Sistemes Encastats i Ubics

Laboratorio 5 – Multitarea con FreeRTOS

Introducción

El objetivo de la práctica es profundizar en las características de multitarea que ofrece FreeRTOS que corre en el ESP32-C6.

Trabajo previo

Una lectura atenta de:

https://docs.espressif.com/projects/esp-idf/en/v5.5.1/esp32c6/api-reference/system/freertos.html

https://www.freertos.org/Documentation/02-Kernel/01-About-the-FreeRTOS-kernel/01-FreeRTOS-kernel

https://www.freertos.org/Documentation/02-Kernel/02-Kernel-features/00-Developer-docs

https://www.freertos.org/Documentation/02-Kernel/03-Supported-devices/02-Customization

Trabajo de laboratorio

Las tareas a realizar son:

1) Explicad qué tipo de scheduling utilizáis en vuestro proyecto por defecto.
2) Una aplicación que realice en paralelo una intermitencia de 0.3 segundos del LED1 y una alternancia entre el LED2 y el LED3 de un segundo utilizando tareas de FreeRTOS y zonas de exclusión mutua. Realizad una versión con un único mutex y otra con un semáforo binario. ¿Existe alguna diferencia en el comportamiento? En ese caso explicad cuál puede ser el motivo.
3) Implementad un programa de dos tareas productoras (P1 y P2) y cuatro consumidoras (C1, C2, C3, C4). P1 generará números pares y P2 de impares. Lo hacen de forma alterna y al mismo ritmo. Los números se inyectarán a una cola de mensajes de capacidad 16. Las tareas consumidoras leen los valores de la cola de forma periódica también al mismo ritmo. Sincronizad las tareas para que C1 y C3 capturen números impares, y C2 y C4 números pares cada cierto tiempo de forma equitativa, es decir, que todas las tareas consumidoras tengan la oportunidad de consumir el mismo número de mensajes. Los productores tienen mayor prioridad que los consumidores.
4) Recuperad el código del apartado 2 e implementad lo necesario para estimar el tiempo de overhead dedicado por FreeRTOS desde que deja una tarea y entra a ejecutar la siguiente. El overhead incluye el tiempo dedicado a las instrucciones de la tarea de origen que provocan el salto de tarea, más la interrupción del kernel (PendSV), más el tiempo que el scheduler tarda en decidir qué tarea es la siguiente en ejecutar, más el cambio de contexto (guardar status de los registros y stack de la tarea anterior), recuperar los de la tarea de tarea anterior. Tened en cuenta que el overhead es del orden de microsegundos si el ESP32-C6 se configura para trabajar a 160MHz.

5) El siguiente código implementa un Mutex de FreeRTOS para intentar evitar una situación de inversión de prioridades gracias al mecanismo de Priority Inheritance. Ejecutadlo en el ESP32-C6 y explicad si es así. En caso que no, explicad porqué y cómo solucionarlo. (Pista: tiene que ver con algunos de los delays).

```c
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/semphr.h"
#include <stdio.h>

SemaphoreHandle_t mutex;

/*  PRIORITATS (de menor a major):
 *  LOW    : 3
 *  MEDIUM : 5
 *  HIGH   : 7
 */
void task_low(void *pv)
{
    while (1) {
        printf("[LOW] Intentant agafar el mutex...\n");
        xSemaphoreTake(mutex, portMAX_DELAY);
        printf("[LOW] Tinc el mutex! (simulant treball llarg)\n");
        // Simula feina llarga que bloqueja el recurs
        for (int i = 0; i < 10; i++) {
            vTaskDelay(pdMS_TO_TICKS(500));  // treball lent
            printf("[LOW] Treball lent... %d/10\n", i+1);
        }
        printf("[LOW] Alliberant mutex\n");
        xSemaphoreGive(mutex);
        printf("[LOW] Mutex alliberat\n");
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}

void task_high(void *pv)
{
    while (1) {
        printf(">>>> [HIGH] Intentant agafar el mutex...\n");
        // Aquí quedarà bloquejada si LOW té el mutex
        xSemaphoreTake(mutex, portMAX_DELAY);
        printf(">>>> [HIGH] Finalment he obtingut el mutex!\n");
        xSemaphoreGive(mutex);
        printf("[HIGH] Mutex alliberat\n");
        vTaskDelay(pdMS_TO_TICKS(3000));
    }
}
```

```c
void task_medium(void *pv)
{
    while (1) {
        // Aquesta tasca només consumeix CPU contínuament
        printf("[MEDIUM] Executant-se (interromp LOW!)\n");
        // Simula feina llarga que bloqueja el recurs
        for (int i = 0; i < 3; i++) {
            vTaskDelay(pdMS_TO_TICKS(500));  // treball lent
            printf("[MEDIUM] Treball lent... %d/3\n", i+1);
        }
        printf("[MEDIUM] Treball Finalitzat\n");
        vTaskDelay(50);
    }
}

void app_main()
{
    mutex = xSemaphoreCreateMutex();

    xTaskCreate(task_low,    "LOW",    4096, NULL, 3, NULL);
    vTaskDelay(pdMS_TO_TICKS(100));  // retard perquè LOW ja tingui el mutex
    xTaskCreate(task_high,   "HIGH",   4096, NULL, 7, NULL);
    xTaskCreate(task_medium, "MEDIUM", 4096, NULL, 5, NULL);
}
```

Entrega

- Código de las aplicaciones implementadas.
- Memoria pdf con las respuestas que se piden en el enunciado.