Sistemas Embebidos II - IMT 322

Laboratorio 2: Recursos compartidos

Deadline: 20/03/2023

Introducción

En este laboratorio, se crea un entorno simulado basado en FreeRTOS. A partir de aquí, se explora algunos de los conceptos y algoritmos presentados durante las clases.

Entorno

Con la ayuda de RTOS, es posible al menos crear una ïlusión"de que las tareas se ejecutan en paralelo.

Analice el código del laboratorio del proyecto Keil uvision. Se implementara un código de tal forma que dos tareas se ejecuten en conjunto. Los LED azul y verde indican si la tarea 1 o la tarea 2 se están ejecutando respectivamente.

Verifique que la configuración para FreeRTOS sea correcta; en particular, asegúrese de que el algoritmo de programación sea Round Robin (ya que proporciona el .^{en}torno"de multiprocesamiento).

Conceptos del Lab

- FreeRTOS
- Cronogramas
- Queue
- Comunicacion
- Threads
- Tasks

FreeRTOS

En FreeRTOS, las tareas pueden ser creadas usando la función xTaskCreate(). La declaración se muestra en el Codigo 1. Como puede ver esta función tiene varios parámetros. Revisemos que quiere decir cada uno de estos.

```
1  BaseType_t xTaskCreate( TaskFunction_t pvTaskCode,      // Pointer to the task function
2                          const char * const pcName,      // Descriptive name of the task
3                          uint16_t usStackDepth,     // Stack size allocated for the task
4                           void *pvParameters ,          // Task specific parameters
5                           UBaseType_t uxPriority ,      // Priority of the task
6                           TaskHandle_t *pxCreatedTask     // Handle to the task
7                        );
```

Codigo 1: Declaracion de xTaskCreate()

- pvTaskCode - este argumento apunta a una función que contiene la lógica de la tarea. Esta función nunca debe terminarse. Por lo tanto, la funcionalidad de la tarea debe encapsularse en un bucle infinito. El Código 2 muestra un ejemplo de una función de tarea de este tipo.

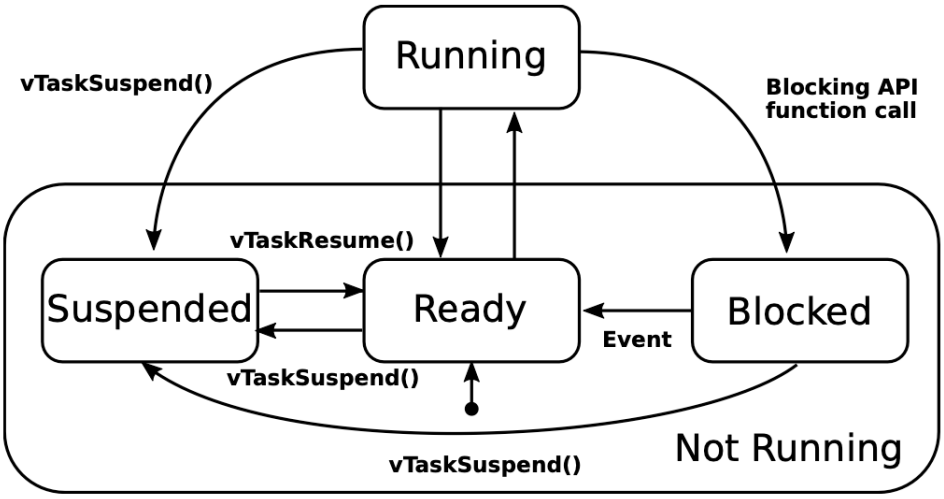

Imagen 1: Estados de una tarea en FreeRTOS

- pcName - este argumento debe elegirse para que sea un nombre descriptivo para la tarea. FreeRTOS no utiliza este nombre, pero facilita el proceso de depuración.
- usStackDepth - para cada tarea, se preasignará un espacio de memoria. El programador debe especificar el tamaño del stack según los requisitos de memoria de la tarea. En este laboratorio, configuramos usStackDepth en un valor predeterminado de configMINIMAL_STACK_SIZE. En práctica, el tamaño del stack requerido se puede determinar con las herramientas de FreeRTOS y luego se puede configurar en un valor razonablemente pequeño.
- pvParameters - este argumento permite a los usuarios pasar información adicional a la función de la tarea para la inicialización, como parámetros que describen un comportamiento más detallado de la lógica de la tarea. Para un uso genérico, esto se logra mediante un puntero. Por lo tanto, cualquier argumento debe pasarse a través de un puntero a un objeto vacío.
- uxPriority - este argumento se puede utilizar para asignar una prioridad a la tarea. El papel de las prioridades se explicará en el Ejercicio.
- pxCreatedTask - este argumento permite al usuario recibir un identificador de la tarea creada. Algunas funciones de FreeRTOS requieren un identificador para realizar operaciones en una tarea específica durante el tiempo de ejecución, por ejemplo, para cambiar la prioridad de una tarea o eliminar una tarea.

Importante: Note que para este laboratorio se obviara el loop infinito. Esto es solo con fines demostrativos para los conceptos explorados en clases.

```
1  void vTaskFunction(void *pvParameters)
2  {
3    // Some code here
4
5  }
```

Codigo 2: Declaracion funcion del task

Setup: FreeRTOSConfig

1. Para este laboratorio asegurese de tener funcionando FreeRTOS en su plataforma
2. Descargue el proyecto ejemplo y abra a FreeRTOSConfig.h
3. Configure el cronograma con Time Slicing y que este sea preventivo
4. En el macro configTICK_RATE_HZ con un valor de 5
5. Configure tres pines como salida (a su eleccion). Esto debe realizarlo en la funcion setup_GPIO y setup_RCC. Note que puede usar bare metal programming o alguna capa de abstraccion como HAL o LL.
6. Una vez haya configurado estos pines como salida conectelos de la siguiente forma, por ejemplo:
   - LED Azul - PA.2 - El Task 1 se hara cargo de este LED. Lo prendera y apagara. El Task 2 no tiene acceso a este recurso. Modifique el Task 1 de tal forma que este prenda y apague los LEDs.
   - LED Verde - PA.3 - El Task 2 se hara cargo de este LED. Lo prendera y apagara. El Task 1 no tiene acceso a este recurso. Modifique el Task 1 de tal forma que este prenda y apague los LEDs.

Filas

Las Filas proporcionan una manera fácil de comunicarse entre tareas y se realizan como búferes de tipo primero en entrar, primero en salir (FIFO) en FreeRTOS. Cada cola puede contener como máximo un número predefinido de elementos de datos. Estos elementos pueden ser de cualquier tamaño, siempre que el tamaño del elemento sea fijo antes de la creación de la cola. En FreeRTOS, las colas se pueden crear con la función xQueueCreate(), donde se debe definir el número de elementos (uxQueueLength) y el tamaño del elemento (uxItemSize). La función devuelve un identificador de la cola que debe usarse para colocar elementos en la cola con xQueueSendToBack() y tomar elementos de la cola con xQueueReceive(). El identificador debe estar disponible para todas las tareas que deseen utilizar la cola.

El argumento xTicksToWait de xQueueSendToBack() y xQueueReceive() especifican un tiempo de bloque. Cuando una tarea intenta leer de una cola vacía, se mantendrá en estado Bloqueado hasta que otra tarea escriba en la cola o expire el tiempo de bloqueo. De manera similar, cuando la cola está llena y una tarea intenta escribir en la cola, se mantendrá en el estado Bloqueado hasta que otra tarea lea de la cola o expire el tiempo de bloqueo. En caso de tiempo de espera, se devolverá un código de error (pdFALSE). Cuando ocurra cualquiera de estos eventos, la tarea se moverá automáticamente al estado Listo.

```
1  QueueHandle_t xQueueCreate( UBaseType_t uxQueueLength,    // Numero de items en la fila
2                              UBaseType_t uxItemSize);  // Tamano de items en bytes
3
4  BaseType_t xQueueSendToBack( QueueHandle_t xQueue,        // Queue Handle
5                               const void *pvItemToQueue,      // Pointer al item que se
       mandara
6                               TickType_t xTicksToWait.          // Ticks hasta que se mueva
       al estado ready
7                               );
8
9  BaseType_t xQueueReceive( QueueHandle_t Queue,       // Queue handle
10                              void *const pvBuffer,  // Copia del item que se recibira
11                              TickType_t xTicksToWait       // Ticks hasta que se mueve al
       estado Ready
12                              );
```

Codigo 3: Manejo de filas (Revisar 'FreeRTOS Referenca Chapter 3.3, 3.16')

Ejercicio

Puede usar el proyecto anterior como base para este ejercicio.

Escriba un programa que tenga dos tareas. Ambas tareas deben declarar una variable local uint32_t. La primera tarea (SquareTask) debe inicialiar la variable local con un valor de 4. La segunda tarea (DecrementTask) debe inicializar la con un valor de 0. Las tareas deben seguir la logica mostrada en la Imagen 2.

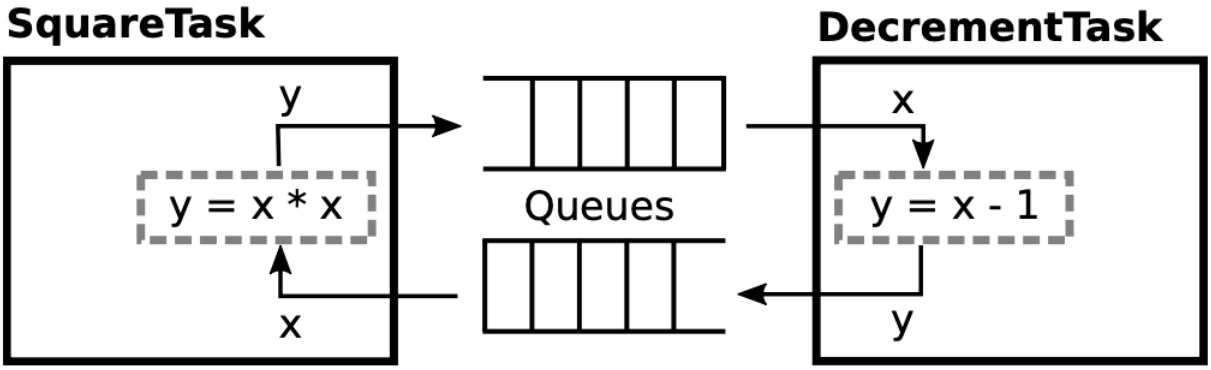

Imagen 2: Aplicacion

La rutina de SquareTask debera:

1. Comunicar los valores de su variable local a la tarea DecrementTask usando una queue.
2. Esperar recibir un valor.
3. Elevar al cuadrado el valor recibido y actualizar la variable local con el resultado de esta operación.
4. Repetir desde (1)

La rutina de DecrementTask debera:

1. Esperar por el valor al ser recibido
2. Disminuir el valor recibido en uno y actualizar su variable local con el resultado de esta operación
3. Comunicar el valor de su variable local a la tarea SquareTask usando una fila
4. Repetir desde (1)

Cada tarea debera realizar lo siguiente después de recibir el valor:

- Prender su LED correspondiente, si es posible use un canal UART para imprimir el valor correspondiente de la variable, en caso de no serlo intente utilizar el debugger.
- Verificar si el valor excede los 10 000. Si este es el caso la tarea deberá eliminar se a si misma.

El objetivo general de este ejercicio es crear la funcionalidad mencionada anteriormente sin utilizar variables globales. Para simplificar la depuración, puede subdividir el ejercicio en dos pasos:

- Comenzar implementando los handles de las filas requeridas como variables globales. Luego, implementar las rutinas de las tareas y probar si la salida del UART concuerda con la funcionalidad requerida, en caso de utilizar UART. Si no utiliza UART verificar que los LEDs correspondientes se prendan y se apaguen cuando correspondan.
- Evite las variables globales pasando los handles de cola como parámetros de función. Verifique que su solución aún funcione.