FreeRTOS

CAPACITACION PIC32

Que es FreeRTOS?

- FreeRTOS es un sistema operativo en tiempo real (RTOS).
- Distribuido libremente bajo la licencia de código abierto del MIT.
- FreeRTOS incluye un kernel y un conjunto creciente de bibliotecas apropiadas para su uso en todos los sectores de la industria.

Entonces… qué es un RTOS?

- Un sistema operativo (SO) es un programa que administra los recursos de hardware y/o software (dependerá de la plataforma), y proporciona servicios comunes a las aplicaciones.
- Los programas en tiempo real (RT) deben garantizar una respuesta dentro de las limitaciones de tiempo especificadas.
- Un sistema operativo en tiempo real (RTOS) es un sistema operativo diseñado para aplicaciones en tiempo real.
- FreeRTOS es un RTOS para aplicaciones integradas con sistemas embebidos.

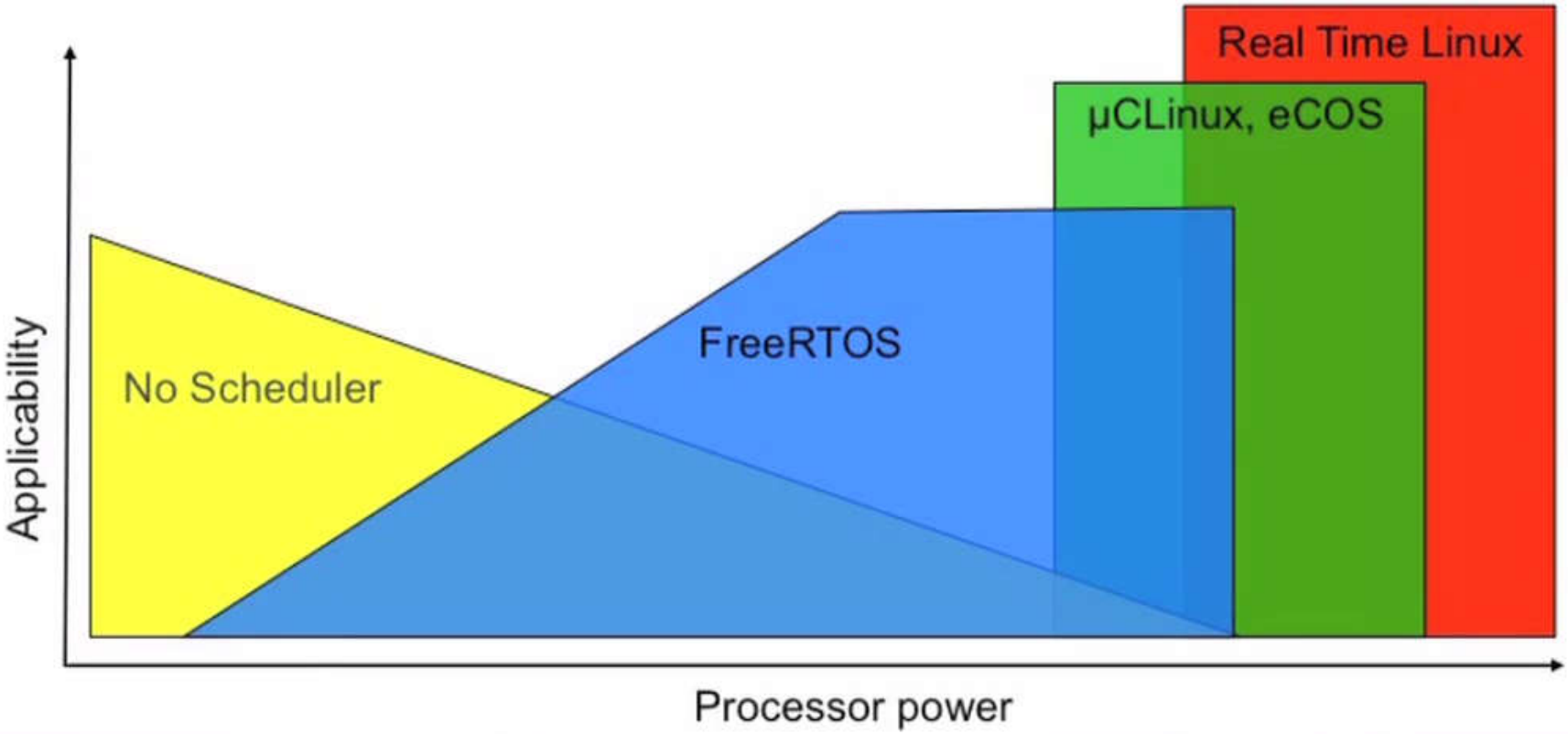
Ventajas y desventajas de un RTOS…

VENTAJAS

- Modularidad: el software se puede desarrollar en forma de una serie de tareas, cada una de las cuales tiene una función bien definida. La modularidad ayuda al desarrollo del equipo, la reutilización del código y las pruebas.
- El software puede ser totalmente impulsado por eventos, sin perder tiempo y energía al sondear eventos que no han ocurrido.
- Aplicaciones de gran capacidad de respuesta: los ISR se pueden hacer muy cortos si se difiere el procesamiento a una tarea.
- El desarrollo de aplicaciones puede ser más corto y sencillo cuando se utilizan las funciones proporcionadas por RTOS.

DESVENTAJAS

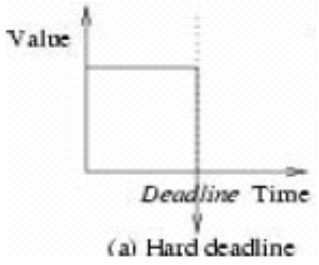
- Un sistema “bare metal” (computador que no contiene software y requiere de uno para que el hardware trabaje) no tiene sistema operativo. Un enfoque básico puede resultar en aplicaciones más eficientes, pero su desarrollo puede llevar más tiempo.
- Algunos recursos están reservados por el RTOS y ya no están disponibles.
- Una interrupción podría tardar mucho tiempo en realizarse demorando el resto del programa y produciendo un error.
- Ocupa memoria de programa y consume regular memoria de datos.


Real Time Linux
µCLinux, eCOS
FreeRTOS
No Scheduler
Applicability
Processor power

Clasificación de sistemas de tiempo real…

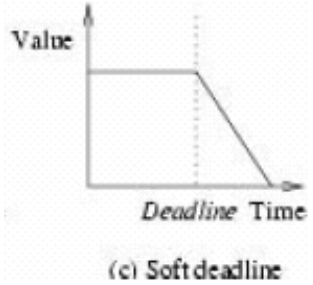
HARD RTS

- Es donde las tareas deben ser realizado no solo correctamente sino también a tiempo. Es decir, una falla catastrófica del sistema puede ocurrir si el sistema no responde al tiempo requerido.
- Por ejemplo, el controlador de turbulencia de la aeronave (inestabilidad en la atmósfera) debe responder a tiempo, de lo contrario, habrá un desastre definitivo.



SOFT RTS

- Es donde las tareas deben ser realizadas lo más rápido posible, pero no necesariamente que se termine dentro de los plazos.
- Puede ir más allá de los plazos especificados sin fallas catastróficas (por ejemplo, si el videojuego no responde a tiempo, es solo un retraso en el que el usuario puede no querer esperar, pero incluso entonces no sucederá nada terrible).



Terminología de un RTOS

- Synchronous(Sincrónico): los eventos ocurren en momentos predecibles en el flujo de control.
- Asynchronous(Asincrónico): interrupciones.
- Multitasking(Multitarea): El proceso de programar y cambiar la CPU entre varias tareas, una sola CPU cambia su atención entre varias tareas secuenciales. La multitarea maximiza la utilización de la CPU y también proporciona construcciones modulares de aplicaciones.
- Task(Tarea): una tarea se llama subproceso, es un programa simple que cree que tiene la CPU para sí mismo. Normalmente, cada tarea es un bucle infinito que puede estar en cualquiera de los 2 estados: EN EJECUCIÓN, NO EJECUTANDO.

Terminología de un RTOS

- Resource(Recurso): un recurso es cualquier entidad utilizada por una tarea. Por tanto, un recurso puede ser un dispositivo de E/S, como una impresora, un teclado o una pantalla, o una variable, una estructura o una matriz.
- Shared Resource (Recurso compartido): un recurso compartido es un recurso que puede ser utilizado por más de una tarea. Cada tarea debe obtener acceso exclusivo al recurso compartido para evitar la corrupción de datos. A esto se le llama Exclusión Mutua. Por ejemplo, una variable global puede ser utilizada por muchas tareas, o una impresora que podría ser compartida por "n" usuarios.
- Critical Section (Sección crítica): también llamada región crítica, es un código que debe tratarse de manera indivisible; una vez que la sección de código comienza a ejecutarse, no debe interrumpirse. Para garantizar esto, las interrupciones generalmente se deshabilitan antes de que se ejecute el código crítico y se habilitan cuando finaliza el código crítico.

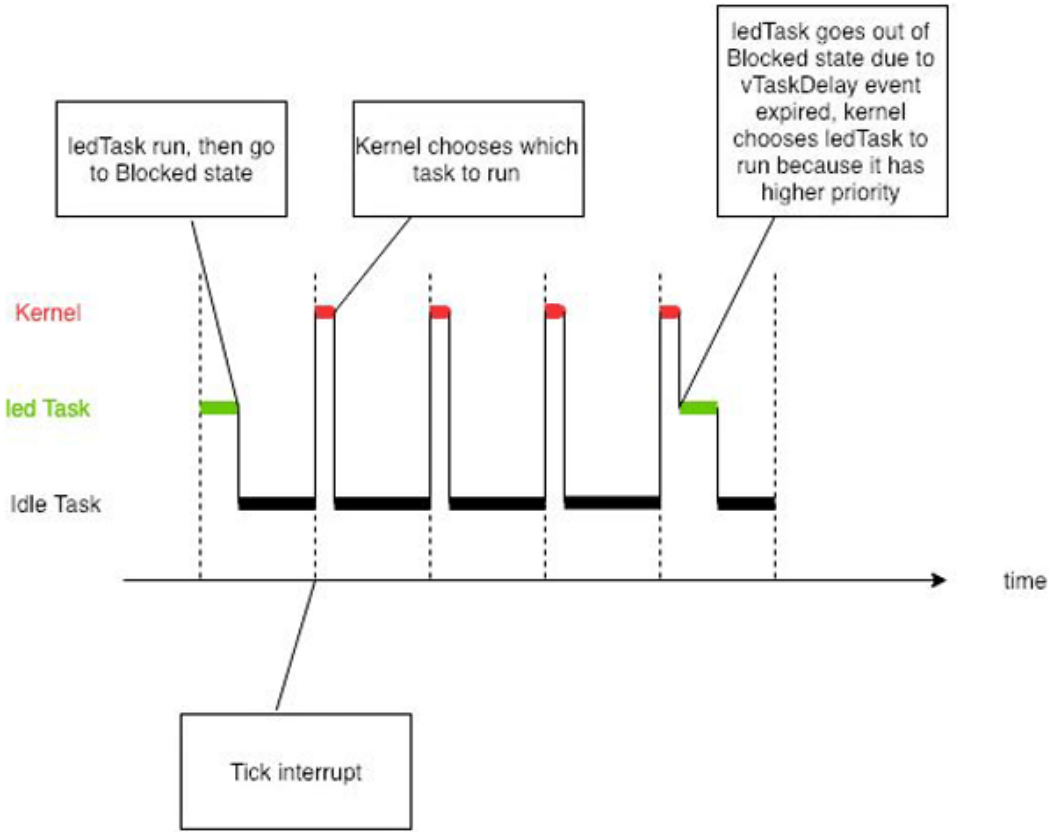
Terminología de un RTOS

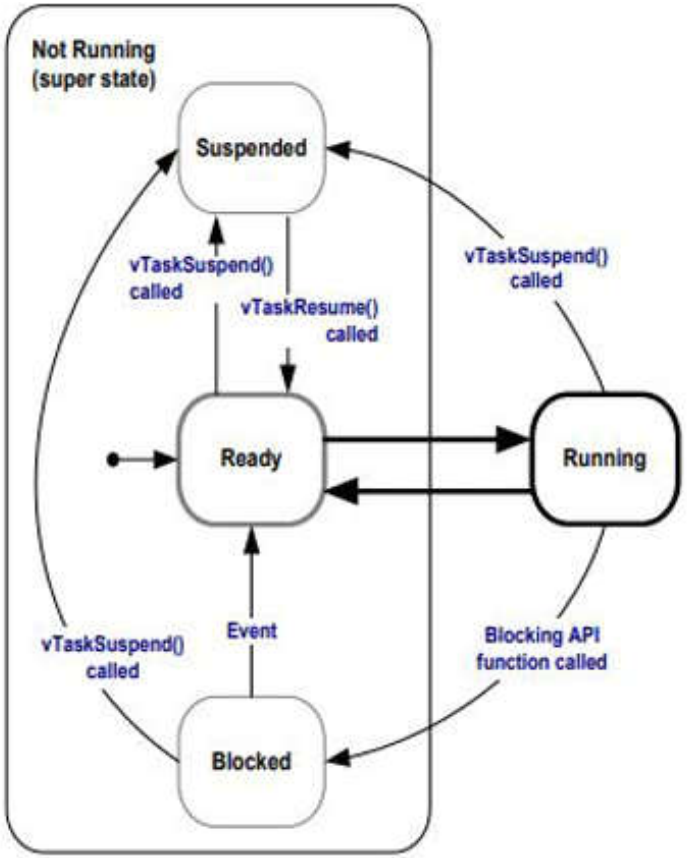
- Reentrancy (Reentrada): una función reentrante puede ser utilizada por más de una tarea sin temor a la corrupción de datos. una función reentrante puede interrumpirse en cualquier momento y reanudarse en un momento posterior sin pérdida de datos. Las funciones reentrantes usan variables locales (es decir, registros de CPU o variables en la pila) o protegen los datos cuando se usan variables globales.
- Kernel (Núcleo): Es el programa central del sistema operativo y determina su comportamiento. Es responsable de los recursos del sistema y del "cambio de contexto". También responsable de la programación de tareas. En RTOS, dado que el tamaño del kernel es pequeño, también se lo conoce como Micro Kernel.
- Task Scheduler (Programador de Tareas): El programador de RTOS en tiempo real determina cuándo debe ejecutarse cada tarea. Cada tarea se proporciona con su propia pila. Cuando una tarea se intercambia para poder ejecutar otra tarea, el contexto de ejecución de la tarea se guarda en la pila de la tarea de modo que puede restaurarse cuando esa misma tarea vuelva reanudar su ejecución más adelante.

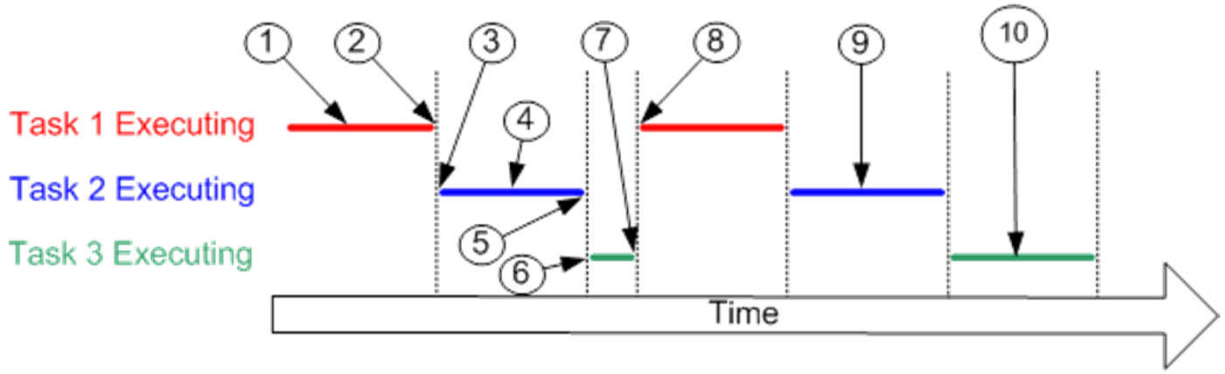
Tipos de programación de tareas (scheduling types)…

- Preemptive Scheduling (Programación preventiva): en este tipo de programación, las tareas se ejecutan con el mismo intervalo de tiempo sin tener en cuenta las prioridades.
- Priority-based Preemptive (Prevención basada en prioridades): la tarea de alta prioridad se ejecutará primero.
- Co-operative Scheduling (Programación cooperativa): el cambio de contexto ocurrirá solo con la cooperación de las tareas en ejecución. La tarea se ejecutará continuamente hasta que se llame al rendimiento de la tarea.

Cómo trabaja un RTOS…







- En (1) la tarea 1 se está ejecutando.
- En (2) el kernel suspende (intercambia) la tarea 1 ... ... y en (3) reanuda la tarea 2. Mientras se ejecuta la tarea 2 (4), bloquea un periférico de procesador para su propio acceso exclusivo.
- En (5) el kernel suspende la tarea 2 ... ... y en (6) reanuda la tarea 3. La tarea 3 intenta acceder al mismo periférico del procesador y encuentra que la tarea 3 bloqueada no puede continuar, por lo que se suspende en (7).
- En (8) el kernel reanuda la tarea 1. Etc.
- La próxima vez que se ejecute la tarea 2 (9) finaliza con el periférico del procesador y lo desbloquea.
- La próxima vez que se ejecute la tarea 3 (10), encuentra que ahora puede acceder al periférico del procesador y esta vez se ejecuta hasta que el kernel la suspende.

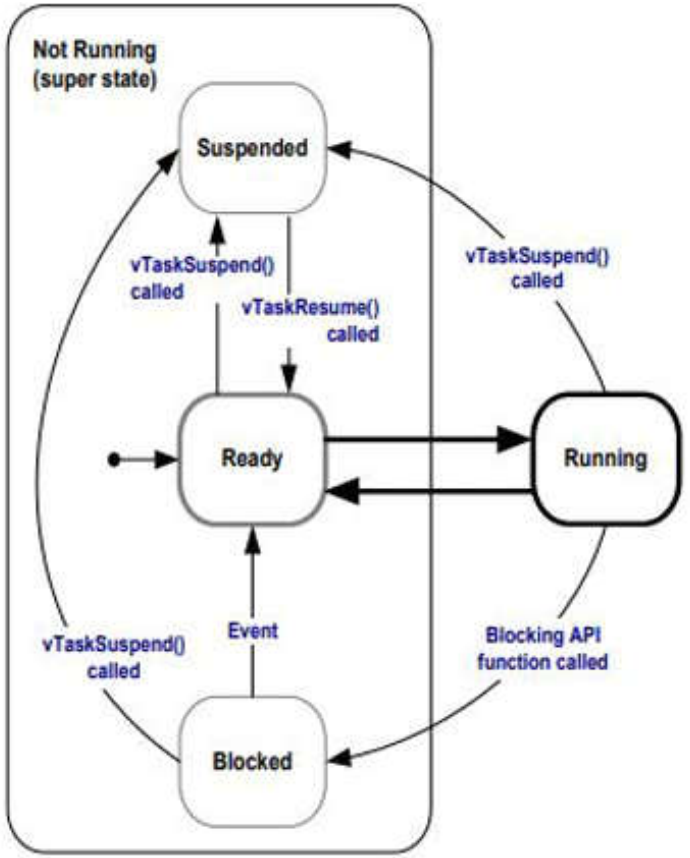
RTOS Tasks (tareas)…

- El usuario escribe el programa en forma de una serie de tareas.
- Una tarea es implementada por una función que nunca regresa.
- FreeRTOS proporciona un programador que garantiza que las tareas definidas por el usuario se ejecuten al mismo tiempo.
- El tiempo de la CPU se divide en porciones.
- Durante cada intervalo de tiempo, exactamente una tarea utiliza la CPU.
- De esta forma, las tareas que están listas para ejecutarse pueden compartir la CPU.

RTOS Tasks (tareas)…

Una tarea está en:

- Running: Ejecutando cuando se usa la CPU.
- Ready: Listo a la espera de la CPU.
- Blocked: Bloqueado al esperar un evento.
- Suspended: Suspendido cuando no está disponible para el programador.



Task Priorities (prioridad de tareas)…

- El usuario es libre de definir la prioridad de sus tareas.
- La prioridad de una tarea es un número entre 0 y configMAX_PRIORITIES – 1.
- Un número más alto indica una prioridad más alta.
- configMAX_PRIORITIES se puede ajustar en FreeRTOSConfig.h
- Además de las tareas definidas por el usuario, el RTOS también iniciará el Idle task y el Timer service task .
- Idle task tiene prioridad 0 (mínima). Entre sus labores intrínsecas tenemos: libera la memoria utilizada por las tareas eliminadas.
- Timer service task se utiliza (entre otros) para gestionar temporizadores definidos por el usuario. Por defecto, tiene máxima prioridad: configTIMER_TASK_PRIORITY

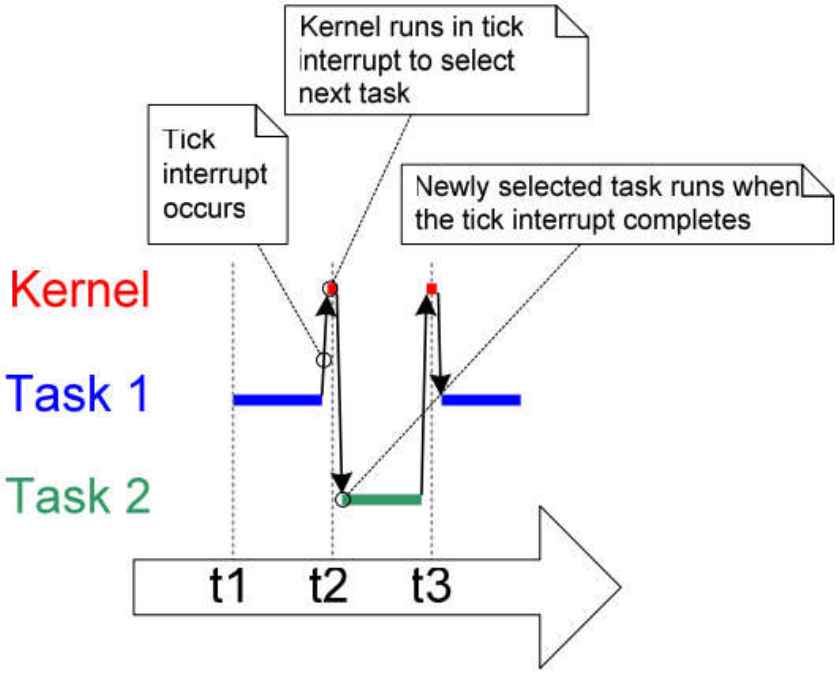
Task Priorities (prioridad de tareas)…

```
int main( void )
{
    /* Create the first task at priority 1.  The priority is the second to last
    parameter. */
    xTaskCreate( vTaskFunction, "Task 1", 240, (void*)pcTextForTask1, 1, NULL );

    /* Create the second task at priority 2. */
    xTaskCreate( vTaskFunction, "Task 2", 240, (void*)pcTextForTask2, 2, NULL );

    /* Start the scheduler so the tasks start executing. */
    vTaskStartScheduler();

    /* If all is well we will never reach here as the scheduler will now be
    running.  If we do reach here then it is likely that there was insufficient
    heap available for the idle task to be created. */
    for( ;; );
}
```

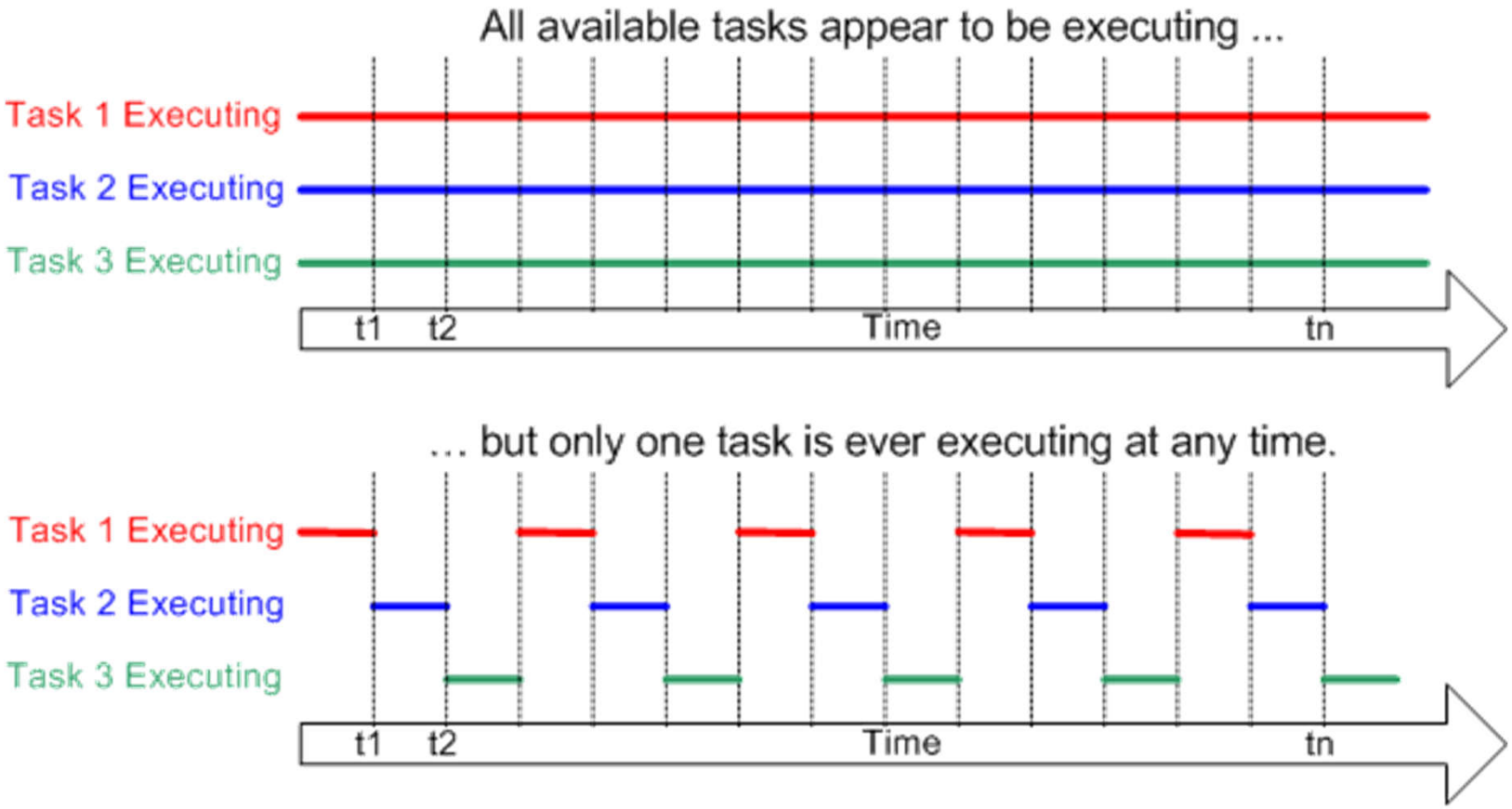


Task Priorities (prioridad de tareas)…

```
    /* Create OS Thread for APP1_Tasks. */
xTaskCreate((TaskFunction_t) _APP1_Tasks,//funcion a ejecutar
            "APP1_Tasks",//nombre de la tarea
            128,//tamano en words (4 bytes)
            NULL,//parametros
            1,//prioridad
            &xAPP1_Tasks);//handler
```

Task Scheduling timing (temporización)…

- Cada vez que una tarea esté lista, si tiene una prioridad más alta que la tarea en ejecución, se adelantará a ella y comenzará a ejecutarse inmediatamente.
- Al final de cada intervalo de tiempo, si hay tareas listas con la misma prioridad que la tarea en ejecución, se seleccionará una tarea lista y se ejecutará a continuación.
- Un segmento de tiempo es igual al intervalo entre dos interrupciones de SysTick. Por defecto: **configTICK_RATE_HZ** = 1000Hz


All available tasks appear to be executing ...
Task 1 Executing
Task 2 Executing
Task 3 Executing
t1
t2
Time
tn
… but only one task is ever executing at any time.
Task 1 Executing
Task 2 Executing
Task 3 Executing
t1
t2
Time
tn

RTOS HOOKS (funciones callback especiales)…

- Idle Hook Function: La tarea inactiva se ejecuta con la prioridad más baja, por lo que dicha función solo se ejecutará cuando no haya tareas de mayor prioridad que puedan ejecutarse. La función vApplicationIdleHook se llama repetidamente mientras se esté ejecutando la tarea inactiva. Es primordial que esta función no llame a ninguna función API que pueda hacer que se bloquee.

  void vApplicationIdleHook(void); // #define configUSE_IDLE_HOOK 1

- Tick Hook Function: Proporciona un lugar conveniente para implementar la funcionalidad del temporizador. La función vApplicationTickHook se ejecuta desde dentro de un ISR, por lo que debe ser muy corto, no usar mucha pila y no llamar a ninguna función API que no termine en "FromISR" o "FROM_ISR".

  void vApplicationTickHook(void); // #define configUSE_TICK_HOOK 1

RTOS HOOKS (funciones callback especiales)…

- Malloc Failed Hook Function: Definir el hook de falla de malloc () ayudará a identificar los problemas causados por la falta de memoria del HEAP asignado para el FreeRTOS, especialmente cuando falla una llamada a pvPortMalloc () dentro de una función de API.

  void vApplicationMallocFailedHook(void); // #define configUSE_MALLOC_FAILED_HOOK 1

- Stack Overflow Hook Function: El desbordamiento de pila es una causa muy común de inestabilidad de la aplicación. FreeRTOS, por lo tanto, proporciona dos mecanismos opcionales que pueden usarse para ayudar en la detección y corrección de tal ocurrencia. La opción utilizada se configura mediante la constante de configuración configCHECK_FOR_STACK_OVERFLOW. De los cuales se le suele asignar la opción 2 porque es más rápido que la opción 1 pero es menos eficiente.

  void vApplicationStackOverflowHook(TaskHandle_t xTask, signed char *pcTaskName);
  //#define configCHECK_FOR_STACK_OVERFLOW 2