Hardware Support For Priority Inheritance

CARLOS MARTÍN FLORES GONZÁLEZ, Carné: 2015183528

Instituto Tecnológico de Costa Rica
Maestría en Computación
Sistemas Operativos Avanzados
Profesor: Francisco Torres Rojas, Ph.D

El planeamiento de tareas(*task scheduling*) involucra una serie de consideraciones debido al hecho que las tareas comparten recursos. En un sistema paralelo con un sistema operativo multitarea en tiempo real(*preemptive RTOS*), la consistencia de los datos compartidos por medio del uso de una variable de bloqueo se mantiene a costa de serializar los accesos a los recursos compartidos(solamente una tarea puede tener acceso a un recurso compartido en un momento dado). Esto puede llevar al siguiente problema de inversión de prioridades: una tarea de baja prioridad pudo haber iniciado el acceso a un dato compartido - obteniendo el bloqueo(*lock*) para ese dato compartido - justo antes que una tarea de alta prioridad intente acceder a ese mismo recurso compartido, en cuyo caso la tarea de alta prioridad verá forzada a esperar por la tarea de baja prioridad. Peor aún, podría haber una tarea de prioridad media que se adelante a la tarea de baja prioridad antes de que esta libere su *lock*, causando tiempos de espera impredecibles e inaceptables para la tarea de alta prioriodad.

SoCLS: System-on-a-Chip: SoCLC es una unidad de hardware simple que puede ser fácilmente integrada a un SoC como un núcleo de propiedad intelectual a través del bus del sistema. El mecanismo SoCLC ha sido implementado en un *preemptive RTOS*, manejando sincronización *non-blocking* y *blocking*.

RTOS support for the SoCLC:. *Short CS locks*: como la duración de la sección crítica(CS) es pequeña y es muy probable que el poseedor del *lock* libere el *lock* pronto, el mecanismo de SoCLC aplica una sincronización *non-blocking*. Como tal, la tarea solicitante espera hasta que obtenga el *lock* y pueda ejecutar su CS. *Long CS locks*: si la duración de la CS es lo suficientemente larga para compensar el cambio de contexto, es más ventajoso aplicar sincronización *blocking*. Por lo tanto en el caso de una CS larga, una tarea que espera por un *lock* de una CS larga se permite que sea rechazada para ceder el procesador a otras tareas.

Background: *Priority inversion problem:* este problema ocurre cuando una tarea de prioridad alta tiene que esperar por una tarea de prioridad baja y este tiempo de espera es impredecible o ilimitado[1]. Por ejemplo, si una tarea de baja prioridad obtiene un *lock* para una CS antes que una tarea de alta prioridad intente adquirir el *lock*, la tarea de alta prioridad está bloqueda. En tales condiciones, a la tarea de prioridad alta podría ocurrirle un bloqueo ilimitado si tareas de prioridad media arriba y se adelantan a la que la tarea de baja prioridad libera su *lock*, en el cual la tarea de alta prioridad está bloqueada. En otras palabras, la tarea de alta prioridad es privada de recursos de CPU durante el tiempo de ejecución de secciones críticas de que lleva a cabo tareas de baja prioridad; esto de altera las propiedades de tareas *de facto* en tiempo de ejecución, perturbando el comportamiento de sistemas en tiempo real.

[1]Unbounded Blocking

Solution: priority inheritance: el problema de inversión de prioridades puede ser evitado por un protocolo de herencia de prioridades(PIP). El PIP básico previene el bloqueo ilimitado de tareas de prioridad alta debido a tareas de prioridad baja. Si una tarea de baja prioridad bloquea una tarea de alta prioridad, entonces la tarea de baja prioridad ejecuta su sección crítica con el nivel de prioridad de la tarea con la prioridad más alta que bloquea. De esta forma la tarea de baja prioridad *hereda* la prioridad de la tarea con la prioridad más alta (que está bloqueada por la tarea de baja prioridad). En PIP el tiempo máximo de bloqueo es igual al largo de una CS y el bloqueo puede ocurrir como máximo una vez por cada *lock*.

En PIP, las tareas con prioridad alta pueden sufrir de bloqueo encadenado(*chaining blocking*). Esta es la condición en donde una tarea de alta prioridad es bloqueada por más de un *lock* por causa de más de una tarea de prioridad baja. *Chained blocking* causa extra *overhead* en el cambio de contexto. Para remediar esto el PIP básico se ha amplicado con el *original priority ceiling protocol*(OPCP) el cual previene ambos, inversión de prioridades y bloqueo encadenado. En OPCP, a cada CS se le asigna una prioridad tope(*ceiling priority*) que es igual a la prioridad de la tarea con la prioridad más alta que se puede bloquear en la CS. A una tarea se le permite entrar en la CS solamente si su prioridad dinámica es más alta que la prioridad tope de la CS. OPCP garantiza que una tarea puede ser bloqueada durante la mayor parte de la duración de la CS como máximo una vez. En OPCP las relaciones de bloqueo son rastreadas en el RTOS, lo que constituye en un *overhead* en la implementación. El *immediate priority ceiling protocol*(IPCP), provee una implementación más fácil y garantiza la prevención de bloqueos encadenados. Tan pronto como el *lock* es otorgado a una tarea, la prioridad dinámica de la tarea es elevada inmediatamente a la prioridad tope de la CS. En IPCP hay potencialmente menos cambios de contexto porque IPCP requiere que ocurran menos *preemptions*. Esta característica de IPCP también es beneficiosa en la asignación de pilas para eventos *task-preemption*, esto es, el número de pilas requeridas puede ser especificado inicialmente a un menor costo.

Methodology. *Atlanta-RTOS priority inheritance vs. SoCLC priority inheritance:* Atlanta-RTOS soporta el protocolo de herencia de prioridades básico. Funciones específicas provistas dentro del RTOS gestionan los niveles de prioridad de las tareas. Esta es una solución basada en software. SoCLC por el contrario es una solución de hardware: a diferencia de Atlanta-RTOS, no se requiere de operaciones de inserción/eliminación desde/hacia una lista de tareas en espera para una CS. Tampoco hay ningún ajuste en la *ready-list* cada vez que un cambio en la prioridad de una tarea se realiza. En el caso del mecanismo de herencia de prioridades en Atlanta-RTOS, las tareas de inserción/eliminación realizadas en la lista de espera de tareas lleva a otro inconveniente: estas listas son listas enlazadas de tareas que están esperando por una CS y el número de tareas en una lista afecta las operaciones de inserción/eliminación. Por ejemplo, dada la eliminación de una tarea, el tiempo de búsqueda y el esfuerzo computacional se aumentará a medida que el número de tareas en la lista crezca. En SoCLC no importa el número de tareas que haya, el hardware puede gestionar los estados de las tareas y actualizar las prioridades de las tareas en un número fijo de ciclos de reloj.

Priority inheritance hardware architecture: Principales componentes: *status board, priority encoder, interrupt generator* y *task-wakeup register*. La *status board* mantiene el estado de cada variable de *lock*[2], información acerca de cuáles están bloqueadas esperando por cada *lock*, la prioridad estática de la tarea dueña del *lock* actual y la prioridad dinámica de cada tarea. El *lock*, el dueño, la propiedad dinámica de cada tarea y el *wakeup-register* puede ser accedidos por cada elemento de procesamiento(PE). Para adquirir un *lock* $lock_i$, una tarea $task_j$ corriendo en un procesador PE_k primero accesa el SoCLC leyendo el correspondiente valor del bit $lock_i$ de la *status board*. Si el valor del *lock* es '0', $task_j$ se convierte en el dueño de $lock_i$. Por lo tanto, la prioridad estática de $task_j$ se escribe dentro de la posición $owner_i$ y su propiedad dinámica se actualiza al valor $ceiling_i$. La prioridad de $task_j$ ha sido elevada a $ceiling_i$ la, lo que implica que el *lock* que la tarea dueña del *lock*, $task_j$, ha heredado la prioridad

[2]El *lock* este libre o no

de la tarea con la prioridad más alta que vaya a tener $lock_i$. Si otra tarea, $task_{j+1}$, está corriendo en un procesador PE_{k+1} y también desea adquirir $lock_i$, $task_{j+1}$ fallará en adquirir el *lock* ($lock_i$ no está libre, lo tiene $task_j$) y su bit de ubicación ($j + 1$), en la posición $lock_i$ del *task state*, se pone en '1' - indicando que $task_{j_1}$ está esperando por $lock_i$. Cuando $task_i$ libera $lock_i$, si $task_{j+1}$ es la única tarea esperando por $lock_i$, entonces el procesador de $task_{j+1}$, PE_{k+1}, recibe una interrupción generada por SoCLC y el manejador de la interrupción re-planifica $task_{j+1}$ en el procesador PE_{k+1}. Nótese qeu si más de una tarea está esperando por el mismo *lock*, entonces el *priority encoder* selecciona la tarea con la prioridad más alta, por ejemplo $task_h$, y hace que el SoCLC envíe una interrupción al procesador que corre $task_h$.

Experimental results: Se presenta el rendimiento obtenido por la herencia de prioridades SoCLC implementada en hardware comparada con la herencia de prioridades de Atlanta-RTOS implementada en software. Se presentan dos arquitecturas de hardware/software para comparar: (Arquitectura 1) comprende cuatro procesadores MPC750 en hardware, aplicación de tareas a nivel de usuario más el Atlanta-RTOS en software. Atlanta RTOS incluye el protocolo de herencia de prioridades y el mecanismo *spin-lock* para sincronización basada en bloqueos de CS largas y cortas. (Arquitectura 2) comprende cuatro procesadore MPC750 además del SoCLC en hardware, aplicaciones de tareas a nivel de usuario y el Atlanta-RTOS en sofware. El Atlanta-RTOS en la segunda arquitectura no incluye el mecanismo de herencia de prioridades ni el mecanismo de *spin-lock*, esto lo implementa el SoCLC en hardware. Las tareas que se simularon en los experimentos fueron: una aplicación de control de robot y un decodificador MPEG.

Se ejecutaron cinco tareas con diferentes prioridades en 4 CPUs. Se midió la latencia del bloqueo (*latency block*) -el tiempo requerido por un PE para adquirir un *lock* en ausencia de contención - el retardo del bloqueo (*lock delay*) -el tiempo que se comprendido cuando un *lock* es liberado y cuando el siguiente PE en espera adquiere el *lock* y el tiempo de ejecución en general. La herencia de prioridades implementada como parte del hardware SoCLC logra 88 % de aceleración en *lock latency* y 36 % de acelaración en el *lock delay* y 15 % de aceleración en el tiempo de ejecución en general cuando se comparó con la implementación de herencia de prioridades en el Atlanta-RTOS. También se analizaron los rastros de las ejecuciones de las cinco tareas que se corrieron. En el caso de la implementación con SoCLC todas las tareas pudieron ejecutarse dentro de los plazos acordados, mientras que en la implementación **sin** SoCLC hubo tres tareas que no que no se pudieron ejecutar en los plazos acordados y que causaron que la aplicación de control del robot tuviera que reiniciarse.

Synthesis results: Se mide el área ocupada por el SoCLC con y sin el hardware de herencia de prioridades en diferentes combinaciones/número de *locks* en relación con el área de dos compuertas de entrada NAND. Por ejemplo, para cuatro procesadores SoC, el hardware de herencia de prioridades soportando 32 *locks* de CS pequeños y 32 *locks* de CS largos ocupa 6628 compuertas en área.

1. ¿CUÁL ES EL PROBLEMA QUE PLANTEA EL *PAPER*?

La sincronización siempre ha sido un problema fundamental en sistemas multiprocesador, y en el caso especifíco de este artículo, el principal problema que se plantea es el de la inversión de prioridad en sistemas de tiempo real en donde si una tarea de baja prioridad obtiene un *lock* para una sección crítica larga antes que una tarea de prioridad alta intente adquirirlo, la tarea de alta prioridad se bloquea y mientras se presente esta condición la tarea de alta prioridad tendrá que permanecer esperando durante un tiempo impredecible. Esto impacta las prioridades de las tareas en tiempo de ejecución y por tanto el comportamiento del sistema en tiempo real.

2. ¿POR QUÉ EL PROBLEMA ES INTERESANTE O IMPORTANTE?

Dado que los multiprocesadores ejecutan aplicaciones multitarea en sistemas operativos de tiempo real, estructuras de datos importantes se compartidas - secciones críticas - son accesadas por comunicación inter-proceso y eventos de sincronización que suceden entre las tareas/procesadores del sistema. Típicamente la consistencia de una sección crítica puede ser mantenida al permitir que un solo proceso pueda operar sobre ella. Esto puede ser provisto con el uso de una variable de bloqueo (*lock variable*): una tarea que espera para entrar a una sección crítica adquiere el *lock* correspondiente; sólo entonces la tarea debería entrar a la sección crítica. El poseedor del *lock* lo libera luego de completar su ejecución de la sección crítica, de esta forma otras tareas son prevenidas de entrar en la misma sección crítica al mismo tiempo que el poseedor del *lock*. Dada la limitación de recursos de comunicación, los *locks* puede convertirse en cuello de botella del sistema: las tareas rotan en el bus de memoria hasta que el *lock* es liberado. Durante este tiempo de espera activa, la cantidad de trabajo útil se ve degradado.

3. ¿QUÉ OTRAS SOLUCIONES SE HAN INTENTADO PARA RESOLVER ESTE PROBLEMA?

En trabajos previos se ha mostrado que un caché *system-on-a-chip lock* reduce el tráfico de memoria *on-chip*, provee una transferencia justa y rápida del *lock*, simplifica el software, incrementa la capacidad de predictibilidad de los sistemas en tiempo real e incremente el rendimiento. Trabajos previos abordaron el problema de la inversión de la prioridad y propusieron herencia de prioridades como solución con protocolos de prioridades para sistemas monoprocesador y multiprocesador. También se han propuesto protocolos de *priority ceiling* para evitar bloqueos ilimitados y *deadlocks*.

4. ¿CUÁL ES LA SOLUCIÓN PROPUESTA POR LOS AUTORES?

Una extensión del mecanismo SoCLC con herencia de prioridades implementada en hardware. Herencia de prioridades provee un nivel más alto de garantías en tiempo-real para tareas de sincronización de aplicaciones. SoCLC es una unidad de hardware simple que puede ser fácilmente integrada a un SoC como un núcleo de propiedad intelectual a través del bus del sistema. El mecanismo SoCLC ha sido implementado en un *preemptive RTOS*, manejando sincronización *non-blocking* y *blocking*.

5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?

Resultados experimentales indican que el mecanimos de hardware SoCLC propuesto con herencia de prioridades logra 46 % de aceleración en el *lock delay*, 88 % de aceleración en *lock latency* y 15 % de aceleración en el tiempo de ejecución en general comparado con su contraparte de software.