Sistemas Empotrados II

Trabajo Práctico:

Práctica 2 en FreeRTOS ESP32

Universidad de Zaragoza

Autores: *Sergio García Esteban* y *Óscar Baselga Lahoz*

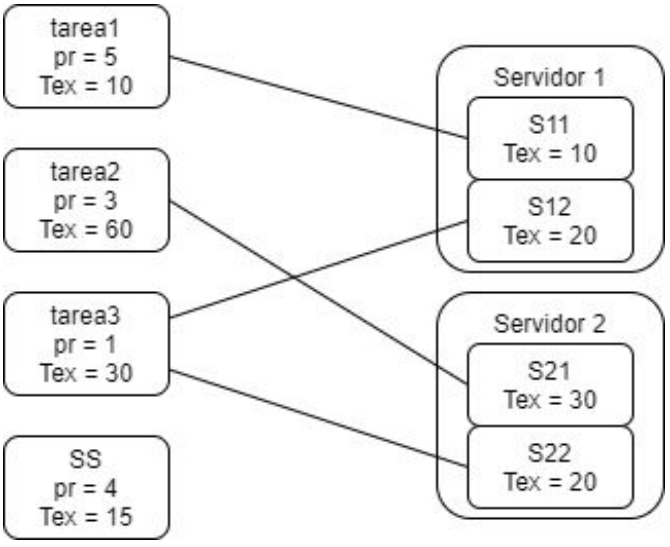
Introducción

Este proyecto se basa en el desarrollo del sistema en tiempo real definido en la práctica 2 de la asignatura en el sistema operativo en tiempo real **FreeRTOS**. Para ello, se va a hacer uso de la placa de desarrollo **AZDelivery ESP32 ESP-WROOM-32**.

Dicho sistema consiste en un conjunto de tareas periódicas, que ejecutan servicios proporcionados por una serie de servidores, basado en asignación de prioridades DM con tareas aperiódicas. Como consecuencia, se producen inversiones de prioridad que pueden ser analizadas con facilidad.

Tareas y servidores

A continuación se muestra un esquema del sistema a implementar. En él se pueden ver las distintas tareas, así como sus tiempos de ejecución, y los servidores con sus servicios.



Es importante comentar que el evento aperiódico, con prioridad hardware, se procesa en el servidor esporádico, el cual tiene prioridad 4. En el peor caso, el servidor esporádico se va a comportar como una tarea más, tratando todos los eventos pendientes que haya hasta el momento.

Tanto el periodo como la deadline de las tareas 1, 2 y 3 es de 100 ms, 200 ms y 400 ms respectivamente, y la ventana de ejecución del servidor esporádico de 120 ms.

Implementación

Para obtener retroalimentación del comportamiento del sistema, se ha empleado **logs** en varios ficheros. Los logs no solo permiten seguir la traza sino también comprobar los tiempos ya que imprime un timestamp en milisegundos.

Los logs se utilizan en distintas circunstancias, por ello existen varias funciones con la misma estructura: un TAG y el mensaje a mostrar por pantalla. La variante más empleada ha sido *`ESP_LOGI()`*, que consiste en un log informativo, pero también ha tenido un amplio uso la variante *`ESP_LOGE()`*, que se encarga de los logs de errores.

Existe un fichero de configuración en FreeRTOS con el nombre de "FreeRTOSConfig.h" donde se declaran multitud de variables que afectan al comportamiento de las funciones de FreeRTOS.

Durante el proceso de implementación ha sido necesario modificar este fichero para conseguir el funcionamiento esperado. En concreto se han puesto a 1 las variables:

- **`configSUPPORT_DYNAMIC_ALLOCATION`**, la cual ha permitido que la memoria RAM que utilizan los semáforos pudiera ser asignada dinámicamente.
- **`INCLUDE_vTaskSuspend`**, con la que se ha conseguido hacer que todas las funciones a las que se le pasaba el valor `portMAX_DELAY` se bloquearan indefinidamente, sin timeout.

Módulo Servidores

En este módulo se han implementado los dos servidores en los cuales las tareas van a ejecutar servicios. Para ello se ha hecho uso de dos semáforos, uno para cada servidor, que se encargan de gestionar el acceso al servidor correspondiente.

En FreeRTOS existen varios tipos de estructuras para conseguir la exclusión mutua. En un principio se estuvo dudando entre escoger los denominados *Semaphore Binary* o *Semaphore Mutex*, ya que son muy similares. Finalmente, se escogieron los ***Semaphore Mutex*** dado que, a diferencia de los otros, cuentan con un mecanismo de **herencia de prioridad** (algo fundamental para este trabajo).

Estos semáforos tienen un funcionamiento muy simple. Son creados con el constructor *`xSemaphoreCreateMutex()`* sin la necesidad de ningún parámetro, y para "ocuparlos" y "liberarlos" únicamente es necesario invocar a las funciones *`xSemaphoreTake()`* y *`xSemaphoreGive()`*, respectivamente. A ambas funciones se les pasa como parámetro el semáforo sobre el que quieren realizar la acción y, adicionalmente a *`xSemaphoreTake()`*, un valor en el que se indica el número de ticks máximo que va a estar la tarea bloqueada

esperando a que se libere el semáforo, una vez pasen esos ticks la tarea abandona la espera y sigue ejecutando. Para establecer un **bloqueo indefinido**, se ha pasado en ese parámetro el valor *portMAX_DELAY*.

Módulo Eventos

En este módulo se ha implementado un timer, el cual va a generar una salida en un GPIO que posteriormente será utilizada como señal de un nuevo evento. Esta interacción será realizada conectando físicamente dos pines: el utilizado por el timer y el empleado para capturar la señal.

Se ha elegido el **GPIO 16**, ya que según el esquema *pinout* del manual de la placa está disponible al usuario. Para configurar el GPIO se han utilizado dos funciones: *gpio_pad_select_gpio()*, cuya función es la de poner el pin 16 en modo GPIO, y *gpio_set_direction()*, a la cual se le pasa por parámetro si el pin va a ser de entrada o de salida. En este caso se trata de un GPIO de salida y por tanto dicho parámetro será *GPIO_MODE_OUTPUT*.

Además, se crea un timer software que va a poner el GPIO a 1 cada cierto tiempo. La creación de nuevo es sencilla, lo primero es construir el timer mediante *xTimerCreate()*, donde se define, aparte de un nombre, un identificador y un periodo, una función *callback* y el tipo de timer, en este caso ***auto-reload*** para que continúe funcionando a lo largo del tiempo. La función *callback* es muy útil porque va a facilitar la acción de poner a 1 el GPIO, que se consigue con la función *gpio_set_level()*. Una vez creado el timer, hay que ponerlo a funcionar mediante *xTimerStart()*, método al que se le ha pasado el valor 0 para indicar que se inicie de inmediato.

Módulo Cómputos

Para realizar este trabajo se ha necesitado implementar una función que simula una carga de trabajo, según un parámetro en milisegundos realizará cómputos hasta completar el tiempo.

Módulo Principal

En el módulo principal del proyecto se han implementado las tareas y el servidor esporádico.

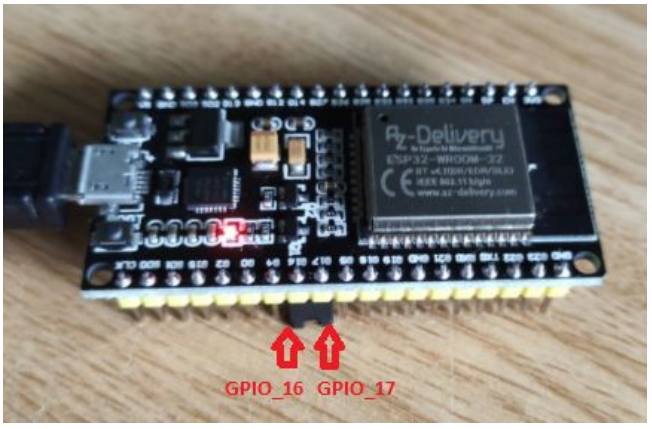
La función *main()* se encuentra en este módulo, desde ella se invoca a las funciones de **inicialización** de los módulos Servidores y Eventos mencionados anteriormente.

Además, se realiza la creación de **3 tareas** mediante *xTaskCreate()*, donde se define mediante sus parámetros: la función que ejecuta la tarea, la prioridad DM de la tarea y el tamaño de pila. Para que las 3 tareas se ejecuten periódicamente se van a crear **3 timer, 3 funciones callback y 3 semáforos**. Cada timer será *auto-reload* y con periodo igual al periodo de la tarea, al vencer el timeout del timer se invocará la función callback asignada, la cual únicamente hará *xSemaphoreGive() para liberar el semáforo*. El tipo de semáforo elegido para ello es el semáforo binario, creado con *xSemaphoreCreateMutex()* y la función que ejecuta la tarea tiene que hacer *xSemaphoreTake()* antes de empezar una ejecución de la tarea.

El servidor esporádico es creado con *xTaskCreate()* y se incluye un **Mailbox** para almacenar eventos con *xQueueCreate()* definiendo el tamaño como parámetro. Para usar el Mailbox se emplea *xQueueSend()* para añadir un elemento, *xQueueSendFromISR()* en este caso ya que es invocada desde la ISR del evento, y *xQueueReceive()* para eliminar el elemento más antiguo.

Para recibir la señal del evento se ha elegido el **GPIO 17**, ya que según el esquema *pinout* del manual de la placa está disponible al usuario. Para configurar el GPIO se han utilizado varias funciones: *gpio_pad_select_gpio()* que permite poner el pin 17 en modo GPIO, *gpio_set_direction()*, a la cual se le pasa por parámetro si el pin va a ser de entrada o de salida, en este caso *GPIO_MODE_INTPUT*, *gpio_install_isr_service()* que instala el driver que permite usar GPIO ISR handler, *gpio_isr_handler_add()* que asigna la función ISR a las interrupciones del pin 17, *gpio_set_intr_type()* a la cual se le pasa por parámetro el modo de disparo de la interrupción, en este caso se usa *GPIO_INTR_POSEDGE* para indicar flanco de subida, y por último *gpio_intr_enable()* para activar las interrupciones.

En la ISR se pone a 0 el GPIO 16 usando *gpio_set_level()* y se añade el evento a la Mailbox. Si se hubiera tenido acceso a un osciloscopio, podría haberse medido la **latencia de interrupción** del sistema midiendo con un canal en el GPIO 16.



Compilación y ejecución

Para trabajar con la placa ESP32 se ha utilizado el entorno ESP-IDF (Espressif IoT Development Framework) de Espressif, que es un entorno basado en scripts en Python que son utilizados para compilar el proyecto y flashearlo a la placa.

El proyecto ha sido programado en C, usando como plantilla el proyecto FreeRTOS de ejemplo que proporciona el entorno e incluye un fichero llamado *CMakeLists* en el que se añadieron los nombres de los ficheros de este proyecto que necesitan ser compilados. Además, se ha implementado un script llamado *auto.sh* que realiza la compilación del proyecto, el flasheo en placa y la monitorización de manera automática.

En la siguiente imagen se puede observar una traza de ejecución del programa, los tiempos medidos son correctos y se observa que las inversiones de prioridad se realizan de manera correcta.

```
I (1407) SE2_TP6: Empieza tarea 1
I (1417) SE2_TP6: Termina tarea 2
I (1427) SE2_TP6: Termina tarea 1
I (1427) SE2_TP6: Empieza tarea 1
I (1447) SE2_TP6: Termina tarea 1
I (1517) SE2_TP6: Termina SS
I (1517) SE2_TP6: Empieza SS
I (1527) SE2_TP6: Empieza tarea 3
I (1537) SE2_TP6: Empieza tarea 2
I (1607) SE2_TP6: Termina tarea 3
I (1627) SE2_TP6: Termina tarea 2
I (1627) SE2_TP6: Empieza tarea 1
I (1637) SE2_TP6: Termina SS
I (1637) SE2_TP6: Empieza SS
I (1647) SE2_TP6: Termina tarea 1
I (1647) SE2_TP6: Empieza tarea 1
I (1667) SE2_TP6: Termina tarea 1
I (1727) SE2_TP6: Empieza tarea 2
I (1757) SE2_TP6: Termina SS
I (1757) SE2_TP6: Empieza SS
I (1767) SE2_TP6: Empieza tarea 1
I (1787) SE2_TP6: Termina tarea 1
I (1817) SE2_TP6: Termina tarea 2
I (1827) SE2_TP6: Empieza tarea 1
I (1847) SE2_TP6: Termina tarea 1
```

Conclusión

Hemos aprendido a utilizar una placa de desarrollo como la ESP32 y hacer uso de su entrada/salida mediante GPIOs. Además, hemos aprendido a utilizar el entorno de desarrollo ESP-IDF para compilar proyectos y flashearlos en nuestra placa.

Nuestro proyecto ha sido realizado sobre FreeRTOS por lo que hemos estudiado en profundidad la API para conseguir finalmente desarrollar un sistema en tiempo real con tareas, servidores y herencia de prioridad. Hemos utilizado timers, semáforos, logs con timestamp, hemos generado y capturado eventos mediante interrupciones por el GPIO y un servidor esporádico que procese estos eventos aperiódicos guardados en un Mailbox.

Nos ha resultado muy interesante la realización de este proyecto porque nos ha ayudado a conocer más en profundidad el mundo del tiempo real. Además, consideramos muy útiles los conocimientos adquiridos porque consideramos que poco a poco se van aplicando más a la realidad, algo que motiva a seguir adelante.

Bibliografía

API FREERTOS
https://www.freertos.org/a00106.html

GPIO ESP32
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/gpio.html

LOG
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/log.html

Ejemplos
http://aquihayapuntes.com/foro/viewtopic.php?f=24&t=1011#