LSE001-2014: Sistemas Operativos en Tiempo Real

Dr. Casimiro Gómez González
Facultad de Electrónica, UPAEP
correo: casimiro.gomez@upaep.mx
Tel: 222 229 9428

Otoño 2014

Prólogo

En el presente reporte se realiza un estudio de los sistemas operativos en tiempo real con el objetivo de aplicarlos en las tarjetas embebidas con microcontroladores ARM. Se ha elegido la plataforma *CMSIS-RTOS RTX* de **Keil** y el microcontrolador *LPC4357*. Se describe la aplicación de los sistemas operativos en tiempo real para un satélite **CubeSat**. Este documento se inicio en el otoño 2014 y estará en actualización continua.

El autor
Casimiro Gómez González
Doctor en Ingeniería Mecatrónica

Índice general

Capítulo 1

Introducción

Los microprocesadores arribaron en 1970. Los sistemas operativos encontraron aplicación rápida en las sistemas basados en microprocesador. Para mediados de 1980 pocas de estas implementaciones usadas se pueden describir como sistemas operativos en tiempo real diseñados formalmente.Dos factores afectan la aceptación de los sistemas operativos en tiempo real, uno debido a los limites de la máquina, y los otros a la cultura de diseño alrededor de los microcontroladores. Los primeros microprocesador estaban bastante limitados en sus habilidades computacionales, velocidad de operación y capacidades de memoria. Tratar de establecer la estructura de un sistema operativo bajo estas bases es muy difícil. Sin embargo, la mayoría de estos sistemas operativos embebidos programados tienen poco o ningún relación con sistemas operativos [1].

1.1. Produciendo software de calidad

Si se desea diseñar software de alta calidad, este debe tener las siguientes características:

- Debe realizar su trabajo correctamente (**funcionalmente correcto**)
- Debe realizar su trabajo en el tiempo correcto (**temporalmente correcto**)
- Su comportamiento debe ser predecible
- Su comportamiento debe ser consistente

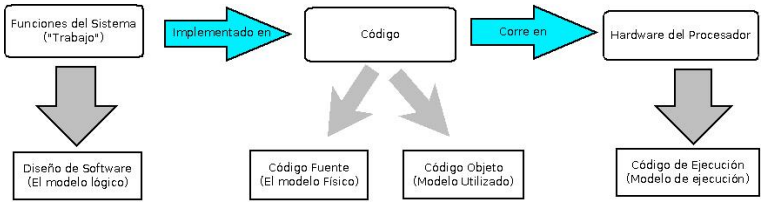

Figura 1.1: Etapas desde el diseño hasta la ejecución

- El código no debe ser difícil de mantener (baja complejidad)
- Lo correcto del código puede ser analizado (análisis estático)
- El comportamiento del código puede ser analizado (análisis de cobertura)
- El comportamiento en tiempo de ejecución debe ser predecible
- Los requerimientos de memoria debe ser predecibles
- El código puede, si es necesario, ser validado de acuerdo con estándares relevantes.

1.2. Modelando Software

Existen diferentes etapas para producir un programa ejecutable, como se indica en la figura 1.1.

Primero el diseño del software, resulta en el diseño o modelo lógico. Esto es implementado como código fuente, el modelo físico. El código fuente es compilado, enlazado y construido a un código objeto, el modelo bajado (deployment model). Finalmente el código objeto es cargado en el procesador y despues ejecutado, el modelo de ejecución. El modelo del código nos proporciona un visión estática del software. En contraste con el modelo de ejecución, una combinación de código, datos y procesador, representa software en ejecución. Esto se define como un proceso de software, también conocida como una tarea en el mundo de los embebidos. En palabras simples, una tarea representa la ejecución de un programa secuencial simple. Por lo regular al modelo de ejecución también se le llama modelo de tareas.

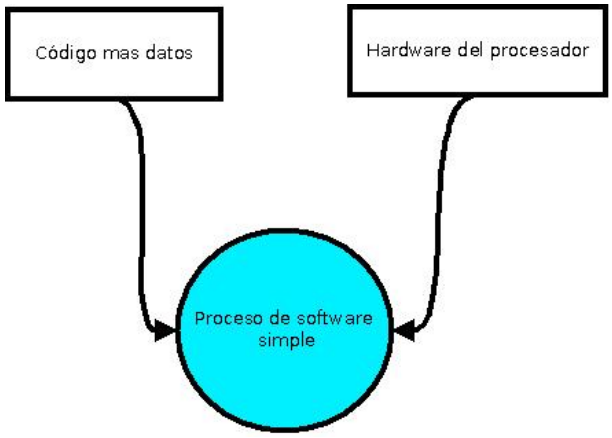

Figura 1.2: El modelo de ejecución del proceso de software

1.3. La importancia del tiempo y el temporizado

En sistemas electrónicos analógicos todas las operaciones pueden, si se requiere, realizarse simultáneamente (concurrentemente), no solamente eso, sino que el procesamiento es hecho instantáneamente. Pero esto no es posible en sistemas basados en procesador ya que este es fundamentalmente discreto en sus operaciones:

- Un procesador puede hacer solamente una cosa a tiempo (una maquina secuencial) y
- Las operaciones toman tiempo - las cosas no suceden instantáneamente

Estos son dos factores que causan muchas angustias en nuestro trabajo de diseño. Y esta es la razón por lo cual si esta diseñando sin conocer tus necesidades de tiempo, debes estar preparado para sorpresas desagradables.

Con esto en mente, los primeros diseños tenían las siguientes características:

- Es ejecutado en un lazo continuo
- Termina su trabajo en cada lazo (Semantica de ejecutar hasta completar)

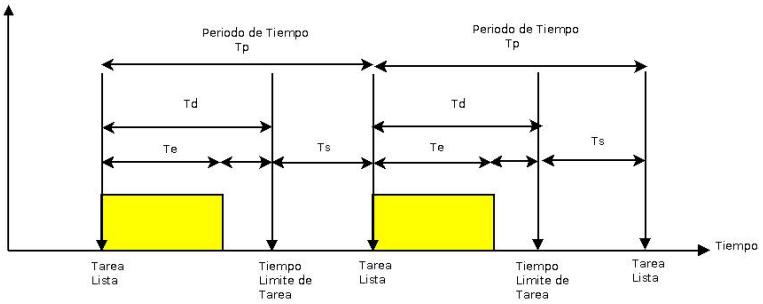

Figura 1.3: Algunas definiciones básicas del Temporizado de Tareas

- Toma tiempo para completar su trabajo (Tiempo de ejecución de tareas- Te)
- Es repetido a intervalos regulares o periodos (operaciones periodicas - Tp)
- No hacer nada mientras esta retardado (tiempo desperdiciado Ts)
- Necesita un mecanismo de temporizado para controlar el periodo de tiempo

En este diseño en particular Tp y Td son requisitos del sistema, Te depende de nuestro código de solución y Ts es nuestro tiempo limite, como puede verse en la figura . Por ejemplo, suponiendo que Tp es $100ms$ y Te es $5ms$, esto nos da un valor de Ts de $95ms$. Esto quiere decir que el procesador esta ejecutando código por $5ms$ cada $100ms$, una utilización (U) de 5 %.

Primero, en los sistemas embebidos se pretende que las tareas se ejecuten hasta ser completadas cada vez que sean activadas. Segundo, en los sitemas prácticos debe haber un tiempo de espera Ts. Tercero, por el hecho de que un código se pueda ejecutar dentro de un periodo de tiempo esto no significa que su rendimiento sea aceptable. El retardo de tiempo del procesamiento de entrada-salida puede causar problemas en el comportamiento de todo el sistema.

1.4. Manejando múltiples trabajos

Para manejar trabajos concurrentes independientes, lo cual significa que tenemos diferentes trabajos que en el mundo real pueden:

1. Ser atendidos a intervalos regulares - Funciones periódicas
2. Ser atendidos en tiempos aleatorios - Funciones asíncronas o aperiodicas
3. Ser procesados simultaneamente
4. Muy diferentes necesidades de temporizado

1.5. Usando Interrupciones como máquina de ejecución

Un **timer** se dispara, un interruptor se presiona, un dispositivo periférico necesita atención, son eventos típicos del mundo real en sistemas basados en computadoras. Hay, de hecho, solamente dos maneras de detectar estos eventos: Buscando estos eventos (polling) o enviando una señal directamente a el procesador (interrupciones de hardware).

A diferencia de los métodos anteriores cuando se usaba el polling dentro de una unidad secuencial de programa, ahora se buscan las interrupciones de hardware, una señal electrónica que es aplicada al procesador. Cuando la interrupción es generada se llama a una respuesta predeterminada (la cual es predefinida por el programador) para ejecutar un software específico. Así que la interrupción puede ser vista como una máquina de ejecución de una pieza particular de software.

Así que lo que tenemos es una ejecución aparentemente concurrente de un conjunto individual de procesos (tareas), llamado "cuasi-concurrencia". En otras palabras, usando interrupciones, podemos correr múltiples tareas simultáneamente: una forma de multitareas simple. De esta forma se puede concluir que:

1. Las interrupciones son mecanismos habilitados por llaves que permiten al diseño trabajar
2. Las interrupciones simplifican el diseño funcional del sistema
3. Las interrupciones simplifican el manejo de diferentes temporizados y requisitos de respuestas
4. El comportamiento actual en tiempo de ejecución no puede ser predicho o analizado estáticamente

Bibliografía

[1] Cooling, J. *Real-Time Operating Systems*, 1st ed. Lindentree Associates, 2014.