مدينة زويل للعلوم والتكنولوجيا
Zewail City of Science and Technology

COMMUNICATION AND INFORMATION ENGINEERING

CIE 314
Embedded Systems Fundamentals

Lecture #11
RTOS Events: Message Queues

Instructor:
Dr. Ahmad El-Banna

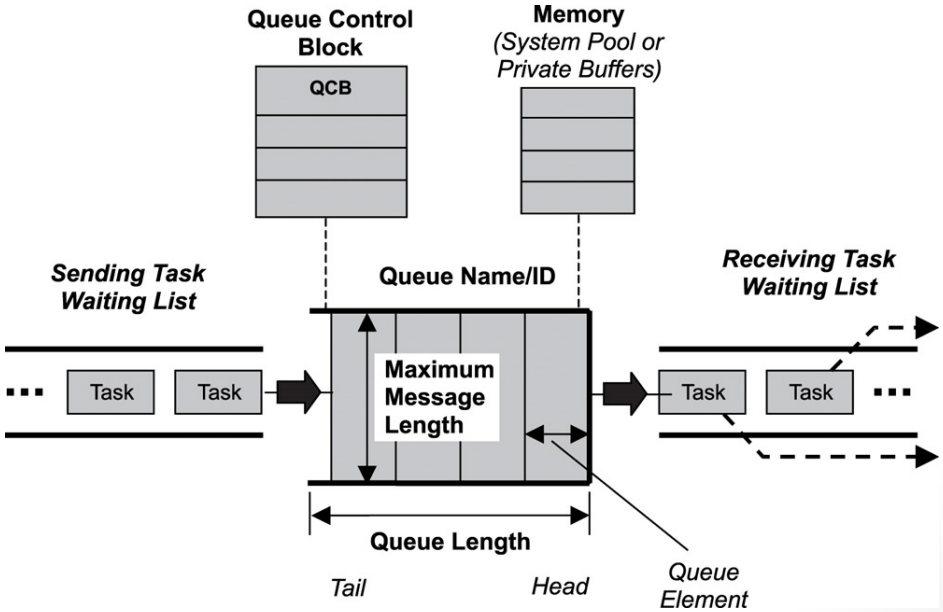
SPRING 2017

Agenda

- Defining message queues
- Message queue states
- Message queue content
- Typical message queue operations
- Typical message queue use.

Defining Message Queues

- A message queue is a buffer-like object through which tasks and ISRs send and receive messages to communicate and synchornize with data.
- A message queue is like a pipeline.
- It temporarily holds messages from a sender until the intended receiver is ready to read them.
- This temporary buffering decouples a sending and receiving task; that is, it frees the tasks from having to send and receive messages simultaneously.

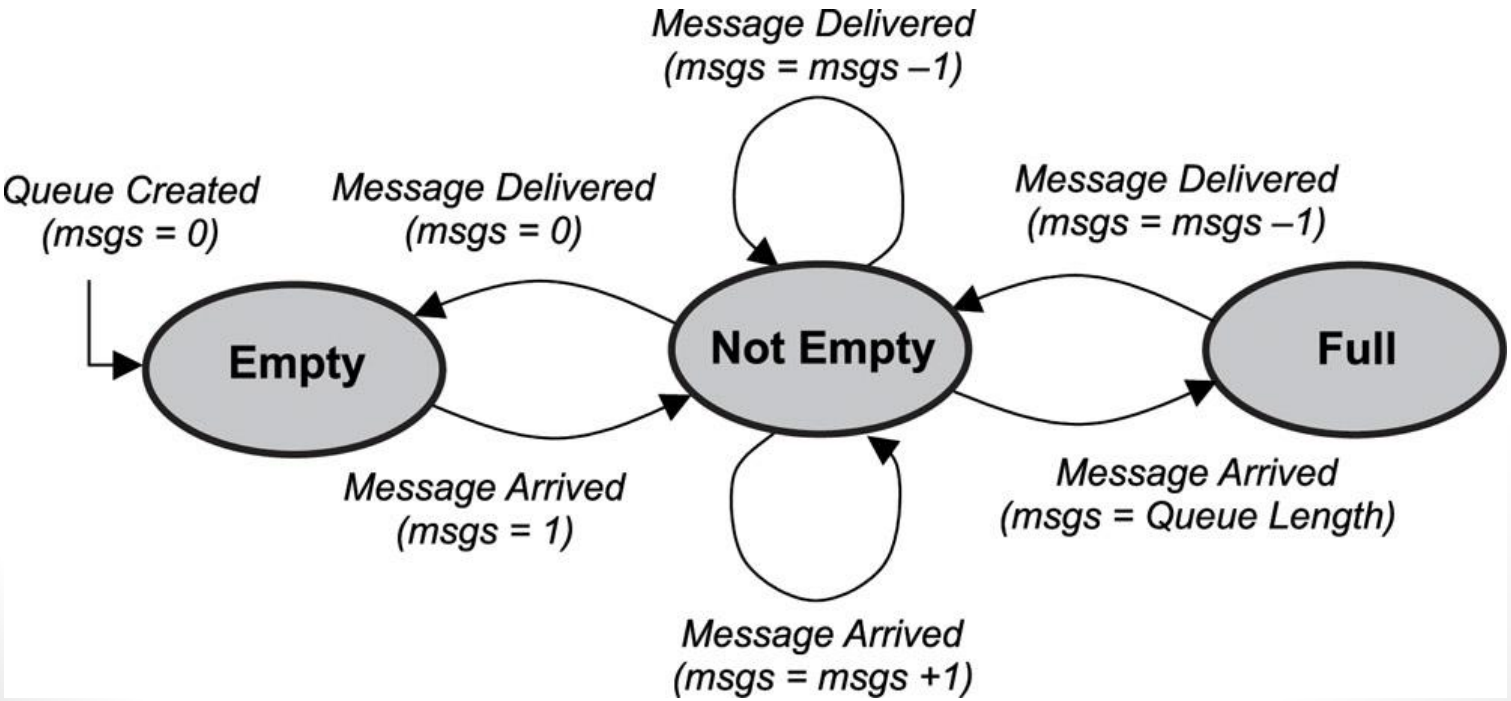


Defining Message Queues..

- It is the kernel's job to assign a unique ID to a message queue and to create its QCB and task-waiting list.
- The kernel also takes developer-supplied parameters—such as the length of the queue and the maximum message length—to determine how much memory is required for the message queue.
- After the kernel has this information, it allocates memory for the message queue from either a pool of system memory or some private memory space.
- The message queue itself consists of a number of elements, each of which can hold a single message.
- The elements holding the first and last messages are called the *head* and *tail* respectively.

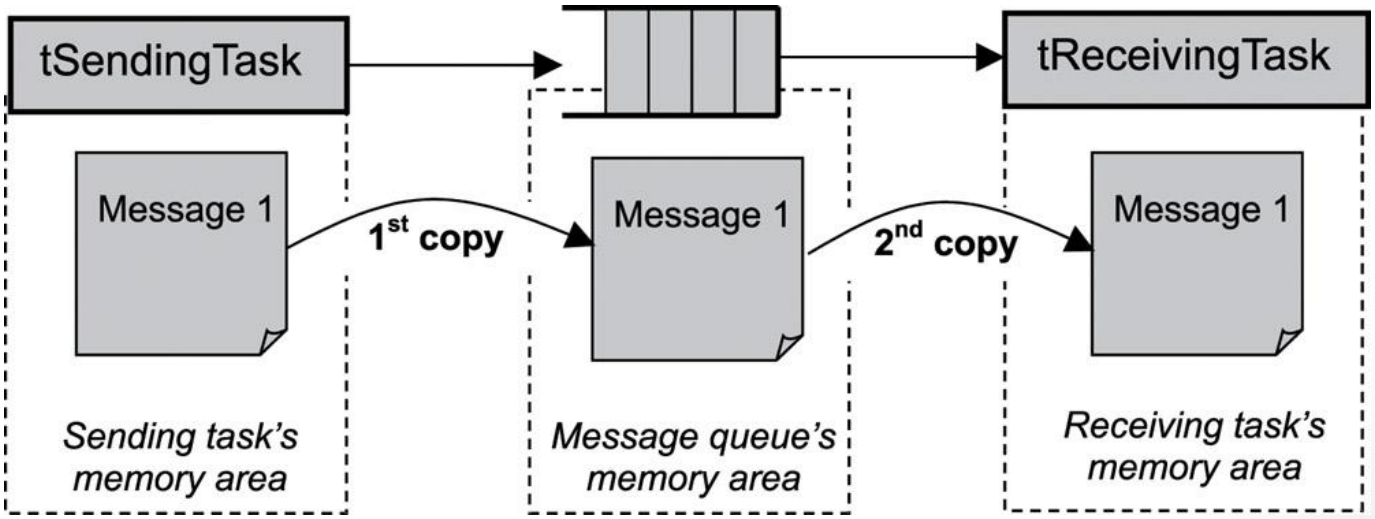
Message Queue States

- As with other kernel objects, message queues follow the logic of a simple FSM.
- When a message queue is first created, the FSM is in the empty state.
- If a task attempts to receive messages from this message queue while the queue is empty, the task blocks and, if it chooses to, is held on the message queue's task-waiting list, in either a FIFO or priority-based order.



Message Queue States..

- In this scenario, if another task sends a message to the message queue, the message is delivered directly to the blocked task.
- The blocked task is then removed from the task-waiting list and moved to either the ready or the running state. The message queue in this case remains empty because it has successfully delivered the message.
- If another message is sent to the same message queue and no tasks are waiting in the message queue's task-waiting list, the message queue's state becomes not empty.



Message Queue States...

- As additional messages arrive at the queue, the queue eventually fills up until it has exhausted its free space. At this point, the number of messages in the queue is equal to the queue's length, and the message queue's state becomes full.
- While a message queue is in this state, any task sending messages to it will not be successful unless some other task first requests a message from that queue, thus freeing a queue element.
- In some kernel implementations when a task attempts to send a message to a full message queue, the sending function returns an error code to that task.
- Other kernel implementations allow such a task to block, moving the blocked task into the sending task-waiting list, which is separate from the receiving task-waiting list.

Message Queue Content

- Message queues can be used to send and receive a variety of data.
- Some examples include:
  - a temperature value from a sensor,
  - a bitmap to draw on a display,
  - a text message to print to an LCD,
  - a keyboard event, and
  - a data packet to send over the network.
- Some of these messages can be quite long and may exceed the maximum message length, which is determined when the queue is created.
- One way to overcome the limit on message length is to send a pointer to the data, rather than the data itself.

Typical Message Queue Operations

- Typical message queue operations include the following:
  - creating and deleting message queues,
  - sending and receiving messages, and
  - obtaining message queue information.

Creating and Deleting Message Queues

Operation	Description
Create	Creates a message queue
Delete	Deletes a message queue

- When created, message queues are treated as global objects and are not owned by any particular task. Typically, the queue to be used by each group of tasks or ISRs is assigned in the design.
- When creating a message queue, a developer needs to make some initial decisions about the length of the message queue, the maximum size of the messages it can handle, and the waiting order for tasks when they block on a message queue.
- Deleting a message queue automatically unblocks waiting tasks. The blocking call in each of these tasks returns with an error. Messages that were queued are lost when the queue is deleted.

Sending and Receiving Messages

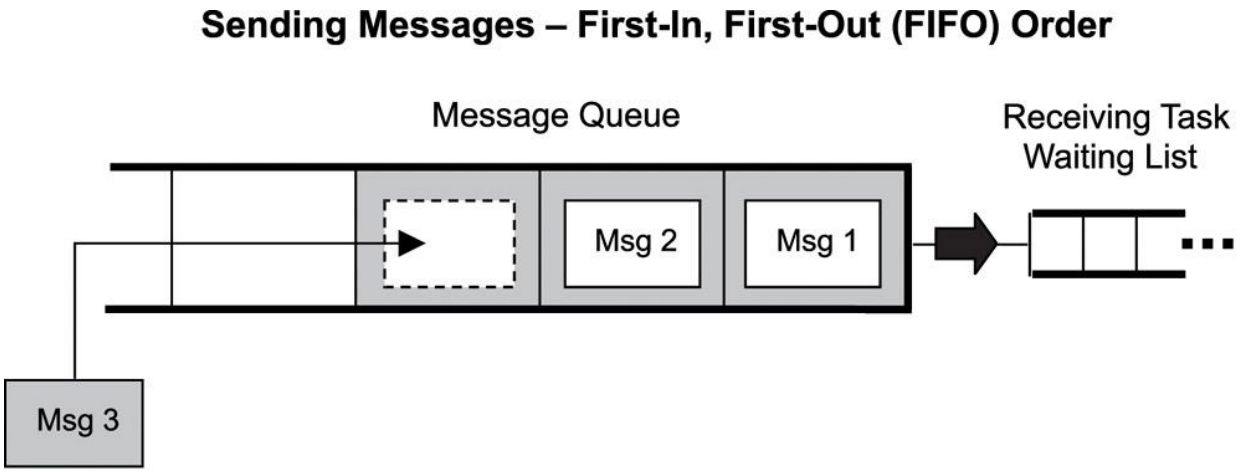
Operation	Description
Send	Sends a message to a message queue
Receive	Receives a message from a message queue
Broadcast	Broadcasts messages

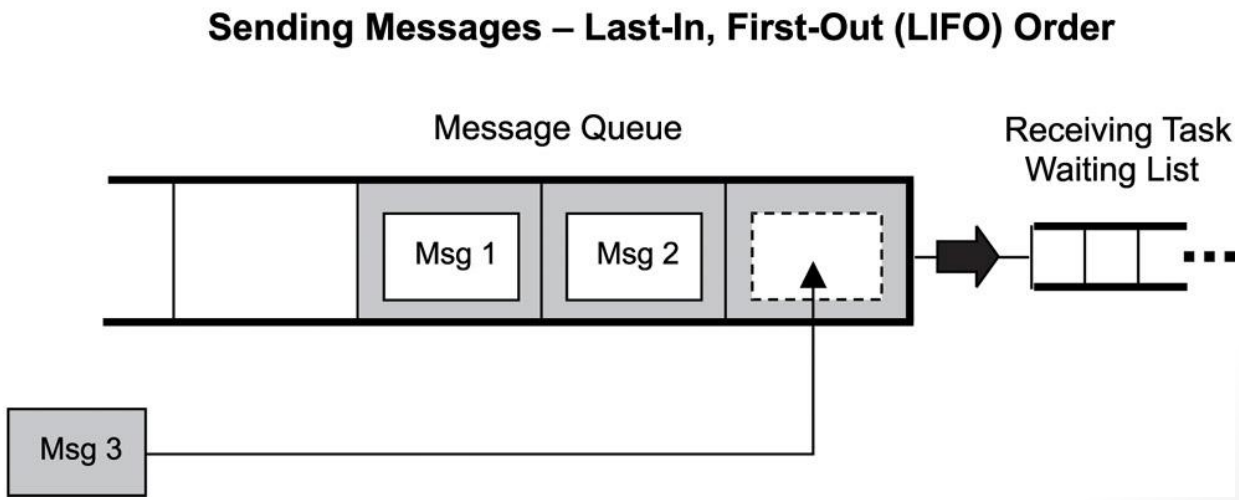
- The most common uses for a message queue are sending and receiving messages.

Sending Messages

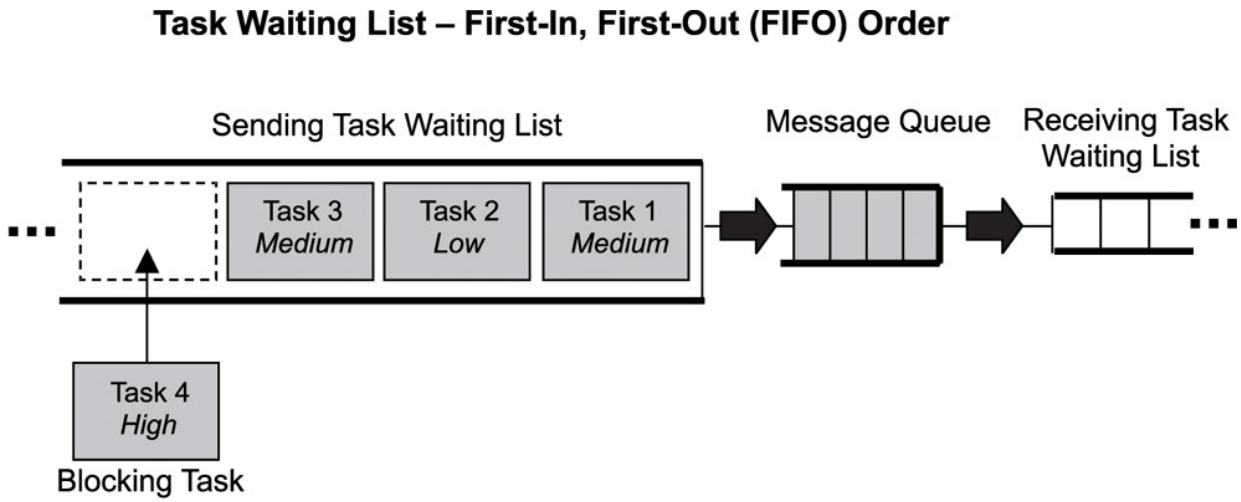
- When sending messages, a kernel typically fills a message queue from head to tail in FIFO or LIFO order.

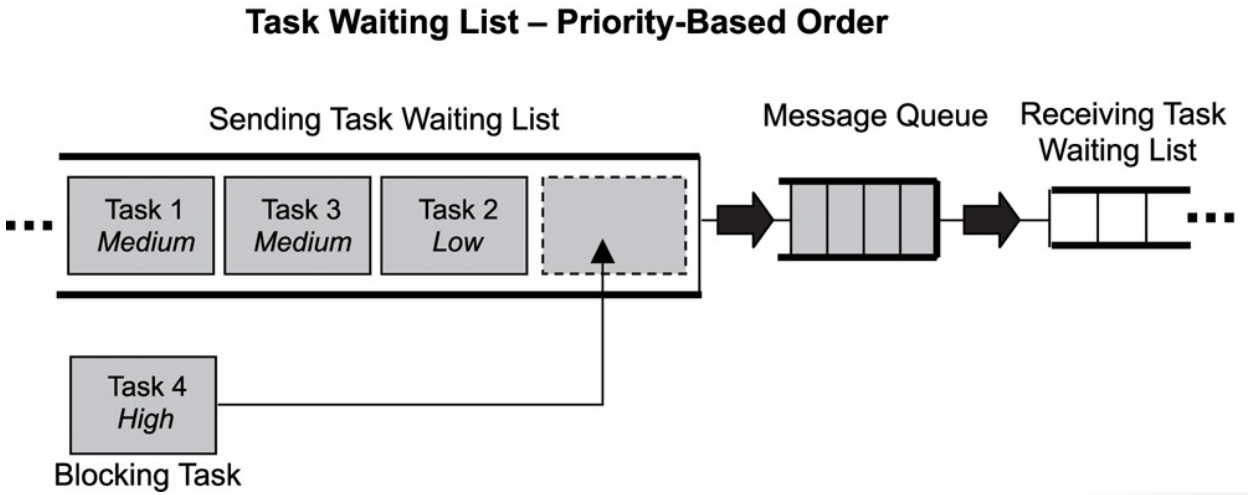
Sending messages in FIFO or LIFO order





FIFO and priority-based task-waiting lists





Receiving Messages

- As with sending messages, tasks can receive messages with different blocking policies—the same way as they send them—with a policy of not blocking, blocking with a timeout, or blocking forever.
- The diagram for the receiving tasks is similar to sending one, except that the blocked receiving tasks are what fills the task list.
- For the message queue to become full, either the receiving task list must be empty or the rate at which messages are posted in the message queue must be greater than the rate at which messages are removed. Only when the message queue is full does the task-waiting list for sending tasks start to fill.
- Conversely, for the task-waiting list for receiving tasks to start to fill, the message queue must be empty.
- Messages can be read from the head of a message queue in two different ways:
  - destructive read, and
  - non-destructive read.

Obtaining Message Queue Information

Operation	Description
Show queue info	Gets information on a message queue
Show queue's task-waiting list	Gets a list of tasks in the queue's task-waiting list

- Different kernels allow developers to obtain different types of information about a message queue, including the message queue ID, the queuing order used for blocked tasks (FIFO or priority-based), and the number of messages queued. Some calls might even allow developers to get a full list of messages that have been queued up.
- As with other calls that get information about a particular kernel object, be careful when using these calls. The information is dynamic and might have changed by the time it's viewed. These types of calls should only be used for debugging purposes.

Typical Message Queue Use

- The following are typical ways to use message queues within an application:
  - non-interlocked, one-way data communication,
  - interlocked, one-way data communication,
  - interlocked, two-way data communication, and
  - broadcast communication.

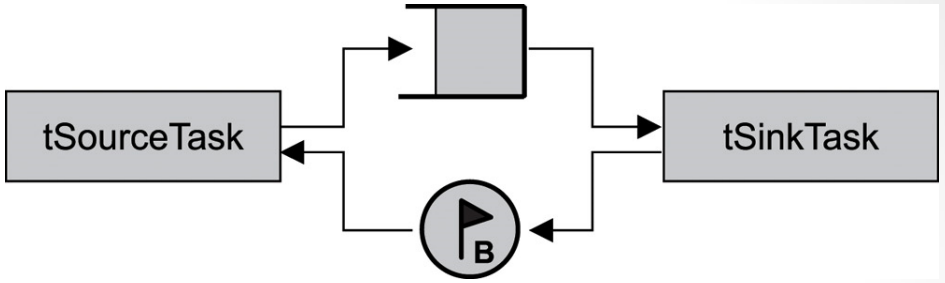
Non-Interlocked, One-Way Data Communication

- One of the simplest scenarios for message-based communications requires a sending task (also called the message source), a message queue, and a receiving task (also called a message sink).

Listing 7.1: Pseudo code for non-interlocked, one-way data communication.

```
tSourceTask ()
{
                :
                Send message to message queue
                :
}

tSinkTask ()
{
                :
                Receive message from message queue
                :
}
```

Interlocked, One-Way Data Communication

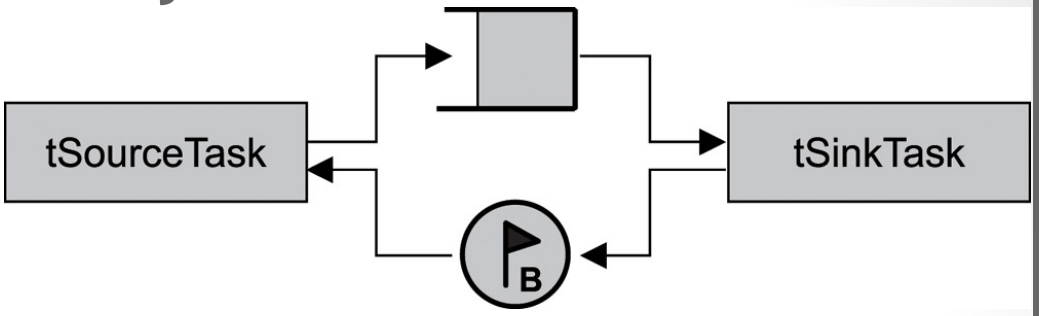


- In some designs, a sending task might require a handshake (acknowledgement) that the receiving task has been successful in receiving the message. This process is called interlocked communication, in which the sending task sends a message and waits to see if the message is received.

Listing 7.1: Pseudo code for non-interlocked, one-way data communication.

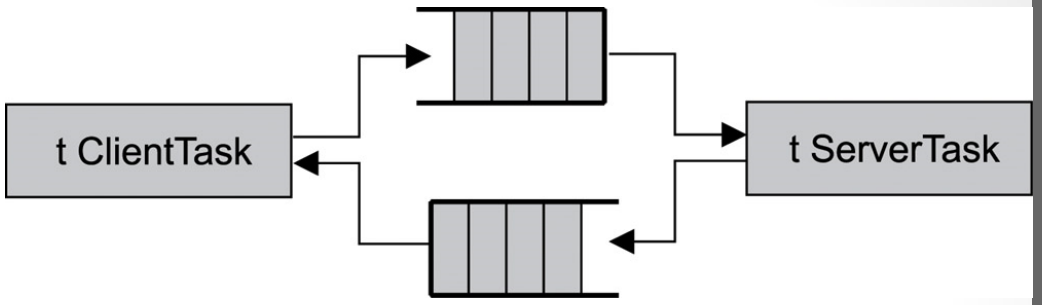
```
tSourceTask ()
{
                :
                Send message to message queue
                :
}

tSinkTask ()
{
                :
                Receive message from message queue
                :
}
```

Interlocked, One-Way Data Communication



- In some designs, a sending task might require a handshake (acknowledgement) that the receiving task has been successful in receiving the message. This process is called interlocked communication, in which the sending task sends a message and waits to see if the message is received.
- This requirement can be useful for reliable communications or task synchronization. For example, if the message for some reason is not received correctly, the sending task can resend it.

Interlocked, Two-Way Data Communication



- Sometimes data must flow bidirectionally between tasks, which is called interlocked, two-way data communication (also called full-duplex or tightly coupled communication). This form of communication can be useful when designing a client/server-based system.

Listing 7.4: Pseudo code for broadcasting messages.

```
tBroadcastTask ()
{
                :
                Send broadcast message to queue
                :
}

Note: similar code for tSignalTasks 1, 2, and 3.

tSignalTask ()
{
                :
                Receive message on queue
                :
}
```

Points to Remember

Some points to remember include the following:

- Message queues are buffer-like kernel objects used for data communication and synchronization between two tasks or between an ISR and a task.
- Message queues have an associated message queue control block (QCB), a name, a unique ID, memory buffers, a message queue length, a maximum message length, and one or more task-waiting lists.
- The beginning and end of message queues are called the head and tail, respectively; each buffer that can hold one message is called a message-queue element.
- Message queues are empty when created, full when all message queue elements contain messages, and not empty when some elements are still available for holding new messages.
- Sending messages to full message queues can cause the sending task to block, and receiving messages from an empty message queue can cause a receiving task to block.

Points to Remember..

- Tasks can send to and receive from message queues without blocking, via blocking with a timeout, or via blocking forever. An ISR can only send messages without blocking.
- The task-waiting list associated with a message-queue can release tasks (unblock them) in FIFO or priority-based order.When messages are sent from one task to another, the message is typically copied twice: once from the sending task's memory area to the message queue's and a second time from the message queue's memory area to the task's.
- The data itself can either be sent as the message or as a pointer to the data as the message. The first case is better suited for smaller messages, and the latter case is better suited for large messages.
- Common message-queue operations include creating and deleting message queues, sending to and receiving from message queues, and obtaining message queue information.
- Urgent messages are inserted at the head of the queue if urgent messages are supported by the message-queue implementation.
- Some common ways to use message queues for data based communication include non-interlocked and interlocked queues providing one-way or two-way data communication.

DESIGN TIPS

Washing machine system

- Think about its hardware design and software !

- For more details, refer to:
  - Chapter 7 at **Real-time concepts for embedded systems**, CMP Books, 2003 by Qing Li and Carolyn Yao (ISBN:1578201241).
  - Chapter 5 at **Embedded Software Development with C**, Springer 2009 by Kai Qian et al.
  - Chapter 8,9,10 at **Introduction to Embedded Systems,** Springer 2014 by Manuel Jiménez et al.

- The lecture is available online at:
  - *http://bu.edu.eg/staff/ahmad.elbanna-courses*

- For inquires, send to:
  - ahmad.elbanna@feng.bu.edu.eg