Dispositivo KURT, KU Tiempo Real

Alejandro Parra-Briones, Douglas Niehaus *
aparra@ittc.ukans.edu, niehaus@ittc.ukans.edu

Octubre de 2000

Esta ponencia forma parte del III Congreso Hispalinux[1] virtual, celebrado en Noviembre de 2000. Puedes participar en él en la sección de eventos de BarraPunto[2]

Resumen

KU Tiempo Real (KURT por sus siglas en inglés) es una extensión al núcleo de Linux. Hasta ahora ha usado llamadas a sistema, adicionales a las ya existentes en el núcleo, como interface para acceder a los servicios ofrecidos por KURT. Este proyecto desaparece dichas llamadas adicionales a sistema y crea un dispositivo orientado a caracteres el cual proveé los mismos servicios. Dicho dispositivo permite acceder a los servicios de KURT usando la llamada a sistema IOCTL. Esto proveerá una interface más limpia para futuros desarrollos usando KURT.

1 Introducción

KU Tiempo Real (KURT por sus siglas en inglés) es una extensión al núcleo de Linux la cual permite planificar tareas en tiempo real con una gran resolución de tiempo. Dicha resolución es del orden de microsegundos. Anteriormente KURT ofrecia una serie de llamadas a sistema, adicionales a las ya existentes en el núcleo, como interface para acceder a sus servicios. En contraste, ahora la interface es un dispositivo el cual permite acceder a estos servicios sin adicionar llamados a sistema al núcleo. Si varias extensiones al núcleo usan llamadas a sistema como interface entonces esto puede crear conflictos entre dichas extensiones.

Este proyecto cambia la interface al programador de varios llamados adicionales a sistema al uso de la llamada a sistema IOCTL. IOCTL nos permite emitir comandos a dispositivos especificos, en este caso al dispositivo KURT. A través de esos comandos podemos acceder a los servicios de KURT. El dispositivo KURT en nuestro caso fué creado como un dispositivo orientado a caracteres. Al ser KURT un dispositivo podemos montar o abilitar estos servicios como cualquier otro dispositivo. Al hacer uso de la llamada a sistema IOCTL para controlar al dispositivo KURT pudimos remover las llamadas a sistema adicionales. Lo que deja intacta la tabla de llamados a sistema, evitando conflictos entre KURT y otras extensiones en la tabla de llamados a sistema. Sin embargo, esto no remueve las otras modificaciones hechas al núcleo.

Este tipo de interface tiene varias bondadez entre las que están limpieza y claridad en futuros desarrollos [11]. Además se encontró un mejor comportamiento como se ve en la sección 5.

Este artículo esta organizado en seis partes. La sección 2 es una revisión de los conceptos básicos de Tiempo Real. La sección 3 es una revisión del trabajo relacionado a Tiempo Real en el ambiente Linux. La sección 4 se concentra en el diseño conceptual del dispositivo KU Tiempo Real. La sección 5 muestra algunos histogramas para comparar las prestaciones de la versión que usa el disposito contra la versión con llamadas de sistema adicionales. La sección 6 proveé de conclusiones y trabajo por hacer.

2 Conceptos Básicos de Tiempo Real

El concepto de Tiempo Real (RT por sus siglas en inglés) esta por todo el mundo. Todos hablan de juegos en RT, transacciones en RT, etc. Sin embargo, poca gente conoce la sutil diferencia entre rapidez de respuesta y

[1] `http://congreso.hispalinux.es`

[2] `http://barrapunto.com/eventos`

RT. Para aclarar esta diferencia necesitamos conocer tres conceptos básicos referentes a la descripción temporal del programa y el impacto de no cumplir con los requerimientos de tiempo de dicho programa.

Conceptos importantes para describir un programa de Tiempo Real

La mayoria de los Sistemas Operativos (SO) son diseñados para tener buen rendimiento de procesamiento y buen tiempo de respuesta. La gran mayoría tienen un planificador de trabajos *"equitativo"* para selecionar el siguiente proceso a ejecutarse. Sin embargo, un SO de RT debe ser capaz de planificar procesos para cumplir los plazos requeridos por las aplicaciones. Esto implica un planificador de trabajos el cual puede que no sea equitativo pero si *correcto* para seleccionar el próximo proceso a correr.

Antes de poder discutir la diferencia entre rapidez de respuesta y RT, es importante tener claros tres conceptos. Cada uno de estos conceptos nos permitirán describir claramente un programa o proceso de RT. Estos conceptos son Plazo, Resolución de tiempo y Periodicidad de acontecimientos.

- Plazo
  Plazo o Dead-line se refiere al hecho de terminar una tarea, trabajo o ejecución a un tiempo pre-determinado o pre-acordado. Por ejemplo, la construcción de una casa en tres meses, el pago de nuestros impuestos antes de la fecha límite, un sistema de los multimedia en el cual una imágen se debe poner al día cada trigésimo de un segundo, imprimir una página cada 10 segundos, etc. En cada uno de los ejemplos anteriores existe un *plazo* que debemos cumplir. En el caso del sistema multimedia, el plazo es un trigésimo de segundo, en el ejemplo de la casa el plazo es tres meses, etc.

- Resolución de tiempo
  Resolución de tiempo se refiere a la unidad mínima de tiempo que el sistema es capaz de registrar para medir un plazo. En sistemas operativos como Linux esta resolución es de 10 milisegundos. Es decir que podemos fijar un plazo en términos de centésimas de segundo. En cambio en un relog de pulsera comunmente la máxima resolución de tiempo es de un segundo. Haciendo una analogía con el sistema postal, la mejor resolución que tiene es día laboral o en algunos casos horas.

- Acontecimientos periódicos
  Acontecimientos periódicos son los que suceden cada período predefinido de tiempo. Por ejemplo, el día dura 24 horas y tendremos un nuevo día cada 24 horas. El año consta de 365 dias, etc.

  Un acontecimiento aperiódico es el que sucede eventualmente o de vez en cuando sin un espacio de tiempo *igual* entre acontecimientos. Un ejemplo puede ser la llegada de personas a un central de autobuses, ya que no llegan igualmente espaciadas en tiempo, etc

Clasificación de aplicaciones

Las aplicaciones se pueden clasificar por el impacto que tiene el fallar un plazo y su resolución de tiempo. El impacto puede ser muy severo o muy tolerable. Es posible que el fallar un plazo no sea tan severo si el margen es muy pequeño o por el contrario que sea catastrófico.

Para hacer más claros estos escenarios existen varias clasificaciones. Basados en la literatura existente se han propuesto tres clasificaciones. Estos son RT no estricto (Soft RT), RT estricto (Hard RT) y RT firme (Firm RT). Cada una de estas se discute a continuación.

Aplicación de RT no estricto Cuando el no cumplir un plazo no trae consecuencias graves, este plazo es considerado un *plazo no estricto* [1]. Ahora si una aplicación tiene plazos no estrictos y la resolución de tiempo no es muy fina, a esta aplicación se le llama *Aplicación RT no estricta* [3]. Un buen ejemplo de una aplicación de RT suave es un editor de texto. No importa que tan rápido tecleemos mientras podamos ver los caracteres que acabamos te teclear. Además podemos permitir un retardo de algunos milisegundos.

Aplicación de RT estricto En algunos casos no cumplir un plazo puede tener consecuencias catastróficas, a este tipo de plazo se le llama *plazo estricto* [1]. Si una aplicación tiene plazos estrictos y una resolución de tiempo muy fina, a esta le llamaremos *Aplicación de RT estricta* [3]. Un buen ejemplo podría ser un robot que maneja autos. Si el robot detecta una escenario de peligro en el cual debe frenar lo más rápido posible y postpone esta tarea para bajar un cristal, las consecuencias pueden ser catastróficas.

Aplicación de RT firme. Existen aplicaciones que permiten fallar algunos plazos pero que requieren de una resolución de tiempo muy fina. A este tipo la denominaremos *Aplicación de RT firme* [3]. Por ejemplo, un sistema de videoconferencia el cual permite que no actualicemos una imágen por cada 1000. Esta aplicación a pesar de tener un *plazo suave* require una resolución de tiempo muy fina.

Rapidez de respuesta vs RT Ahora estamos en condiciones de enterder la diferencia entre rapidez de respuesta y RT. Rapidez de respuesta es simplemente eso, que tan rápido va responder el sistema. Esto muchas veces se puede lograr comprando equipos con capacidad sobrada. En cambio RT significa obtener esa rapidez de respuesta basado en la administración de los recursos del sistema.

El impacto es subjetivo Muchas aplicaciones tienen plazos. Sin embargo, algunos sistemas no pueden cumplir esos plazos produciendo diferentes impactos en la aplicación. Es importante remarcar que el impacto de la falla en el cumplimiento de los plazos es algo *subjetivo*. Esta subjetividad hace difícil tener una frontera clara entre la clasificación de las aplicaciones de RT. En consecuencia algunos usuarios pueden considerar una aplicación RT estricta mientras que otros la consideren RT firme.

3 Trabajo relacionado

Varios esfuerzos se han hecho para tener sistemas operativos de RT. Esas soluciones se pueden clasificar como núcleos pequeños pero rápidos, extensiones a SO comerciales y SO de investigación. En este artículo nos enfocaremos a las extensiones a Linux.No entraremos en los detalles de cada solución.

Hay algunos SO de RT disponibles comercialmente, pero en su gran mayoría son costosos. En cambio, existen soluciones gratuitas basadas en LINUX. Algunas de ellas son:

- KURT
- RTLinux
- REDLinux
- LINUX/RK.

Cada una de estas soluciones resuelve el problema de cumplimiento de plazos y resolución de tiempo de forma diferente.

KURT

KURT básicamente hace dos modificaciones al núcleo de Linux. Cambia la resolución temporal y usa un planificador de trabajos fuera de línea [3]. Para mejorar la resolución de tiempo y evitar el sobre trabajo en mantener el relog KURT usa el contador del grupo fecha/hora (Time Stamp Counter) [6]. Para tener mayores información en referencia al TSC se invita al lector revisar la referencia número [6].

La segunda modifcación es en planificador de trabajos. Este requiere que la aplicación fije los plazos, esto puede ser considerado un ejemplo del paradigma de programar procesos a través de una tabla [5].

RT-LINUX

RT-Linux provee de una solución distinta. RT-Linux separa los mecanismos usados en RT de los de propósito general en el núcleo [3]. Con el fin de coordinar las multiples llamadas a interrupción Este tiene un proceso el cual atiende las interrupciones. Al igual que KURT modifica el planificador de trabajos y emplea dos modos de planificación de tareas: paso a paso o periodica [4]. La idea usada en RT-Linux se asemeja a la idea de núcleo pequeño propuesta por Ramamritham [5].

[3] `http://www.rtlinux.org/rtlinux/index.html`
[4] `http://www.rtlinux.org/rtlinux/index.html`

RED-Linux.

RED-Linux usa la resolución de tiempo y el mecanismo de emulación de atención a interrupciones hechos por TR-Linux [9] pero usa una politica diferente de programamción de tareas. Su programador de tareas se divide en dos partes: reservador y repartidor. El reservador es responsable de preparar los recursos de un nuevo proceso o tarea. Este reservador produce una *prioridad efectiva* para cada proceso. Basado en esta prioridad el despachador decidirá el order de ejecución [9]. La *prioridad efectiva* es una refinación de tres paradigmas de programación: Por prioridad, por tiempo y por proporción.

Linux/RK

Linux/RK reserva recursos como procesador, accesos a disco, etc. Este es similar a una agencia de viajes la cual hace reservaciones para el viajero. Este aprovecha la resolución temporal y emplea una programación dinámica.

Es importante recordar que no existe la solución perfecta. Algunas soluciones se ajustan mejor para un tipo de aplicación que otras. En las investigaciones hechas en nuestra institución KURT ha sido la mejor solución a nuestras necesidades. Además se ha ido adecuando a nuestros requerimientos.

4 Diseño Conceptual del dispositivo KURT

Dentro de los límites de mi conocimiento no he encontrado una extensión de RT al núcleo de Linux que no modifique la tabla de llamados a sistema. Es nuestra creencia que el no modificar el núcleo es la base para construir una serie de dispositos que permitan tener servicios de muy diversa naturaleza sin conflictos y con interfaces bien definidas [11]. Además esto permitirá a los desarrolladores de extensiones moverse del lento ciclo programar-compilar-arrancar máquina-apagar a el ciclo montar-desmontar-programar-compilar. Esto permitirá en el futuro a los usuarios tener diversos servicios sin el temor de modificar el código fuente del núcleo.

KURT se puede conceptualizar como la adición de la resolución de tiempo en microsegundos y cambios en el programador de tareas. Los servicios provistos por dicha extensión antes se accesaban a través de llamadas a sistema; dichas llamadas eran adicionales a las originalmente incluidas en el núcleo. La interface al programador desaparecio las llamadas adicionales a sistema y en cambio sólo usa la llamada a sistema IOCTL. IOCTL nos permite enviar comandos específicos al dispositivo KURT. Las rutinas que antes ejecutaban al accesar las llamadas adicionales a sistema ahora se ejecutan al enviar comandos específicos al dispositivo denominado KURT. La figura 1 muestra este concepto.

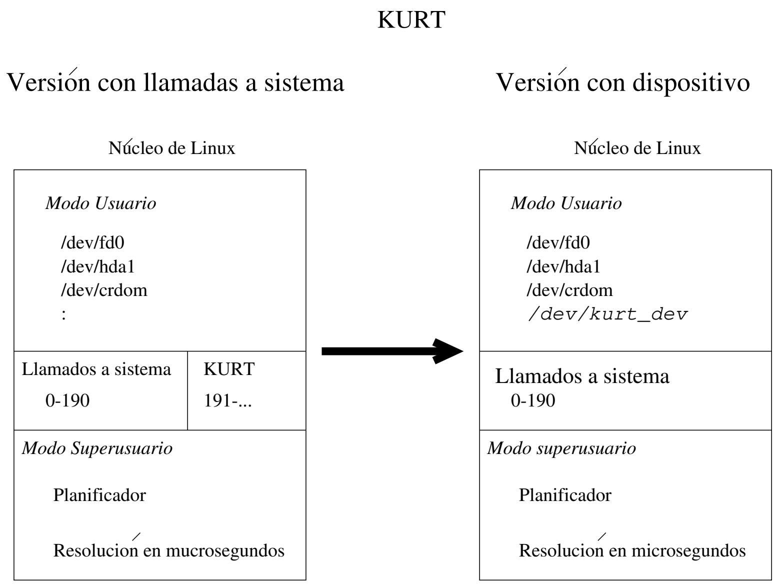

Figura 1: Concepto del dispositivo KURT

Dispositivo KURT

Ante todo mi más profundo agradecimiento a William Dinkel [5] sin cuya ayuda no habría sido posible terminar este proyecto.

El dispositivo KURT puede dividirse en dos partes: Interface al usuario y el dispositivo en sí. La interface al usuario ofrece acceso al dispositivo a través de varias funciones. Esas funciones sustituyen a las antiguas llamadas a sistema. Solo se aumento una funcion llamada *kurt_open* la cual regresa un descriptor de archivo el cual apunta al dispositivo.

El dispositivo KURT es un dispositivo el cual se accesa como un archivo. El dispositivo KURT se diseño como dispositivo orientado a caracteres. Este dispositivo también necesita registrarse en el sistema. A este dispositivo le puede cambiar, fijar o leer parametros usando señales de control (IOCTL). Todas las antiguas llamadas a sistema se accesan usando la llamada a sistema IOCTL. Esto se puede ver en la Figura 2. .

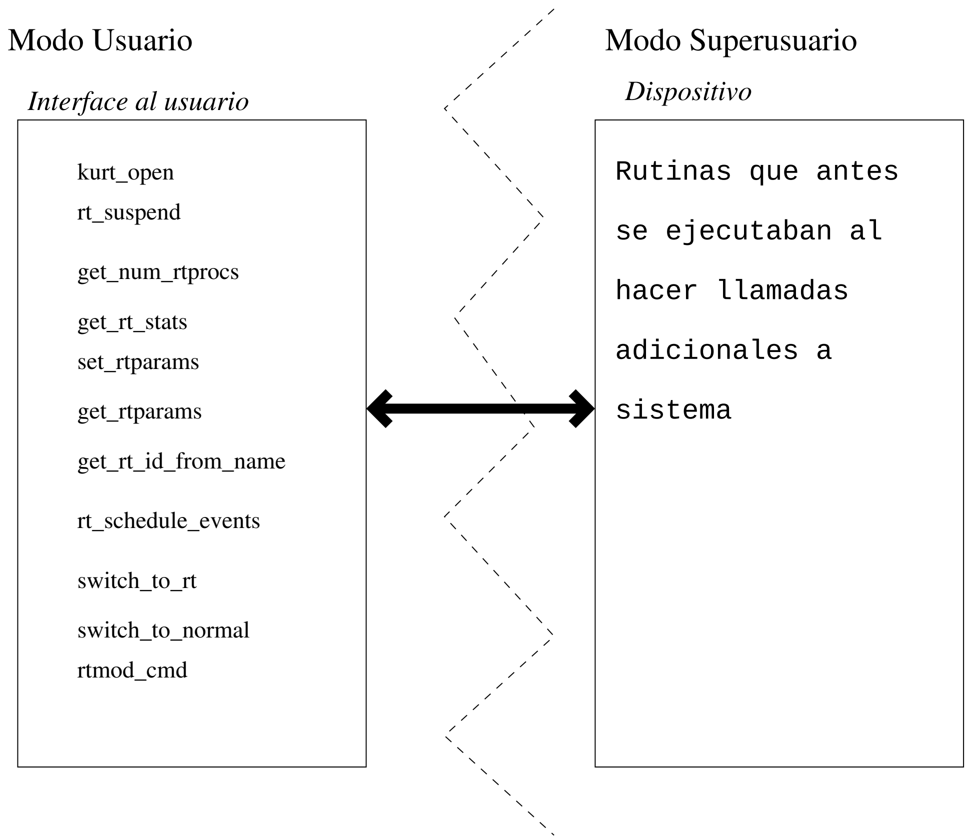

Figura 2: Interface al usuario y dispositivo KURT

Detalles operativos

Antes que nada se debe crear el archivo especial para el dispositivo KURT usando el comando *mknod* [2]. En nuestro caso use el siguiente comando como root o sudo:

mknod /dev/kurt c 172 0

Donde estamos declarando que el dispositivo KURT es un dispotivo orientado a caracteres con un número mayor 172 y un menor de 0.

Después monte el dispositivo usando el siguiente comando:

insmod kurt.o

El cual nos permite registrar el dispositivo KURT.

---

[5] William Dinkel es asistente de investigador en el Centro de Tecnología en Información y Telecomunicaiones (ITTC) de la Universidad de Kansas dinkel@ittc.ukans.edu

5 Prestaciones

Intuitivamente, podriamos decir que no debe de haber cambio en las prestaciones, ya que solo cambiamos la interface pero las rutinas son las mismas. Sin embargo, encontramos una mejora en el comportamiento de KURT usando esta nueva interface. A continuación voy a describir las condiciones en las que se llevaron a cabo los experimentos.

Todos los experimentos se corrieron en un sistema ligeramente cargado. El sistema usado cuenta con un procesador Pentium Pro a 200Mhz, 64 MB de RAM, 6GB de disco duro con un núcleo 2.2.13. La primera prueba se hizo usando la versión KURT con llamados a sistema. El segundo experimento se corrió usando la versión con el dispositivo KURT, el cual no usa adiciones a la tabla de llamadas a sistema. El dispositivo KURT mostró un mejor comportamiento en el caso de acontecimientos periódicos.

KURT 2.0 usando llamados a sistema

El programa usado en ambos experimentos fué *periodic.c* el cual es incluido en la distribición estandard de

KURT 2.0[6]. periodic se corrio para generar 300 iteraciones. Cada Iteración requirió 600 microsegundos con periodo de 1 milisegundo. Se corrio en el modo rt-2, el cual permita tener procesos de RT y normales al mismo tiempo. Esta prueba se corrió varias veces con resultados similares.

El comando usado fué: *periodic 1 600 300 1 ./ rt-2 1000*

A continuación vamos a observar el comportamiento de KURT, usando llamados a sistema, en cuanto a su capacidad de ejecutar tareas en un periodo prederminado. Idealmente todas las tareas deberían ejecutarse con un espacio de 1 milisegundo entre inicio de tareas. Es decir que un histograma de la diferencia en microsegundos entre eventos consecutivos contra la frecuencia ese número de microsegundos, deberiamos obtener una línea vertical en 1 milisegundo con una frecuencia de 300.

En la figura 3 podemos observar que de los 300 eventos programados a ocurrir en intervalos de 1 milisegundo la mayoría, 90 % aproximadamente, estuvieron alrededor del milisegundo. Sin embargo, el 10 % restante se ejecutó con retardos de hasta 10 milisegundos. El comportamiento de ese 10 % se muestra en la figura 4.

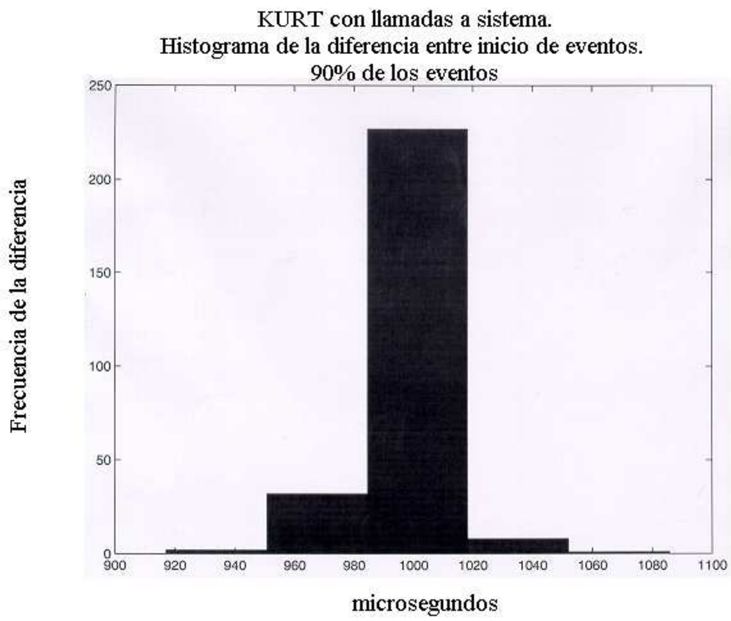

Figura 3: KURT con llamadas a sistema. Histograma con 90% de eventos

Ahora es importante analisar la periodicidad de los eventos desde una perspectiva temporal y no un histograma.Es importante que notar que los plazos fallados son periódicos. Este comportamiento se aprecia en la Figura 5. La razón para este mal comportamiento es todavía una incognita para nosotros.

[6]`http://www.ittc.ukans.edu/kurt/`

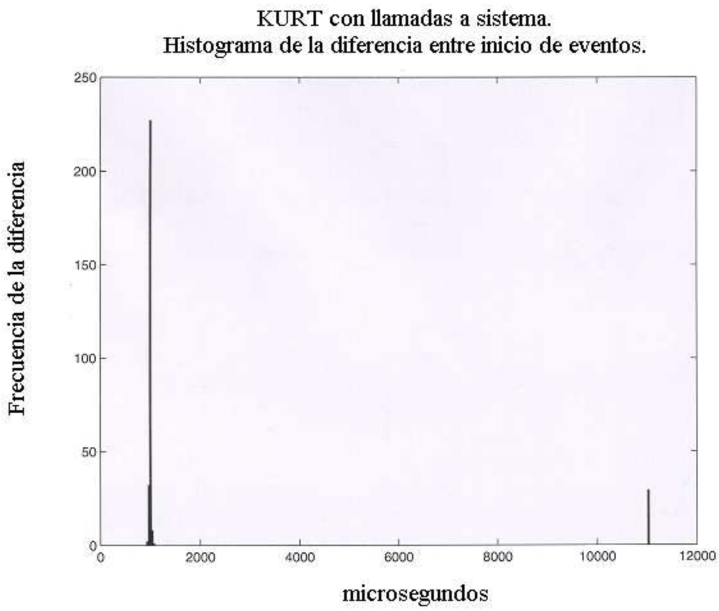

Figura 4: KURT con llamadas a sistema. Histograma con 100% de eventos

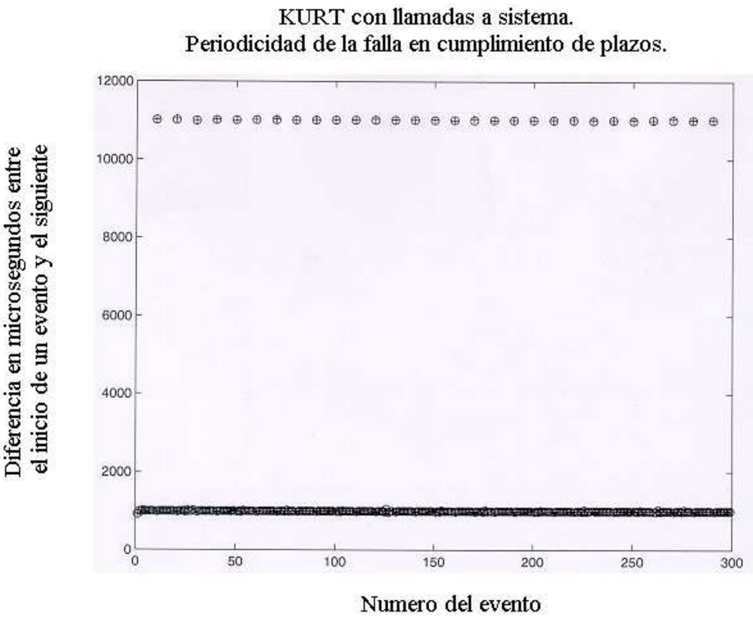

Figura 5: KURT con llamadas a sistema. Periodicidad de los plazos fallados

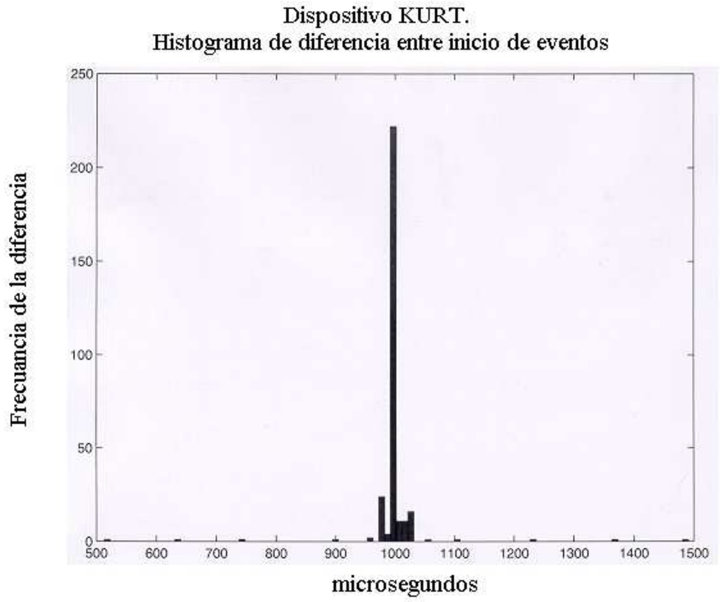

Figura 6: Dispositivo KURT. Histograma del 100% de los eventos

Dispositivo KURT

El programa usado en esta caso es una versión modificada de *periodic.c* para usar el dispositivo KURT. periodic se corrio para generar 300 iteraciones. Cada iteración requirio 600 microsegundos con un periodo de 1 milisegundo. También se corrió en modo rt-2. Igualmente se corrió varias veces obteniendo resultados similares.
El comando usado fué: *periodic 1 600 300 1 ./ rt-2 1000*

La gran mayoría de los plazos se cumplieron. Sin embargo, hay una gran variación alrededor del plazo de 1 milisegundo. Los picos que aparecian en la versión con llamados a sistema desaparecieron como se ve en la Figura 6.

La desaparición de esos picos alrededor de los 10 milisegundos es una gran ventaja, ya que ahora sabemos a pesar de que habrá una variación esta no será de la magnitud que se tenia antes.

Con la intensión de comparar el comportamiento de ambas versiones en el eje temporal se ha incluido la figura 7. En esta figura podemos observar que la variación de los plazos alrededor del milisegundo es mucho menor usando el dispositivo KURT que usando la versión con llamados a sistema, vease la figura 5. Es importante notar que en la gráfica de tiempo se presentan oscilaciones en el inicio y se estabilizan.

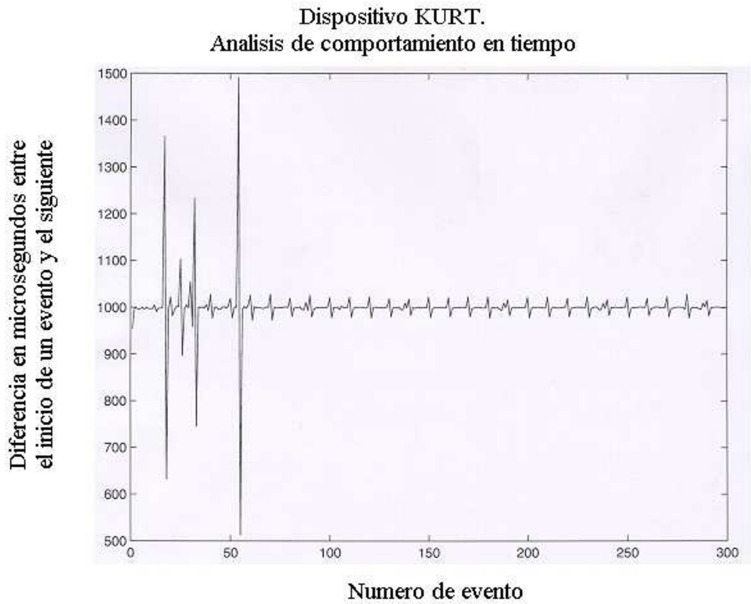

Figura 7: Dispositivo KURT. Análisis temporal del comportamiento del dispositivo KURT

6 Conclusiones y trabajo a futuro

El uso de dispositivos permite reducir las modificaciones al núcleo, si bien la propuesta ideal es no modificarlo aun estamos lejos de ello. Al tener dispositivos la interface es bastante más limpia. Sólo se puede acceder a los servicios del dispositivo usando la llamda a sistema IOCTL.

En cuanto a nuestro trabajo a futuro planeamos convertir varias extensiones al núcleo a dispositivos, lo cual permitirá los desarrolladores tener varios servicios con menos modificaciones al núcleo.

Referencias

[1] Marco Spuri and Giorgio Buttazzo.*Scheduling Aperiodic Tasks in Dynamic Priority.* The Journal of Real-Time Systems,Vol. 10, No. 2, pp. 179-210, March 1996.

[2] Richard Stones and Neil Matthew. *Beginning Linux Programming 2nd Edition.* Wrox Press LTD. Nov 1999.

[3] Balaji Srinivasan et al. *A firm real-time system implementation using commercial off-the shelf hardware and software.* In Proc. IEEE Real-Time Technology and Application Symposium, pp. 112-119. June 1998.

[4] Manoj Apte et al. *Time Based Linux for Real-Time NOWs and MPI/RT.* IEEE RTAS 1999.

[5] Krithi Ramamrithan and John A. Stankovin. *Scheduling Algorithms and Operative Systems Support for Real-Time Systems* Proceedings of the IEEE. Vol 82. No. 1. pp 55-67. January 1994.

[6] Douglas Niehaus et al. Effective Real-Time system Implementation with KURT LINUX.

[7] `http://www.ittc.ukans.edu/kurt`.

[8] Victor Yodaiken. *The RTlinux Manifesto.*`http://www.rtlinux.org/rtlinux/index.html`

[9] Yu-Chung Wang and Kwei-Jay Lin. *Implementing a General Real-Time Scheduling Framework in RED-Linux Real-Time Kernel.* IEEE RATS 1999.

[10] Shuichi Oikawa and Ragunathan Rajkumar. *Portable RK: A Portable Resource Kernel for Guaranted and Enforced Timing Behavior.* www.cs.cmu.edu

[11] Beck et all. *Linux Kernel Internals.*Addison-Wesley. 1st edition. 1996. ISBN 0201877414

[12] C. L. Liu. and J.W.Layland. *Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment*, JACM, Vol. 20, No. 1, pp46-61, 1973.