Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

Pruebas de desempeño al sistema operativo GNU/Linux como sistema operativo de tiempo real.

Autor: Erio Elio Peña Rodríguez

Tutor: Msc. Fidel González Vásquez

Santa Clara

2008

"Año 50 de la Revolución"

Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

Pruebas de desempeño al sistema operativo GNU/Linux como sistema operativo de tiempo real.

Autor: Erio Elio Peña Rodríguez

erio_elio@uclv.edu.cu

Tutor: Msc. Fidel González Vásquez

fidel@uclv.edu.cu

Profesor Auxiliar

Departamento de Automática y Sistemas Computacionales.

Santa Clara

2008

"Año 50 de la Revolución"

Hago constar que el presente trabajo de diploma fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de estudios de la especialidad de Ingeniería en Automática, autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

_____

Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del Autor

Firma del Jefe de Departamento donde se defiende el trabajo

_____

Firma del Responsable de Información Científico-Técnica

PENSAMIENTO

El futuro tiene muchos nombres. Para los débiles es lo inalcanzable. Para los temerosos, lo desconocido. Para los valientes es la oportunidad.

Víctor Hugo

DEDICATORIA

A mi padre por ser la persona que más admiro en el mundo…

AGRADECIMIENTOS

Agradezco a mi padre, a mis compañeros y amigos, además la participación de los profesores que estuvieron en cada momento junto al grupo dándonos ánimo y consejos. No dejo de recordar a aquellos que no se encuentran entre nosotros pero que jugaron un papel decisivo en los cinco años de universidad, a todos ellos debo lo que soy.

TAREA TÉCNICA

- Estudio del estado del arte de sistemas operativo de tiempo real.
- Análisis de las métricas de software para sistemas de tiempo real, con el objetivo del diseño de las pruebas a realizar.
- Estudio de pruebas existentes.
- Diseño e implementación de las pruebas a realizar.
- Realización de las pruebas concebidas.
- Comparación con sistemas ya probados para aplicaciones de tiempo real.
- Análisis de los resultados.
- Elaboración del informe.

Firma del Autor

Firma del Tutor

RESUMEN

En el presente trabajo se presenta como objetivo demostrar mediante pruebas realizadas al kernel de Linux, la posibilidad de que la versión 2.6.24 tanto con el parche de baja latencia del núcleo activado como sin él, puede ser usada en aplicaciones de tiempo real. Para estas pruebas se siguieron caminos como verificar la planificación del sistema, en este caso crearía un amplio rango de prioridades en cuanto a lo que sería gestión de procesos, y el objetivo de dicha prueba sería determinar el tiempo en el cual un hilo, que está listo para ejecutarse, tarda en tomar el procesador. También se realizaron pruebas con las señales que van por los hilos hacia el kernel. Aquí se calcula el tiempo desde que se generan señales hasta que la primera instrucción del manejo de esta señal es ejecutada. Se comprueba también la temporización, o sea la demora en pedir el tiempo al sistema. Estas pruebas arrojaron como conclusión la necesidad de no perder de vista al 2.6.24 a la hora de pensar en aplicaciones de tiempo real, para así obtener la calidad requerida en la investigación.

TABLA DE CONTENIDOS

INTRODUCCIÓN

Hoy en día se hace necesario el uso de los sistemas computacionales en un nivel creciente, la dependencia de las máquinas nos hace indagar en busca de lo que sea mejor y más barato en este ámbito para solucionar los problemas que nos rodean. Todo elemento relacionado con la computación está regido por un sistema operativo. En el mundo existe una variedad inmensa de sistemas operativos que no solo controlan procesos sino que tienen una gama de aplicaciones tan grande que van de lo simple a lo complejo. El uso de los que serían los sistemas más caros harían los procesos más fiables, pero una realidad que no podemos evadir es que vivimos en un país del tercer mundo y que además permanece bloqueado, lo que nos hace a la hora de diseñar, ser dependientes de algunas cuestiones como el presupuesto. Además no sólo debemos pensar en esta limitación sino en la importancia del proceso que necesita tal sistema.

El tránsito hacia el software libre constituye una prioridad, que pasa por criterios tanto de seguridad informática como de corte económico, en medio de un cada vez más monopolizado mercado del software.

Una de las cuestiones que ha revolucionado el mundo es el uso del Linux en las aplicaciones más diversas, la presencia de este sistema operativo ha abierto un horizonte a los que como nosotros, han buscado siempre la fiabilidad de procesos a los que se le aplican los S.O. El mismo respeta la libertad de los usuarios, una vez que sea obtenido. Además el hecho de que sea libre nos permitirá usarlo, copiarlo, estudiarlo, modificarlo y redistribuirlo libremente.

Claro que aquella empresa que pida tus servicios no estará tan convencida de que las cosas son de la forma que lo explicabas y más aún aquellas que han de manejar variables de sistemas que actúan en tiempo real, lo que nos llevaría a demostrar el uso de este S.O. en aplicaciones de tiempo real.

Ya de Linux se han venido sucediendo una buena cantidad de versiones pero más específicamente en este Trabajo de Diploma se presenta como objetivo el análisis de algunas de las métricas principales, así como el diseño de pruebas para comprobar que es

viable el uso de la versión 2.6.24 del núcleo del S.O. GNU\Linux específicamente para aplicaciones de tiempo real. Y como objetivos específicos:

- Estudio del estado del arte de sistemas operativo de tiempo real.
- Análisis de las métricas de software para sistemas de tiempo real, con el objetivo del diseño de las pruebas a realizar.
- Estudio de pruebas existentes.
- Diseño e implementación de las pruebas a realizar.
- Realización de las pruebas concebidas.
- Comparación con sistemas ya probados para aplicaciones de tiempo real.

Para el desarrollo de estas pruebas de desempeño del sistema se aplicará un tiempo real donde se analizarán algunos de los parámetros de más importancia como lo serían el caso de la planificación. La latencia de la misma es el objetivo de la prueba que se le aplicará, algo que no es más que determinar el tiempo en el cual un hilo, que está listo para ejecutarse, es capaz de tomar el procesador, o sea aquí se calcula el tiempo desde que se generan señales hasta que la primera instrucción del manejo de esta señal es ejecutada.
En caso de las señales se ha de probar qué demora existe cuando una señal es enviada por un hilo hacia el sistema, en todo caso nos ofrecerá el tiempo que demora en realizarse el manejo de la señal. Por el mismo ámbito se envía una señal de un hilo a otro y reflejar el tiempo que se demora en capturar y manejar una señal enviada por uno a otro hilo. También es un parámetro la temporización, cuyo objetivo de la prueba es estimar cuánto se demora en preguntar el tiempo al sistema.
Todas estas pruebas se realizarán para la versión 2.6.24 estándar, así como para una misma versión, pero ya personalizada buscando una comparación entre los resultados de las mismas, para ampliar nuestras fronteras en la demostración de la aplicación del 2.6.24 en aplicaciones de tiempo real. Cuando se habla de una versión personalizada es la misma variante pero con la diferencia de que el parche de baja latencia que trae incluido en el núcleo se le activará en este caso, y se le realizarán las mismas pruebas con el objetivo de hacer comparaciones de los resultados de ambas versiones con los obtenidos de la versión 2.6.18 ya probada para aplicaciones de tiempo real.

CAPÍTULO 1. REVISIÓN BIBLIOGRÁFICA

1.1 ¿Qué es Tiempo-Real?

Debemos antes de ver lo que sería RT-Linux analizar algunas ideas esenciales sobre que es tiempo real. Diremos que:

"Un sistema de tiempo real es aquel sistema informático en el que la corrección del sistema no sólo depende de los resultados lógicos de los algoritmos, sino que también depende del momento en el que estos se producen."

Aquí no sólo lo que se busca es que al final los resultados sean los correctos, sino que hay un intervalo hábil para la obtención de los mismos o sea un intervalo de tiempo especificado dentro del cual ha de ser cumplida la tarea. Con lo anteriormente especificado no queremos decir que los sistemas de tiempo real tengan que ser necesariamente rápidos, como quizás se pudiera pensar, solo tienen que cumplir con su acción en un tiempo predeterminado, el cual no afecte el funcionamiento del proceso donde sea usado. Observe que lo que se ha definido es un sistema "de tiempo-real" y no un sistema "en tiempo-real". Un sistema "en tiempo-real" es lo que normalmente se entiende por un sistema rápido, capaz de dar la impresión de "realidad". Típicamente todas las simulaciones y juegos interactivos quieren dan la impresión de continuidad en el tiempo y cuantas más imágenes generen mejor.

Si el sistema está compuesto por una sola tarea, entonces no hay problema de tiempo real: o sea el procesador puede o no hacer el trabajo en el tiempo requerido. En el caso de no ser lo suficientemente rápido el procesador, entonces simplemente se cambia el procesador.

Los problemas de TR aparecen cuando el sistema está compuesto por varias tareas, y hay que repartir el procesador (o procesadores) entre todas ellas. Para ello no podemos utilizar

un sistema clásico de tiempo compartido como puede ser el utilizado por Linux con los procesos normales.

El diseño de un sistema de tiempo real pasa por varias fases. Primero se identifican las tareas a realizar y las restricciones temporales que deben cumplir; luego se escriben los programas que ejecutarán las tareas; después se mide el tiempo de cómputo de cada tarea y se realiza un análisis de la planificabilidad. El análisis de la planificabilidad consiste en aplicar unas pruebas al conjunto de tareas de tal forma que si estas pasan el test entonces se puede garantizar que ninguna tarea perderá su plazo de ejecución, si no pasan el test entonces se tiene que volver al principio y empezar de nuevo utilizando otro procesador más potente o utilizando otros algoritmos para implementar las tareas.

1.1.1 Clasificación de sistemas de tiempo real.

Existe una clasificación de los sistemas de tiempo real en función de la gravedad de la situación que se produciría en caso de que el sistema sobrepasara los plazos que se le asignaron.

Sistemas de tiempo real acrítico (Soft Real Time Systems):

En este tipo de sistemas, el incumplimiento de plazos conlleva una degradación de la calidad del servicio ofrecido. Si el incumplimiento es continuado, la degradación se puede hacer perceptible por el usuario, llegando a ser intolerable en el peor de los casos. Por lo tanto, la peor consecuencia del hecho de que el sistema sobrepase los plazos es la de provocar al usuario un cierto grado de malestar. Dentro de este grupo de sistemas podemos encontrar los servicios de audio y vídeo en tiempo real a través de Internet. Estos servicios pretenden transmitir un flujo constante de información a través de una red de paquetes que no garantiza la entrega y menos aún la entrega en un plazo determinado. (García 2001)

Sistemas de tiempo real crítico (Hard Real Time Systems):

Los sistemas pertenecientes a este grupo deben cumplir estrictamente los plazos de tiempo asignados, de lo contrario pueden producirse pérdidas económicas importantes e incluso pérdida de vidas humanas. Ejemplos representativos de esta clase son los sistemas empleados en control industrial y de centrales nucleares o los utilizados en los sectores

ferroviario, automovilístico y aeroespacial por citar algunos de los más importantes. El más mínimo fallo en este tipo de sistemas puede ser fatal y debido a sus características, tener repercusión internacional. (García 2001)

1.1.2 Sistemas operativos y los sistemas de tiempo real

La corrección semántica de la respuesta es responsabilidad del programador, y la corrección temporal depende del S.O..

Es el S.O. el que tiene que dar soporte y organizar la ejecución de todas las tareas, es también su labor gestionar las interrupciones.

El S.O. ha de ofrecer:

- El algoritmo de planificación.
- Los mecanismos de comunicación entre tareas.
- Gestionar las interrupciones.
- Activar las tareas en cada uno de sus períodos.

Al contrario de lo que sucede en los S.O. "normales", el objetivo en los de tiempo real es minimizar la complejidad para minimizar la incertidumbre (o sea la falta de predecibilidad). No se quiere un S.O. que haga muchas cosas, sino uno lo haga de forma predecible y rápida. Es preferible uno que normalmente tarde 10 unidades de tiempo (u.t.) en realizar un cambio de contexto y que en el peor de los casos tarde 12, que otro que por término medio tarde 3 u.t. pero de vez en cuando necesite 20 u.t. (Corporation 2007)

No hay que sorprenderse si descubrimos que los sistemas operativos de tiempo real son más "lentos" que los S.O. normales. En ocasiones, para obtener un comportamiento predecible, se puede llegar incluso a deshabilitar la memoria cache, con la consiguiente pérdida de velocidad. La memoria cache, los procesadores con unidades pipeline y los algoritmos de predicción de saltos son claros enemigos de la predecibilidad y por tanto de los sistemas de tiempo real. (Locke 1992)

En la actualidad en muchos de las industrias así como algunos procesos aislados hace uso de los S.O. tanto para la adquisición de datos como para el propio control de sistemas, por eso es de vital importancia en nuestra vida cotidiana .Pero como habíamos dicho antes para cada caso hay que analizar bien porque no siempre aquel sistema más rápido será el más fiable. (Corporation 2007)

Para nadie es un secreto que algo que influye de forma contundente en todos aquellos proyectos que se presentan es garantizar la fiabilidad, ahora el S.O. que más ha revolucionado en los últimos tiempos es el GNU\Linux por las ventajas que ha mostrado para grandes aplicaciones y no solo eso, también en el propio desempeño del mismo, por lo que es en gran medida el señalado para cumplir con la efectividad de los proceso.

Este sistema ha sido promovido para aplicaciones que cada vez alcanzan una mayor escala por lo que es de gran interés para todos aquellos que serán sus usuarios el saber hasta donde es capaz de llegar y en que es superior.

1.2 GNU/Linux.

Linux es un núcleo (kernel) de S.O.. Habitualmente se distribuye junto con una serie de aplicaciones desarrolladas por el proyecto GNU para formar un S.O. completo llamado GNU/Linux. Es software libre, lo que quiere decir, fundamentalmente, que se puede leer o modificar el código, distribuir (original o modificado), copiar y utilizar libremente. No se permite plagiarlo, ni distribuirlo sin dar acceso al código fuente, ni cambiarle la licencia. (ZAMARRÓN 2004)

Linux debe su nombre a su desarrollador original, Linux Torvalds, que era estudiante de Informática en una universidad de Helsinki (Finlandia) cuando en 1991 se decidió a hacer un núcleo de S.O. que funcionara como MINIX (un derivado de UNIX). Es muy compatible con UNIX, pero no es un UNIX en el sentido de que por dentro funciona como ha querido Linux y los cientos de desarrolladores que trabajan en el por Internet desde su creación.

El planificador (Schedule) es la parte del S.O. que permite la multiprogramación y el multiproceso mediante la asignación de procesos a la CPU del sistema y el intercambio de procesos en ejecución. Ha cambiado completamente de una versión a otra, se mantienen las principales estructuras de datos, pero se ha cambiado el algoritmo para hacerlo y con ello, más escalable. (Pranevich 2003)

1.2.1 Características generales.

- Linux es un kernel ***monolítico*** *o* ***modular,*** en algunos casos. Eso quiere decir que el núcleo hace todo (proporciona todos los servicios del S.O.) y todas las capas del núcleo tienen acceso a todas las estructuras de datos, rutinas y componentes del sistema.

Monolítico: El kernel trabaja soportando el hardware como característica, es decir, no hay nada de manera modular, el kernel se arranca con el sistema y no hay más controladores que los que necesita el hardware que tenemos en ese momento, eso sí, en caso de que instalemos un nuevo hardware, tendremos que recompilar el kernel para dar soporte al nuevo dispositivo. Esto ahorra espacio en disco entre otras cosas, antes cuando las máquinas eran más lentas y de menos capacidad estaba bien para poder optimizar al máximo el rendimiento, con el hardware de hoy en día no tiene demasiado sentido.(Autores 2008)

Modular: El tamaño del kernel en el disco es mayor, tenemos soporte para todo el hardware que soporta el kernel, por lo tanto al arrancar el S.O., este detectará el hardware que tenemos y cargará los módulos que sean necesarios. En los días de hoy es la mejor opción porque los ordenadores son muy potentes y nos sobra tanto RAM como espacio en el disco duro.

- Es capaz de *linkar módulos en tiempo de ejecución*.

Los módulos contienen funcionalidades particulares que son cargadas dentro del Kernel cada vez que se requiere y no están permanentemente alojados dentro del Kernel en sí, ni dentro de la memoria. Además de la reducción en el uso de memoria, los módulos tienen la

ventaja de ser actualizables. Al compilar el núcleo se da con cada módulo la opción de incorporarlo estáticamente al binario del kernel o bien compilarlo aparte para que luego se pueda cargar según demanda.

- Tiene un diseño *reentrante* de modo que pueden existir varios procesos en modo núcleo 'ejecutándose' a la vez (en monoprocesadores, todos menos uno estarían en la cola de listos).

- Tiene soporte para *aplicaciones de usuario multi-hilo*.

Saber cómo manejar adecuadamente los hilos debe ser una parte cotidiana del repertorio de todo buen programador. Los hilos son similares a procesos. A los hilos, como a los procesos, se les asignan porciones de tiempo por el kernel. En sistemas con un solo procesador el núcleo divide el tiempo asignado a cada hilo para simular la ejecución simultánea de hilos de forma muy similar a como lo divide para los procesos. En sistemas con más de un procesador, los hilos pueden ejecutarse simultáneamente, del mismo modo que dos o más procesos pueden ejecutarse simultáneamente también. Así que, ¿por qué son los multi-hilos preferibles a múltiples procesos independientes cooperativos? Bien, los hilos comparten la misma ubicación en memoria. Hilos independientes pueden acceder a las mismas variables en memoria. Así pues todos los hilos del programa pueden leer o escribir a los enteros (integers) declarados globalmente.

Antes de la versión 2.6 del *kernel Linux*, no había soporte real para manejar hilos a nivel de *kernel*. Las primeras versiones para el manejo de hilos en Linux (*LinuxThreads*) se implementaron en el área de usuario mediante llamadas a una función de sistema clone, capaz de crear una copia de un proceso. Esta primera implementación además estaba muy lejos de ser compatible con los estándares *POSIX* de hilos de ejecución (*POSIX threading standards*). Para mejorar el soporte de hilos en Linux, fueron necesarios retoques tanto a nivel de kernel (y su *ABI*) como a nivel de las librerías *GNU C Library (Glibc)* que implementaban el concepto de hilo. Esto dio lugar a varios modelos/proyectos para la gestión de hilos, dependiendo de lo nuevo que fuera el sistema:

1. ***LinuxThreads*** (con tamaño fijo de la pila).
2. ***LinuxThreads*** (con pilas flotantes).
3. ***NGPT (NET Generation POSIX Threads)*** liderado por IBM y abandonado por NPTL.
4. ***NPTL (Native POSIX Thread)*** Proyecto liderado por Red Hat y liberado en 2003.

Al añadir nuevas posibilidades al trabajo con hilos en el *kernel* de *Linux*, se inició una etapa de transición en la que algunas aplicaciones requerían un esquema de hilos moderno (*NPTL*), mientras que otras utilizaban otro más tradicional como *LinuxThreads*. Todo ello, teniendo el sistema que poder correr aplicaciones basadas en un modelo de hilos más o menos avanzado.

Como se ha comentado al inicio, los nuevos modelos de soporte de hilos requerían cambios también a nivel de *kernel*. Así, las diferentes librerías dinámicas (*DSO Dinamic Shared Objectso/Shared Libraries*), además de una librería de hilos concreta, requerirán de una versión del *kernel* con una interfaz capaz de trabajar con dicha librería. A esta interfaz se la conoce con el nombre de *Application Binary Interface (ABI).*Una *ABI* define la interfaz de bajo nivel entre la Aplicación y el S.O. o *kernel,* mientras que una *API* define el interfaz entre el código fuente y las librerías. De la misma forma que una *API* permite que un código fuente sea compilado en cualquier sistema que soporte para dicha *API*, la *ABI* permite que un objeto binario pueda funcionar en un sistema con una *ABI* compatible.

- Compatible con *estándares*: POSIX, las APIs del UNIX SysV y los sockets BSD, sistemas de archivos para todos los gustos.
- Admite *multiproceso simétrico* (SMP). Una de las formas más fáciles y baratas de aumentar el rendimiento del hardware es poner más de una CPU en la placa. Esto se puede realizar haciendo que CPUs diferentes tengan trabajos diferentes (multiproceso asimétrico) o haciendo que todos se ejecuten en paralelo, realizando el mismo trabajo (multiproceso simétrico o SMP). El hacer multiproceso asimétrico requiere un conocimiento especializado sobre las tareas que la computadora debe ejecutar, lo que no está a nuestro alcance en un sistema operativo de propósito

general como Linux. En cambio el multiproceso simétrico es relativamente fácil de implementar.(Rey 2007)

1.3 Características de Linux en tiempo real.

Las características que se obtienen con un sistema Linux en tiempo real son las siguientes:

- Soporte de múltiples arquitecturas y válida para arquitecturas multiprocesador.
- Gestión de procesos: Planificación, amplio rango de prioridades, creación y eliminado de hilos, etc.
- Gestión de memoria: No protección de memoria en el kernel y no asignación de memoria de forma dinámica.
- Comunicación entre procesos: Semáforos, Mutex, control de inversión de prioridades, memoria compartida y FIFO's.
- Tiempo y relojes: Resolución de nanosegundos. No relojes de usuario. facilidades para añadir nuevos relojes hardware.
- Programación de drivers: Se proporcionan funciones de acceso a dispositivos.

No proporciona herramientas de calidad de servicio.

1.3.2 Novedades en la versión 2.6.24 del núcleo de Linux.

A continuación veremos algunas de las características más novedosas que trajo la aparición del 2.6.24(Autores 2007):

- **Soporte de Tickless**: Una nueva infraestructura para gestionar el tiempo muerto de la CPU.
- **Markers**: Esta es una de las partes que conforma lo que podríamos llamar "DTrace de Linux".

- **Puntos de montaje "bind" de sólo lectura**: Ahora se pueden hacer monturas "bind" amarre de sólo lectura.
- **Mejoras del gestor de procesos**: Se podrá configurar cómo particionar el tiempo de CPU entre usuarios o grupos.
- **Antifragmentación de la memoria**: Una mejora definitiva que afecta a la asignación de memoria al más bajo nivel posible.
- **Task Control Groups**: Esta es una característica que sirve para la asignación y control de recursos a grupos de procesos.
- **Drivers para dispositivos inalámbricos**: En Linux se añaden drivers continuamente, pero en esta ocasión se ha añadido una buena cantidad más.
- **Autorización USB**: Ahora la pila USB puede configurarse a través de sysfs para que los dispositivos USB conectados al sistema no estén "autorizados" para funcionar.
- **Reunificación x86-32 y x86-64**: Ya no hay arch/i386 y arch/x86_64, ahora hay arch/x86 (aunque se conservan algunos enlaces simbólicos por compatibilidad).
- **Espacios de nombres para los PIDs y la pila de red**: Esto tiene que ver con la virtualización a nivel de S.O., tipo vServero así como OpenVZ, las "jailso" de los BSDs o los *Containers* de Solaris.
- **Límites de memoria "sucia" para cada dispositivo**: Los límites de la cantidad de memoria que puede estar "sucia" (sin escribir a disco) son independientes para cada dispositivo y se calculan dinámicamente.
- **Soporte SPI/SDIO en la capa MMC**: Las ranuras donde se ponen las tarjetas de memoria MMC en las cámaras digitales pueden actuar como *bus* en el caso de que fabricante le añada los cables y chips necesarios, y que se les puede conectar tarjetas de red además. Ahora Linux soporta esa tecnología.

1.4 Soluciones al problema de tiempo real.

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 de fuente cerrada. Pero las variantes de Linux para tiempo real ofrecen una alternativa de fuente

abierta a las soluciones comerciales así que veremos de qué forma se enfrentó Linux al problema de tiempo real.

1.4.1 Soluciones al problema de tiempo real. RT-Linux.

RT-Linux ha sido desarrollado en el departamento de informática en el Instituto de Minería y Tecnología de Nuevo México, por Víctor Yodaiken y Michael Barabanov. RT-Linux fue el trabajo de Michael para completar su Maestría en Ciencia de la Computación (algo así como el proyecto fin de carrera). (Liu 2000)

RT-Linux resuelve el problema de una forma radicalmente distinta. En lugar de modificar el núcleo de Linux para ofrecer nuevas llamadas al sistema y que sea predecible, lo que hace es construir directamente sobre el procesador (i386) un pequeño núcleo (que no tiene nada que ver con el núcleo de Linux) con un planificador. Sobre este núcleo el S.O. Linux se ejecuta como una tarea más. Linux se ejecuta compartiendo el procesador con otras tareas. Más concretamente: Linux se ejecuta en background, cuando no hay otras tareas que ejecutar.

RTLinux (*Real-Time Linux*) ofrece prestaciones de ***sistema de tiempo real crítico***, 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 esta 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 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 ser eficiente en ninguna de ellas.

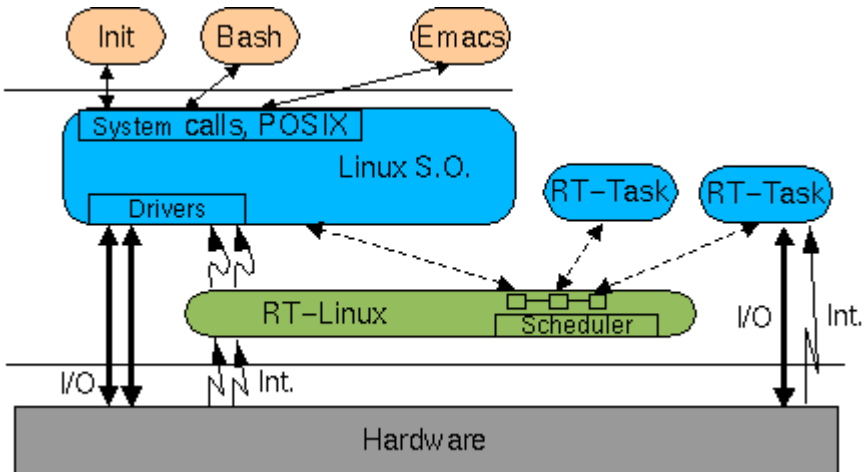
Las tareas de RTLinux no pueden hacer un uso directo de los servicios del kernel, ya que no es posible que realicen llamadas al sistema. Lo que si es posible es un uso indirecto si los hilos de tiempo real se comunican con procesos Linux. Para realizar esto, RTLinux incorpora dos mecanismos: la memoria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

El código del kernel de Linux (como cualquier S.O.) suele deshabilitar las interrupciones como medio de sincronización o para implementar secciones críticas. Si mientras Linux tiene deshabilitadas las interrupciones se produce una interrupción de reloj, esta quedará bloqueada, con la consiguiente pérdida de precisión temporal. En RT-Linux se ha implementado una solución muy elegante: se ha sustituido todas las llamadas a cli, sti e iret (instrucciones ensamblador que modifican el estado de las interrupciones) por S_CLI, S_STI y S_IRET que las emulan, de forma que Linux no puede nunca deshabilitar las interrupciones.

El planificador por defecto que viene con RT-Linux es un planificador basado en prioridades estáticas y trata a la tarea Linux como la tarea de menor prioridad. Si las tareas de tiempo real consumen todo el tiempo del procesador, entonces la tarea Linux no recibe tiempo de procesador y da la impresión de que el sistema se ha "colgado".

Con RT-Linux tenemos a la vez un sistema de tiempo real y un S.O. clásico. Podemos navegar por la red a la vez que estamos muestreando y controlando un proceso físico.

Figura 1.1 Sistema RT-Linux.



Un S.O. multiusuario y multiprogramado (multitarea) pretende crear la ilusión a sus usuarios de que se dispone del sistema al completo. La capacidad de un procesador de cambiar de tarea o contexto es infinitamente más rápida que la que pueda tener una persona normal, por lo que habitualmente el sistema cumple este objetivo. Es algo parecido a lo que pasa en un restaurante de comida rápida: por muy rápido que seas comiendo, normalmente la velocidad de servir comida es mucho mayor. Si un camarero fuese atendiéndote cada 5 minutos, podrías tener la sensación de que eres el cliente más importante del local, pero en realidad lo que está haciendo es compartir sus servicios (recursos) entre todos los clientes de forma rápida ("time-sharing").

1.3.1 Soluciones al problema de tiempo real. RTAI.

RTAI es una implementación de Linux para tiempo real basada en RTLinux. RTAI además proporciona una amplia selección de mecanismos de comunicación entre procesos y otros servicios de tiempo real.

Adicionalmente, RTAI proporciona un módulo llamado LXRT para facilitar el desarrollo de aplicaciones de tiempo real en el espacio de usuario.

El enfoque de diseñó 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 implementación. 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 deseadores 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 aquel para la implementación de aplicaciones multimedia. Sin embargo, en las versiones más recientes se ha añadido el modulo 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 firme *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. La orientación de RTAI es la misma que NOVATICA 64 Edición digital/ ©ATI 2003 may./jun. 2003 n¼163 la de RTLinux, pero sus desarrolladores han optado por añadirle más funcionalidades. RTAI es más flexible y puede dar garantías de firma *real-time* en espacio de usuario sin renunciar a la realización de llamadas al sistema.

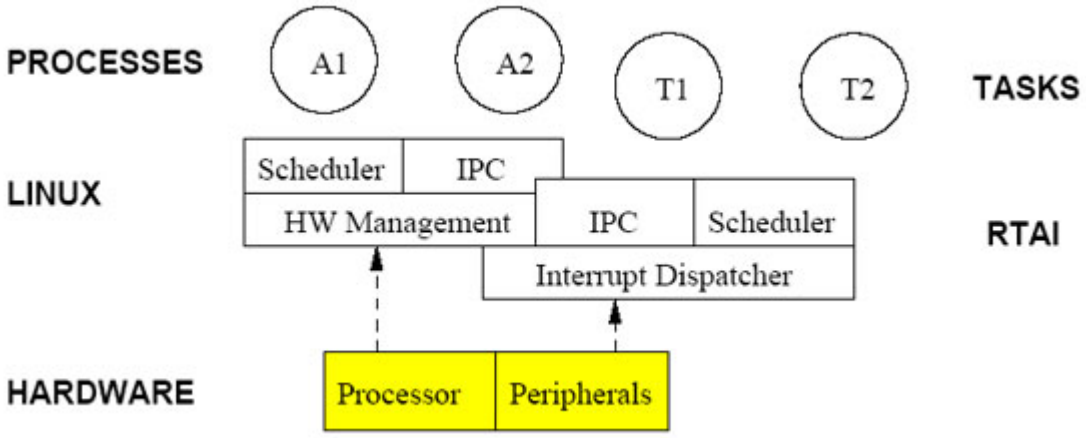
1.3.2 Fundamentos (arquitectura).

RTAI tiene una arquitectura similar a RTLinux. Al igual que RTLinux, RTAI trata el kernel estándar de Linux como una tarea de tiempo real con la menor prioridad, lo que hace posible que se ejecute cuando no haya ninguna tarea con mayor prioridad ejecutándose. Las operaciones básicas de las tareas de tiempo real son implementadas como módulos del kernel al igual que RTLinux. RTAI maneja las interrupciones de periféricos y son atendidas por el kernel de Linux después de las posibles acciones de tiempo real que hayan podido ser lanzadas por efecto de la interrupción.

La siguiente figura muestra la arquitectura básica de RTAI, que es muy similar a la de RTLinux. Las interrupciones se originan en el procesador y en los periféricos. Las originadas en el procesador (principalmente señales de error como división por cero), son manejadas por el kernel estándar, pero las interrupciones de los periféricos (como los relojes) son manejadas por RTAI Interrupt Dispatcher. RTAI envía las interrupciones a los

manejadores del kernel estándar de Linux cuando no hay tareas de tiempo real activas. Las instrucciones de activar/desactivar las interrupciones del kernel estándar son reemplazadas por macros que se enlazan con las instrucciones de RTAI. Cuando las interrupciones están desactivadas en el kernel estándar, RTAI encola las interrupciones para ser repartidas después de que el kernel estándar haya activado las interrupciones de nuevo.

Figura 1.2 Arquitectura RTAI**.**



Adicionalmente, se puede ver en la figura el mecanismo de comunicación entre procesos (IPC), que esta implementado de forma separado por Linux y por RTAI. También existe un planificador (sheduler) distinto para Linux y para RTAI.

Los desarrolladores de RTAI introducen el concepto de Real Time Hardware Abstraction Layer (RTHAL) que es usado para interceptar las interrupciones hardware y procesarlas después. RTHAL es una estructura instalada en el kernel de Linux que reúne los punteros a los datos internos del hardware relacionados en el kernel y las funciones necesarias por RTAI para operar. El objetivo de RTHAL es minimizar el número de cambios necesarios sobre el código del kernel y por tanto mejorar el mantenimiento de RTAI y del código del kernel de Linux. Con RTHAL, las diferentes operaciones (ej. manejar interrupciones) son más fáciles de cambiar ó modificar sin tener que interferir con la implementación de Linux. Por ejemplo, la estructura de RTHAL contiene la tabla de manejadores de interrupción, la

cual es una lista de las funciones que son llamadas para manejar las diferentes interrupciones. El cambio consiste únicamente en modificar unas líneas del código del kernel de Linux y añadir unas nuevas líneas.

Cuando RTHAL es instalado en Linux, las funciones y las estructuras de datos relacionada con la interacción con el hardware son reemplazadas por punteros a la estructura de RTHAL.

- **Planificación.**

La unidad de planificación de RTAI es la tarea. Siempre hay al menos una tarea, llamada kernel de Linux, que ejecuta como la tarea de menor prioridad. Cuando las tareas de tiempo real son añadidas, el planificador da entonces mayor prioridad a éstas sobre la tarea del kernel de Linux. El planificador proporciona servicios tales como suspend, resume, yield, make periodic, wait until, que son usadas en varios sistemas operativos de tiempo real.(Vela 2004)

El planificador es implementado como un módulo del kernel dedicado (contrario a RTLinux) lo que facilita la implementación de planificadores alternativos si es necesario.

1.4.2 Soluciones al problema de tiempo real. Parches.

Hasta ahora para darle a Linux características de sistema en tiempo real había que aplicar un parche en el kernel. Sin embargo a partir de Linux 2.6.18.x el código de este parche será incorporado en el núcleo (ZAMARRÓN 2004).TimeSys es la empresa que había desarrollado el parche, que ahora se centrará en prestar servicios asociados. Inicialmente el parche vino a dar solución a la problemática creada por muchos de que el Linux no era viable para aplicaciones de tiempo real (Tejedor 2004), primeramente venía aparte del núcleo y luego este ya estaba incluido en el mismo pero desactivado o sea que habría que pasar a una configuración personalizada para el uso del mismo .

El parche (CONFIG_PREEMPT_RT) nos permite obtener un sistema de Tiempo real en Linux en donde es posible ejecutar tareas en tiempo real y manejar las interrupciones en una máquina Linux estándar. Estas tareas y los manejadores se ejecutan cuando se necesitan, el peor caso es entre que se detecta la interrupción de hardware y el procesador ejecuta la primera instrucción del manejador de la interrupción. Este tiempo es del orden de los 10 microsegundos en la plataforma x86.

CAPÍTULO 2. ANÁLISIS DE MÉTRICAS DE SISTEMAS Y DISEÑO DE PRUEBAS PARA SU COMPROBACIÓN.

Aquí en este capítulo abordaremos en que forma responde una PC ante distintas pruebas haciendo de forma general una comparación entre los resultados de un sistema estándar y uno personalizado buscando con esto probar la capacidad de usar la versión 2.6.24 del núcleo de GNU\Linux para aplicaciones de tiempo real. Entre las principales métricas a comparar se encuentran la comprobación de la planificación buscando resultados considerables a la hora de utilizar en el desarrollo de aplicaciones con altas restricciones temporales. También se ejecutaran acciones buscando el tiempo que demoraría el manejo de señales en una variedad de formas, el tiempo que tarda enviar una señal de un hilo al sistema, tiempo que tardaría de un hilo a otro entre otras.

En el caso de las pruebas las realizaremos sobre las principales métricas de un PC que posee las siguientes características:

PC UCLV:

RAM 512 MB

Pentium IV 2.8 GHz
Cache 1024 KB

2.1 Pruebas de desempeño en los sistemas operativos de tiempo real.

Existen diversos criterios a medir en las pruebas de desempeño:

1. Uno clásico pudiera ser el rendimiento (throughput).
    - ¿Cuán rápido puedo realizar la operación X?

- ¿Cuantas veces puedo realizar X en un segundo?
- Otro de los criterios importantes es el nivel de respuesta del sistema.
- ¿Cuán rápido el sistema responde a una entrada?
- Un criterio importante el determinismo del sistema.
- ¿Varía el desempeño de un sistema ante variaciones de la carga?

- Por último los recursos consumidos por el sistema pudieran ser importantes, como por ejemplo el espacio en memoria.

El primer criterio suele ser la prueba de desempeño más importante en los sistemas que no son de tiempo real. Si bien este criterio es también importante en los sistemas de tiempo real, si nos basamos en la propia definición de sistema de tiempo real, podemos concluir que el segundo criterio suele tener mayor peso. El tercero de los criterios suele ser uno inviolable cuando hablamos de los sistemas de tiempo real críticos. Y el último tiene un mayor peso en los sistemas empotrados.

2.1.1 Rendimiento del procesamiento (Throughput).

El rendimiento del procesamiento suele ser medido en número de operaciones por segundo: MIPS (millones de instrucciones por segundo), bytes por segundos, número de señales que se pueden enviar por segundo. Alternativamente se puede hablar en términos del tiempo que toma ejecutar una operación: 10 microsegundos para una llamada a una función *x()*.(Pacheco 2008)

2.1.2 Nivel de respuesta del sistema (Responsiveness).

Cuando hablamos del nivel de respuesta del sistema estamos hablando de la velocidad con la cual el sistema en cuestión responde a un evento determinado. Un ejemplo, en los sistemas de tiempo compartido, es el nivel de interactividad, respuesta, del sistema al accionar de las teclas. (Pacheco 2008) En los sistemas de tiempo real generalmente hablamos de nivel de respuesta ante interrupciones, ya que todos los eventos asíncronos del sistema (E/S del disco, atención a dispositivos, etc.) se implementan utilizando interrupciones.

Las mediciones más comunes incluyen:

- Latencia de las interrupciones.
- Latencia en la planificación.
- Tiempo de cambio de contexto.

Existen multitudes de mediciones adicionales, y uno de los grandes problemas, incluyendo las tres enunciadas anteriormente, es que diferentes investigaciones pueden significar cosas diferentes con la misma medición. De esta manera al revisar comparaciones debemos tener el cuidado de saber exactamente que nos están diciendo en cada caso concreto. Por ejemplo al analizar el peor de los casos en la atención a interrupción alguna investigación pudiera obviar la posibilidad de que dicha interrupción tenga que esperar por la ejecución de otras interrupciones y otra investigación no haya excluido esta posibilidad, entonces si realizáramos una comparación esta pudiera ser no del todo justa. Pero bueno en caso de este trabajo atenderemos a la latencia de planificación.

2.1.3 Determinismo.

El determinismo, en contraposición con la incertidumbre, nos indica el optimismo a la hora de que nuestro sistema nos responda con un nivel de respuesta deseado. En un sistema de tiempo real lo normal es encontrarnos con varios procesos que compiten por determinados recursos, y es en este esquema que debemos de ser capaces de determinar una respuesta al peor de los casos para cada proceso en análisis sin importar lo que este sucediendo en el sistema en su totalidad. Es decir, una aplicación en concreto debe responder, a condiciones variables, en un tiempo acotado sin importar cuántos procesos se estén ejecutando ni qué código estén ejecutando dichos procesos.

Por ejemplo ante la posibilidad de que nuestro sistema atienda varios dispositivos, accesos al disco duro, atención a la interfaz de red, interacción con el teclado etc., la aplicación de mayor prioridad debe mantener su ejecución sin violar el peor de los casos establecido para ella.

El determinismo es un parámetro difícil de medir, no obstante hay mecanismos que se pueden utilizar para estimar cuan determinista es nuestro sistema. Por ejemplo pudiéramos establecer una prueba en la cual se generan varias interrupciones periódicamente y cada

interrupción asociada a un proceso que realiza determinas operaciones en respuesta ante la ocurrencia de la interrupción a la cual esta asociado. Midiendo el intervalo de tiempo desde la ocurrencia de una interrupción determinada y la respuesta del proceso asociado a dicha interrupción se puede tener una idea de la respuesta temporal de nuestro sistema. Cuando este experimento se ejecuta constantemente, imponiéndole al sistema diferentes condiciones de carga, entonces se podrá observar si se mantiene el nivel de respuesta del sistema. Al realizar esta prueba realmente lo que podemos decir es que, en el caso de que así sea, nuestro sistema no varía su desempeño ante las cargas bajo las cuales las pruebas se realizaron, quizá bajo otras condiciones si lo haga. Para una comprobación exhaustiva se necesita un análisis minucioso de todo el S.O. de manera que encontremos la combinación de ejecución de secciones de código que produzca el peor de los casos. Este análisis es prácticamente imposible, debido a lo complejidad inherente a un S.O..

2.1.4 Espacio en memoria.

Los criterios analizados hasta el momento se refieren al desempeño con respecto a respuestas temporales de nuestro sistema. Otro de los criterios importantes es cuánto espacio en memoria necesita el sistema en análisis.

En este aspecto las opciones muchas veces se reducen a estimar cuan fácil se adapta el sistema en estudio a las necesidades de la implementación. Por ejemplo cometer las siguientes acciones:

- Si las estructuras de datos son más grandes que lo necesario, estas pueden ser modificadas en orden de reducir su tamaño.
- Si existen secciones de código que no se utilizarán, estas pueden ser eliminadas, a través de algún menú de configuración por ejemplo.

Algunos sistemas son altamente configurables, permiten eliminar casi todas las funcionalidades que se puedan considerar innecesarias. En ese caso están por ejemplo las arquitecturas de micronúcleo como RTLinux, donde los servicios están organizados en módulos separados los cuales se cargan si son necesarios. Unas de las pruebas que se pueden realizar es la de configurar el sistema más pequeño y el más grande que se pueda y de esta forma tener una idea de la flexibilidad de nuestro sistema.

2.2 Realizar las pruebas necesarias

El aspecto principal a la hora de medir el desempeño de un sistema es establecer los objetivos que se pretenden alcanzar en tal medición. No existe métrica alguna aplicable a todas las posibles aplicaciones, existen sistemas que pueden ser muy eficientes para la implementación de un tipo de aplicaciones (Estructuras de multi procesamiento orientadas a la velocidad de la ejecución de las aplicaciones que sobre ella se implementen) y por el contrario tener un desempeño muy pobre para otras implementaciones (Control en tiempo real, etc.). La idea principal a la hora de confeccionar las pruebas será, lo que necesitamos que nuestro sistema realice.

Hasta ahora hemos descrito cuatro criterios a tener en cuenta al realizar las pruebas de desempeño: rendimiento (ancho de banda), nivel de respuesta del sistema (tasa de transacciones del sistema), determinismo (peor de los casos de una transacción en el sistema) y espacio en memoria). En cada uno de esos aspectos se establecerán determinados requisitos para el sistema.

2.3 Factores fundamentales para los sistemas de tiempo real crítico

En los sistemas de tiempo real críticos un proceso al cual se le retrasa su ejecución pudiera no ejecutarse nunca, los criterios de determinismo y nivel de respuesta del sistema son fundamentales. Esto no significa que los demás criterios no sean importantes, si lo son, pero los dos primeros hablan de situaciones que puede poner en peligro real el funcionamiento del sistema. Los demás factores responden si con una plataforma dada somos capaces o no implementar el sistema. Por ejemplo si el procesador escogido es capaz de procesar nuestras n tareas, o la memoria física existente es capaz de contener nuestra mayor aplicación sin hacer intercambios con la memoria virtual.

2.4 Factores fundamentales para los sistemas de tiempo real acríticos.

A diferencia de lo analizado anteriormente, para los sistemas de tiempo real no críticos así como para los sistemas que no son de tiempo real del todo, el parámetro más relevante a tener en cuenta es el rendimiento en el procesamiento. Esto es debido, sobre todo, a la máxima que habla de que a más velocidad mejor sistema.

2.5 El factor espacio en memoria.

Este factor es importante para casi todos los sistemas sin importar los requisitos temporales del mismo. Ejemplo de ellos son los sistemas empotrados muy utilizados hoy en día. Al realizar un diseño de las aplicaciones para estos sistemas, incluidos sus sistemas operativos, un análisis del espacio que ocuparan en memoria es vital. Por otra parte a los sistemas de tiempo real crítico les interesa que no exista intercambio con el disco duro y se requiere que la aplicación se ejecute completamente desde memoria principal. De esta manera se nos revela la importancia de este factor para todos los sistemas a implementar. Quizás este sea uno de los criterios, en algunos casos, más fáciles de delimitar. Muchas veces sólo se necesitará delimitar qué componentes nos hace falta del S.O., lo cuál es el tamaño del S.O. que nos proporcione tales servicios. Y la decisión final es simple nuestro sistema se ajusta o no a los recursos disponibles.

2.6 Métricas para el análisis de sistemas de tiempo real.

La mayoría de las métricas de lo sistemas existentes hablan sobre el rendimiento del mismo, a pesar de ello existen referencias a importantes métricas para los sistemas de tiempo real, la latencia en la interrupción y la latencia en la planificación. No obstante se debe aclarar que el nivel de respuesta y el determinismo de un sistema en estudio no pueden ser medidos en una forma estándar. Para ello se necesita realizar un análisis que tenga en cuenta el hardware sobre el cual se ejecuta el sistema, entre otras cosas. Aunque en este trabajo no se hará referencia a todas las pruebas solo algunas básicas no está de más mencionar algunas representativas.

Métricas importantes para los sistemas de tiempo real:

- Planificación.
- Temporización.
- Señales.
- Objetos de sincronismo.
- Intercambio de mensajes.

- Nivel de respuesta del sistema y determinismo.

2.7 Planificación.

Como todos saben la planificación es una de los servicios esenciales dentro del S.O. pues es el planificador quien se encarga de darle prioridad a las tareas que llegan a el.

En un sistema operativo la planificación es la solución al fenómeno de la concurrencia o a cuando por ejemplo múltiples unidades funcionales necesitan acceder al mismo objeto. La planificación es usada por el S.O. para permitir a cada unidad compartir el objeto en el tiempo. Es aplicada también en varias partes de un sistema para el tiempo del procesador, comunicación, acceso al disco, etc...Actualmente la planificación es ejecutada en base a criterios tales como prioridad, latencia, requerimientos de tiempo , etc.(Yung-Hsiang 2001)La mayoría de los sistemas invierten solo una fracción de tiempo ejecutando una computación útil y el resto del tiempo se encuentra desocupado EL administrador de energía del S.O. debe seguir la pista de los períodos de computación ,así cuando se haya entrado en un periodo de descanso, este puede inmediatamente apagar la mayoría de las partes del sistema que no se este utilizando(Ravinda 204). Para que la planificación sea más eficiente el S.O. debe conocer los requerimientos del tanto de tiempo como de los dispositivos que van a usar las aplicaciones de antemano.

La teoría de planificación proporciona diversas soluciones para conseguir un comportamiento temporal predecible junto a un alto nivel de utilización de la CPU. Una de las soluciones más simples y mejor conocidas es la planificación expulsora (o desalojante) por prioridades fijas. En esta política cada hilo tiene una prioridad asignada, y el planificador elige para ejecución aquel hilo que tiene mayor prioridad, de entre los que están listos para ejecutarse. (Harbour 1996)

2.7.1 Latencia de la planificación.

Objetivo de la prueba: El objetivo de esta prueba es la de determinar el tiempo en el cual un hilo, que está listo para ejecutarse, es capaz de tomar el procesador o sea que tiempo demora en ejecutarse de forma general.

2.7.2 Prueba del servicio de planificación.

- **Pruebas el parámetro cambio de latencia de la planificación.**

Esta es una de las métricas fundamentales a tener en cuenta para el análisis de los servicios de tiempo real(Burns 1990). Con ella se responde una de las preguntas principales de la planificación, cuánto es la demora que existe desde el momento que una tarea debe estar ejecutándose hasta el momento en que realmente se empieza a ejecutar. Para realizar esta medición existen varios escenarios

Escenario 1:

Se implementa un hilo periódico con la más alta prioridad del sistema, al implementarlo con las más alta prioridad evitamos que se produzcan interferencias por pare de otros hilos del sistema en el experimento que llevamos a cabo. En el cuerpo del hilo en cuestión se invoca una función de espera, (sleep() , nanosleep()) de manera que el hilo realiza una espera por un tiempo determinado, entonces se determina de cuanto fue realmente esta espera la diferencia será la latencia buscada.

Deficiencias de esta prueba: Si bien esta prueba nos ofrece una idea de la latencia de la planificación, tiene la falta de que la latencia encontrada es la suma de la latencia de varios componentes. Una de las más importantes es la latencia de la temporización que hayamos utilizado y otra las ejecuciones que puedan existir de secciones no expropiables del núcleo de Linux. Esta última se elimina al utilizar versiones superiores a la 2.6.18.

2.8 Temporización.

Antes que nada debemos dedicarle un pensamiento a dos términos similares pero no iguales:

Reloj (clock).

Dispositivo que registra el paso del tiempo. Las operaciones que acepta son: leer/escribir la hora y obtener sus características (resolución, precisión, etc.). Ver anexo 1.

Temporizador (timer).

Dispositivo que produce eventos relacionados con el paso del tiempo a los que se les puede asociar acciones. Las operaciones que acepta son: Programar periodo, asociar función, consultar estado del contador, etc.

- POSIX define operaciones para trabajar con relojes y temporizadores. RTLinux sólo implementa los relojes POSIX, pero es posible realizar temporizaciones, pero con funciones no estándar.

2.8 1 Pruebas del servicio de temporizacion.

- **Pruebas para estimar el tiempo de pedirle el tiempo al sistema.**

El objetivo de esta prueba es la estimar cuanto se demora en preguntar el tiempo al sistema.

2.8.2 Prueba del servicio de temporización.

El acceso al sistema de temporización del sistema es una de las funcionalidades verdaderamente críticas. Entre las mediciones más importantes están la resolución de los relojes del sistema que se puedan utilizar y la exactitud de los temporizadores.

2.8.3 Pruebas para estimar el tiempo al realizar la solicitud del tiempo al sistema.

Esta medición impacta las demás de alguna manera, claro si el experimento que se lleva a cabo tiene una larga duración la demora de pedirle el tiempo al sistema puede ser despreciable no así en experimentos de corta duración. El objetivo de la prueba es estimar cuanto se demora en preguntar el tiempo al sistema.

Escenario 2:

Se realiza un ciclo donde se realizan dos peticiones sucesivas del tiempo al sistema.

Se determina el peor de los casos y el caso promedio en las mediciones realizadas.

2.9 Señales.

Pruebas del servicio de señales.

De forma general las señales son interrupciones de software a diferencia de las generadas por dispositivos externos, denominadas de hardware. Un parámetro interesante a medir en ellas es su latencia, es decir el tiempo desde que son generadas hasta comienza a ejecutarse la primera instrucción asociada con el manejo de la misma. En todas las mediciones se busca el cálculo del peor de los casos.

2.9.1 Prueba donde se mide la latencia en el envío de una señal generada por el sistema a un hilo.

El objetivo de esta prueba es medir el tiempo que se demora en capturar y manejar una señal enviada por el sistema a un hilo.

2.9.2 Prueba donde se mide la latencia en el envío de una señal de un hilo (proceso) a otro hilo.

El objetivo de esta prueba es la de medir el tiempo que se demora en capturar y manejar una señal enviada por un hilo a otro hilo.

2.9.3 Prueba donde se mide la latencia en el envío de una señal de un hilo a otro hilo.

Escenario 3: Señal enviada a otro hilo (proceso).

Aquí se mide, además, la sobrecarga al traspasar los límites del hilo o proceso que envía la señal.

Se debe tener en cuenta que el hilo (proceso) que envía la señal debe de ser de menor prioridad que el que la recibe.

2.9.4 Prueba donde se mide la latencia en el envío de una señal generada por el sistema a un hilo.

Para la medición de este parámetro se tiene el siguiente escenario.

Escenario 4:

Se implementa un hilo de alta prioridad.

El hilo implementa un temporizador para que expire en un momento dado (tiempo absoluto) y que envíe una señal dada.

Se lee la hora en cual fue atendida la señal programada.

Se calcula la diferencia de tiempo entre la hora de expiración de temporizador y la hora leída en el manejador de la señal.

CAPÍTULO 3. RESULTADOS Y DISCUSIÓN

A continuación veremos los resultados de las pruebas que se le realizaron al núcleo 2.6.24 del GNU\Linux tanto en su versión estándar así como para la personalizada. Los resultados de las mismas al ser una serie de valores numéricos se plotearon en una gráfica de Tiempo vs. Cantidad de veces que se probó cada parámetro que este caso serían un total de 500. Además los valores resultantes serán comparados con otros que ya se obtuvieron de otra versión que fue probada con anterioridad.

3.1 Análisis de resultados planificación.

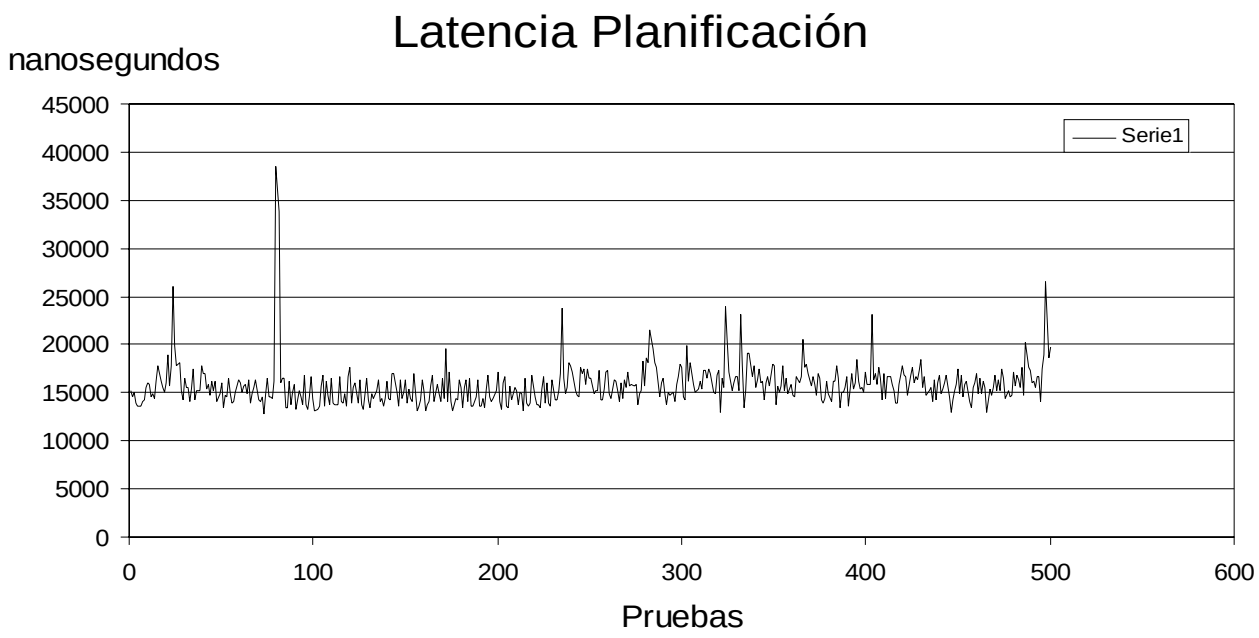
Empezaremos por el análisis de resultados obtenidos por la prueba de planificación realizada al S.O. en el caso de la versión estándar.

PC UCLV:

Condiciones: núcleo 2.6.24 en estado estándar.

Obteniéndose un paquete de quinientos resultados numéricos en cuanto a tiempo de respuesta muestreados en formato (.dat) y llevándose a una gráfica para un ploteo de Cantidad de pruebas vs. Tiempo de respuesta obtenemos que:

Figura 3.1. Casos extremos de latencia de planificación.



Donde los valores limites serán:

Tabla 3.1 Casos extremos planificación.

Peor caso	mejor caso	caso promedio
38528	12722	15791,936

Como se puede ver da como resultado una latencia de la planificación bastante aceptable, esta es una de las métricas más importantes a la hora de considerar el desarrollo de aplicaciones con altas restricciones temporales. En este caso se dice que es buena por el hecho de que la varianza entre un resultado y otro sea pequeña que hace que casi todas las respuestas estén en el mismo orden o sea que las respuestas serán pocas veces más demoradas.

3.1.1 Planificación en versión personalizada.

Veremos entonces que sucederá en el caso de que sea activado el parche de baja latencia.

PC UCLV:

Condiciones: núcleo 2.6.24 con parche activado

Figura 3.2 Casos extremos de latencia de planificación personalizada.

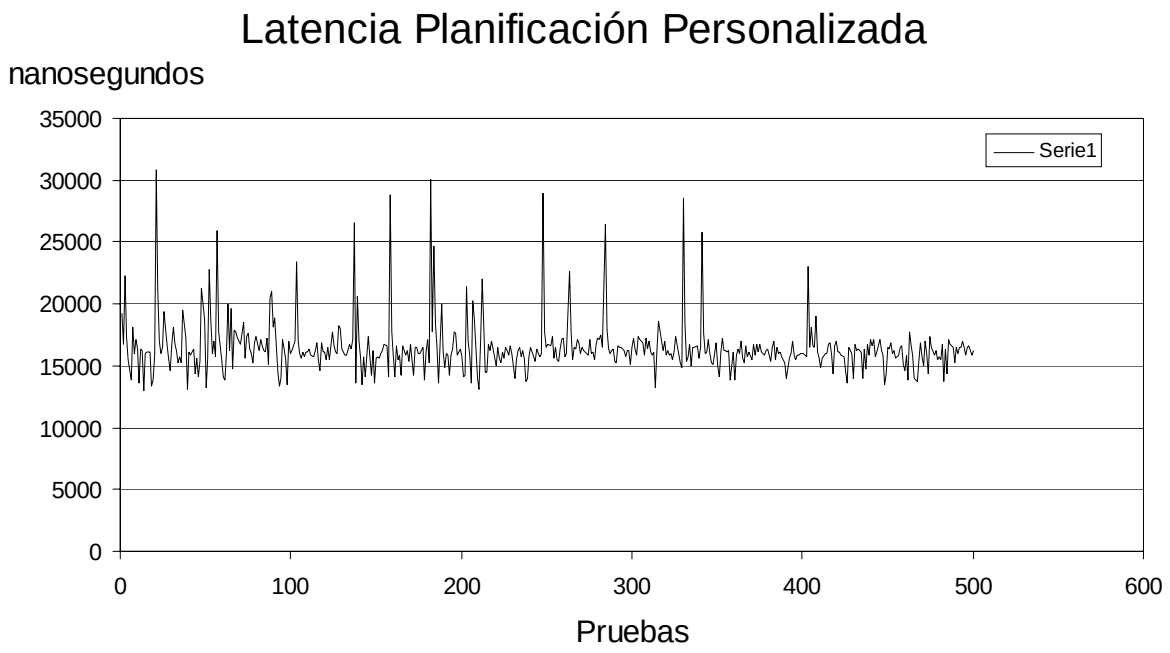


Tabla 3.2 Casos extremos latencia planificación versión personalizada.

Peor caso	mejor caso	caso promedio
30899	**12939**	**16459,886**

Aquí ya vemos que difiere un poco de la otra versión, por el hecho de que va a existir una variación un poco mayor que en la otra, pero bueno no hay casos extremos que indiquen la existencia de valores que puedan afectar un proceso ni largas demoras del mismo.

3.2 Temporización.

Ahora veremos la respuesta dada por la prueba de llamada del sistema para la versión estándar.

PC UCLV:

Condiciones: núcleo 2.6.24 en estado estándar.

Figura 3.3 Llamada al sistema.

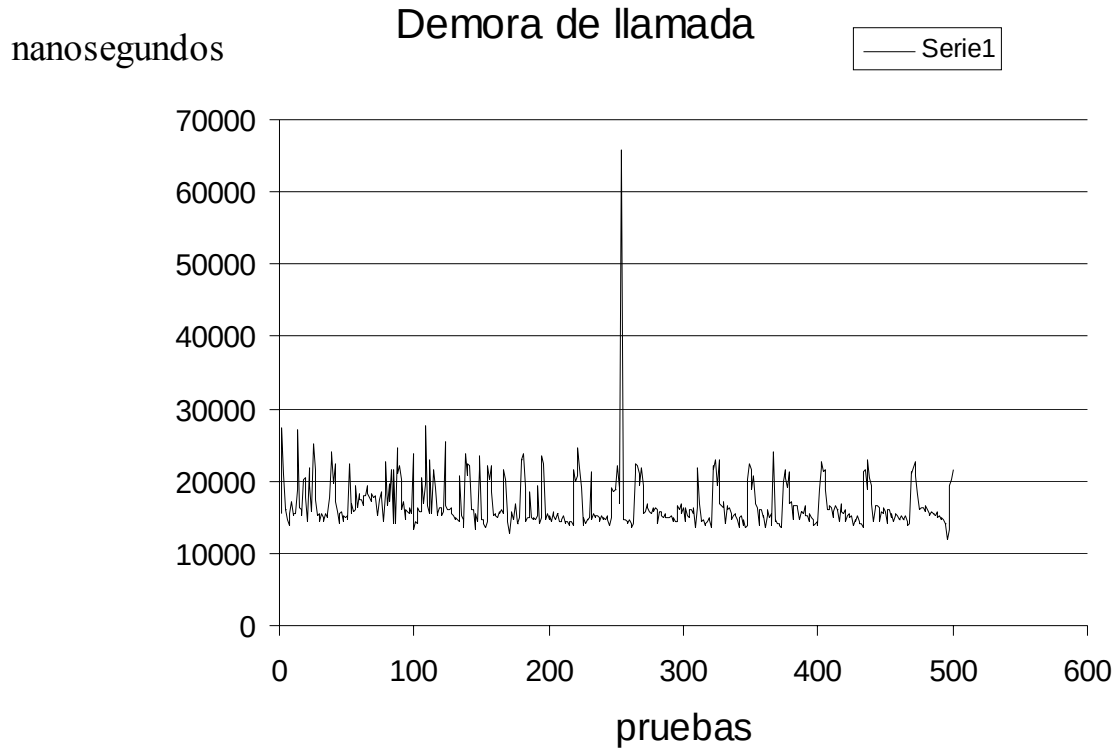


Tabla 3.3 Casos extremos demora de llamada.

Peor caso	mejor caso	caso promedio
65855	11999	16787,154

En esta prueba los resultados son buenos en lo que se refiere a la velocidad de la respuesta no así la robustez de la respuesta debido a la varianza entre una respuesta y otra.

3.2.1 Temporización en versión personalizada.

Al activar el parche de baja latencia obtendríamos que:

PC UCLV:

Condiciones: núcleo 2.6.24 parche activado

Figura 3.4 Llamada al sistema.

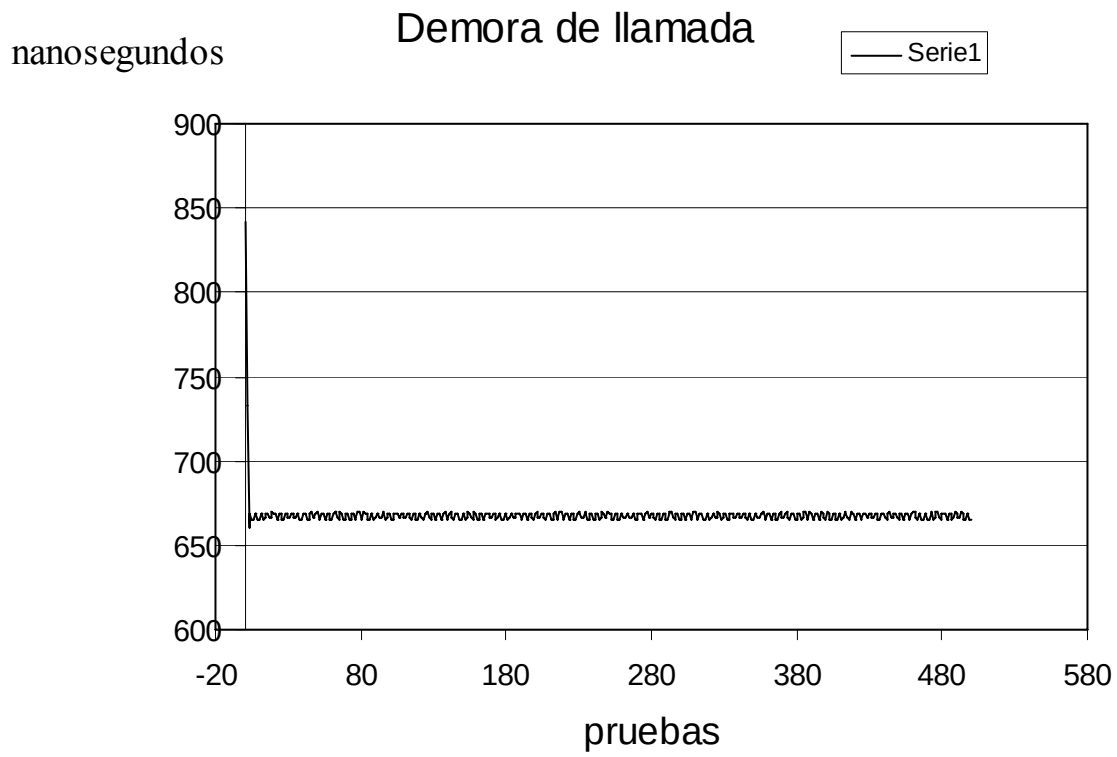


Tabla 3.4 Casos extremos demora de llamada en versión personalizada.

Peor caso	mejor caso	caso promedio
842	661	667,832

Vemos que el cambio de la temporización aumentó considerablemente con la activación del parche de baja latencia y que hay una variación extremadamente pequeña entre un resultado y otro demostrando esto la rapidez con la que puede esta versión hacerse cargo de la temporización.

3.3 Señales.

Para ver la parte de señales empezaremos con una prueba de rapidez en manejo de señales un caso específico: Medir el tiempo que se demora en capturar y manejar una señal enviada por el sistema a un hilo.

PC UCLV:

Condiciones: núcleo 2.6.24 en estado estándar.

Figura 3.5 Casos de manejo de señal del sistema a un hilo.

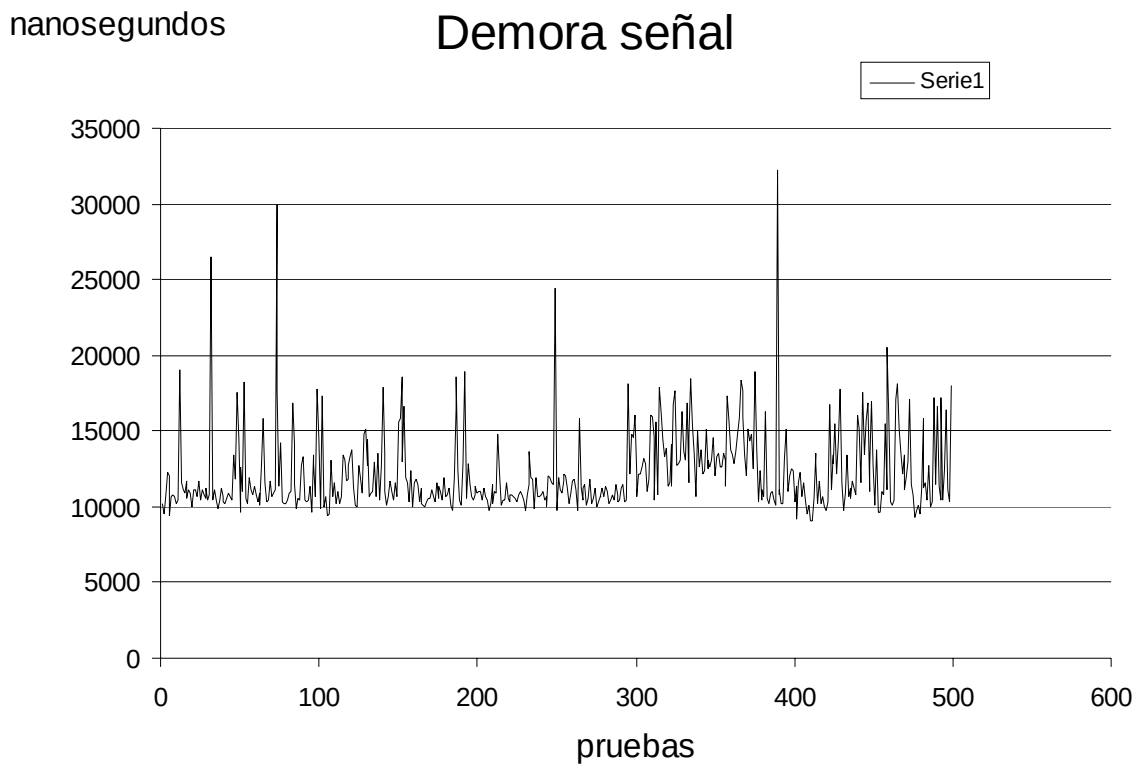


Tabla 3.5 Casos extremos demora de sistema-hilo.

Peor caso	mejor caso	caso promedio
32225	9056	12183,926

En esta prueba se obtiene que al utilizar 2.6.24 los resultados denotan rapidez en el sistema. Notar que en esta prueba se hace uso de los temporizadores del sistema y los resultados son rápidos aunque claro teniendo en cuenta que no existió bloqueo de la señal.

3.3.1 Señal sistema-hilo personalizada.

Aquí volveremos a la parte donde el parche de tiempo real se encuentra activado y para este caso se medirá el tiempo que tarda en tomarse una señal que fue activada y manejar la misma o sea la misma prueba que la anterior pero ya aquí con el parche de baja latencia activado.

PC UCLV:

Condiciones: núcleo 2.6.24 con parche activado.

Figura 3.6 Casos de manejo de señal del sistema a un hilo versión personalizada.

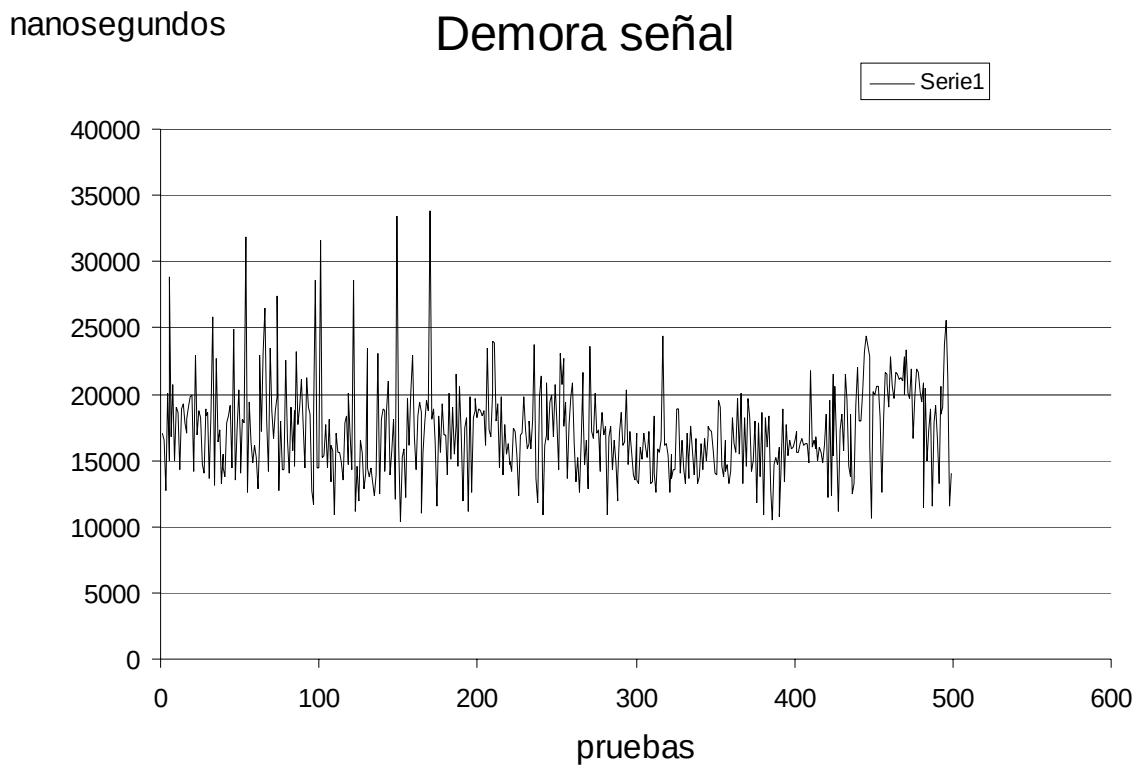


Tabla 3.6 Casos extremos demora de sistema-hilo personalizada.

Peor caso	mejor caso	caso promedio
33871	10368	17156,874

Ha aumentado la varianza para este caso considerablemente, fíjense que el ancho que poseía el gráfico ha aumentado en un buen por ciento pero claro no hablamos todavía de valores que no sean manejables, cuando nos referimos a valores que no sean manejables nos referimos a aquellos que son del orden de segundos. Tenemos que recordar que todos aquellos valores que no son viables para tiempo real son los que pudieran comprometer el sistema y como el rango de respuestas en este caso no es de una amplitud considerablemente grande podemos afirmar que no estamos en presencia de un mal resultado.

3.3.2 Manejo señal de un hilo a otro hilo.

Ahora comprobaremos en esta versión otra forma de ver el manejo de señales y para la siguiente prueba tenemos como objetivo: Medir el tiempo que se demora en capturar y manejar una señal enviada por un hilo a otro hilo.

PC UCLV:
Condiciones: núcleo 2.6.24 en estado estándar.

Figura 3.7 Casos de manejo de señal de un hilo a otro hilo.

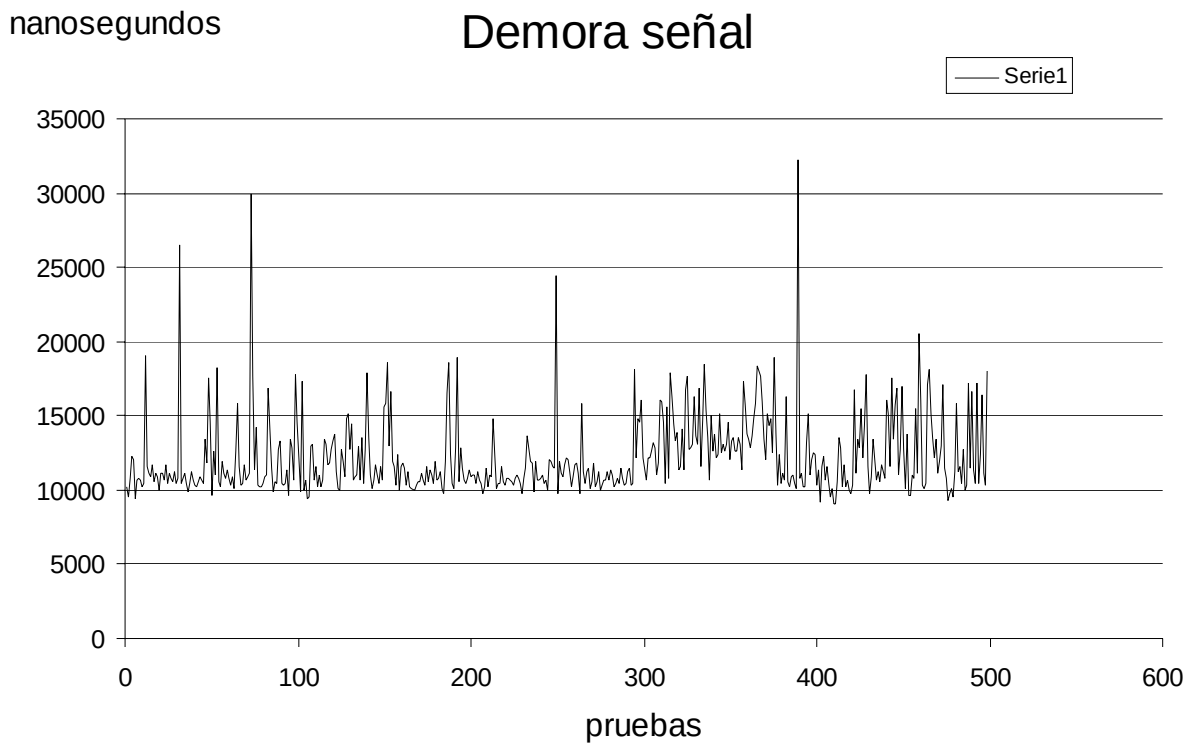


Tabla 3.7 Casos extremos demora de hilo-hilo.

Peor caso	mejor caso	caso promedio
32225	9056	12183,926

En esta prueba las respuestas obtenidas son aceptables en cuanto a un rango de tiempo pequeño teniendo en cuenta que la varianza entre los resultados no es de lo mejor pero no llegan a un estado crítico.

3.3.3 Señal hilo-hilo personalizada.

Los resultados de esta prueba de transmisión de un hilo otro pero ya en el caso personalizado o sea con el parche de baja latencia activado que daría de la siguiente forma.

PC UCLV:

Condiciones: núcleo 2.6.24 con parche activado.

Figura 3.8 Manejo de señal de tipo hilo-hilo.

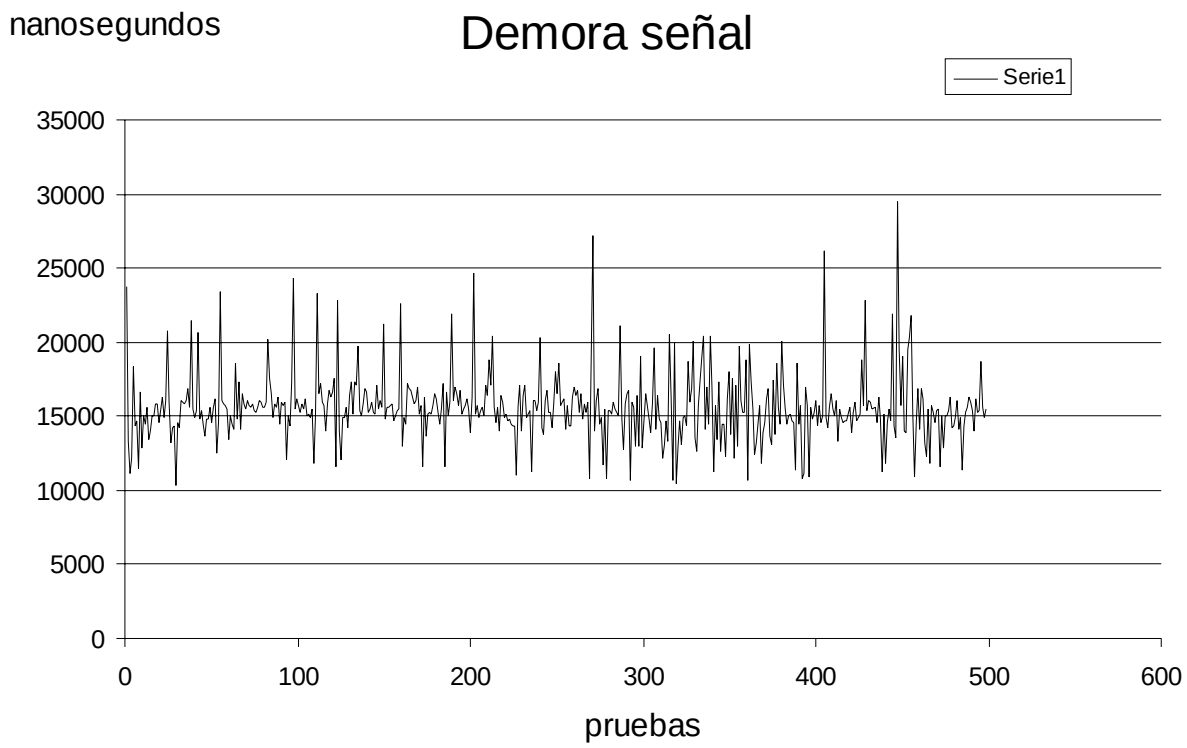


Tabla 3.8 Casos extremos demora de hilo-hilo personalizada.

Peor caso	mejor caso	caso promedio
29451	10380	15579,068

Aquí aumentará la separación entre una respuesta en el caso personalizado pero no son valores extremos por lo que se puede asumir que estamos en presencia de un resultado aceptable.

3.3 Análisis general de los resultados y comparación con otras versiones.

Ahora veremos que tal se vería esta versión comparada con una de sus antecesoras, la 2.6.18 en sus versiones estándar y con el parche de tiempo real activado. Debemos tener en cuenta a

la hora de comparar que la versión con la que estamos llevando a cabo esta comparación fue probada para aplicaciones de tiempo real. Aquí tendremos que los tiempos de respuestas son en nanosegundos y que los tiempos de estamos hablando son de los promediantes obtenido en las pruebas realizadas.

Además en el caso de la latencia de planificación, que como ya fue explicada antes es una de las métricas mas importantes cuando nos referimos a sistemas aplicables a tiempo real es abismal el cambio dada la rapidez de respuesta de la versión 2.6.24 tanto para la estándar así como para la personalizada .

También tenemos que para ver el manejo de señales en algunos casos tenemos que no serán en su totalidad mejores los resultados pero son del todo comparables. Mas sin embargo en el caso de la temporización si es más rápida la versión 2.6.18 que la 2.6.24 pero no con ello estamos diciendo que no sean buenos los resultado del 2.6.24 pero bueno si que los del 2.6.18 son excelentes.

Tabla 3.9. Análisis y comparación con otras versiones.

	Versión 2.6.24 estándar	**Versión 2.6.24 personalizada Parche activado**	**Versión 2.6.18 estándar**	**Versión 2.6.18 personalizada Parche activado**
Latencia de planificación (nanosegundos)	**15791,936**	**16459,886**	**5985350.18**	**42151.79**
Petición de tiempo al sistema (nanosegundos)	**16787,154**	**667,832**	**541.94**	**1676.19**
Tiempo de señal sistema-hilo (nanosegundos)	**12183,926**	**17156,874**	**5821495.24**	**45644.87**
Tiempo de señal hilo-hilo (nanosegundos)	**12183,926**	**15579,068**	**13065.4**	**18989.53**

3.4 Análisis económico.

Al utilizar herramientas del software las cuales han sido implementadas siguiendo el paradigma de software libre se obtienen numerosas ventajas como la transferencia de tecnología de punta , ayuda al desarrollo local, así como otros.

Hoy en día aparejado con la problemática de la solución de los problemas de tiempo real está la financiera, los costos de las distintas versiones de sistemas operativos y sus herramientas se han disparado llevando consigo que la mayoría de las veces sea más importante el análisis del costo de una aplicación que la propia búsqueda de la fiabilidad y rapidez del la misma.

Las pruebas que se realizaron en este capítulo señalan la posibilidad de aplicación de la versión 2.6.24 del kernel de Linux en sistemas de tiempo real y la adquisición de la misma es gratis. ¿Saben que significación tendría esto para muchos?:

Que se podría buscar la eficiencia de todo proceso de tiempo real sin costo alguno.

Para hacer una idea el Windows CE es uno de los sistemas operativo con aplicaciones de tiempo real que se encuentra entre los más utilizados y solo las herramientas de desarrollo y licencia de concecidad tienen un costo de 1014 USD. Otro ejemplo pudiera ser la versión 6.2 del Qnx Neutrino cuyo costo con todas las herramientas se encuentra valorado en 8000 USD y la versión 6.4 solo la licencia tiene un costo de 1655 USD.

De forma general las ventajas que aporta la utilización del 2.6.24 del núcleo de Linux son claras en cuanto a ahorrar se refiere si hacemos una comparación con los costos de los sistemas antes mencionados.

3.5 Conclusiones del capitulo

La versión 2.6.24 del núcleo de Linux arrojó resultados que satisfacen la problemática que planteábamos al principio sobre la posibilidad de la aplicación de esta versión en sistemas de tiempo real.

El análisis de las gráficas nos presenta un sistema que responde con una variación bien pequeña entre respuestas a una misma prueba, lo que provocaría en cualquier proceso al que sea aplicado que los períodos estimados de respuestas del sistema sean cortos haciéndolo un sistema más rápido y predecible en cuanto a tiempo respuesta.

El sistema proporciona una planificación que hace gala de rapidez y organización de los procesos e interrupciones del sistema incluso comparable con otras versiones como la 2.6.18 tanto para la versión estándar así como para la personalizada, como llamaríamos a

esta versión donde está activado el parche de tiempo real. Lo mismo se puede decir del manejo de señales, donde la varianza es muy pequeña entre los distintos resultados a una prueba. Además los resultados a la prueba de temporización aportan una gráfica que no muestra variación prácticamente entre ellos. O sea estamos en presencia de un sistema que se aplica a los requerimientos de tiempo real por ser sus métricas comprables con otros de este tipo y que la utilización del mismo aportaría grandes ventajas tanto económica como en busca de mejoras de procesos de tiempo real haciéndolo más baratos, rápidos y fiables.

.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones.

Al término de esta investigación se puede concluir que el sistema operativo GNU\Linux dentro de la evolución de su medio ha traído importantes avances en su diseño e implementación para el uso como plataforma para el desarrollo de tiempo real.

A partir de la introducción del parche de tiempo real empieza un soporte para aplicaciones de tiempo real duro con la disminución de parámetros como la latencia de las interrupciones y que ya en la versión 2.6.24, que se analizó en este trabajo, se puede decir que ha llegado a un alto grado de madurez al estar incluidos como parte del núcleo muchas de las características antes mencionadas con la introducción del parche.

El sistema operativo en su versión 2.6.24 del núcleo de GNU\Linux indicó que su aplicación en sistemas de tiempo real es un hecho. El sistema demostró con una latencia de planificación aceptable que puede ser un sistema determinístico alejando de la incertidumbre los procesos a los que sea aplicable. Otro factor a tener en cuenta que se analizó en la aprobación de este sistemas para aplicaciones de tiempo real fue el costo del producto cuya adquicisión es completamente gratis y que solo esto comparado con el costo de inversión en licencias y herramientas para otros sistemas operativos se vuelve más que recomendable para su uso.

El manejo de señales y la temporización del sistema son en su generalidad aceptables para el objetivo que se persigue, encontrándose los resultados dentro del rango de valores que hacen que un proceso sea rápido y que la poca varianza entre los mismos provoca una estabilidad en cuanto a plazos de entrega siendo esto unos de los parámetros que necesitábamos para un sistema de tiempo real.

Como se pudo apreciar en las métricas más significativas la versión 2.6.24 del núcleo de Linux, tanto con el parche de tiempo real y sin él, representa una buena solución a la hora de ser utilizado como plataforma para el desarrollo de aplicaciones con altas restricciones temporales.

Recomendaciones.

Como se conoce hoy en día deberíamos hacer hincapíe en la migración hacia sofware libre en nuestro país, y que el conocimiento de los valores de las métricas fundamentales de cada sistema sean conocido por todos de forma tal que se puedan comparar los sistemas en cuanto a gama de aplicaciones teniendo en cuenta sus métricas por lo que se recomienda que :

- Se debe seguir investigando en cuanto métricas de sistemas para conocimiento de sus grado de aplicaciones para esta versión.
- Continuar estudiando versiones de software libre.
- Seguir analizando las posibilidades de aplicaciones de software libre.
- Hacer enfasis en el uso del Linux en aplicaciones de tiempo real.
- La principal recomendación que aporta este trabajo es la de no perder de vista la introducción de este sistema en tiempo real.

Anexo I Relojes

El estándar POSIX especifica la posibilidad de disponer de varios relojes. Cualquier implementación ha de disponer al menos del reloj CLOCK_REALTIME.

- RTLinux dispone de tres relojes lógicos: `CLOCK_REALTIME, CLOCK_MONOTONIC` y `CLOCK_RTL_SCHED` ;y dos físicos: `CLOCK_8254` y `CLOCK_APIC`.Durante la configuración previa a la compilación de RTLinux se puede elegir entre dos resoluciones de reloj: tickso del 8254 o nanosegundos.

- La gestión del tiempo está definida en el estándar 1003.1a (Extensiones de tiempo real) y no en el estándar que define los PThreads (1003.1b).

`CLOCK_REALTIME`

Reloj del sistema. Si nuestro sistema soporta la instrucción ensamblador `rdtsc` (instrucción disponible a partir del procesador Pentium®) entonces se utiliza esta instrucción para obtener la hora (con esta fuente de tiempos se obtiene una resolución próxima al nanosegundo), en caso contrario se obtiene del chip 8254 alimentado a 1193180Hz (0.83 microsegundos). Este reloj puede sufrir ajustes para corregir la fecha.

`CLOCK_MONOTONIC`

Igual que CLOCK_REALTIME pero no se realizan ajustes, por tanto su cuenta es creciente sin saltos bruscos. Útil para medir duraciones de eventos.

Anexo II POSIX Timers Interface

La forma de usar temporizadores POSIX en Linux es usando señales, y estos no están directamente ligados al hilo que los crea, además su implementación requiere saber qué señal será la enviada cuando expire el temporizador, para hacer más natural el uso de los temporizadores Posix se crea una nueva interfaz basada en estos mismos temporizadores POSIX, que brinda la facilidad de saber que hilo creó el temporizador(o cualquier dato que se desee hacer persistente fuera del temporizador) y encapsular la señal enviada, solo se necesitan saber los tiempos de activación y expiración y la función a ejecutar cuando expire el temporizador.

Esta capa usa solamente las señales de tiempo real (33-64), con el objetivo de no permitir interferir con el uso del sistema, sus mecanismos de notificación o malinterpretar un mensaje del sistema con una de un temporizador.

scada_timer arm_timer(time_scada asec, time_scada ansec, time_scada isec,
time_scada insec, signal_scada signal, signal_handler handler);

Crea un timer y retorna su identificador.

asec y ansec: tiempo de activación del hilo en segundos y nanosegundos para mayor precisión.

isec y insec: tiempo de expiración del hilo en segundos y nanosegundos para mayor precisión.

signal: mediante qué señal se le notificará al proceso que es tiempo;

handler: función a llamar cuando llegue la señal

scada_timer program_timer(time_scada asec, time_scada ansec, time_scada isec,

time_scada insec, signal_handler handler);

Igual que la anterior, pero no necesita conocer que señal usar, el solo selecciona una disponible, en este caso retorna el temporizador, en otro caso un scada timer con sus campos = -1.

void scada_timer_delete(scada_timer timer);

Elimina el temporizador timer del sistema.

pthread_t retrieve_th_pt(signal_scada signal);

Dada la señal identifica que hilo tiene asociado el temporizador identificado por dicha señal, retorna el id de dicho hilo (en teoría se puede retornar cualquier dato guardado en el momento de creación del temporizador).

Referencias

Autores. "Sistemas Operativos I." from http://www.monografias.com/trabajos16/novell-cuatro-x/novell-cuatro-x.shtml.

Autores. (2007, 05\12\2007). "Lo que traerá el Kernel 2.6.24." from www.vivabOrg.com.

Autores. (2008). "Núcleo Monolítico." from www.wiki/GNU/Linux.com.

Burns, N. A. a. A. (1990). "Real-Time Systems Scheduling."

Corporation, N. I. (2007). "Desarrollo de Sistemas de Adquisición de Datos y Control de Tiempo real con Tecnologías Estándar." from http://zone.ni.com/devzone/fn/p/sn/n23:8.

García, R. G. (2001). "Sistema Operativo de Tiempo Real para Aplicaciones Espaciales."

Harbour, M. G. (1996). "Tostadores y Posix."

Liu, J. W. S. (2000). "REAL-TIME SYSTEMs."

Locke, C. D. (1992). "Software architectures for hard real-time applications." Real-Time Systems.

Pacheco, A. (2008). "Administración de Procesos (Sistemas Operativos1)
".

Pranevich, J. (2003). "El Maravilloso Mundo de Linux 2.6." from http://www.escomposlinux.org/wwol26/wwol26.html.

Ravinda, J. (204). "Dinamic voltage scaling for systemwide energy minimization in real-time embedded systems." ISLPED 2004: 78-81.

Rey, E. A. (2007). "Hilos y SMP."

Tejedor, J. I. g. (2004). "Planificacion y Gestion de Procesos."

Vela, G. N. N.-L.-J. A. R. (2004). "Planificador del kernel Linux." from charla-kernelv1-0-html.htm.

Yung-Hsiang, L. (2001). "Comparing Sistem Level Power Management Policies." **18**: 10-19.

ZAMARRÓN, D. L. (2004). "Elección de un Sistema Operativo de Tiempo Real." from select.htm.