Carrera de Especialización en Sistemas Embebidos

Sistemas Operativos en Tiempo Real

Clase 2: Temporización

ACSE
Asociación Civil para la Investigación, Promoción y Desarrollo de los Sistemas Electrónicos Embebidos
FACULTAD DE INGENIERIA
Universidad de Buenos Aires
CIM
CADIEEL

RTOS: Aspectos fundamentales

- Sabemos, de la clase anterior, que los RTOS se encargan de gestionar los recursos de nuestro SE, destacando:
  - Tiempo de procesador
    Generando la ilusión de **multitareas** y asegurando la ejecución de cada tarea, de acuerdo con su disponibilidad y prioridad, en un **plazo de tiempo acotado.**
  - Memoria
    Reservando los espacios de memoria de cada tarea y asegurando el almacenamiento del **Contexto de Ejecución.**

RTOS: Aspectos fundamentales

- Además los RTOS nos ofrecen herramientas de programación tales como:
  - Temporización
    Permitiendo manejar esperas y pausas sin gestionar manualmente temporizadores de hardware ni contadores de software.
  - Sincronización
    De tareas con eventos externos y con otras tareas.
  - Comunicación
    De tareas con manejadores de periféricos u otras tareas en forma segura y ordenada.
  - Gestión de otros recursos de Hardware.

Comenzando con FreeRTOS

- Para comenzar a trabajar con FreeRTOS, podemos utilizar como base el ejemplo del blinky.

- Una vez que hemos **diseñado** nuestro sistema para la aplicación a implementar, escribiremos como funciones, las tareas que serán ejecutadas por el SO utilizando el siguiente prototipo:

```
void tarea(void *pvParameters);
```

- Finalmente, será necesario inicializar el Hardware, crear las tareas en el contexto del sistema operativo e inicializar el RTOS. Todo esto, en el main de nuestro programa en C.

Ejercicio B1 - Demoras Fijas

- Para comenzar implementaremos un **Heartbeat** en el led azul de la EDU-CIAA, utilizando FreeRTOS, donde el período sea de **1s** y el led cambie de estado cada **500ms**, como se muestra en el siguiente gráfico:

Ejercicio B1 - Demoras Fijas

- Para implementar este Heartbeat necesitaremos:

  - Tarea de control de encendido/apagado del led
    Que deberá activarse cada 500ms encendiendo el led al primer llamado y apagándolo al segundo, repitiéndose cíclicamente.

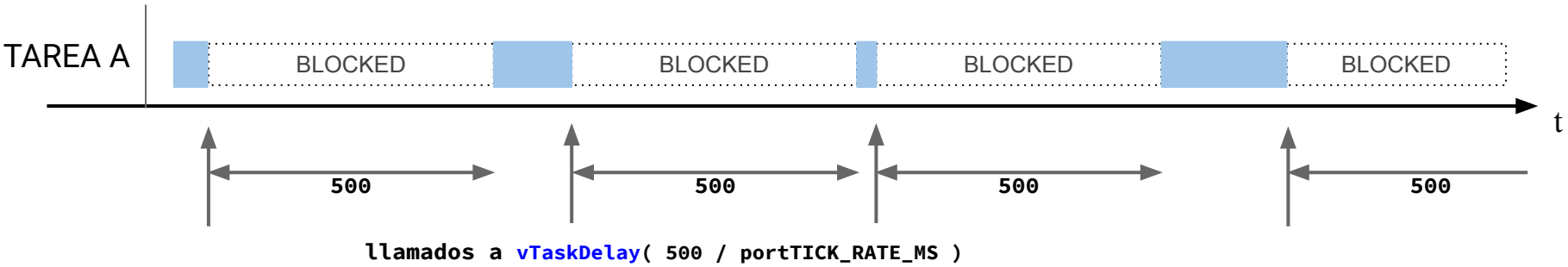
  - Un temporizador
    Que asegure la llamada de la tarea cada 500ms y que permita al sistema liberar el procesador entre cada llamada de la tarea, una vez que ésta haya finalizado su labor.

    Para este último utilizaremos una API de FreeRTOS que puede ser llamada desde la tarea, a la cual bloquea durante el tiempo seteado:
    `void vTaskDelay( const TickType_t xTicksToDelay );`

API: **Delay**

- `void` **`vTaskDelay`**`( const TickType_t` **`xTicksToDelay`** `)`
  - Produce una demora en la tarea que la llama. Cede el control del CPU mientras este tiempo no expira.

```
void TareaA( void* params )
{
	while(1)
	{
		/*
		CODIGO DE PROCESO "PERIODICO" QUE TARDA EN EJECUTARSE (SIN INTERRUPCIONES ENTRE 1 Y 200 TICKS)
		*/
		vTaskDelay( 500 / portTICK_RATE_MS );
	}
}
```



Ejercicio B1 - Demoras Fijas

- Con esta herramienta, la tarea podría ser escrita de la siguiente forma:

```
void Heartbeat( void* taskParmPtr )
{
    uint8_t LedState = 0;
    while(TRUE)
    {
        // Escribe el estado en el Led
        gpioWrite( LEDB, LedState );
        // Intercambia el estado del Led Azul para la próxima ejecución

        if ( 0 == LedState )
            LedState = 1;
        else
            LedState = 0;
        // Bloquea la tarea durante 500ms, liberando el uso del CPU
        vTaskDelay( 500 / portTICK_RATE_MS );
    }
}
```

Ejercicio B1 - Demoras Fijas

- Nótese que la definición de la tarea contiene una inicialización y luego una rutina encerrada en un loop.

  Gracias a que FreeRTOS es apropiativo, esto nos permite escribir cada tarea como si fuera la única en ser ejecutada por el procesador.

  Una vez definida cada una de las tareas solamente nos queda preparar el main, con la inicialización del Hardware, la creación de la tarea y la inicialización del SO.

Ejercicio B1 - Demoras Fijas

- Escribimos el main de la siguiente manera:

```
int main(void)
{
   // Inicializamos la EDU-CIAA
   boardConfig();

   // Creamos la tarea en freeRTOS
   xTaskCreate(Heartbeat, (const char *)"Heartbeat", configMINIMAL_STACK_SIZE*2,
                                                  NULL, tskIDLE_PRIORITY+1, NULL);
   // Iniciar scheduler
   vTaskStartScheduler();

   //Aseguramos que no salga del programa principal, en caso de un error de
inicialización
   while( TRUE ) {  }
   return 0;
}
```

Ejercicio B1 - Demoras Fijas

- Resta compilar y cargar en la EDU-CIAA

¡MANOS A LA OBRA!

Ejercicio B2 - Período Fijo

- Ahora implementaremos una **Onda Cuadrada** en el led azul de la EDU-CIAA, utilizando FreeRTOS, donde el período sea de **1s** y el tiempo de led encendido crezca de **100 a 900ms en pasos de 100ms**, de forma cíclica, como se muestra en el siguiente gráfico:

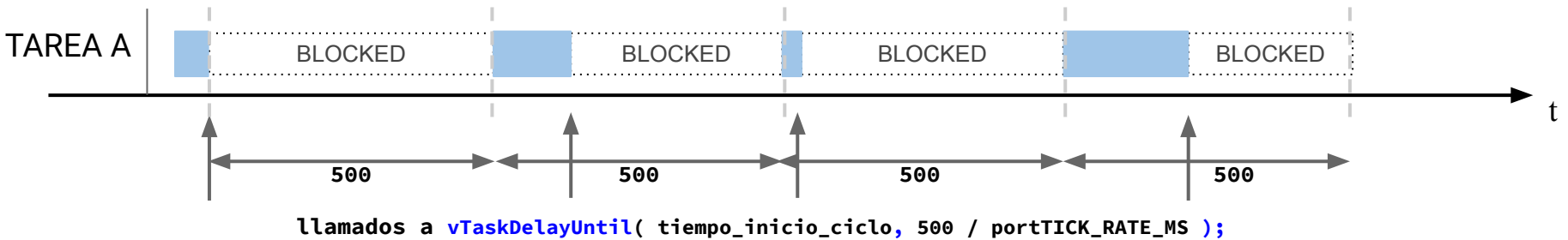
- <u>Recomendación</u>: Apoyarse en la segunda API de temporización que ofrece FreeRTOS, para asegurar el período.

```
void vTaskDelayUntil( TickType_t *pxPreviousWakeTime,
                      const TickType_t xTimeIncrement )
```

API: **vTaskDelayUntil**

- `void` **`vTaskDelayUntil`**`( TickType_t *pxPreviousWakeTime, const TickType_t xTimeIncrement );`
  - Asegura un delay entre cada uno de los llamados a esta función. Cede el control del CPU mientras este tiempo no expira.

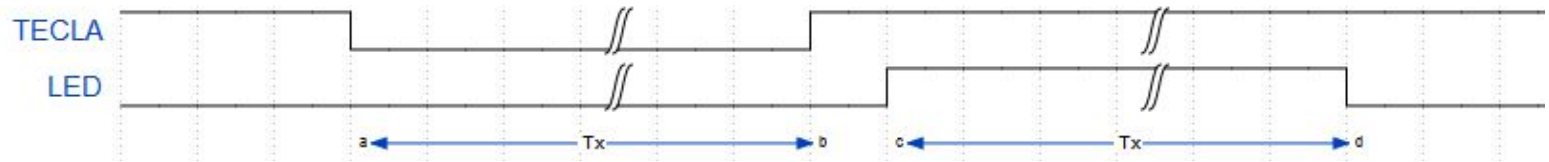
```
void TareaA( void* params )
{
	TickType_t tiempo_inicio_ciclo;
	Tiempo_inicio_ciclo = xTaskGetTickCount();
	while(1)
	{
		/* CODIGO DE PROCESO PERIODICO QUE TARDA EN EJECUTARSE (SIN INTERRUPCIONES ENTRE 1 Y 200 TICKS) */
		vTaskDelayUntil( tiempo_inicio_ciclo, 500 / portTICK_RATE_MS );
	}
}
```



llamados a `vTaskDelayUntil( tiempo_inicio_ciclo, 500 / portTICK_RATE_MS );`

Ejercicio B3 - Medir tiempo

- Para comprobar el estado de funcionamiento de un pulsador, queremos reproducir el tiempo de pulsación en un led de la EDU-CIAA. Implementando un mecanismo anti-rebote por software, desarrolla un programa que mida el tiempo de pulsación y que luego lo reproduzca en el led azul como se muestra en el siguiente gráfico:



- Recomendación: Se puede consultar el contador de ticks del SO para medir el tiempo. Sin embargo, se debe tomar en cuenta que la variable que contiene el contador de ticks puede desbordarse.

Ejercicio B4 - Medir tiempos

- Reutilizar el código del ejercicio 3 para que permita encender 4 leds según un grupo de 4 teclas asociadas a cada uno de ellos.

- Recomendación:
  - No copie código. Más bien utilice las facilidades de parametrización de las tareas.
  - Antes de modificar el código de lo hecho en el ejercicio 3, deténgase a pensar qué elementos hay que independizar, para cada grupo Tecla/LED

Resolución temporal del sistema (tick)

- La resolución temporal del sistema está definida en la macro TICK_RATE.

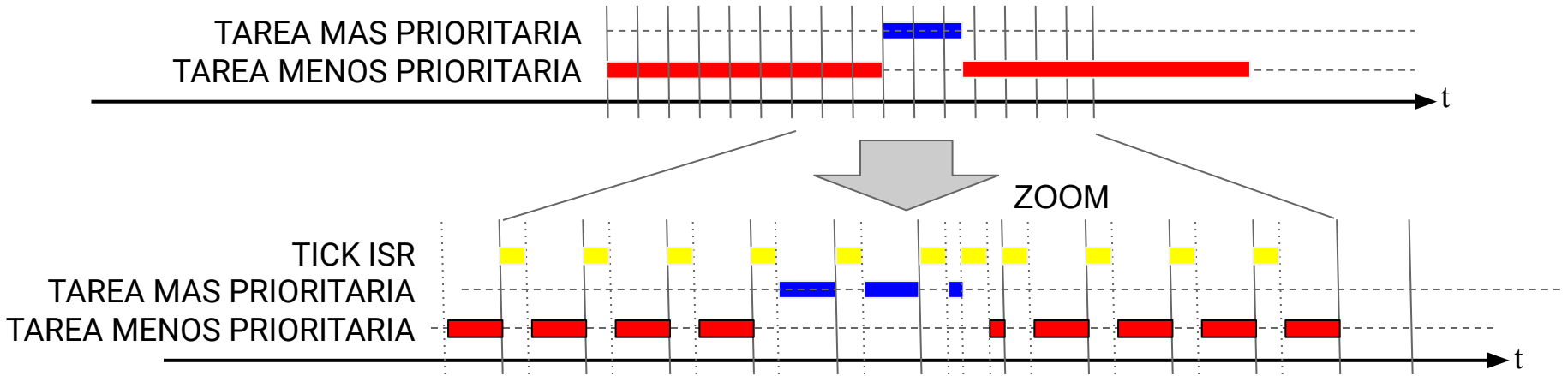
- El valor de TICK_RATE debe asignarse para garantizar que todos los plazos temporales de cubran a la perfección.

- Un valor muy alto de TICK_RATE hace que el scheduler trabaje más seguido, ocupando más tiempo del CPU.
  - Este tiempo debiera ser despreciable respecto del que consume la aplicación.

Ejercicio B6!

Letra chica 1

- Latencias temporales
  - Se gasta tiempo del CPU en determinar en todo momento qué tarea debe estar corriendo. Si el sistema debe manejar eventos que ocurren demasiado rápido tal vez no haya tiempo para esto.
  - Se gasta tiempo del CPU cada vez que debe cambiarse la tarea en ejecución.



Bibliografia

- https://www.freertos.org
- FreeRTOS Kernel Documentation
- Sistemas Operativos de Tiempo Real - Guía de Ejercicios, Franco Bucafusco, 2019
- Sistemas Operativos de Tiempo Real - Guía de Ejercicios Adicionales, Franco Bucafusco, 2019
- Introducción a los Sistemas operativos de Tiempo Real, Alejandro Celery - 2014
- Introducción a los Sistemas Operativos de Tiempo Real, Pablo Ridolfi, UTN, 2015.
- Introducción a Planificación de Tareas, CAPSE, Franco Bucafusco, 2017
- Introducción a Sistemas cooperativos, CAPSE, Franco Bucafusco, 2017
- FreeRTOS - Temporización, Cursos INET, Franco Bucafusco, 2017
- Rate-monotonic scheduling, Wikipedia Consultado 19-5-2
- Earliest Deadline First, Wikipedia Consultado 19-5-2

Licencia

"Introducción a los RTOS"