

**Laboratorio 1 “RTOS”**

**Nombre:**

Jorge Mizael Rodríguez Gutiérrez

Luis Roberto Lomeli Plascencia

**Expediente:**

IE698323

IE700093

**Correo:**

[ie698323@iteso.](mailto:ie698323@iteso.)mx

[ie700093@iteso.](mailto:ie700093@iteso.)mx

**Carrera:**

Ing. Electrónica

**Fecha:**

17/Enero/2017

**Materia:**

Sistemas embebidos 2

**Maestro:**

Rodrigo Aldana López



**Pagina**

**Portada**.……….………………………………………………………………………………….... 1

**Índice**……………………………………………………………………………………………...…2

**Introducción**………………………………………………………………………………………..3

**Planteamiento del problema**…………………………………………………………………….4

**Modificaciones al código**………………………………………………………………………...5

**Funciones de RTOS……**………………………………………………………………………….6

**Introducción**

Como primer laboratorio de la clase de sistemas embebidos 2, se realizó un sistema operativo para la tarjeta freedom K64F, la implementación de dicho sistema operativo se llevó a cabo gracias a la ayuda del profesor de la clase, al implementar este sistema operativo podemos decir que es la implementación de un sistema operativo muy básico, que incluso no tiene interfaz gráfica.

**Planteamiento del problema**

Para esta actividad deberás clonar el proyecto que se encuentra en esta liga: [https://github.com/raldanal/rtos.](https://github.com/raldanal/rtos) En ella encontrarás el código fuente para que coloques en un proyecto nuevo de MCUXpresso para la tarjeta freedom board K64.

El archivo rtos\_main.c contiene el punto de entrada del programa, en el cual se definen 3 tareas las cuales el objetivo es que terminen corriendo en paralelo manteniendo su funcionalidad y contexto.

El módulo rtos contiene la API e implementación incompleta de un calendarizador por prioridades que deberás completar. Por lo pronto no hay nada implementado a excepción del SysTick timer configurado para generar una señal de "vida” que servirá para saber si el sistema operativo está vivo.

La labor será terminar de implementar el resto del archivo rtos.c de manera que las tareas en el archivo main corran como se desea. A continuación, se muestran pseudocódigos de lo que las funciones deberán realizar.

**Modificaciones realizadas al código**

**Archivo “rtos.h”**

Para este archivo cosas que corregimos fueron etiquetas **#define** y **enum**, con la finalidad de quitar los números mágicos en el programa. Ejemplo de líneas que agregamos son las siguientes:

Los defines son para nueros que representaron algo en el código, que tal vez para el usuario no era claro. Por lo tanto, agregamos estas etiquetas para respetar los lineamientos de codificación y hacer más entendible el código.

**#define** ALL\_TASKS\_CREATED\_CORRECTLY 66

**#define** RESERVED\_MEMORY 10

**#define** STACK\_OFFSET\_ISR\_AND\_EXEC 9

**#define** TASK\_IDLE 1

**#define** END\_OF\_STACK 1

**#define** VALUE\_START\_PERIOD 1

Pasa algo similar con los enums, ya que podríamos decir que estos tienen la misma función, la diferencia radica en que los números que representan estas etiquetas se representan con más frecuencia y por lo general son dos o más etiquetas en las cuales su número es una secuencia del otro.

/\*! @brief type of variable start \*/

**typedef** **enum**

{

*START\_ZERO* = 0, *START\_ONE*

} start\_values\_type\_e;

/\*! @brief priority of tasks \*/

**typedef** **enum**

{

*LOWEST\_PRIORITY* = -1, *PRIORITY0* = 0, *PRIORITY1*, *PRIORITY2*

} task\_priority\_type\_e;

/\*! @brief led state \*/

**typedef** **enum**

{

*ON* = 0, *OFF*

} led\_state\_type\_e;

**Funciones del archivo “rtos.c”:**

Una de las funciones a las cuales se les hizo una modificación importante fue:



Esta es la clave de un sistema operativo, cuando realizamos el cambio de contexto de una tarea a otra, sin perder los datos que se estaban usando en ese momento cuando la tarea dejó de ejecutarse.

Hubo que conocer por tanto cuando se realizaba el cambio de contexto desde una interrupción, el cual nos desplazaba el stack frame 9 posiciones hacia arriba del stack pointer que obteníamos.

A diferencia de un cambio de contexto en ejecución normal el cual casualmente tenía el mismo valor de offset pero en lado contrario, esto no es precisamente así en cualquier situación pero en esta ocasión el contexto de las tareas eran bastante similar por lo que suponemos tiene un desplazamiento parecido.

Se implemento también una función que específicamente busca entre todas las tareas y propone la siguiente tarea a ejecutar según las condiciones que le hayamos pasado como parámetros, a esta función le llamamos calendarizador el cual junto con la que realiza el cambio de contexto, se forma a lo que llamamos sistema operativo el cual no es otra cosa mas que un programa que permite ejecutar de manera ordenada y sin errores dos o más procesos en línea que parecen ser simultáneos debido a que se les da cantidades de tiempo pequeñas en las que utilizan tiempo del procesador, y no se les puede llamar tareas paralelas porque solamente contamos con un procesador del cortex M-4.

Tuvimos en una de las funciones un pequeño percance al estar almacenando el stack pointer de la tarea únicamente en la primera corrida, lo cual carece de sentido por que al ser la primera corrida le estamos diciendo ya donde debe estar el stack de esa tarea y sin embargo lo esta modificando, entonces a la hora de hacer el cambio de contexto la información del frame no se encuentra disponible.

Entonces lo único que se modifico fue que en lugar de almacenar el stack de la tarea la primera vez, se hiciera todo el tiempo excepto la primera entrada. Y así podríamos con certeza volver al contexto preciso de esa tarea indicando el stack pointer en el lugar adecuado.