Sistemas operativos de tiempo real - 2025

Práctica N°2

Delays y Software Timers

Todo el código debe entregarse. Puede compartirse en github (usuario del docente: **hernangmol**) o enviarse por correo electrónico (her.gomez@bue.edu.ar) en una carpeta comprimida con toda la práctica completa. Identificar claramente el correo (por ejemplo: **RTOS 2025 - Práctica 1 - Hernán Gomez Molino**)
En cada ejercicio se indican en letra **negrita** los puntos que deben entregarse obligatoriamente además del código,. Las capturas de pantalla (salidas de programa) deben tener incluida la identificación del alumno y del ejercicio. La entrega debe hacerse en un documento en formato pdf donde se pegan las capturas y se responden las preguntas. Este documento debe estar incluido en la carpeta, si se responde por correo o puede compartirse como google docs si se entrega por github.

Fecha de entrega: **14/10/2025**

---

1) Delays - Uso del estado *blocked* para generar una demora (*delay*):
   Las tareas creadas en la práctica N°1 eran de tipo periódicas, imprimen un mensaje en pantalla y luego ejecutan un delay repetitivamente. Los delays implementados se hicieron ejecutando un lazo *for* muy largo que no ejecuta ninguna acción (lazo nulo). Por supuesto que esta no es una forma útil de realizar demoras ya que las tareas mantienen el control mientras están ociosas.

   a) Crear un clon del ejercicio 4 de la práctica N°1

   b) Modificar el programa para que realice las demoras utilizando la función API VTaskDelay. Las demoras deben ser de 500 ms.

   c) Ejecutar y comprobar funcionamiento. **Hacer captura de pantalla de la salida del programa**.

   d) **Explicar la diferencia en funcionamiento con respecto al ejercicio clonado.**

---

2) Delays - Uso de vTaskDelayUntil:
Dado que la función vTaskDelay no asegura una frecuencia fija para los eventos del lazo (asegura una demora mínima) debe utilizarse vTaskDelayUntil para ello.

   a) Clonar el ejercicio 1

   b) Modificar el programa para que la cadencia del programa sea fija (de 500 ms).

c) Ejecutar y comprobar funcionamiento. **Hacer captura de pantalla de la salida del programa**.

---

3) Delays - Tareas continuas y tareas periódicas:
En los ejercicios anteriores utilizamos tareas que nunca se bloquean (tareas continuas) y tareas que se bloquean (tareas periódicas) por separado.
Ahora vamos a combinarlas para que trabajen juntas.

a) Basándose en el ejercicio anterior que más le convenga generar dos tareas continuas que impriman un mensaje, sin demoras (lazo nulo) y que se ejecuten en prioridad 1.

b) Agregar una tercer tarea periódica que tenga prioridad 2 usando vTaskDelayUntil(). Configurar su periodo de manera que permita ver dos o tres ejecuciones de la tarea 3 en una sola captura de pantalla.

c) Ejecutar y comprobar funcionamiento. **Hacer captura de pantalla de la salida del programa**.

d) Según lo que puede observar entre ejecuciones de la tarea 3 **¿el sistema está trabajando en modo "time slice" o "run to completion"?**

---

4) Software Timers - Creación de timers ("one-shot" y periódico):
La utilización de software timers es opcional. Para habilitarlos y configurarlos deben setearse los siguientes parámetros (FreeRTOSConfig.h):

configUSE_TIMERS	1
configTIMER_TASK_PRIORITY	(configMAX_PRIORITIES - 1)
configTIMER_QUEUE_LENGTH	20
configTIMER_TASK_STACK_DEPTH	(configMINIMAL_STACK_SIZE * 2)

El primero es obligatorio. Los restantes son valores propuestos que pueden ser cambiados según la necesidad.
Debe tenerse en cuenta que es necesario además incluir la biblioteca "timers.h".

a) Generar un proyecto nuevo (copia). Configurar el uso de timers como se detalla en la introducción. Reemplazar el contenido del archivo *main.c* por la plantilla 2.4.

b) Ejecutar y comprobar funcionamiento. Debe ejecutarse un timer periódico de 500 ms.

c) Usando el código de la plantilla como referencia crear un timer *one shot* con un periodo de 3,333 s.
d) Ejecutar y comprobar funcionamiento. **Hacer captura de pantalla de la salida del programa**.

---

5) Software timers - Timer Reset:

a) Clonar el programa del ejercicio anterior y modificar el callback del timer periódico para que haga un reset del timer one shot.

b) Ejecutar y comprobar funcionamiento. **Hacer captura de pantalla de la salida del programa**.

c) **Explicar brevemente el resultado observado**.

---

6) Software timers - Simulación de un *watchdog*:
Un mecanismo de watchdog es una rutina de software que se dispara por tiempo cuando algún evento que se esperaba no ocurre. Por ejemplo: una rutina que se supervisa no envía una señal de vida periódica (*heart beat* o señal de *keep alive*) dentro de un tiempo determinado.

a) Basándose en los dos ejercicios anteriores generar una rutina que envíe un *heartbeat* (impresión en pantalla) y resetee un watchdog. Si este pasa un tiempo mayor a 4 señales de vida perdídas debe activarse.

b) Ejecutar y comprobar funcionamiento. **Hacer captura de pantalla de la salida del programa**.

c) Simular una falla en el envío de señal de vida después de 10 ciclos, ejecutar y comprobar funcionamiento. **Hacer captura de pantalla de la salida del programa**.