POSIX DE TIEMPO REAL

Michael González Harbour[1]
Departamento de Electrónica
Universidad de Cantabria
Avda. los Castros s/n
39005 - Santander
Correo electrónico: mgh@ccucvx.unican.es

Resumen

El estándar POSIX define una interfase portable para aplicaciones basadas en el popular sistema operativo UNIX[2]. El principal objetivo de este estándar es la portabilidad de las aplicaciones a nivel de código fuente, mediante la unificación de las diferentes versiones del UNIX. Una parte importante del POSIX aborda las necesidades de las aplicaciones de tiempo real. La portabilidad de estas aplicaciones es hoy en día prácticamente imposible debido a la gran cantidad de sistemas operativos y núcleos de tiempo real existentes. Aunque el UNIX no ha sido hasta ahora un sistema operativo para sistemas de tiempo real, es posible adaptarlo a estos sistemas si se le añaden los servicios necesarios, y se eliminan también aquellas funciones que dificultan implementaciones pequeñas y eficientes. En este artículo[3] se comentarán las extensiones de tiempo real del POSIX y cómo estas extensiones permiten abordar las necesidades de aplicaciones con requerimientos de tiempo real.

1 INTRODUCCION

1.1 El Estándar POSIX

El término POSIX corresponde a las iniciales de interfase de sistema operativo portable (Portable Operating System Interface). Es un estándar de interfase de sistema operativo, basado en el popular sistema operativo UNIX[2]. El estándar POSIX está actualmente en desarrollo, y su principal objetivo es permitir la portabilidad de aplicaciones a nivel de código fuente, es decir, que sea posible portar una aplicación de un computador a otro sin más que recompilar su código. Está siendo desarrollado por la *Computer Society* de IEEE, con la referencia IEEE-P1003. También está siendo estandarizado a nivel internacional con la referencia ISO/IEC-9945.

El POSIX es un grupo de estándares en evolución. Cada uno de los estándares que lo componen cubre diferentes aspectos de los sistemas operativos. Algunos de ellos ya han sido aprobados, mientras que otros están aún en fase de desarrollo. Los estándares POSIX se pueden agrupar en tres categorías diferentes:

---

[1] Este trabajo ha sido financiado en parte por la Comisión Interministerial de Ciencia y Tecnología, a través de los proyectos ROB91-0288 y ROB91-1553-E.

[2] UNIX es una marca registrada de AT&T.

[3] Este artículo está basado en el artículo en inglés [2].

Tabla I. Lista de estándares base del POSIX

POSIX.1	Interfases del sistema (estándar básico)[a,b]
POSIX.2	*Shell* y utilidades[a]
POSIX.3	Métodos para medir la conformidad con POSIX[a]
POSIX.4	Extensiones de tiempo real
POSIX.4a	Extensión de *threads*, o múltiples flujos de control
POSIX.4b	Extensiones adicionales de tiempo real
POSIX.6	Extensiones de seguridad
POSIX.7	Administración del sistema
POSIX.8	Acceso a ficheros transparente a la red
POSIX.12	Interfases de red independientes del protocolo
POSIX.15	Extensiones de colas *batch*
POSIX.17	Servicios de directorios

[a] Estándares IEEE ya aprobados
[b] Estándar ISO/IEC ya aprobado

1) *Estándares Base*: Definen interfases del sistema relacionadas con diferentes aspectos del sistema operativo. El estándar especifica la sintaxis y la semántica de estos servicios del sistema operativo, de modo que los programas de aplicación puedan invocarlos directamente. El estándar no especifica cómo se implementan estos servicios; de este modo, los implementadores de sistemas pueden elegir la implementación que crean más conveniente—y así competir entre ellos—, siempre que cumplan la especificación de la interfase. Todos los estándares base desarrollados hasta el momento lo han sido para lenguaje C. En el momento de escribir este artículo está abierto el debate sobre si los estándares base deben desarrollarse de forma independiente del lenguaje, y luego especificar interfases concretas para los diferentes lenguajes de programación. La Tabla I y la Tabla II muestran los estándares base que están siendo desarrollados por los grupos de trabajo del POSIX.

Tabla II. Estándares base POSIX adicionales

P1224	Servicios de mensajería electrónica (X.400)
P1224.1	Interfase para portabilidad de aplicaciones X.400
P1238	Interfase de comunicaciones OSI
P1238.1	Interfase OSI de transferencia de ficheros
P1201.1	Interfase gráfica a usuario (ventanas)

2) *Interfases en diferentes lenguajes de programación*: Son estándares secundarios que traducen a un lenguaje de programación concreto los estándares base. Los lenguajes utilizados hasta el momento son Ada, Fortran 77, y Fortran 90, además del lenguaje C, en el que se han especificado hasta el momento los estándares base. La Tabla III muestra las interfases POSIX que están actualmente en desarrollo para diferentes lenguajes de programación.

Tabla III. Lista de interfases POSIX para diferentes lenguajes de programación

POSIX.5	Interfases Ada[a]
POSIX.9	Interfases Fortran 77[a]
POSIX.19	Interfases Fortran 90
POSIX.20	Interfases Ada para las extensiones de tiempo real
[a] Estándares IEEE ya aprobados	

3) *Entorno de Sistemas Abiertos*. Estos estándares incluyen una guía al entorno POSIX y los perfiles de entornos de aplicación. Un perfil de aplicación es una lista de los estándares POSIX, con especificación de las opciones y parámetros necesarios, que se requieren para un cierto entorno de aplicación. El objetivo principal de los perfiles de aplicación es conseguir un conjunto pequeño de clases de implementaciones de sistemas operativos bien definidas y que sean apropiadas para entornos particulares de aplicaciones. La Tabla IV muestra la lista de estándares que están siendo desarrollados en este grupo.

Tabla IV. Lista de estándares POSIX de entornos de aplicaciones

POSIX.0	Guía al entorno POSIX de sistemas abiertos
POSIX.10	Perfil de entorno de aplicaciones de supercomputación
POSIX.11	Perfil de entorno de aplicaciones de procesado de transacciones
POSIX.13	Perfiles de entornos de aplicaciones de tiempo real
POSIX.14	Perfil de entorno de aplicaciones multiprocesadoras
POSIX.18	Perfil de entorno de aplicación de plataforma POSIX

La necesidad del desarrollo de un estándar de sistema operativo se deriva del hecho de que, aunque el UNIX es un estándar *de facto*, hay suficientes diferencias entre las diferentes implementaciones para que las aplicaciones no sean completamente portables. Más aún, si una aplicación UNIX puede necesitar ciertos cambios para ser portada a una plataforma diferente, la portabilidad de las aplicaciones de tiempo real es muchísimo más difícil, ya que existe una gran diversidad de sistemas operativos de tiempo real. El UNIX no es un sistema operativo de tiempo real, y no existe un estándar *de facto* para estas aplicaciones.

1.2 Estandarización de Sistemas Operativos de Tiempo Real

Debido a la necesidad de conseguir la portabilidad de las aplicaciones de tiempo real, se estableció en el POSIX un grupo de trabajo de tiempo real. Este grupo desarrolla estándares para añadir al POSIX básico (o UNIX) los servicios de sistema operativo necesarios para poder desarrollar aplicaciones de tiempo real. Estas aplicaciones se caracterizan porque el funcionamiento correcto no sólo depende de los resultados del cálculo, sino también del instante en el que se generan estos resultados. Con objeto de garantizar que los cálculos se realizan en los instantes requeridos, es preciso que el sistema de tiempo real tenga un comportamiento temporal predecible, y para ello, es preciso también que los servicios del sistema operativo sean capaces de proporcionar el nivel de servicio requerido con un tiempo de respuesta acotado. El objetivo principal del grupo de trabajo de tiempo real del POSIX es "desarrollar estándares que sean los mínimos cambios y adiciones a los estándares POSIX para soportar la portabilidad de aplicaciones con requerimientos de tiempo real".

Muchas aplicaciones de tiempo real, y especialmente los sistemas empotrados, tienen restricciones físicas especiales que imponen el uso de sistemas operativos con un conjunto reducido de funciones o servicios del sistema. Por ejemplo, existen muchos sistemas que no disponen de disco duro, no tienen unidad hardware de manejo de memoria (MMU), o tienen poca memoria. Para estos sistemas es necesario que el estándar permita implementaciones que sólo soporten un subconjunto de los servicios POSIX. Los subconjuntos necesarios para las aplicaciones de tiempo real han sido abordados por el grupo de trabajo de tiempo real, que ha propuesto cuatro perfiles para entornos de aplicaciones de tiempo real: sistemas empotrados pequeños, controladores de tiempo real, sistemas empotrados grandes, y sistemas grandes con requerimientos de tiempo real.

De acuerdo con estos requerimientos, el grupo de trabajo de tiempo real está actualmente desarrollando cuatro estándares, que se tratarán en este artículo:

POSIX.4*:*
: *Extensiones de tiempo real*. Define interfases para soportar la portabilidad de aplicaciones con requerimientos de tiempo real.

POSIX.4a:
: *Extensión de threads*. Define interfases para soportar múltiples *threads* o flujos de control dentro de cada proceso POSIX.

POSIX.4b:
: *Extensiones adicionales de tiempo real*. Define interfases para soportar servicios de tiempo real adicionales.

POSIX.13:
: *Perfiles de entornos de aplicaciones de tiempo real*. Cada perfil especifica una lista de los servicios que se requieren para un entorno de aplicación particular.

Los estándares base POSIX.4, POSIX.4a y POSIX.4b están especificados para lenguaje C. Existe un grupo de trabajo en el POSIX dedicado a la especificación de interfases Ada, que produjo ya las interfases Ada al estándar base POSIX.1 [15], y está actualmente desarrollando las interfases Ada para las extensiones de tiempo real, bajo el nombre POSIX.20 [17].

Las secciones siguientes de este artículo tratan los aspectos más relevantes de cada uno de los estándares mencionados. La discusión se basa en el estado de estos estándares en el momento de escribir el artículo, que corresponde al borrador 14 del POSIX.4 [12], Borrador 7 del POSIX.4a [13], Borrador 7 del POSIX.4b [14], Borrador 5 del POSIX.13 [16], y Borrador 2 del POSIX.20 [17]. Puesto que todos estos estándares están aún en desarrollo, los cambios que sufran pueden afectar a lo expuesto en este artículo. Sin embargo, pensamos que la esencia de los temas que se discuten aquí será aplicable a los estándares que se aprueben finalmente.

2 EXTENSIONES DE TIEMPO REAL

Esta sección muestra algunas de las funciones más importantes del POSIX.4 [12], que es la parte del POSIX que define interfases del sistema para soportar aplicaciones con requerimientos de tiempo real. El estándar POSIX.4 está muy cercano a su aprobación definitiva.

2.1 Planificación de Procesos de Tiempo Real

El estándar base POSIX.1 [11] define un modelo con actividades concurrentes denominadas procesos, pero no especifica ninguna política de planificación ni ningún concepto de prioridad. Para que las aplicaciones de tiempo real puedan ser portables, es preciso especificar políticas de planificación que permitan obtener tiempos de respuesta predecibles. El POSIX.4 define tres políticas de planificación; cada proceso, a través de un atributo de planificación, puede elegir la que desee:

- SCHED_FIFO: Es una política de planificación expulsora basada en prioridades estáticas[4], en la que los procesos con la misma prioridad se atienden en el orden de llegada (cola FIFO). Está política tendrá al menos 32 niveles de prioridad.

- SCHED_RR: Esta política es muy similar a SCHED_FIFO, pero emplea un método de rodaja temporal (*round-robin*) para planificar procesos de la misma prioridad. También tiene 32 niveles de prioridad como mínimo.

- SCHED_OTHER: Es una política de planificación definida por la implementación.

La planificación expulsora de prioridad estática es una estrategia de prioridad utilizada con mucha frecuencia para sistemas de tiempo real. Es muy sencilla, y permite alcanzar altos niveles de utilización del sistema si se realiza la asignación de prioridades de acuerdo con los métodos del ritmo monotónico (*rate monotonic*)[6] o plazo monotónico (*deadline monotonic*)[5]. Con las políticas de planificación especificadas en el estándar, junto a las funciones asociadas que permiten modificar y leer las políticas y prioridades de cada proceso, es posible planificar aplicaciones de tiempo real en sistemas operativos POSIX. En [9] aparece una buena introducción al diseño y análisis de este tipo de sistemas de tiempo real utilizando resultados recientes de planificación de prioridades estáticas; en [3] se puede encontrar una referencia más exhaustiva.

[4] Empleamos aquí el término prioridad estática para indicar una prioridad que no cambia con el tiempo o las condiciones de carga del sistema. Sin embargo, la prioridad estática puede cambiar a requerimiento del proceso en cualquier instante.

2.2 Inhibición de la Memoria Virtual

Aunque el estándar POSIX.1 no requiere que las implementaciones suministren mecanismos de memoria virtual, es práctica común en los sistemas UNIX el proporcionarlos. La memoria virtual presenta grandes ventajas para aplicaciones que no son de tiempo real, pero introduce una gran incertidumbre en la respuesta temporal. Con objeto de acotar los tiempos de acceso a memoria—y por tanto la respuesta temporal de la aplicación—el POSIX.4 define funciones para bloquear en memoria física o bien todo el espacio de direccionamiento de un proceso, o bien rangos seleccionados de ese espacio. Estas funciones deberán de ser utilizadas por los procesos con requerimientos temporales estrictos, así como por aquellos procesos con los que se sincronicen. De esta forma, se pueden conseguir tiempos de respuesta predecibles.

2.3 Sincronización de Procesos

El POSIX.4 define funciones para permitir la sincronización de procesos a través de semáforos contadores. Estos semáforos se identifican por un nombre que pertenece a un espacio de nombres definido por la implementación. Este espacio de nombres puede coincidir o no con el espacio de nombres de ficheros, por lo que no se hace necesaria la existencia del sistema de ficheros para utilizar los semáforos. El semáforo contador es un mecanismo de sincronización muy común, que permite el acceso mutuamente exclusivo a recursos compartidos, la señalización y espera entre procesos, y otros tipos de sincronización. Uno de los usos más comunes de los semáforos es permitir que diferentes procesos puedan compartir datos; esto se consigue en POSIX.4 utilizando objetos de memoria compartida (ver la sección 2.4), junto con los semáforos.

Desafortunadamente, los semáforos contadores especificados en POSIX.4 no evitan el fenómeno conocido por *inversión de prioridad no acotada* [8]. La inversión de prioridad ocurre cuando un proceso de prioridad alta tiene que esperar a que un proceso de prioridad baja termine de utilizar un determinado recurso que tiene reservado. Utilizando los protocolos de sincronización adecuados, se puede conseguir que la inversión de prioridad esté acotada por la duración de secciones críticas, es decir, de las secciones de código durante las cuales el proceso reserva un recurso para su uso exclusivo. Sin embargo, con los semáforos convencionales, puede aparecer inversión de prioridad no acotada; esto significa que el retraso experimentado por tareas de prioridad alta no está acotado por la duración de secciones críticas, sino que depende del tiempo de ejecución total de tareas de prioridad más baja. Esta situación puede ocurrir cuando una tarea de alta prioridad está esperando a que una tarea de prioridad baja libere un semáforo que controla el acceso a un recurso compartido, y la tarea de prioridad baja es expulsada—del uso de la CPU— por una tarea de prioridad intermedia. La Figura 1 muestra un ejemplo de esta situación. Los largos retrasos que se experimentan en estos casos son normalmente inaceptables para las tareas con requerimientos de tiempo real estricto. Si se utilizan protocolos de sincronización apropiados [8], la cantidad de inversión de prioridad puede quedar acotada a la duración de secciones críticas, que es normalmente muy pequeña. En la sección 3.3, se comentará un mecanismo de sincronización diferente—el *mutex*—que evita la inversión de prioridad no acotada y, opcionalmente, puede ser utilizado para sincronizar diferentes procesos.

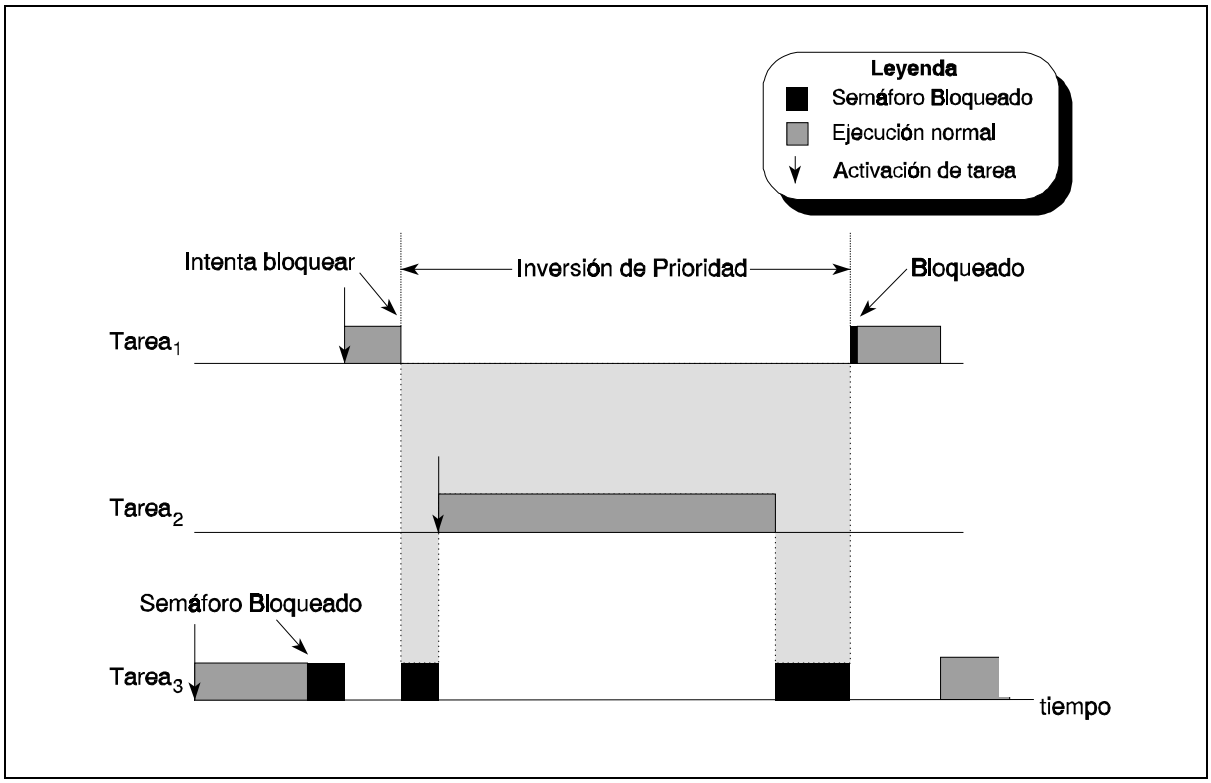

Figura 1. Ejemplo de inversión de prioridad no acotada

2.4 Memoria Compartida

Los procesos POSIX.1 tienen espacios de direccionamiento que son independientes entre sí. Sin embargo, muchas aplicaciones de tiempo real (y también muchas que no son de tiempo real) necesitan compartir grandes cantidades de datos de una manera eficiente. Esto se puede conseguir si los procesos son capaces de compartir regiones de memoria física. Con este propósito, el estándar POSIX.4 define los objetos de memoria compartida, que son regiones de memoria que pueden ser mapeadas en el espacio de direcciones de un proceso. Cuando dos o más procesos mapean el mismo objeto de memoria entonces comparten la región de memoria asociada. Si los objetos de datos que se colocan en memoria compartida requieren un acceso mutuamente exclusivo por parte de distintos procesos, se pueden utilizar semáforos para efectuar estos accesos. Al igual que en el caso de los semáforos, los objetos de memoria compartida se identifican por un nombre que pertenece a un espacio de nombres dependiente de la implementación.

También es posible en POSIX.4 mapear ficheros en memoria. La información del fichero se escribe o lee en memoria principal directamente, y al cerrar el fichero el sistema actualiza la información en la memoria secundaria. Los ficheros mapeados en memoria también pueden ser compartidos por varios procesos.

2.5 Señales de Tiempo Real

El mecanismo de señales definido en el POSIX.1 permite notificar eventos que ocurren en el sistema, pero no es completamente satisfactorio para aplicaciones de tiempo real. Las señales no se almacenan en colas y, por tanto, algunos eventos se pueden perder. La señales no están priorizadas, y esto implica tiempos de respuesta más largos para eventos

urgentes. Además, los eventos del mismo tipo producen señales con el mismo número, que son indistinguibles. Puesto que muchas aplicaciones de tiempo real están basadas en el rápido intercambio de eventos en el sistema, el POSIX.4 ha extendido la interfase de señales para conseguir las siguientes características:

- Las señales de tiempo real se guardan en colas, por lo que los eventos no se pierden.

- Las señales de tiempo real pendientes de procesado se extraen de la cola en orden de prioridad, usando el número de señal como prioridad. Esto permite diseñar aplicaciones con tiempos de respuesta más rápidos ante eventos urgentes.

- Las señales de tiempo real contienen un campo adicional de información, que la aplicación puede utilizar para intercambiar datos entre el generador de la señal y el módulo que la procesa. Por ejemplo, este campo puede ser utilizado para identificar la fuente y la causa de la señal.

- El rango de señales disponibles para la aplicación ha sido expandido.

2.6 Comunicación Entre Procesos

Se especifica un mecanismo sencillo de colas de mensajes para la comunicación entre procesos. Las colas de mensajes están identificadas por un nombre perteneciente a un espacio de nombres dependiente de la implementación. Los mensajes tienen asociado un campo de prioridad, y se extraen de las colas en orden de prioridad. Esta facilidad permite minimizar la cantidad total de inversión de prioridad en el sistema. La recepción y la transmisión de mensajes puede hacerse tanto de forma bloqueante—si es necesario el proceso se suspende hasta que llegue un mensaje o haya espacio en la cola—como no bloqueante. La transmisión y la recepción no están sincronizadas, es decir, el transmisor no necesita esperar a que el receptor haya recibido el mensaje. Los tamaños máximos de mensajes y colas se pueden seleccionar durante la creación de la cola, lo que permite incrementar la predecibilidad de las operaciones con colas de mensajes.

2.7 Relojes y Temporizadores

Se define un reloj de tiempo real que debe tener una precisión de al menos 20 ms. El tiempo se representa con resolución de nanosegundos, por lo que si una implementación dispone de un reloj hardware de alta precisión, lo puede aprovechar con plena resolución. También se pueden crear temporizadores que cuentan intervalos de tiempo utilizando como base temporal el reloj de tiempo real u otros relojes definidos por la implementación. Cuando el intervalo especificado en un temporizador ha transcurrido, se envía una señal al proceso que lo creó. Existen diferentes opciones para los temporizadores, tales como disparo único, activación periódica, etc., que permiten manejar el tiempo de forma flexible y sencilla. Se define también una función para dormir un proceso durante un intervalo relativo especificado en nanosegundos.

2.8 Entrada/Salida Asíncrona

El POSIX.4 define funciones que permiten solapar el procesado de aplicaciones con las operaciones de entrada/salida iniciadas por la aplicación. Una operación de entrada/salida asíncrona es similar a las operaciones de entrada/salida normales, con la excepción de que

una vez que la operación asíncrona ha sido iniciada por un proceso, este proceso no se suspende y puede continuar ejecutando instrucciones, en paralelo con la operación de entrada/salida. Cuando la operación se termina, es posible enviar una señal a la aplicación.

2.9 Entrada/Salida Sincronizada

El POSIX.4 define también funciones para que las operaciones de lectura y escritura normales se realicen de forma sincronizada. Una escritura sincronizada es aquella que sólo se completa cuando los datos han sido transferidos correctamente—por ejemplo escritos en el medio físico en el que se almacena el fichero— y cuando la información del sistema de ficheros necesaria para recuperar estos datos ha sido también transferida correctamente. Para que una operación de lectura sincronizada se complete es preciso que, si hay operaciones pendientes de escritura que afecten a los datos a leer, estas operaciones de escritura se completen—antes de la lectura de los datos—de forma sincronizada. El software de tiempo real suele tener una gran interacción con el hardware asociado, y la entrada/salida sincronizada permite a la aplicación dejar en un estado conocido las operaciones de entrada/salida realizadas sobre el hardware del sistema.

3 EXTENSION DE *THREADS*

El modelo de procesos del POSIX.1 no es completamente adecuado para aquellos sistemas que requieren alta eficiencia y procesan gran cantidad de eventos en intervalos pequeños de tiempo, debido a que los procesos tienen tiempos de cambio de contexto muy elevados, el tiempo necesario para crearlos o destruirlos es muy elevado, se necesita hardware especial (MMUs) para proporcionar a cada proceso un espacio de direcciones independiente, y el modelo no es adecuado para sistemas multiprocesadores de memoria compartida. En la mayoría de los núcleos de tiempo real comercialmente disponibles para sistemas empotrados pequeños, el modelo de concurrencia está basado en tareas que comparten el mismo espacio de direccionamiento y tienen un estado asociado poco voluminoso, comparado con los procesos POSIX. El Grupo de Trabajo de Tiempo Real consideró estas características y decidió desarrollar la extensión de *threads*.

El estándar POSIX.4a define interfases para soportar múltiples actividades concurrentes, denominadas *threads*, dentro de cada proceso POSIX. Los *threads* definidos en el POSIX.4a tienen un estado asociado más pequeño que el de un proceso. Todos los *threads* que pertenecen al mismo proceso comparten el mismo espacio de direccionamiento. Pueden ser implementados con tiempos de cambio de contexto y de creación y destrucción más bajos que los de los procesos. El POSIX.4a ha sido específicamente desarrollado para abordar las necesidades de los sistemas multiprocesadores de memoria compartida. Con estas características, el modelo de *threads* está mucho más próximo al modelo de concurrencia de los núcleos de tiempo real comerciales que el modelo de procesos. Los *threads* no sólo están pensados para aplicaciones de tiempo real, sino que también pueden ser empleados para sistemas que, no siendo de tiempo real, requieren cambios de contexto eficientes y tiempos de creación/destrucción pequeños, como en aplicaciones de ventanas, software multiprocesador, etc.

Los *threads* pueden usar todas las funciones definidas en el POSIX.4 y POSIX.1, además de las funciones definidas específicamente para *threads* en el POSIX.4a. Las funciones más relevantes del POSIX.4a se describen a continuación.

3.1 Control de *Threads*

Estas funciones permiten controlar la creación y terminación de *threads*, así como las funciones relacionadas con estas operaciones. Se definen funciones para crear un *thread*, esperar a la terminación de un *thread*, terminar un *thread* de forma normal, manejar identificadores de *threads*, etc. Asimismo se definen funciones para el manejo de los atributos de creación de un *thread*, tales como el tamaño de *stack*.

3.2 Planificación de *Threads*

Las políticas de planificación definidas para los *threads* son las mismas que se definen para los procesos en el POSIX.4—expulsoras por prioridad, con tratamiento en cola o en rodaja temporal de los *threads* con igual prioridad. Sin embargo, la existencia de los *threads* introduce un problema nuevo, ya que puede haber dos tipos de planificadores coexistiendo en el sistema: el planificador de procesos, y el planificador de *threads*. Por este motivo, se define el concepto de *dominio de contención* de un *thread*, que es el conjunto de *threads* con los que compite por el uso de la CPU. Dependiendo de los tipos de dominio de contención permitidos, pueden existir tres tipos básicos de implementaciones:

- *Planificación Global*: Todos los *threads* tienen dominio de contención global—o de sistema—, y por tanto cada *thread* se planifica compitiendo con todos los demás *threads* del sistema, sin que el proceso al que pertenecen tenga ninguna importancia. El planificador, por tanto, funciona sólo al nivel de *threads*, y los parámetros de planificación de los procesos se ignoran.

- *Planificación Local*. Los *threads* sólo compiten con otros *threads* pertenecientes al mismo proceso. La planificación se realiza en dos niveles. Primero, se planifican los procesos entre sí. Después, los *threads* del proceso (o procesos) seleccionado compiten entre ellos por el uso de la CPU.

- *Planificación Mixta*. Algunos *threads* tienen dominio de contención global—o de sistema—, y otros tienen dominio de contención local—o de proceso—. La planificación se realiza a dos niveles. En primer lugar, se planifican los procesos y los *threads* de dominio global. En segundo lugar, si es necesario, se planifican los *threads* con dominio local pertenecientes al proceso (o procesos) seleccionado.

Tanto los sistemas de planificación global como los de planificación mixta ofrecen las mejores perspectivas para la mayoría de las aplicaciones de tiempo real, ya que permiten planificar todos los distintos objetos concurrentes que tengan requerimientos temporales estrictos, al mismo nivel. Los sistemas con planificación mixta pueden además realizar planificación local para algunos *threads* concretos. La planificación local es normalmente mucho más eficiente y rápida que la planificación global. Sin embargo, sólo debe utilizarse para grupos de *threads* cuya prioridad sea globalmente menor o mayor que las prioridades de otros grupos de *threads* en el sistema, es decir, cuando ningún otro *thread* en el sistema necesite tener una prioridad efectiva situada entre los niveles de prioridad de los *threads* del grupo. El motivo de esta restricción es que la prioridad del proceso, y no la prioridad de los *threads*, es la que se utiliza para planificar el grupo de *threads* con dominio local. Los mismos criterios se aplican a sistemas con planificación sólo local, lo que se traduce en que este tipo de planificación es poco adecuada para una gran parte de los sistemas de tiempo real estricto.

3.3 Sincronización de *Threads*

Se definen dos primitivas de sincronización para *threads*: los *mutex*—o secciones mutuamente exclusivas—, y las variables condicionales. Los *mutex* se utilizan en la sincronización de *threads* para el acceso mutuamente exclusivo a recursos compartidos. Son similares a los semáforos, pero requieren que el *thread* que bloquea el *mutex*—denominado el propietario de ese *mutex*— sea el mismo que los libera. Las variables condicionales se pueden utilizar para espera y señalización de eventos entre *threads*, aunque su uso está ligado al de los *mutex* en una forma similar a las secciones críticas condicionales[5]. La espera a una variable condicional se puede especificar con un tiempo máximo de espera (*timeout*). Ambas primitivas de sincronización pueden ser opcionalmente utilizadas por *threads* pertenecientes a diferentes procesos.

Los *mutex* se definen con tres protocolos de sincronización opcionales:

- NO_PRIO_INHERIT: La prioridad del *thread* no depende de sus relaciones de propiedad sobre ningún *mutex* (se dice que un *thread* es propietario del *mutex* que bloquea).

- PRIO_INHERIT: El *thread* propietario de un *mutex* hereda las prioridades de los *threads* que están en espera de adquirir el *mutex*. Este es el protocolo de herencia básica de prioridad [8].

- PRIO_PROTECT: Cuando un *thread* adquiere un *mutex*, hereda la prioridad denominada *techo de prioridad* del *mutex*, que se define generalmente como la prioridad de la tarea de prioridad más alta que puede bloquear ese *mutex*. La aplicación asigna el techo de prioridad a cada *mutex*; con los techos de prioridad adecuados, el funcionamiento es el denominado protocolo de protección de prioridad, también denominado emulación del protocolo de techo de prioridad [4][9].

La inversión de prioridad no acotada (ver Sección 2.3) se puede evitar utilizando los protocolos de herencia básica de prioridad o de protección de prioridad, y así se pueden conseguir altos niveles de utilización en sistemas con requerimientos de tiempo real estricto. El protocolo de protección de prioridad, con las definiciones apropiadas de techo de prioridad, se puede utilizar también para evitar un tipo de inversión de prioridad especial que aparece en los sistemas multiprocesadores, denominada *bloqueo remoto*. En [7] aparece una descripción detallada del bloqueo remoto y la sincronización en multiprocesadores.

---

[5] Existen dos operaciones básicas asociadas con las variables condicionales: la señalización y la espera. Cuando un *thread* inicia la espera a una variable condicional se suspende, al mismo tiempo que libera un *mutex*. Cuando otro *thread* señaliza la misma variable condicional, el *thread* suspendido se reactiva y, de forma atómica, adquiere el *mutex* liberado al comenzar la espera. El *mutex* se puede usar para el intercambio de información—de forma mutuamente exclusiva—entre el *thread* que espera y el que señaliza.

3.4 Otras Funciones

En el POSIX.4a se definen otras funciones para el manejo de datos asociados a cada *thread*, la cancelación de *threads*, envío de señales a *threads*, así como versiones reentrantes de otras funciones definidas en el POSIX.1. Para mayor información sobre estas funciones se puede recurrir al borrador del estándar [13].

4 EXTENSIONES ADICIONALES DE TIEMPO REAL

El estándar POSIX.4b define extensiones adicionales de tiempo real para soportar la portabilidad de aplicaciones con requerimientos de tiempo real. La razón por la que se dividen las extensiones de tiempo real en dos estándares (y quizás un tercero que se iniciaría próximamente) ha sido el facilitar una aprobación más rápida de los servicios que se consideraron esenciales para tiempo real—aquellos especificados en el POSIX.4—, dejando otros servicios de tiempo real también convenientes pero menos necesarios para un segundo estándar.

Puesto que el POSIX.4b ha comenzado su proceso de estandarización más tarde que el POSIX.4, los servicios que se incluyen en los borradores actuales tienen más posibilidades de cambiar que los del POSIX.4. A continuación se describen brevemente los servicios que están siendo estandarizados en el POSIX.4b:

4.1 Tiempos Límite (*Timeouts*)

Algunos de los servicios definidos en el POSIX.1, POSIX.4, y POSIX.4a pueden suspender al proceso que los invoca durante un período indefinido de tiempo, hasta que los recursos necesarios para completar el servicio estén disponibles. En sistemas de tiempo real estricto es importante limitar la cantidad máxima de tiempo que un proceso puede emplear esperando a que uno de estos servicios se complete. Esto permite detectar condiciones anormales, y por tanto incrementa la robustez del programa permitiendo implementaciones tolerantes a fallos. Los tiempos límite especifican la máxima cantidad de tiempo que el proceso puede estar suspendido en espera de la terminación de un servicio. Los servicios elegidos para tener tiempos límite han sido aquellos que todavía no tenían capacidad de especificar un tiempo límite, y cuyo uso se consideró más probable en los segmentos de código en los que la respuesta temporal es crítica:

- Esperar a que un semáforo se desbloquee
- Esperar a la llegada de un mensaje a una cola de mensajes
- Enviar un mensaje a una cola de mensajes
- Esperar a que un *mutex* sea liberado.

4.2 Relojes de Tiempo de Ejecución

Se define un reloj opcional de tiempo de CPU para cada proceso y para cada *thread*, y se utiliza la interfase POSIX.4 de relojes y temporizadores para manejar estos relojes. Además de la medida del tiempo de CPU, que es especialmente útil en sistemas de tiempo real para caracterizar y analizar el sistema, se pueden crear temporizadores basados en los relojes de tiempo de CPU, con objeto de detectar el consumo de una cantidad excesiva de tiempo de ejecución por parte de un proceso o *thread*. De esta forma, se puede detectar durante la ejecución si ha habido errores de software, o errores en la estimación de los

tiempos de ejecución de peor caso. La detección de la situación en la que una tarea excede el tiempo de ejecución de peor caso asumido durante la fase de análisis es muy importante en sistemas de tiempo real robustos, porque si los tiempos asumidos no se cumplen, los resultados del análisis de planificabilidad ya no son válidos, y el sistema puede incumplir sus requerimientos temporales. Los relojes de tiempo de ejecución permiten detectar cuando ocurre un consumo excesivo de tiempo de CPU, para activar las acciones apropiadas de manejo de esta condición de error.

4.3 Servidor Esporádico

Se define una nueva política de planificación—llamada SCHED_SPORADIC—que implementa el algoritmo de planificación del servidor esporádico [10]. Esta política puede ser utilizada para procesar eventos aperiódicos al nivel de prioridad deseado, permitiendo garantizar los requerimientos temporales de tareas de prioridad inferior. El servidor esporádico proporciona tiempos de respuesta rápidos y hace predecibles los sistemas que procesan eventos aperiódicos.

4.4 Control de Interrupciones

Muchos sistemas de tiempo real requieren poder capturar interrupciones generadas por dispositivos hardware especiales, y gestionar estas interrupciones desde el programa de aplicación. Las funciones propuestas en el estándar permiten a un proceso o *thread* capturar una interrupción a través de la asignación de una rutina de servicio de interrupción escrita por el usuario, suspender la ejecución del proceso o *thread* hasta que llegue una interrupción, y proteger secciones de código de ser interrumpidas. Las interfases definidas no conseguirán una portabilidad completa de los programas de aplicación debido a las muchas diferencias existentes en los mecanismos de manejo de interrupciones de las diferentes arquitecturas. Sin embargo, la portabilidad de la aplicación se incrementará con el uso de esta interfase, ya que se establece un modelo de referencia, y además el código no portable se confina a módulos específicos claramente delimitados.

4.5 Control de Dispositivos de Entrada/Salida

En sistemas de tiempo real es frecuente interaccionar con el entorno a través de dispositivos especiales como entradas/salidas analógicas o digitales, contadores, etc. Típicamente, es el usuario responsable de la aplicación el que escribe los *drivers* de entrada/salida—o porciones de código que acceden directamente al dispositivo hardware—para estos dispositivos especiales. Una forma estandarizada de acceder a los *drivers* de entrada/salida para realizar operaciones de control sobre el dispositivo asociado permitiría que estas operaciones estuviesen claramente definidas. El POSIX.4b define una función que permite a un programa de aplicación transferir información de control hacia y desde el *driver* del dispositivo. Del mismo modo que para las funciones de control de interrupciones, los programas que utilicen la función de control de dispositivos pueden no ser completamente portables, pero su portabilidad se mejora por el uso de esta interfase que proporciona un modelo de referencia para acceder a los *drivers* de dispositivos.

4.6 Creación de Procesos

Otra interesante función que se define en POSIX.4b es la creación eficiente de procesos, sin necesidad de utilizar la secuencia de funciones `fork`() y `exec`() típica del UNIX, en la que primero se hace una copia del proceso original, para después destruir la copia y sustituirla por una nueva imagen de proceso. Aunque la secuencia mencionada presenta algunas ventajas—en especial en lo relativo a las operaciones de herencia de descriptores de fichero entre el proceso padre y el hijo—en un porcentaje muy alto de veces sería suficiente una primitiva mucho más sencilla y eficiente que simplemente crease un nuevo proceso utilizando la imagen de proceso almacenada en un determinado fichero. Esto es lo que hace la nueva función definida en el POSIX.4b, denominada `spawn`(), y que permitirá reducir el tiempo de creación de un alto porcentaje de los procesos.

5 INTERFASES PARA LENGUAJE ADA

El POSIX y el lenguaje Ada son dos estándares complementarios de aplicación a los sistemas de tiempo real. Aunque es posible implementar las funciones del ejecutivo Ada sin necesidad de contar con un sistema operativo, la utilización de éste permite reducir la complejidad del ejecutivo y, sobre todo, permite al programa Ada relacionarse con otros programas—escritos en Ada u otros lenguajes de programación—que corren en el mismo computador. Las interfases POSIX, al estar estandarizadas, permiten acceder a los servicios del sistema operativo de forma portable.

El estándar POSIX.5 [15]—ya aprobado—define las interfases básicas del sistema para lenguaje Ada. En este estándar se especifica que un programa Ada se comporta como un proceso POSIX. Aunque los servicios del sistema operativo son los mismos que los especificados para lenguaje C en el POSIX.1, se han tenido en cuenta las facilidades especiales del lenguaje Ada al desarrollar las interfases. Así, por ejemplo, el tratamiento de errores se realiza elevando una excepción Ada, en lugar de retornar un valor de error como en las funciones C. El manejo de las señales del POSIX se hace utilizando mecanismos Ada: aquellas señales que representan condiciones de error se tratan como excepciones, mientras que las señales que representan eventos se tratan como llamadas *entry* de nivel de interrupción. Algunas de las funciones C, como las de manejo del tiempo—`sleep`(), `time`(), etc.—se omiten, pues están ya soportadas por el lenguaje.

Las interfases Ada para las extensiones de tiempo real están siendo estandarizadas en este momento bajo el estándar POSIX.20 [17]. Salvo las diferencias de sintaxis, existen muy pocas diferencias entre las interfases Ada y C de tiempo real. Unicamente mencionar que el uso de memoria compartida debe realizarse con cuidado, ya que en Ada83 se permite al compilador hacer optimizaciones que eviten algunas lecturas o escrituras en memoria; en el caso de la memoria compartida, estas optimizaciones resultarían incorrectas. Este problema se puede solucionar mediante el uso de pragmas dependientes de la implementación. En Ada 9X—el nuevo estándar en desarrollo para el lenguaje Ada—existen dos pragmas (`volatile` y `atomic`) que permiten evitar este tipo de optimizaciones para los objetos que se especifiquen.

Las interfases Ada más prometedoras por su funcionalidad añadida, son las interfases para la extensión de *threads*. Aunque se trata sólo de borradores preliminares [18], la propuesta se centra básicamente en la equiparación entre los *threads* y las tareas Ada. El ejecutivo Ada se implementaría sobre el sistema operativo de modo que cada tarea Ada

fuese un *thread*, y permitiendo al sistema operativo realizar la planificación de las tareas. Las principales ventajas que se consiguen con esta estrategia son:

- Facilitar la interoperabilidad entre tareas Ada y *threads* escritos en C o en otros lenguajes; todos ellos pueden convivir en el mismo proceso e incluso sincronizarse entre sí e interaccionar a través de los mecanismos POSIX.

- Tratamiento uniforme de todas las tareas o *threads* del sistema. Si el sistema dispone de varios procesos, por ejemplo varios programas Ada y/o programas escritos en lenguaje C, todos los *threads* y tareas del sistema se pueden planificar a nivel global. Por ejemplo, una tarea de un programa puede expulsar a otra tarea de otro programa que tenga menor prioridad; esto no es posible si la planificación de tareas no es realizada por el sistema operativo de forma global.

- No se bloquea el programa completo al invocar servicios del sistema operativo, sino sólo el *thread* o tarea que ha realizado la llamada.

- El ejecutivo es portable, pues está implementado sobre POSIX, y es también más sencillo al dejar la implementación de parte de sus funciones al sistema operativo.

La mayoría de estas ventajas no están disponibles en los compiladores Ada actuales, y suponen para los sistemas de tiempo real—y en especial los más grandes—un importante salto cualitativo. Como contrapartida a estas ventajas, es preciso indicar que un ejecutivo desarrollado sobre la extensión de *threads* será memos eficiente que un ejecutivo desarrollado específicamente para soportar los servicios del lenguaje Ada, debido a que los servicios del sistema operativo no son exactamente iguales a los del ejecutivo, y a que el sistema operativo tiene muchos otros requisitos adicionales que cumplir.

6 PERFILES DE ENTORNOS DE APLICACIONES DE TIEMPO REAL

El estándar POSIX.1, junto con las extensiones de tiempo real y la extensión de *threads*, constituyen un poderoso conjunto de interfases que permiten implementar sistemas operativos capaces de dar respuesta a las necesidades de sistemas grandes con requerimientos de tiempo real. Sin embargo, para sistemas de tiempo real empotrados y pequeños, sería deseable un subconjunto de estas interfases. Por ejemplo, muchos sistemas empotrados tienen un hardware necesariamente limitado, que hace muy difícil implementar servicios tales como el sistema de ficheros o espacios de direccionamiento independientes para los procesos. Los perfiles de entornos de aplicaciones de tiempo real (AEP según sus siglas en inglés) definidos en el estándar POSIX.13 proporcionan los subconjuntos de los servicios definidos en los estándares base que se consideran adecuados para un entorno de aplicación particular. En el estándar POSIX.13 se han definido cuatro AEPs de tiempo real:

1) *Sistema Mínimo*: Corresponde a un sistema empotrado pequeño sin necesidad de unidad de manejo de memoria (MMU), sin sistema de ficheros (sin disco), y si terminal de entrada/salida. Sólo se permite un proceso, aunque puede haber múltiples *threads* ejecutándose de forma concurrente.

2) *Controlador de Tiempo Real*: Corresponde a un sistema controlador de propósito especial. Es como el perfil mínimo, pero añadiendo un sistema de ficheros y un terminal de entrada/salida. Sólo se permite un proceso, aunque se permiten múltiples *threads*.

3) *Sistema Dedicado*: Corresponde a un sistema empotrado grande, sin sistema de ficheros. Puede tener múltiples procesos y múltiples *threads*.

4) *Sistema Multi-Propósito*: Corresponde a un sistema grande de tiempo real con todos los servicios soportados.

La Tabla V resume las principales características de cada uno de los perfiles de tiempo real.

Tabla V. Características de los Perfiles de Tiempo Real

Perfil	**Sistema de Ficheros**	**Múltiples Procesos**	**Múltiples *Threads***
Sistema Mínimo de Tiempo Real	NO	NO	SI
Controlador de Tiempo Real	SI	NO	SI
Sistema Dedicado de Tiempo Real	NO	SI	SI
Sistema de Tiempo Real Multi-Propósito	SI	SI	Opcional

Con los perfiles de tiempo real definidos, es posible implementar sistemas operativos conformes al estándar POSIX que sean aptos para una gran variedad de plataformas de tiempo real de diferentes tamaños, y con diferentes requerimientos. Las aplicaciones podrán ser portables de una plataforma a otra, siempre que sean conformes al mismo perfil, o que la nueva plataforma incluya todos los servicios de la anterior. Por ejemplo, una aplicación podrá ser portada de un sistema mínimo a un sistema con perfil de controlador de tiempo real, o a una plataforma con el perfil de sistema dedicado. Más aún, la misma aplicación que se ejecuta en un sistema empotrado muy pequeño podrá correr sobre un sistema de desarrollo dotado de todos los servicios, para poder ser depurada. Con el amplio espectro de posibilidades definidas por los perfiles actuales, los núcleos de tiempo real que se comercializan actualmente tendrán la posibilidad de proporcionar una interfase POSIX. Se prevé que la mayor parte de los núcleos y sistemas operativos de tiempo real que se distribuyan comercialmente en los próximos años serán conformes a uno de los perfiles POSIX de tiempo real; esto traerá la portabilidad de las aplicaciones al mundo del tiempo real.

7 CONCLUSIONES

El POSIX es un estándar de sistema operativo en evolución, que se prevé que será ampliamente utilizado en los próximos años. Una importante parte de este estándar está pensada para proporcionar la portabilidad de las aplicaciones con requerimientos de tiempo real. Junto a las interfases de servicios del sistema, se estandarizan también perfiles de entornos de aplicaciones que permitirán a los implementadores desarrollar

sistemas operativos POSIX de tiempo real para una gran variedad de plataformas, desde los sistemas empotrados pequeños hasta los sistemas de tiempo real grandes. El estándar define interfases en diferentes lenguajes de programación. En particular, las interfases de tiempo real están siendo definidas para C y Ada, que son los lenguajes estándar de programación más importantes para los sistemas prácticos de tiempo real.

Para la selección de un sistema operativo POSIX de tiempo real es preciso ser muy cuidadoso, ya que existe un gran número de funciones y servicios optativos. Además de comprobar que el sistema operativo seleccionado corresponde al perfil (AEP) con las características necesarias, es preciso comprobar aspectos como el tipo de planificador de *threads*—global, local, o mixto—, la precisión del reloj hardware utilizado y, en general, la magnitud de los tiempos de respuesta de peor caso de los servicios que sean más importantes para nuestra aplicación.

La funcionalidad especificada en el estándar POSIX es similar a la que se encuentra en la mayoría de los núcleos y sistemas operativos de tiempo real disponibles comercialmente. Las interfases POSIX se han definido de acuerdo con resultados recientes de la teoría de planificación con prioridades estáticas. Algunas implementaciones basadas en borradores iniciales de los estándares POSIX.4 y POSIX.4a ya han sido desarrolladas [1], y muestran resultados muy prometedores. En resumen, el estándar POSIX permitirá construir sistemas predecibles y analizables que cumplen sus requerimientos de tiempo real, y que pueden ser fácilmente portables de unas plataformas a otras.

8 REFERENCIAS

[1] B.O. Gallmeister, y C. Lanier. "Early Experience with POSIX 1003.4 and POSIX 1003.4a". *Proceedings of the IEEE Real-Time Systems Symposium*, Diciembre 1991, pp. 190-198.

[2] M. González Harbour. "Real-Time POSIX: An Overview". *Proceedings of the VVCONEX-93 International Conference*, Moscow, Junio 1993.

[3] M. Klein, T. Ralya, B. Pollak, R. Obenza, y M. González Harbour. "A Practitioner's Handbook for Real-Time Analysis". *Kluwer Academic Press*, 1993.

[4] B.W. Lampson, y D.D. Redell. "Experience with Processes and Monitors in Mesa". *Communications of the ACM* 23, 2, Febrero 1980, pp. 105-107.

[5] J. Leung, y J.W. Layland. "On Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks". *Performance Evaluation 2*, 237-50, 1982.

[6] C.L. Liu y J.W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment". *Journal of the ACM*, Vol. 20, No. 1, 1973, pp. 46-61.

[7] R. Rajkumar, L. Sha, y J.P. Lehoczky. "Real-Time Synchronization Protocols for Multiprocessors". *IEEE Real-Time Systems Symposium*, Diciembre 1988.

[8] L. Sha, R. Rajkumar, y J.P. Lehoczky. "Priority Inheritance Protocols: An approach to Real-Time Synchronization". *IEEE Trans. on Computers*, Septiembre 1990.

[9] L. Sha, y J.B. Goodenough. "Real-Time Scheduling Theory and Ada". *IEEE Computer*, Vol. 23, No. 4, Abril 1990.

[10] B. Sprunt, L. Sha, y J.P. Lehoczky. "Aperiodic Task Scheduling for Hard Real-Time Systems". *The Journal of Real-Time Systems*, Vol. 1, 1989, pp. 27-60.

[11] ISO/IEC Standard 9945-1:1990, and IEEE Standard 1003.1-1990, *"Information Technology —Portable Operating System Interface (POSIX)— Part 1: System Application Program Interface (API) [C Language]*". Institute of Electrical and Electronic Engineers, 1990.

[12] IEEE Standards Project P1003.4, "*Draft Standard for Information Technology —Portable Operating System Interface (POSIX)— Part 1: System Application Program Interface (API) — Amendment 1: Realtime Extension [C Language]*". Draft 14. The Institute of Electrical and Electronics Engineers, Marzo-1993.

[13] IEEE Standards Project P1003.4a, "*Threads Extension for Portable Operating Systems*". Draft 7. The Institute of Electrical and Electronics Engineers, Abril 1993.

[14] IEEE Standards Project P1003.4b, "*Draft Standard for Information Technology —Portable Operating System Interface (POSIX)— Part 1: Realtime System API Extension*". Draft 7. The Institute of Electrical and Electronics Engineers, Agosto 1993.

[15] IEEE Standard 1003.5-1992, "*Information Technology —POSIX Ada Languaje Interfaces—Part 1: Binding for System Application Program Interface (API)*". Institute of Electrical and Electronic Engineers, 1992.

[16] IEEE Standards Project P1003.13, "*Draft Standard for Information Technology —Standardized Application Environment Profile— POSIX Realtime Application Support (AEP)*". Draft 5. The Institute of Electrical and Electronics Engineers, Febrero 1992.

[17] IEEE Standards Project P1003.20, "*Information Technology —POSIX Ada Languaje Interfaces—Part 2: Binding for Realtime Extensions*". Draft 2. Institute of Electrical and Electronic Engineers, Abril 1993.

[18] Ted Giering y Ted Baker, "*Threads Extension for Portable Operating Systems: Thin Ada Binding*", IEEE-P1003.5 Working Group paper, Draft 0.0, Noviembre 1992.