CENG 448 — Real Time Operating Systems

Message Queues

Dr. Larry D. Pyeatt

South Dakota School of Mines and Technology

Message Queue

A buffer-like object through which tasks and ISRs send and receive messages to communicate and synchronize.

Holds messages from sender until the intended recipient reads them.

Decouples the sending and receiving tasks, so they can each run at their own pace.

Message Queue

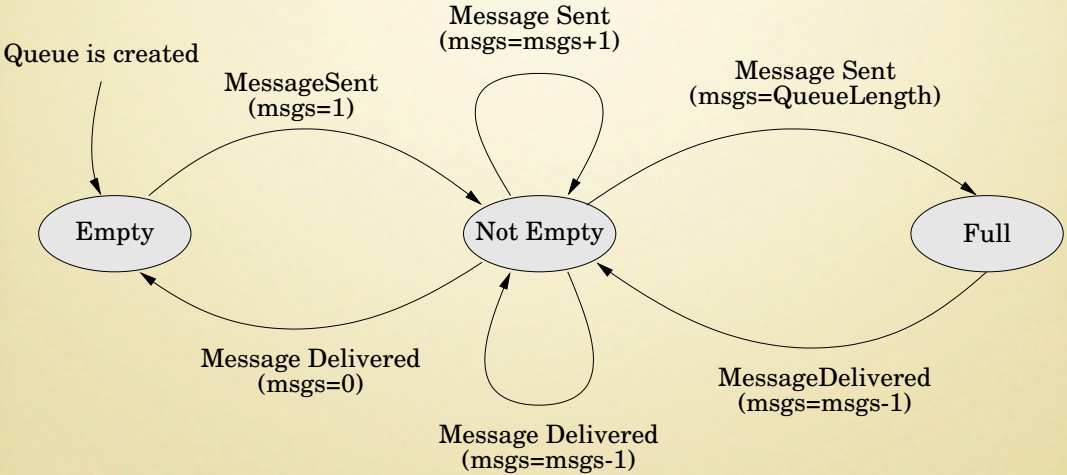
A message queue consists of these four data structures:

- Queue control block (QCB)
- Message storage area (array of message structures)
- Waiting list for senders (possibly prioritized)
- Waiting list for recipients (possibly prioritized)

The queue contains a number of elements, each of which can contain a single message.

May be implemented as a circular queue (array), or a linked list, etc.

Queue States



Blocking

A task that tries to read from an empty queue *may* be moved from the running state to the queue read waiting list (the task becomes blocked for reading).

A task that tries to write to a full queue *may* be moved from the running state to the queue write waiting list (blocked for writing).

If the queue is in the full state, and transitions to the not empty state, then the first task on the write waiting list is moved to the ready queue.

If queue is in the empty state, and transitions to the not empty state, then the first task on the read waiting list is moved to the ready queue.

Difference Between Message Queue, Message Buffer, and Stream Buffer

Pipe (or Stream Buffer) is byte-oriented. It is a simple stream of bytes.

Message queue deals with multi-byte data (structs, ints, pointers, etc).

FreeRTOS provides three variants for sending data between tasks.

Stream Buffer: Allows send and recieve for a simple stream of bytes.

Message Buffer: Handles messages of varying sizes.

Messague Queue: Handles messages of a fixed size.

Message content

The message content is usually user defined.

- temperature value from a sensor,
- bitmap to draw on a display,
- keyboard event,
- mouse event,
- network data packet,
- pointer to a buffer containing audio data,
- etc.

Most of these would be a C struct, but the temperature value could be an integer (fixed point) or floating point value.

For large messages, it may be better to put pointers in the queue.

System pool vs private buffer

System pool can be more efficient if there is limited memory and you know that some message queues will be empty whenever others are full. Memory is shared among queues, so as long as they are not all being used at once, then there may not be a problem. May require that queues be dynamically sized.

Private buffers guarantee that each queue will always have the memory it needs, but could waste memory overall.

Operations

- Create
- Delete
- Send
- Receive
- Broadcast
- Show info
- Show waiting lists

It usually does not make sense to have more than one source or more than one sink unless you have more than one CPU. Broadcasting is an exception.

Non-interlocked, one-way messages

Sources send whenever they have something to send.

Sinks receive whenever they are ready to do so.

There is communication in only one direction, though it could be many-to-many.

Typically, sinks have higher priority than sources, so they run as soon as the queue is not empty.

Interlocked, one-way messages

A *semaphore* is initialized with the number of sinks, and the length of the queue is set to the number of sinks.

Sources send whenever the semaphore indicates that it is clear to send (`semaphore` > 0), then decrement the semaphore.

Sinks receive whenever they are ready to do so, then increment the semaphore.

There is communication in only one direction, though it could be many-to-many, and synchronization is done with a semaphore.

Typically, sources have higher priority than sinks, so they run as soon as there are empty slots in the queue.

Interlocked, two-way messages

There are two message queues q_1 and q_2, and two or more tasks.

Some tasks send to q_1 and receive from q_2, while other tasks send to q_2 and receive from q_1.

None of the tasks can send to both queues.

None of the tasks can receive from both queues.

All tasks alternate between sending and receiving messages.

Broadcast

There is a single message queue.

There are at least two sinks that both must receive every message.

There is at least one source.

The sinks are all higher priority than the source. They run and try to receive, but become blocked.

The source sends a message that all *waiting* sinks receive.

How do you ensure that the source only sends when all sinks are waiting?