Caracterización temporal de los servicios del sistema operativo de tiempo real MaRTE OS

Marta Alonso, Mario Aldea y Michael González Harbour
Departamento de Electrónica y Computadores
Universidad de Cantabria
39005-Santander, SPAIN
marta.alonso@gmail.com, aldeam@unican.es, mgh@unican.es

Resumen

En este trabajo se describe el diseño y desarrollo de una herramienta que permite evaluar la respuesta temporal de algunos de los servicios que componen el perfil de "Sistemas mínimos de tiempo real" del estándar POSIX de interfaces de sistemas operativos portables. La herramienta de caracterización desarrollada se ha probado con diferentes sistemas operativos POSIX. En este trabajo se detalla su aplicación a la caracterización temporal del sistema operativo de tiempo real MaRTE OS.

1. Introducción[1]

Dado que el sistema operativo es el elemento que más restringe la capacidad de reacción de un sistema de tiempo real, resulta de vital importancia conocer el comportamiento temporal de sus servicios. Esta información facilita el análisis y la evaluación del sistema operativo en su fase de desarrollo, lo que permite detectar posibles "puntos débiles" en el diseño. Por otro lado, esta evaluación facilita al usuario un conjunto de datos con los que puede realizar comparaciones entre diferentes sistemas operativos que ofrezca el mercado, o simplemente comprobar si uno en concreto cumple los requisitos temporales necesarios para el sistema en que va a ser implementado. Por último, facilita el análisis del tiempo de respuesta de las aplicaciones que corren bajo ese determinado sistema.

Por este motivo, se propone ofrecer al diseñador una herramienta portable que le permita evaluar las respuestas temporales de los servicios de diferentes sistemas operativos. Para ello, nos hemos centrado en el estándar de interfaces de sistemas operativos portables (POSIX) [6], ya que la mayoría de los sistemas operativos comerciales que existen hoy en día para aplicaciones de tiempo real siguen este estándar.

POSIX es un estándar muy grande, por lo que al objeto de acotar la dimensión del trabajo y al mismo tiempo proporcionar una herramienta que sea útil nos hemos centrado en el más pequeño de sus subconjuntos, definidos en el estándar POSIX.13 [7]: es el conocido como perfil de sistema de tiempo real mínimo (PSE51). Este subconjunto define los servicios necesarios para sistemas empotrados relativamente pequeños y con requisitos de tiempo real.

Al objeto de probar la herramienta desarrollada y mostrar su portabilidad se ha utilizado para caracterizar dos sistemas operativos que siguen en mayor o menor medida el estándar POSIX: MaRTE OS [3] y el conocido sistema operativo Linux. El primero es un sistema operativo de tiempo real, mientras que el segundo es de propósito general y no está diseñado para ofrecer comportamiento de tiempo real. Los resultados completos de la caracterización se pueden encontrar en [1]. En este trabajo sólamente comentamos los resultados obtenidos para MaRTE OS, al ser más relevantes desde el punto de vista del tiempo real.

Aunque muchas instituciones disponen de programas de caracterización temporal similares al que se expone en este trabajo, son pocos los que se han generalizado. Durante la década de los ochenta y principios de los noventa, los miembros de PIWG, "Performance Issues Working Group", crearon y distribuyeron un grupo de pruebas, conocido como el "PIWG Test Suite", que consistía en

1. Este trabajo ha sido financiado parcialmente por el *Ministerio de Educación y Ciencia* con el proyecto TIC 2002-04123-C03 (TRECOM)

Actas del Simposio de Sistemas de Tiempo Real, pp.53-60
ISBN: 84-9732-448-X

más de 200 pruebas, incluyendo pruebas de evaluación de la velocidad del procesador (Whetstone), medida de ejecución de sentencias de código por unidad de tiempo (Dhrystone), así como otras pruebas sobre varios servicios del lenguaje Ada y sus implementaciones bajo compiladores específicos [9].

Este conjunto de pruebas estaba destinado al lenguaje Ada83, y a medida que se fue extendiendo el uso de Ada 95 este grupo de trabajo decidió definir un nuevo conjunto de pruebas de evaluación, que dio lugar al ACES, "Ada Compiler Evaluation Systems" [8]. ACES permite realizar pruebas de rendimiento, análisis de la gestión de software, así como determinar el funcionamiento de los sistemas de compilación y ejecución de Ada. También proporciona herramientas para examinar el sistema de diagnóstico de la implementación, el sistema de gestión de librerías, y un depurador simbólico, así como para medir el tiempo de compilación y de ejecución de la implementación. Este grupo de pruebas está orientado a Ada 95, y está compuesto de aproximadamente 2120 pruebas. No hemos encontrado programas de caracterización temporal para sistemas operativos de tipo POSIX.

El artículo está organizado del siguiente modo. En primer lugar aparece un amplio resumen del sistema operativo MaRTE OS, ya que es conveniente conocer sus características y los servicios que ofrece, para entender correctamente la herramienta de caracterización temporal. En el apartado 3 se define el conjunto de los servicios que van a ser caracterizados. En el apartado 4 se explican algunas características generales de la herramienta de caracterización temporal, y se dan algunos detalles de un servicio concreto, a modo de ejemplo. En el apartado 5 se describen los principales resultados obtenidos para MaRTE OS. Por último en el apartado 6 aparecen las conclusiones de este trabajo.

2. MaRTE OS

MaRTE OS es un sistema operativo de tiempo real para aplicaciones empotradas que sigue el perfil "Sistema de Tiempo Real Mínimo" definido en el estándar POSIX.13 [7]. MaRTE OS proporciona el soporte necesario para ejecutar aplicaciones concurrentes sobre una máquina "desnuda", proporcionando, además del soporte multitarea, facilidades para la inicialización del sistema y el lanzamiento de la aplicación, y para la gestión de los dispositivos.

Puesto que está destinado a ser utilizado en aplicaciones de tiempo real, uno de los principales requisitos en el diseño de MaRTE OS ha sido que todos los servicios que proporciona tengan tiempos de respuesta acotados.

MaRTE OS dispone de un entorno de desarrollo cruzado basado en el compilador GCC y sus "frontales" para diferentes lenguajes de programación (GNAT, GCJ). Sobre MaRTE OS es posible ejecutar aplicaciones concurrentes escritas en C, Ada, C++ y Java de tiempo real[1].

Aunque MaRTE OS se ha diseñado para ser fácilmente portable a diversas arquitecturas, el desarrollo inicial ha sido realizado para la arquitectura PC. Además del núcleo del sistema operativo, para esta arquitectura, MaRTE OS también incluye manejadores para algunos de los dispositivos más utilizados, como son el teclado, la pantalla, los puertos serie y paralelo, algunas tarjetas de red, etc.

Además de la versión para PC "desnudo", MaRTE OS también puede utilizarse como una librería de threads POSIX para Linux, lo que permite ejecutar las aplicaciones como procesos de usuario en un sistema Linux convencional.

La mayor parte del código de MaRTE OS ha sido escrito utilizando el lenguaje de programación Ada 95, con pequeñas partes en C y ensamblador.

MaRTE OS se distribuye como código libre bajo licencia GPL.

2.1. Funcionalidad soportada

MaRTE OS proporciona a las aplicaciones un subconjunto de toda la funcionalidad POSIX, que puede ser implementado en un núcleo de sistema operativo pequeño y eficiente apto para su utilización en sistemas empotrados pequeños con requerimientos de tiempo real.

La funcionalidad descrita en el perfil "Sistema de Tiempo Real Mínimo" consiste básicamente en el soporte para la concurrencia a nivel de threads. No se incluyen en este perfil el soporte para un sistema de ficheros completo (aunque existe un sistema de ficheros reducido para los dispositivos de

1. El soporte para Java de tiempo real se encuentra actualmente en desarrollo

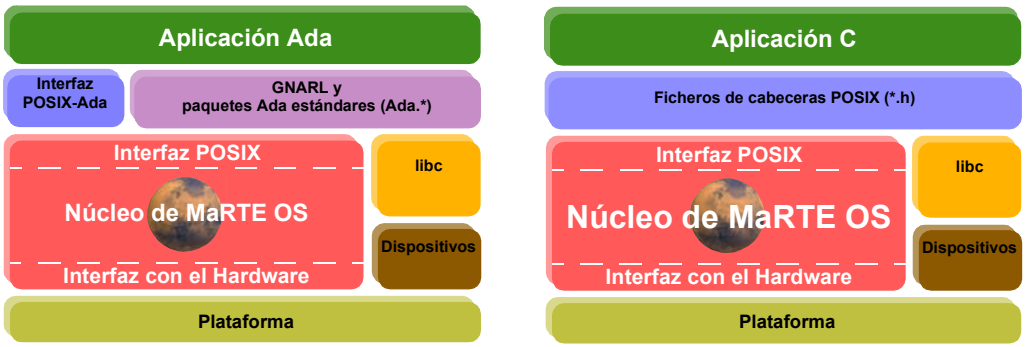

Figura 1 Aplicaciones Ada y C ejecutando sobre MaRTE OS

entrada/salida) ni la existencia de múltiples procesos. Los servicios proporcionados por MaRTE OS son:

- Gestión de threads: creación, finalización, atributos, ...
- Planificación basada en prioridades: se soportan las políticas FIFO, *round-robin* y servidor esporádico.
- Threads independientes y sincronizados.
- Mutexes y variables condicionales.
- Semáforos.
- Señales de tiempo real.
- Reloj monotónico y de tiempo real. Temporizadores.
- Relojes y temporizadores de tiempo de ejecución.
- Servicios de temporización de threads: suspensión absoluta y relativa.
- Dispositivos de Entrada/Salida.
- Gestión de memoria dinámica.

Las aplicaciones escritas en Ada y Java requieren una librería de tiempo de ejecución que dé soporte a la concurrencia definida por los respectivos lenguajes. Ambas librerías de tiempo de ejecución han sido portadas a MaRTE OS, de forma que aplicaciones escritas en esos lenguajes pueden hacer uso de sus propias primitivas para programación concurrente. El portado de estas librerías se ha visto facilitado enormemente por el hecho de que ambas se basen en una interfaz POSIX como la facilitada por nuestro sistema operativo MaRTE OS.

Además de la funcionalidad incluida en el "Sistema de Tiempo Real Mínimo", en MaRTE OS hemos implementado otros servicios que consideramos muy interesantes para las aplicaciones de tiempo real:

- Planificación definida por la aplicación: mediante esta interfaz las aplicaciones pueden definir los algoritmos de planificación con los que quieren planificar sus tareas [2]. La interfaz ha sido propuesta para su inclusión en el estándar POSIX.
- Gestión de interrupciones a nivel de aplicación: se proporciona una interfaz para gestionar las interrupciones hardware basada en el borrador de un estándar POSIX ("*Interrupt Control API*" P1003.2X/D1.0. Febrero de 2001).
- Marco para la instalación de manejadores de dispositivo: facilita la instalación de manejadores de dispositivo en MaRTE OS.

2.2. Arquitectura

La Figura 1 muestra las distintas capas software que intervienen en la ejecución de una aplicación sobre MaRTE OS. La parte central la constituye el núcleo de MaRTE OS, el cual implementa la funcionalidad incluida en el perfil "Sistema de Tiempo Real Mínimo" que ha sido citada en el apartado anterior. La interfaz del núcleo con la plataforma se realiza a través de una interfaz abstracta con el hardware, que proporciona operaciones para gestión de las interrupciones, acceso al reloj y al temporizador del sistema y cambio de contexto entre tareas. La existencia de esta capa software permite

independizar el diseño del núcleo de la plataforma sobre la que se ejecuta, facilitado por tanto su portabilidad.

A pesar de estar escrito en Ada, el núcleo presenta a las aplicaciones una interfaz POSIX para lenguaje C, lo que supone dos importantes ventajas:

- Las aplicaciones C pueden utilizar las operaciones del núcleo de forma directa, sin más que utilizar las funciones definidas en los ficheros de cabeceras POSIX.
- La librería de ejecución del compilador GNAT (GNARL) está basada en funciones POSIX, por lo que su adaptación a MaRTE OS resulta sencilla. Lo mismo ocurre con otras librerías, como es el caso de la librería de tiempo de ejecución del lenguaje Java.

2.3. Entorno de desarrollo

El desarrollo de las aplicaciones en MaRTE OS se realiza en un entorno cruzado, compuesto por un "Computador de Desarrollo" y una plataforma de ejecución o "Sistema Empotrado".

Como computador de desarrollo se utiliza un PC con Linux como sistema operativo en el que se han instalado los compiladores GCC y GNAT así como la distribución de MaRTE OS. Como plataforma de ejecución basta con un PC con procesador 80386 o superior que disponga de algún dispositivo de arranque (disquete, Flash-RAM, disco duro, etc.).

El ciclo de desarrollo de una aplicación es muy rápido, consistiendo en los siguientes pasos:

1. La aplicación se compila y enlaza en el sistema de desarrollo utilizando las herramientas proporcionadas por MaRTE OS.
2. El sistema empotrado ejecuta un programa de arranque por red que descarga la aplicación desde el equipo de desarrollo.
3. La aplicación ejecuta libremente en el sistema empotrado o es depurada de forma remota desde el equipo de desarrollo a través de la línea serie utilizando el depurador GDB.

Una vez finalizado su desarrollo, la aplicación puede ser cargada directamente desde el dispositivo de arranque, por lo que en el destino final de la aplicación no es necesario disponer del equipo de desarrollo.

3. Servicios caracterizados

Se han elegido para su caracterización los servicios usuales en aplicaciones de tiempo real de acuerdo con la experiencia del grupo de Computadores y Tiempo Real de la Universidad de Cantabria en el desarrollo de controladores de robots industriales. Estos servicios son los que pasan a enumerarse en las siguientes líneas:

Mutexes. Para los protocolos de herencia de prioridad, `PTHREAD_PRIO_INHERIT`, y de protección de prioridad, `PTHREAD_PRIO_PROTECT`, se han analizado la creación y destrucción de un mutex, el cambio y obtención del valor del techo de prioridad, la toma del mutex (normal y condicional), y la liberación en varios casos: en un thread de alta prioridad cuando no hay otro thread esperando en el mutex, en un thread de baja prioridad cuando uno de mayor prioridad está esperando el acceso, y cuando hay más threads de mayor prioridad esperándolo.

Variables Condicionales. Para este servicio se analizan la creación y destrucción de una variable condicional, la señalización individual y conjunta para threads de diferentes prioridades, y la espera con tiempo límite cuando ese límite se alcanza.

Gestión del tiempo: Se incluyen aquí los servicios de relojes de alta resolución, temporizadores, y retrasos de alta resolución en modo absoluto y relativo. En primer lugar, se contrasta la resolución teórica de los relojes de tiempo real y monotónico, con las reales obtenidas en el proceso de medida. En segundo lugar, se analizan los tiempos reales de activación de un thread después de un retraso de alta resolución, tanto en modo relativo como absoluto, para diferentes intervalos temporales habituales en sistemas de tiempo real. En tercer lugar se consideran los temporizadores en modo simple relativo. Para ellos se compara el tiempo teórico y real de activación y expiración en el thread creado para manejar su expiración. Por último, se evalúa el cambio de contexto de threads de diferentes prioridades cuando experimentan retrasos de alta resolución.

Señales de tiempo real. Dada su importancia en la gestión y manejo de eventos en los sistemas de

tiempo real, se caracteriza el intervalo temporal necesario para realizar el envío de señales, a uno o varios threads de diferentes prioridades que estaban esperando dicha señales mediante los mecanismos propios de espera.

Gestión de memoria dinámica. Con objeto de realizar una caracterización temporal de los intervalos de tiempo característicos en la reserva y liberación de bloques de memoria de forma dinámica, se han realizado experimentos con diferentes tamaños de bloques y bajo diferentes circunstancias.

Interrupciones. Se evalúa el tiempo durante el cual un thread de usuario no puede ejecutar debido a la ejecución de rutinas de atención a interrupción. Para ello se detectan intervalos de tiempo en que la lectura del reloj desde un thread de alta prioridad excede de un valor umbral, que se establece en el doble del tiempo de respuesta normal de esa función.

El resto de servicios que conforman el perfil de un "Sistema de Tiempo Real Mínimo" descrito en el perfil del estándar POSIX que nos ocupa han sido omitidos en la caracterización realizada por no considerarse tan relevantes en las operaciones de tiempo real que habitualmente se emplean en aplicaciones reales.

4. Estrategia de medida

La estructura seguida para llevar a cabo la caracterización temporal de sistemas operativos de tiempo real mínimo consta de uno o varios experimentos concretos para cada uno de los servicios, así como de unas librerías de uso común, que permiten por un lado, realizar las operaciones pertinentes con las estructuras temporales con las que trabaja, y por otro facilitan la representación de los resultados obtenidos, así como la transmisión de dichos resultados desde la plataforma de ejecución hacia un equipo de desarrollo. El lenguaje de programación elegido para esta herramienta es el C, ya que está más extendido entre los sistemas operativos de tiempo real comerciales.

Con objeto de reducir el error en la estimación de la medida, a los tiempos de ejecución se les decrementa siempre el tiempo que tarda el sistema en leer el reloj en el mejor de los casos.

Es preciso indicar que las medidas obtenidas no permiten determinar con garantía absoluta los tiempo de respuesta de peor caso, ya que pueden darse en la práctica situaciones peores a las observadas. La obtención de valores garantizados de tiempo de respuesta de peor caso es muy compleja, y requiere del uso de técnicas especiales que se escapan a los objetivos de este trabajo.

4.1. Ejemplo de estrategia de medida: Mutexes

Dado que el desarrollo de las estrategias de medida para la caracterización de cada uno de los servicios considerados sería demasiado extenso, se describe a continuación, a modo de ejemplo, la estrategia seguida para caracterizar el mecanismo de sincronización implementado mediante mutexes.

Los parámetros de configuración para la realización de este experimento son:

- Número de grupos de medidas: 14; ya que se lleva a cabo el análisis de 7 posibles situaciones para cada uno de los protocolos de prioridad del mutex: herencia de prioridad, y protección de prioridad.
- Número de threads que se ejecutan simultáneamente: 10; debido a que es el número habitual de tareas simultáneas en un robot implementado con un sistema de las características propuestas en este trabajo.

Para poder considerar los eventos bajo diferentes circunstancias, se divide el análisis en casos. Todos ellos tienen las siguientes características comunes:

- La política de planificación es `SCHED_FIFO`, porque es la más comúnmente implementada en los sistemas de tiempo real.
- El thread principal, `main()` es el de mayor prioridad, para garantizar que se creen en primer lugar los threads necesarios para los diferentes casos. Una vez creados los threads del experimento, el thread principal se queda suspendido, a la espera de la finalización del experimento.
- Los threads para la realización de los experimentos se crean con los siguientes atributos:
  - `PTHREAD_EXPLICIT_SCHED`: para poder aplicar diferentes atributos de planificación a los threads, y que estos no los hereden del thread principal.

- `PTHREAD_CREATE_JOINABLE`: para la espera sincronizada a la terminación de los threads desde el thread principal.

En primer lugar, y sin atender a los parámetros de configuración, se analizan los tiempos de creación y destrucción del mutex así como de las funciones de configuración del techo de prioridad. Posteriormente se realizan las siguientes medidas:

Medida 1: Tomar y liberar un mutex, sin que haya otro thread esperándolo.

Se analizan los tiempos de toma, toma condicional, y liberación del mutex. Para ello se crea un thread que accederá a tomar y liberar este objeto de sincronización, sin que haya ningún otro thread esperando para acceder a él, es decir, con el acceso al mutex libre. Se toman en cuenta los protocolos de herencia de prioridad y protección de prioridad.

Medida 2: Liberación de un mutex desde un thread de baja prioridad más tiempo de activación de un thread de mayor prioridad que estaba esperándolo

Dos threads de diferentes prioridades intentan acceder al mutex. El thread de menor prioridad se activa antes, toma el mutex, y justo antes de liberarlo inicia la medida. El thread de mayor prioridad se activa durante el intervalo en que el thread de menor prioridad tiene el mutex tomado, e intenta tomar el mutex, esperando a que quede libre para terminar en ese momento de realizar la medida. Los protocolos analizados son los dos considerados en el análisis anterior.

En la Figura 2 podemos observar el proceso de medida seguido para este análisis con el protocolo de herencia de prioridad

Medida 3: Liberación de un mutex en un thread de baja prioridad más tiempo de activación de un thread de mayor prioridad que estaba esperándolo (varios threads)

Este último caso sigue el mismo procedimiento que el anterior, salvo por la diferencia de que son varios los threads que se están esperando al mutex. El que termina la medida es el de mayor prioridad, mientras que el resto de threads que estaban esperando, lo único que hacen es tomar y liberar el mutex cuando por orden de prioridad les vaya correspondiendo el acceso. De este modo podemos verificar el tiempo de ejecución de las operaciones del mutex cuando hay varios threads esperando, en lugar de uno solo.

5. Resultados obtenidos para MaRTE OS

La aplicación práctica de los experimentos, se ha llevado a cabo en un equipo que actúa como plataforma de ejecución de un sistema distribuido para controlar robots. Trabaja con un procesador Pentium III a 750 MHz, y está conectado por el puerto serie al sistema de desarrollo, mediante el puerto serie COM1. La versión de MaRTE OS empleada para realizar los experimentos es la versión 1.56, junto con los compiladores gcc y Gnat GAP 1.0.

Consideraciones generales sobre los resultados:

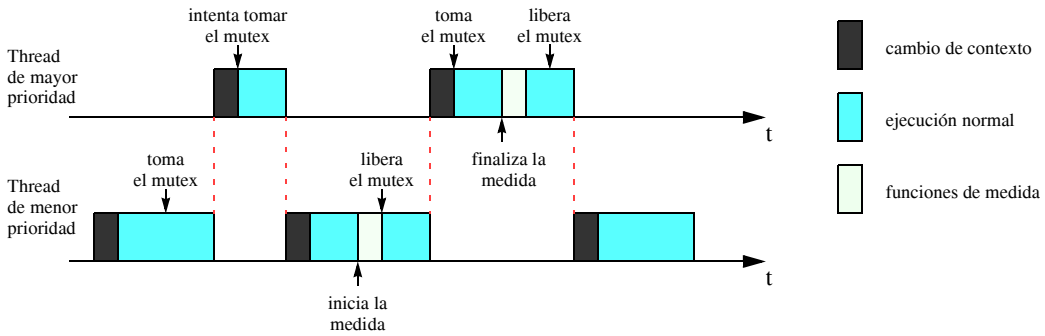

Figura 2. Diagrama temporal para la "medida 2" de los mutexes, con el protocolo de herencia de prioridad

- Como era previsible, los tiempos de la primera medida de cada experimento, que se almacenan separadamente del resto, son habitualmente bastante superiores al resto. Esto se debe a los efectos de carga de la caché y de la "pipeline" del procesador.
- En general, salvo para la primera medida, las diferencias entre los tiempos mínimos y máximos son pequeñas.
- El tiempo de respuesta prácticamente no varía en función del número de threads que estén siendo ejecutados o que esperan a un recurso.

A continuación se muestra un extracto de los resultados más interesantes y se realiza un breve análisis de los resultados obtenidos para cada uno de los servicios evaluados. Los valores temporales que aparecen en las tablas se refieren a los tiempos de peor caso, exceptuando la primera medida.

Mutexes (Tabla 1 y Tabla 2): Los tiempos de ejecución con el protocolo por protección de prioridad son claramente menores que con el protocolo de herencia de prioridad cuando no hay contención. Los tiempos promedio de toma y liberación de mutexes oscilan, según el caso, entre 0,23 µs y 5 µs.

Tabla 1. Resultados para mutexes, herencia de prioridad

Operación	**Tiempo (µs)**
Tomar un mutex libre (lock)	0.923
Liberar un mutex desde un thread de alta prioridad con la cola de espera libre	1.521
Liberar un mutex desde un thread de baja prioridad y activar un thread de mayor prioridad que está esperando	4.866
Idem, con 10 threads esperando	4.403

Tabla 2. Resultados para mutexes, protección de prioridad

Operación	**Tiempo (µs)**
Tomar un mutex libre (lock)	0.240
Liberar un mutex desde un thread de alta prioridad con la cola de espera libre	0.244
Liberar un mutex desde un thread de baja prioridad y activar un thread de mayor prioridad que está esperando	4.984
Idem, con 10 threads esperando	4.338

Variables Condicionales (Tabla 3): Los tiempos por señalización individual y por "broadcast" son semejantes. Los tiempos promedio de señalización y espera de variables condicionales oscilan, según el caso, entre 0,23 µs y 5 µs.

Tabla 3. Resultados para variables condicionales

Operación	**Tiempo (µs)**
Señalizar desde un thread de baja prioridad más activar un thread de alta prioridad que espera la condición	6,014
Señalizar desde un thread de alta prioridad cuando un thread de menor prioridad espera la condición	0,541

Gestión del tiempo (Tabla 4): La resolución media teórica de los relojes es de 1ns. Pero los valores reales, que incluyen el tiempo del servicio de medida son algo superiores, y rondan los 600 ns. En relación a la ejecución un retraso de alta resolución relativo, para intervalos de suspensión menores de 20 µs (valor que es configurable) no se llega a ejecutar la función, sino que se ejecuta una operación de cesión de la CPU (*yield*) en su lugar. Por este motivo, se registra un valor de aproximadamente 1,2 µs. En tiempos promedios, la ejecución del retraso relativo ronda los 4 µs mientras que para el retraso en modo absoluto este valor es cercano a los 2 µs.

Tabla 4. Resultados para las funciones de retraso

Instante de activación: Intervalo de	**Con retraso relativo (µs)**	**Con retraso absoluto(µs)**
1 µs	1,219	3,946
10 µs	1,145	12,694
100 µs	104,705	102,416
1000 µs	1004,127	1002,137

Señales de Tiempo Real (Tabla 5): El tiempo para la señalización con datos asociados es ligeramente superior al de señalización si ellos. Esto mismo ocurre con los mecanismos de recepción. Los tiempos de ejecución promedio oscilan entre 2,3 y 3,8 µs.

Gestión de memoria (Tabla 6): MaRTE OS utiliza TLSF, un sistema de gestión de memoria de tiempo real desarrollado en la Universidad Politécnica de Valencia [4]. Los tiempos de reserva de bloques son independientes del tamaño solicitado. Los tiempos para la liberación de bloques son mucho menores. El valor promedio para reservar

Tabla 5. Resultados para señales de tiempo real

Operación	Tiempo (μs)
Señalizar desde un thread de baja prioridad más activar un thread de alta prioridad que espera la señal	3,428
Idem, añadiendo información a la señal	3,847
Señalizar desde un thread de alta prioridad cuando un thread de baja prioridad espera la señal	2,451

un bloque de memoria oscila entorno a los 0,86 μs, mientras que el valor promedio para la liberación de un bloque de memoria es 0,45 μs. Los peores casos se muestran en la tabla

Tabla 6. Resultados para la gestión de memoria

Tamaño del bloque	malloc() (μs)	free() (μs)
100 bytes	1,542	1,062
1000 bytes	1,400	1,026
10000 bytes	1,418	1,642

Sobrecarga producida por interrupciones: No ocurrían prácticamente nunca.

6. Conclusiones

Se ha desarrollado un herramienta portable que permite conocer el comportamiento temporal de los principales servicios de un sistema operativo de tiempo real que sea conforme al perfil mínimo del estándar POSIX de tiempo real.

En este trabajo se describen los resultados de la aplicación de esta herramienta al sistema operativo MaRTE OS. Podemos concluir que todos los servicios analizados ofrecen tiempos de respuesta predecibles, con escasa diferencia entre los casos peor, mejor y promedio, si exceptuamos el tiempo de la primera medida, que es mucho mayor debido al tiempo de carga de la caché y la "pipeline" del procesador.

Aunque la herramienta desarrollada ofrece los tiempos observados durante cada experimento y por tanto no garantiza las medidas de tiempos de peor caso, los resultados obtenidos son útiles para evaluar el comportamiento de un determinado sistema operativo, compararlo con otros, y estimar de manera aproximada el comportamiento temporal.

Referencias

[1] Marta Alonso. "Caracterización temporal de sistemas operativos de tiempo real". Proyecto fin de carrera, ETSIIT, Universidad de Cantabria, Marzo 2005.

[2] Mario Aldea Rivas, Michael González Harbour. "POSIX-Compatible Application Defined-Scheduling in MaRTE OS" Proceedings of 14th Euromicro Conference on Real-Time Systems, Vienna, Austria, IEEE Computer Society Press, Junio 2002

[3] Mario Aldea Rivas y Michael González Harbour. "MaRTE OS: Minimal Real-Time Operating System for Embedded Applications" (página Web). http://marte.unican.es

[4] Miguel Masmano Tello, Ismael Ripoll, Alfons Crespo and Jorge Real. "TLSF: A New Dynamic Memory Allocator for Real-Time Systems". 16th Euromicro Conference on Real-Time Systems, 2004

[5] M. Aldea and M. González. "MaRTE OS: An Ada Kernel for Real-Time Embedded Applications". Proceedings of the International Conference on Reliable Software Technologies, Ada-Europe-2001, Leuven, Belgium, Lecture Notes in Computer Science, LNCS 2043, May, 2001.

[6] ISO/IEC 9945-1:2003. Standard for Information Technology -Portable Operating System Interface (POSIX).

[7] IEEE Std. 1003.13-2003. Information Technology - Standardized Application Environment Profile-POSIX Realtime and Embedded Application Support (AEP). The Institute of Electrical and Electronics Engineers.

[8] ACES. Ada Compilers Test Suites. http://www.adaic.org/compilers/articles/benchmrk.html

[9] PIGW. Performance Issues Working Group. SIGAda, Special Interest Group in Ada. http://sigada.org/wg/piwg/piwg.html