Sistemas operativos embebidos

Sistemas Operativos de Tiempo Real

M. C. Miguelangel Fraga Aguilar

Requerimientos de tiempo

- Requerimiento suave: el sistema sigue funcionando aún cuando no se cumpla con el tiempo de respuesta en ocasiones
  - Ejemplo: respuesta a presión de teclas
- Requerimiento duro: el sistema falla si se deja de cumplir con el tiempo de respuesta
  - Ejemplo: Apertura lenta de una bolsa de aire en un auto

¿Por qué usar un RTOS?

- Las aplicaciones reales requieren de ejecución concurrente de múltiples tareas (**Multitasking**)
- Con excepción de los sistemas más simples, la implementación de multitasking por medio de un kernel o conmutador de tareas (**task scheduler**) es más eficiente.
- El kernel provee una interfaz para programas de aplicación (Application program interface - **API**) que maneja los requerimientos de tiempo.

¿Por qué usar un RTOS? (2)

- La API de tempo real permite que el código sea más simple y en general más pequeño
- La abstracción del manejo de los requerimientos de tiempo resulta en menos interdependencias entre módulos
- Lo anterior permite un desarrollo más predecible y controlado del software
- El desempeño es menos susceptible a cambios en el Hardware

¿Por qué usar un RTOS? (3)

- El desarrollo se vuelve **modular**, donde cada tarea es un modulo
- Permite el trabajo en equipo, al definirse interfaces entre cada tarea
- Es más fácil de probar y **depurar** cada tarea por separado
- Permite la **reutilización de código**
- Promueve el uso de técnicas de programación **orientadas a eventos** más eficientes.

¿Por qué usar un RTOS? (4)

- Tarea de reposo (**Idle task**): Esta tarea se crea automáticamente y se ejecuta cuando todas las tareas de aplicación se han desocupado
- Se puede usar para saber que porcentaje de la capacidad de procesamiento disponible se esta usando
- Puede realizar verificaciones del funcionamiento interno
- Puede entrar a un modo de bajo consumo de energía

¿Por qué usar un RTOS? (5)

- **Manejo** eficiente **de interrupciones**: Las rutinas de atención de interrupciones se mantienen lo más cortas posibles.
- Los pasos más largos de procesamiento se difieren a tareas regulares que pueden ser interrumpidas.
- Se disminuye la latencia de respuesta a un evento de interrupción y se evitan conflictos de prioridad de interrupción

¿Por qué usar un RTOS? (6)

- Se puede obtener una mezcla de patrones de diseño periódicos, continuos y de respuesta a evento
- Usando prioridades de tareas e interrupciones, se puede cumplir con requerimientos de tiempo tanto suaves como duros

FreeRTOS

- Desarrollado por Real Time Engineers LTD, propietaria de los derechos de autor del mismo.
- Código abierto con permiso para uso comercial sin regalías
- Puede trabajar de forma cooperativa o apropiativa (preemptive Multitasking)
- Asignación de prioridad de tareas flexible

FreeRTOS (2)

- Varios mecanismos de comunicación de procesos: colas, semáforos binarios y de conteo, variables mutex y mutex recursivos.
- Temporizadores por software
- Funciones periódicas y de tiempo de reposo
- Revisión de sobreflujo de la pila
- Recolección de estadísticas de tiempo de ejecución

FreeRTOS (3)

- FreeRTOS puede considerarse como una biblioteca que provee las funciones necesarias para la concurrencia (Multitasking).
- Consta de dos a cinco archivos en c de uso general y uno o más archivos específicos de cada port
- Un port es una combinación especifica de compilador y microcontrolador. Se incluye una demostración para cada port.

Documentos importantes

- Mastering the freeRTOS real time kernel. Richard barry. http://www.freertos.org/Documentation/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf
- FreeRTOS Reference Manual. http://www.freertos.org/Documentation/FreeRTOS_Reference_Manual_V9.0.0.pdf

FreeRTOSConfig.h

- Contiene constantes que modifican el comportamiento de FreeRTOS. Ejemplos:
- configUSE_PREEMPTION indica si FreeRTOS debe usar multitareas cooperativa o apropiativa
- Debe estar en una carpeta que contenga la aplicación y no la distribución de FreeRTOS
- Cada aplicación de demostración incluye una copia configurada para la misma

Estructura de directorios

- FreeRTOS/Source – código fuente de FreeRTOS
- FreeRTOS/Demo – Aplicaciones de demostración preconfiguradas
- FreeRTOS-PLUS/Source – código fuente de los componentes opcionales
- FreeRTOS-PLUS/Demo – Aplicaciones de demostración de los componentes opcionales preconfiguradas

Archivos comunes a todos los ports

- Obligatorios: tasks.c y lists.c
- Opcionales:
  - queue.c Provee colas y semáforos para la comunicación entre tareas. Casi siempre se requiere
  - timers.c – Temporizadores por software
  - event_groups.c – Grupos de eventos
  - croutine.c - Corutinas

Archivos específicos a cada port

- Se encuentran en la carpeta: FreeRTOS/Source/portable.
- Se incluyen en esta carpeta los archivos que implementan el manejo dinámico de memoria, heap_1.c a heap_5.c .Cada uno implementa un esquema diferente, aumentando en complejidad y flexibilidad. Uno debe incluirse en la aplicación.