

# **Sistemas embebidos basados en microcontroladores 2.**

## **“Laboratorio: Mini\_RTOS.”**

# **Profesor: Rodrigo Aldana López.**

# **Alumnos:**

# Gustavo Araiza Obeso ie699209.

# *Daniel Celis Tapia ie689705.*

# Fecha: 19/02/2018

Laboratorio 1: MiniRTOS

El microkernel se basa en un calendarizador que entrega una serie de instrucciones ligadas entre sí. Su orden se maneja dependiendo de su prioridad y estado, ya que el calendarizador usado fue uno basado en la prioridad de las instrucciones. Como queremos que todas se ejecuten en algún punto, cambiamos su estado de disponibles / corriendo a esperando, ya que cumplen su función y se añaden a una “lista de espera” donde el sistema operativo hará un conteo para su reactivación. Mientras tanto, las otras tareas de más baja prioridad se podrán ejecutar.

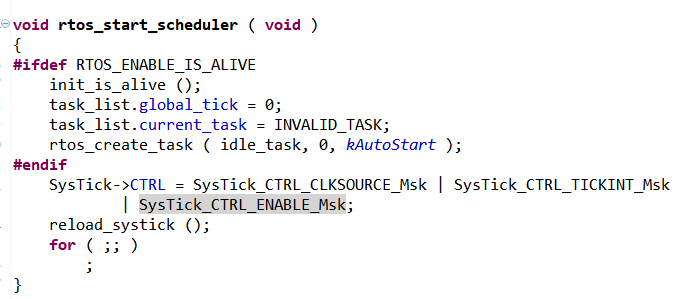
Lo importante en este proceso es el guardado del contexto, ya que buscamos que cada instrucción se ejecute independientemente de las otras, pero a través del mismo sistema. Para ello, utilizamos el guardado del *stack*, así como de los registros de contexto que permiten relanzar la función engañando al procesador y haciéndole creer que nunca dejó de ejecutar dicha instrucción.

La parte más interesante es hallar el espacio de memoria correspondiente al guardado del contexto, ya que depende de las variables totales a utilizar y de quién llamó a el administrador de cambio de contexto. Es decir, si fue llamado por la creación de una nueva instrucción o el sistema que intenta restaurar el estado de una función anterior.

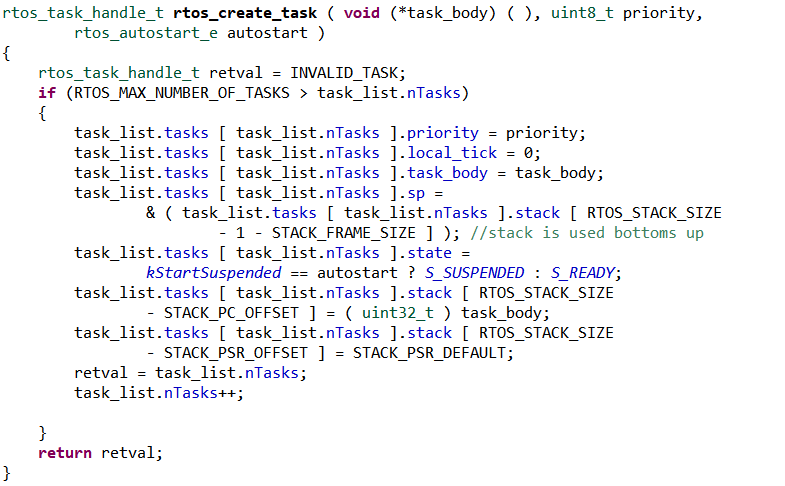
El código que se ejecuta se puede ver en <https://github.com/GustavoAraizaO/Mini_RTOS>

A continuación, se ejemplificará el proceso de las funciones principales a utilizar:

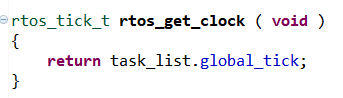
Función Start scheduler, Inicializamos el sistema operativo, colocando el reloj global en cero y cargando una tarea inválida como base.



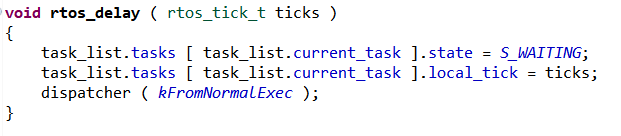
Función create task, donde podemos realizar la creación de la tarea.



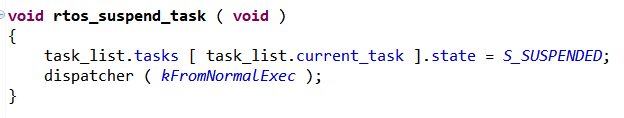
Función *get clock*, con esta función obtenemos el valor del reloj del sistema (global tick)



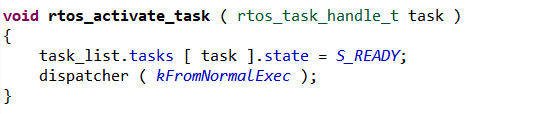
Función *delay*, mandamos a dormir una tarea que sea llamada en cierto tiempo, colocando primero la tarea en espera para después colocar los ticks en el reloj local de la tarea finalizando con el llamado a dispatcher.



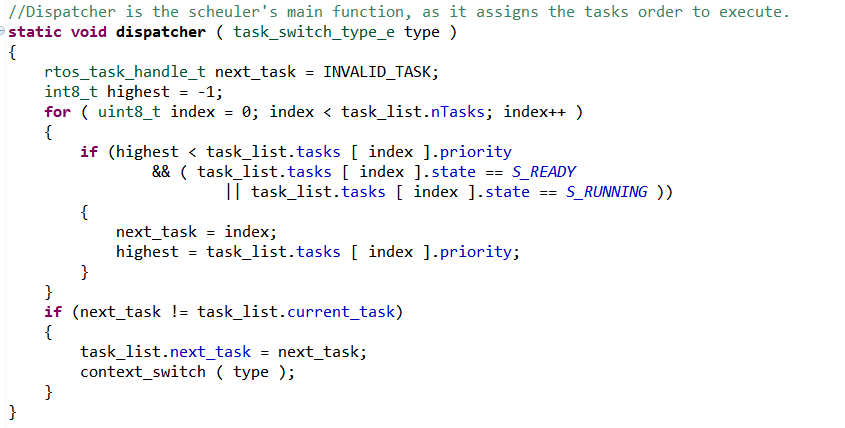
Función *Suspend task,* suspendemos la tarea llamada finalizando con el dispatcher.



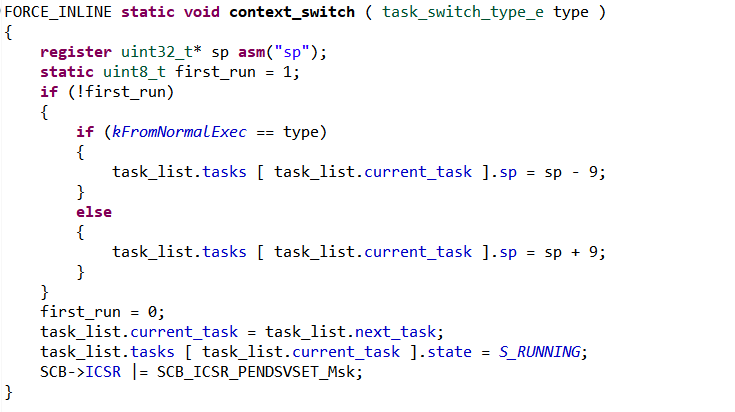
Función *actívate task,* con esta tarea activamos la tarea solicitada, colocándola en estado de “Ready”.



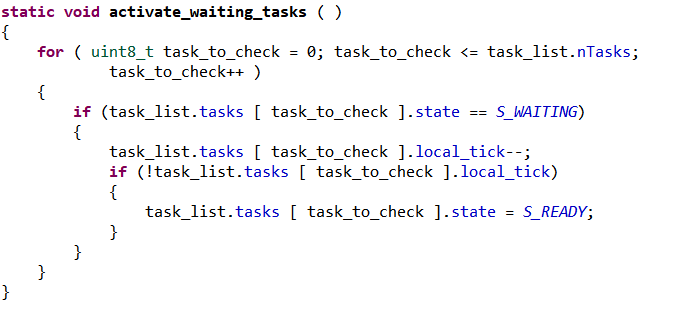
Función *dispatcher*, (calendarizador), con esta tarea nos encargamos de asignar qué tareas siguen de acuerdo a la prioridad y su estado.



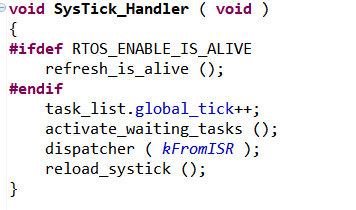
Función *context switch*, la cual se encarga de cambiar el contexto de acuerdo a cada tarea que se recarga para ser ejecutada y se reestablezca su entorno.



Función que activa las tareas que se encuentran en espera de acuerdo al conteo general del sistema operativo.



SysTick handler es la interrupción donde el sistema operativo se encarga de checar el estado de las tareas.



Interrupción de bajo nivel que recarga el contexto de acuerdo al stack pointer que se trae para engañar al procesador.

