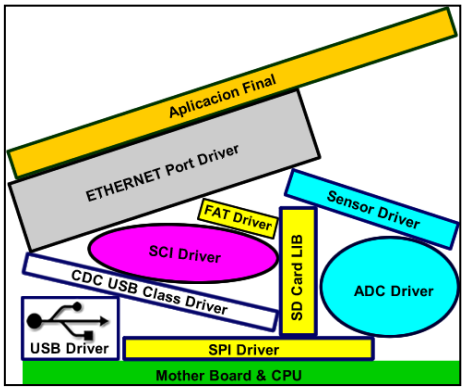
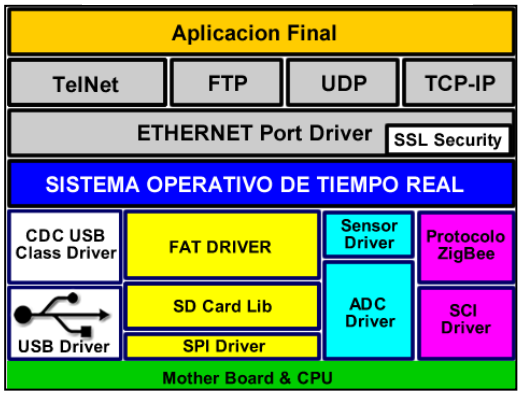
4.2 SISTEMAS OPERATIVOS DE TIEMPO REAL

Figura 13. Soluciones integrando un único proveedor y diferentes proveedores

Soluciones integrando herramientas de diferentes proveedores

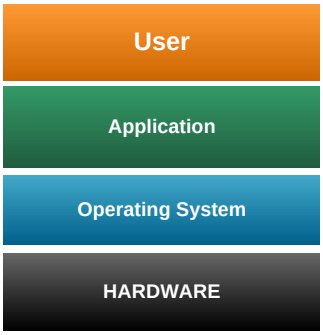


Solución integrando un único proveedor



4.2.1 ¿QUÉ ES UN SISTEMA OPERATIVO DE TIEMPO COMPARTIDO?

El término “sistema operativo” puede ser usado para describir una colección de software que gestiona de manera eficiente unos recursos de hardware.

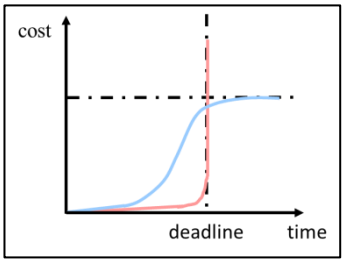


Características:

- Gestión de los recursos
- Interfaz entre la aplicación y el hardware
- Biblioteca de funciones para la aplicación

4.2.2 ¿QUÉ ES TIEMPO REAL?

Si una tarea debe ser completada en un momento dado, se dice que esa tarea debe ser ejecutada en tiempo real.



Tipos de sistemas de tiempo real:

- hard real time
- soft real time

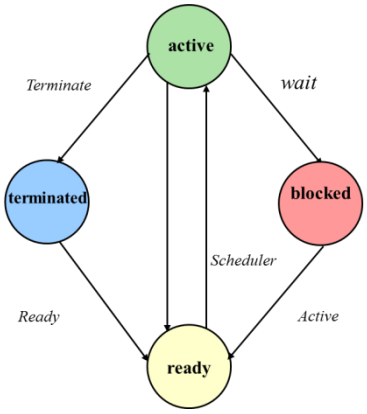
Un sistema operativo de tiempo real gestiona los tiempos en un microcontrolador o microprocesador.

Características:

- Soporta Multi-tareas.
- El planificador soporta tareas con prioridades.
- Sincroniza el acceso de las tareas a recursos que sean compartidos.
- Soporta comunicación entre tareas.
- Tiempos predecibles.
- Gestión de interrupciones.

4.2.3 TAREAS (TASK)

- Para dar solución a un proyecto MACRO usando RTOS, este es subdividido en pequeñas tareas.
- Estas tareas pueden ser ejecutadas de manera "simultanea" o en secuencia.
- Las tareas compiten entre sí por el acceso al procesador y los recursos como ADC, SCI, I2C, SPI, funciones de software, etc.



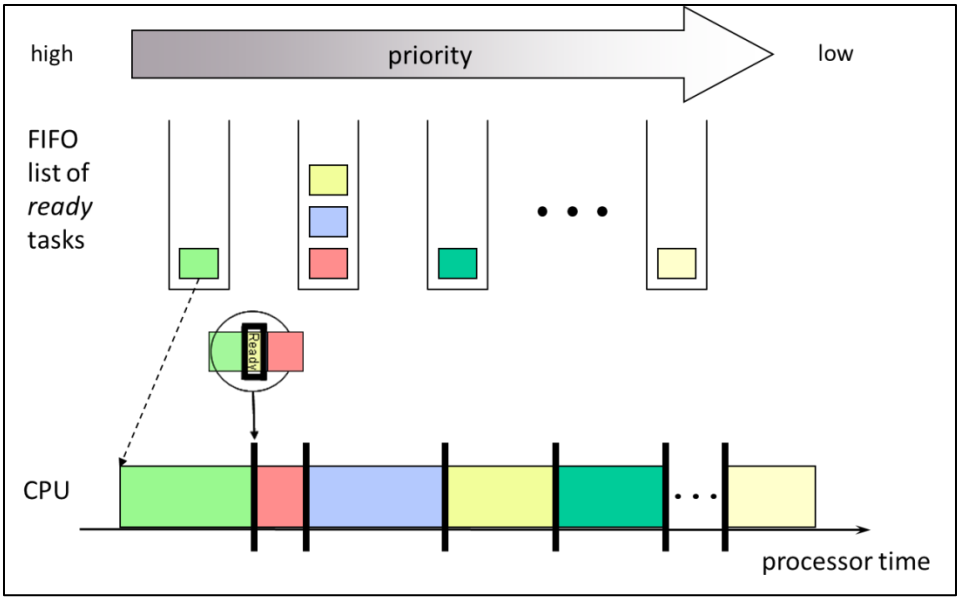
4.2.4 PLANIFICADOR (SCHEDULER)

El planificador es quien decide a que tarea se le asigna el procesador para que sea ejecutada.

- Planificador según secuencia de tareas a ejecutar:
  - Planificador Estático.
  - Planificador Dinámico.
- Políticas de planificación:
  - Non pre-emptive scheduling (no cooperative)
  - Pre-emptive scheduling
- Métodos de planificación.
  - Orden de llegada con control de prioridad (FIFO).
  - Tiempos de ejecución (Time slice - Round robin).

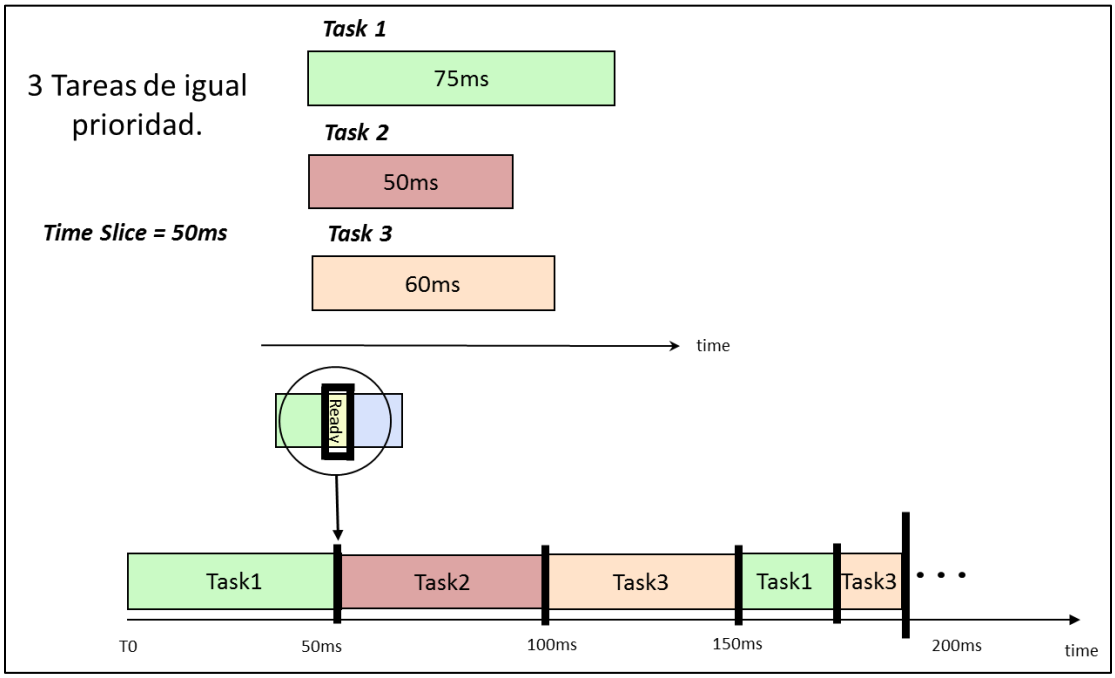
4.2.5 ORDEN DE LLEGADA CON CONTROL DE PRIORIDAD (FIFO)

Figura 14. Orden de llegada con control de prioridad



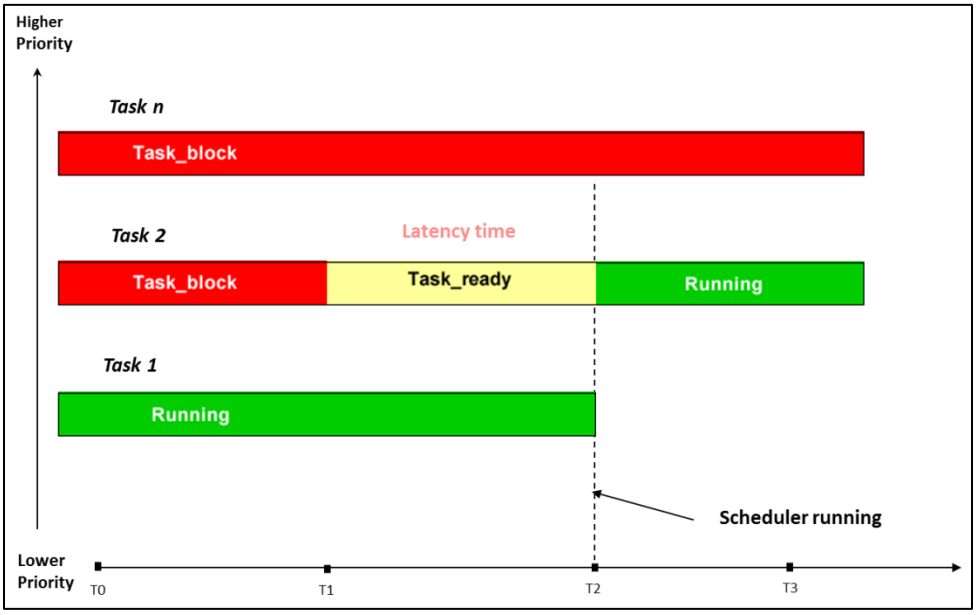
4.2.6 TIEMPO DE EJECUCIÓN (TIME SLICE - ROUND ROBIN)

Figura 15. Tiempo de ejecución



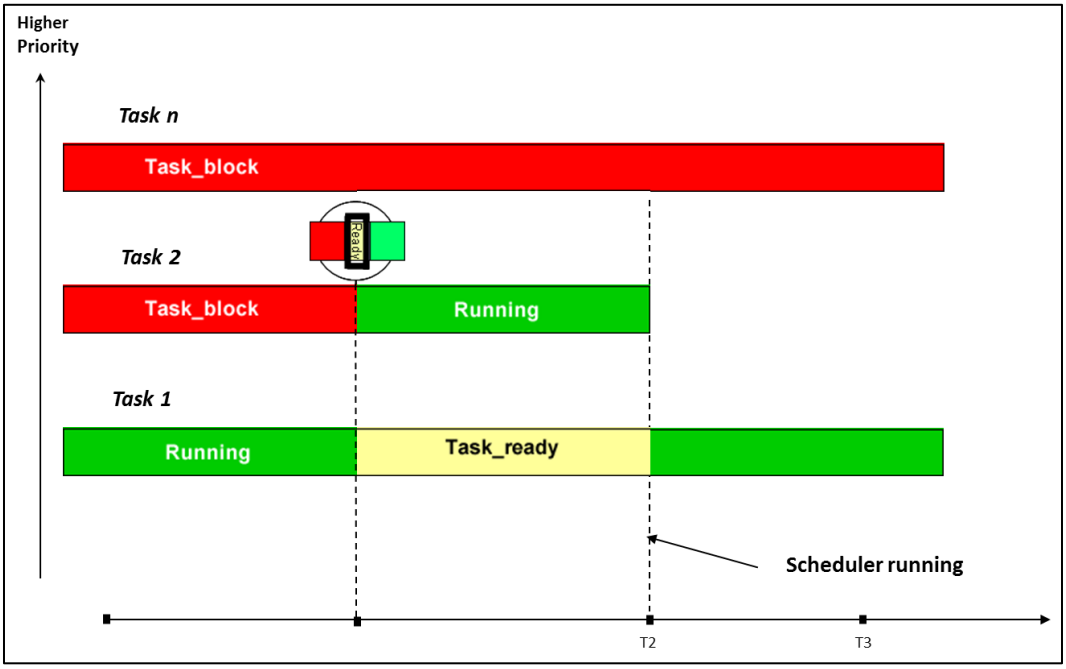
4.2.7 PLANIFICADOR NON PRE-EMPTIVE

Figura 16. Planificador Non Pre-emptive



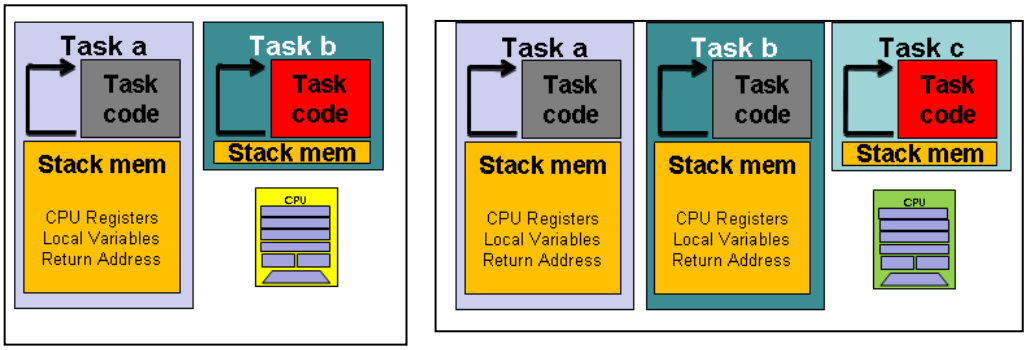
4.2.8 PLANIFICADOR PRE-EMPTIVE

Figura 17. Planificador Pre-emptive



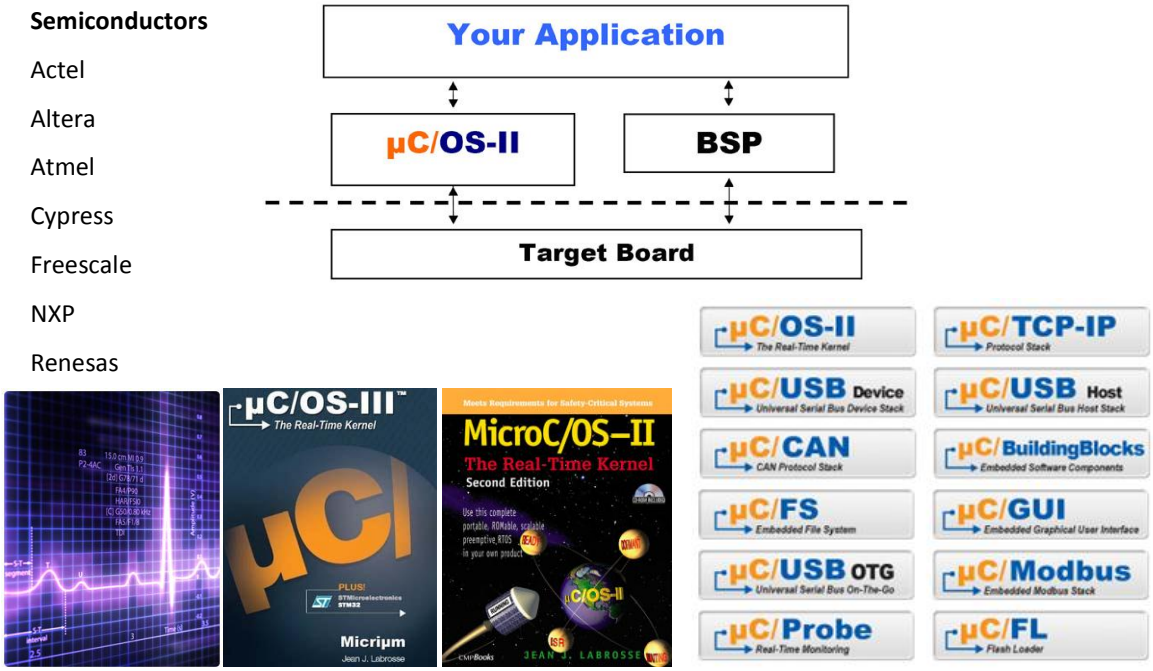
4.2.9 CAMBIOS DE CONTEXTO

Figura 18. Cambios de contexto



4.2.10 SISTEMA OPERATIVO DE TIEMPO REAL – UC/OS-II

Figura 19. Sistema Operativo de Tiempo Real – UC/OS-II



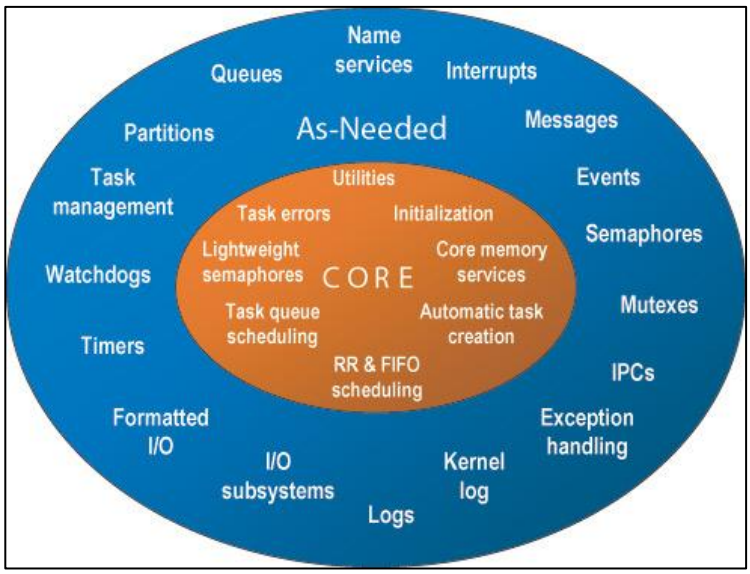
4.2.11 PLANIFICADOR– UC/OS-II

- Soporta las siguientes políticas:
  - FIFO (También llamado “priority-based preemptive”)
    - Se le da el procesador a la tarea de mayor prioridad que lleva más tiempo esperando.

- Round Robin **(NO SOPORTADO)**
    - Se le da el procesador a la tarea de mayor prioridad pero solo tiene un tiempo para que se ejecute, luego pasara a la siguiente tarea.
- *Explicit (optional)* ***(OPCIÓN LIMITADA)***
    - *Colas de tareas son usadas para decirle al planificador de manera explícita que tipo de política debe seguir con determinada tarea. Esto permite crear mecanismos más complejos de sincronización a los recursos.*

4.2.12 SISTEMA OPERATIVO DE TIEMPO REAL – MQX

Figura 20. Sistema operativo de tiempo real – MQX



Sistema Operativo de Tiempo Real – MQX

- **Todas las características que se puedan desear de un RTOS.**
    - Sistema operativo de Tiempo real, escalable, migrable y portable, apoyado con stacks USB y TCP-IP, además de otras librerías.

- **Propiedad de Freescale**
    - El código fuente es propiedad de Freescale y está totalmente disponible para usar con procesadores de su marca. El RTOS es código abierto.

- **Elimina una inversión inicial en software.**
    - Disponible en la WEB totalmente gratuito.

- Se pueden ahorrar hasta U$95.000, que es el valor comercial de un RTOS con similares características.

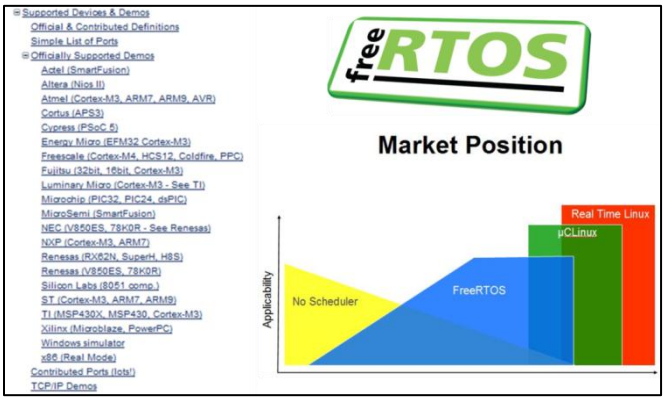
- **Comprado por más de 500 compañías en más de 15 años de experiencia.**

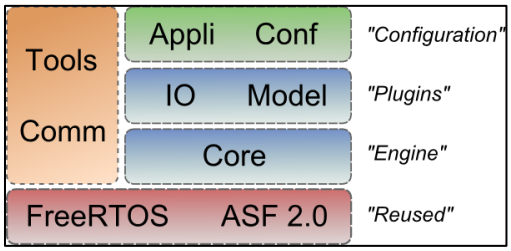
4.2.13 PLANIFICADOR– MQX

- Soporta las siguientes políticas:
  - FIFO (También llamado “priority-based preemptive”)
    - Se le da el procesador a la tarea de mayor prioridad que lleva más tiempo esperando.
  - Round Robin
    - Se le da el procesador a la tarea de mayor prioridad pero solo tiene un tiempo para que se ejecute, luego pasara a la siguiente tarea.
  - Explicit (optional)
    - Colas de tareas son usadas para decirle al planificador de manera explícita que tipo de política debe seguir con determinada tarea. Esto permite crear mecanismos más complejos de sincronización a los recursos.

4.2.14 SISTEMA OPERATIVO DE TIEMPO REAL – FREERTOS

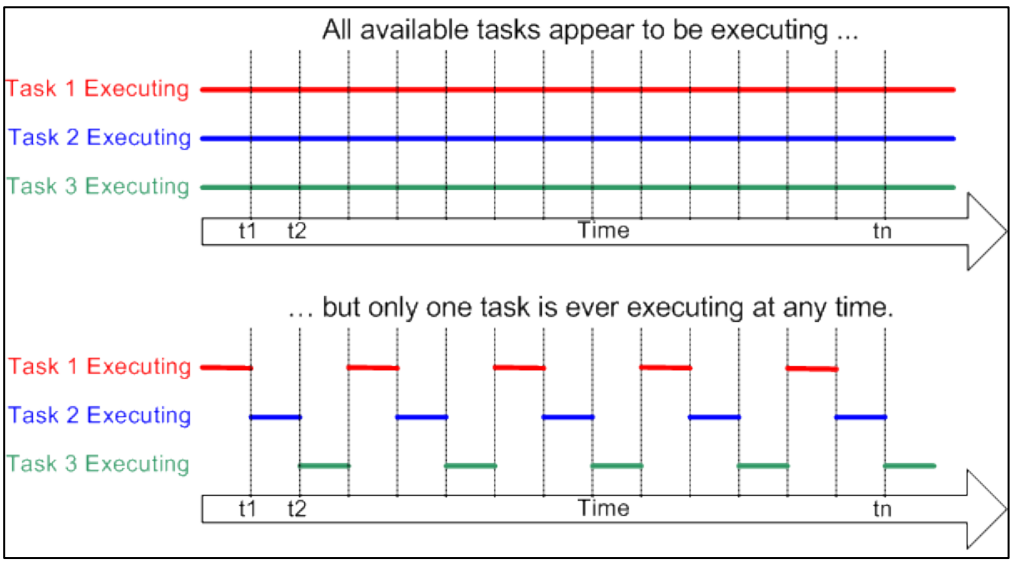
Figura 21. Freertos

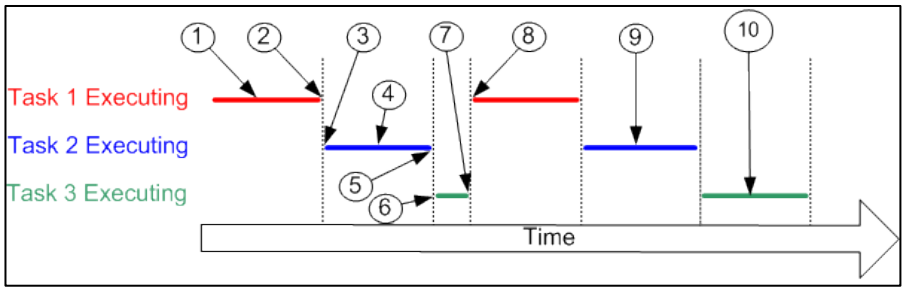




4.2.15 PLANIFICADOR– FREERTOS

Figura 22. Planificador Freertos

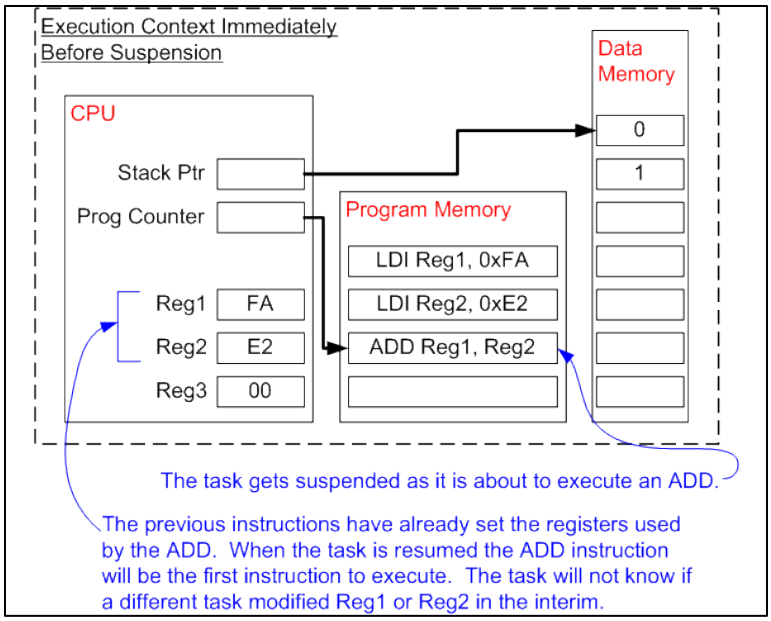




At (1) task 1 is executing.
At (2) the kernel suspends task 1 ...
... and at (3) resumes task 2.
While task 2 is executing (4), it locks a processor peripheral for its own exclusive access.
At (5) the kernel suspends task 2 ...
... and at (6) resumes task 3.
Task 3 tries to access the same processor peripheral, finding it locked task 3 cannot continue so suspends itself at (7).
At (8) the kernel resumes task 1.
Etc.
The next time task 2 is executing (9) it finishes with the processor peripheral and unlocks it.
The next time task 3 is executing (10) it finds it can now access the processor peripheral and this time executes until suspended by the kernel.

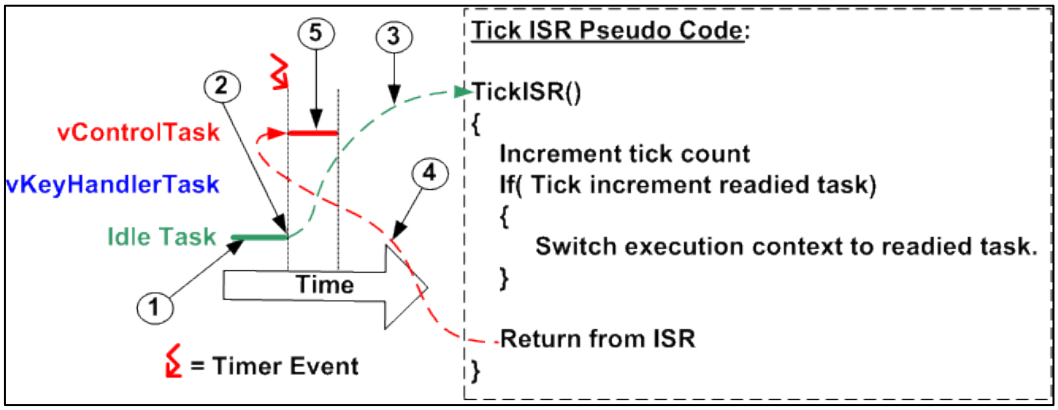
4.2.16 CAMBIOS DE CONTEXTO– FREERTOS

Figura 23. Cambios de contexto-Freertos



4.2.17 IMPLEMENTACIÓN DE – FREERTOS

Figura 24. Implementación de Freertos

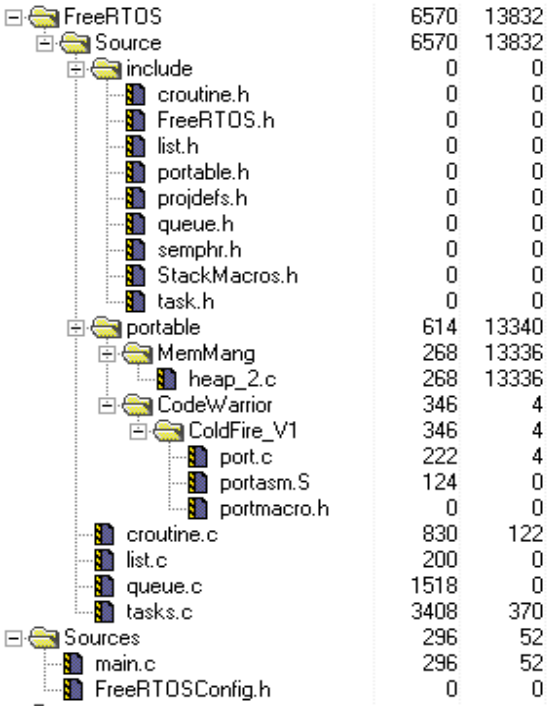


- At (1) the RTOS idle task is executing.
- At (2) the RTOS tick occurs, and control transfers to the tick ISR (3).
- The RTOS tick ISR makes vControlTask ready to run, and as vControlTask has a higher priority than the RTOS idle task, switches the context to that of vControlTask.
- As the execution context is now that of vControlTask, exiting the ISR (4) returns control to vControlTask, which starts executing (5).

A context switch occurring in this way is said to be Preemptive, as the interrupted task is preempted without suspending itself voluntarily.

4.2.18 ARCHIVOS FREERTOS

Figura 25. Archivos Freertos



En Carpeta "Include":
Archivos fuentes de sistema operativo programado en ANSI-C. Son archivos que serán usados en cualquier puerto del FreeRTOS.

En Carpeta "portable":
Archivos fuentes de diferentes puertos del FreeRTOS. Es en este espacio donde se encuentra programación específica para cada sistema embebido donde se desea usar FreeRTOS, incluye algunas veces programación en asm para mejorar rendimiento.

Archivo "FreeRTOSConfig.h":
Debe ser único para cada proyecto. Es en este.h donde se definen parámetros de rendimiento, permitiendo escalar el proyecto de manera rápida.

4.2.18.1 Crear Proyecto con FreeRTOS

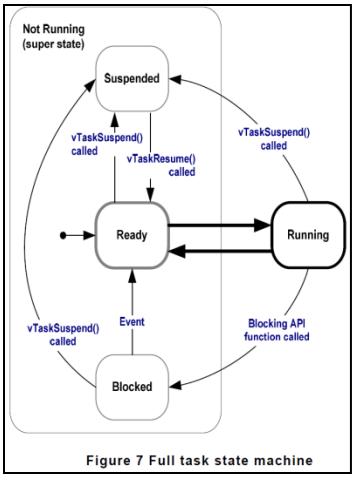
1. Descargar archivos fuente y ejemplos de FreeRTOS desde www.freertos.org
2. Instalar RTOS en PC, lo cual sólo genera una carpeta con todos los ejemplos y plantillas.
3. Buscar ejemplo con procesador que se desea usar y usar este ejemplo como plantilla.

Figura 26. Código fuente

```
#define configUSE_PREEMPTION                1
#define configUSE_IDLE_HOOK                 0
#define configUSE_TICK_HOOK                 0
#define configCPU_CLOCK_HZ                  ( 50000000UL )
#define configTICK_RATE_HZ                  ( ( portTickType ) 100 )
#define configMINIMAL_STACK_SIZE            ( ( unsigned portSHORT ) 100 )
#define configTOTAL_HEAP_SIZE               ( ( size_t ) ( 13 * 1024 ) )
#define configMAX_TASK_NAME_LEN             ( 12 )
#define configUSE_TRACE_FACILITY            1
#define configUSE_16_BIT_TICKS              1
#define configIDLE_SHOULD_YIELD             0
#define configUSE_CO_ROUTINES               0
#define configUSE_MUTEXES                   1
#define configCHECK_FOR_STACK_OVERFLOW      0
#define configUSE_RECURSIVE_MUTEXES         0
#define configQUEUE_REGISTRY_SIZE           0
#define configUSE_COUNTING_SEMAPHORES       0

#define configMAX_PRIORITIES    ( ( unsigned portBASE_TYPE ) 6 )
#define configMAX_CO_ROUTINE_PRIORITIES ( 2 )

/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */

#define INCLUDE_vTaskPrioritySet            1
#define INCLUDE_uxTaskPriorityGet           1
#define INCLUDE_vTaskDelete                 1
#define INCLUDE_vTaskCleanUpResources       0
#define INCLUDE_vTaskSuspend                1
#define INCLUDE_vTaskDelayUntil             1
#define INCLUDE_vTaskDelay                  1
#define INCLUDE_uxTaskGetStackHighWaterMark 1

/* It is not advisable to change these values on a Coldfire V1 core. */
#define configKERNEL_INTERRUPT_PRIORITY       1
#define configMAX_SYSCALL_INTERRUPT_PRIORITY  7
```

4.2.18.2 Funciones básicas FreeRTOS

Figura 27. Funciones básicas FreeRTOS



4.2.18.3 Funciones básicas FreeRTOS (Tareas)

Una tarea es un subproceso que debe gestionar el sistema operativo.

Este sub-proceso tendrá determinados requisitos de tiempo a cumplir por el RTOS.

Cada tarea tiene su ciclo cerrado o loop infinito. Solo se saldrá de este ciclo cuando la tarea sea destruida. Se debe tener en cuenta el tipo de scheduler seleccionado para así ceder la CPU a otras tareas.

Figura 28. Ejemplo tarea

```
void Tarea_Ejemplo1(void *pvParameters){

    for(;;){

        vTaskDelay(500 / portTICK_RATE_MS);  // Cede CPU por 500 milisegundos
    }

}
```

4.2.18.4 Funciones básicas FreeRTOS (Asignación de tareas a Planificador)

Figura 29. Ejemplo Asignación de Tareas a Planificador

```
portBASE_TYPE xTaskCreate( pdTASK_CODE pvTaskCode,
                           const signed portCHAR * const pcName,
                           unsigned portSHORT usStackDepth,
                           void *pvParameters,
                           unsigned portBASE_TYPE uxPriority,
                           xTaskHandle *pxCreatedTask
                           );
Ejemplo:

(void)    xTaskCreate(
            Tarea_Ejemplo1,
            ( signed portCHAR *) "Tarea_Ejemplo1",
            configMINIMAL_STACK_SIZE,
            (void *) 0,
            (tskIDLE_PRIORITY + 1),
            (xTaskHandle *) NULL
          );
```