Comparación de Sistemas Operativos Embebidos sobre la Computadora Industrial Abierta Argentina

Medina Santiago[1], Pi Puig Martín[1], Dell'Oso Matías[1], Romero Fernando[1], De Giusti Armando[1,2], Tinetti Fernando G.[1,3]

[1]Instituto de Investigación en Informática LIDI (III-LIDI), Facultad de Informática, Universidad Nacional de La Plata, 50 y 120 2do piso, La Plata, Argentina.
[2]CONICET – Consejo Nacional de Investigaciones Científicas y Técnicas
[3]CIC – Comisión de Investigaciones de la Pcia. de Buenos Aires

{smedina, mpipuig, mdelloso, fromero, degiusti, fernando}@lidi.info.unlp.edu.ar

Abstract. En el siguiente trabajo se presenta una comparación de dos sistemas operativos embebidos aptos para uso en Sistemas de Tiempo Real (STR). Los puntos de comparación abarcan aspectos tales como planificadores, manejo de memoria, portabilidad, entre otros; en esta evaluación se hizo hincapié en la diferencia principal entre un Sistema operativo estático y uno dinámico. Se respaldó la comparación para obtener conclusiones concretas, realizándose mediciones sobre los tiempos en el manejo de tareas y el tiempo de respuesta a un estímulo externo.

Keywords: Tiempo Real, Sistemas Operativos, Sistemas Embebidos, Latencia, Planificación.

1 Introducción

Los Sistemas de Tiempo Real (STR) [1] [2] [3] [4] son aquellos que permiten proveer en forma garantizada un servicio dentro de un intervalo de tiempo de respuesta limitado. Esta restricción exige un cuidadoso diseño tanto del hardware como del software.
Dentro de los STR podemos distinguir:

1) Los que no usan SO: muchas veces en STR pequeños, el software no incluye un SO.

2) Los que usan SOE con MMU: estos tienen capacidad de gestionar multiprogramación, requieren un hardware más complejo. Por lo general, utilizan capacidades gráficas con lo que mejoran la interacción con los operadores. Muchos son derivados de SO convencionales: RT-Linux, RTAI, Linux RT-Preempt.

3) Los Sistemas Operativos Embebidos (SOE): si bien todos los sistemas de tiempo real suelen estar embebidos en un sistema mayor al cual controlan, los que tienen un SOE se caracterizan porque no se les puede agregar otros programas, solo tiene los que se compilan en forma conjunta con dicho SOE.

Los sistemas operativos embebidos son en general de mínimo tamaño. Sus principales características son:

- Se usan en sistemas embebidos, o sea aquellos que forman parte y controlan un sistema mayor.
- Aseguran rápidos cambios de contexto, dirigido a bajar la latencia del planificador y de las interrupciones.
- Bajo intercambio entre almacenamiento secundario y memoria.
- Gestión de archivos orientada a la velocidad de acceso más que a la eficiencia.
- No son eficientes procesando información. Por lo general no utilizan procesadores complejos.
- Se usan procesadores predecibles más que rápidos.
- No se utiliza memoria virtual, para evitar variaciones en los tiempos de acceso a memoria.
- Tiene un tiempo de respuesta acotado tanto a las interrupciones como a eventos que disparen tareas, tales como alarmas. Esto se debe a un cuidadoso diseño en esta área, principalmente del planificador.
- Objetivos muy limitados, por lo general pocas tareas.
- Diseñados para soportar numerosas arquitecturas y montarse sobre muchos dispositivos.
- Funcionan sobre hardware orientado a la entrada salida más que al procesamiento, con poca memoria, tales como los microcontroladores.

Luego, los SOE pueden clasificarse por la manera en que se realiza la alocación de memoria del sistema que controlan. Los mismos, pueden ser estáticos, en los cuales la alocación de recursos se realiza en tiempo de compilación, o dinámicos, donde se realiza en tiempo de ejecución.

En este trabajo se lleva a cabo una comparación de características de dos Sistemas Operativos Embebidos que corren sobre la placa de desarrollo EDU-CIAA-NXP, el primero de ellos es FreeOSEK, estático y el segundo FreeRTOS, dinámico.

Estos SOE se seleccionaron entre varios de este tipo ya que tienen cada uno características que los destacan entre los demás. Ambos han adquirido gran popularidad y una gran cantidad de usuarios en ambos casos; esto llevo a su portabilidad a una gran cantidad de

plataformas hardware. En ambos, la comunidad que presta soporte es numerosa, con lo que puede conseguirse abundante documentación y ejemplos. No es el caso de otros SOE que fueron desarrollados para arquitecturas específicas: MQX para Freescale (NXP), ERIKA para Microchip, etc.

2 Objetivos y metodología

Este trabajo tiene como objetivo comparar dos SOE, FreeOSEK y FreeRTOS corriendo sobre la misma plataforma de hardware a fin de evaluar las características fundamentales y necesarias en los Sistemas de Tiempo Real.

La metodología utilizada consiste en la programación de escenarios de pruebas que simulan la misma situación en ambos SOE, sobre la misma placa de desarrollo.

2.1 Hardware utilizado

Se utilizó una placa EDU-CIAA-NXP [13]. Está basada en la placa LPC4337. Cuenta con un microcontrolador ARM Cortex-M4 que incluye un coprocesador ARM Cortex-M0, 1 MB de memoria flash, 136 kB de SRAM, 16 kB de memoria EEPROM, periféricos como el timer de estado configurable (State Configurable Timer) (SCT) y el Serial General Purpose I/O (SGPIO) interface, dos controladores de alta velocidad USB, Ethernet, LCD, un controlador de memoria externa y múltiples entradas/salidas digitales y entradas analógicas. Opera a una frecuencia de reloj de más de 204 MHz. El ARM Cortex-M4 es la próxima generación de cores de 32 bit que ofrece bajo consumo de energía, mejoras en las características de debug, y alto nivel para soportar integración de bloques. Incorpora un pipeline de 3 etapas, arquitectura Harvard con buses separados para datos e instrucciones y un tercer bus para periféricos. Incluye una unidad interna de prebúsqueda que soporta ejecución especulativa en bifurcaciones. Soporta procesamiento de señales en un ciclo e instrucciones SIMD (Single Instruction Múltiple Data). También, tiene integrado en el core hardware para procesamiento en punto flotante.

El coprocesador ARM Cortex-M0 es de uso eficiente de energía, de 32 bit compatible en código y herramientas con el Cortex-M4. El coprocesador Cortex-M0, está diseñado para reemplazar los microcontroladores existentes de 8/16-bit.

Otras características del ARM Cortex-M4 son:

- Memoria interna
- Soporta protección de memoria (Memory Protection Unit) de ocho regiones
- Nested Vectored Interrupt Controller (NVIC)
- Unidad de Hardware de punto flotante
- Entrada de interrupción no enmascarable (Non-maskable Interrupt)
- JTAG y Serial Wire Debug (SWD), serial trace y ocho breakpoints
- System tick timer

El Cortex-M0 es capaz de liberar la carga del procesador ARM Cortex-M4. En él se destaca:

- Velocidad de reloj de 204 MHz
- JTAG, Serial Wire Debug, NVIC built-in
- Memoria on-chip, hasta 1 MB flash de dos bancos
- 16 kB de memoria de datos on-chip EEPROM
- 136 kB de SRAM para código y uso de datos
- Múltiples bloques SRAM con acceso separado al bus
- 64 kB de ROM con el código de arranque y drivers de software on-chip
- 32 bit de memoria de propósito general One-Time Programmable (OTP)
- Interfaz Serial GPIO (SGPIO)
- Subsistema State Configurable Timer (SCT) en AHB
- Dispone de un Global Input Multiplexer Array (GIMA) para conectar múltiples entradas y salidas hacia periféricos manejados por eventos

2.2 Sistemas Operativos Evaluados

La base de este trabajo de investigación radica en la comparación entre un sistema operativo estático, FreeOSEK, y uno dinámico, FreeRTOS, teniendo en cuenta que estos tienen una estructura de desarrollo diferente.

En los dinámicos, la alocación de memoria, creación y definición de tareas, semáforos y demás características propias del sistema, se llevan a cabo en tiempo ejecución, permitiendo así mayor flexibilidad en el desarrollo con menos límites en el manejo de tareas. Esto lleva a una reducción del control que tiene el programador en la ejecución del programa y por ende genera algunos riesgos que implican no cumplir tiempos límites.

Luego, en los sistemas estáticos, las tareas y todas sus características, los recursos que utilizan, los eventos y las interrupciones que manejan, conjunto con la alocación de memoria, deben ser definidas antes de compilar el código, durante la programación. Esta forma de desarrollo le brinda al programador un mayor control sobre la aplicación, debido a un mayor determinismo en la ejecución de tareas y los tiempos de respuesta en cada caso. Sin embargo, esta ventaja implica un mayor esfuerzo en la programación y limitaciones en tiempo de ejecución, ya que si se define erróneamente algún aspecto de la planificación o de una tarea, se debe programar el código nuevamente. Además, la alocación estática requiere una mayor cantidad de memoria respecto al esquema dinámico, ya que durante la ejecución no es posible liberar el espacio reservado.

2.2.1 FreeRTOS

FreeRTOS [5] [6] [7] [8] [9] es un sistema operativo de tiempo real para sistemas embebidos que puede funcionar sobre 35 arquitecturas de microcontroladores. Se distribuye bajo licencia GPL con una excepción: permite que el código propietario de los usuarios siga siendo código cerrado, manteniendo el núcleo en sí como código abierto, lo que facilita el uso de FreeRTOS en aplicaciones propietarias. Está formado por un conjunto de archivos en lenguaje C. A su vez, en algunas arquitecturas, incluye algunas líneas de código ensamblador, mayormente en las rutinas del planificador.

FreeRTOS provee:

- Poco uso de memoria
- Bajo nivel de sobrecarga
- Ejecución rápida
- Métodos para múltiples hilos o tareas
- Mutexes y semáforos
- Timers de software
- Reducción del tick para aplicaciones de bajo consumo.
- Prioridades para las tareas
- Cuatro esquemas de asignación de memoria
- Es de código abierto: está bajo continuo desarrollo, y no hay costo de implementación.
- Amplia comunidad de usuarios: ayuda a resolver problemas que surgen.

El método de múltiples hilos cambia tareas en función de la prioridad y un esquema de planificación round-robin. El planificador puede ser configurado para operar en forma apropiativa o colaborativa. Las tareas tienen una prioridad de ejecución, donde 0 es la menor prioridad (se recomienda usar referencias a tskIDLE_PRIORITY +1,+2, etc).

Cada tarea tiene una prioridad asignada entre 0 y un valor definido en compilación configMAX_PRIORITIES – 1. Es decir, si configMAX_PRIORITIES es 5, FreeRTOS tiene 5 niveles de prioridad entre 0 y 4.

2.2.2 FreeOSEK

Es un sistema operativo de tiempo real especialmente diseñado para sistemas embebidos críticos donde la ejecución determinística del sistema es un punto esencial [12].

Fue creado por la industria automotriz y es de libre licencia. Su ventaja respecto de otros SOE es que es escalable y esta basado en las especificaciones OSEK-VDX, un estándar. Por ende, existen muchas implementaciones (open source y cerradas). Además, existen muchas empresas que proveen soporte.

El planificador OSEK usa una FIFO por prioridad. Existen tareas que pueden ser interrumpidas (SCHEDULE = FULL) y otras que no (SCHEDULE = NON). En la configuración se pueden dar tres opciones: que todas las tareas puedan ser interrumpidas, que ninguna de ellas o una mezcla de ambos tipos de scheduling. En función a esto, el sistema operativo puede contener una parte del código, la otra o ser un scheduling con todas las posibilidades y que se decida en tiempo de ejecución cual tarea puede o no ser interrumpida.

Este SOE ofrece rutinas de servicio de interrupción (ISR) de dos tipos:

- ISR1: Sin llamadas a la API, más simple y más rápido
- ISR2: Con llamadas a la API, puede llamar a algunas primitivas del SOE

Las ISR1 tienen siempre prioridad más alta que las ISR2 y, estas últimas, ejecutan el planificador al final de la rutina.

2.4 Sistema de Pruebas

Para llevar a cabo las pruebas, se plantearon escenarios específicos que son de interés para el desarrollo de los Sistemas de Tiempo Real, evaluando el desempeño y validándolo, en algunos casos, a través del tiempo de respuesta de ambos SOE.

Para obtener los tiempos en ambos SOE, se utilizó la misma estrategia. Esta consiste en la consulta y modificación del registro Cycle Count Register del procesador [11].

2.4.1 Latencia en Interrupciones

En este punto, se procedió a medir el tiempo de respuesta de cada uno de los SOE frente a la aparición de una interrupción externa.

Para ello, se utilizaron dos pines de la placa de desarrollo, un pin de salida que genera la interrupción y un pin de entrada que la captura. El procedimiento correspondiente se detalla en el siguiente esquema:

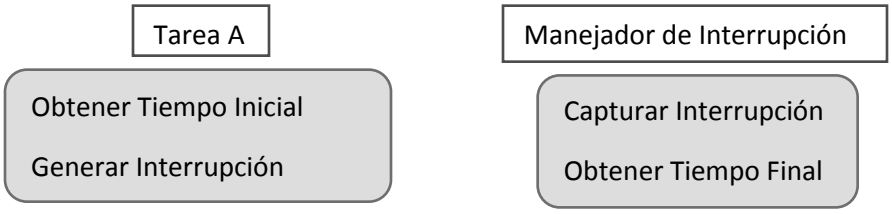

Fig. 1: Esquema de medición de la latencia de interrupción

Luego de medir los tiempos iniciales y finales, se realiza la resta de los mismos para obtener la latencia de interrupción. Se realizó este proceso de manera repetida para obtener un valor promedio de respuesta.

2.4.2 Inversión de prioridades

El caso de inversión de prioridades [10] es un problema muy común en la ejecución de tareas de un sistema. Ocurre cuando dos tareas de distinta prioridad utilizan un mismo recurso y la tarea de menor prioridad lo bloquea para poder utilizarlo de manera consistente. Luego, si una o varias tareas de mayor prioridad adquieren el procesador, el recurso quedara bloqueado hasta que la tarea de menor prioridad lo libere. Por tanto, si durante ese proceso se ejecuta la tarea de mayor prioridad que necesita el recurso, la misma quedará bloqueada a la espera de la liberación del mismo. Si en este lapso una tarea de prioridad intermedia se apropia del procesador, el bloqueo puede extenderse a un tiempo indeterminado.

Para la resolución de este problema, FreeOSEK ofrece la estrategia de Techo de Prioridad (Priority Ceiling), que consiste en darle a la tarea que utiliza el recurso el valor máximo de prioridad para evitar que le quiten el procesador y pueda ejecutarse hasta liberar el recurso. Por otro lado, FreeRTOS implementa Herencia de Prioridad (Priority Inheritance), donde la tarea de baja prioridad hereda el mayor valor de prioridad de las tareas en espera de ese recurso.

2.4.3 Manejo de tareas

Se realizaron pruebas básicas de manejo de tareas, creación, destrucción y suspensión en cada uno de los SOE, evaluando los tiempos de transición entre diferentes estados de las mismas, incluyendo el tiempo de intervención del planificador.

3 Resultados

A modo de comparación, se detallan, en cada una de las siguientes tablas, los tiempos obtenidos en la experimentación.

3.1 Interrupciones

Como muestra la Tabla 1, FreeOSEK permite la utilización de dos tipos de interrupciones. Por un lado, las de categoría 1, sin intervención del SOE, y las de categoría 2 en las cuales el vector de interrupciones llama a un manejador definido dentro del SOE.

En FreeRTOS ninguna función del sistema operativo tiene intervención en el manejo de las interrupciones.

Interrupción	Tiempo
FreeOSEK - Categoría 2	7,65 µs
FreeOSEK - Categoría 1	0,90 µs
FreeRTOS	0,50 µs

Tabla 1: Tiempos latencia de interrupción

Se observa, de los tiempos de respuesta que se muestran en la tabla anterior, que los valores se encuentran en el orden de los microsegundos en todos los casos.

Luego, en el caso de las interrupciones de tipo 2 en FreeOSEK, producen que la intervención del SOE introduzca un overhead adicional por la ejecución del planificador.

Por otra parte, cuando el evento es directamente manejado por el vector de interrupciones del procesador (NVIC), los tiempos disminuyen un orden respecto a cuándo se requiere la intervención del planificador.

3.2 Manejo de Tareas

La Tabla 2 muestra los diferentes tiempos que se obtuvieron en el manejo de tareas en cada SOE.

Cabe aclarar que, la alocación de memoria para cada tarea no se tiene en cuenta en estas mediciones. FreeOSEK, al ser estático, realiza este procedimiento en tiempo de compilación. Por esta razón, este tiempo no influye en el funcionamiento del sistema.

En cambio, en FreeRTOS, la declaración de las tareas se realiza durante la ejecución del sector principal del programa, antes del inicio del planificador y de la ejecución de las tareas.

FreeRTOS, al ser dinámico, proporciona la posibilidad de crear tareas en tiempo de ejecución. Es en este momento cuándo se realiza la alocación de memoria de todos los recursos que se definen para dicha tarea. Para este caso, se realizó una medición de la creación de la tarea y su puesta en marcha por el planificador, resultando en un valor de 22µs.

-	Tiempo - FreeOSEK	Tiempo - FreeRTOS
Cambio de contexto	4,55 µs	5,55 µs
Suspensión de Tarea	4,25 µs	4,10 µs
Suspensión - Activación	5,10 µs	4,70 µs

Tabla 2: Tiempos de manejo de tareas

En la tabla anterior se observa que los tiempos de respuesta en el manejo de tareas se encuentran en el orden de los microsegundos, como exigen los Sistemas de Tiempo Real. Además, en ambos SOE, los valores para cada escenario son similares.

Los tiempos relacionados a la suspensión de tareas son mínimamente inferiores en el caso de FreeRTOS. Por otro lado, en el caso del cambio de contexto, dónde una de las tareas termina y la otra pasa al estado de ejecución, existe una breve diferencia a favor de FreeOSEK.

4 Conclusión

En este trabajo se presenta una comparación de dos Sistemas Operativos Embebidos fundamentalmente seleccionados por sus características respecto a la manera en que se realiza la alocación de recursos y tareas.

Los valores obtenidos comprueban que ambos son aptos para controlar un Sistema de Tiempo Real, ya que poseen tiempos de respuesta del orden de los microsegundos, independientemente de que el tipo de SOE sea dinámico o estático.

A la hora de seleccionar un tipo de SOE, los tiempos de respuesta no son determinantes. En cambio, el tamaño de memoria disponible es un factor que influye directamente en la elección. Esta característica juega a favor de los sistemas dinámicos, debido a que la cantidad de tareas que pueden ser creadas durante el ciclo de vida del sistema, no está limitada. En cambio, en los sistemas estáticos, el número de tareas se encuentra restringido por el tamaño de la memoria del sistema embebido. Por otro lado, la fiabilidad del sistema es una ventaja de los sistemas estáticos ya que las fallas por falta de memoria se verifican en tiempo de compilación.

Se plantea como trabajo futuro realizar pruebas sobre la nueva versión de FreeRTOS (9.0.0) la cual da soporte a tareas y recursos estáticos. Además, sería interesante evaluar los mismos parámetros frente a una mayor sobrecarga del sistema, por ejemplo, a través del incremento de la cantidad de tareas.

5 Bibliografía

1. N.C. Audsley , A. Burns , M. F. Richardson , A. J. Wellings: Hard Real-Time Scheduling: The Deadline-Monotonic Approach. Proc. IEEE Workshop on Real-Time Operating Systems and Software (1991).
2. L. Buhr. “An Introduction to Real Time Systems”. Prentice Hall (1999).
3. A. Burns, A. J. Wellings: Designing hard real-time systems. Proceedings of the 11th Ada-Europe international conference on Ada. ISBN:0-387-55585-4 (1992).

4. A. Burns & A. Wellings: Real-Time Systems and Programming Languages. Addison Wesley, ISBN 90-201-40365-x.
5. R. Barry. Using the FreeRTOS Real Time Kernel.
6. C. Becker. RTOS en sistemas embebidos, disponible en http://ieee.eie.fceia.unr.edu.ar/PDF_RTOS.pdf
7. A. Celery. Sistemas Operativos de Tiempo Real, disponible en http://www.sase.com.ar/2011/files/2010/11/SASE2011Introduccion_RTOS.pdf
8. FreeRTOS, disponible en http://en.wikipedia.org/wiki/FreeRTOS
9. FreeRTOS, disponible en http://www.freertos.org/
10. L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. IEEE Trans. Comput., 39(9):1175{1185, September 1990.
11. Cortex-M4 Technical Reference Manual, disponible en http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0439b/BABJFFGJ.html
12. Introducción a OSEK-OS, El Sistema Operativo del CIAA-Firmware, disponible en http://proyecto ciaa.com.ar/devwiki/lib/exe/fetch.php?media=desarrollo: firmware:rtos:introduccion_a_osek-os.pdf
13. Documentación EDU-CIAA-NXP, disponible en http://www.proyecto-ciaa.com.ar/devwiki/doku.php?id=desarrollo:edu-ciaa:edu-ciaa-nxp