

Temas Selectos de electrónica y computación I

Sistemas Embebidos

**Hernández Machuca Sergio Francisco**

**EQUIPO 01**

* Hernández Hernández Juan Manuel

(Código y reporte)

* Fomperoza Salgado Isaí   
  (Código y reporte)
* Ramírez Torres Martha Diana   
  (Código y reporte)

**26 de noviembre del 2020, Xalapa, Ver.**

**FreeRTOS**

La mayoría de los sistemas operativos parecen permitir la ejecución de varios programas al mismo tiempo. A esto se le llama multitarea. En realidad, cada núcleo de procesador solo puede ejecutar un único hilo de ejecución en un momento dado. Una parte del sistema operativo llamada planificador es responsable de decidir qué programa ejecutar y cuándo, y proporciona la ilusión de ejecución simultánea al cambiar rápidamente entre cada programa.

El scheduler en un sistema operativo en tiempo real (RTOS) está diseñado para proporcionar un patrón de ejecución predecible (normalmente descrito como determinista). Esto es particularmente de interés para los sistemas integrados, ya que los sistemas integrados a menudo tienen requisitos de tiempo real. Un requisito de tiempo real es aquel que especifica que el sistema integrado debe responder a un evento determinado dentro de un tiempo estrictamente definido. Solo se puede garantizar el cumplimiento de los requisitos de tiempo real si se puede predecir el comportamiento del programador del sistema operativo.

Existen dos tipos fundamentales de RTS (Real Time Sistems):

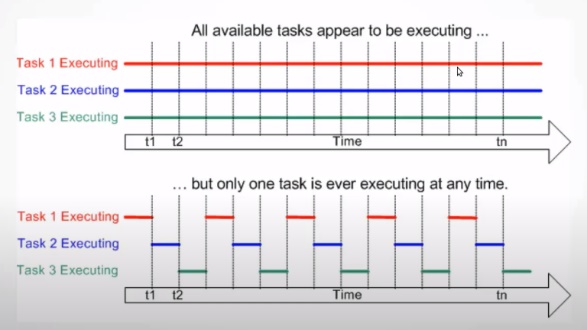
Soft: Las tareas se tienen que completar en el menor tiempo posible, pero éstas no tienen que concluir en un tiempo específico.

Hard: Las tareas tienen que realizarse correctamente y en un tiempo en particular.

FreeRTOS es un kernel de tiempo real (o un Scheduler de tiempo real), una clase de RTOS que está diseñado para ser lo suficientemente pequeño como para ejecutarse en un microcontrolador, aunque su uso no se limita a aplicaciones de microcontroladores.

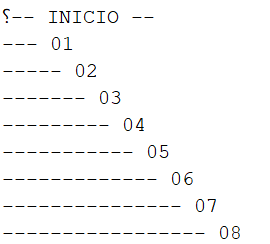
**Comandos importantes utilizados**

|  |  |
| --- | --- |
| LÍNEA DE CÓDIGO | DESCRIPCIÓN |
| QueueHandle\_t mensaje; | Manejador de colas. |
| TaskHandle\_t Tarea\_Handler, Tarea7\_Handler; | Manejador de tareas. Se declaran por separado ya que se hará uso específicamente de ellas en determinadas tareas, en este caso las tareas 7 y 8. |
| TaskHandle\_t xHandle; | Manejador de tareas. |
| mensaje = xQueueCreate(10, sizeof (String) ); | Se crea la variable para el manejador de colas. |
| xTaskCreate(); | Con esta instrucción se crea una tarea. |
| xQueueSend(mensaje, &aux , portMAX\_DELAY); | Esta instrucción manda el mensaje alojado en la variable antes declarada. |
| vTaskDelete( NULL ); | Se borra la tarea. |
| (void) pvParameters; | Este void nos permite manipular las tareas. En este caso lo hacemos desde el puerto serial. Con la atención a eventos. |
| vTaskSuspend | Se suspende la ejecución de una tarea determinada. |
| vTaskSuspendAll(); | Se suspende la ejecución de todas las tareas. |
| vTaskResume | Se reanuda la ejecución de determinada tarea. |
| vTaskPrioritySet | Permite modificar la prioridad de una tarea dada. |
| for (;;) // Ciclo infinito | Se ejecuta de manera constante esta sección de código que corresponde a la recepción del puerto serial. |

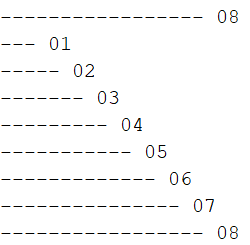
**Aplicación**

La aplicación se basa en la creación de múltiples tareas donde cada una imprime un mensaje en el puerto serial. La idea de utilizar FreeRTOS es de ejecutar todas las tareas casi al mismo tiempo, estableciendo prioridades para cada una y mandando datos entre las tareas, como fue sugerido en la actividad.

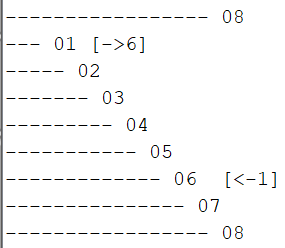
**Ejecución**

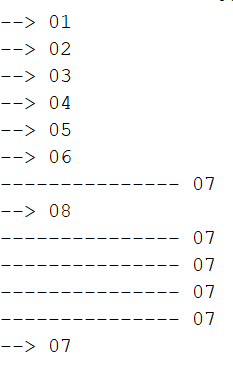


En la primera ejecución el planificador ejecuta todas las tareas independientemente de la prioridad.



Como la tarea 8 tiene mas prioridad y es la que tarda menos, en las siguientes ejecuciones esta tarea se ejecutará antes que las anteriores.

En la novena ejecución de la tarea 1, esta manda un mensaje a la cola para la tarea 6, esto se ve reflejado al imprimir el mensaje recibido en la tarea 6.

Cuando las tareas terminen sus ejecuciones correspondientes imprimen un mensaje donde anuncian su eliminación. Aunque la tarea 8 se ejecuta mas veces que las demás, esta termina con las tareas de la 1 a la 6, ya que, al tener más prioridad, se ejecuta más veces y termina más rápido.

La tarea 7 tiene más duración debido a que se ejecuta 15 veces.

**Cambios hechos**

* Aumentamos el número de tareas a 8 utilizando la notación sugerida
* La tarea 1 envía un mensaje a la tarea 6 en la novena ejecución. (Utilizamos una función para enviar a la cola e imprimimos en el monitor serial que se envía y donde se recibe).
* La tarea 8 tiene mayor prioridad y se ejecuta 20 veces
* La tarea 7 se ejecuta 15 veces
* Se añadieron tres funciones que consideramos importantes:

1. Paro de emergencia y reinicio escribiendo “x” en el puerto serial
2. Suspensión de tareas 7 y 8 reinicio escribiendo “s” en el puerto serial
3. Reanudar tareas 7 y 8 reinicio escribiendo “r” en el puerto serial

* Líneas de código comentadas

**Conclusión.**

Los sistemas de tiempo real nos permiten la ejecución de múltiples tareas casi al mismo tiempo gracias al planificador, aparentando una ejecución simultánea. Esto nos permite funciones avanzadas para procesadores de uno o dos núcleos en vez de comprar procesadores con más capacidades, ahorrando costos.