FreeRTOS with Arduino | Queues

Engr. Hamza Ali Imran

Note: The source of this content is entirely freeRTOS.org. In case you need any further information you can refer to its API section.

TOC

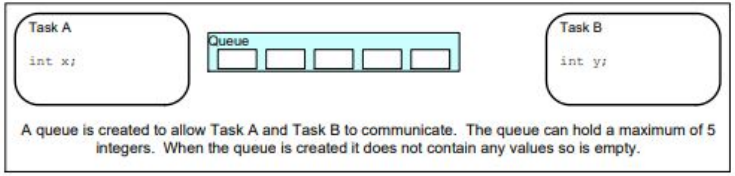
Queue

Data Storage

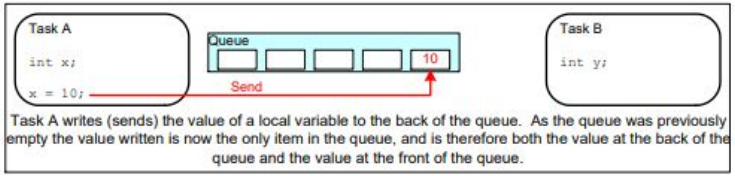
A queue can hold a finite number of fixed size data items. The maximum number of items a queue can hold is called its ‘length’. Both the length and the size of each data item are set when the queue is created.

Queues are normally used as First In First Out (FIFO) buffers, where data is written to the end (tail) of the queue and removed from the front (head) of the queue. Figure ahead demonstrates

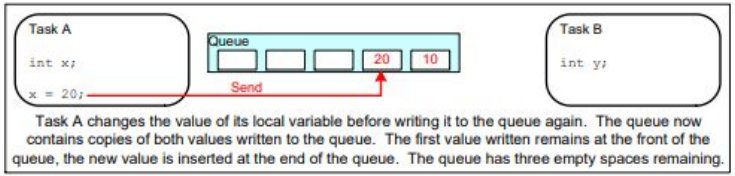
data being written to and read from a queue that is being used as a FIFO. It is also possible to write to the front of a queue, and to overwrite data that is already at the front of a queue.



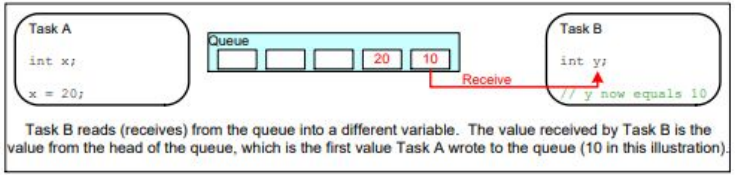
A queue is created to allow Task A and Task B to communicate. The queue can hold a maximum of 5 integers. When the queue is created it does not contain any values so is empty.



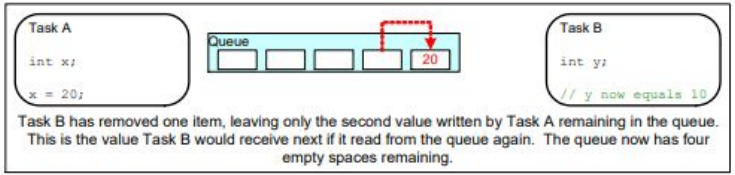
Task A writes (sends) the value of a local variable to the back of the queue. As the queue was previously empty the value written is now the only item in the queue, and is therefore both the value at the back of the queue and the value at the front of the queue.



Task A changes the value of its local variable before writing it to the queue again. The queue now contains copies of both values written to the queue. The first value written remains at the front of the queue, the new value is inserted at the end of the queue. The queue has three empty spaces remaining.



Task B reads (receives) from the queue into a different variable. The value received by Task B is the value from the head of the queue, which is the first value Task A wrote to the queue (10 in this illustration).



Task B has removed one item, leaving only the second value written by Task A remaining in the queue. This is the value Task B would receive next if it read from the queue again. The queue now has four empty spaces remaining.

Create a Queue

The xQueueCreate() API Function

A queue must be explicitly created before it can be used.

Queues are referenced by handles, which are variables of type QueueHandle_t. The

xQueueCreate() API function creates a queue and returns a QueueHandle_t that references

the queue it created.

The RAM is used to hold both the queue data structures and the items that are contained in the queue. xQueueCreate() will return NULL if there is insufficient heap RAM available for the queue to be created. Chapter 2 provides more information on the FreeRTOS heap.

QueueHandle_t xQueueCreate(UBaseType_t uxQueueLength, UBaseType_t uxItemSize)

Create a Queue

Parameter Name	Description
uxQueueLength	The maximum number of items that the queue being created can hold at any one time.
uxItemSize	The size in bytes of each data item that can be stored in the queue.
Return Value	If NULL is returned, then the queue cannot be created because there is insufficient heap memory available for FreeRTOS to allocate the queue data structures and storage area. A non-NULL value being returned indicates that the queue has been created successfully. The returned value should be stored as the handle to the created queue.

Send Data

BaseType_t xQueueSend(QueueHandle_t xQueue, const void * pvItemToQueue, TickType_t xTicksToWait);

xQueueSend() is equivalent to, and exactly the same as, xQueueSendToBack().

Parameters:

1. xQueue : The handle to the queue on which the item is to be posted.
2. pvItemToQueue : A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
3. xTicksToWait : The maximum amount of time the task should block waiting for space to become available on the queue, should it already be full. The call will return immediately if the queue is full and xTicksToWait is set to 0. The time is defined in tick periods so the constant portTICK_PERIOD_MS should be used to convert to real time if this is required.

Setting xTicksToWait to portMAX_DELAY will cause the task to wait indefinitely (without timing out), provided INCLUDE_vTaskSuspend is set to 1 in FreeRTOSConfig.h.

Receive Data

xQueueReceive() is used to receive (read) an item from a queue. The item that is received is removed from the queue.

BaseType_t xQueueReceive(QueueHandle_t xQueue, void * const pvBuffer, TickType_t xTicksToWait);

Parameter Name/ Returned value	Description
xQueue	The handle of the queue from which the data is being received (read). The queue handle will have been returned from the call to xQueueCreate() used to create the queue.
pvBuffer	A pointer to the memory into which the received data will be copied. The size of each data item that the queue holds is set when the queue is created. The memory pointed to by pvBuffer must be at least large enough to hold that many bytes.
xTicksToWait	The maximum amount of time the task should remain in the Blocked state to wait for data to become available on the queue, should the queue already be empty. If xTicksToWait is zero, then xQueueReceive() will return immediately if the queue is already empty. The block time is specified in tick periods, so the absolute time it represents is dependent on the tick frequency. The macro pdMS_TO_TICKS() can be used to convert a time specified in milliseconds into a time specified in ticks. Setting xTicksToWait to portMAX_DELAY will cause the task to wait indefinitely (without timing out) provided INCLUDE_vTaskSuspend is set to 1 in FreeRTOSConfig.h.

Code

1) Sending and Receiving only integer Data

https://github.com/engrhamzaaliimran/FreeRTOSArduino/blob/2023/Source/Queue.cpp

2) Sending and Receiving Structure Data (DHT11 like example)

https://github.com/engrhamzaaliimran/FreeRTOSArduino/blob/master/Source/QueueStructure.cpp