FreeRTOS queues & mutexes

ese3025

queues & mutex API features

- we'll summarize the API functions here for convenience
- note also, for embedded systems:
  - *queues are especially useful for buffering data*
    - for example, data generated by a high-speed ADC cannot always be processed immediately by a multi-tasking CPU; the data should be placed into a buffer (queue) which retains a chunk of the data in memory until it can be handled by the CPU (either via an Interrupt Service Routine or RTOS task)
  - *mutexes are also known as a "binary semaphores"*
    - they are so-called because they provide "mutual exclusion" : only one RTOS task can access a resource at a time
    - to understand a mutex, think of a bathroom key at a gas station: only one customer at a time may use the bathroom; the person in the bathroom holds the mutex (in this case, the key!);

FreeRTOS Queue API functions (vary port to port…)

- QueueHandle_t **xQueueCreate**(portBaseType_t uxQueueLength, portBaseType_t uxItemSize);
- BaseType_t **xQueueSend(**QueueHandle_t xQueue, const void *pvItemtoQueue, TickType_t xTicksToWait);
- BaseType_t **xQueueReceive**(QueueHandle_t xQueue, const void *pvBuffer, TickType_t xTicksToWait);
- uBaseType_t **uxQueueMessagesWaiting**(QueueHandle_t xQueue);

Example

```
/* Define the data type that will be queued. */
typedef struct A_Message
{
    char ucMessageID;
    char ucData[ 20 ];
} AMessage;

/* Define the queue parameters. */
#define QUEUE_LENGTH 5
#define QUEUE_ITEM_SIZE sizeof( AMessage )

int main( void )
{
QueueHandle_t xQueue;

    /* Create the queue, storing the returned handle in the xQueue variable. */
    xQueue = xQueueCreate( QUEUE_LENGTH, QUEUE_ITEM_SIZE );
    if( xQueue == NULL )
    {
        /* The queue could not be created. */
    }

    /* Rest of code goes here. */
}
```

Listing 109 Example use of xQueueCreate()

Example

```
/* Define the data type that will be queued
typedef struct A_Message

    char ucMessageID;
    char ucData[ 20 ];
} AMessage;

/* Define the queue parameters. */
#define QUEUE_LENGTH 5
#define QUEUE_ITEM_SIZE sizeof( AMessage )

int main( void )
{
QueueHandle_t xQueue;

    /* Create the queue, storing the returned handle in the xQueue variable. */
    xQueue = xQueueCreate( QUEUE_LENGTH, QUEUE_ITEM_SIZE );
    if( xQueue == NULL )
    {
        /* The queue could not be created - do something. */
    }

    /* Create a task, passing in                eue handle as the task parameter. */
    xTaskCreate( vAnotherTask,
                 "Task",
                 STACK_SIZE,
                 ( void * ) xQueue, /* xQueue is used as the task parameter. */
                 TASK_PRIORITY,
                 NULL );

    /* Start the task executing. */
    vTaskStartScheduler();

    /* Execution will only reach here if there was not enough FreeRTOS heap memory
    remaining for the idle task to be created. */
    for( ;; );
}

void vATask( void *pvParameters )
{
QueueHandle_t xQueue;
AMessage xMessage;

    /* The queue handle is passed into this task as the task parameter.  Cast
    the parameter back to a queue handle. */
    xQueue = ( QueueHandle_t ) pvParameters;

    for( ;; )
    {
        /* Create a message to send on the queue.
        xMessage.ucMessageID = SEND_EXAMPLE;

        /* Send the message to the queue, waiting for 10 ticks for space to become
        available if the queue is already full. */
        if( xQueueSendToBack( xQueue, &xMessage, 10 ) != pdPASS )
        {
            /* Data could not be sent to the queue even after waiting 10 ticks. */
        }
    }
}
```

Example

```
/* Define the data type that will be queued. */
typedef struct A_Message
{
    char ucMessageID;
    char ucData[ 20 ];
} AMessage;

/* Define the queue parameters. */
#define QUEUE_LENGTH 5
#define QUEUE_ITEM_SIZE sizeof( AMessage )

int main( void )
{
QueueHandle_t xQueue;

    /* Create the queue, storing the returned handle in the xQueue variable. */
    xQueue = xQueueCreate( QUEUE_LENGTH, QUEUE_ITEM_SIZE );
    if( xQueue == NULL )
    {
        /* The queue could not be created - do something. */
    }

    /* Create a task, passing in the queue handle as the task parameter. */
    xTaskCreate( vAnotherTask,
                 "Task",
                 STACK_SIZE,
                 ( void * ) xQueue, /* The queue handle is used as the task parameter. */
                 TASK_PRIORITY,
                 NULL );

    /* Start the task executing. */
    vTaskStartScheduler();

    /* Execution will only reach here if there was not enough FreeRTOS heap memory
    remaining for the idle task to be created. */
    for( ;; );
}

void vAnotherTask( void *pvParameters )
{
QueueHandle_t xQueue;
AMessage xMessage;

    /* The queue handle is passed into this task as the task parameter.  Cast the
    void * parameter back to a queue handle. */
    xQueue = ( QueueHandle_t ) pvParameters;

    for( ;; )
    {
        /* Wait for the maximum period for data to become available on the queue.
        The period will be indefinite if INCLUDE_vTaskSuspend is set to 1 in
        FreeRTOSConfig.h. */
        if( xQueueReceive( xQueue, &xMessage, portMAX_DELAY ) != pdPASS )
        {
            /* Nothing was received from the queue - even after blocking to wait
            for data to arrive. */
        }
        else
        {
            /* xMessage now contains the received data. */
        }
    }
}
```

Listing 131 Example use of xQueueReceive()

FreeRTOS Mutex APIs

- need #include “semphr.h”
- SemaphoreHandle_t **xSemaphoreCreateMutex**(void);

```
SemaphoreHandle_t xSemaphore;
void vATask( void * pvParameters )
{
	/* Attempt to create a mutex type semaphore. */
	xSemaphore = xSemaphoreCreateMutex();
	if( xSemaphore == NULL )
	{
		/* There was insufficient heap memory available for the mutex to be
		created. */
	}
	else
	{
		/* The mutex can now be used. The handle of the created mutex will be
		stored in the xSemaphore variable. */
	}
}
```

mutex API (cont’d)

- BaseType_t **xSemaphoreTake**(SemaphoreHandle_t xSemaphore, TickType_t xTicksToWait);
- BaseType_t **xSemphoreGive**(SemaphoreHandle_t xSemaphore);
- from page 246 in FreeRTOS ref manual:

Example

```
SemaphoreHandle_t xSemaphore = NULL;

/* A task that creates a mutex type semaphore. */
void vATask( void * pvParameters )
{
    /* A semaphore is going to be used to guard a shared resource.  In this case
    a mutex type semaphore is created because it includes priority inheritance
    functionality. */
    xSemaphore = xSemaphoreCreateMutex();

    /* The rest of the task code goes here. */
    for( ;; )
    {
        /* ... */
    }
}

/* A task that uses the mutex. */
void vAnotherTask( void * pvParameters )
{
    for( ;; )
    {
        /* ... Do other things. */

        if( xSemaphore != NULL )
        {
            /* See if the mutex can be obtained.  If the mutex is not available
            wait 10 ticks to see if it becomes free. */
            if( xSemaphoreTake( xSemaphore, 10 ) == pdTRUE )
            {
                /* The mutex was successfully obtained so the shared resource can be
                accessed safely. */

                /* ... */

                /* Access to the shared resource is complete, so the mutex is
                returned. */
                xSemaphoreGive( xSemaphore );
            }
            else
            {
                /* The mutex could not be obtained even after waiting 10 ticks, so
                the shared resource cannot be accessed. */
            }
        }
    }
}
```

Listing 174 Example use of xSemaphoreTake()