Carrera de Especialización en Sistemas Embebidos

Sistemas Operativos en Tiempo Real

Clases 6 y 7: Prácticas de integración en FreeRTOS

ACSE
Asociación Civil para la Investigación, Promoción y Desarrollo de los Sistemas Electrónicos Embebidos
FACULTAD DE INGENIERIA
Universidad de Buenos Aires
CIM
CADIEEL

Resumen: FreeRTOS

- Como pudimos ver a lo largo de la materia, FreeRTOS nos ofrece las siguientes herramientas.
  - Gestión del uso de procesador y memoria para varias tareas en ejecución.
  - Manejo de temporización.
  - Sincronización de tareas y acceso recurrente a recursos.
  - Comunicación entre tareas.
  - Interacción ordenada de rutinas de interrupción con tareas en ejecución.

Resumen: API

- Y conocimos distintas APIs para el uso de estas herramientas:
  - Creación de tareas e inicio del OS (task. h):
    - `void taskFunction(void *pvParameters); //Prototipo de tarea`
    - `xTaskCreate(tareaA, (signed char *) "descripcion", configMINIMAL_STACK_SIZE, NULL, (tskIDLE_PRIORITY + 1UL), (xTaskHandle *) NULL); //Creación de tarea (en el contexto del OS)`
    - `void vTaskDelete( TaskHandle_t xTask ); //Eliminación de tarea`
    - `void vTaskStartScheduler(); //Inicio del OS (Entrega del CPU al OS)`
  - Temporización
    - `void vTaskDelay( const TickType_t xTicksToDelay ); //Retardo`
    - `void vTaskDelayUntil( TickType_t *pxPreviousWakeTime, const TickType_t xTimeIncrement ); //Hacer periódica una tarea`

Resumen: API

- Sincronización de tareas y acceso concurrente a recursos (semphr.h)
  - `SemaphoreHandle_t xSemaphoreCreateBinary(); //Creación de semáforo binario`
  - `SemaphoreHandle_t xSemaphoreCreateCounting( UBaseType_t uxMaxCount, UBaseType_t uxInitialCount); //Creación de un semáforo contador`
  - `SemaphoreHandle_t xSemaphoreCreateMutex( void ); //Creación de mutex`
  - `xSemaphoreTake( SemaphoreHandle_t xSemaphore, TickType_t xTicksToWait ); //Tomar semáforo o mutex`
  - `xSemaphoreGive( SemaphoreHandle_t xSemaphore ); //Liberar sem o mutex`

- Comunicación entre tareas (queue. h)
  - `QueueHandle_t xQueueCreate( UBaseType_t uxQueueLength, UBaseType_t uxItemSize ); //Creación de una cola de mensajes`
  - `xQueueSend( QueueHandle_t xQueue, const void * pvItemToQueue, TickType_t xTicksToWait ); //Depositar un elemento`

Resumen: API

- Comunicación entre tareas (Continuación)
  - **xQueueReceive**(QueueHandle_t xQueue, void *pvBuffer, TickType_t xTicksToWait); //Extraer un elemento
  - **xQueuePeek**(QueueHandle_t xQueue, void *pvBuffer, TickType_t xTicksToWait); //Leer un elemento sin retirarlo
  - UBaseType_t **uxQueueMessagesWaiting**(QueueHandle_t xQueue); /* Consultar cuántos mensajes hay en la cola */
  - UBaseType_t **uxQueueSpacesAvailable**(QueueHandle_t xQueue); /* Consultar cuántos espacios disponibles hay en la cola */
  - UBaseType_t **uxQueueSpacesAvailable**(QueueHandle_t xQueue); /* Consultar cuántos espacios disponibles hay en la cola */
  - void **vQueueDelete**(QueueHandle_t xQueue); //Eliminar la cola
  - BaseType_t **xQueueReset**(QueueHandle_t xQueue); //Limpiar la cola

Resumen: API

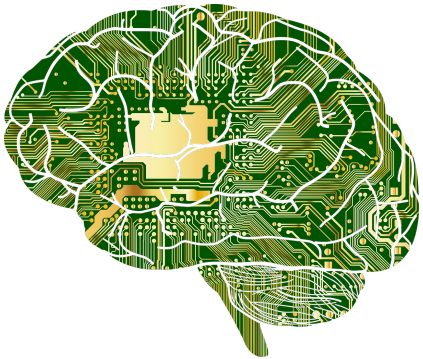
- Además, aprendimos a utilizar estas APIs en handlers de interrupción
  - Utilizando la estructura:
    - `static signed BaseType_t xHigherPriorityTaskWoken= pdFALSE;`
    - `//Contenido del handler, incluidas APIs_FROM_ISR`
    - **`portYIELD_FROM_ISR`**`( xHigherPriorityTaskWoken );`
  - Algunas APIs que soportan el uso en ISR son:
    - **`xSemaphoreTakeFromISR`** `( SemaphoreHandle_t xSemaphore, signed BaseType_t *pxHigherPriorityTaskWoken );`
    - **`xSemaphoreGiveFromISR`**`( SemaphoreHandle_t xSemaphore, signed BaseType_t *pxHigherPriorityTaskWoken );`
    - **`xQueueSendFromISR`**`(QueueHandle_t xQueue, const void *pvItemToQueue, BaseType_t *pxHigherPriorityTaskWoken );`
    - **`xQueueReceiveFromISR`**`( QueueHandle_t xQueue, void *pvBuffer, BaseType_t *pxHigherPriorityTaskWoken );`

Ejercicio #1 - Colas e ISR

- En la clase anterior quedamos con el siguiente ejercicio:
  - Tarea A:
    - Deberá gestionar la medición de una tecla de la EDUCIAA, pero deberá funcionar utilizando interrupciones.
    - No deberá ser periódica, sino que disparada por eventos.
    - La medición deberá realizarse usando algoritmo antirebote.
  - Tarea B:
    - Periódicamente, deberá encender y apagar un led con un período del doble de tiempo del último tiempo de pulsación medido (ciclo de trabajo 50%)
  - ISR
    - Deberá enviar una señal a la tarea A a través de una cola. La información de cada elemento de la cola será:
      - Indice de tecla
      - Valor de TickCount al momento del evento
      - Flanco de ocurrencia del evento.

Ejercicio #1 - Colas e ISR

Pensemos cómo afrontarlo



- ¿Cual esperamos que sea el comportamiento de los leds en cada ocasión?
- ¿Cómo podemos hacer para obtener y enviar a la tarea A, los datos solicitados?
- ¿Qué alternativas tenemos para aplicar antirrebote por software con lectura por interrupción?

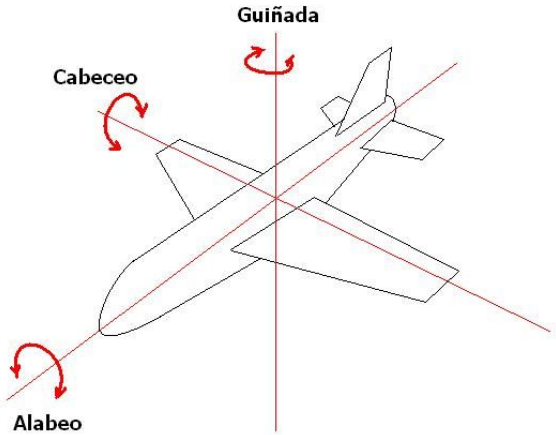
Ejercicio #2 - Integrador

Basados en el ejercicio anterior

- **Mostremos en los leds B, 1, 2, 3 respectivamente, los tiempos medidos en las teclas 1,2,3,4. ¿Qué precauciones debo tomar?**
- **Imprimamos en UART cada vez que ocurre un flanco en una tecla, que una medición sea tomada y que enciende o apaga un led. ¿Qué alternativas tengo para hacerlo?**

Ejercicio #3 - Monitor de navegación

- Supongamos que estamos en el desarrollo de una aeronave que tendrá control automático de altitud (cabeceo) que no permite otras maniobras durante su operación, un control automático de guiñada y un sistema que nos permite controlar el alabeo con las teclas 2 y 3 de la EDU-CIAA.



Ejercicio #3 - Monitor de navegación

Nos encargan implementar sobre la EDU-CIAA, con apoyo del módulo MPU9250, un prototipo de monitor de navegación para maniobras que haga lo siguiente:

- Al hacer cualquier movimiento de alabeo, deberá medir la inclinación de la aeronave durante 3 segundos. Si la inclinación no supera los 45º en ningún momento, parpadeará el led G, si es igual o mayor a 45º parpadeará el led R. En ambos casos, el parpadeo durará 1 segundo.

- Monitoreará constantemente la temperatura de cabina: si es mayor a 30º informará al piloto, dejando encendido el led 4 hasta que baje la misma.

- Tendrá un sistema de Log que permitirá recibir mensajes de distintos módulos de SW y los imprimirá por UART con el formato [modulo],[valor]

Bibliografia

- RTOS 1 - Clase 1 - Introducción a los RTOS, Franco Bucafusco, 2018
- RTOS 1 - Clase 3 - Sincronización entre tareas (Partes 1 y 2), Franco Bucafusco, 2018
- RTOS 1 - Clase 5 - Colas, Franco Bucafusco, 2018
- RTOS 1 - Clase 5 - Gestión de interrupciones, Franco Bucafusco, 2018
- RTOS 1 - Clase 5 - Práctica: ISR y Colas, Franco Bucafusco, 2018
- https://www.freertos.org/a00019.html - Revisado el 2018-08-04
- https://www.freertos.org/a00112.html - Revisado el 2018-08-04
- https://www.freertos.org/a00018.html - Revisado el 2018-08-04
- https://www.freertos.org/a00113.html - Revisado el 2018-08-04