EDUARDO MORA GONZÁLEZ

PRACTICA 5

SISTEMAS EMPOTRADOS Y UBICUOS

Ejercicio 1.

En este ejercicio se crea dos procedimientos bien diferenciados:

- **Creación y lógica de la tarea:**

```
static void LED_ROJO(void *param) {
    (void) param;
    for (;;) {
        ROJO_PutVal(0);
        vTaskDelay(500 / portTICK_RATE_MS);
        ROJO_PutVal(1);
        vTaskDelay(500 / portTICK_RATE_MS);
    }
}
```

Esta tarea tiene la funcionalidad de encender y apagar un led cada 0.5 segundos.

- **Llamada de la tarea:**

```
if (xTaskCreate(LED_ROJO, /* función de la tarea*/
                "rojo", /* nombre de la tarea para el kernel */
                configMINIMAL_STACK_SIZE, /* tamaño pila asociada a la tarea */
                (void*) NULL, /*puntero a los parámetros iniciales de la tarea */
                3,/* prioridad de la tarea, cuanto más bajo es el número menor es la prioridad */
                NULL /* manejo de la tarea, NULL si ni se va a crear o destruir */
                ) != pdPASS) { /* devuelve pdPASS si se ha creado la tarea */

    for (;;) {
    } /* error! Probablemente sin memoria */
}
FRTOS1_vTaskStartScheduler();
```

En este bloque dentro del main() se llama a la tarea y se le asigna un nombre, prioridad, manejador… Después de llama al planificador para empezar las tareas asignadas.

Ejercicio 2.

En este ejercicio es igual que el anterior, pero en este caso se crea otra tarea que enciende otro led y luego desde el main() se llama a dicha tarea.

El código de la tarea es el siguiente:

```
static void LED_VERDE(void *param) {
    (void) param;
    for (;;) {
        VERDE_PutVal(0);
        WAIT_Waitms(700);
        VERDE_PutVal(1);
        vTaskDelay(700 / portTICK_RATE_MS);
    }
}
```

Esta vez, se enciende el led verde cada 0,7 segundos.

El código de la llamada del main() es:

```
if (xTaskCreate(LED_VERDE, /* función de la tarea*/
"verde", /* nombre de la tarea para el kernel */
configMINIMAL_STACK_SIZE, /* tamaño pila asociada a la tarea */
(void*) NULL, /*puntero a los parámetros iniciales de la tarea */
3,/* prioridad de la tarea, cuanto más bajo es el número menor es la prioridad */
NULL /* manejo de la tarea, NULL si ni se va a crear o destruir */
) != pdPASS) { /* devuelve pdPASS si se ha creado la tarea */

}
```

Cuestiones

- **¿Qué diferencias observas con una prioridad y la contraria?**

Cuando la tarea del Led Rojo tiene mayor prioridad que la Verde, se inicia el led de color Verde, pero al momento se cambia al de color Rojo.

En el caso contrario donde el Led Verde tiene más prioridad que el Rojo, se empieza con el color Rojo, pero enseguida cambia al verde y hay un momento que se enciende los dos, apareciendo el color amarillo hasta que finalmente se queda el verde.

- **Probar a modificar el planificador para que sea no expropiativo, ¿Qué diferencias aparecen?**

En este caso hasta que no termina una tarea no empieza la otra.

Ejercicio 3.

En este ejercicio el código no varía a excepción de la línea donde aparece el código WAIT_waitms(500/700) que cambia a (el valor 500 es para el Led Rojo y para el Led Verde es 700):

```
vTaskDelay(500 / portTICK_RATE_MS);
```

Cuestiones

➔ **Para el planificador expropiativo**

- **¿Qué diferencias hay respecto a la actividad anterior? ¿Por qué?**

En este caso empieza el Led Verde, se solapan los dos y al final se quedan el Rojo. La razón es por que inicialmente se esta ejecutando la tarea del Led Verde y hasta que se realiza el cambio hay un momento en que conviven las dos tareas de manera recurrente.

➔ **Para el planificador no sea expropiativo**

- **¿Qué diferencias hay respecto a la actividad anterior? ¿Por qué?**

En este caso, se ve claramente como hay una lucha interna por quien tiene mayor prioridad, por lo que hay un momento en el cambio al de mayor prioridad se enciende los dos Led.

Ejercicio 4.

En este ejercicio se ha añadido un semáforo, el cual se ha declarado de manera global:

```
#include "semphr.h"

xSemaphoreHandle sem1;
```

Inicializado:

```
sem1 = FRTOS1_xSemaphoreCreateBinary();
FRTOS1_xSemaphoreGive(sem1);
```

Y usado en las distintas subtareas:

```
if (FRTOS1_xSemaphoreTake(sem1,(portTickType) 1000) == pdTRUE) {
    VERDE_PutVal(0);
    vTaskDelay(700 / portTICK_RATE_MS);
}
FRTOS1_xSemaphoreGive(sem1);
```

Cuestiones

¿Qué diferencias aparecen entre este caso y el caso de las actividades anteriores?

En este caso se nota de una manera muy clara que la tarea que tiene el semáforo (da igual la prioridad) ejecuta su acción hasta que termina el tiempo de custodia del semáforo.

Ejercicio 5.

En este ultimo ejercicio se ha usado colas, para ello se ha declarado la cola:

```
xQueueHandle colores;
```

Inicializado:

```
colores = FRTOS1_xQueueCreate(24, 8);
```

Y llamado en cada tarea a esta para añadir información:

```
char color[8] = "Verde";
for (;;) {
    FRTOS1_xQueueSendToFront(colores, color, (portTickType) 100);
```

Para ver el contenido de la cola y enviarlo por el puerto serie se ha implementado otra tarea:

```
static void Imprime(void *param) {
    char color[8];
    int i;
    for (;;) {
        if (xQueueReceive(colores, (void *) color,
                (portTickType) 0xFFFFFFFF) == pdTRUE) {
            /* Se ha recibido un dato. Se escribe por el puerto serie */
            for (i = 0; i < sizeof(color); i++)
                while (AS1_SendChar(color[i]) != ERR_OK) {
                }
            while (AS1_SendChar(10) != ERR_OK) {
            }
            while (AS1_SendChar(13) != ERR_OK) {
            }
        }
    }
}
```

Que al igual que las anteriores se llama en el main():

```
if (xTaskCreate(Imprime, /* función de la tarea*/
"imprimir", /* nombre de la tarea para el kernel */
configMINIMAL_STACK_SIZE, /* tamaño pila asociada a la tarea */
(void*) NULL, /*puntero a los parámetros iniciales de la tarea */
1,/* prioridad de la tarea, cuanto más bajo es el número menor es la prioridad */
NULL /* manejo de la tarea, NULL si ni se va a crear o destruir */
) != pdPASS) { /* devuelve pdPASS si se ha creado la tarea */
}
```

La salida de este ejercicio por el puerto serie es la siguiente:

```
Verde<NUL>␣<NUL>␣<NUL>␣<LF><CR>
Verde<NUL>␣<NUL>␣<NUL>␣<LF><CR>
Rojo<NUL>␣<NUL>␣<NUL>␣<NUL>␣<LF><CR>
Verde<NUL>␣<NUL>␣<NUL>␣<LF><CR>
Rojo<NUL>␣<NUL>␣<NUL>␣<NUL>␣<LF><CR>
Verde<NUL>␣<NUL>␣<NUL>␣<LF><CR>
Rojo<NUL>␣<NUL>␣<NUL>␣<NUL>␣<LF><CR>
Verde<NUL>␣<NUL>␣<NUL>␣<LF><CR>
Rojo<NUL>␣<NUL>␣<NUL>␣<NUL>␣<LF><CR>
```