Análisis del Problema de Inversión de Prioridades para Planificación de Tareas de Tiempo Real en RTAI/Linux

Claudio Aciti y Nelson Acosta

***Resumen*—En un entorno de multiprogramación con prioridades, la competencia por el acceso a los recursos puede crear anomalías, y en consecuencia estropear la planificación temporal. Estas anomalías son conocidas como los interbloqueos y la inversión de prioridades. Esta última no tiene solución pero si puede ser minimizada. En el sistema de tiempo real RTAI/Linux, el control de la inversión de prioridades se logra usando el protocolo Herencia de Prioridad o el protocolo Techo de Prioridad. En este trabajo se explica el funcionamiento de ambos protocolos y se estudia la performance en diferentes situaciones.**

***Index Terms*—RTAI, Tiempo Real, Inversión de Prioridades, Herencia de Prioridad, Techo de Prioridad.**

I. Introducción

En la actualidad, los sistemas embebidos han logrado un gran auge gracias a sus diferentes campos de aplicaciones y sus bajos costos comparados con sistemas tradicionales [4] [7]. Es muy común el uso cotidiano de sistemas embebidos, ya sea en electrónica de consumo (lavarropas, heladeras, microondas, relojes, consolas de juegos, control remoto, cámaras de video, fax, CD, DVD, GPS, televisión digital), en sistemas de comunicación (sistemas de telefonía, contestadores, celulares, beepers, PDAs, routers), en automóviles (inyección electrónica, frenos, elevadores de vidrios, control de asientos, instrumentación, seguridad), en la industria (instrumentación, monitoreo, control, robótica, control de tráfico, manejo de códigos de barras, ascensores), en medicina (monitores cardíacos, renales y de apnea, marcapasos, máquina de diálisis)[5], entre otros . Estas aplicaciones necesitan claramente ser controladas por sistemas de tiempo real, por lo que el interés por estos se ha visto renovado al igual que sus problemáticas.

Aunque el término tiempo real se utiliza con excesiva frecuencia para un gran número de aplicaciones que tienen respuesta rápida, la correcta definición de este tipo de sistemas se puede enunciar como sistemas informáticos en los que la respuesta de la aplicación ante estímulos externos, no solo debe ser lógicamente correcto sino que debe realizarse dentro de un plazo de tiempo establecido [6]. Para lograr que se cumplan los plazos, es necesario que el planificador de tareas logre un desempeño eficiente, asignado recursos a las tareas de mayor prioridad y evitando que ocurran anomalías, como pueden ser la inanición de tareas, el bloqueo mutuo o la inversión de prioridades.

El parche RTAI (Re-al Ti-me A-pli-ca-tion In-ter-fa-ces), fue de-sa-rro-lla-do i-ni-cial-men-te por el Dipartimento di Ingeneria Aerospaziale Politecnico di Milano (DIAPM) como una variante de the New Mexico Institute of Technology's (NMT) RTLinux [3]. Este parche le provee determinismo y preemptividad (deshalojo) al kernel Linux. De esta forma, un sistema operativo de tiempo real RTAI/Linux tiene la capacidad de deshalojar a una tarea del kernel para darle paso a otra tarea de mayor prioridad. En los sistemas de tiempo real con deshalojo de tareas del kernel ocurre el problema de inversión de prioridades. Se denomina así a la situación que sufre una tarea cuando es retrasada por otra de menor prioridad. Este problema no tiene solución pero si puede ser minimizado. RTAI/Linux implementa los protocolos Herencia de Prioridad y Techo de Prioridad para minimizar este problema [1].

A continuación, se hace una descripción del problema. En la 3^{ra} sección se presentan los mecanismos para evitar este problema. En la 4^{ta} sección se evalúa el rendimiento de estos mecanismos. En la 5^{ta} sección se sacan conclusiones y por último se presenta la bibliografía.

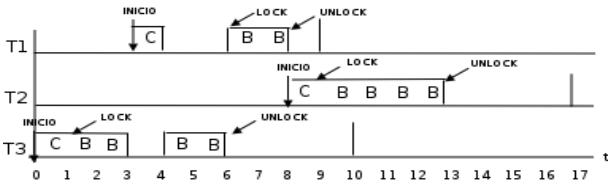
II. Descripción del Problema

En sistemas de tiempo real complejos la sincronización de tareas con prioridades es dificultosa. Las tareas, generalmente, tienen que utilizar un mismo recurso por lo que deben competir entre sí para obtenerlos. Los recursos, a los que acceden las tareas, pueden ser dispositivos de hardware externos (sensores, actuadores, cámaras, etc), elementos de computadora (memorias, discos rigidos, disqueteras, etc) y elementos de software (mutex, colas, semáforos, etc). Cada recurso n tiene k unidades, y cada unidad se usa de forma no interrumpible y exclusiva, aunque si pueden detenerse para hacer un cambio de contexto con otra tarea. Los recursos que pueden ser accedidos por más de una tarea a la vez, se modelan como varias unidades, para lograr que el acceso sea exclusivo.

Cuando una acción (unidad mínima de una tarea) tiene que usar una unidad k de un recurso n, primero debe bloquearlo para usarlo de forma exclusiva y no interrumpible (sección crítica). Si la petición de un bloqueo falla, la acción solicitante se bloquea y espera hasta que el recurso que necesita sea liberado. Si la tarea que espera por un recurso, está siendo bloqueada por una tarea de menor prioridad se dice que ocurre una inversión de prioridades. Esta anomalía provoca que la tarea de mayor prioridad retrase su tiempo de ejecución. Este retraso no puede ser solucionado pero si puede ser minimizado por algún mecanismo de control.

En la Figura 1, se puede ver que en un tiempo $t = 3$, se inicia la tarea $T1$ y al tener mayor prioridad desaloja a la tarea $T3$. En un tiempo $t = 4$, la tarea $T1$ quiere entrar en su sección crítica pero no puede bloquear al recurso R porque ya lo tiene bloqueado $T3$. Entonces $T1$, debe esperar a que $T3$ termine y libere el recurso R.

Departmento de Computación y Sistemas, Fac. Ciencias Exactas. Universidad Nacional del Centro de la Provincia de Buenos Aires (7000), Tandil, Argentina. Phone:+51 (02293) 43-9680,
email:{caciti, nacosta}@exa.unicen.edu.ar

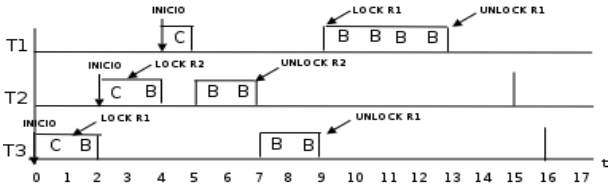


	Prioridad	Intervalo	Sec Crítica
T1	1	(3,9]	[R,2]
T2	2	(8,17]	[R,4]
T3	3	(0,10]	[R,4]

B: Bloqueando C: Corriendo

Figura 1. Inversión de prioridades.

Esta anomalía puede tambien darse de forma transitiva. Es decir que una acción de alta prioridad espera por un recurso bloqueado por una acción de baja prioridad, quien a su vez está siendo demorada por otra acción de prioridad intermedia. Transitivamente, la acción de prioridad intermedia detiene a una acción de prioridad alta (Figura 2).



	Prioridad	Intervalo	Sec Crítica
T1	1	(4,18]	[R1,4]
T2	2	(2,15]	[R2,3]
T3	3	(0,16]	[R1,3]

B: Bloqueando C: Corriendo

Figura 2. Inversión de prioridades transitiva.

A continuación se presentan los mecanismos Herencia de Prioridad y Techo de Prioridad, para controlar estas anomalías.

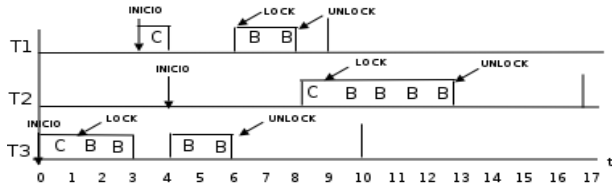
III. Mecanismos de control de la inversión de prioridades

Los mecanismos de control no pueden evitar el problema de la inversión de prioridades, pero si pueden minimizarlo. El propósito de estos es qué, una tarea no retrase demasiado esperando por algún recurso bloqueado por otra tarea con menor prioridad. Este inconveniente surge ya que una vez que una tarea ingresó en su sección crítica debe terminar antes de liberar el recurso sino se produciría una inconsistencia.

A. Protocolo de Herencia de Prioridad

Este protocolo ajusta la prioridad dinámicamente de la acción que está usando un recurso, y este es requerido por otra acción de mayor prioridad. Si se suponen dos tareas T_x y T_y, donde $\pi(T_x)$ es la prioridad de T_x y $\pi(T_y)$ es la prioridad de T_y y $\pi(T_x) < \pi(T_y)$. Si T_x está bloqueando al recurso R mientras lo usa, y la acción T_y intenta bloquearlo, entonces T_x hereda la prioridad de T_y mientras esté dentro de su sección crítica. De esta forma se evita que T_x sea desalojado [2].

En la Figura 3, en un tiempo $t = 3$ la tarea $T1$ inicia y en $t = 4$ intenta bloquear al recurso R pero no logra porque está bloqueado por la tarea $T3$. En ese instante, $T3$ hereda dinámicamente la prioridad de $T1$, solo por el tiempo que le lleve terminar de usar el recurso R que necesita $T1$, y no por el resto de su ejecución. Con este cambio, se puede observar que en $t = 4$, la tarea $T2$ no tiene suficiente prioridad para tomar el control. De esta forma se evita la inversión de prioridades transitiva.



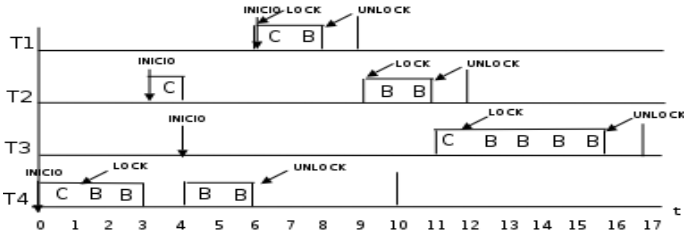
	Prioridad	Intervalo	Sec Crítica
T1	1	(3,9]	[R,2]
T2	2	(4,17]	[R,4]
T3	3	(0,10]	[R,4]

B: Bloqueando C: Corriendo

Figura 3. Protocolo de Herencia de Prioridad

B. Protocolo de Techo de Prioridad

Este protocolo designa como techo de prioridad de un recurso a la máxima prioridad de las tareas que lo usan. El protocolo consiste en que la prioridad de una tarea sea dinámica y que su valor sea el máximo entre su prioridad básica y las prioridades de las tareas a las que bloquea. Entonces, una tarea solo puede usar un recurso si su prioridad dinámica es mayor que el techo de todos los recursos en uso por otras tareas [2].



	Prioridad	Intervalo	Sec Crítica
	1	(6,9]	[R3,1]
T2	2	(3,12]	[R1,2]
T3	3	(4,17]	[R2,4]
T3	4	(0,10]	[R1,4]

B: Bloqueando C: Corriendo

Figura 4. Protocolo Techo de Prioridad

En la Figura 4, se ve que en $t = 3$ la tarea $T2$ da inicio a su ejecución y un instante despues, quiere bloquear al recurso $R1$ pero este ya está bloqueado por la tarea $T4$. Entonces, $T4$

cambia su prioridad dinámicamente por la prioridad de la tarea de mayor prioridad que la bloquea ($T2$ en este caso). En $t = 4$, la tarea $T3$ no puede ni siquiera iniciar porque su prioridad es menor que la actual que tiene $T4$, pero en $t = 6$, el control lo toma $T1$, ya que su prioridad es la mayor y quiere usar el recurso disponible $R3$.

Con este mecanismo, cada tarea se puede bloquear una vez como máximo en cada ciclo. Además, evita los interbloqueos.

IV. INVERSIÓN DE PRIORIDADES EN RTAI/LINUX

La planificación de procesos es, además del manejo del reloj de alta resolución y del manejo de atención de interrupciones, uno de los factores críticos que afecta directamente la performance de Tiempo Real en sistemas multitarea. Un sistema multitarea permite que mas de una tarea puedan ser cargadas para su ejecución al mismo tiempo. La performance del sistema depende de la elección del algoritmo de planificación que permita realizar una secuencia acertada, evitando la inanición de un proceso o bloqueos entre ellos y minimizando retrasos en la ejecución por problemas de inversión de prioridades.

RTAI, es un parche en el kernel Linux que le agrega funcionalidad de tiempo Real. Este sistema es desarrollado por el Politecnico di Milano - Dipartimento di Ingeneria Aerospaziale (DIAPM). El scheduler,el corazón de RTAI, proporciona una serie de mecanismos con capacidades de tiempo real. Al utilizar el scheduler de RTAI, un proceso puede cumplir con limitaciones de tiempo real duro y ser capaz de ejecutarse deterministicamente, que significa que el proceso puede ser ejecutado exactamente tal y como se diseó y no estar limitado por el scheduler general GNU/Linux programador. Un proceso de tiempo real ejecutado por el scheduler de RTAI, puede utilizarse para controlar aplicaciones complejas, como el control numérico, proceso industrial y cualquier tarea compleja que requiere de una programación correcta en tiempo y forma.

RTAI proporciona servicios simétricos en tiempo real duro inter/intra los espacios del usuario y del kernel. Este soporte viene dado a través de dos schedulers, rtai_lxrt y rtai_sched. Estos schedulers pueden operar tanto en modo kernel y en modo usuario, y difieren únicamente en relación con los objetos que pueden planificar. rtai_lxrt es un co-scheduler de GNU/Linux, dando características de tiempo real duro a objetos como procesos hilos kthreads. En cambio, rtai_sched soporta, no sólo, tiempo real duro para todos los objetos de Linux (procesos hilos kthreads), sino también para tareas propias del núcleo RTAI, que están en el espacio del kernel y solo son planificables por objetos propios del scheduler de RTAI.

Así, mientras que en el espacio de usuario no hay otra alternativa que la planificación de los procesos y subprocesos de Linux, se puede solicitar él mismo lo que vale la pena la redundancia de la programación con diferentes objetos en el kernel espacio, como a primera vista, que permitirá tanto las mismas funcionalidades.

Comprendiendo las ventajas y las desventajas de un scheduler con respecto a otro es importante para decidir cuál utilizar y aplicar en un proyecto de software en tiempo real.

Solamente, para las aplicaciones en el espacio de usuario, los dos schedulers son lo mismo.

La gran ventaja de las tareas del kernel RTAI es su velocidad en el tiempo de conmutacion respecto a los threads del kernel Linux. Pero tambin es importante conocer que estas tareas operan al margen de cualquier entorno Linux. Este comportamiento requiere cuidados especiales si se necesita interoperar con Linux. Por lo tanto, a menos que exista una verdadera limitación de diseo utilizando el GNU / Linux co-planificador es la mejor opción, o al menos es la que da más libertad para el proceso de desarrollo.

A. Funciomiento de los schedulers

- `RT_SCHED_HIGHEST_PRIORITY 0`
- `RT_SCHED_LOWEST_PRIORITY 0x3fffffff`
- `RT_SCHED_LINUX_PRIORITY 0x7fffffff`

donde se puede ver que las prioridades de menor valor son las de mayor prioridad.

Las funciones de para el manejo de prioridades son:

- `int rt_get_prio (RT_TASK *task)`: Re-tor-na la prio-ri-dad base nativa de la tarea actual.
- `int rt_get_inher_prio (RT_TASK *task)`: Retor-na la prio-ri-dad base que ha heredado de otras tareas.
- `int rt_change_prio(RT_TASK *task,int priority)`: Cambia la prioridad de una tarea.

En RTAI los semáforos de recursos soportan dos caminos para manejar la he-ren-cia de prioridad, ... El tipo de herencia de prioridad se indica por medio de la opcion FULL_INHER_PRIORITY.

Si esta opción está desactivada, una tarea que se apropia de algún recurso recuperará su prioridad base solamente cuando esta libere el último recurso del cual se apropió. Sin activar esta opción una tarea poseer cualquier recurso recuperar su base de prioridad sólo al dar a conocer el último recurso de su propiedad. Así que toda la prioridad a la herencia de trabajo, ya que generalmente se espera de un recurso de propiedad individual, sólo porque, cuando un grupo alcanza la propiedad de más de un recurso, la prioridad herencia se convertirá en un límite máximo de adaptación dinámica prioridad. En tal caso, de hecho, la tarea prioritaria es el aumento de acuerdo con la más alta prioridad de tareas en espera en cualquier recurso es propietaria de una tarea, que a su vez incluyen a causa de mensajes intercambiados con él, pero será devuelto a su base de prioridad sólo cuando todos los recursos son de propiedad puesto en libertad. Esta es una elección de compromiso de diseo encaminadas a evitar la búsqueda de la nueva prioridad que se inheredited a través de multiplicar los recursos de propiedad y enviar tareas bloqueado / rpcing a la tarea.

Habilitar esta opción de configuración, permite una herencia de prioridad completa, de modo que cuando una tarea libera algún recurso que posee el semáforo, este adquirirá la prioridad de la tarea con mayor prioridad, ya sea que esté en espera de un recurso que todavía es propiedad de la tarea o que esté bloqueado enviando / recibiendo.

V. CONCLUSIONES

La sincronización de tareas con prioridades es un tema sin resolución. La mejor estrategia para reducir los conflictos es reducir la competición por los recursos compartidos. Entre los

mecanismos para minimizar el problema de inversión, el protocolo techo de prioridad es el que mejor performance tiene aunque no está implementado casi en ningún sistema operativo (en MaRTEOS y una extensión de RTLinux). El mecanismo herencia de prioridad, si bien no elimina la inversión de prioridades (ni los interbloqueos), es la forma más común que tienen los sistemas operativos para afrontar la inversión de prioridades. Prioridad de inversión es un problema grave que, si se permite que ocurra, puede causar un sistema al fracaso. Generalmente es más sencillo para evitar que la inversión de prioridad para resolverlo en el software. Si es posible, eliminar la necesidad de recursos compartidos por completo, evitando toda posibilidad de inversión prioritarios. Si no puede evitar la inversión de prioridad, al menos asegúrese de que sea limitada. Prioridad sin límites puede dejar inversiones de alta prioridad no puede ejecutar las tareas, lo que resulta en aplicación fracaso. Dos métodos comunes de inversión se limitan prioridad la prioridad tope de protocolo y la prioridad herencia protocolo. Ni el protocolo es perfecto para todas las situaciones. Por lo tanto, buen análisis y el diseo son siempre necesarios para comprender que la solución, o combinación de soluciones, es necesario para su aplicación particular. La elección depende principalmente de las necesidades de programación. Sólo tener en cuenta que el límite máximo de prioridad por defecto dinámica es más simple, un poco más eficaz y menos propensa a muerto toda la prioridad a la herencia.

REFERENCIAS

[1] Rtai_user_manual_34_03.pdf, April 2006.
[2] G. Buttazzo. Why real-time computing? *Proccedings of the 2006 ANIPLA. International Congress on Methodologies for Emerging Techonologies in Automation*, 2006.
[3] Politecnico di Milano Dipartimento di Ingegneria Aerospaziale. Rtai - the realtime application interface for linux from diapm.
[4] H. J. Osorio Ríos. J. A. Jaramillo Villegas, L. M. Perez Perez. Linux sobre una fpga. *Sciencia et technica. Año XIII. Número 37*, 2007.
[5] J. Rapallini A. Quijano J. Osio, F. Salguero. Análisis de modelos computacionales para sistemas embebidos. *XII Iberchip, IWS06. Costa Rica*, 2006.
[6] Abraham Silverchatz and Peter Baer Galvin. *Sistemas Operativos*. Prentice Hall, 1999.
[7] J. Valvano. *Introducción a los sistemas de microcomputadores*. Thomson Learning Ibero. ISBN 9706863168, 2003.