

Otros servicios de los RTOS

Sistemas embebidos para tiempo real

Índice

- Introducción
- Comunicación entre tareas:
 - Queues (colas de mensajes)
 - Buzón (mailbox)
 - Eventos
- Gestión del tiempo
- Gestión dinámica de memoria
- Interrupciones en RTOS

Introducción

- Servicios básicos de RTOS
 - Planificador de tareas (multitarea)
 - Semáforos: comunicación y sincronización entre tareas
- Servicios adicionales para comunicación entre tareas:
 - Cola de mensajes (queue)
 - Buzón de correo (mailbox)
 - Tuberías (pipes)

Cola de mensajes

- Características:
 - Permite el envío de mensajes entre tareas.
 - Mensajes organizados como un buffer circular.
- Normalmente se usan cuando:
 - Se necesita un almacenamiento temporal para ráfagas de datos.
 - Existen varios generadores de datos y un sólo consumidor y no queremos bloquear a los generadores.
- Dos estrategias:
 - Se envía un puntero a los datos del mensaje (μ C/OS-II)
 - Se copian los datos directamente (otros)

Cola de mensajes: Preguntas

- ¿Qué pasa si leemos cuando la cola esta vacía?
 - Más común: se bloquea a la tarea que llama.
 - Muchos RTOS ofrecen dos funciones para leer:
 - Lee de la cola y se bloquea si está vacía.
 - Lee de la cola y retorna (inmediatamente) un error si está vacía.
- ¿Qué pasa si queremos escribir y está llena?
 - Más común: se retorna un error.
 - Quien llama debe verificar el valor retornado: ¿Qué hacer?
 - Menos común: se bloquea hasta que haya lugar.
 - Obviamente esta versión no puede llamarse desde ISR.
 - Ninguna solución se adapta a todas la necesidades.

Cola de mensajes: ejemplo

- Registro de errores (logs)
 - Task1 y Task2: tareas de alta prioridad
 - requerimientos temporales exigentes
 - Si detectan errores necesitan registrarlos
 - almacenamiento en memoria no volátil o transmisión (demora mucho)
 - Solución: ErrorTask (tarea baja prioridad)
 - No afecta el tiempo de respuesta de Task1 y Task2
 - Comunicación de errores
 - Opción: cola de mensajes (se indica tipo error con un entero)

```
/* RTOS queue function prototypes */
```

```
void AddToQueue (int iData);
```

```
void ReadFromQueue (int *p_iData);
```

```
/* Task1: high priority*/
```

```
void Task1 (void)
```

```
{  
    ...  
    if (!! problem arises)  
        vLogError (ERROR_TYPE_X);  
    !! other things that need to be done  
    ...  
}
```

```
/* Task2: high priority */
```

```
void Task2 (void)
```

```
{  
    ...  
    if (!! problem arises)  
        vLogError (ERROR_TYPE_Y);  
    !! other things that need to be done  
    ...  
}
```

```
void vLogError (int iErrorType)
```

```
{  
    AddToQueue (iErrorType);  
}
```

```
static int cErrors;
```

```
/* ErrorTask: low priority*/
```

```
void ErrorsTask (void)
```

```
{  
    int iErrorType;  
    while (TRUE)  
    {  
        ReadFromQueue (&iErrorType);  
        ++cErrors;  
        !! Send cErrors, iErrorType  
    }  
}
```

Cola de mensajes: funciones

- Dos funciones reentrantes:
 - AddToQueue()
 - Agrega un mensaje a la cola
 - ReadFromQueue()
 - Obtiene un mensaje de la cola
- Repaso:
 - ¿Qué significa que las funciones sean reentrantes?
 - ¿Qué garantiza?

Buzones (mailbox)

- Características
 - Permite el pasaje de un mensaje entre tareas
 - Similar uso a la cola de mensaje

Buzones en μ C/OS-II: resumen

- `OS_EVENT *OSMboxCreate(void *msg)`
 - Crea el buzón con el mensaje inicial `msg`.
- `void* OSMboxPend(OS_EVENT *pevent, INT16U timeout, INT8U *err)`
 - Espera a que llegue un mensaje y devuelve su dirección.
- `INT8U OSMboxPost(OS_EVENT *pevent, void *msg)`
 - Pone un mensaje al buzón. Si ya hay un mensaje devuelve un código de error.

Buzón: Ejemplo

- Tarea de impresión por display/pantalla
 - Una tarea encargada de administrar la impresión en un display
 - Comunicación entre la tarea de impresión y el resto de las tareas: vía un Buzón
- Estudio de caso:
 - rutina de interrupción del *timer* manda actualizar el tiempo mostrado en el *display*.

```

#include <stdio.h>
#include "display.h"

/* Cabeceras de uC/OS-II */
#include <includes.h>

typedef struct{
    INTU8 hh;
    INTU8 mm;
    INTU8 ss;
} Time;

OS_EVENT *pMbox_time;
void main(void)
{
    /*bla bla bla*/
    pMbox_time = OSMboxCreate(NULL);
    /*bla bla bla*/
}

interrupt void TimerISR (void)
{
    static Time time_act;
    time_act.ss++;
    if(time_act.ss == 60){
        ...
    }
    OSMboxPost(pMbox_time, &time_act);
}

```

```

void TimePrint (void)
{
    char buf[10];
    Time time_copy, *pTime;
    int error;

    while(1){
        pTime = (Time *) OSMboxPend(
            pMbox_time, 0, &error);
        if(error == OS_NO_ERR){
            /* copia de dato compartido */
            __disableInt();
            time_copy = *pTime;
            __enableInt();
            sprintf(buf,
                "\n%02d:%02d:%02d",
                time_copy.hh,
                time_copy.mm,
                time_copy.ss);
            DisplayPutString(buf);
        }else{
            /* Procesar el error */
        }
    }
}

```

Pipes (tuberías, caño)

- Las tuberías (pipes) son similares a las colas.
- Las diferencias principales son:
 - Mensajes pueden ser de distinto tamaño.
 - Cada tarea puede escribir/leer un número arbitrario de bytes.
 - Normalmente se usan para cadenas de caracteres
- Es necesario establecer un protocolo:
 - Terminadas por un carácter nulo
 - Enviar el número de bytes total y luego el mensaje.
- μ C/OS-II no dispone de esta funcionalidad.

Eventos

- Es una bandera booleana que una tarea puede:
 - Crear, Disparar (Set), Borrar (Reset), Esperar (Wait for)
- Generalmente manejados en grupos
- Características:
 - Varias tareas se pueden desbloquear con un mismo evento
 - Tarea se desbloquea por cualquier evento del grupo
 - Después ocurre el evento, todas las tareas son desbloqueadas y el evento se borra.

Comunicación entre tareas: comparación

- Semáforos
 - Más rápido y simple, pero información si/no (ocupado)
 - La tarea se bloquea en un único semáforo por vez.
- Eventos:
 - Un poco más complejo, pero más flexible.
 - Una tarea puede esperar por muchos eventos a la vez.
 - Múltiples tareas pueden desbloquearse por un evento.
- Colas de mensajes (y otros como buzones o pipes).
 - Más complejas (mayor procesamiento y mayor posibilidad de bugs), pero permiten enviar más información
 - La tarea solo se bloquea en una única cola.

Gestión del tiempo

- En los RTOS la gestión de tiempo es fundamental:
 - Rutinas de control deben ejecutarse periódicamente: T_s .
 - Ahorro de energía: actividades no críticas que se realizan cada cierto tiempo (no continuamente).
 - Diálogo con el hardware.
 - Gestión de timeouts.

Gestión del tiempo

- Interrupción periódica
 - Incrementa un contador: cuenta desde el arranque.
 - Cada incremento se denomina *tick* de reloj.
 - Configurable al compilar el kernel.
 - Para el μ C/OS-II el tick es de 10 ms por defecto.
- Función retardo (delay)
 - Retarda la tarea por el periodo de tiempo especificado.
 - Se bloquea hasta que el tiempo expira.

Gestion del tiempo: μ C/OS-II

- `void OSTimeDly(INT16U ticks)`
 - Se bloquea la tarea durante el número de ticks especificado.
- `void OSTimeDlyHMSM(INT8U horas, INT8U min,
INT8U seg, INT8U mili)`
 - Se bloquea la tarea durante el tiempo especificado en horas, minutos, segundos y milisegundos.

Gestión del tiempo: Preguntas

- ¿Cuán precisos son los retardos?
- ¿Cómo se logran tiempos más precisos?
- ¿Son útiles los *timeout* de los semáforos, colas, etc.?

Gestión dinámica de memoria

- Funciones: malloc() y free() de la stdlib
 - No son apropiadas para sistemas de tiempo real
 - lentas
 - tiempos de ejecución no predecibles.
- RTOS disponen de funciones alternativas
 - Pools de memoria: buffers de tamaño fijo
 - Comportamiento predecible
 - Ciertas limitaciones

Gestión de memoria: μ C/OS-II

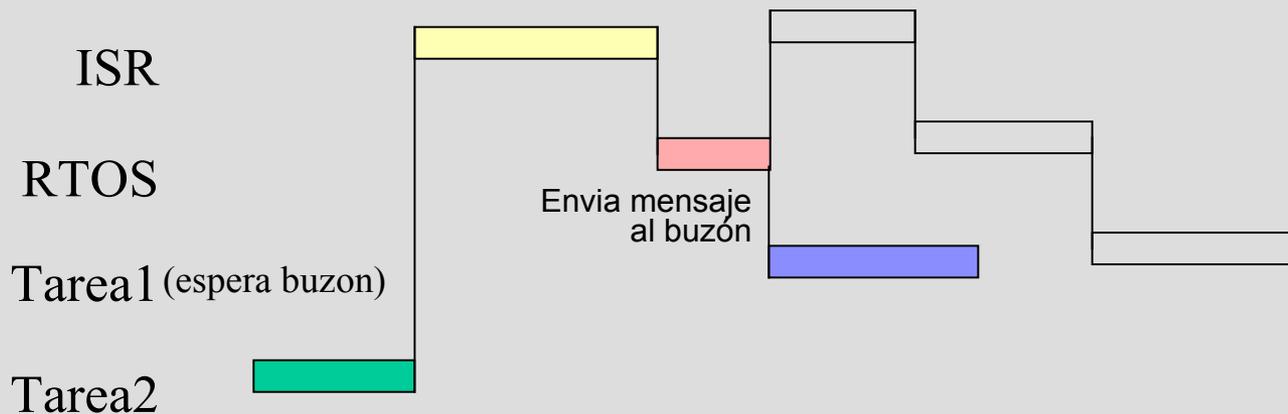
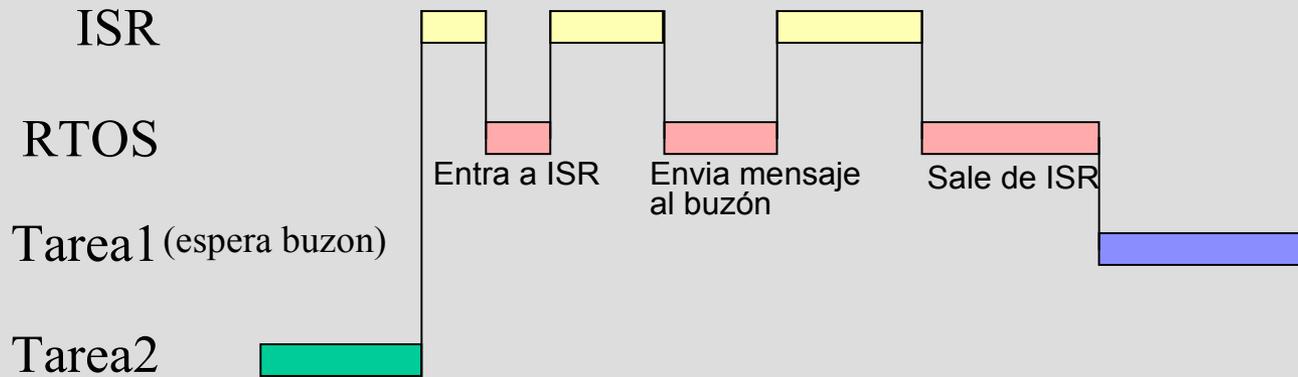
- La memoria se divide en particiones y cada partición se divide en bloques del mismo tamaño.
- La aplicación puede obtener bloques de una partición.
- `OS_MEM *OSMemCreate(void *addr, INT32U nblks, INT32U blksize, INT8U *err).`
 - Crea una partición de memoria.
- `void *OSMemGet(OS_MEM *pmem, INT8U *err)`
 - Obtiene un bloque de memoria.
- `INT8U OSMemPut(OS_MEM *pmem, void *pblk)`
 - Devuelve un bloque de memoria.

Interrupciones en RTOS

- Pregunta: ¿Qué sucede si en las ISR se llama funciones que...
 - puedan bloquearse?
 - Tomar semáforos, leer de colas, esperar por eventos
 - pueda hacer conmutar tareas?
 - Liberar semáforos, escribir en colas, disparar eventos
- Reglas dentro de las ISR
 - 1) NO llamar funciones que puedan bloquearse
 - 2) NO llamar funciones que puedan hacer conmutar tareas

Interrupciones: 2^{da} regla

NO llamar funciones que pueda hacer conmutar tareas

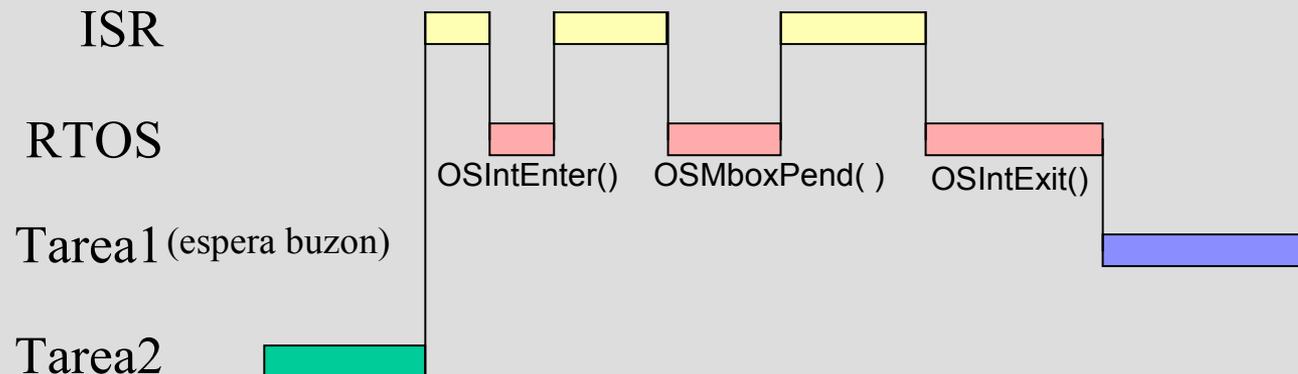


Interrupciones en RTOS

- Modificación 2da Regla:
 - No llamar funciones que puedan hacer conmutar tareas sin que el RTOS sepa que se está en una ISR.
- Diferentes estrategias:
 1. RTOS intercepta todas las interrupciones y luego llama a la ISR proporcionada por la aplicación.
 2. Avisar al RTOS de la entrada y salida de la ISR. Existen funciones especiales para su llamada desde las ISR.

Interrupciones en RTOS: μ C/OS-II

- `void OSIntEnter(void)`
 - Indica al RTOS que se empieza a ejecutar una ISR.
- `void OSIntExit(void)`
 - Indica al RTOS que se termina de ejecutar la ISR.



Bibliografía

- “An Embedded Software Primer”, David E. Simon
 - Chapter 7: More Operating System Services