Interrupciones en FreeRTOS.

Juan Alarcón.
jalarcon@electron.frba.utn.edu.ar

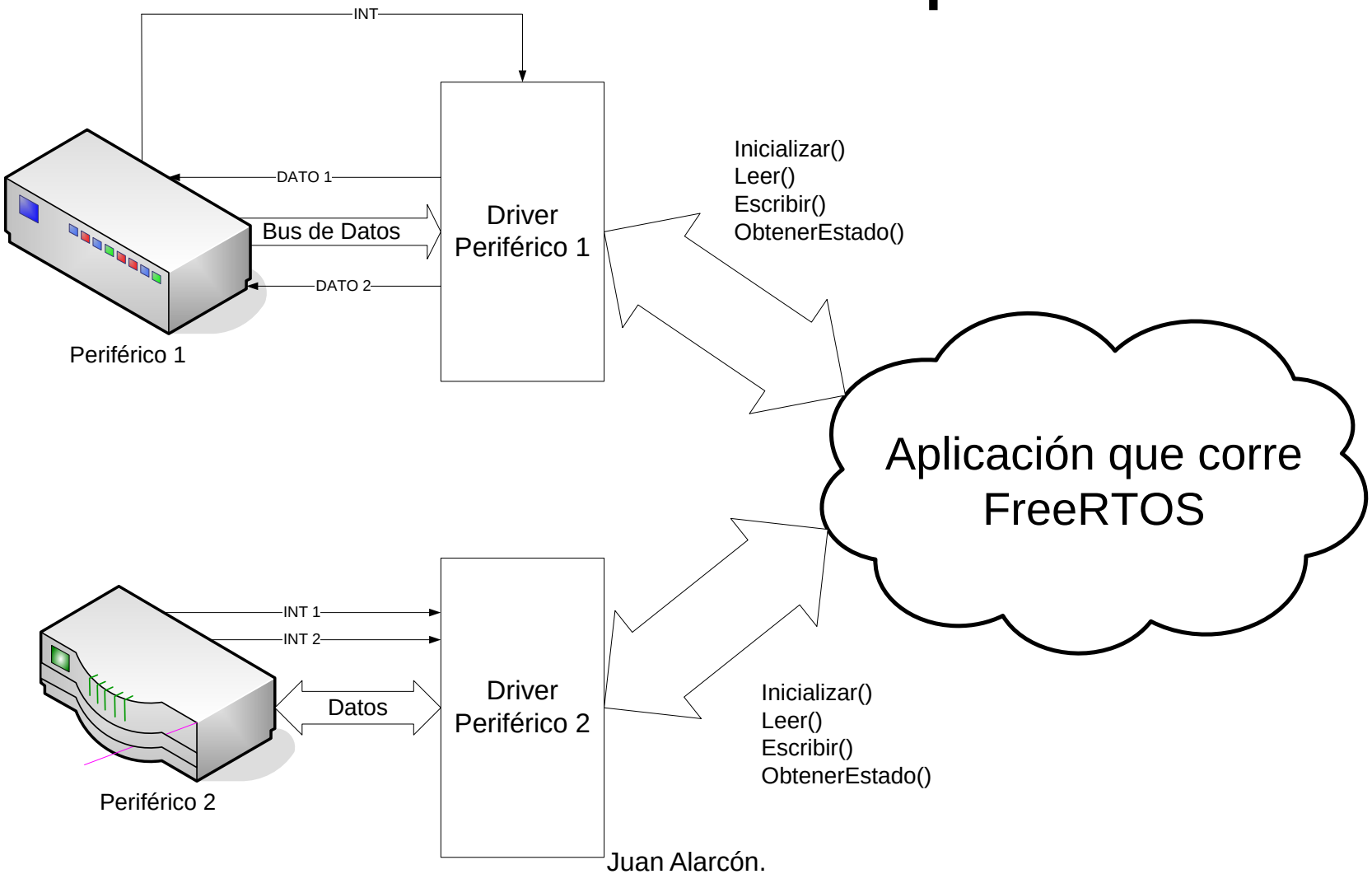
Agenda.

- Concepto de controlador de hardware (drivers).
- Eventos.
- Semáforos en Interrupciones.
- Colas para sincronizar interrupciones
- Ejemplo.

Controlador de Hardware. Driver.

- La estrategia propuesta para manejar los diferentes periféricos va a ser la siguiente:
  - Manejar las particularidades del hardware en código bien definido y separado de la lógica del programa (sólo tratar con el periférico, no introducir la lógica propia del programa en el código del driver).
  - Generar interfaces tan generales y claras como sea posible. Por ejemplo, generar funciones: inicializar(), escribir(), leer(), obtenerEstado().
  - Utilizar las primitivas del RTOS para la interfaz del sistema (semáforos, colas de mensajes).
  - En la medida de lo posible, generar interfaces bloqueantes (ser “gentil” avisar cuando se espera por un periférico)

Drivers. Ámbito de aplicación.



Juan Alarcón.
jalarcon@electron.frba.utn.edu.ar

Drivers. Objetivo.

- El objetivo principal de escribir el código de los controladores de los diferentes periféricos con interfaces comunes y con código bien delimitado es ***uniformizar el tratamiento de todos (o casi todos) los periféricos.***

Eventos.

- Los sistemas en tiempo real necesitan responder a los eventos generados por su entorno en tiempo conocido y acotado.
- ¿Cómo se implementa?
  - Encuesta del dispositivo (polling).
  - Por interrupciones.

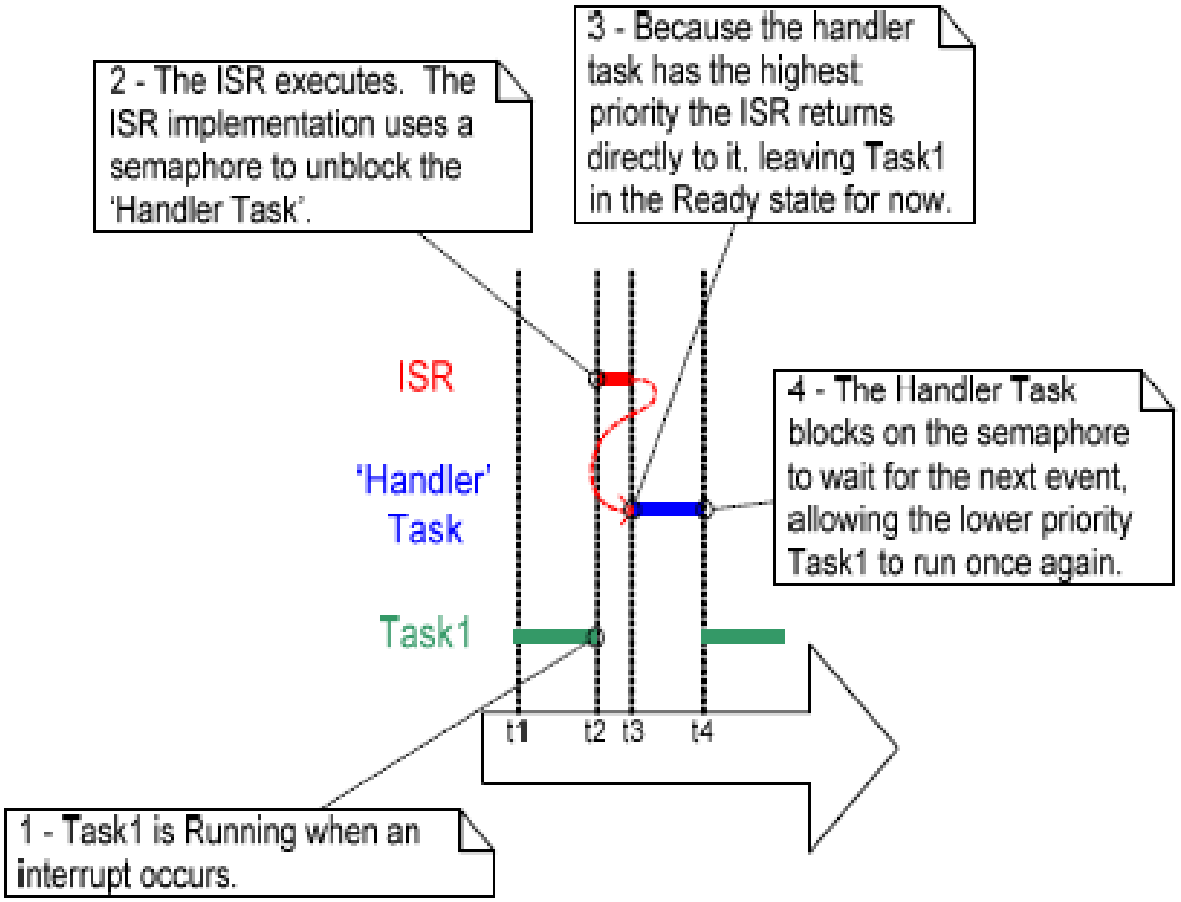
Atención de eventos por interrupciones.

- Al introducir el uso de interrupciones, ¿Cuánto tiempo usar en ellas?¿Por qué?.
- ¿Cómo comunicar el código de las interrupciones con el del resto del sistema?
- ¿Cómo hacer esta comunicación segura?.

Uso de semáforos en interrupciones.

- Un semáforo binario se lo puede utilizar para sincronizar una tarea con una interrupción.
- Si es necesario procesar con muy baja latencia un evento externo, el código de la interrupción puede desbloquear una tarea de alta prioridad para atenderlo.

Semáforos en interrupciones



Semáforos en interrupciones.

- xSemaphoreGiveFromISR
  (
  xSemaphoreHandle xSemaphore,
  ***signed portBASE_TYPE ****pxHigherPriorityTaskWoken
)

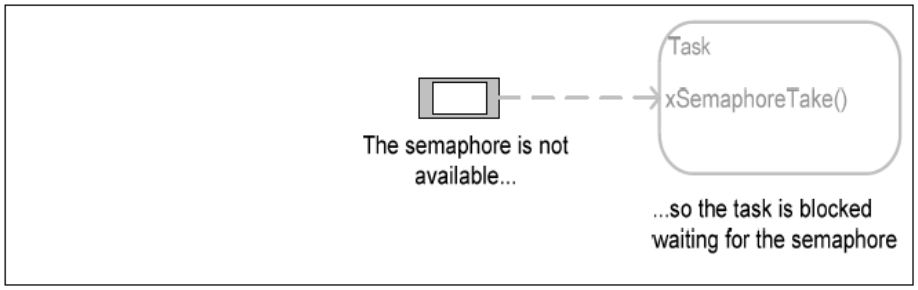
- portEND_SWITCHING_ISR
  (
  portBASE_TYPE HigherPriorityTaskWoken
)

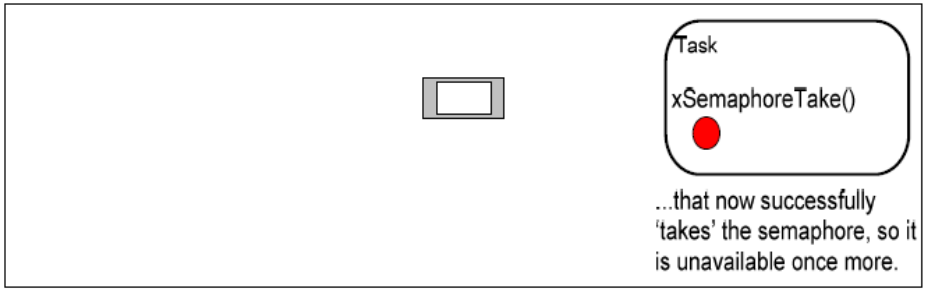
Semáforos en interrupciones.

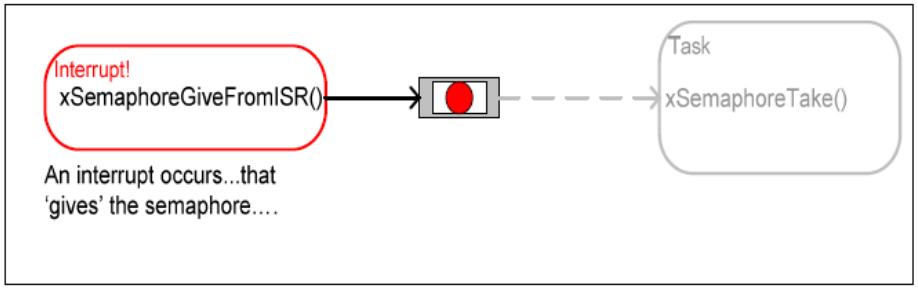
```
void ISR(void)
{
    portBASE_TYPE   ccontexto;
    xSemaphoreGiveFromISR(sem,&ccontexto);
    portEND_SWITCHING_ISR(ccontexto);
    devolvercontrol();
}
```

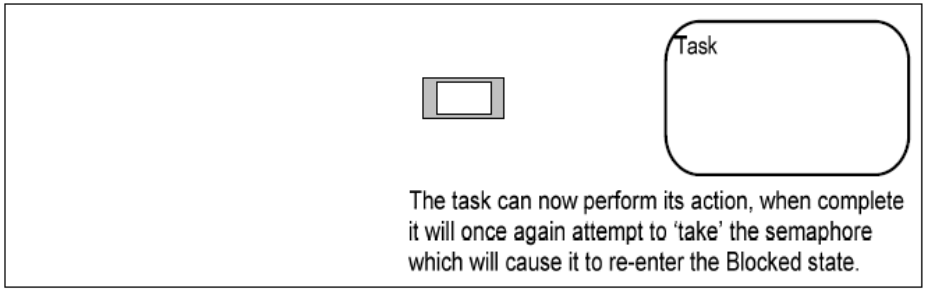
```
void Task1(void *parametros)
{
    for(;;)
    {
        codigoaejecutar();
        mascodigo();
    }
}

void HandlerTask(void *parametros)
{
    xSemaphoreTake(sem,portMAX_DELAY);
    for(;;)
    {
        xSemaphoreTake(sem,portMAX_DELAY);
        hacerloquehayquehacer();
    }
}
```

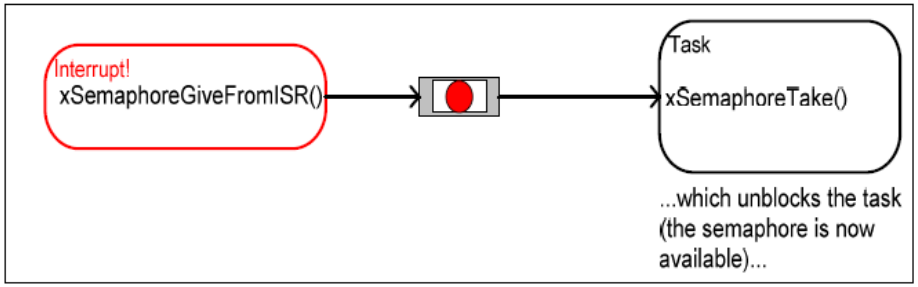
Semáforos en interrupciones.











Colas de mensajes en interrupciones.

- Los semáforos son usados para comunicar eventos entre tareas y entre tareas e interrupciones.
- Las colas de mensajes son usadas para comunicar eventos ***y transferir datos*** entre tareas y entre tareas e interrupciones

Colas de mensajes. Funciones.

- ***portBASE_TYPE*** xQueueReceiveFromISR
  (
  xQueueHandle pxQueue,
  void * pvBuffer,
  portBASE_TYPE * pxTaskWoken
)

Colas de mensajes. Funciones.

- ***portBASE_TYPE*** xQueueSendFromISR
  (
  xQueueHandle pxQueue,
  ***const void ****pvItemToQueue,
  ***portBASE_TYPE ****pxHigherPriorityTaskWoken
)
- ***portBASE_TYPE*** xQueueSendToBackFromISR
  (
  xQueueHandle pxQueue,
  ***const void ****pvItemToQueue,
  ***portBASE_TYPE ****pxHigherPriorityTaskWoken
)
- ***portBASE_TYPE*** xQueueSendToFrontFromISR
  (
  xQueueHandle pxQueue,
  ***const void ****pvItemToQueue,
  ***portBASE_TYPE ****pxHigherPriorityTaskWoken
)

Uso eficiente de las colas.

- Las colas de FreeRTOS son colas que trabajan por ***copia*** por lo que se puede caer en uso poco eficiente de las colas si:
  - Se transfieren elementos de muchos bytes.
  - Se transfieren elementos a alta frecuencia.

Uso eficiente de las colas.

- Para tener un uso eficiente en condiciones de muchos datos o mucha frecuencia de datos:
  - Poner en un buffer común los elementos y utilizar semáforos para sincronizar las tareas.
  - Interpretar los datos en la ISR y encolar la cantidad mínima (teniendo en cuenta que el tiempo en las ISR debe ser muy poco).

FreeRTOSConfig.h

- Las constantes que pueden afectar el anidamiento de interrupciones:
  - ***configKERNEL_INTERRUPT_PRIORITY***. Define la prioridad del núcleo del sistema operativo en si mismo.
  - ***configMAX_SYSCALL_INTERRUPT_PRIORITY***. Define la prioridad máxima que puede tener una interrupción para utilizar las funciones terminadas en FromISR.
- Si ***configMAX_SYSCALL_INTERRUPT_PRIORITY*** tiene más prioridad que ***configKERNEL_INTERRUPT_PRIORITY*** se va a trabajar con un esquema de anidamiento de interrupciones.

Configuración por defecto.

- ***configMAX_SYSCALL_INTERRUPT_PRIORITY*** ***5***
- ***configKERNEL_INTERRUPT_PRIORITY*** ***31***

Las interrupciones que no necesitan usar ninguna función del sistema operativo pueden tener cualquier prioridad

Las interrupciones que tienen prioridad desde 0 ***(por defecto para todos los periféricos)*** a 4 ***No pueden usar ninguna función de FreeRTOS!!!!!!.***

Las interrupciones que tienen prioridad desde 5 hasta 31 ***pueden llamar a las funciones de FreeRTOS que terminan en FromISR***

Bibliografía.

- Using the FreeRTOS Real Time Kernel. NXP LPC17xx Edition. Richard Barry.
- FreeRTOS http://www.freertos.org/