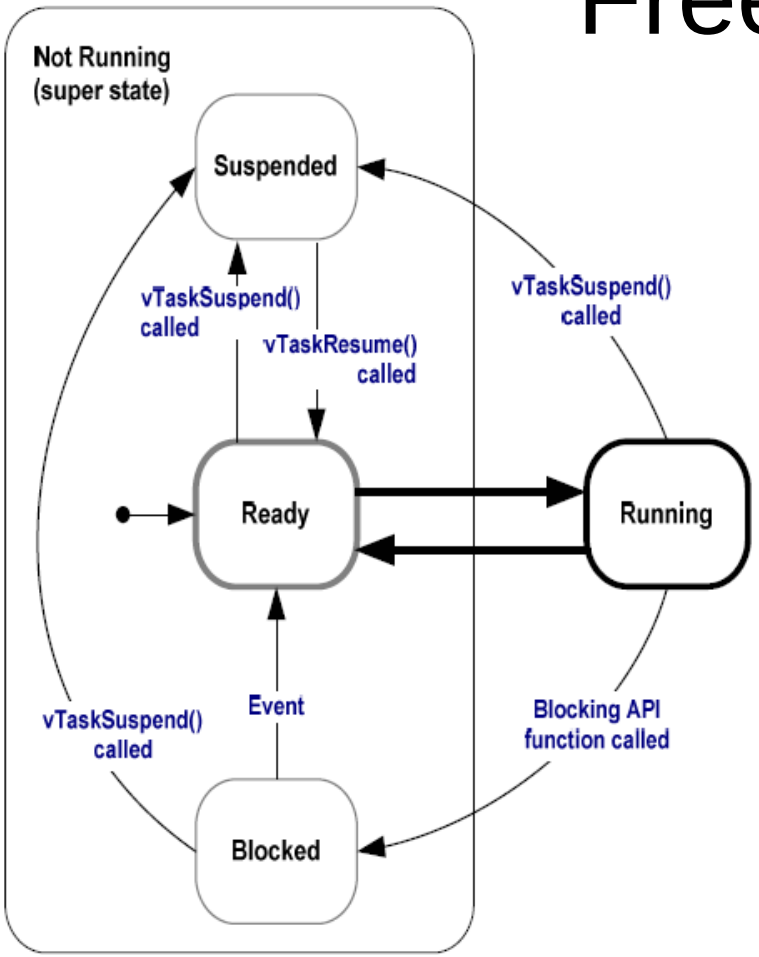
Configurando FreeRTOS.

Juan Alarcón.
jalarcon@electron.frba.utn.edu.ar

Agenda.

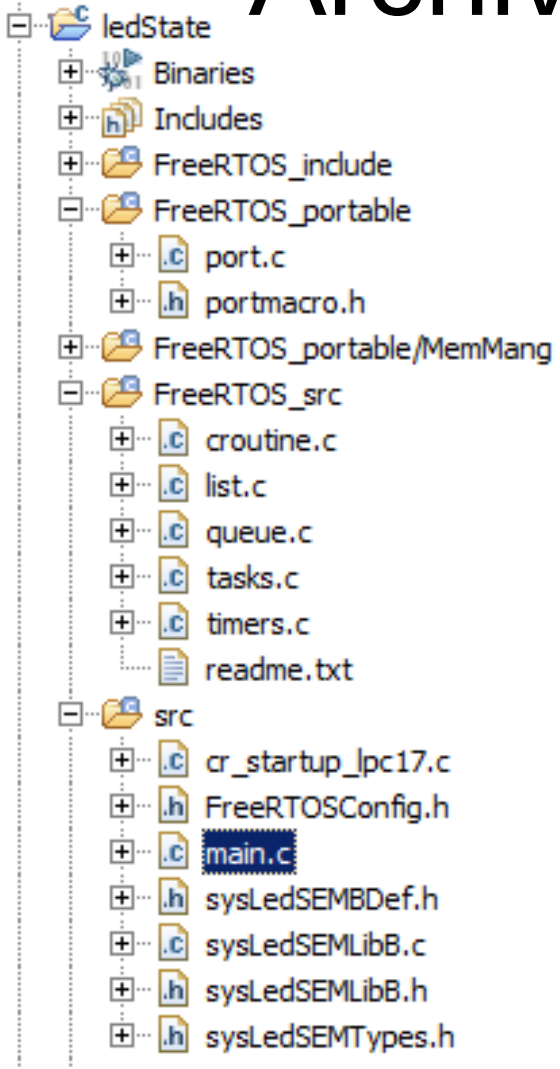
- Estados de las tareas en FreeRTOS (repaso).
- Archivos que componen FreeRTOS.
- Parámetros del sistema operativo
- ¿Qué pasa cuando se crea una tarea y se inicial el sistema operativo?.
- Esquemas de memoria de FreeRTOS.
- Uso del Heap en FreeRTOS.
- A practicar un poco!!!.

Estados de las tareas en FreeRTOS.



- El planificador de RTOS es un planificador (por defecto) del tipo apropiativo (preemptive).
- ***El planificador va a ejecutar la tarea de mayor prioridad que se encuentre en condiciones de ejecutarse.***

Archivos de FreeRTOS



- La mayoría del código del FreeRTOS va a estar contenido en tasks.c, queue.c y list.c
- Las funciones dependientes de la arquitectura van a estar en port.c
- El archivo de configuración de FreeRTOS es FreeRTOSConfig.h

Configurando FreeRTOS

```
#define configUSE_PREEMPTION        1
#define configUSE_IDLE_HOOK         0
#define configMAX_PRIORITIES        ( ( unsigned portBASE_TYPE ) 5 )
#define configUSE_TICK_HOOK         0
#define configCPU_CLOCK_HZ          ( ( unsigned long ) 99000000 )
#define configTICK_RATE_HZ          ( ( portTickType ) 1000 )
#define configMINIMAL_STACK_SIZE    ( ( unsigned short ) 80 )
#define configTOTAL_HEAP_SIZE       ( ( size_t ) ( 2 * 1024 ) )
#define configMAX_TASK_NAME_LEN     ( 12 )
#define configUSE_TRACE_FACILITY    1
#define configUSE_16_BIT_TICKS      0
#define configIDLE_SHOULD_YIELD     0
#define configUSE_CO_ROUTINES       0
#define configUSE_MUTEXES           1
```

Parámetros de FreeRTOS.

- **`configUSE_PREEMPTION`**: Sí esta constante vale 1 el sistema operativo se va a comportar de manera apropiativa con tareas de diferente prioridad y round robin con tareas de igual prioridad. Sí vale 0 el sistema se va a comportar de manera cooperativa. **`taskYIELD()`**;
- **`configUSE_IDLE_HOOK`**. Sí vale 1 cuando el sistema está ocioso llama a la función **`void vApplicationIdleHook(void)`**. Sí vale 0 no genera esta llamada.

Llamada a la Función ociosa (Idle Hook).

- Usos más comunes:
  - Medir el uso del procesador.
  - Ejecutar procesos de muy baja prioridad de manera continua.
  - Poner el procesador en bajo consumo.
- Limitaciones:
  - Nunca debe bloquearse o suspenderse. Ya que no hay otra tarea a la que recurrir si la idle se bloquea.
  - Si el sistema usa **`vTaskDelete()`** debe retornar de manera lo antes posible ya que la idle task debe liberar los recursos del sistema

```
void vApplicationIdleHook(void)
{
     haciendonada++;
}
```

Parámetros de FreeRTOS

- **`configMAX_PRIORITIES`**. Es un valor sin signo que define la cantidad de prioridades del sistema. A mayor uso de prioridades, más memoria se va a consumir ya que se va a generar una lista por cada nivel de prioridad.

```
PRIVILEGED_DATA static xList pxReadyTasksLists[ configMAX_PRIORITIES ];
```

Parámetros de FreeRTOS

- **`configUSE_TICK_HOOK`** . Sí vale 1 se llama a la función **`void vApplicationTickHook(void)`** el contenido es esta función debe ser muy breve usar muy poca pila y no llamar funciones del FreeRTOS que no terminen en “FromISR” o “FROM_ISR”.
- **`configCPU_CLOCK_HZ`**. Frecuencia del core del chip, es necesaria para que el FreeRTOS lleve el tiempo de sistema de manera correcta.
- **`configTICK_RATE_HZ`**. Configura cada cuanto interrumpe el sistema operativo, mientras mayor sea esta frecuencia, las tareas que se ejecuten en round robin conmutaran entre sí más rápidamente, pero el SO usará más tiempo del procesador.

Parámetros del FreeRTOS

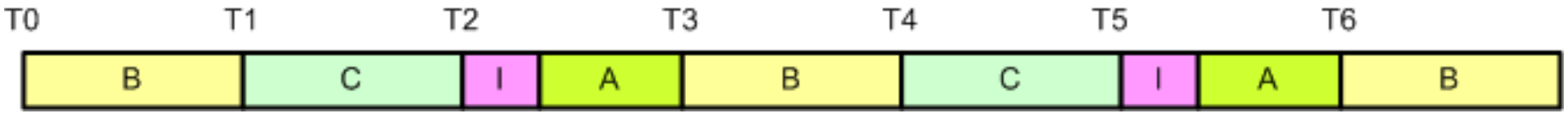
- **configMINIMAL_STACK_SIZE**. Define el tamaño de la pila para la tarea ociosa. Depende fuertemente de la arquitectura.
- **configTOTAL_HEAP_SIZE**. Define la cantidad de memoria en bytes que va a estar disponible para el sistema operativo.
- **configMAX_TASK_NAME_LEN**. Cuando se crea una tarea uno de los parámetros es el nombre de la tarea, esta constante define el largo máximo de estos nombres.
- **configUSE_TRACE_FACILITY**. Si está en 1 se activan todos las estructuras y funciones de ayuda a la visualización y depuración.

Parámetros del FreeRTOS

- **`configUSE_16_BIT_TICKS`**. Sí está en 1 la variable que cuenta los ticks del sistema se define de 16 bits. Sí está en 0 se define como una variable de 32 bits. Hay que tener en cuenta que (considerando una frecuencia de tick de 250Hz) en 16 bits se puede bloquear 262 segundos mientras que en 32 bits puede bloquear 17179869 segundos.

Parámetros del FreeRTOS

- **`configIDLE_SHOULD_YIELD`**. Este parámetro configura el comportamiento de la tarea ociosa. Para que esto funcione el planificador debe funcionar en modo apropiativo y deben existir tareas definidas con prioridad de la tarea ociosa. Sí se configura en 1 la tarea ociosa “roba” tiempo de las tareas con las que comparte prioridad. Se recomienda que este parámetro se ponga a cero y usar **`vApplicationIdleHook(void)`**.



Otros parámetros.

```
/* 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
```

¿Qué pasa cuando se crea una tarea?

- Se busca en memoria un bloque que contenga el TCB (Task Control Block) y la pila.
- Se calculan los punteros a la pila.
- Se inicializa la pila
- Se inicializa el TCB, con su nombre y prioridad.
- Si es la primera de todas las tareas, se inicializan las listas de prioridades.
- Se incluye la tarea en la lista de prioridad correspondiente.
- Se marca la tarea como lista para correr

Inicializando el sistema.

```
int main(void)
{
    setupHardware();
    xTaskCreate(     vUserTask1,
                    ( signed portCHAR * ) "Task1",
                    USERTASK_STACK_SIZE,
                    NULL,
                    tskIDLE_PRIORITY+1,
                    NULL );

    vTaskStartScheduler();
    return 1;
}
```

- Al llamar a la función vTaskStartScheduler() van a suceder las siguientes cosas:
  - Se va a generar la tarea IDLE.
  - Se ponen los ticks a cero.
  - Se inicializan las interrupciones y el SysTick
  - Se llama a la SVC 0 para cargar la primer tarea.

Memoria en FreeRTOS

- Cuando se crea una tarea, se determina la cantidad de bytes de la misma. Se limita la cantidad de anidamientos de llamadas a función y la cantidad de ***variables locales.***
- Para utilizar memoria de la zona del heap sólo se puede consumir y liberar utilizando ***pvPortMalloc()*** y ***pvPortFree()*** respectivamente.
- Las variables globales se van a reservar en ***tiempo de compilación***.

Configuración de la memoria en FreeRTOS.

- FreeRTOS va a necesitar memoria para cada tarea (almacenar contexto) y para muchos de los recursos del sistema (semáforos y colas). Por lo que se va a requerir un esquema de manejo de memoria.
- FreeRTOS define tres esquemas de manejo de memoria, que van a depender fundamentalmente de la arquitectura en donde se corre el RTOS.

Configuración de la memoria.

- FreeRTOS va a necesitar memoria para cada tarea (almacenar contexto) y para muchos de los recursos del sistema (semáforos y colas, a ver en próximas clases). Por lo que se va a requerir un esquema de manejo de memoria.
- FreeRTOS define tres esquemas de manejo de memoria, que van a depender fundamentalmente de la arquitectura en donde se corre el RTOS.

Configuración de la memoria (1)

- El primer esquema de memoria, definido en heap_1.c es el más simple de los tres.
- No permite que la memoria sea liberada una vez que se asigna.
- El algoritmo se basa en dividir un bloque de memoria RAM en que se divide y se asigna a cada tarea.
- El tamaño del buffer está definido por el parámetro **`configTOTAL_HEAP_SIZE`**
- Se suele utilizar en arquitectura simples, como los 8051, AVR, PIC, etc.

Configuración de la memoria (2).

- Este esquema de memoria está definido en heap_2.c
- Este esquema define un bloque de memoria de tamaño **`configTOTAL_HEAP_SIZE`** al igual que el modelo anterior.
- Este modelo permite asignar bloques de memoria utilizando el método del mejor ajuste y permite liberar bloques de memoria. No permite unir dos bloques de memoria contiguos liberados anteriormente.
- Hay que tener en cuenta que el algoritmo del mejor ajuste asigna el bloque de memoria más chico que cumpla con la cantidad de bytes que se necesitan. Siempre se pasa por toda la lista.
- Es el esquema utilizado por defecto para el LPC1769.

Configuración de la memoria (3)

- El tercer esquema de configuración de la memoria se basa en encapsular las funciones malloc() y free() tradicionales.
- Requiere que el linker genera el heap y compilar el código de malloc() y free(). Suele incrementar el tamaño del kernel.
- Es no deterministico.
- Se suele utilizar en arquitecturas x86 (PC)

Configuración de memoria. Interface.

```
/*
 * Map to the memory management routines required for the port.
 */
void *pvPortMalloc( size_t xSize ) PRIVILEGED_FUNCTION;
void vPortFree( void *pv ) PRIVILEGED_FUNCTION;
void vPortInitialiseBlocks( void ) PRIVILEGED_FUNCTION;
size_t xPortGetFreeHeapSize( void ) PRIVILEGED_FUNCTION;
```

Uso de memoria.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int x[4];

int main(void)
{
    int a,b;
    int *ptr;
    ptr = (int*)malloc(64*sizeof(int));
    return 0;
}
```

- ¿Cuántos bytes de stack se consumen?
- ¿Cuánto heap se utiliza?
- ¿Dónde se almacenan las variables globales?

Uso de la memoria LPCXpresso

```
make --no-print-directory post-build
Performing post-build steps
arm-none-eabi-size "LedRTOS.axf"; # arm-none-eabi-ob
LPC1769 -d "LedRTOS.bin";
   text    data     bss     dec     hex filename
   6776      16    2356    9148    23bc LedRTOS.axf
```

- text representa la cantidad de memoria de programa (6776 bytes en este caso).
- data indica cantidad de memoria de lectura escritura inicializada explícitamente.
- bss indica la cantidad de memoria de lectura, escritura inicializada en cero (este programa utiliza un total de 2362 bytes de RAM).

Memoria en LPC1769.

- El LPC1769 tiene 64Kbytes de memoria divididos en dos zonas de 32Kbytes.
- El LPCXpresso los define como:
  - RamLoc32 que empieza en 0x1000000
  - RamAHB32 que empieza en 0x2007C000
- Por defecto el LPCXpresso ***utilizará sólo*** la memoria de ***RamLoc32.***
- Para utiliza ***RamAHB32*** es necesario asignarla de ***manera explícita***, utilizando las macros de *“cr_section_macros.h”*

Heap en FreeRTOS.

```
static union xRTOS_HEAP
{
    #if portBYTE_ALIGNMENT == 8
        volatile portDOUBLE dDummy;
    #else
        volatile unsigned long ulDummy;
    #endif
    unsigned char ucHeap[ configTOTAL_HEAP_SIZE ];
} xHeap;
```

- Está definido en el archivo de administración de memoria.
- Por defecto para el LPC1769 es el heap_2.c

Pasando el heap a la memoria alta.

```
#include "FreeRTOS.h"
#include "task.h"
#include "cr_section_macros.h"

#undef MPU_WRAPPERS_INCLUDED_FROM_API_FILE

/* Allocate the memory for the heap.  The struct is used to force byte
alignment without using any non-portable code. */
__BSS(RAM2) static union xRTOS_HEAP
{
    #if portBYTE_ALIGNMENT == 8
        volatile portDOUBLE dDummy;
    #else
        volatile unsigned long ulDummy;
    #endif
    unsigned char ucHeap[ configTOTAL_HEAP_SIZE ];
} xHeap;
```

A trabajar.

- Tomar el programa LedRTOS, modificarlo para que el led siga parpadeando y se genere una señal cuadrada de 100ms en el pin 2.13 utilizando código generado de manera automática por la herramienta de StateCharts.

Bibliografía.

- Sistemas Operativos. Diseño e Implementación. Andrew S. Tanenbaum.
- Sistemas operativos modernos. Andrew Tanembaum. Segunda Edición.
- Using the FreeRTOS Real Time Kernel. NXP LPC17xx Edition. Richard Barry.
- FreeRTOS http://www.freertos.org/