Novática, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de ATI (Asociación de Técnicos de Informática). _Novática_ edita también _Upgrade_, revista digital de CEPIS (_Council of European Professional Informatics Societies_), en lengua inglesa.

<http://www.ati.es/novatica/>
<http://www.upgrade-cepis.org/>

ATI es miembro de CEPIS (*Council of European Professional Informatics Societies*) y tiene un acuerdo de colaboración con ACM (*Association for Computing Machinery*). Tiene asimismo acuerdos de vinculación o colaboración con AdaSpain, AI2 y ASTIC

Coordinación Editorial y Redacción Central (ATI Madrid)
Padilla 66, 3º, dcha., 28006 Madrid
Tlf.914029391; fax.913093685 <*novatica@ati.es*>
Composición, Edición y Redacción ATI Valencia
Reino de Valencia 23, 46005 Valencia
Tlf./fax 963330392 <*secreval@ati.es*>
Administración y Redacción ATI Cataluña
Via Laietana 41, 1º, 1ª, 08003 Barcelona
Tlf.934125235; fax 934127713 <*secregen@ati.es*>
Redacción ATI Andalucía
Isaac Newton, s/n, Ed. Sadiel, Isla Cartuja 41092 Sevilla
Tlf./fax 954460779 <*secreand@ati.es*>
Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza
Tlf./fax 976235181 <*secreara@ati.es*>
Redacción ATI Asturias-Cantabria <*gp-astucant@ati.es*>
Redacción ATI Castilla-La Mancha <*gp-clmancha@ati.es*>
Redacción ATI Galicia
Recinto Ferial s/n, 36540 Silleda (Pontevedra)
Tlf.986581413; fax 986580162 <*secregal@ati.es*>
Suscripción y Ventas: <*http://www.ati.es/novatica/interes.html*>, o en ATI Cataluña y ATI Madrid
Publicidad: Padilla 66, 3º, dcha., 28006 Madrid
Tlf.914029391; fax.913093685 <*novatica.publicidad@ati.es*>

Imprenta: 9·Impressió S.A., Juan de Austria 66, 08005 Barcelona.
Depósito Legal: B 15.154-1975
ISSN: 0211-2124; CODEN NOVAEC

Portada: Antonio Crespo Foix / © ATI 2003

novática
UPGRADE
Revista de la Asociación de Técnicos de Informática

MAYO · JUNIO 2003
163

SUMARIO

Monografía del próximo número:
«Ingeniería del Software: estado de un arte»

Sistemas de Tiempo Real

Sistemas Linux de tiempo real

Javier Miqueleiz Álamos
Grupo de Redes, Sistemas y Servicios Telemáticos, Departamento de Automática y Computación, Universidad Pública de Navarra

<javier.miqueleiz@unavarra.es>

Resumen: *Linux implementa el estándar POSIX para* soft real-time*, pero no proporciona un buen servicio a las aplicaciones de tiempo real. Muchos desarrolladores han modificado el kernel para mejorar esta situación. En este artículo se describirán las extensiones de Linux para tiempo real que existen a día de hoy.*

Palabras clave: *extensiones, kernel,Linux, tiempo real.*

1. Introducción

Los sistemas operativos de tiempo real existen comercialmente desde hace años. Se caracterizan por proporcionar buen rendimiento, un entorno completo de desarrollo y buena asistencia técnica. Sus aspectos negativos son un coste alto y su naturaleza *closed source.*

Las variantes de Linux para tiempo real ofrecen una alternativa *open source* a las soluciones comerciales. En este trabajo se van a describir sus objetivos, garantías ofrecidas y características.

2. RTLinux

RTLinux (*Real-Time Linux*) ofrece prestaciones de *hard real-time*, por lo que es adecuado para tareas con límites temporales estrictos.

RTLinux consiste en un microkernel sobre el que se ejecutan tareas de tiempo real. El kernel Linux está implementado como una tarea de prioridad mínima y no tiene control directo sobre el bloqueo de las IRQs. Cuando hay tareas de tiempo real que deben ser ejecutadas, el microkernel interrumpe la ejecución de Linux. El mecanismo para conseguir esto es una técnica de emulación software del controlador de interrupciones .

La filosofía de RTLinux es separar por completo la funcionalidad de tiempo real de la de propósito general, ya que sus principios de diseño son contrapuestos. En la la parte de propósito general se optimizan los casos comunes a costa de empeorar los casos peores. En la de tiempo real importa el determinismo y no la rapidez de respuesta. En opinión de los desarrolladores de RTLinux, un sistema que incorpore ambas funcionalidades no será eficiente en ninguna de ellas .

Las tareas de RTLinux no puede hacer un uso directo de los servicios del kernel, ya que no es posible que realicen llamadas al sistema. Lo que sí es posible es un uso indirecto si los hilos de tiempo real se comunican con procesos Linux. Para realizar esto, RTLinux incorpora dos mecanismos: la memoria compartida y las colas FIFO de tiempo real. Las garantías que se obtienen con este uso indirecto son las habituales de los procesos Linux.

3. RTAI

El enfoque de diseño de RTAI (*Real Time Application Interface*) es similar al de RTLinux: tratar a Linux como una tarea de prioridad baja que no tiene control directo sobre el programador de interrupciones. El mecanismo para conseguirlo es diferente en los detalles de imple-mentación. RTAI utiliza una capa HAL (*Hardware Abstraction Layer*) que simplifica el interfaz entre el microkernel de tiempo real y el kernel Linux. La función de esta capa es sustituir las llamadas a ciertas funciones del kernel por las llamadas propias de RTAI. La ventaja de esta técnica frente a la de RTLinux es una implementación más clara y sencilla.

Las diferencias más importantes entre RTAI y RTLinux radican en las funcionalidades que proporcionan. La filosofía de los desarrolladores de RTAI es implementar muchas funcionalidades con el fin de que los usuarios no echen en falta ninguna. Por el contrario, los diseñadores de RTLinux prefieren ofrecer pocos servicios y poner énfasis en la corrección y la rapidez.

RTAI proporciona garantías de *hard real-time*, por lo que resulta adecuado para el mismo tipo de aplicaciones que RTLinux. En sus primeras versiones presentaba los mismos problemas que aquél para la implementación de aplicaciones multimedia. Sin embargo, en las versiones más recientes se ha añadido el módulo LXRT (*Linux Real-Time*), que incluye funcionalidades muy interesantes:

- Proporciona una API simétrica, que se utiliza de forma idéntica desde los módulos del kernel o los procesos de usuario.
- Permite acceder a los servicios de RTAI en modo de usuario sin renunciar a la realización de llamadas al sistema y con rendimiento de *firm real-time*.
- Proporciona prestaciones de *hard real-time* desde procesos de usuario, evitando de esta forma los problemas de incompatibilidad de versión de los módulos del kernel.

A modo de conclusión, la orientación de RTAI es la misma que

la de RTLinux, pero sus desarrolladores han optado por añadirle más funcionalidades. RTAI es más flexible y puede dar garantías de *firm real-time* en espacio de usuario sin renunciar a la realización de llamadas al sistema.

4. KURT

Los desarrolladores de KURT (*Kansas University Real-Time*) han pretendido implementar en Linux características de tiempo real para que pudiera ser usado en algunos de sus proyectos de investigación. Su objetivo no ha sido diseñar una extensión para tiempo real de propósito general, sino que han buscado soluciones *ad hoc*.

La clase de servicio que proporciona KURT es de *firm real-time*. Para conseguir el aumento de la resolución temporal con respecto al kernel Linux, KURT funciona sobre el paquete UTIME. Este paquete reprograma dinámicamente la frecuencia con la que interrumpe el reloj HW y proporciona resolución de microsegundos. Puede funcionar en dos modos, periódico y no periódico. En el primero de ellos la frecuencia de interrupción es fija, resultando adecuado para aplicaciones periódicas. En el segundo de los modos se programa el reloj para que interrumpa en los momentos en que hay eventos de tiempo real que deben ser atendidos. Si no hay ningún evento a atender, el reloj interrumpe con la frecuencia habitual: 100 Hz. Este esquema dinámico permite ahorrar tiempo de CPU cuando la resolución alta no es necesaria. El modo de funcionamiento periódico tiene como ventaja que no hay costos de CPU asociados a la reprogramación de la frecuencia del reloj.

KURT consiste básicamente en un planificador específico para tiempo real. Puede funcionar en tres modos distintos: normal, *mixed* y *focused*. El primero de ellos es equivalente a utilizar el kernel Linux normal, con la diferencia de que se tiene la resolución temporal alta de UTIME. Los dos últimos son los modos específicos de tiempo real. El modo *mixed* permite que los procesos normales coexistan con los de tiempo real. En el modo *focused* sólo se planifican los procesos de tiempo real, por lo que es adecuado para aplicaciones con requisitos exigentes.

El planificador de KURT es del tipo *time-driven* y se puede utilizar de dos formas distintas. En la primera de ellas se debe proporcionar un fichero con información sobre los tiempos en los que debe empezar la ejecución de los procesos o módulos de tiempo real. Esto resulta adecuado cuando las tareas a ejecutar son conocidas a priori y es factible el cálculo *off-line* de los tiempos de inicio de ejecución. La segunda de las formas es adecuada para procesos periódicos. Cada uno de éstos le comunica al kernel sus características para que éste realice un control de admisión. Los parámetros que le pasa al kernel son el periodo y el tiempo de CPU que se necesita por periodo.

A nivel interno, KURT se divide en dos partes: el núcleo KURT y los módulos KURT. El núcleo se encarga de invocar a los módulos para que éstos ejecuten las tareas de tiempo real. Hay un módulo especial, llamado Process RTMod, que se encarga de pasar el control a los procesos de usuario. La **figura 1** describe la relación entre los distintos componentes de KURT.

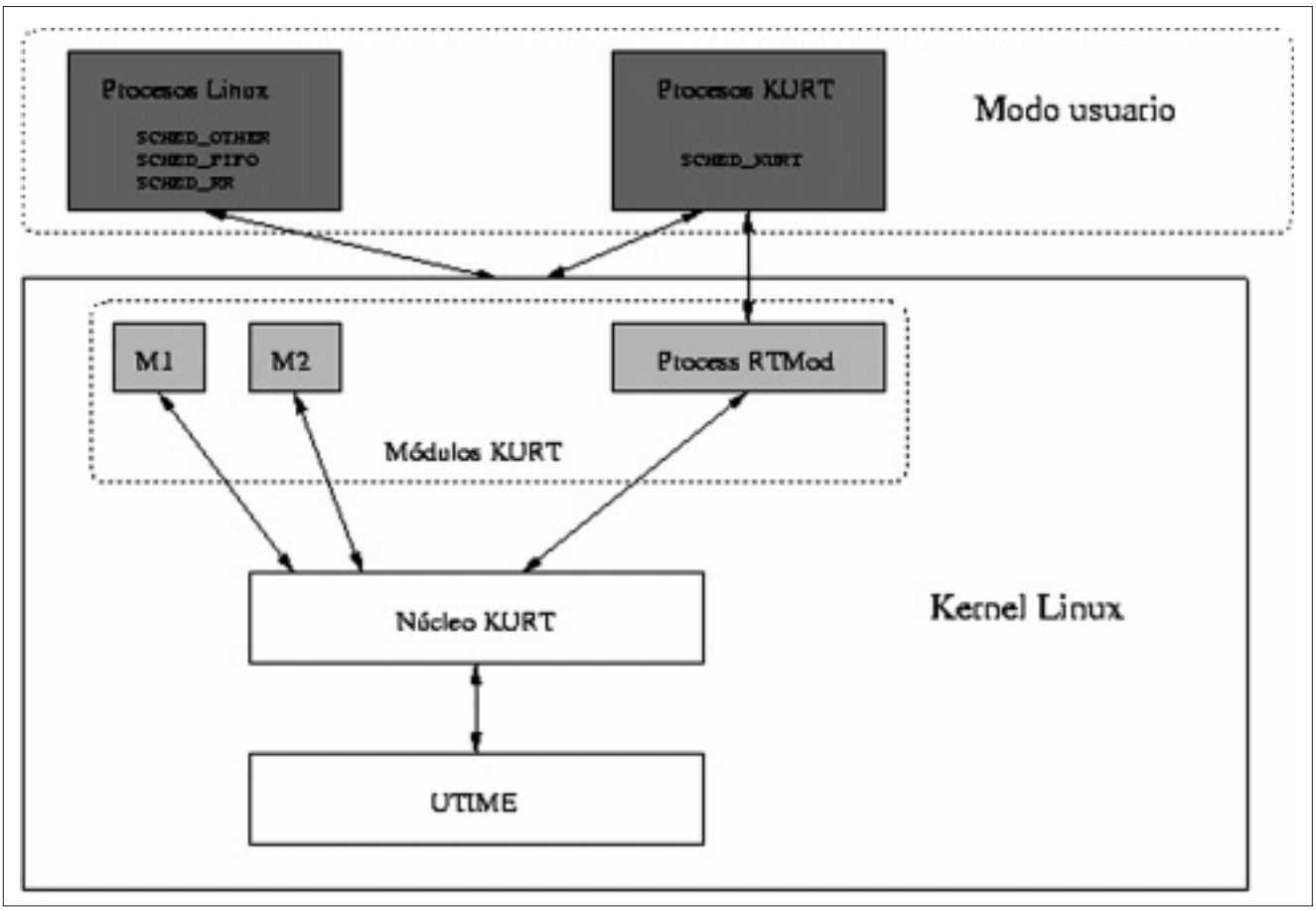

Figura 1. Arquitectura de KURT

A diferencia de RTLinux y de RTAI, en KURT la funcionalidad de tiempo real está integrada en el kernel, por lo que tanto los procesos como los módulos pueden utilizar todos sus servicios. Es preciso tener en cuenta que el rendimiento de KURT no es igual de bueno que el de las dos variantes de *hard real-time*. Las razón de esto es que KURT utiliza subsistemas del kernel Linux no diseñados para tiempo real. Estos subsistemas son el origen de distorsiones en los tiempos de planificación de las tareas.

5. RED-Linux

RED-Linux es el acrónimo de *Real-Time and Embedded Linux*. Consiste en un kernel Linux normal que ha sido modificado para mejorar sus propiedades con las aplicaciones de tiempo real. Las garantías que ofrece son de *firm real-time*. RED-Linux es compatible con las aplicaciones tradicionales y, al igual que en KURT, los procesos de tiempo real pueden solicitar cualquier servicio al kernel.

Sus autores han analizado las causas por las que Linux no es adecuado para tiempo real y han propuesto e implementado soluciones. Las funcionalidades implementadas han sido las siguientes:

· *Microtimer*: en un SO de propósito general como Linux, se utiliza un temporizador periódico para repartir la CPU entre los procesos. El periodo por defecto es de 10 ms, que es demasiado alto para un SOTR. En un sistema de este tipo resulta imprescindible tener un temporizador con resolución de microsegundos.

· Emulación software de las interrupciones: en Linux, cuando la CPU detecta una interrupción, pasa a atender la rutina de servicio a la interrupción. Este proceso requiere tiempo de CPU y puede afectar a las tareas de tiempo real. Los desarrolladores de RED-Linux han implementado un mecanismo de emulación software de interrupciones basado en el de RT-Linux. Su funcionamiento es el siguiente. Cuando se produce una IRQ, no se atiende la interrupción, sino que simplemente se registra el evento en una tabla. Posteriormente se comprueba si es necesario atender interrupciones y, en caso afirmativo, se ejecutan las rutinas de servicio. Este mecanismo permite reducir el tiempo de respuesta de las tareas de tiempo real.

· Puntos de interrupción: en el kernel Linux, la ejecución de una llamada al sistema es un proceso atómico que no puede ser interrumpido por ningún proceso. Esto se debe a que el kernel no es reentrante, por lo que no es posible detener la ejecución de una llamada al sistema para dar la CPU a un proceso de tiempo real, ya que éste puede invocar en su ejecución la misma llamada al sistema. Las consecuencia de esto es un incremento del tiempo de respuesta de las tareas de tiempo real, ya que éstas deben esperar a que termine la ejecución de las llamadas al sistema. Un posible solución a este problema sería convertir al kernel Linux en reentrante. Esto implicaría un tiempo de interrupción del kernel muy bajo, pero supondría un trabajo largo y difícil. Los desarrolladores de RED-Linux han optado por añadir al código de las llamadas al sistema puntos de interrupción. A grosso modo, su funcionamiento es el siguiente.

El código de las llamadas al sistema se divide en bloques más

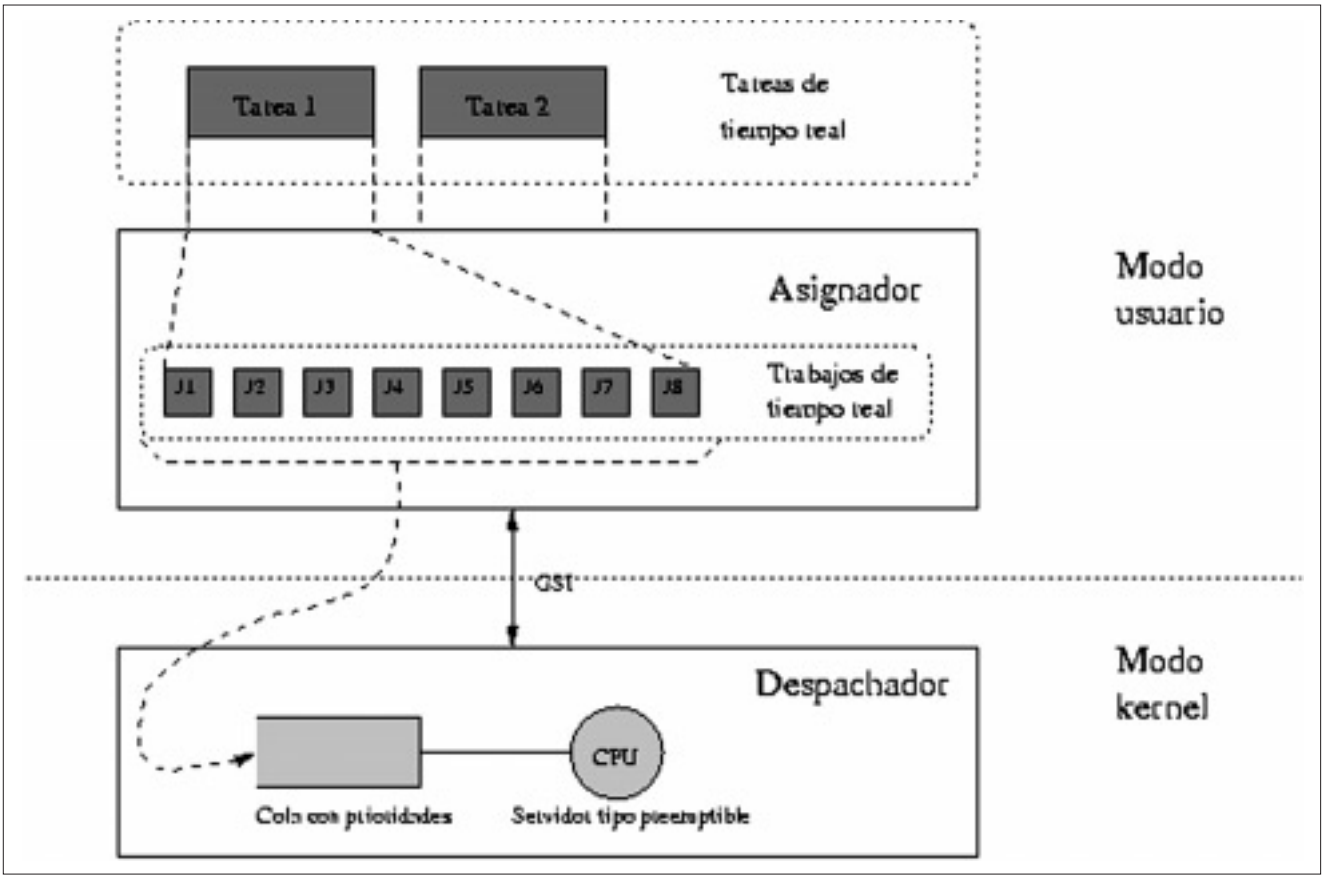

Figura 2. Planificador de RED-Linux

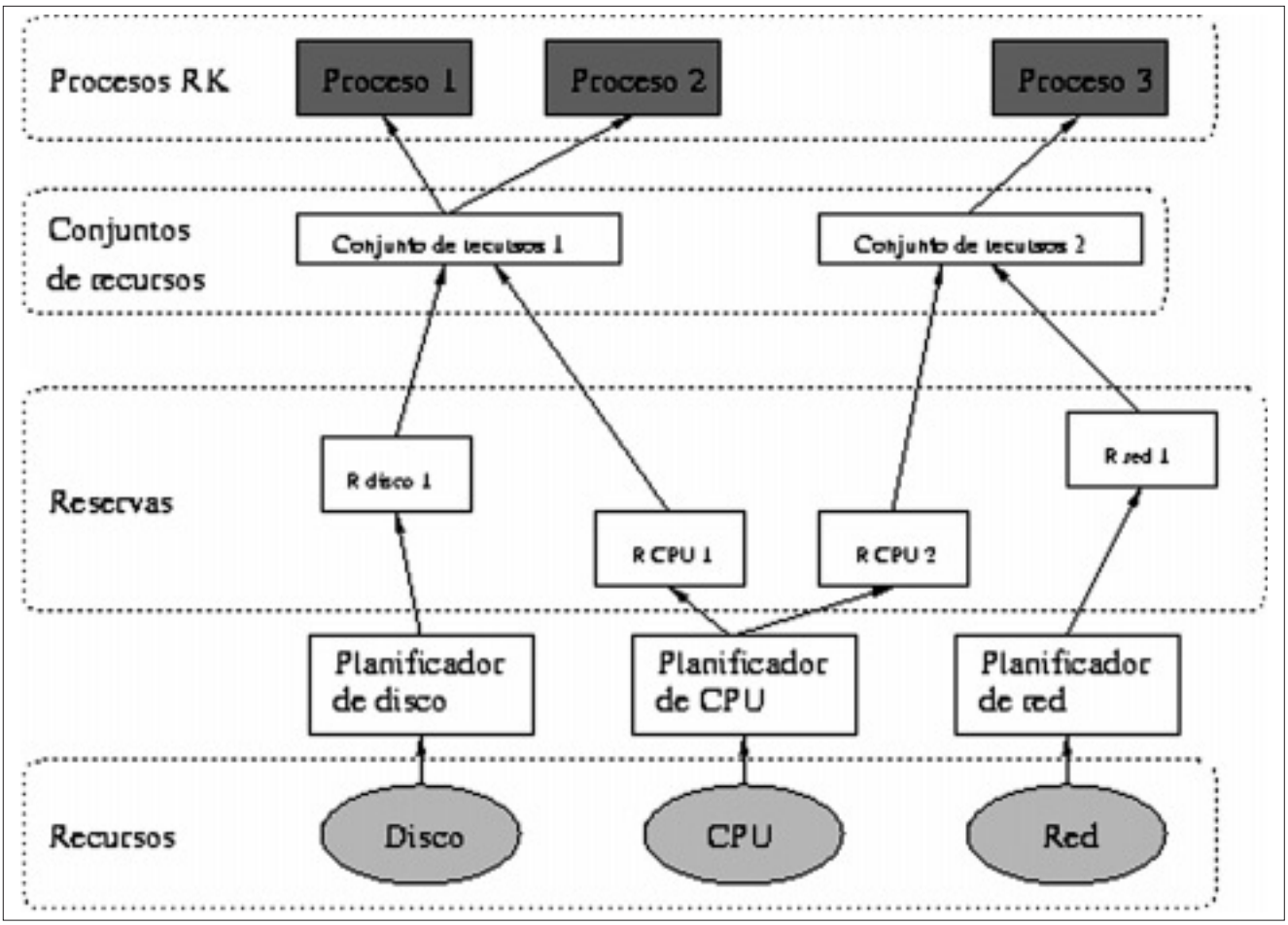

Figura 3. Arquitectura de Linux/RK

pequeños. Entre cada uno de esos bloques se inserta un punto de interrupción, es decir, un lugar en el que es seguro interrumpir la ejecución de la llamada y pasar a ejecutar un proceso más prioritario. En cada punto se comprueba si hay eventos de tiempo real que atender y si los hay, se invoca al planificador. Si no los hay, el *overhead* que introduce cada punto es muy pequeño, ya que sólo consisten en una comparación. Cuando se devuelve el control a la llamada que había sido interrumpida, ésta debe volver a comenzar en un punto seguro, pues no hay garantía de que los recursos compartidos del kernel no hayan sido modificados. La adición de estos puntos permite reducir el tiempo de interrupción del kernel. El valor máximo corresponde al tiempo que se tarda en ejecutar el bloque más largo.

La característica más novedosa de RED-Linux y que lo hace único es su planificador. En los SOTR habituales se ofrecen unos pocos algoritmos de planificación que están implementados en el kernel (o microkernel). La mayoría de esos algoritmos pertenecen al tipo *priority-driven*. Los desarrolladores de RED-Linux han pretendido que en él se puedan implementar la mayoría de los algoritmos de planificación, ya sean del tipo *time-driven, share-driven* o *priority-driven*.

En RED-Linux, el planificador está dividido en dos componentes, el despachador y el asignador (**figura 2**). El asignador divide las tareas en trabajos, asigna a cada trabajo cuatro parámetros de planificación y pasa el trabajo al despachador. El despachador guarda en una cola los trabajos y los planifica teniendo en cuenta los parámetros que le pasa el asignador. Los cuatro parámetros de planificación son los siguientes:

1. Prioridad: define la importancia de cada trabajo y tiene la misma semántica que en otros planificadores.
2. Tiempo de comienzo: es el instante en que un trabajo puede comenzar a ejecutarse.
3. Tiempo de finalización: es el plazo de vencimiento del trabajo. Si este tiempo se supera, su ejecución debe ser detenida.
4. Cantidad: es la cantidad de tiempo de CPU (u otros recursos) que necesita el trabajo.

Además de dar valores a estos cuatro parámetros, el asignador también elige el orden de evaluación de éstos. El despachador ordena los trabajos según el orden establecido por el asignador y planifica el primero de ellos. El asignador determina el algoritmo de planificación y puede ser modificado para adaptarse a las necesidades del usuario. El despachador, en cambio, es siempre el mismo y no necesita ser modificado.

En la versión actual de RED-Linux, el despachador está implementado mediante un módulo del kernel y el asignador mediante un proceso de usuario de prioridad alta.

De momento, el desarrollador de aplicaciones debe ocuparse de la programación del asignador. La intención de los desarrolladores de RED-Linux es ofrecer en un futuro una

librería de asignadores en la que estén incluidos los algoritmos de planificación más conocidos.

6. Linux/RK

Linux/RK es el acrónimo de *Linux/Resource Kernel*. El equipo desarrollador de este sistema pertenece al Laboratorio de Tiempo Real y Multimedia de la Universidad de Carnegie Mellon (EE.UU.). Linux/RK está diseñado para ser útil en aplicaciones multimedia y de tiempo real.

El primer concepto que es necesario definir para describir Linux/RK es el de kernel de recursos. Es un kernel que proporciona a las aplicaciones un acceso temporal, garantizado y controlado a los recursos del sistema. Los programas de aplicación deben solicitar al kernel reservas de recursos. Si las solicitudes son admitidas, el kernel garantiza que se recibirán de manera exclusiva las cantidades solicitadas del recurso.

Una de las virtudes de este tipo de kernels es que las aplicaciones se deben ocupar únicamente de especificar sus necesidades temporales. La gestión de los recursos es responsabilidad exclusiva del kernel. Los algoritmos que utiliza internamente el kernel pueden ser cambiados sin necesidad de modificar las aplicaciones.

RK portable es un kernel diseñado para añadir al kernel de un SO habitual las funcionalidades propias de un kernel de recursos. En realidad, no se trata de un kernel, sino de un subsistema que interacciona con el kernel normal. Su implementación no utiliza ninguna característica de un kernel concreto, por lo que es portable a cualquier plataforma. El único de sus componentes que debe ser adaptado es el encargado de implementar el interface con el kernel normal. Al conjunto formado por Linux y la implementación de RK portable en este SO se le denomina Linux/RK.

La API de Linux/RK está basada en el manejo de reservas y conjuntos de recursos. Una reserva es una cantidad de algún recurso del sistema que se destina para el uso exclusivo de una aplicación. Un conjunto de recursos agrupa varias reservas.

La **figura 3** describe gráficamente la relación entre aplicaciones, reservas, conjuntos de recursos y recursos. En el nivel inferior se encuentran los recursos del sistema. Cada uno de ellos es gestionado por un planificador, que se encarga de que a cada proceso se le dé la cantidad del recurso que ha solicitado. Cada proceso puede estar enlazado con un conjunto de recursos como máximo. Es posible enlazar varios procesos con un solo conjunto. Esto resulta interesante en aplicaciones que desempeñan una función mediante múltiples procesos.

En los recursos que son multiplexados temporalmente, la API

	Servicio	Paradigma	Llamadas al sistema	Microtimer	Tareas periódicas	Otras características
ART-Linux	Hard	Priority	No	Sí	Sí	
Fairsched	Soft	Share	Sí	No	No	Planificador HSFQ
Hard Hat Linux	Soft	Priority	Sí	No	No	Kernel interrumpible
KURT	Firm	Time	Sí	Sí	Sí	
Linux/RK	Firm	Priority	Sí	Sí	No	
Linux/RT	Firm	Priority	Sí	Sí	Sí	
Linux-SRT	Soft	Share	Sí	No	No	
QLinux	Soft	Share	Sí	No	No	Planificador HSFQ
RED-Linux	Firm	Prioriy, Share y Time	Sí	Sí	Sí	Planificador genérico, puntos de interrupción
RTAI	Hard y Firm	Priority	Sí (LXRT)	Sí	Sí	
RTLinux	Hard	Priority	No	Sí	Sí	

Tabla 1. Características de las variantes para tiempo real del kernel Linux

Figura 4. Parámetros de los recursos multiplexados en el tiempo

de Linux/RK permite especificar tres parámetros: T, C y D (**figura 4**) . C es la cantidad de recurso que se debe proporcionar en cada periodo de duración T. D es el tiempo límite (referido al comienzo del periodo) en el que los recursos deben haber sido proporcionados. La implementación actual de Linux/RK únicamente permite realizar reservas de CPU. Los planes para el futuro son añadir las reservas de BW de red y de BW de disco.

7. Conclusión

En este trabajo se han descrito algunas de las extensiones para tiempo real del kernel Linux existentes en la actualidad. Existen otras variantes que no han sido tratadas y que se van a citar a continuación: ART-Linux, Fairsched, Hard Hat Linux, Linux/RT, Linux-SRT , QLinux y el *patch* de baja latencia de Ingo Molnar.

Ninguna de las extensiones descritas o citadas es lo suficientemente flexible como para dar servicio a todas las aplicaciones. Sin embargo, constituyen un punto de partida muy valioso e interesante.

La **tabla 1** sintetiza las características de las diferentes variantes. La columna «Servicio» especifica las garantías que proporciona cada variante. «Paradigma» se refiere al paradigma de planificación utilizado. La posibilidad de realizar llamadas al sistema desde las tareas de tiempo real aparece indicada en la columna «Llamadas al sistema». El campo «Microtimer» especifica si la variante utiliza algún mecanismo para aumentar la resolución temporal. La columna denominada «Tareas periódicas» se refiere a si en la API se proporcionan funciones específicas para las tareas periódicas. Por último, en «Otras características» aparecen particularidades propias de cada variante.

Referencias

1. Pierre Cloutier, Paolo Mantegazza, Steve Papacharalambous, Ian Soanes, Stuart Hughes. «DIAMP-RTAI Position Paper, Nov. 2000». *Proceedings of the Real-Time Systems Symposium*, noviembre de 2000.

2. Robert Hill, Balaji Srinivasan, Shyam Pather, y Douglas Niehaus. «Temporal Resolution and Real-Time Extensions to Linux». Technical Report ITTC-FY98-TR-11510-03, Information and Telecommunication Technology Center, Department of Electrical Engineering and Computer Sciences. University of Kansas, junio de 1998. .

3. David Ingram. «Integrated Quality of Service Management». Dissertation Submitted for the Degree of Doctor of Philosophy. Jesus College, University of Cambridge, agosto de 2000.

4. Douglas Niehaus, William Dinkel, Sean B. House. «Effective Real-Time System Implementation with KURT Linux». *Real-Time Linux Workshop*, Vienna, Austria, diciembre de 1999.

5. Suichi Oikawa, y Ragunathan Rajkumar. «Portable RK: A Portable Resource Kernel for Guaranteed and Enforced Timing Behavior». *Proceedings of the Real-Time Technology and Applications Symposium*, junio de 1998.

6. Suichi Oikawa, y Ragunathan Rajkumar. «Linux/RK: A Portable Resource Kernel in Linux». *Proceedings of the Real-Time Systems Symposium*, 1998.

7. Balaji Srinivasan, Shyamalan Pather, Robert Hill, Furquan Ansari, Douglas Niehaus. «A Firm Real-Time System Implementation Using Commercial Off-The-Shelf Hardware and Free Software». *Proceedings of the Real-Time Technology and Applications Symposium*, junio de 1998.

8. Yu-Chung Wang, y Kwei-Jay Lin. «Enhancing the Real-Time Capability of the Linux Kernel». *Proceedings of the RTCSA*, octubre de 1998.

9. Yu-Chung Wang, y Kwei-Jay Lin. «Providing Real-Time Support in the Linux Kernel». *Proceedings of the Real-Time Technology and Applications Symposium*, 1999.

10. Yu-Chung Wang, y Kwei-Jay Lin. «Some Discussion on the Low Latency Patch for Linux». *Real-Time Linux Workshop*, noviembre de 2000.

11. Victor Yodaiken. «The RTLinux Manifesto». Department of Computer Science, New Mexico Intitute of Technology, Socorro, New Mexico.

12. Victor Yodaiken, Michael Barabanov. «RTLinux Version Two». VJY Associates LLC, 1999.