Usando el Kernel de Tiempo Real FreeRTOS:

(Traducción de la Guía práctica por Richard Barry)

Capítulo 6: *Solución de Problemas*

1.1 Introducción y Alcance de este capítulo:

En este capítulo se pretende poner de manifiesto los problemas más comunes que encuentran los usuarios que son nuevos en FreeRTOS. Se centra principalmente en el desbordamiento de pila y la detección de desbordamiento de pila, dado que éstos han demostrado ser la fuente más frecuente de las solicitudes de apoyo a través de los años. A continuación, brevemente y en un estilo de preguntas más frecuentes, se mencionan otros errores comunes, sus posibles causas y sus soluciones.

printf-stdarg.c

El uso de la pila puede llegar particularmente alto, cuando se utilizan las funciones de la libreria estándar C, especialmente IO y funciones de manejo de cadenas tales como sprintf (). La descarga de FreeRTOS incluye un archivo llamado printf-stdarg.c que contiene una versión mínima y con un uso eficiente de la pila de sprintf(), que se puede utilizar en lugar de la versión de la librería estándar. En la mayoría de los casos esto permitirá una pila mucho más pequeña asignada a cada tarea que llama a sprintf () y funciones relacionadas.

Printf-stdarg.c es de código abierto, pero propiedad de un tercero por lo que se licencia por separado de FreeRTOS. Los términos **de licencia se encuentran en la parte superior del archivo de origen.**

1.2 Desbordamiento de pila:

FreeRTOS ofrece varias características para ayudar a los temas relacionados con la pila y depuración.

Función API uxTaskGetStackHighWaterMark():

Cada tarea mantiene su propia pila, el tamaño total de la que se especifica cuando la tarea es creada. uxTaskGetStackHighWaterMark () se utiliza para consultar cuán cerca una tarea está de desbordar el espacio de pila que le sea asignado. Este valor se denomina "cota máxima" de la pila.

```
unsigned portBASE_TYPE uxTaskGetStackHighWaterMark( xTaskHandle xTask );
```

Listado 75 – Prototipo de la función API uxTaskGetStackHighWaterMark().

Tabla 20 Parámetros y valor de retorno de uxTaskGetStackHighWaterMark().

Parámetro y valor de retorno	Descripción
xTask	El manipulador de la tarea cuya cota máxima está siendo consultada (la tarea sujeto). Ver el parámetro pxCreatedTask de la función API xTaskCreate () para obtener información sobre la obtención de los manipuladores de las tareas. Una tarea puede consultar su cota máxima de pila pasando NULL en lugar de un identificador de tarea válido.
Valor de Retorno	La cantidad de pila que la tarea en realidad está utilizando, va a crecer y reducirse dependiendo si la tarea se ejecuta o las interrupciones son procesadas. uxTaskGetStackHighWaterMark() devuelve la cantidad mínima de espacio de pila restante que estuvo disponible desde que la tarea inició la ejecución. Esto es la cantidad de pila que no fue utiliza cuando el uso de la pila estaba en su máximo valor. Cuanto más cerca la cota máxima esta del valor 0, más cerca estuvo la tarea de desbordar su pila.

Verificación del tiempo de ejecución de pila. Información general:

FreeRTOS incluye dos mecanismos opcionales de verificación de tiempo de ejecución de la pila. Estos son controlados por la constante configCHECK_FOR_STACK_OVERFLOW dentro de FreeRTOSConfig.h. Ambos métodos incrementarán el tiempo que se tarda en realizar un cambio de contexto.

El enlace de desbordamiento de pila (o de devolución de llamada) es una función que es llamada por el kernel cuando detecta un desbordamiento de pila. Para utilizar una función de enlace de desbordamiento de pila:

- Establecer configCHECK_FOR_STACK_OVERFLOW en 1 ó 2 en FreeRTOSConfig.h
- Proporciona la implementación de la función de enlace, utilizando el nombre de la función exacta y el prototipo mostrado en el Listado 76.

```
void vApplicationStackOverflowHook( xTaskHandle *pxTask, signed portCHAR *pcTaskName );
```

Listado 76 – Prototipo de la función de enlace de desbordamiento de pila.

El enlace de desbordamiento de pila se proporciona para encontrar y depurar errores de pila más fácil, pero no hay forma de recuperarse de un desbordamiento de pila una vez que ha ocurrido. Los parámetros pasan el identificador y el nombre de la tarea cuya pila ha desbordado en la función de enlace, aunque es posible que el desbordamiento haya corrompido el nombre de la tarea.

El enlace de desbordamiento de pila puede ser llamado desde el contexto de una interrupción.

Algunos microcontroladores generarán una excepción de falta cuando detectan un acceso de memoria incorrecta y es posible que una falta se active antes que el kernel tenga la oportunidad de llamar a la función de enlace de desbordamiento.

Verificación del tiempo de ejecución de pila. Método 1:

El método 1 se selecciona cuando configCHECK_FOR_STACK_OVERFLOW se establece en 1.
El contexto de ejecución entero de una tarea se guarda en su pila cada vez que sale del estado de ejecución. Es probable que este sea el tiempo en el que el uso de la pila alcanza su pico. Cuando configCHECK_FOR_STACK_OVERFLOW se establece en 1, el kernel verificará que el puntero de pila permanece dentro del espacio de pila válida, después de que el contexto se ha guardado. El enlace de desbordamiento de pila se llama si el puntero de pila se encuentra fuera de su rango válido.
El método 1 es rápido de ejecutar, pero se pueden perder desbordamientos de pila que se produzcan entre los cambios de contexto.

Verificación del tiempo de ejecución de pila. Método 2:

El método 2 realiza comprobaciones adicionales a los ya descriptos para el método 1. Se selecciona cuando configCHECK_FOR_STACK_OVERFLOW está establecido en 2.

Cuando se crea una tarea su pila se llena con un patrón conocido. El método 2 camina los últimos 20 bytes válidos del espacio de pila de tareas para comprobar que este patrón no se ha sobrescrito. La función de enlace de desbordamiento de pila se llama si alguno de los 20 bytes han cambiado su valor.
El método 2 no es tan rápido como el método 1 pero sigue siendo relativamente rápido, ya que sólo 20 bytes se ponen a prueba. Es muy probable notar todos los desbordamientos de pila, aunque es concebible (pero altamente improbable) de que algunos todavía se puede perder.

1.3 Otras Fuentes Comunes de error:

Síntoma: Agregar una simple tarea a una demostración causa que la aplicación deje de funcionar.

La creación de una tarea requiere memoria que se obtiene del bloque de memoria disponible. Muchos de los proyectos de demostración dimensionan el tamaño del bloque de memoria para ser exactamente lo suficientemente grande como para crear las tareas de demostración, por lo que después, cuando se crean las tareas adicionales, colas o semáforos, la memoria restante será insuficiente.
La tarea ociosa se crea automáticamente cuando se llama a vTaskStartScheduler(). Esta función sólo devolverá si no hay suficiente memoria restante en el bloque para que se cree la tarea ociosa. Incluir un bucle nulo [for (;;);] después de la llamada a vTaskStartScheduler() puede hacer que este error sea más fácil de depurar.
Para poder añadir más tareas o bien se puede aumentar el tamaño del bloque o eliminar algunas de las tareas de demostración existentes.

Síntoma: Usar una función API dentro de una interrupción causa que la aplicación deje de funcionar.

No use funciones API dentro de rutinas de servicio de interrupción a menos que el nombre de la función API termine en "…FromISR()".

Síntoma: A veces la aplicación deja de funcionar durante una rutina de servicio de interrupción.

Lo primero a comprobar es que la interrupción no esté causando un desbordamiento de pila.
La forma en que las interrupciones se definen y utilizan difiere entre puertos y entre los compiladores, por lo que la segunda cosa a comprobar es que la sintaxis, macros y convenciones de llamada utilizados en la rutina de servicio de interrupción sean exactamente como se describe en la página de documentación de la demostración, y exactamente como se demuestra en otras rutinas de servicio de interrupción.

Si la aplicación se ejecuta en un Cortex M3 entonces garantizar la prioridad asignada a cada interrupción teniendo en cuenta que los números bajos se usan para representar interrupciones lógicamente de alta prioridad. Esto puede parecer contra-intuitivo. Es un error común asignar accidentalmente una interrupción que utiliza una función API de FreeRTOS una prioridad que está por encima de la definida por configMAX_SYSCALL_INTERRUPT_PRIORITY**.**

Síntoma: El Scheduler deja de funcionar cuando intenta iniciar la primera tarea.

Si un microcontrolador ARM7 se utiliza, aegurarse que el procesador está en el modo de supervisor antes de realizar el llamado a vTaskStartScheduler(). La forma más sencilla de lograr esto es colocar el procesador en el modo Supervisor dentro del código de inicio C antes de llamar al main(). Así es como se configuran las aplicaciones ARM7 de demostración.

El scheduler no se iniciará a menos que el procesador esté en el modo Supervisor.

Síntoma: Las secciones críticas no se anidan correctamente.

No altere los bits de habilitación de interrupciones o los flags de prioridad del microcontrolador utilizando cualquier método que no sea las llamadas a taskENTER_CRITICAL() y taskEXIT_CRITICAL(). Estas macros mantienen una cuenta de la profundidad de anidamiento de las llamadas, para asegurar que las interrupciones sólo se habiliten de nuevo cuando la anidación ha vuelto completamente a cero.

Síntoma: La aplicación deja de funcionar incluso antes de iniciar el Scheduler.

Una rutina de servicio de interrupción que potencialmente podría causar un cambio de contexto no debe permitirse que se ejecute antes que se ha iniciado el planificador. Lo mismo es cierto para cualquier

rutina de servicio de interrupción que intenta enviar o recibir de una cola o semáforos. Un cambio de contexto no puede ocurrir hasta después que se haya iniciado el planificador.

Muchas funciones API no pueden ser llamadas antes de ser iniciado el planificador. Es mejor restringir el uso de las API para la creación de tareas, las colas y semáforos hasta después que haya sido llamada la función vTaskStartScheduler().

Síntoma: Llamados a funciones API mientras el scheduler está suspendido causa que la aplicación deje de funcionar.

El scheduler se suspende llamando a vTaskSuspendAll() y es reanudado llamando xTaskResumeAll(). No llame a funciones de la API, mientras que el programador esté suspendido.

Síntoma: El prototipo pxPortInitialiseStack() causa que la compilación falle.

Cada puerto requiere una macro para ser definido, que asegura que los archivos header correctos del kernel están incluidos en la compilación. Un error al compilar el prototipo xPortInitialiseStack () es casi sin duda un síntoma que esta macro se establece de forma incorrecta para el puerto que se utiliza. Ver ANEXO 4: para más información.

APENDICE 1: CREACIÓN DE LOS EJEMPLOS

Este libro presenta numerosos ejemplos, el código fuente se proporciona en un archivo .zip adjunto, junto con los archivos de proyecto, que pueden ser abiertos y construidos dentro del Open Watcom IDE.
Los ejecutables resultantes pueden entonces ser ejecutados ya sea dentro de un terminal de comandos de Windows o alternativamente bajo el emulador DOSBox DOS. Ver http://www.openwatcom.org y http://www.dosbox.com para descargas de herramientas.
Asegúrese de incluir las opciones 16bit del DOS cuando se instale el compilador Open Watcom!
Los archivos de proyecto de Open Watcom están llamados RTOSDemo.wpj y pueden estar ubicados en los ejemplos \ directorios Example0nn, donde "nn" es el número de ejemplo.
El DOS está lejos de ser un objetivo ideal para FreeRTOS y las aplicaciones de ejemplo no funcionarán con verdaderas características de tiempo real. DOS se utiliza simplemente porque permite a los usuarios experimentar con los ejemplos sin tener que invertir en hardware o herramientas especiales.
Tenga en cuenta que el depurador Open Watcom permitirá que las interrupciones se ejecuten entre las operaciones de paso, incluso cuando avanzo paso a paso por el código que se encuentra dentro de una sección crítica. Esto, desafortunadamente, hace que sea imposible pasar por el proceso de cambio de contexto. Lo mejor es correr los ejecutables generados a partir de un símbolo del sistema en lugar de hacerlo desde el IDE de Open Watcom.

APENDICE 2: LAS APLICACIONES DE DEMOSTRACIÓN.

Cada puerto oficial de FreeRTOS viene con una aplicación de demostración que debe compilar sin que se generen errores o advertencias. La aplicación de demostración tiene varios propósitos:

1. Proporcionar un ejemplo de un proyecto preconfigurado de trabajo y con los archivos correctos incluidos y las opciones correctas del compilador establecidas.
2. Para permitir experimentación “fuera de la caja” con una configuración mínima o conocimiento previo.
3. Demostrar la API de FreeRTOS.
4. Como una base a partir del cual se pueden crear aplicaciones reales.

Cada proyecto de demostración se encuentra en un directorio único en el directorio Demo (ver Apéndice 3:). El nombre del directorio indicará a que puerto el proyecto de demostración se refiere.

Cada aplicación de demostración también viene con una página de documentación que está alojada en el sitio web FreeRTOS.org. La página de documentación incluye información sobre la localización de aplicaciones de demostración individuales en la estructura de directorios FreeRTOS.

Todos los proyectos de demostración crean tareas que se definen en los archivos de origen que se encuentran en el árbol de directorios demo \ Common. La mayoría de los archivos usan el directorio \Demo\Common\Minimal.

Un archivo llamado main.c se incluye en cada proyecto. Este contiene la función main(), desde donde se crean todas las tareas de la aplicación de demostración. Ver los comentarios dentro de los archivos main.c individuales para obtener más información sobre lo que hace una aplicación de demostración específica.

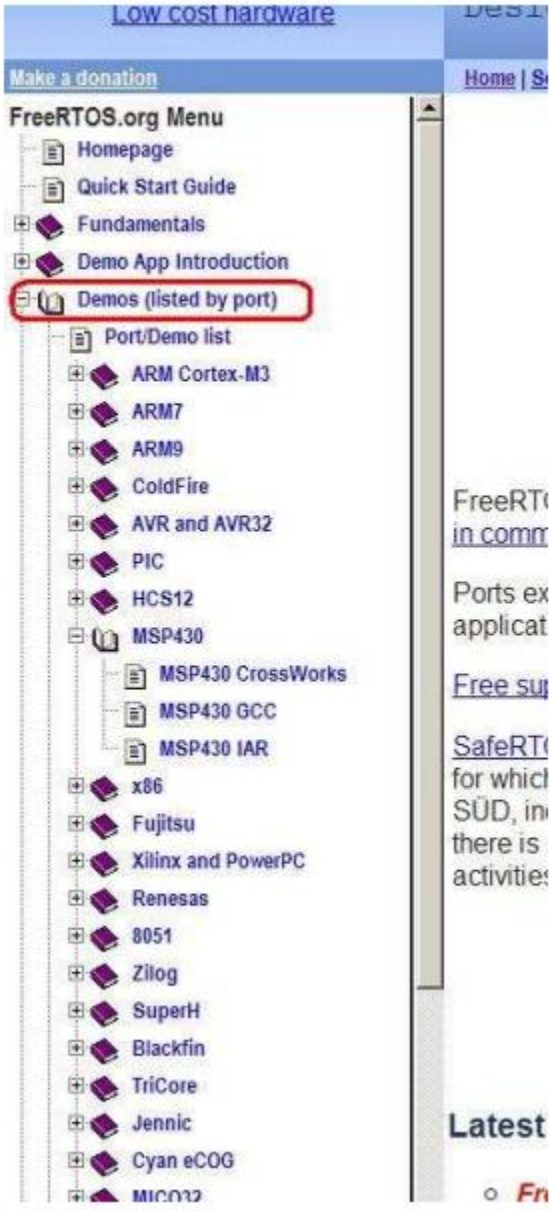

Figura 44 – Ubicando la aplicación de demostración dentro del cuadro de meno del sitio web de FreeRTOS.org

APENDICE 3: ARCHIVOS Y DIRECTORIOS DE FREERTOS.

La estructura de directorios que se describe en este apéndice se refiere únicamente al archivo .zip que se puede descargar desde el sitio WEB FreeRTOS.org. Los ejemplos que vienen con este libro utilizan una organización ligeramente diferente.
FreeRTOS se descarga como un único archivo .zip que contiene:

- El código fuente FreeRTOS. Este es el código que es común a todos los puertos.
- Una capa de puerto para cada microcontrolador y compilador que se proporciona.
- Un archivo de proyecto o makefile para construir una aplicación de demostración para cada combinación de microcontrolador y compilador que se admite.
- Un conjunto de tareas de demostración que son comunes a cada aplicación de demostración. Estas tareas de demostración son referenciadas desde los puertos especiíficos de los proyectos de demostración.

El archivo .zip tiene dos directorios de primer nivel, uno llamado Fuente y la otra llamada demostración. El árbol de directorios Fuente contiene toda la implementación del núcleo de FreeRTOS, tanto los componentes comunes como los componentes específicos del puerto. El árbol de directorios demo contiene sólo los archivos de proyecto de la aplicación de demostración y los archivos de origen que definen las tareas de demostración.

```
FreeRTOS
        ¦
        +-Demo          Contiene la fuente y los proyectos de las aplicaciones de demostración.
        ¦
        +-Source        Contiene la implementación del Kernel de tiempo real.
```

Figura45 – Directorios de Alto nivel – Fuente y demostración.

El código fuente del núcleo de FreeRTOS está contenido en sólo tres archivos de C que son comunes a todos los puertos del microcontrolador. Estos se llaman queue.c, tasks.c y list.c, y pueden estar ubicados directamente bajo el directorio de origen. Los archivos de puertos específicos se encuentran en el árbol de directorios portátil, que también se encuentra directamente en el directorio de origen.

Un cuarto archivo de origen opcional llamado croutine.c implementa la funcionalidad co-rutina de FreeRTOS. Sólo necesita ser incluido en la construcción si las co-rutinas van a ser utilizadas.

```
FreeRTOS
        ¦
        +-Demo                  Contiene la fuente y proyecto de la aplicación demo.
        ¦
        +-Source                Contiene la implementación del kernel de tiempo real.
                |
                +-tasks.c       Uno de los tres archivos núcleos del Kernel.
                +-queue.c       Uno de los tres archivos núcleos del Kernel.
                +-list.c        Uno de los tres archivos núcleos del Kernel.
```

+-portable El sub-directorio que contiene todos los archivos específicos de los puertos.

Figura 46 – Los tres archivos núcleo que implementan el Kernel de FreeRTOS.

Eliminación de archivos no utilizados:

El archivo principal de FreeRTOS .zip incluye los archivos para todos los puertos y todas las aplicaciones de demostración, por lo contiene muchos más archivos de los que se necesitan utilizar en cualquier puerto. El proyecto de aplicación de demostración o makefile que acompaña el puerto que se utiliza, se puede utilizar como referencia para saber que archivos se requieren y cuales se pueden eliminar.

La "capa portátil" es el código que adapta el kernel FreeRTOS a un compilador y arquitectura particular. Los archivos de origen de la capa portátil se encuentran dentro del directorio FreeRTOS\Source\portable\[compilador]\[arquitectura] , donde [compilador] es la cadena de herramientas que se utiliza y [arquitectura] es la variante microcontrolador que se utiliza.

- Todos los sub-directorios bajo FreeRTOS\Source\portátil que no se refieren a la cadena de herramientas que se utilizan, se pueden borrar excepto el directorio: FreeRTOS\Source\portable\MemMang.
- Todos los sub-directorios bajo FreeRTOS\Source\portable\[compilador] que no se refieren a la variante del microcontrolador se utilizan se pueden borrar.
- Todos los sub-directorios bajo FreeRTOS\demo que no se refieren a la aplicación de demostración que se utiliza se pueden borrar excepto FreeRTOS\demo\Common, que contiene los archivos que hacen referencia a todas las aplicaciones de la demostración.

FreeRTOS\demo\Common contiene muchos más archivos de los que se hace referencia desde cualquier aplicación de una demo por lo que este directorio también se puede borrar si se desea.

APENDICE 4: CREACIÓN DE UN PROYECTO DE FREERTOS.

Adaptando uno de los proyectos de demostración suministrados:

Cada puerto oficial FreeRTOS viene con una aplicación de demostración pre-configurada que debe compilar sin ningún error o advertencia (véase el Apéndice 2). Se recomienda que los nuevos proyectos se creen mediante la adaptación de uno de estos proyectos ya existentes. De esta manera el proyecto incluirá los archivos correctos y las opciones del compilador correctas establecidas. Para iniciar una nueva aplicación de un proyecto de demostración existente:

1. Abra el proyecto de demostración suministrado y asegúrese de que compila y se ejecuta como se esperaba.
2. Cualquier archivo que se encuentra dentro del árbol de directorios \Common de la demostración se puede quitar desde el archivo de proyecto o makefile.
3. Elimine todas las funciones dentro main.c que no sea prvSetupHardware().
4. Asegurar que configUSE_IDLE_HOOK, configUSE_TICK_HOOK y configCHECK_FOR_STACK_OVERFLOW estén seteadas en 0 dentro de FreeRTOSConfig.h. Esto evitará que el enlazador busque funciones de enlace. Las funciones de enlace se pueden agregar más tarde si es necesario.
5. Crear una nueva función main() de la plantilla mostrada en el Listado 77.
6. Compruebe que el proyecto aún se compila.

Siguiendo estos pasos se proporcionará un proyecto que incluye los archivos de origen FreeRTOS pero no define ninguna funcionalidad.

```
int main( void )
{
	/* Realize cualquier setup de hardware necesario. */
	prvSetupHardware();

	/* --- LAS TAREAS DE LA APLICACIÓN PUEDEN CREARSE AQUÍ --- */

	/* Incia el scheduler para que las tareas empiecen a ejecutarse. */
	vTaskStartScheduler();

	/* La ejecución solo alcanzará este punto si no había suficiente memoria para iniciar el scheduler. */
	for( ;; );
	return 0;
}
```

Listado 77 – Modelo para una nueva función main().

Crear un nuevo proyecto desde cero:

Como se acaba de mencionar, se recomienda que los nuevos proyectos sean creados a partir de los proyectos de demostración existentes.
Si por alguna razón esto no es deseable, un nuevo proyecto se puede crear mediante los siguientes pasos:

1. Cree un nuevo archivo de proyecto vacío o makefile mediante su cadena de herramienta elegida.
2. Agregue los archivos que se detallan en la Tabla 21 para el proyecto o makefile recién creado.

3. Copie un archivo FreeRTOSConfig.h existente en el directorio del proyecto.
4. Añada tanto el directorio del proyecto y FreeRTOS\Source\include lugar donde el proyecto se buscará para localizar los archivos de cabecera.
5. Copie la configuración del compilador del proyecto o makefile de demostración. En particular, cada puerto requiere una macro para setear, que asegura que los archivos de cabecera correctos están incluidos en la compilación.

Tabla 21 – Archivos fuente de FreeRTOS incluidos en el proyecto.

Archivos	Locación
tasks.c	FreeRTOS\Source
queue.c	FreeRTOS\Source
list.c	FreeRTOS\Source
port.c	FreeRTOS\Source\portable\[compiler]\[architecture] donde [compiler] es el compilador que se está usando y [architecture] es la variante de microcontrolador que se está usando.
port.x	Algunos puertos también requieren el proyecto para incluir un archivo de ensamblaje. El archivo se encuentra en el mismo directorio que port.c. La extensión de nombre de archivo dependerá de la cadena de herramientas que se utiliza (la x debe ser reemplazada por la extensión del archivo real).

Archivos de encabezado:
Un archivo de origen que utiliza la API de FreeRTOS debe incluir "FreeRTOS.h", entonces el archivo de cabecera que contiene el prototipo de la función API se utiliza, ya sea "task.h", "queue.h" o "semphr.h".