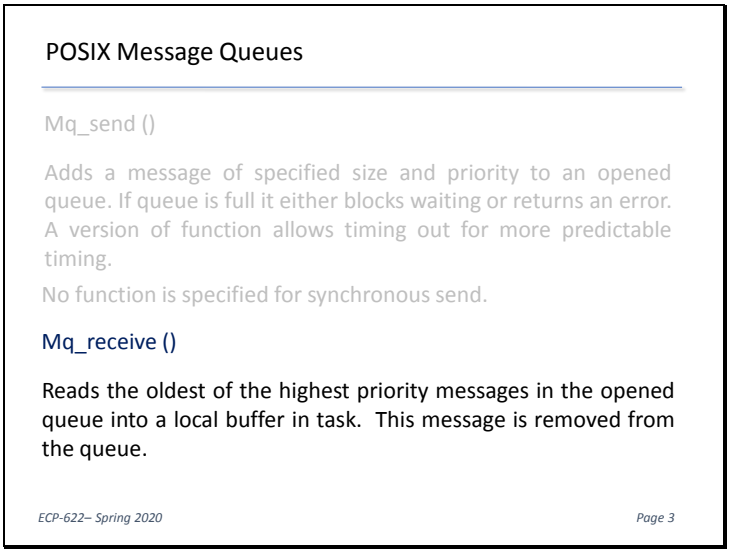
Slide 4

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.

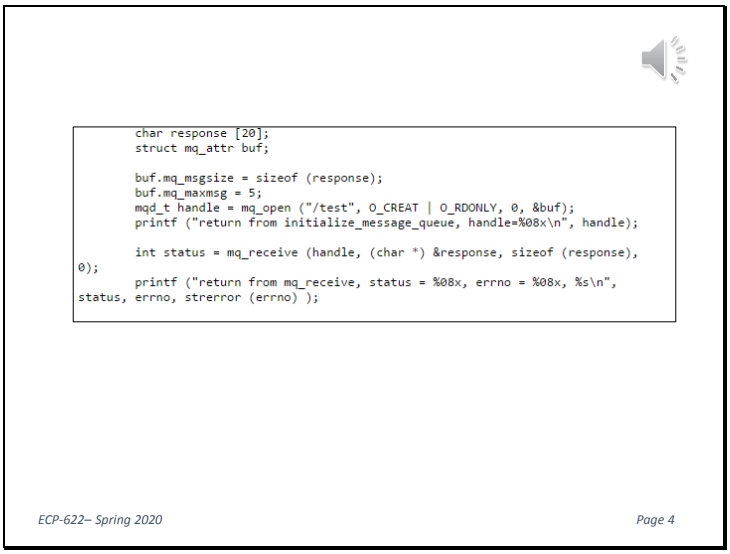


Slide 5

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).

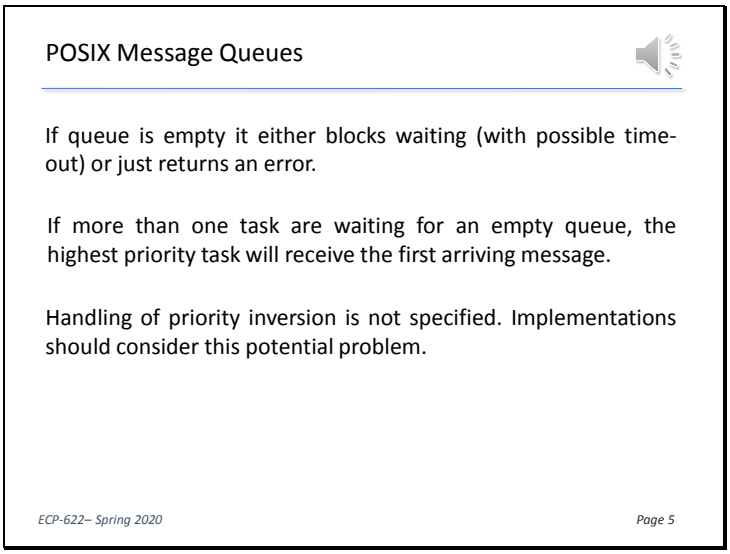


Slide 6



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.



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.