Que es un RTOS

Un RTOS es un administrador de recursos que hace las funciones normales de un sistema operativo pero garantiza una línea definida de tiempo (cumplir el deadline) en otras palabras no solo es importante la ejecución de las tareas sino su tiempo de ejecución.

De los sistemas operativos de PC como windows que realizan la operación de enviar emails, ver películas del DVD, tener un editor de texto como MSWord y demás el RTOS realiza operaciones corre en máquinas más limitadas y realiza cosas más específicas para el cliente, p.e., computar, conectarse a una red, dispensar dinero, situaciones que son críticas.

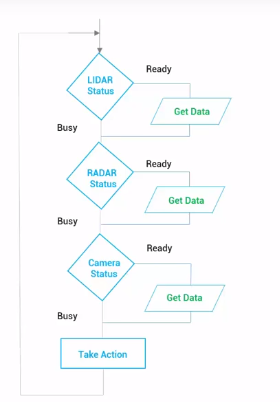
RTOS operan con una limitante de tiempo y tienen un comportamiento más predecible.

Métricas de un RTOS

Una de las métricas más importantes de un RTOS son los deadlines (derivar el resultado correcto con una limitante de tiempo).

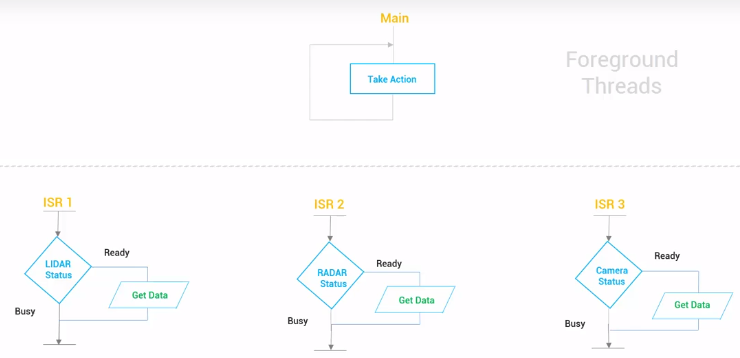
Otra métrica importante es la confiabilidad, que tiene que ver con la predictibilidad del sistema y garantiza una respuesta.

Busy-Wait System



Asumamos que tenemos un sistema de vehículo autónomo, las operaciones se ejecutan de manera secuencial y siempre está cambiando, la información de cámaras, radar y LIDAR cambian en el tiempo y no podemos esperar a que esto suceda, pues, puede causar accidentes, un sistema busy-wait es problemático.

Sistemas Multi-tareas

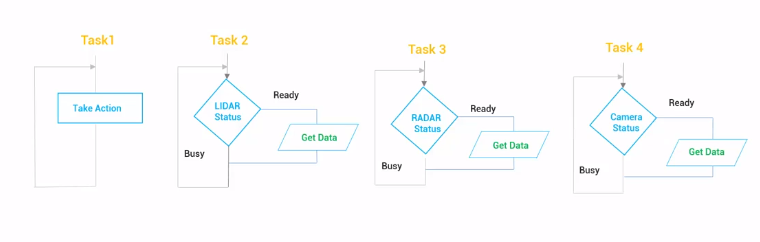


La solución a un busy wait system es dividir el sistema en tareas independientes en subrutinas de interrupción por ejemplo (background threads) y tener una tarea principal (foreground thread) de esta mnaera, por ejemplo, el LIDAR reaccionará a nueva data mientra que la tarea principal se ejecuta en el tiempo restante de las operaciones.

Si asignamos prioridades algunas tareas entonces ya el sistema se estaría convirtiendo en un soft-real-time system.

Recordemos que un RTOS cumple tiempo y es confiable. Este sistema en sí no es de este tipo, necesitamos un sistema con múltiples tareas en “foreground” y con sincronizaciones entre ellas.

RTOS



Finalmente llegamos a una solución más confiable, un sistema que es dividido por tareas. Para esto necesitamos un administrador de tareas llamado “Thread Scheduler”

El “Scheduler” o Administrador de Tareas

Básicamente es una función que corre en el trasfondo y que administra las tareas actuales que tiene el sistema y determina que tarea correr en cierto tiempo.

El sistema más básico de administración de tareas tiene tres componentes (sabemos que son más)

RUNNING = tarea actualmente ejecutandose

READY = tarea lista a ejecutarse

BLOCKED = tarea que está ejecutándose pero por algún motivo espera la llegada de un dato, por ejemplo, que el usuario presione un botón o una variable de otra tarea.

RTX AVANZADO

**Note que las respuestas pueden variar dependiendo del entorno utilizado (Keil y el MCU).**

**Vistazo General**

En este laboratorio verá configuración más avanzada de creación y destrucción de algunas tareas utilizando Keil RTX y se espera que este camino de ayuda a entender los Sistemas operativos aclaren algunas situaciones como “creación de tareas”, “terminación, “operaciones de cambio de contexto”.

Adicionalmente, este laboratorio posee algunos conceptos del manejador de tareas, por consiguiente observe bien como funciona. Si no entiende algunos conceptos avanzados no se preocupe, más adelante estos conceptos en el curso serán aclarados.

**Conexiones de Hardware**

Conecte las entradas y salidas como sigue en la siguiente figura, si no las posee, por el momento trabaje sin ellas y con un multímetro.

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Descripción | Dirección | MCU |
| SW1 | Switch Input | Entrada al MCU | PC\_13 |
| LED\_R | Rojo | Salida del MCU | PA\_5 |
| LED\_B | Azul | Salida del MCU | PA\_7 |
| LED\_G | Verde | Salida del MCU | PA\_6 |

**Ajustando Keil RTX**

Si aún no lo ha hecho debe descargar la version legacy de Keil RTX para ARM Cortex-M de <http://www2.keil.com/mdk5/legacy>.

Si desea ver más allá, el código del sistema operativo es abierto y está en:

<<Directorio de proyecto>>\ARM\RL\RTX\SRC

Ajuste el Keil RTX de la siguiente manera:

1. Inicie el proyecto Lab\_2
2. Verifique las opciones del “Target” y seleccione RTX Kernel como el Sistema Operativo preferido.

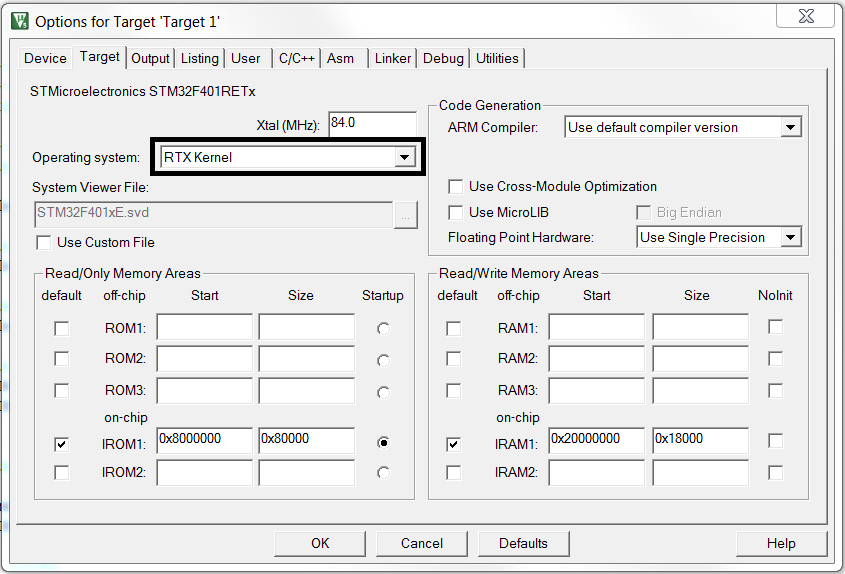


Figure 2: Select the RTX Kernel

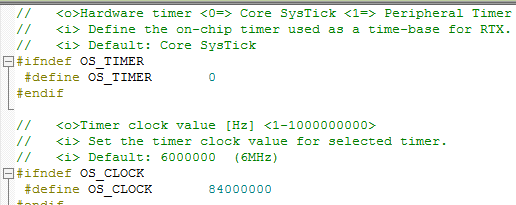
1. En el main.c inculya: #include <RTL.h>

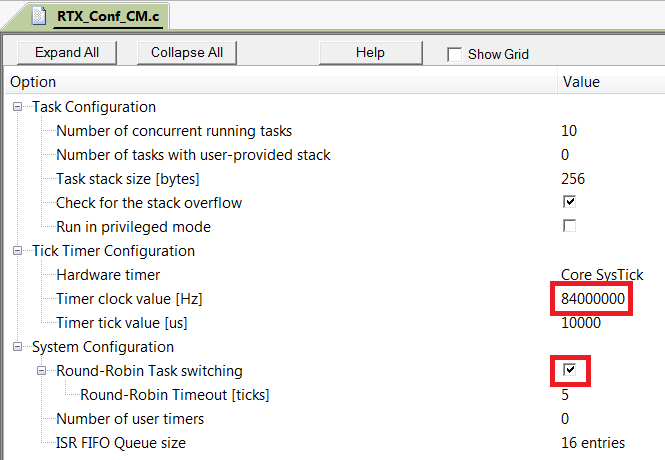


1. Note en la carpeta del proyecto el archivo RTX\_Conf\_CM.c y observe que este posee las configuraciones del sistema.

Lo demás será modificado automáticamente por nuestro compilador.

Ud. Puede configurar RTX modificando el archivo RTX\_Conf\_CM.c directamente o seleccionando el modo expert (wizard) para seleccionar opciones más gráficas a nivel de interfaz de usuario (hablando de CMSIS RTOS). Asegúrese de que el administrador de tareas sea Round-Robin Task switching y que el valor del timer clock sea fijado al del CPU.





**Las tareas en Keil RTX**

Keil RTX entre algunas cosas es:

* Libre y creado para ARM Cortex-M
* Provee funciones de C de alto nivel para construir aplicaciones de tiempo real usando tareas quasi-paralelas en el CPU
* De alta velocidad y baja latencia a interrupciones
* Tareas ilimitadoas de 254 prioridades
* Mailboxes ilimitados
* Kernel-aware debug para el sistema oprativo

RTX usa las notaciones de “task” a los procesos o tareas. Basicamente es una función de C con palabra clave **\_\_task** (dos barras abajo) que precede a una funci´n con retorno. Esto indica que el compilador no manejará la función como normalmente de entrada y salida que son las responsables de administrar el stack, como veremos más Adelante.

Tipicamente una tarea luce como sigue:

\_\_task void task(void){

for(;;){

……

}

}

Note algo importante, una tarea en un sistema embebido nunca termina y debe ser considerada como un pequeño programa dentro del sistema. Ud. Debe iniciar y terminar los procesos, esto se tocará más Adelante en el laboratorio. Cabe destacar que el retorno de las tareas es siempre **void**.

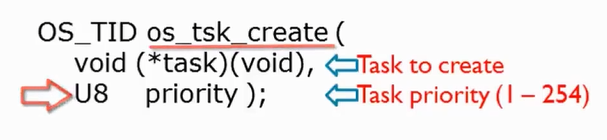
**Creación de las tareas**

Existen algunas funciones utilizadas para la creación de las tareas:

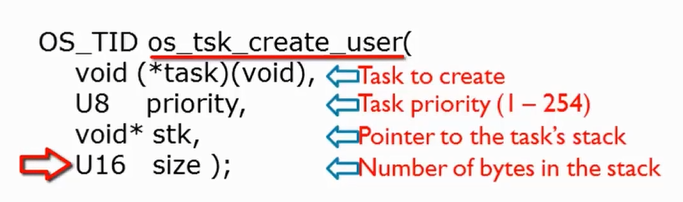
os\_tsk\_create(task,priority);

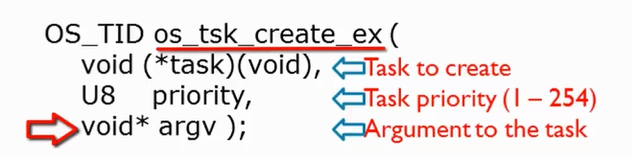
os\_tsk\_create\_user(task,priority,&stack,sizeof(stack));

os\_tsk\_create\_ex(task,priority,parameter);



Es permitido asignar la prioridad de las tareas por ud. También se puede asignar un “stack size” de memoria extensor usando la función **\_user**; de otra manera, RTX asignará un stack por defecto. Es también permitido pasar parámetrosa las tareas usando la función **\_ex**.





Todas estas funciones retornarán las tareas con un ID (tipo OS\_TID), y la manera estándar de crearlas es como sigue:

OS\_TID taskID;

\_\_task void task(void);

taskID = os\_tsk\_create(task,0);

El ID de la tarea es una manera conveniente de manejarlas y será utilizada por algunas funciones para manejar la tarea (ejemplo, suspenderlas).

Una tarea puede ser creada por otra tarea. Y usualmente, ud. creará la primera tarea del sistema operativo utilizando la function \_init:

os\_sys\_init(task);

Esto hace el trabajo necesario de configuración del sisetma operativo para iniciar la tarea. La primera tarea usualemnte crea otras tareas más significativas y luego se destruye a ella misma. Ver el siguiente párrafo.

**Terminación de las tareas**

Es también possible terminar las tareas utilizando las siguientes funciones:

os\_tsk\_delete(taskID);

os\_tsk\_delete\_self();

La primera función borra una existente (tarea), mientras la segunda función borra la tarea a ejecutarse o ejecutandose, por ejemplo, ella misma.

La function \_delete tiene el argumento de retorno OS\_RESULT, el cual es tipicamente un código de error.



**Conmutación de Contexto**

Una tarea mantiene dos pedazos de información:

TCB (Task Control Block) y el Stack

El TCB es alojado dinamicamente cuando se llama la función os\_task\_create(). Todas las tareas en los ajustes preemptivos tendrán su propio “call stack“ que es una zona de memoria que guarda información referente a dicha tarea; el tamaño es dinámicamente alojado por la función os\_task\_create().

Para realizar el context switching de una tarea a otra:

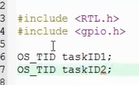
* El SO guarda el estado del stack y las variables (algunas guardadas por el stack del sistema debido a las interrupciones)
* Guarda el estado actual y el de ejecución (runtime) en el TCB.
* Los registros actuals del CPU serán empujados al stack
* El stack pointer carga en el puntero de la siguiente tarea la próxima tarea a ejecutarse
* El siguiente contexto es cargado a los registros del CPU
* La siguiente tarea se pone en ejecución cargando el PC (recorder que apunta a la siguiente instrucción a ejecutarse).

**Procedimiento de laboratorio**

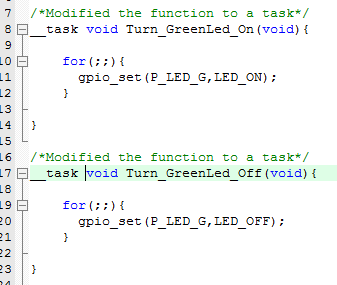
1. Si no lo ha realizado, arranque el proyecto
2. Si no está presente copie el main.c
3. Copie la configuración de RTX como es descrito anteriormente.
4. Compile el código.
5. Cargue en la tarjeta nucleo Nucleo.
6. Presione el Reset en la tarjeta (boton negro – SW1).
7. Describa que pasa cuando esto se da.

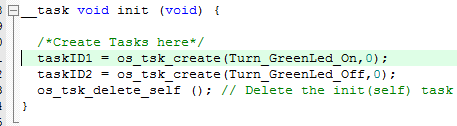
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Modifique la rutina de Turn\_GreenLed\_On y Turn\_GreenLed\_Off que son funciones a tareas. Cree estas tareas con el init task y prioridad 0 antes del llamado a os\_tsk\_delete\_self call. Asegurese de haber salvado el ID de retorno en una variable global de tipo OS\_TID.



Añada \_\_task antes de cada una de estas declaraciones de funciones. Use la función os\_tsk\_create en init().





1. Cuando corre el programa en la tarjeta, describa que es lo que observa, concuerda con lo esperado?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Ponga un breakpoint en una de las funciones “create”. Haga un paso a paso (F11)para ver la ventana de desensamblador y lo que hace. Liste las diferentes etapas de la creación de las tareas. (Note que el texto en marron en la ventana del desensamblador son las funciones del sistema operative que ejecuta paso a paso)



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Liste los mayores pasos (nombre de funciones) para crear una tarea.

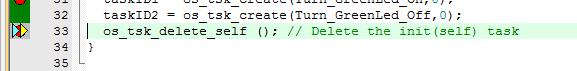
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

De los pasos 10 y 11 ud puede verificar el código, no necesita ir y ejecutar para tener la respuesta simplemente debe tener idea de que necesita el sistema operativo para generar las tareas.

1. Existe una función para crear tareas: os\_tsk\_create\_user\_ex de la cual se necesitan enviar 5 parametros. ¿Cuáles son estos parámetros y que hace esta función? Adivine primeramente, luego busque el código para ver si ud esta correcto.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

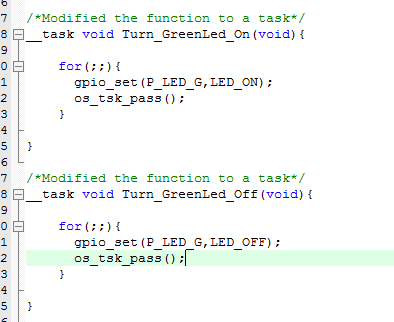
1. Similarmente, ud puede ajustar un breakpoint en os\_tsk\_delete\_self(). Ejecute paso a paso y describa que hace esta función.



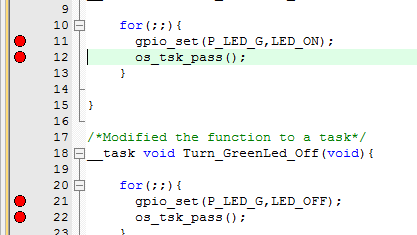
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Puede verificar el código para obtener la respuesta tambien, no necesita ir a la ventana de desensamblador.

1. Ahora intente añadir os\_tsk\_pass(); a las tareas creadas en el ciclo for, inmediatamente luego de gpio\_set call. Esto le dice voluntariamente al Sistema Operativo que culmine su ejecución de la tarea actual y ejecute la siguiente tarea lista a ejecutar. El context switch ocurrirá en estas líneas. Describa como esto afecta el comportamiento del LED asociado. Corra el programa y verifique su hipotesis.



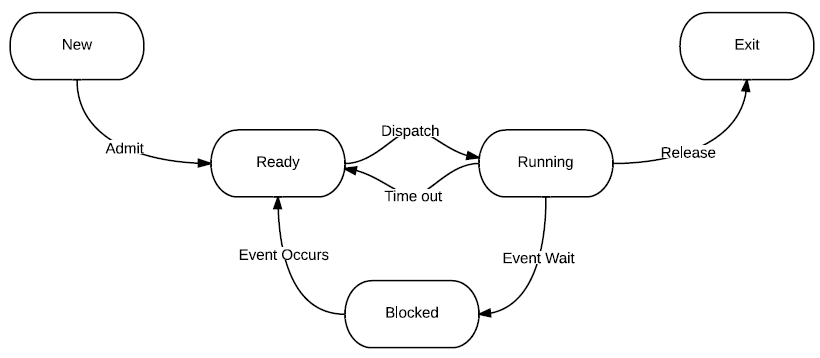
1. Set a breakpoint at one of the os\_tsk\_pass() lines. Step through it to see what it does. List the major steps for context switching. Also, pay attention to the green LED while you are stepping through.



Conceptualmente la siguiente tarea lista tiene la misma prioridad, cambiar el estado de la tarea actual a lista y ponerla en estado de lista. Dispara entonces luego la siguiente tarea cambiando a estado de correr.

Ud puede verificar el código fuente de Keil RTX y ver que implica tener esto.

1. Finalmente se ha introducido a grosso modo en los estados de las tareas y el modelo estándarcomo se muestra a continuación.



Intente hacer pareo de las siguientes funciones en sus transiciones de estado:

os\_task\_create():

os\_task\_delete():

os\_tsk\_pass():

rt\_dispatch():

**El laboratorio está dividido en 50/50, 50% es la parte superior y 50% es la parte inferior que debe completer.**

Hellen

Cree una función que se ejecute indefinidamente sumando un número de 1 en 1 cada vez que sea llamada utilizando las funciones del SO, luego de 100, elimínela

Diego

Cree una función que se ejecute indefinidamente generando un numero aleatorio entre 0 y 10, si el numero generado es 5, elimínela.

Laura

Cree una función que se ejecute indefinidamente generando un XOR entre el estado de los leds, si el resultado es 1 cree una tarea, de lo contrario elimínela.