Conflictos de hilos de Kernel en un sistema operativo didáctico

Hugo Ryckeboer, Nicanor Casas, Graciela De Luca, Martín Cortina,
Gerardo Puyo, Waldo Valiente
{hugor,ncasas,gdeluca,mcortina,wvaliente}@unlam.edu.ar
Balaguer Cristian, del Rio Myrian, Grisolia Maria Laura, Herrera Maite
{balaguercristian,myriam.delrio,laura.grisolia,herrera.mai}@gmail.com

Universidad Nacional de la Matanza
Departamento de Ingeniería e Investigaciones Tecnológicas
Dirección: *Florencio Varela 1703* - Código Postal: 1754 - Teléfono: *4480-8900/8835*

Abstract: The user´s threads implementation needs its compensation through the Kernel threads to complement the necessary structure of any operative system, especially one with didactic features.
The imperious aim in the creation of an operative system is to determine if it can, or not, must, or not, support threads because once generated it can only carry out patches, because, it is very hard to rewrite all the Kernel.
The interspersion of threads users, generated under scheduler activations and/or under POSIX norms, with Kernel threads, having in consideration their characteristics, is what we are showing in this document.
The parameters used at the design and construction for the alternation among threads, taken in both types of developments and the predictably of the threads are made evident in agreement with he fertility and planning capacity.

Keywords: Scheduler Activations, POSIX, Kernel, HTR, wait, signal, primitives, scheduling|

Resumen: La implementación de hilos de usuario necesita su contrapartida a través de los hilos de Kernel para complementar la estructura necesaria de cualquier sistema operativo y en especial uno de características didácticas.
Lo imperioso en la creación de un sistema operativo es determinar si el mismo puede, o no, debe, o no, soportar hilos porque una vez generado solo se pueden realizar parches debido a que, en términos generales, reescribir todo el Kernel es una tarea tediosa. Intercalar los hilos de usuario, generados bajo scheduler activations y/o bajo normas POSIX, con hilos de Kernel, considerando sus características es lo que plasmamos en este documento. Los parámetros utilizados para el diseño y su construcción para la alternancia entre hilos, tomadas en los dos tipos de desarrollos y la predictibilidad de los hilos es puesta de manifiesto en concordancia con su fertilidad y su capacidad de planificación

Palabras Claves: Scheduler Activations, POSIX, Kernel, HTR, espera, señal, primitivas, planificación jerárquica

1. Introducción

En muchos sistemas operativos, que no son en tiempo real, los hilos se han adoptado como modelo de programación. A diferencia del modelo de proceso, un modelo de hilo separa el espacio de direcciones del flujo de ejecución dentro del espacio de direcciones. [01]
Algunos intentos de desarrollar hilos en tiempo real se basan en modificaciones de núcleos, que no trabajan en tiempo real, que soportan hilos, convirtiendo una problemática en una

propiedad clave de los hilos: la predictibilidad. En este caso, el planificador se encarga de brindar la solución integrada de hilos en tiempo real [02], en lugar de que lo haga el Kernel.
Los Hilos de Tiempo Real (HTR) proveen un nivel-usuario capaz de sustituir al kernel en ejecución, siendo administrados mediante sus tiempos de comienzo, prioridades y plazos. Todos los hilos, en particular en un entorno HTR, comparten el mismo espacio de direcciones. Implementan administración de hilos, sincronización, y funciones de comunicación, incluyendo comunicación entre entornos HTR (con diferentes espacios de dirección, posiblemente en diferentes máquinas y diferentes arquitecturas). Cada entorno HTR es designado para ser independiente, excepto por el servicio de pasaje de mensajes.
Un entorno de HTR se caracteriza por ser cooperativo. Ningún hilo debe interferir con los recursos del sistema ya suministrados a otro hilo, considerando que el sistema no tiene excepción de manejo de errores para usos inapropiados de los recursos utilizados por hilos individuales. Las llamadas generadas por los HTR bloquean su propia ejecución pero no la de otros hilos administrables. Sin embargo, las llamadas al sistema que no son de HTR o de bibliotecas externas bloquean totalmente el entorno de los HTR. Estas clases de llamados se hacen sólo con extrema precaución. [03]
Las aplicaciones multiproceso son algo común y la planificación tiene un rol importante.
Las decisiones de la óptima planificación dependerán del estado actual y el comportamiento de la aplicación. Cuando ocurren frecuentes fallos de página, el planificador puede usar muchos segmentos de tiempo o suspender alguno de estos hilos momentáneamente. [04]
Un planificador que implementa una óptima estrategia de planificación debe ser capaz de observar el comportamiento de una aplicación en el sistema realizando el seguimiento de su estado y de las tareas que los hilos cumplirán [05].

2. Diseño integrado de hilos

Para el diseño integrado de paquetes HTR nos apoyamos completamente en la especificación de propiedades claves mediante un lenguaje de programación en tiempo real y un lenguaje de descripción del sistema.
Tomamos en cuenta las principales propiedades de los HTR que son la fertilidad, el desempeño, el rendimiento y su nivel de interacción con los otros hilos.
Cuando un hilo genera otro hilo en tiempo de ejecución se lo considera fértil. Determinar si lo hace dentro de su biblioteca o fuera de ella es una decisión importante para detrminar el futuro de los mismos. Al reconocer un hilo estéril, el kernel realiza ciertas optimizaciones en comparación con su apoyo a los hilos fértiles. Estas comparaciones surgen de saber cuáles son los espacios de direcciones que el mismo va a tener y cuáles son los atributos que se deben tener en cuenta para su planifcación.
Un hilo tiene mayor desempeño cuanto más específico es su objetivo de trabajo. El rendimiento se basa en si el padre es planificado individualmente o en conjunto con todos sus hijos. La interacción se establece entre hilos en tiempo real e hilos existentes en un kernel que no trabaja en tiempo real.
El Kernel utiliza esta información con garantías dinámicas On-Line para asegurar la ejecución del hilo.
El problema se presenta con la predicibilidad en la interacción entre hilos. Se deben determinar cuidadosamente las interacciones entre el tiempo real y el tiempo compartido, ya que pueden generarse pérdidas en los plazos o fechas límite de HTR, si acciones como la creación o planificación de un hilo toman más tiempo de lo esperado. Tales acciones son consideradas predecibles si sus resultados y duraciones se deducen antes de que las mismas se ejecuten. [01]
La solución integrada de hilos en tiempo real que alcanza la predicibilidad se consigue mediante la participación de un lenguaje de programación en tiempo real, las opciones del

kernel durante el arranque y el tiempo de ejecución del kernel. Este paquete co-existe con una garantía basada en el Planificador dinámico on-line.
Entre los beneficios del modelo de hilos se destacan su portabilidad, al contener un número de hilos independiente del número de procesadores disponibles; la facilidad de desarrollo de programas asíncronos; y el rendimiento, que permite cambiar de un hilo a otro en un espacio de direcciones único sin reasignar el espacio de direcciones.

3. Mecanismos de administración de hilos de tiempo real

Los hilos son administrados considerando su planificación, su prioridad y sus plazos, establecidos en el momento de su creación. Un hilo no será planificado si su tiempo de comienzo está estipulado en el futuro. Estos son los hilos más complejos de planificar porque los mismos son generados fuera del tiempo de ejecución inmediata y determina además inconvenientes en la planificación del padre. En más de una ocasión, y debido a deficiencias en sistemas operativos que no contemplan esa estructura, se pide a los programadores que los mismos no sean generados hasta el momento de su uso. Esto se hace para evitar tiempos de Kernel desperdiciados en detrimento de otros procesos o hilos. En caso contrario se planificará, teniendo en cuenta los atributos y el estado de los otros hilos. Si se desea que el hilo inicie inmediatamente, el tiempo de comienzo debe inicializarse a cero. Algunos autores hablan de la estipulación en el pasado es decir inicialización negativa. Si bien en un principio se considero la posibilidad de una inicialización negativa en la última revisión, se decidió darlo como un elemento de error y manejar la cola de listos solo con funciones positivas. Esta disciplina administrativa es la primera en tenerse en cuenta.
El siguiente atributo considerado en los algoritmos de administración es la prioridad, que indica cual de los hilos involucrados tendrá preferencia. En este caso el sistema operativo debe poder determinar cuáles son los valores, o rango de valores que puede administrar una biblioteca de hilos para que no se superpongan con valores propios de los procedimientos correspondientes al propio sistema operativo.
Por último se considera el plazo, es decir, el tiempo esperado para el fin de la tarea. Si la tarea no ha sido terminada en el plazo estipulado, ésta continua corriendo. Considerando dos tareas dentro de un mismo nivel de prioridad, tendrá mayor prioridad aquella que no haya cumplido con su plazo, dado que la otra aún puede cumplir con el suyo. Esto puede provocar algunos errores de concepto porque se parte de la base de que una tarea, siempre se ejecuta hasta el final pero en el caso del tiempo real la demora en una tarea puede provocar la necesidad de que otra tarea reemplace a la que estaba en ejecución y había demorado más de lo previsto. En casos como este es el programador el que decide si la tarea termina, continúa. Como se dijo anteriormente el sistema operativo permite que la misma concluya.
Los atributos administrativos son obtenidos y modificados usando diferentes llamadas al sistema. Un hilo puede acceder a sus propios atributos administrativos o a los de otro hilo utilizando dichas llamadas. Cambiar los atributos de los hilos puede tener efectos inmediatos en la administración.

4. Manejo de hilos de tiempo real

Los hilos son creados utilizando un indicador al nombre del hilo creado por referencia; un indicador a una función que actúa como indicador de entrada para el hilo creado; el tamaño en bytes de la nueva pila de hilos que varia de acuerdo al número y tamaño de las variables locales y los parámetros, y la profundidad de las llamadas a subrutinas; un alias que actúa como identificador del hilo suministrado por el usuario, que puede ser utilizado para

propósitos de debugging y por cualquier número de hilos; información del momento de creación del hilo, transparente al mismo, de cualquier tipo pero casteado a *void**, que puede no declararse si no es requerido en la creación; atributos administrativos del hilo, mencionados anteriormente; y un identificador que diferencia si el hilo creado es de nivel kernel o de nivel usuario considerando que el entorno de los hilos de tiempo real terminará cuando no haya más hilos de nivel usuario, por lo que los hilos del servidor constantes deberían ser creados con nivel kernel (a menos que su servicio sea requerido por otros entornos HTR, en ese caso es recomendable utilizar el nivel usuario) [9].
Un hilo es eliminado utilizando una llamada de terminación, que causa que deje el sistema instantáneamente; o "cayendo en el final", lo que es equivalente a utilizar la llamada de terminación en la última declaración de la subrutina; o siendo eliminado por algún otro hilo, el cual requiere el identificador del hilo a eliminar.
Un entorno HTR termina cuando todos los hilos de nivel usuario terminaron, registrándose la finalización.
Los hilos se sincronizan a través de semáforos.
Los atributos especificados por el usuario al asignar un nuevo semáforo son el nombre del semáforo y un valor inicial que indica cuantas instancias del mismo se requieren, es decir, el número de veces que la primitiva *wait* puede ser llamada antes de que un hilo sea bloqueado, asumiendo que no se harán llamadas de la primitiva *signal*. [10]
La eliminación del semáforo se realiza sólo con su nombre.
Para utilizar los semáforos también se proveen las convencionales primitivas *wait* y *signal* y una forma de obtener el valor actual del semáforo, teniendo en cuenta que se necesita como referencia el nombre del semáforo.
Dado que la memoria es compartida dentro de un entorno HTR, los semáforos pueden fácilmente compartirse entre hilos. El sistema no asegura que los semáforos eliminados no sean utilizados. Para ayudar a detectar tal situación, un intento de eliminar un semáforo en el cual otro hilo está bloqueado resultará en falla. En el caso que un hilo sea eliminado mientras era bloqueado por un semáforo, el estado del semáforo será automáticamente modificado (su valor es incrementado).

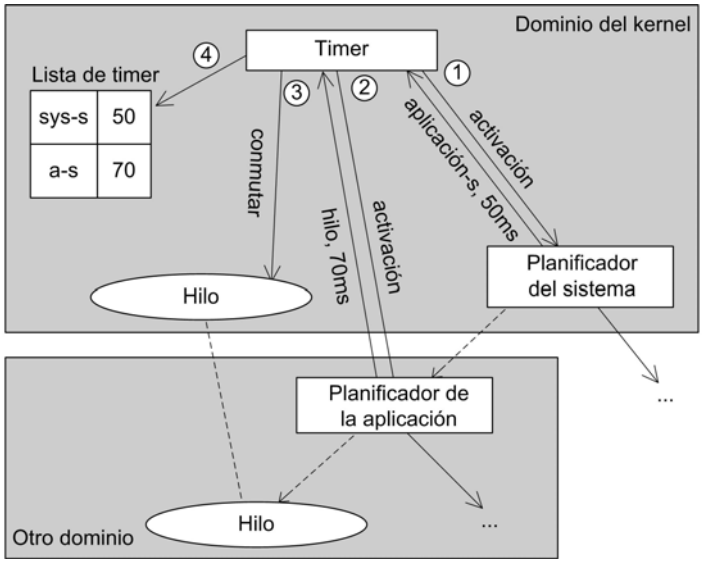
5. Comunicación de hilos

Los hilos de tiempo real proveen envíos/recepciones/respuestas bloqueantes al estilo de primitivas de comunicación entre hilos del mismo espacio de direcciones. Un hilo que envía es bloqueado hasta que el mensaje haya sido recibido y una respuesta haya sido efectuada. Un hilo que recibe es bloqueado hasta que recibe el mensaje. Hay también un testeo no bloqueante que permite a un hilo comprobar si hay un mensaje esperando para que él lo reciba.
Un envío requiere un identificador del hilo que recibirá el mensaje y la longitud y dirección del mensaje a enviar y a recibir.
Una recepción requiere una referencia del hilo que envía; un indicador a un *buffer* dentro del cual el mensaje será recibido y cuya memoria será asignada por la aplicación; y un parámetro de entrada/salida que servirá para indicar la máxima longitud esperada para el mensaje recibido, de ser mayor será truncado.
Una respuesta requiere un identificador de hilo destinatario de la respuesta, un indicador al mensaje respondido y la longitud del mensaje respondido.
Las capacidades descriptas en la sección anterior también son realizadas para permitir comunicación entre hilos en diferentes espacios de dirección (incluyendo en diferentes máquinas). Los identificadores de hilos se pasan a través de espacios de direcciones como valores de retorno de llamadas a procedimientos o en mensajes enviados por una aplicación. Alternativamente, un servicio nombrado puede ser establecido para registrar y buscar el identificador de hilo para hilos remotos. [11]

Cuando un entorno de hilos desea establecer un puerto de comunicación de alguna manera con otro espacio de dirección, la inicialización de la dirección IP y el número de puerto a utilizar es esencial.
Los cruces de comunicación de espacios de direcciones requieren una forma de determinar el identificador de los hilos en otros entornos HTR. Esto se realiza a través de un nombramiento del hilo servidor creado por el usuario, que debe ser único en cada entorno de hilos de tiempo real. El identificador de hilo para el nombramiento de hilo servidor puede determinarse desde otro entorno de hilos de tiempo real, pero requiere que la dirección IP y número de puerto del entorno objetivo sean conocidos.
El nombramiento de hilo servidor mantendrá un nombre de hilo mapeado y servicios requeridos utilizando cruces de comunicación de espacios de direcciones, retornando el identificador de hilo en la respuesta.
Se puede enviar el identificador de hilo en un mensaje entre espacios de direcciones que pueda ser codificado y decodificado utilizando XDR (external data representation). **[01**]

6. Planificación Jerárquica (scheduling). Mecanismos.

Diferentes planificadores de implementaciones de usuario permiten adaptar las estrategias de planificación por hilos de aplicación estructurada en forma de árbol, manteniendo el origen en el kernel por razones de eficiencia y seguridad. Para cambiar entre aplicaciones y subsistemas del kernel se utiliza un planificador. Cada aplicación que necesita, para la implementación de su propia estrategia de planificación, crea instancias de un planificador en su espacio de direcciones. Los hilos y planificadores son administrados como otro planificador. En el espacio de usuario se localizan la aplicación y los planificadores de subsistemas. El usuario es el que implementa la estrategia de planificación para sus hilos. Como la implementación necesita hilos de Kernel, una parte de cada hilo es tomada como un objeto para ser mantenido en el Kernel y otra parte en la pila que está en el espacio de usuario.
Cada Planificador maneja una lista de hilos y/o planificadores. Cuando es activado, se decide cual de ellos va a ejecutar o correr y que intervalo de tiempo virtual toma cada uno. El planificador toma nuevamente el procesador después del intervalo de tiempo virtual especificado. Cada planificador funciona con el intervalo de tiempo virtual que recibe del planificador ya mencionado [06].
Cuando se activa un hilo a través de un planificador, éste marca el hilo como corriendo o ejecutándose, es decir, lo mueve desde la cola de listo a la función de ejecutando, es decir que este le entrega la CPU. Mientras que esté marcado como corriendo el planificador no vuelve a activarlo porque está fuera de su área de administración.
El timer por procesador es la instancia principal para la activación y la apropiación. Es el que pregunta al sistema planificador qué hilo o planificador tiene que ser activado **(1)**, quien retorna una referencia al hilo o planificador que desea activar y la división de tiempo que debe obtener. La interacción entre el timer y el planificador se hace por métodos de invocaciones (invocaciones de puerta en caso de llamadas entre dominios). El timer guarda la información de las divisiones de tiempo y activan al objeto retornado. Si el mismo es un planificador, lo activa preguntando qué hilo o planificador debe activar **(2)**. Si es un hilo, lo activa por conmutación **(3).** El timer primero almacena toda la información de las divisiones de tiempo que obtiene de los planificadores y luego busca un hilo para conmutar **(4)**.



El timer obtiene peticiones desde cada planificador que quiere implementar multitareas apropiativas, siendo capaz de almacenar varias jerarquías de peticiones. En el caso de tener dos solicitudes, una de una aplicación planificadora y otra de un sistema planificador, el hilo activo es apropiado luego de transcurrido el menor tiempo de una de las solicitudes. Su estado y el estado inferior del timer son guardados en su objeto hilo y el sistema planificador se activará y notificará que la derivación para la aplicación planificadora es apropiativa. Así, el objeto hilo representa la totalidad de la derivación [07].
Si el sistema planificador quiere reactivar la derivación, lo hace directamente reponiendo la lista de timer guardada y el contexto del hilo.
Un planificador contiene referencias a las derivaciones que puede activar. Si se activa una derivación, se notifica al timer cuantas ranuras de tiempo debe otorgarle. Luego de ser usadas, el planificador es notificado y cuestionado por cual de estas derivaciones debe ser activada después. Es transparente para el planificador si éste o estos hilos hijos son asociados por un nivel superior de planificador. Cada planificador puede elegir su división de tiempo individualmente. Si un hilo o un planificador se bloquea o termina, sucede lo mismo que en el caso de apropiación: el planificador padre es notificado y puede decidir que derivación quiere activar.

7. Seguridad

Un problema de seguridad es la llamada desde el timer, cuando éste pregunta a un planificador que hilo va a activar. Esta puede ser una llamada dentro del nivel de usuario, por lo que el Kernel no está seguro de que el planificador a nivel-usuario devuelva el control.
Mientras la conmutación de hilos corre, puede haber una lista de timer de nivel-superior instalada. Si la conmutación de hilo es apropiativa o bloqueante dentro de la derivación del administrador apropiativo. Cuando el sistema planificador activa la derivación nuevamente, la conmutación de hilo es activada y no es creado un nuevo hilo.
En la sincronización de las llamadas entre dominios existen dos tipos de objetos activos. Tenemos los “Transportadores” (“*shuttles*”) los cuales son las entidades planificadoras que contienen guardado el estado del procesador; y tenemos los hilos que contienen las partes

por-dominio del transporte. Si un hilo llama a otro dominio, un nuevo hilo en el destino del dominio es agregado al transporte simple.
Este no es aceptado por muchas aplicaciones y servidores. Si un planificador puede apropiarse de un hilo en otro dominio, puede afectar a la funcionalidad y la integridad del dominio.
Se resuelve este problema pasando la responsabilidad del transporte al dominio llamado.
Este mecanismo es especialmente importante si el usuario llama desde el Kernel: Una invocación al Kernel en el sistema operativo es normalmente una llamada entre dominios; nos aseguramos con este mecanismo que un hilo que corre en el Kernel siempre será planificado por el Kernel.
Una llamada entre dominios (puerta de llamada) se implementa eficientemente, porque es la base del mecanismo de interacción entre dominios en el sistema operativo. Así, el *overhead* de los cuatro métodos de llamadas adicionales (agregar y quitar el hilo desde / hasta el planificador) no es aceptable. Solucionamos este problema con un concepto que llamamos "Planificación perezosa" ("*lazy scheduling*"):
Mientras un hilo ejecuta no hacemos nada especial. Cuando el hilo es apropiado o bloqueado, chequeamos si el hilo en ese momento está ejecutándose en otro dominio. Si es así, se notifica a ambos. Así, el planificador del dominio llamador conceptualmente planifica el hilo antes que sea inhabilitado o bloqueado por alguien, mientras esta ejecutándose en otro dominio.
Si un planificador activa otro planificador, la derivación activada es deshabilitada en el planificador superior. Así, el planificador activado no puede obtener más de un procesador. Esto es sólo posible si tenemos un único procesador o si el planificador sólo tiene un hilo por ranura. En multiprocesadores el planificador puede decir a sus planificadores padres, que será capaz de manejar n procesadores. Entonces el planificador puede activarlo n veces. **[04]**

8. Paralelismo con los hilos de usuario

El planificador del sistema programa una tarea en un procesador solamente si tiene acceso a los recursos requeridos en sus modos de accesos específicos, tanto exclusivo como compartido.
La modificación de un recurso desde un hilo afecta al entorno del resto de los hilos del mismo proceso. Se necesita sincronizar la actividad de los distintos hilos para que no interfieran unos con otros ni se corrompan las estructuras de datos.
El planificador elimina la espera asociada con el acceso a los recursos.
El contexto de un hilo reside en memoria física en el tablero del procesador que ejecuta el hilo. Si un hilo se ejecuta en el procesador de un tablero que contiene el espacio de direcciones del proceso al que pertenece el hilo, el acceso a datos en el espacio de direcciones no requiere el bus, por lo que los accesos a memoria son más rápidos y otro procesador no es discriminado. [01]
A fin de ejecutar un hilo en un procesador diferente, el segmento de código del proceso se carga en el arranque.
El planificador reconoce la diferencia de tiempo en el acceso a memoria e intenta construir un planificador de tal manera que se maximice la utilización del procesador y se minimice el costo de referencias a memoria.
El diseñador de sistemas de tiempo real, establece un conjunto de acciones que el sistema puede realizar para llevar a cabo su funcionalidad. Pueden existir múltiples acciones para el mismo evento, difiriendo en la precisión, exactitud e integridad. La mejor elección de la acción, como respuesta a un evento externo, puede depender de la situación exacta en la que el evento ocurre. Diseñar múltiples respuestas genera un sistema mucho más robusto.

Dado que el kernel de Spring retiene una gran cantidad de información en tiempo de ejecución, el proceso de planificación es capaz de proporcionar información en respuesta a pedidos de planificación. En la planificación se debe tener en cuenta el tiempo de ejecución del hilo que intentará generar otros hilos. **[01]**

El concepto de hilos livianos o *threads* nació como un exitoso intento de utilizar memoria compartida basada en sistemas paralelos para algoritmos paralelos detallados. Más allá de sus beneficios, los hilos de kernel poseen ciertos problemas como el *overhead* en aplicaciones informáticas de alto rendimiento. La solución a este problema en particular es la administración de hilos de usuario de manera efectiva, usando los hilos de kernel como procesadores virtuales. Pero esto conlleva otros problemas debido a la falta de integración del sistema con los hilos de usuario.

Los hilos durmientes (*Sleeping threads*) [08] son un mecanismo que utiliza la comunicación entre el kernel y el nivel usuario para abordar los problemas de hilos a nivel usuario bloqueados en kernel y las interferencias entre en nivel usuario y la administración del kernel. Este mecanismo implementa las funcionalidades de hilos de repuesto en kernel, desplazando la toma de decisiones administrativas del kernel al nivel usuario. La administración de los hilos de repuesto debe ser realizada por la biblioteca de hilos, así como todas las decisiones administrativas. Esto permite evitar la interferencia con la planificación del kernel.

Existen dos maneras extremas de implementar una nueva funcionalidad a nivel usuario que requiere de modificaciones del kernel: Integrar tanto como sea posible en el kernel, ofreciendo una interfaz comprensible al usuario; o integrar sólo lo necesario como base a una implementación a nivel de usuario. La primer opción tiene la ventaja de que la integración en el kernel simplifica la implementación. Toda la información del kernel puede accederse y el estado del sistema es mayormente controlado por el kernel como única instancia. Las limitaciones de la integración en el kernel están dadas por la eficiencia y la sobrecarga del kernel. La desventaja de esta opción es la necesidad de realizar modificaciones extensas al kernel. No es posible realizar modificaciones o extensiones rápidas y cortas. La segunda opción puede llevar a una implementación más compleja. Esto se debe a la distribución de la información de estado entre dos instancias. Pero la segunda opción tiene también importantes ventajas:

- Separación sencilla de los mecanismos y las estrategias (realizadas en las bibliotecas).
- Intercambio sencillo de módulos de estrategia.
- Las partes realizadas en el nivel de usuario no deben ser revisadas para mantener la integridad del sistema.
- Buena portabilidad hacia otros sistemas.
- Es sencillo conseguir interfaces uniformes sobre los límites del sistema.

El mecanismo de Hilos durmientes propone que la biblioteca de hilos mantenga hilos de repuesto en el kernel. De esta manera, en el supuesto de un bloqueo en el kernel, el mecanismo puede cambiar a uno de los hilos de repuesto. Esto puede realizarse eficientemente utilizando páginas de memoria compartidas entre el kernel y la biblioteca de usuario. De esta manera el propio kernel puede manipular el estado de un hilo de usuario.

La estrategia debe mantener un seguimiento de los hilos de repuesto que reemplazan a los procesadores virtuales, de manera de mantener constante la cantidad de procesadores virtuales en uso en todo momento. Pero la creación de hilos de repuesto genera un problema: el kernel no debe administrar el nuevo hilo hasta que se necesite como hilo de repuesto. Para solucionarlo se utiliza un *system call* especial para la creación de hilos que encole los nuevos hilos como hilos de repuesto y todo el manejo de excepciones es hecho en el mismo procesador.

9. Conclusiones

Se han obtenido resultados altamente satisfactorios en la administración de las prioridades en las colas de listos pero no así en la administración de colas con algoritmos Round Robin debido a que generan esperas más largas de lo previsto con un importante overhead y hasta el momento, una variación entre hilos mayor de lo normal.
El pasaje de Kernel a usuario y de usuario a Kernel es satisfactorio y no se prevé que en el transcurso de este año se realice alguna mejora.
Los scheduler activations tiene un gran comportamiento como hilos de usuario pero muy bajo rendimiento como hilos de kernel
Los hilos durmientes son un mecanismo que permite resolver algunos problemas en las bibliotecas de hilos de usuario existentes. Reservando hilos de repuesto, la biblioteca de hilos es capaz de reaccionar a un bloqueo de un hilo en kernel. El número de procesadores virtuales se mantiene fijo. Toda la administración dentro de una aplicación es trasladada al nivel usuario, incluso en el caso de un hilo bloqueado en el kernel. Esto permite una administración a nivel usuario que respeta las características de las computadoras modernas paralelizadas (ej. arquitecturas de memoria y cache) sin interferencia de *context switch* del kernel. Con este mecanismo se consigue un buen sistema de integración a través de una amplia comunicación entre las bibliotecas de hilos y el kernel.

10. Referencias

[01] Humprey Marty, Gary Wallace and John A. Stancovick - Kernel-Level Threads for Dynamic, Hard Real-Time Environments – Department of Computer Science University of Massachusetts - 1995

[02] Liedtke Jochen On Kernel Construction GMD—German National Research Center for Information Technology - Association for Computing Machinery, Inc. (ACM) – 1995

[03] Finkelstein David, Norman C. Hutchinson, Dwight J. Makaroff, Roland Mechler and Gerald W. Neufeld, Real Time Threads Interface, Department of Computer Science, University of British Columbia, Canada

[04] Riechmann Thomas, Jürgen Kleinöder - User-Level Scheduling with Kernel Threads - Computer Science Department Operating Systems - IMMD IV Friedrich Alexander University Erlangen - Nürnberg, Germany - 1996

[05] Anderson Thomas E., Brian N. Bershad, Edward D. Lazowa, Henry M. Levy - Scheduler activations: effective kernel support for the user-level management of parallelism – ACM Transactions on Computer Systems (TOCS) – Páginas 53-79 - 1992

[06] Hans J. - Boehm Threads Cannot Be Implemented As a Library - HP Laboratories - PLDI ’05, June 12–15, 2005, Chicago, Illinois, USA

[07] Gene Cooperman, Jason Ansel and Xiaoqin Ma. - Transparent adaptive library-based checkpointing for master-worker style parallelism. In *Proceedings of the 6th IEEE International Symposium on ClusterComputing and the Grid (CCGrid06)*, Singapore, 2006. IEEE Press.

[08] Koppe Christoph - Sleeping Threads: A Kernel Mechanism for Support of efficient User Level Threads – Computer Science Department Operating Systems - IMMD IV Friedrich Alexander University Erlangen - Nürnberg, Germany – 1995

[09] Bershad Brian Natan, Stefan R. Savage, Przemyslaw Pardyak, Emin Gun Sirer, Marc Eric Fiuczynski, David Becker, Craig David Chambers, Susan Jane Eggers - Extensibility safety and performance in the SPIN operating system - ACM Transactions on Computer Systems (TOCS)

[10] Ulrich Drepper, Ingo Molnar - The Native POSIX Thread Library for Linux - Red Hat, Inc. – 2003

[11] Brian D. Marsh, Michael L. Scott, Thomas J. LeBlanc, Evangelos P. Markatos - First-class user-level threads - Computer Science Department - University of Rochester – ACM SIGOPS Operating Systems Review Volume 25, Issue 5 - Páginas 110 - 121 - 1991