FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Sistemas Embebidos
FreeRTOS - Tareas

Laboratorio de Sistemas Embebidos

http://laboratorios.fi.uba.ar/lse/

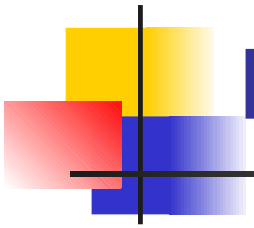
seminario-embebidos@googlegroups.com

66.48 & 66.66 Seminario de Electrónica: Sistemas Embebidos

http://laboratorios.fi.uba.ar/lse/seminario/

Ingeniería en Electrónica – FI – UBA

Buenos Aires, 22 de Mayo de 2013



Funciones de Tarea

En FreeRTOS las **tareas** (Task) se implementan con **funciones de C**

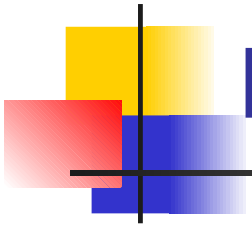
La única consideración especial es el **prototipo** de dichas funciones (debe retornar **void** y recibir como parámetro un **puntero a void**)

```
void ATaskFunction (void *pvParameters);
```

Cada tarea es un pequeño programa en si mismo que tiene un punto de entrada, que contiene las inicializaciones necesarias, que normalmente correrá por siempre en un lazo infinito, del que no sale nunca

Las tareas en FreeRTOS no tienen permitido retornar (**! return;** ni llegar a la **} de cierre**), se puede **eliminar** explícitamente **tareas innecesarias**

La simple definición de una función de tarea se puede usar para crear cualquier número de tareas (cada tarea creada es una instancia de ejecución con sus propios recursos: stack, variables, etc.)



Estructura de una Función de Tarea

```
void ATaskFunction (void *pvParameters)
{
    // Punto de entrada - Sección de Inicialización de la Tarea
    int iVariableExample = 0;

    // Cuerpo de la Tarea - Lazo infinito de iteración
    for (;;)
    {
        // Código de implementación de funcionalidades de la Tarea
    }

    // La Tarea NUNCA debe pasar de éste punto, se debe eliminar
    vTaskDelete (NULL);
}
```

Tareas de mayor nivel y transiciones

Una aplicación puede consistir en varias Tareas. Si corre sobre un micro de un solo núcleo se ejecutará una sola de ellas a la vez, lo que implica que una tarea puede estar en dos estados (modelo simplificado)

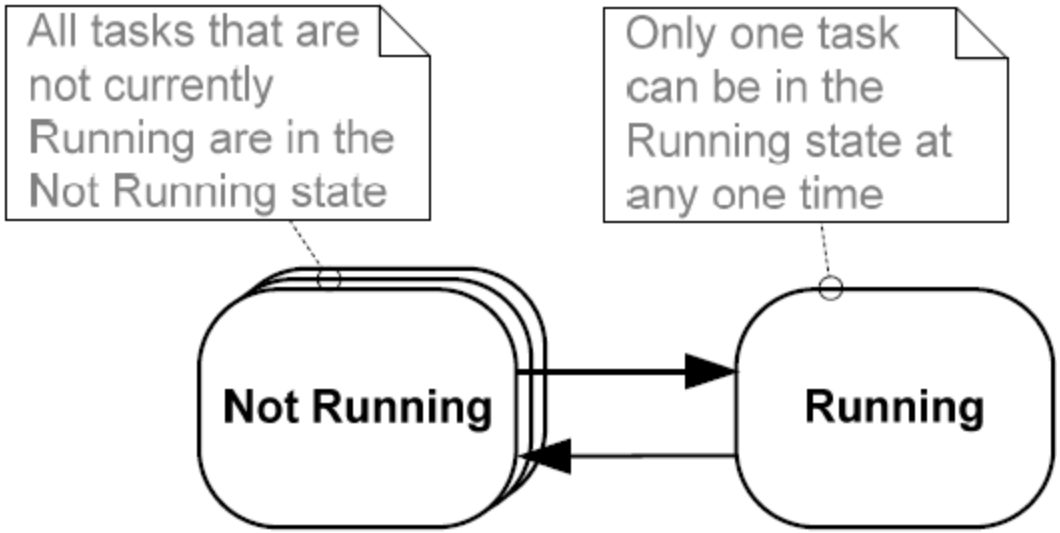
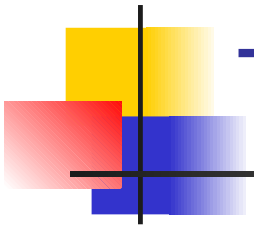


Figure 7. Top level task states and transitions



Tareas de mayor nivel y transiciones

Cuando una tarea está en Running State el micro la ejecutará

Cuando una tarea está en Not Running State estará dormida, su estado habrá sido salvado como ready para reanudar su ejecución la próxima vez que el scheduler decida que debe entrar en Running State

Cuando una tarea reanuda su ejecución lo hace desde la instrucción que debía ejecutar antes de abandonar Running State

La transición de una tarea de Not Running State a Running State se denominan “switched in” o “swapped in”, mientras que a la transición inversa se denomina “sitched out” o “swapped out”
El scheduler de FreeRTOS es la única entidad que puede switchear (in u out) tareas

Ejecución de dos tareas de igual nivel

:

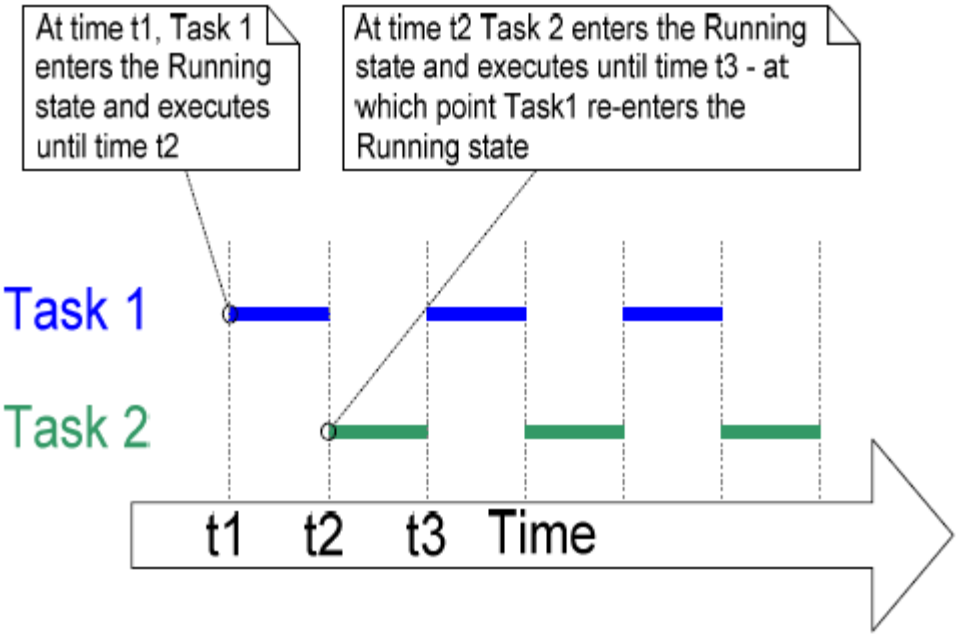


Figure 9. The execution pattern of the two Example 1 tasks

¿Cuándo se ejecuta el Kernel?

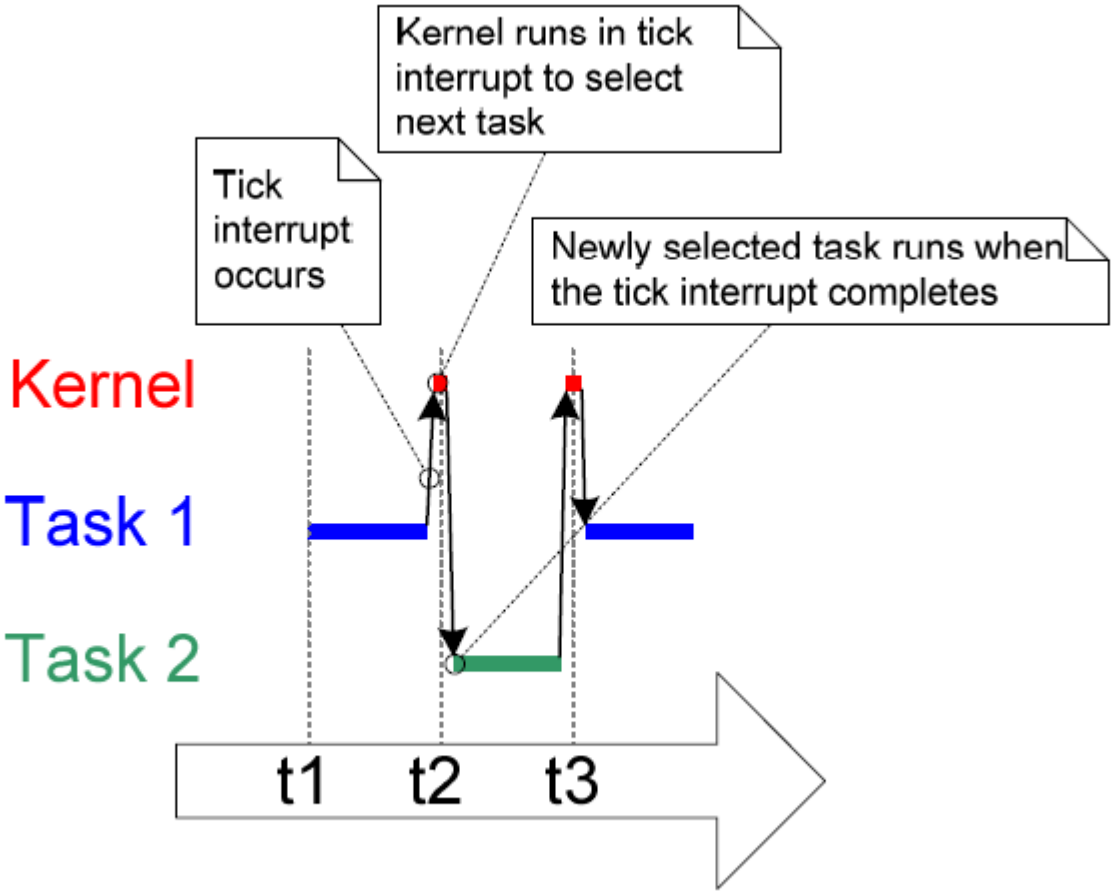


Figure 10. The execution sequence expanded to show the tick interrupt executing

¿Qué sucede si una es de más nivel?

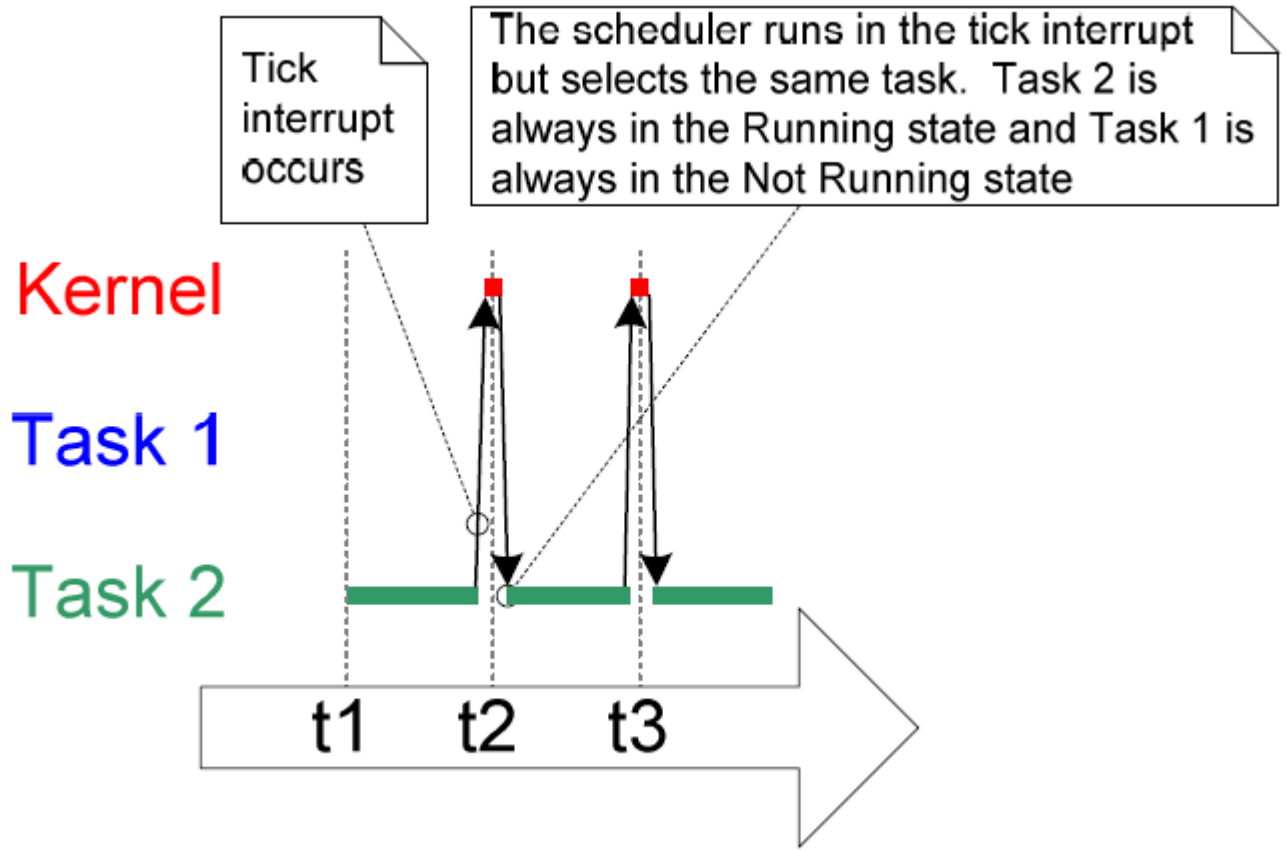


Figure 12. The execution pattern when one task has a higher priority than the other

Diagrama de Estado de una Tarea

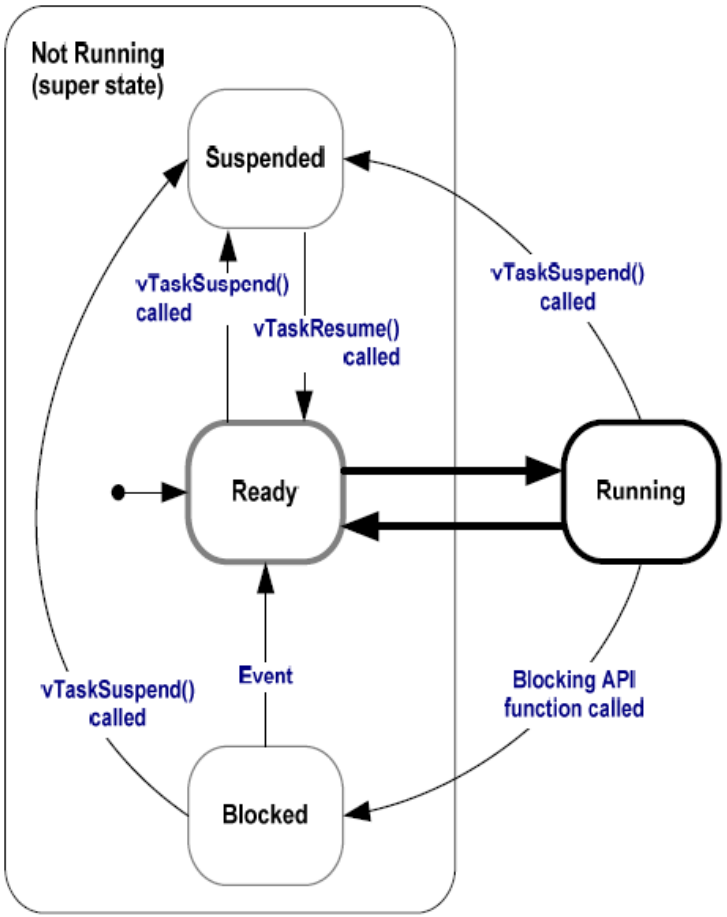


Figure 13. Full task state machine

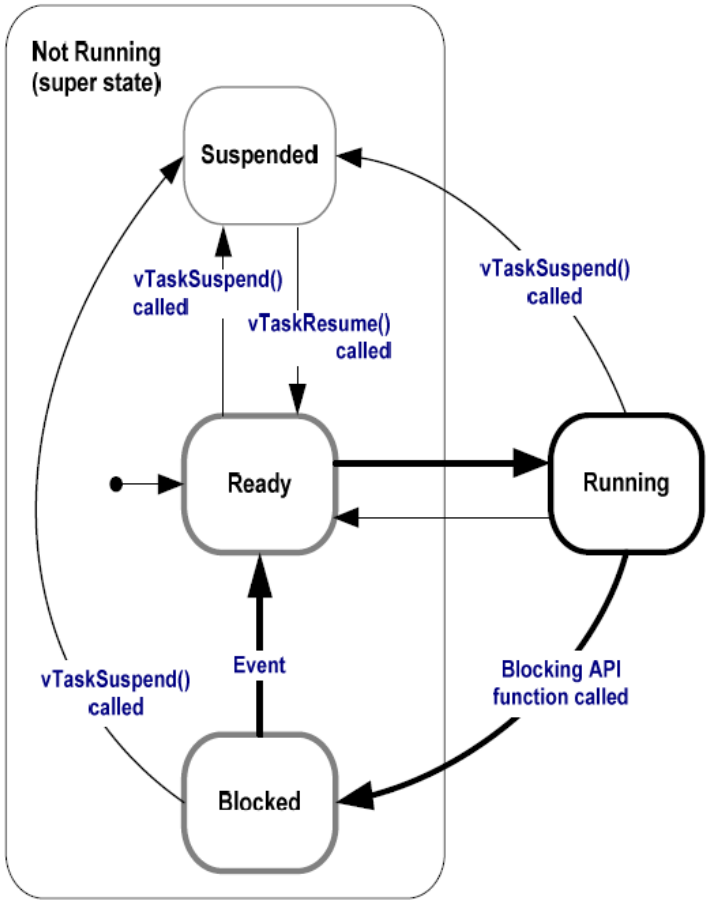


Figure 16. Bold lines indicate the state transitions performed by the tasks in Example 4

Bloqueo de Tareas (vTaskDelay)

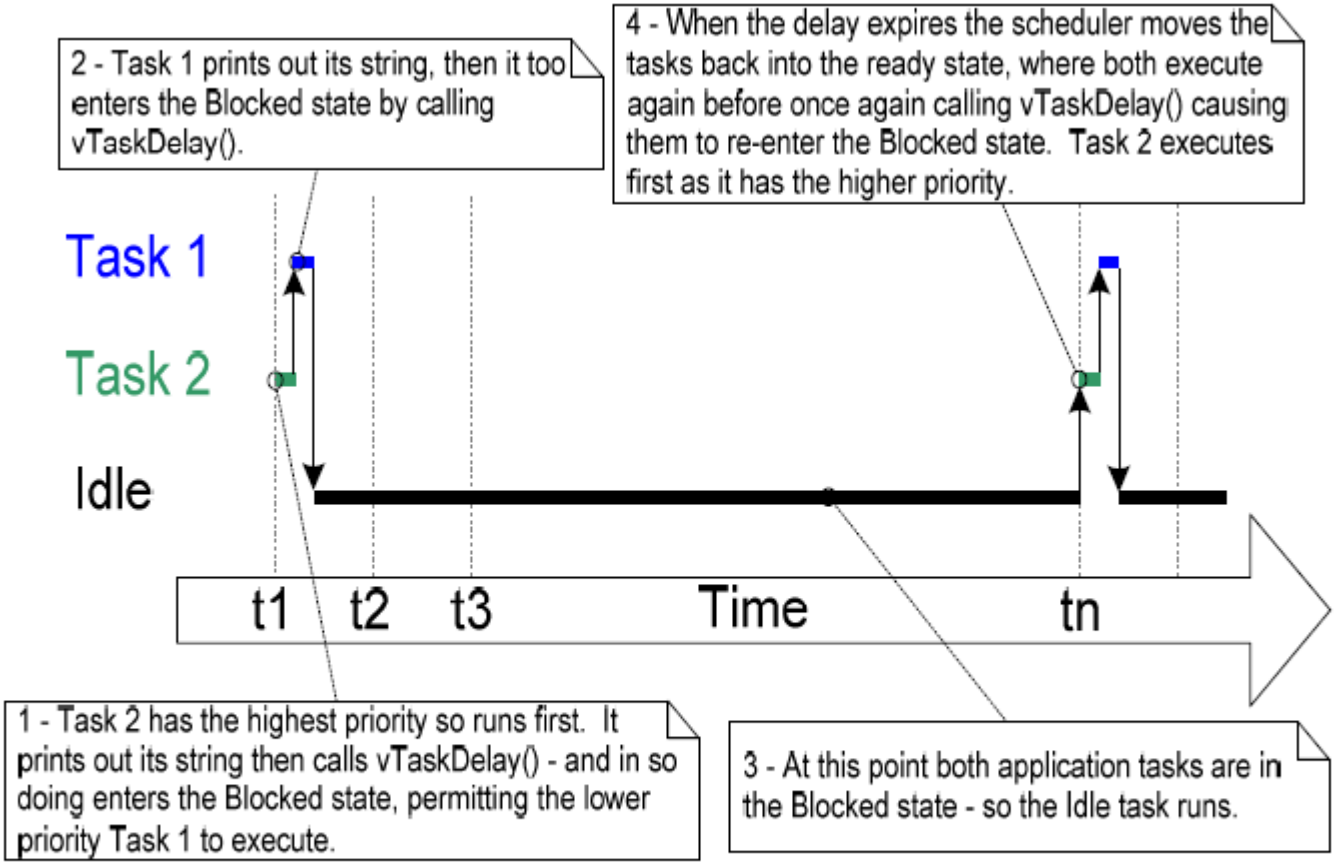


Figure 15. The execution sequence when the tasks use vTaskDelay() in place of the NULL loop

Bloqueo de Tareas (vTaskDelay)

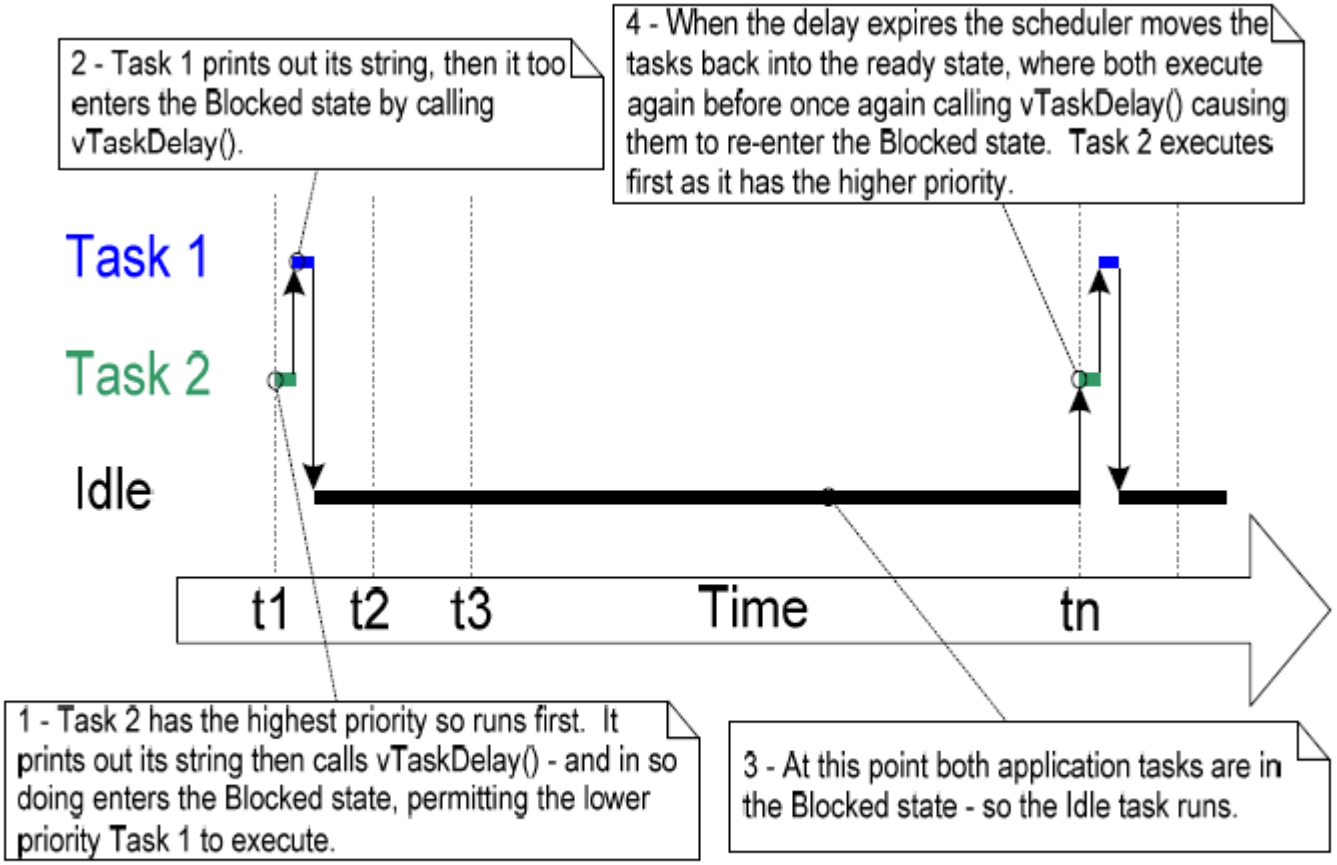


Figure 15. The execution sequence when the tasks use vTaskDelay() in place of the NULL loop

Agreguemos una tarea periódica

:

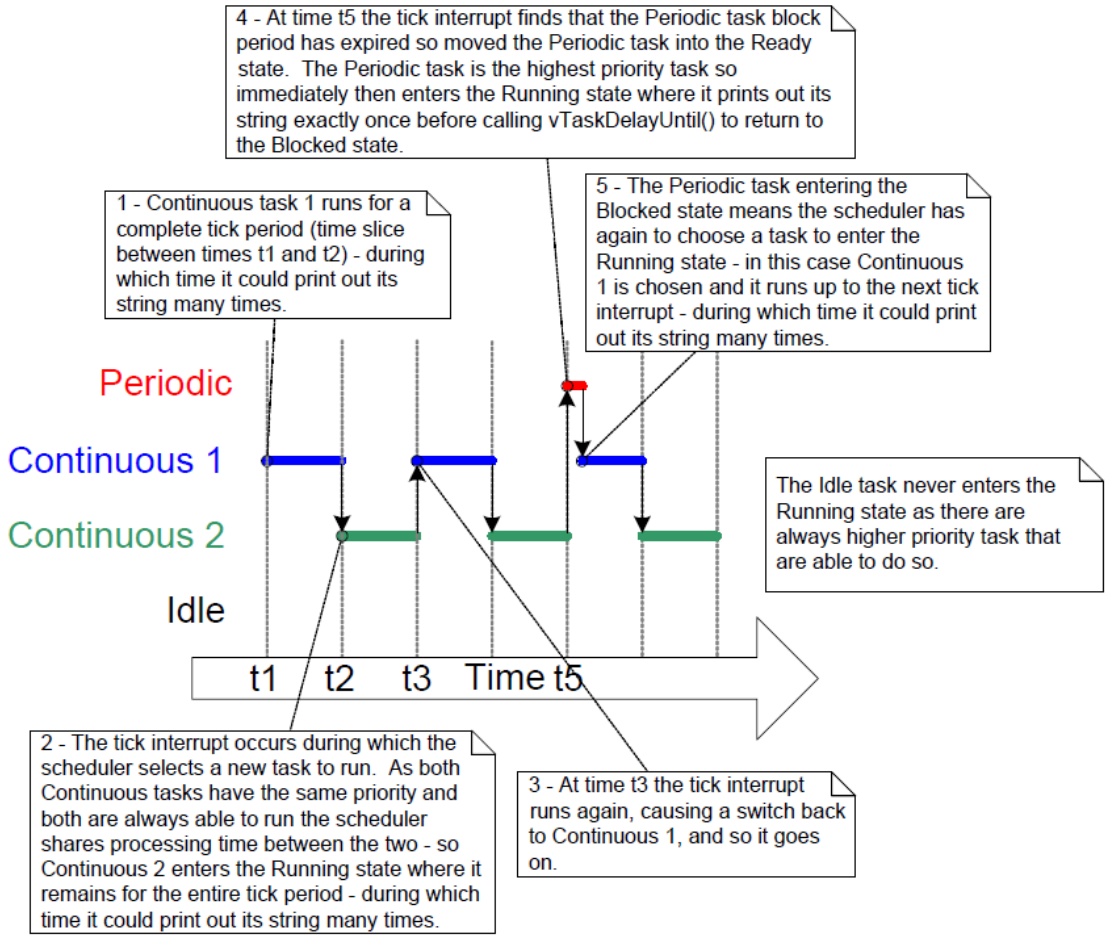


Figure 18. The execution pattern of Example 6

Modificando niveles en ejecución (vTaskPrioritySet, uxTaskPriorityGet)

:

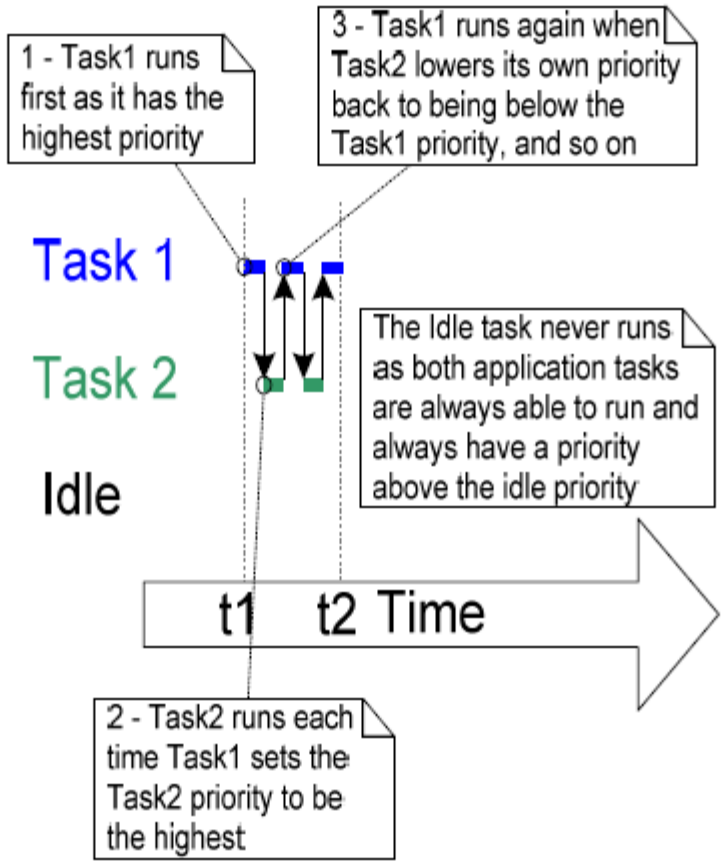


Figure 20. The sequence of task execution when running Example 8

Borrando Tareas (vTaskDelete)

:

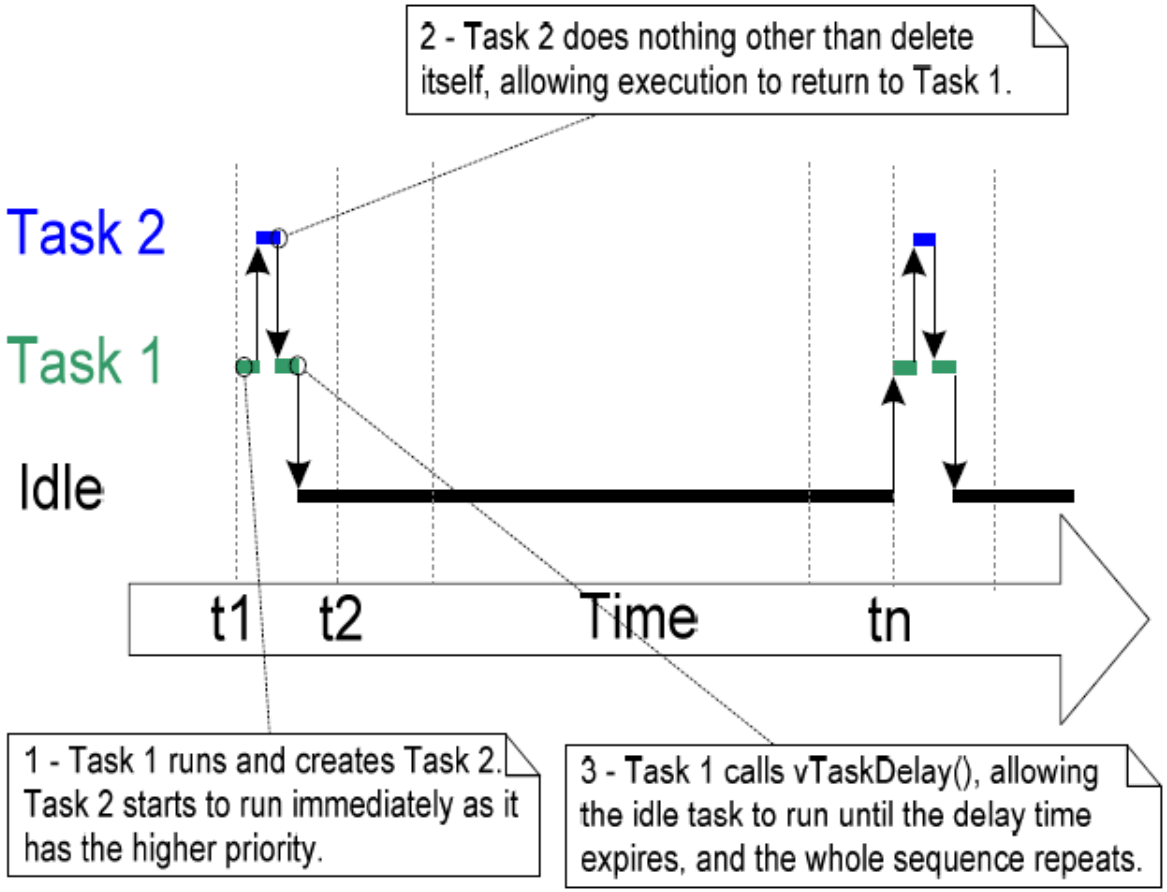
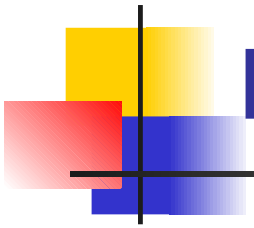


Figure 23. The execution sequence for Example 9



Prioritized Pre-emptive Scheduling

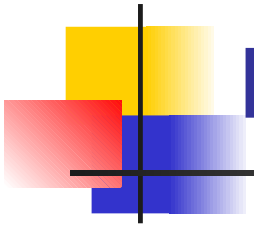
Los ejemplos ilustran cómo y cuándo FreeRTOS elije qué tarea debe estar en ejecución (**Running State**)

- Cada tarea tiene una prioridad asignada
- Cada tarea puede estar en uno de sus varios estados
- Sólo una tarea puede estar en ejecución (Running State) al momento
- El scheduler siempre elije la tarea de mayor prioridad lista (Ready State) para ponerla a ejecutar (Running State)

Se denomina: **Fixed priority Pre-emptive Scheduling**

Fixed Priority La prioridad asignada a c/tarea no puede ser modificada por el kernel por si solo, pero las tareas si pueden hacerlo

Pre-emptive Una tarea al estar lista o alterar su prioridad puede desplazar a la tarea en ejecución si ésta es de menor prioridad



Prioritized Pre-emptive Scheduling

Las tareas pueden permanecer detenidas a la espera de eventos (Blocked State) y al ocurrir un evento vuelven inmediatamente a estar listas (Ready State)

Eventos temporales ocurren a tiempos particulares (al expirar un tiempo de bloqueo). Se utilizan en general para implementar comportamientos periódicos, demoras o esperas

Eventos de sincronización ocurren cuando una tarea o rutina de atención de interrupción envía información a una cola (queue) o a uno o más semáforos (semaphores). Se utilizan en general para señalar actividad asincrónica, tal como el arribo de datos a un periférico

La figura 24 demuestra todos estos comportamientos ilustrando el patrón de ejecución de una hipotética aplicación

Prioritized Pre-emptive Scheduling

:

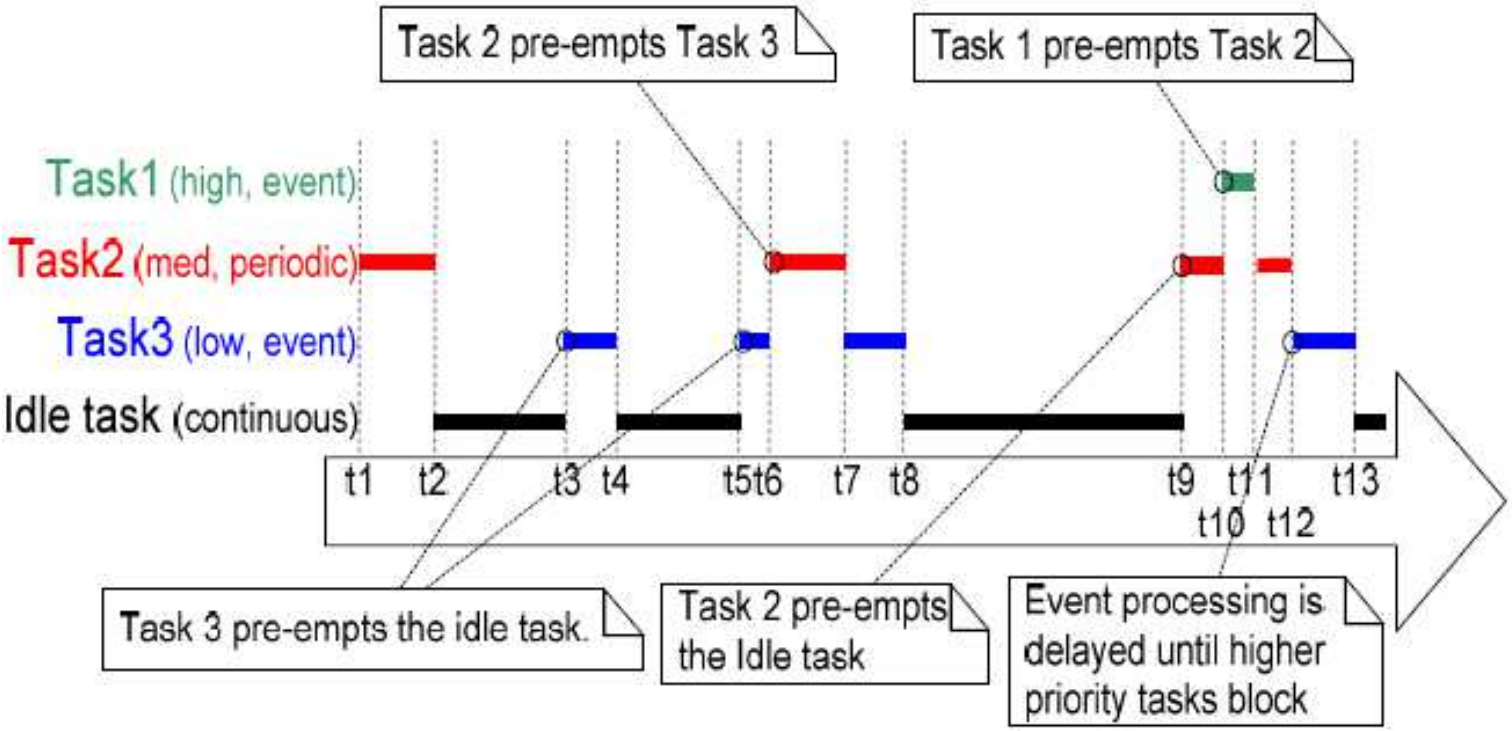
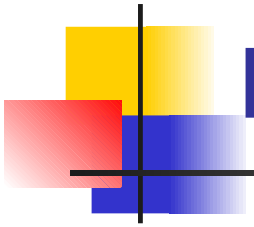


Figure 24. Execution pattern with pre-emption points highlighted



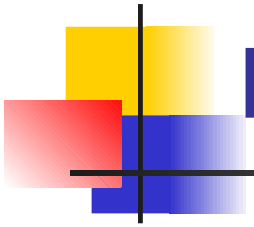
Prioritized Pre-emptive Scheduling

Idle Task Se ejecuta al menor nivel de prioridad, puede ser desplazada en cualquier momento cuando una tarea de mayor prioridad pasa a estar lista (Ready State), en t3, t5 y t9

Task3 Tarea manejada por evento (event-driven) de bajo nivel de prioridad (pero mayor que Idle). Permanece la mayor parte del tiempo detenida a la espera de eventos de su interés (Blocked State), pasando a estar lista (Ready State) cada vez que ocurren eventos

Todos los mecanismos de comunicación entre tareas de FreeRTOS (queues, semaphores, etc.) pueden usarse para señalar eventos y de este modo desbloquear tareas

Ocurren eventos en t3 y t5, y además algo ocurre entre t9 y t12 (es ejecutada cuando se convierte en la de mayor nivel prioridad lista)



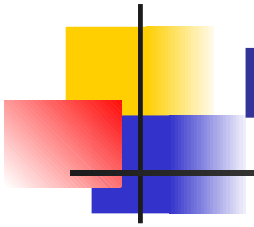
Prioritized Pre-emptive Scheduling

Task 2 Tarea periódica (event-driven) de medio nivel de prioridad (pero mayor que Task 3). Se quiere ejecutar a intervalos periódicos de tiempo (t1, t6 y t9)

Es ejecutada cuando se convierte en la de mayor nivel prioridad lista y desplazada al estar lista otra tarea de mayor prioridad

Task 1 Tarea manejada por evento (event-driven) de mayor nivel de prioridad (t10)

Asignación de Prioridades Como regla general, asignar mayor nivel a las tareas de cumplimento estricto (hard) respecto a las de cumplimiento no estricto (soft). Tener en cuenta tiempo de ejecución y utilización del procesador a fin de asegurar que la aplicación no supera el plazo de ejecución especificado



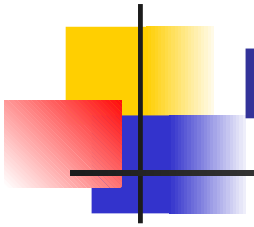
Prioritized Pre-emptive Scheduling

Rate Monolitic Scheduling (**RMS**) Se asigna una única prioridad a cada tarea de acuerdo a su frecuencia de ejecución periódica

La mayor prioridad se asigna a la tarea de mayor frecuencia y la menor a la de menor frecuencia. Está demostrado que esta manera de asignar prioridades maximiza la planificabilidad (schedulability), pero al ejecutar variaciones de tiempo y el hecho de que no todas las tareas son periódicas hacen de los cálculos absolutos un complejo proceso

Co-operative Scheduling FreeRTOS soporta esta configuración. El cambio de contexto ocurre cuando la tarea en ejecución se bloquea o llama explícitamente a taskYIELD(). Su operación es la más simple pero puede potencialmente resultar un sistema más lento en responder

Se puede implementar un sistema híbrido, donde las rutinas de atención de interrupción se encargan del cambio de contexto (c/eventos de sincronización, s/eventos temporales => Pre-emptivo s/time slicing)



Referencias

Using the FreeRTOS Real Time Kernel – A Practical Guide – NXP LPC17xx Edition, Richard Barry

Source Code for NXP LPC17xx Edition - Examples (.ZIP)

FreeRTOS Reference Manual – API Functions and Configuration Options, Richard Barry