Universidad de Buenos Aires
Facultad de Ingeniería

Seminario de Sistemas Embebidos

TP freeRTOS

2^{do} cuatrimestre de 2012

Alumnos: Matías Petrella (68405)
Patricio Bos (81163)
Fecha: 22/02/13

Índice

1. Objetivo

El LPCXpresso es un toolchain completo de desarrollo y evaluación para microcontroladores de NXP. Se compone de los siguientes elementos:

- LPCXpresso IDE y herramientas de desarrollo
- LPCXpresso target board (stick)
- LPCXpresso Baseboard EA-XPR-021

Se propone utilizar el entorno de desarrollo LPCXpresso para implementar las siguientes tareas sobre el sistema operativo en tiempo real **freeRTOS**.

- **Tarea 1:**
  De forma periódica cada 100ms se debe enviar por la UART3 el mensaje constante "Tarea 1 - Enviando información por la UART 3".
- **Tarea 2:**
  Realizar un eco de los caracteres recibidos por la UART3 utilizando un semáforo para sincronizar la tarea con la interrupción del puerto serie.
- **Tarea 3:**
  Se debe inicializar el RTC en 00:00:00 y habilitar una interrupción externa por pulsador del BaseBoard. Cuando el pulsador se oprime se debe utilizar una cola para pasar el dato del RTC a una función que lo envie por la UART3

Para enviar información por la UART se debe implementar una función Enviar UART con el siguiente encabezado:

```
void UART3_Enviar (uint8_t* buffer, uint32_t size);
```

2. Desarrollo

2.1. Introduccion

Para cumplir los objetivos planteados se hará uso de los siguientes conceptos:

- **cola:** Las colas son los elementos con los que se establecen todas las comunicaciones dentro del entorno freeRTOS. Pueden contener un número finito de elementos de un tamaño fijo. Cuando se crean se definen dos parámetros: la cantidad de elementos y el tamaño de los mismos. Normalmente se utilizan como un buffer FIFO, aunque se puede leer en ambos sentidos.

- **semáforo:** Un semáforo binario puede ser considerado como una cola de 1 solo elemento cuando se lo usa para sincronizar un evento con una tarea de procesamiento de la misma.

- **mutex:**

  Es un tipo particular de semáforo binario que se utiliza para bloquear el acceso a un recurso compartido por mas una tarea. Cuando se utiliza un mutex se puede evitar que una tarea de mayor prioridad deje en un estado inconsistente el uso de un recurso por parte de una tarea de menor prioridad. Por ejemplo un mensaje que se este imprimiendo en un display. La desventaja es que ocurre una inversión de prioridades ya que una tarea de menor prioridad le impede el acceso al procesador a una de mayor prioridad. Los mutex se distinguen de los semáforos en que cada vez que una función lo toma, debe darlo al finalizar de la utilización del recurso.

  En el presente trabajo se utilizará un mutex cada vez que se quiera enviar información por la UART.

2.2. Includes, prototípos y variables globales

```
/* FreeRTOS.org includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"
#include "queue.h"

/* Demo includes. */
#include "basic_io.h"

#ifdef __USE_CMSIS
#include "LPC17xx.h"
#include "lpc17xx_uart.h"
```

```
#include "lpc17xx_pinsel.h"
#include "lpc17xx_rtc.h"
#include "lpc_types.h"
#endif

#include <string.h>
#include <cr_section_macros.h>
#include <NXP/crp.h>
#include <math.h>

/* The tasks to be created. */
static void vPeriodicTask1(void);
static void vAsincronicTask2(void);
static void vAsincronicTask3(void);

void vSetupRtc(void);
void vSetupIntExt(void);
void vSetupUart3(void);
void UART3_Enviar(unsigned char* buffer, uint32_t size);

/* Declare a variable of type xSemaphoreHandle. This is used to reference the
semaphore that is used to synchronize a task with an interrupt. */
xSemaphoreHandle xCountingSemaphore;

/* Declare a variable of type xSemaphoreHandle. This is used to reference the
mutex type semaphore that is used to ensure mutual exclusive access to UART3. */
xSemaphoreHandle xMutex;

/* Declare a variable of type xQueueHandle. This is used to store the queue
that is accessed by task 3. */
xQueueHandle xQueue;

const char welcome[]={"LPC1769 encendido...UART3 configurada\r\n"};
const char TextForPeriodicTask1[]={"Tarea 1 - Enviando informacion por la UART3\
const char TextForTask2[]={"Tarea 2 - Caracter recibido: "};
const char TextForTask3[]={"Tarea 3 - "};
unsigned char datoString[]={"xx:xx:xx"};

unsigned char RxBuffer;

static RTC_TIME_Type localTime;
```

2.3. Funciones de configuración

Se crean funciones específicas de configuración de los periféricos que se ejecutan por única vez antes de la llamada al *schedule*. Estas son:

- **void vSetupRtc(void)**

```
void vSetupRtc()
{
/* RTC init module*/
RTC_Init(LPC_RTC);

/* Enable rtc (starts the tick counter) */
RTC_ResetClockTickCounter(LPC_RTC);
RTC_Cmd(LPC_RTC, ENABLE);

/* Set current time for RTC
 * Current time is 00:00:00 AM, 2013-01-01
 */
RTC_SetTime(LPC_RTC, RTC_TIMETYPE_SECOND, 0);
RTC_SetTime(LPC_RTC, RTC_TIMETYPE_MINUTE, 0);
RTC_SetTime(LPC_RTC, RTC_TIMETYPE_HOUR, 0);
RTC_SetTime(LPC_RTC, RTC_TIMETYPE_MONTH, 1);
RTC_SetTime(LPC_RTC, RTC_TIMETYPE_YEAR, 2013);
RTC_SetTime(LPC_RTC, RTC_TIMETYPE_DAYOFMONTH, 1);
/* Get and print current time */
RTC_GetFullTime(LPC_RTC, &localTime);
}
```

- **void vSetupIntExt(void)**

```
void vSetupIntExt()
{
// Configure EINT0 to detect
// set P2.10 as EINT0
LPC_PINCON->PINSEL4 = 0x00100000;
// Port 2.10 is rising edge .
LPC_GPIOINT->IO2IntEnR = 0x400;
// INT0 edge trigger
LPC_SC->EXTMODE = 0x00000001;
// INT0 is rising edge
```

```
LPC_SC->EXTPOLAR = 0x00000001;
NVIC_EnableIRQ(EINT0_IRQn);
}
```

- **void vSetupUart3(void)**

```
void vSetupUart3()
{
PINSEL_CFG_Type PinCfg;

PinCfg.Funcnum = PINSEL_FUNC_2;
PinCfg.OpenDrain = PINSEL_PINMODE_NORMAL;
PinCfg.Pinmode = PINSEL_PINMODE_PULLUP;
PinCfg.Pinnum = PINSEL_PIN_0;
PinCfg.Portnum = PINSEL_PORT_0;

PINSEL_ConfigPin(&PinCfg);
PinCfg.Pinnum = PINSEL_PIN_1;
PINSEL_ConfigPin(&PinCfg);

UART_CFG_Type UARTConfigStruct;
UARTConfigStruct.Baud_rate = 115200;
UARTConfigStruct.Databits = UART_DATABIT_8;
UARTConfigStruct.Parity = UART_PARITY_NONE;
UARTConfigStruct.Stopbits = UART_STOPBIT_1;
UART_Init(LPC_UART3, &UARTConfigStruct);
UART_TxCmd(LPC_UART3, ENABLE);

UART_IntConfig(LPC_UART3, UART_INTCFG_RBR, ENABLE);
NVIC_EnableIRQ(UART3_IRQn);

/* mensaje de startup. No tomo el semáforo xMutex
 * porque todavía no arrancó el schedule */
UART3_Enviar(welcome,strlen(welcome));
}
```

2.4. Enviar_UART3

Para enviar información por la UART3 se establece una función que toma un puntero a char donde se encuentra la información a transmitir por la UART como primer parámetro y la cantidad de elementos como segundo.

```
void UART3_Enviar(unsigned char* buffer, uint32_t size)
{
UART_Send(LPC_UART3, buffer, size, BLOCKING);

}
```

2.5. Tareas

- **Tarea 1:**

  Se crea la tarea 1 con los siguientes parámetros:

  xTaskCreate(vPeriodicTask1, "Tarea1", 240, NULL, 1, NULL);

  Se inicializa una variable de tipo *portTickType* llamada *xLastWakeTime* con el valor actual de ticks al momento de ejecución de la tarea 1. Esta variable sirve como parámetro para la función vTaskDelayUntil() para especificar la cantidad exacta de ticks que la tarea debe esperar en el estado *bloqueado* antes de pasar al estado *ready*. De esta manera se puede asegurar una ejecución periódica de la tarea.

```
static void vPeriodicTask1( void *pvParameters )
{
portTickType xLastWakeTime;

xLastWakeTime = xTaskGetTickCount();

/* As per most tasks, this task is implemented in an infinite loop. */
for( ;; )
{
xSemaphoreTake(xMutex,portMAX_DELAY);

/* Envia por la UART3 el mensaje constante. */
UART3_Enviar(TextForPeriodicTask1,strlen(TextForPeriodicTask1));

xSemaphoreGive(xMutex);

/* queremos que la tarea se ejecute exactamente cada 100 milisegundos. */
vTaskDelayUntil( &xLastWakeTime, ( 100 / portTICK_RATE_MS ) );
}
}
```

- **Tarea 2**

  Se crea la tarea 2 con los siguientes parámetros:

  xTaskCreate(vAsincronicTask2, "Tarea2", 240, NULL, 2, NULL);

  La tarea permanece bloqueada mientras el semáforo *xCountingSemaphore* no este disponible. Cuando la ISR correspondiente a la UART3 lo habilita, la tarea envia el caracter recibido junto con un mensaje constante.

```
static void vAsincronicTask2()
{
unsigned char * buffer;

/* Inicializacion del puntero al buffer de recepcion */
buffer = &RxBuffer;

/* As per most tasks, this task is implemented within an infinite loop. */
for( ;; )
{
/* Use the semaphore to wait for the event. The semaphore was created
 * before the scheduler was started so before this task ran for the first
 * time. The task blocks indefinitely meaning this function call will only
 * return once the semaphore has been successfully obtained - so there is no
 * need to check the returned value. */
xSemaphoreTake( xCountingSemaphore, portMAX_DELAY );
xSemaphoreTake(xMutex,portMAX_DELAY);

/* Envia por la UART3 el mensaje constante. */
UART3_Enviar(TextForTask2,strlen(TextForTask2));

UART3_Enviar(buffer,1);
UART_SendByte(LPC_UART3, '\n'); UART_SendByte(LPC_UART3, '\r');

xSemaphoreGive(xMutex);
}
}
```

- **Tarea 3**

  Se crea la tarea 3 con los siguientes parámetros:

xTaskCreate(vAsincronicTask3, "Tarea3", 240, NULL, 2, NULL);

La tarea 3 permanece en el estado bloqueado hasta que haya información disponible en la cola *xQueue*. La ISR correspondiente al pulsador externo se encarga de escribir en dicha cola.

```
static void vAsincronicTask3( void *pvParameters )
{
portBASE_TYPE xStatus;
char dato[8];

/* This task is also defined within an infinite loop. */
for( ;; )
{
/* As this task unblocks immediately that data is written to the queue this
 * call should always find the queue empty. */
if( uxQueueMessagesWaiting( xQueue ) != 0 )
{
xStatus = xQueueReceive( xQueue, dato, 0 );
xSemaphoreTake(xMutex,portMAX_DELAY);

UART3_Enviar("RTC inicializado en ",strlen("RTC inicializado en "));

UART3_Enviar(dato,8);
UART_SendByte(LPC_UART3, '\n');
UART_SendByte(LPC_UART3, '\r');

xSemaphoreGive(xMutex);
}

xStatus = xQueueReceive( xQueue, dato, portMAX_DELAY );

if( xStatus == pdPASS )
{
/* Data was successfully received from the queue, print out the received
value. */
xSemaphoreTake(xMutex,portMAX_DELAY);

UART3_Enviar(TextForTask3,strlen(TextForTask3));

UART3_Enviar(dato,8);
```

```
UART_SendByte(LPC_UART3, '\n');
UART_SendByte(LPC_UART3, '\r');

xSemaphoreGive(xMutex);
}
}
}
```

2.6. Routinas de interrupción

- **EINT0_IRQHandler**

```
void EINT0_IRQHandler ()
{
portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;
portBASE_TYPE xStatus;

uint32_t seg_int,min_int,hor_int;
char decena,unidad;

/* se limpia la interrupcion */
LPC_SC->EXTINT = 0x00000001;

RTC_GetFullTime(LPC_RTC, &localTime);

seg_int=localTime.SEC;
min_int=localTime.MIN;
hor_int=localTime.HOUR;

decena=(char)floor(hor_int/10)+48;
unidad=(char)fmod(hor_int,10)+48;
datoString[0]=decena;
datoString[1]=unidad;

decena=(char)floor(min_int/10)+48;
unidad=(char)fmod(min_int,10)+48;
datoString[3]=decena;
datoString[4]=unidad;

decena=(char)floor(seg_int/10)+48;
unidad=(char)fmod(seg_int,10)+48;
```

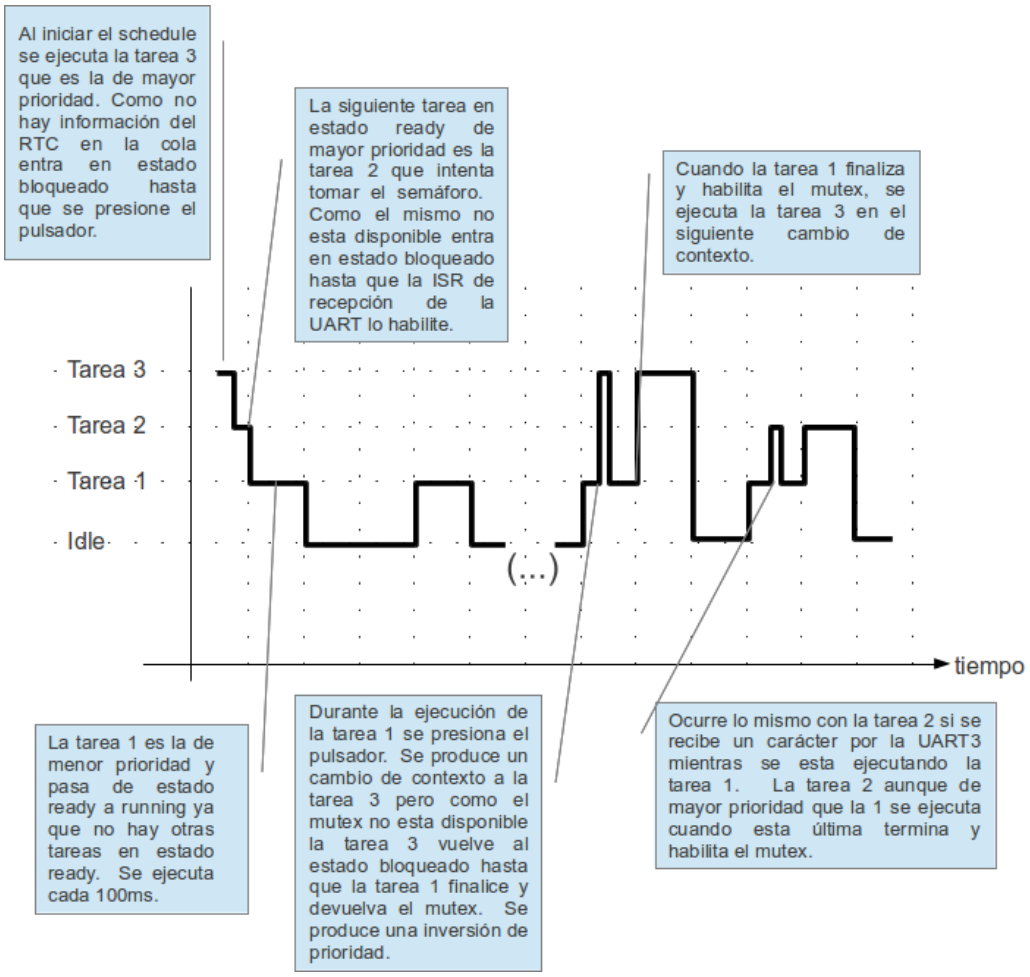
```
datoString[6]=decena;
datoString[7]=unidad;

xStatus = xQueueSendToBack(xQueue, datoString,0);

if (xStatus != pdPASS) {
/* the send operation could not be completed. xQueue full?
 * must be an error cause the queue could never contain
 * more than 1 element */
for(;;){
}
}

portEND_SWITCHING_ISR( xHigherPriorityTaskWoken );
}
```

- **UART3_IRQHandler**

```
void UART3_IRQHandler(void)
{
int i;

// Ugly Delay
for (i=0;i<100000;i++);

RxBuffer = UART_ReceiveByte(LPC_UART3);
unsigned char dummy= UART_ReceiveByte(LPC_UART3);

portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;

/* 'Give' the semaphore to unlock tarea2 */
xSemaphoreGiveFromISR( xCountingSemaphore, &xHigherPriorityTaskWoken );

/* Si una tarea de igual o mayor prioridad pasa a estado ready,
 * xHigherPriorityTaskWoken se establece en pdTRUE y
 * portEND_SWITCHING_ISR() asegura un cambio de contexto

 * La instruccion para realizar un cambio de contexto
 * desde dentro de una ISR es la macro
 * portEND_SWITCHING_ISR(). No se debe usar taskYIELD()
 * ya que no es ISRsafe  */
```

```
portEND_SWITCHING_ISR( xHigherPriorityTaskWoken );
}
```

3. Diagrama temporal



4. Conclusiones