

FreeRTOS

TP3

Facultad de Informática UNLP
Sistemas de Tiempo Real

- Melina Caciali Toniolo

melicaciali@gmail.com

02866/1

- Joaquín Chanquía

joaquin.chanquia@alu.ing.unlp.edu.ar

02887/7

- Mateo Emmanuel Larsen

larsenmateo.ml@gmail.com

02993/7

- Gabriel Ollier

gabyollier@hotmail.com

02958/4

- Franco Niderhaus

franconiderhaus@gmail.com

02976/6

- Bruno Zanetti

bzanetti09@gmail.com

02975/5

EJERCICIO 1

- **Objetivo**

Investigue y analice las posibilidades que brinda FreeRTOS para la creación y el manejo de tareas. Identifique cada una de las características que deben ser consideradas al momento de definir una tarea.

- **Resolución**

FreeRTOS ofrece una amplia variedad de herramientas y funciones para la creación y gestión de tareas, fundamentales en los sistemas de tiempo real. Algunas características importantes al definir tareas en FreeRTOS incluyen:

1. **Creación de tareas:** Se realiza mediante la función `xTaskCreate`, que toma parámetros clave como:
 - Puntero a la función de la tarea: Indica la lógica a ejecutar.
 - Nombre de la tarea: Identificador para depuración.
 - Tamaño de la pila: Especifica el espacio de la pila.
 - Prioridad: Controla el orden de ejecución.
 - Manejador de tarea: Opcional, permite controlar la tarea luego de su creación.
2. **Manejo de prioridades:** Cada tarea tiene una prioridad, y FreeRTOS permite cambiarla dinámicamente con `vTaskPrioritySet`. El sistema puede operar en modo preemptivo, permitiendo que las tareas de mayor prioridad interrumpan las de menor.
3. **Planificación y suspensión:** FreeRTOS es flexible con las técnicas de planificación. En el modo preemptivo, las tareas con mayor prioridad pueden tomar control sin esperar que la tarea actual termine. También se puede optar por la planificación cooperativa, donde una tarea sigue ejecutándose hasta que voluntariamente cede el control. Funciones como `vTaskSuspend` y `vTaskResume` permiten suspender y reanudar tareas manualmente.
4. **Retardos y sincronización:**
 - Retardos: `vTaskDelay` permite a una tarea detenerse por un tiempo determinado.
 - Sincronización: FreeRTOS ofrece herramientas de sincronización como semáforos (binarios y de conteo) y colas para la comunicación entre tareas.
5. **Manejo de recursos compartidos:** Es posible usar semáforos y MUTEX para evitar problemas de condiciones de carrera al acceder a recursos compartidos.
6. **Configuración básica:** La configuración del sistema, como la habilitación de prioridades, tamaño de pila mínimo y uso de MUTEX, se define en el archivo `FreeRTOSConfig.h`.

EJERCICIO 2

- **Objetivo**

- A. Desarrolle una aplicación de planificación que contenga 3 tareas. Las mismas deben competir por imprimir en la terminal un mensaje con el nombre de cada tarea.
- B. Realice pruebas variando las prioridades de cada una de las tareas, generando así diferentes escenarios en el mensaje de salida de la terminal.

- **Resolución**

- A. Para desarrollar esta aplicación se utilizó el material subido por la cátedra para la configuración del IDE de Arduino. Luego se programó un código en C que utiliza la librería freeRTOS para el manejo de las tres tareas que pide el enunciado.

El código es el siguiente:

```
// Incluimos la libreria Arduino FreeRTOS
#include <Arduino_FreeRTOS.h>

//Declaración de las tareas
void Task1(void *pvParameters);
void Task2(void *pvParameters);
void Task3(void *pvParameters);

//Código de arranque
void setup() {
    //Frecuencia de transmisión UART
    Serial.begin(9600);
    Serial.println("Comienza el programa");

    // Crear las tres tareas
    xTaskCreate(Task1, "Tarea 1", 128, NULL, 1, NULL);
    xTaskCreate(Task2, "Tarea 2", 128, NULL, 2, NULL);
    xTaskCreate(Task3, "Tarea 3", 128, NULL, 3, NULL);

    vTaskStartScheduler();
}

//Loop infinito
void loop() {
}
```

```
//Definición de las tres tareas
//Cada tarea imprime su nombre y luego cede la CPU por 1 segundo
void Task1(void *pvParameters) {
    while(true) {
        Serial.println("Tarea 1 \n\r");
        vTaskDelay(1000 / portTICK_PERIOD_MS); // Espera 1 segundo
    }
}
void Task2(void *pvParameters) {
    while(true) {
        Serial.println("Tarea 2 \n\r");
        vTaskDelay(1000 / portTICK_PERIOD_MS); // Espera 1 segundo
    }
}
void Task3(void *pvParameters) {
    while(true) {
        Serial.println("Tarea 3 \n\r");
        vTaskDelay(1000 / portTICK_PERIOD_MS); // Espera 1 segundo
    }
}
```

- B.** Haciendo pruebas con distintas prioridades entre las tareas llegamos a las siguientes conclusiones:

Igual prioridad entre todas las tareas:

Si se usa la misma prioridad entre todas las tareas, estas se ejecutan en el orden en el que están escritas en el programa.

```
Virtual Terminal
Comienza el programa
Tarea 1
Tarea 2
Tarea 3
Tarea 1
Tarea 2
Tarea 3
Tarea 1
Tarea 2
Tarea 3
```

Ejemplo con todas las tareas con prioridad 1.

Dos tareas con igual prioridad y una con una prioridad mayor:

En este caso se ejecuta primero la tarea con mayor prioridad y luego las otras dos tareas se ejecutan según el orden del programa.

```
Virtual Terminal
Comienza el programa
Tarea 3

Tarea 1

Tarea 2

Tarea 3

Tarea 1

Tarea 2

Tarea 3

Tarea 1

Tarea 2
```

Ejemplo con la tarea 3 con prioridad 2 y las tareas 1 y 2 con prioridad 1.

Dos tareas con igual prioridad y una con una prioridad menor:

Al igual que en el caso anterior las tareas con la misma prioridad se ejecutan en orden y luego se ejecuta la tarea con menor prioridad.

```
Virtual Terminal
Comienza el programa
Tarea 2

Tarea 3

Tarea 1

Tarea 2

Tarea 3

Tarea 1

Tarea 2

Tarea 3

Tarea 1
```

Ejemplo con la tarea 1 con prioridad 1 y las tareas 2 y 3 con prioridad 2.

Todas las tareas con prioridad diferente:

En este caso las tareas se ejecutan siguiendo el orden de la prioridad que tienen.

```
Virtual Terminal
Comienza el programa
Tarea 3
Tarea 2
Tarea 1
Tarea 3
Tarea 2
Tarea 1
Tarea 3
Tarea 2
Tarea 1
```

Ejemplo con las prioridades: Tarea3: 3, Tarea2: 2 y Tarea1: 1.

EJERCICIO 3

- **Objetivo**

Investigue y analice las herramientas que provee FreeRTOS para la sincronización de tareas.

- **Resolución**

FreeRTOS ofrece una serie de herramientas para la sincronización de tareas, diseñadas para gestionar la comunicación entre tareas y asegurar un acceso seguro a los recursos compartidos en sistemas de tiempo real. Considerando que cada una de las tareas es un pequeño programa en sí mismo, para el funcionamiento óptimo del sistema que estemos desarrollando necesitaremos brindar comunicación entre estas. En base a esto surgen los 2 tipos de herramientas de sincronización:

- Herramientas que comunican tarea con rutina de interrupción y viceversa: Semáforos.
- Herramientas que comunican tareas con tareas: Colas.

1. Semáforos

Los semáforos son uno de los mecanismos más utilizados en FreeRTOS para sincronización entre tareas o entre tareas e interrupciones. Existen dos tipos principales:

- **Semáforos binarios:** Estos semáforos solo tienen dos estados: tomado o disponible. Se utilizan para sincronizar tareas o para gestionar la señalización de eventos, como la sincronización de tareas con interrupciones. Un semáforo binario puede ser tomado (`xSemaphoreTake()`) o liberado (`xSemaphoreGive()`).

Funciones clave:

- `xSemaphoreCreateBinary()`: Crea un semáforo binario.
 - `xSemaphoreTake(sem, timeout)`: La tarea toma el semáforo, bloqueándose si no está disponible.
 - `xSemaphoreGive(sem)`: Libera el semáforo, permitiendo que otras tareas lo tomen.
 - `vSemaphoreDelete(sem)`: Elimina el semáforo.
- **Semáforos contadores:** A diferencia de los binarios, los semáforos de conteo permiten almacenar un valor entero, lo que significa que pueden gestionar más de un recurso o evento a la vez. Son útiles cuando se quiere controlar el acceso a múltiples instancias de un recurso.

Funciones clave:

- `xSemaphoreCreateCounting(max_count, initial_count)`: Crea un semáforo de conteo.
- `xSemaphoreTake(sem, timeout)` y `xSemaphoreGive(sem)`: Operan igual que los binarios, pero permiten múltiples accesos simultáneos hasta un máximo definido.
- `vSemaphoreDelete(sem)`: Elimina el semáforo.

2. Colas (Queues)

Las colas permiten la comunicación entre tareas mediante el envío y recepción de datos de manera segura. Son útiles para transferir información entre tareas o entre tareas y las interrupciones. Las colas en FreeRTOS pueden almacenar datos de cualquier tipo y tamaño.

Funciones clave:

- `xQueueCreate(size, item_size)`: Crea una cola especificando la cantidad de elementos y el tamaño de cada elemento.
- `xQueueSend(queue, item, timeout)`: Envía un nuevo ítem a la cola.
- `xQueueReceive(queue, buffer, timeout)`: Extrae un ítem de la cola y lo almacena en el buffer especificado.
- `xQueuePeek(queue, buffer, timeout)`: Permite leer un ítem de la cola sin eliminarlo.
- `uxQueueSpacesAvailable(queue)`: Devuelve la cantidad de lugares libres en la cola.

EJERCICIO 4

• Objetivo

Utilizando las herramientas de sincronización, implementar una aplicación multitarea que muestre constantemente en la terminal las siguientes secuencias:

- A. Tarea 1 - Tarea 3 - Tarea 2
- B. Tarea 2 - Tarea 2 - Tarea 3 - Tarea 1
- C. Tarea 3 - Tarea 3 - Tarea 3 - Tarea 1 - Tarea 2

Nota: Todas las tareas deben tener la misma prioridad

• Resolución

A. Tarea 1 - Tarea 3 - Tarea 2

```
#include <Arduino_FreeRTOS.h>
#include <semphr.h>
```



```

// Declaramos los semáforos
SemaphoreHandle_t semaforoTarea1;
SemaphoreHandle_t semaforoTarea2;
SemaphoreHandle_t semaforoTarea3;

// Tarea 1
void tarea1(void *pvParameters) {
    while (1) {
        // Esperamos el semáforo de la tarea 1
        xSemaphoreTake(semaforoTarea1, portMAX_DELAY);
        Serial.println("Tarea 1\n\r");
        // Damos el semáforo para la tarea 3
        xSemaphoreGive(semaforoTarea3);
        vTaskDelay(500 / portTICK_PERIOD_MS); // Añadimos un pequeño delay
    }
}

// Tarea 2
void tarea2(void *pvParameters) {
    while (1) {
        static int passed = false;
        // Esperamos el semáforo de la tarea 2
        xSemaphoreTake(semaforoTarea2, portMAX_DELAY);
        Serial.println("Tarea 2\n\r");
        // Damos el semáforo para la tarea 1
        xSemaphoreGive(semaforoTarea1);
        vTaskDelay(500 / portTICK_PERIOD_MS); // Añadimos un pequeño delay
    }
}

// Tarea 3
void tarea3(void *pvParameters) {
    while (1) {
        // Esperamos el semáforo de la tarea 3
        xSemaphoreTake(semaforoTarea3, portMAX_DELAY);
        Serial.println("Tarea 3\n\r");
        // Damos el semáforo para la tarea 2
        xSemaphoreGive(semaforoTarea2);
        vTaskDelay(500 / portTICK_PERIOD_MS); // Añadimos un pequeño delay
    }
}

void setup() {
    Serial.begin(9600);

    // Creamos los semáforos
    semaforoTarea1 = xSemaphoreCreateBinary();
    semaforoTarea2 = xSemaphoreCreateBinary();
    semaforoTarea3 = xSemaphoreCreateBinary();
}

```

```
// Creamos las tareas
xTaskCreate(tarea1, "Tarea 1", 128, NULL, 1, NULL);
xTaskCreate(tarea2, "Tarea 2", 128, NULL, 1, NULL);
xTaskCreate(tarea3, "Tarea 3", 128, NULL, 1, NULL);

// Inicializamos la secuencia dando el semáforo de la Tarea 1
xSemaphoreGive(semaforoTarea1);
}

void loop() {
    // El loop se queda vacío, ya que FreeRTOS maneja las tareas
}
```



```
Virtual Terminal
Tarea 1
Tarea 3
Tarea 2
Tarea 1
Tarea 3
Tarea 2
Tarea 1
Tarea 3
Tarea 2
Tarea 1
```

B. Tarea 2 - Tarea 2 - Tarea 3 - Tarea 1

```
#include <Arduino_FreeRTOS.h>
#include <semphr.h>

// Declaramos los semáforos
SemaphoreHandle_t semaforoTarea1;
SemaphoreHandle_t semaforoTarea2;
SemaphoreHandle_t semaforoTarea3;

// Tarea 1
void tarea1(void *pvParameters) {
    while (1) {
        // Esperamos el semáforo de la tarea 1
        xSemaphoreTake(semaforoTarea1, portMAX_DELAY);
        Serial.println("Tarea 1\n\r");
        // Damos el semáforo para la tarea 2
        xSemaphoreGive(semaforoTarea2);
        vTaskDelay(500 / portTICK_PERIOD_MS); // Añadimos un pequeño delay
    }
}
```

```

// Tarea 2
void tarea2(void *pvParameters) {
    while (1) {
        static int passed = 0;
        // Esperamos el semáforo de la tarea 2
        xSemaphoreTake(semaforoTarea2, portMAX_DELAY);
        Serial.println("Tarea 2\n\r");
        if (passed){
            passed = 0;
            // Damos el semáforo para la tarea 3
            xSemaphoreGive(semaforoTarea3);
        }else{
            passed = 1;
            // Damos el semáforo para la tarea 2
            xSemaphoreGive(semaforoTarea2);
        }
        vTaskDelay(500 / portTICK_PERIOD_MS); // Añadimos un pequeño delay
    }
}

// Tarea 3
void tarea3(void *pvParameters) {
    while (1) {
        // Esperamos el semáforo de la tarea 3
        xSemaphoreTake(semaforoTarea3, portMAX_DELAY);
        Serial.println("Tarea 3\n\r");
        // Damos el semáforo para la tarea 1
        xSemaphoreGive(semaforoTarea1);
        vTaskDelay(500 / portTICK_PERIOD_MS); // Añadimos un pequeño delay
    }
}

void setup() {
    Serial.begin(9600);

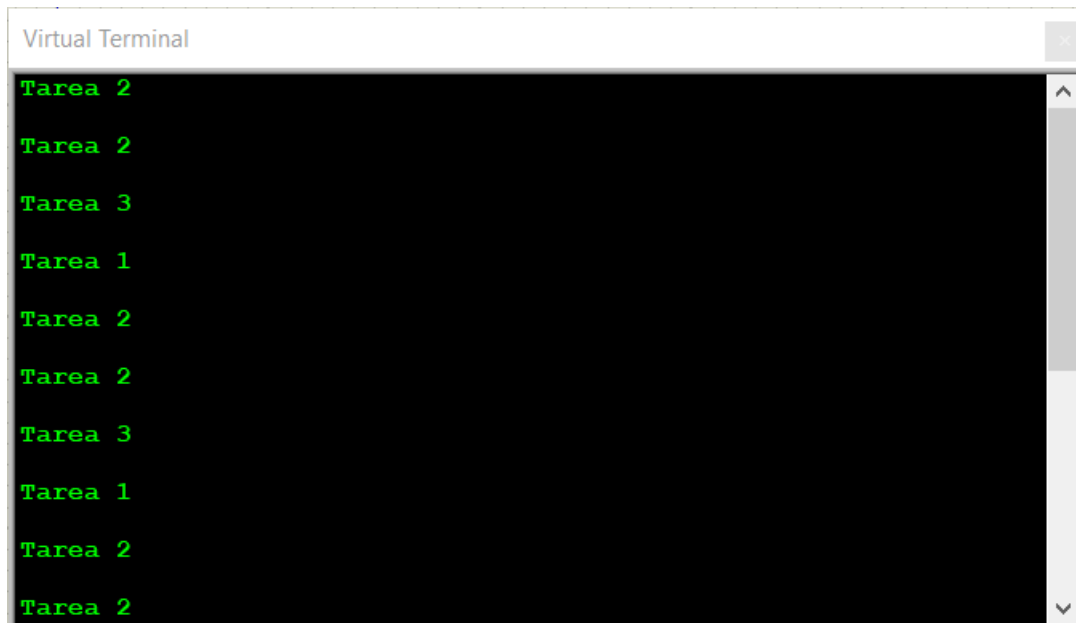
    // Creamos los semáforos
    semaforoTarea1 = xSemaphoreCreateBinary();
    semaforoTarea2 = xSemaphoreCreateBinary();
    semaforoTarea3 = xSemaphoreCreateBinary();

    // Creamos las tareas
    xTaskCreate(tarea1, "Tarea 1", 128, NULL, 1, NULL);
    xTaskCreate(tarea2, "Tarea 2", 128, NULL, 1, NULL);
    xTaskCreate(tarea3, "Tarea 3", 128, NULL, 1, NULL);

    // Inicializamos la secuencia dando el semáforo de la Tarea 1
    xSemaphoreGive(semaforoTarea2);
}

void loop() {
    // El loop se queda vacío, ya que FreeRTOS maneja las tareas
}

```



Virtual Terminal

```
Tarea 2
Tarea 2
Tarea 3
Tarea 1
Tarea 2
Tarea 2
Tarea 3
Tarea 1
Tarea 2
Tarea 2
```

C. Tarea 3 - Tarea 3 - Tarea 3 - Tarea 1 - Tarea 2

```
#include <Arduino_FreeRTOS.h>
#include <semphr.h>

// Declaramos los semáforos
SemaphoreHandle_t semaforoTarea1;
SemaphoreHandle_t semaforoTarea2;
SemaphoreHandle_t semaforoTarea3;

// Tarea 1
void tarea1(void *pvParameters) {
    while (1) {
        // Esperamos el semáforo de la tarea 1
        xSemaphoreTake(semaforoTarea1, portMAX_DELAY);
        Serial.println("Tarea 1\n\r");
        // Damos el semáforo para la tarea 2
        xSemaphoreGive(semaforoTarea2);
        vTaskDelay(500 / portTICK_PERIOD_MS); // Añadimos un pequeño delay
    }
}

// Tarea 2
void tarea2(void *pvParameters) {
    while (1) {
        static int passed = false;
        // Esperamos el semáforo de la tarea 2
        xSemaphoreTake(semaforoTarea2, portMAX_DELAY);
        Serial.println("Tarea 2\n\r");
        // Damos el semáforo para la tarea 3
        xSemaphoreGive(semaforoTarea3);
        vTaskDelay(500 / portTICK_PERIOD_MS); // Añadimos un pequeño delay
    }
}
```

```

// Tarea 3
void tarea3(void *pvParameters) {
    while (1) {
        static int count = 3;
        // Esperamos el semáforo de la tarea 3
        xSemaphoreTake(semaforoTarea3, portMAX_DELAY);
        Serial.println("Tarea 3\n\r");
        count--;
        if (count){
            // Damos el semáforo para la tarea 3
            xSemaphoreGive(semaforoTarea3);
        }else{
            count = 3;
            // Damos el semáforo para la tarea 1
            xSemaphoreGive(semaforoTarea1);
        }
        vTaskDelay(500 / portTICK_PERIOD_MS); // Añadimos un pequeño delay
    }
}

void setup() {
    Serial.begin(9600);

    // Creamos los semáforos
    semaforoTarea1 = xSemaphoreCreateBinary();
    semaforoTarea2 = xSemaphoreCreateBinary();
    semaforoTarea3 = xSemaphoreCreateBinary();

    // Creamos las tareas
    xTaskCreate(tarea1, "Tarea 1", 128, NULL, 1, NULL);
    xTaskCreate(tarea2, "Tarea 2", 128, NULL, 1, NULL);
    xTaskCreate(tarea3, "Tarea 3", 128, NULL, 1, NULL);

    // Inicializamos la secuencia dando el semáforo de la Tarea 1
    xSemaphoreGive(semaforoTarea3);
}

void loop() {
    // El loop se queda vacío, ya que FreeRTOS maneja las tareas
}

```

Virtual Terminal

```

Tarea 3
Tarea 3
Tarea 3
Tarea 1
Tarea 2
Tarea 3
Tarea 3
Tarea 3
Tarea 1
Tarea 2

```