UNA SOLUCION PARA EL MANEJO DE LA EXCLUSION MUTUA USANDO PROTOCOLO DE TECHO DE PRIORIDAD

GUILLERMO R. FRIEDRICH† y JORGE R.ARDENGHI‡

† *Dto. Electrónica, Univ. Tecnológica Nac., Fac. Reg. B. Blanca, 8000 Bahía Blanca, Argentina.*
gfried@frbb.utn.edu.ar
‡ *Dto. Cs. e Ing. de la Computación, Univ. Nac. del Sur, 8000 Bahía Blanca, Argentina.*
Laboratorio De Investigacion en Sistemas Distribuidos (LiSiDi) miembro del IICyTI
(Instituto de Investigacion en Ciencia y Tecnologia Informatica)
jra@cs.uns.edu.ar

Resumen

El uso de semáforos es una forma clásica de lograr la exclusión mutua entre dos o más procesos que concurren sobre una determinada región crítica. Un efecto no deseado del uso de semáforos consiste en la inversión ilimitada de prioridades, que puede dar lugar a que un proceso permanezca bloqueado por otros de menor prioridad durante un intervalo de tiempo excesivamente largo. Este problema es aun más grave en los sistemas de tiempo real, en los cuales se requiere que los intervalos de inversión de prioridad sean breves y de duración predecible. A tal efecto se han desarrollado algunas soluciones[1], una de ellas se basa en la aplicación del "protocolo básico de herencia de prioridades" y otra, mejor aún, se basa en el "protocolo de techo de prioridad". Debido a la complejidad creciente de estos dos métodos, es habitual que algunos sistemas operativos dispongan sólo de un manejo de semáforos elemental, algunos otros pueden tener implementado un esquema básico de herencia de prioridades, y unos pocos disponen de una *emulación* de techo de prioridad.
El presente trabajo describe un par de módulos que extienden el manejo básico de semáforos brindado por el sistema operativo, implementando el protocolo de techo de prioridad. Si bien la implementación experimental ha sido desarrollada sobre QNX Neutrino[4], la misma puede ser portada fácilmente a otros sistemas operativos.

1. Introducción

Los sistemas de tiempo real tienen la particularidad de que deben completar cada requerimiento antes de un determinado tiempo de vencimiento. A lo largo del tiempo se han estudiado diferentes estrategias para planificar la ejecución de un conjunto de tareas, a fin de poder predecir si el sistema podrá cumplir con las restricciones temporales de cada una. En el trabajo de Liu y Layland[2] se plantean algunas suposiciones: las tareas son periódicas e independientes, y se proponen dos alternativas: una planificación con prioridades fijas (Prioridades Monotónicas Crecientes -PMC-), y una planificación con prioridades variables (Menor Tiempo al Vencimiento -MTV-). En ambos casos el análisis se realiza en base al factor de utilización del procesador, expresado de la siguiente manera:

$$U = \sum_{i=1}^{m} \left(\frac{C_i}{T_i} \right) \qquad (1)$$

Donde: m : cantidad de tareas,
C_i : Peor tiempo de ejecución de la tarea i,
T_i : Período de la tarea i.

En el primer caso, PMC, la condición suficiente para asegurar que el conjunto de tareas tendrá una planificación factible es que el factor de utilización no supere un valor máximo dado por la siguiente expresión:

$$U \leq m\,(2^{1/m} - 1) \qquad (2)$$

Para un gran conjunto de tareas, es decir para $m \to \infty$, este resultado se acerca al 69%.
Si bien para el caso de MTV es posible llegar a un factor de utilización teórico del 100%, este esquema tiene asociados una mayor complejidad y un mayor costo en tiempo de ejecución. En vista de su simplicidad, sería deseable poder utilizar PMC, aunque con un mejor aprovechamiento de la capacidad de procesamiento. Por tal motivo, trabajos posteriores trataron de encontrar alguna manera de determinar si es posible asegurar una planificación factible, aún sin cumplir la condición suficiente expresada en (2). En tal sentido es de destacar el trabajo de Lehoczky, Sha y Ding[3], en el que se describe un método para lograr una caracterización exacta del conjunto de tareas cuya planificación se intenta analizar. Este método se basa en una serie de cálculos en base a la denominada *función trabajo*, la cual se calcula mediante la siguiente expresión:

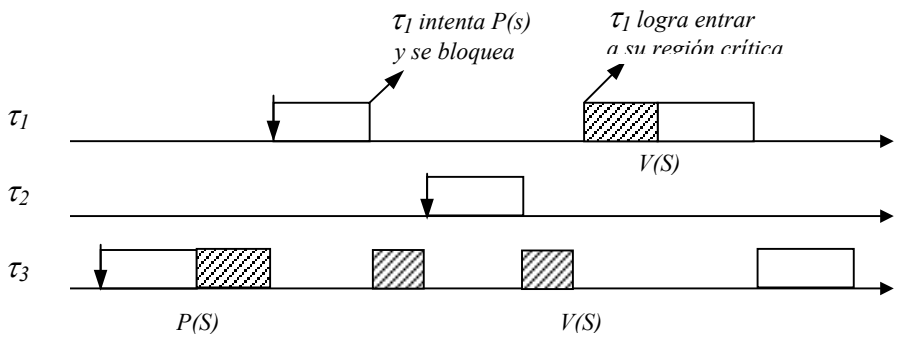
$$W_i(t) = \sum_{j=1}^{i} C_j \lceil t / T_j \rceil \qquad (3)$$

Donde: $\lceil x \rceil$ es el entero igual o mayor que x

La función trabajo mide la carga requerida al sistema por las i tareas de mayor prioridad hasta el instante t (midiendo el tiempo a partir de un instante crítico, es decir el requerimiento simultáneo de todas las tareas). La condición para que la planificación sea factible desde el punto de vista de los requerimientos de tiempo real se basa en el cálculo de:

$$L_i(t) = \frac{W_i(t)}{t}$$

Donde t pertenece a un conjunto de instantes de tiempo denominados "puntos críticos" del sistema, correspondientes a instantes en los que se produce el vencimiento de alguna de las i tareas consideradas. Si para todo i, tal que $1 \leq i \leq m$ es posible encontrar un instante t para el que $L_i(t) \leq 1$, entonces la planificación será factible.
Ahora bien, todas estas consideraciones se basan en la suposición de que las tareas son independientes, lo cual en sistemas reales puede no ser válido. Una situación típica se da cuando dos o más tareas comparten algún recurso (por ejemplo una variable de memoria), y a fin de evitar resultados inconsistentes durante la manipulación del mismo se debe asegurar que mientras una tarea tenga el derecho de acceder a dicho recurso compartido, otras tareas de mayor prioridad que eventualmente la puedan desalojar, no puedan acceder al mismo hasta que aquella lo haya liberado. Un mecanismo básico para el manejo de la exclusión mutua se basa en el uso de semáforos, que delimitan una porción del código denominada "sección crítica": una tarea debe *bajar* un semáforo antes de entrar y lo debe *levantar* al salir. Si al momento de intentar *bajar* un semáforo éste ya está bajo, la tarea que hizo el intento se bloquea y permanece en ese estado hasta que la tarea que tenía tomado el semáforo salga de la sección crítica y lo levante.
Si se analiza la situación recién planteada desde la perspectiva de los sistemas de tiempo real, se puede observar que un efecto negativo del uso de semáforos consiste en la inversión de prioridades, que puede llevar a que se extienda el tiempo de finalización de una tarea más allá de su tiempo límite de vencimiento. En la figura 1 se muestra un diagrama que representa la ocurrencia de una inversión de prioridad, en la cual la tarea de máxima prioridad, τ_1, debe esperar hasta que la tarea de mínima prioridad, τ_3, libere la región crítica. Mas aún, esa demora se ve acrecentada debido a que una tarea de prioridad intermedia, τ_2, desaloja a τ_3 mientras ésta se encuentra bloqueando a τ_1.



Las partes sombreadas corresponden a los intervalos de ejecución dentro de la región crítica.
P(S) significa bloquear el semáforo S.
V(S) significa liberar el semáforo S.

Fig. 1 Ejemplo de inversión de prioridad

En un sistema con *m* tareas, el ejemplo anterior se podría extender de tal modo que se produzca un encadenamiento de eventos que conduzca a que la tarea de mayor prioridad sufra una inversión por parte de las *m*-1 restantes. El problema es aún mayor si las tareas tienen requisitos de tiempo real, ya que es necesario asegurar el tiempo máximo de inversión de prioridad que podrá sufrir una tarea, a fin de estudiar la diagramabilidad del sistema. En el trabajo de Sha, Rajkumar y Lehoczky[1] se proponen dos soluciones en base a un esquema de herencia de prioridades. El objetivo de este trabajo es determinar para un determinado conjunto de tareas cual será el tiempo máximo de inversión de prioridad B_i que podrá sufrir cada tarea *i*. Una vez obtenido dicho valor se podrá analizar si la planificación será o no posible desde el punto de vista de tiempo real. Por ejemplo, la condición suficiente que para un conjunto de tareas independientes estaba dada por las expresiones (1) y (2), ahora se deberá de la siguiente manera:

$$\forall\ i,\ 1 \le i \le m \qquad \sum_{j}^{i-1} C_j / T_j \ + (C_i + B_i)/T_i \ \le \ i\,(\,2^{1/i} - 1\,)$$

Donde: i : número de orden de la tarea
m : cantidad de tareas

Esta expresión refleja el hecho de que para cumplir sus objetivos de tiempo real, una tarea τ_i se ve afectada por las i–1 tareas de mayor prioridad como así también por la inversión (B_i) que le pueden producir las restantes tareas de menor prioridad.

Asimismo, si se quisiera estudiar la diagramabilidad de un conjunto de tareas de tiempo real según la caracterización exacta propuesta por Lehoczky y otros[3], se deberá adoptar un criterio similar, es decir que se deberá considerar que el tiempo de ejecución de la tarea τ_i se ha ampliado en B_i unidades de tiempo.

La primer alternativa es en base al denominado "protocolo básico de herencia de prioridades", consistente en que mientras un proceso se encuentre bloqueando a otro de mayor prioridad, heredará la prioridad de aquel, a fin de poder finalizar cuanto antes la ejecución dentro de la sección crítica, sin interferencia por parte de tareas de prioridad intermedia. A fin de poder determinar la

duración máxima de inversión de prioridad que podrá sufrir cada tarea, se exige que en caso haber secciones críticas anidadas, las mismas estén propiamente anidadas, tal como se muestra en la figura 2. Un requisito adicional para este protocolo, a fin de evitar la posibilidad de ocurrencia de un abrazo mortal, es que se imponga un ordenamiento global para el anidamiento de secciones críticas.

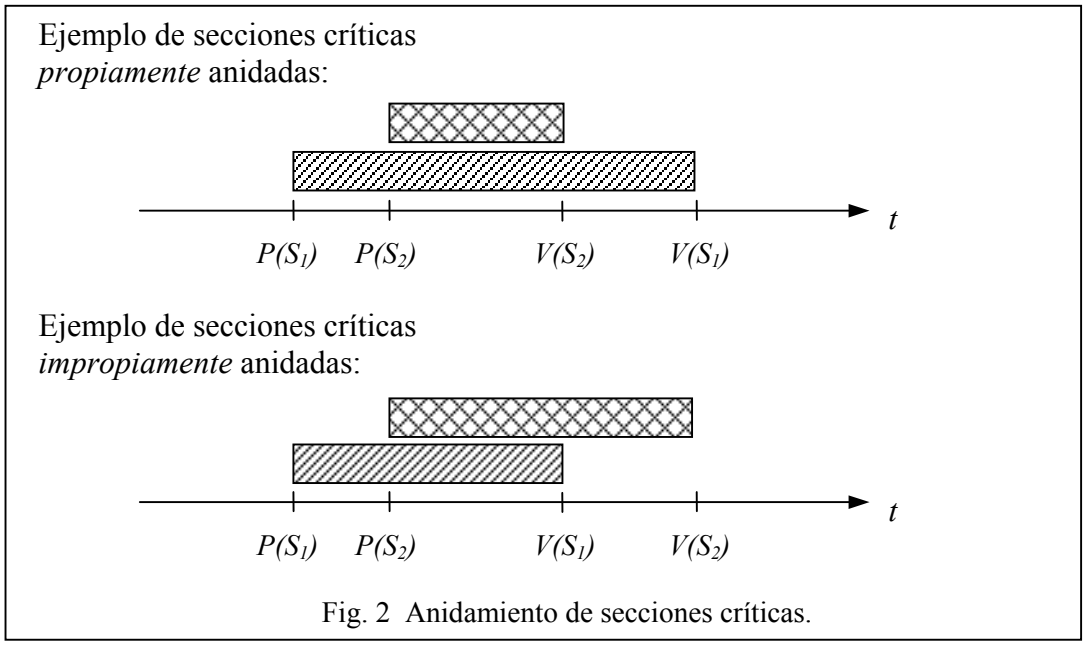


Fig. 2 Anidamiento de secciones críticas.

El tiempo máximo de inversión de prioridad que podrá sufrir una tarea, será igual a la suma del mayor tiempo de bloqueo que por cada semáforo le podrán imponer las tareas de menor prioridad. Por ejemplo, si una tarea τ_i tiene un anidamiento de secciones críticas como el siguiente:

P(1)...P(2)....(P3).....V(3).....V(2).....V(1)

El tiempo de bloqueo B_i será igual a: $B_i = máx\ B_{1,\ i-1} + máx\ B_{2,\ i-1} + máx\ B_{3,\ i-1}$

Donde: $máx\ B_{k,i}$ es el tiempo de bloqueo impuesto al semáforo *k* por parte de las tareas de prioridad menor o igual a *i*.

Si bien este protocolo permite establecer un cota superior a la duración de la inversión de prioridad que puede sufrir cada tareas, y en consecuencia estudiar la diagramabilidad del sistema, dicha cota podría ser muy elevada debido a que es una suma de tiempos de bloqueo. Esto es lo que se trata de mejorar mediante el protocolo de techo de prioridad (PCP: Priority Ceiling Protocol).

El objetivo del protocolo de techo de prioridad es prevenir la ocurrencia del "abrazo mortal" y los bloqueos encadenados. La idea de fondo es que si una tarea τ desaloja a otra tarea que está ejecutando en su sección crítica, e intenta ejecutar su propia sección crítica *Z*, la prioridad con la cual se ejecutará esta nueva sección crítica *Z* deberá ser *mayor* que las prioridades heredadas de todas las secciones críticas que se encuentran desalojadas. Si no se cumple ésta condición, la tarea

τ no podrá ingresar a su sección crítica Z y será suspendida, y la tarea que se encuentra bloqueando a τ heredará su prioridad.
Esta idea se implementa asignando a cada semáforo un "techo de prioridad", igual a la prioridad de la tarea de mayor prioridad que utiliza a dicho semáforo. Una tarea podrá acceder a una sección crítica sólo si su prioridad es mayor que la de todos los semáforos que se encuentren bloqueados por otras tareas.
Una importante propiedad de este protocolo es que el tiempo máximo que cualquier tarea τ podrá permanecer bloqueada por inversión, es igual al tiempo de duración de una sección crítica de alguna tarea de prioridad inferior a τ. Es decir, una tarea τ_i podrá permanecer bloqueada por inversión a lo sumo durante una sola de las secciones criticas compartidas con tareas de menor prioridad. Por lo tanto, el tiempo máximo de bloqueo por inversión que podrá sufrir una tarea τ_i será igual a la duración de la sección crítica de mayor duración impuesta por tareas de menor prioridad. Utilizando el mismo ejemplo anterior, en el que la tarea τ_i realizaba un anidamiento de tres secciones críticas según el siguiente esquema:

P(1)...P(2)....(P3).....V(3).....V(2).....V(1)

El tiempo de bloqueo B_i será igual a: $B_i = máx \{ máx\ B_{1, i-1} , máx\ B_{2, i-1} , máx\ B_{3, i-1} \}$

Donde: $máx\ B_{k,i}$ es el tiempo de bloqueo impuesto al semáforo *k* por parte de las tareas de prioridad menor o igual a *i*.

El protocolo de techo de prioridad intrínsecamente evita la ocurrencia del abrazo mortal, por lo que no es necesario imponer un orden global al anidamiento de secciones críticas. En cambio, se mantiene la necesidad de que las secciones críticas estén propiamente anidadas.

2. Manejo de semáforos provisto por los sistemas operativos.

En base a lo analizado en el punto anterior, se puede ver la conveniencia del protocolo de techo de prioridad para manejar la exclusión mutua en sistemas de tiempo real. Sin embargo, también se advierte que su implementación reviste un mayor grado de complejidad. Por otra parte, como para muchos sistemas de no tiempo real o de tiempo real blando puede ser suficiente el uso de semáforos comunes o a lo sumo semáforos con un manejo básico de herencia de prioridad, no es habitual que los sistemas operativos dispongan de semáforos con techo de prioridad.
En el punto siguiente se describe una extensión al manejo básico de semáforos provisto por el sistema operativo, que puede ser utilizada por aquellas tareas de tiempo real que requieran un manejo de la exclusión mutua mediante el protocolo de techo de prioridad.

3. Ampliando el manejo básico de semáforos.

En primer lugar, cabe tener presente que un semáforo es una variable de memoria compartida por distintos procesos. Por lo tanto, es necesario definir un área de memoria compartida en la que se encuentran los semáforos manejados según su techo de prioridad (semáforos PCP). Asimismo, en función de la prioridad de los procesos que utilizan cada semáforo, deberá establecerse el techo de prioridad de cada uno.

Los semáforos PCP se han organizado a modo de un arreglo, de tal manera que los procesos usuarios deben hacer referencia a los mismos por el número de orden de cada uno. Por ejemplo, para intentar el bloqueo del semáforo PCP número 5, la sentencia sería:

```
pcp_wait(5);
```

La solución que se ha desarrollado se basa en la utilización de dos tipos de módulos complementarios:

(a) Un proceso gestor de los semáforos de techo de prioridad (PCP_MANAGER), encargado de las siguientes tareas:

– Crear el área de memoria compartida, inicializar los semáforos PCP y otros semáforos y variables de uso común por parte de los procesos usuarios.

– Permitir a los procesos usuarios que se registren, indicando que semáforos PCP van a utilizar. En base a este registro se le asigna a cada semáforo su techo de prioridad, que es igual a la prioridad del proceso de mayor prioridad que lo vaya a utilizar.

– Permitir a los procesos que notifiquen que van a dejar de utilizar los semáforos PCP previamente declarados, a fin de ajustar el techo de prioridad de los mismos.

Este módulo corre como un proceso separado, a modo de servidor, que debe ser puesto en ejecución con anterioridad a los procesos que utilizan semáforos PCP.

(b) Un módulo que se linkea con cada aplicación, y que contiene las siguientes funciones para ser utilizadas por la aplicación:

- `attach_pcp( int * slist );`

  Esta función hace lo siguiente:

  – Se conecta con el área de memoria compartida.
  – Envía un mensaje al gestor (PCP_MANAGER), declarando la lista de semáforos que serán utilizados.

- `detach_pcp();`

  Esta función hace lo siguiente:

  – Se desconecta del área de memoria compartida.

  – Envía un mensaje al gestor (PCP_MANAGER), notificando que el proceso que la invoca deja de utilizar los semáforos PCP que tenía declarados.

- `pcp_wait( int ns );`

  Esta función es la que debe ser invocada por el proceso que requiera bloquear el semáforo PCP cuyo número es indicado en *ns*. El proceso que la invoca quedará bloqueado hasta tanto pueda concretar el bloqueo del semáforo solicitado. Esta función es la que implementa el protocolo de techo de prioridad, incluido el manejo de la herencia de prioridad.

- `pcp_post( int ns );`

  Esta función es la que debe ser invocada por el proceso que requiera desbloquear el semáforo PCP cuyo número es indicado en *ns*. Si el proceso que la invoca había heredado una prioridad mayor durante el tiempo de retención del semáforo *ns*, al liberar dicho semáforo el proceso en cuestión recuperará su prioridad anterior, por lo que podrá ser desalojado por otro proceso de mayor prioridad al que estaba invirtiendo.

3.1 Estructuras de datos

Como se ha señalado anteriormente, el módulo PCP_MANAGER crea un área de memoria compartida sobre la que se alojan los semáforos y otras variables de uso compartido. Asimismo, PCP_MANAGER maneja otro conjunto de datos en su área privada de memoria, que guarda relación con las variables en el área compartida. En la figura XX se muestra la organización de estas regiones de memoria.

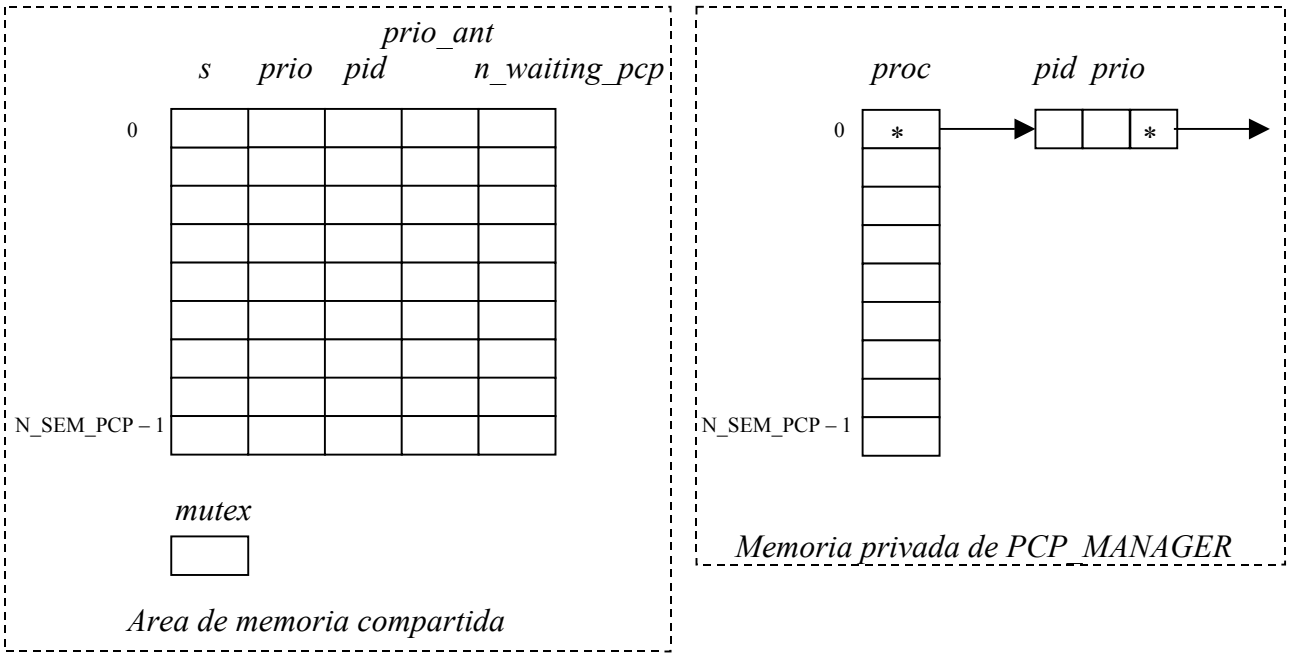


Figura 3. Estructuras de datos usadas por el gestor y los procesos.

En el área compartida hay una tabla en la que cada fila corresponde a un semáforo PCP. Las columnas son las siguientes:

- *s* : semáforo PCP (es un semáforo común).
- *prio* : techo de prioridad asignado al semáforo *s*.
- *pid* : identificador del proceso que mantiene bloqueado el semáforo *s*.
- *prio_ant* : prioridad que tenía *pid* al momento de ganar el bloqueo de *s*.
- *n_waiting_pcp* : semáforo que lleva la cuenta de los procesos que están durmiendo a la espera de este semáforo.

- *mutex* : semáforo usado para lograr la exclusión mutua durante los accesos a la región compartida.

N_SEM_PCP es la cantidad máxima de semáforos PCP de que se dispone.

Por otra parte, en su sección privada de memoria el gestor PCP_MANAGER va construyendo una lista de los procesos que se han registrado para usar determinados semáforos PCP. Hay una lista de procesos por cada semáforo PCP, y para cada proceso se registra su identificador (*pid*) y su nivel de prioridad (*prio*). El techo de prioridad del semáforo PCP número *n* es la máxima prioridad almacenada en la lista de procesos que se encuentra a partir de *proc[n]*.
Cabe tener presente que las particularidades de estas estructuras de datos, como así también las cuestiones internas relativas al manejo de los semáforos quedan ocultas a las aplicaciones, las cuales disponen de una interfaz sencilla para la utilización de los semáforos, como se puede ver en el ejemplo del inciso 2.3.

3.2 Algunos detalles del funcionamiento.

Cuando un proceso desea obtener el bloqueo de un determinado semáforo PCP, efectúa una invocación a la función *pcp_wait(<n>)* , pasándole como parámetro el número de semáforo que desea bloquear. Las condiciones para poder bloquear un semáforo PCP son las siguientes:

(i) El semáforo no debe estar bloqueado
(ii) No debe haber ningún semáforo de prioridad mayor o igual que ya se encuentre bloqueado por otro proceso.

En caso de no cumplirse la condición (ii), se llevarán a cabo las siguientes acciones:

- Se buscará el proceso de menor prioridad que esté ocasionando el bloqueo descripto en (ii),
- Se elevará su nivel de prioridad, llevándolo al nivel de prioridad del proceso solicitante.
- El proceso solicitante incrementará la cuenta de procesos durmiendo a la espera de la liberación de este semáforo (*n_waiting_pcp*), y se irá a dormir en el semáforo *s*.

Cabe acotar que cuando el semáforo se libere, el proceso solicitante será despertado y se volverá a repetir el análisis anterior (i y ii). Esto da lugar a que el proceso de mayor prioridad de entre los que están compitiendo por el semáforo, tenga precedencia para intentar el bloqueo.

Por otra parte, cuando un proceso decide liberar un semáforo PCP que tenía bloqueado, debe efectuar una llamada a *pcp_post(<n>)*, lo cual da lugar a que internamente se desarrollen las siguientes acciones:

- Se levanta el semáforo *s*.
- Por cada proceso durmiendo en este semáforo se efectúa una levantada adicional del semáforo *s*.
- Se restaura la prioridad anterior del proceso que efectúa la liberación.

3.3 Aspectos prácticos de la utilización de semáforos con techo de prioridad.

Es interesante ver algunos detalles prácticos de la utilización de los semáforos con techo de prioridad. Cabe tener presente que de acuerdo al análisis desarrollado en [1], el protocolo de techo de prioridad tiene inherentemente resuelto el problema del interbloqueo; mientras que el protocolo básico de herencia de prioridades requiere respetar un ordenamiento global para el anidamiento de secciones críticas, el protocolo de techo de prioridad solamente requiere que las secciones críticas estén propiamente anidadas, aunque no interesa que distintos procesos realicen el anidamiento en diferente orden. En el siguiente ejemplo se ve el caso de un programa que va utilizar tres semáforos PCP, cuyos números son 2, 3 y 5. Los semáforos 2 y 5 se usan en forma anidada, mientras que el 3 se usa en forma individual.

```
#include ......     // Archivos de encabezado requeridos por la aplicación.
#include "pcp.h"    // Archivo de encabezado con las declaraciones y
                    // definiciones del módulo de manejo de semáforos PCP
void tareaXX()
{
       // Lista de semáforos PCP a utilizar. El primer número indica la cantidad de
       // semáforos, los restantes son los identificadores de cada semáforo.
       static int slist={3, 2, 3, 5};
       .....
       // Declaración de los semáforos PCP que va a utilizar
       if( attach_pcp(slist) == -1 )
              exit(1);     // error
       .....
       while( .... )
       {
              // Código fuera de la sección crítica
              ............

              // Secciones críticas anidadas (2 y 5)
              // Intenta bloquear el semáforo 2
              wait_pcp(2);
              ............
              // Intenta bloquear el semáforo 5
              wait_pcp(5);
              ............
              // Libera el semáforo 5
              post_pcp(5);
              ............
              // Libera el semáforo 2
              post_pcp(2);
              ............

              // Sección crítica 3  → Intenta bloquear el semáforo 3
              wait_pcp(3);
              ............
              // Libera el semáforo 3
              post_pcp(3);
       }
       // Notifica que deja de utilizar los semáforos PCP que tenía declarados.
       detach_pcp();
       exit(0);
}
```

4. Ensayos realizados.

A fin de evaluar las mejoras introducidas por los semáforos PCP se ha realizado una serie de pruebas, mediante programas desarrollados al efecto, que fueron ejecutados sobre sistema operativo QNX Neutrino. Los semáforos han sido utilizados para controlar acceso de varias tareas a sus regiones críticas, las cuales tienen diferentes esquemas de anidamiento. Asimismo, se ha comparado el resultado obtenido utilizando los semáforos PCP con respecto al resultado obtenido usando los semáforos básicos provistos por el sistema operativo.
Si bien se podría efectuar una infinidad de pruebas diferentes, con mayor o menor cantidad de tareas, con diversas formas de anidamiento de las secciones críticas, con diferentes secuencias de requerimientos de las tareas, etc., es interesante analizar los resultados obtenidos en algunas situaciones particulares, cuya problemática es conocida de antemano.
En particular, cabe destacar los resultados obtenidos al hacer ejecutar el conjunto de tareas que se detalla en la tabla siguiente, usando uno y otro tipo de semáforos. En dicha tabla se enumeran las tareas en orden de prioridad decreciente, y se detallan los semáforos usados por cada una.

Semáforo / Tarea	S_1	S_2	S_3	S_4	S_5
τ_1	√		√		√
τ_2		√		√	
τ_3		√	√		
τ_4		√		√	
τ_5	√				√

En la implementación se ha considerado que cada tarea se ejecute a partir de un requerimiento, y luego de completada su ejecución, la tarea quede bloqueada la espera de un nuevo requerimiento. A continuación se detalla el orden en que cada tarea intenta bloquear y liberar los semáforos:

τ_1 : .....P(1)....P(3).....P(5).....V(5).....V(3).....V(1).....Fin
τ_2 : .....P(2)....P(4).....V(4).....V(2).....Fin
τ_3 : .....P(3)....P(4).....V(4).....V(3).....Fin
τ_4 : .....P(4)....P(2).....V(2).....V(4).....Fin
τ_5 : .....P(5)....P(1).....V(1).....V(5).....Fin

El siguiente diagrama temporal muestra el orden de llegada de los requerimientos para cada tarea, y la secuencia de acciones desarrollada por cada una. La notación utilizada es la siguiente:

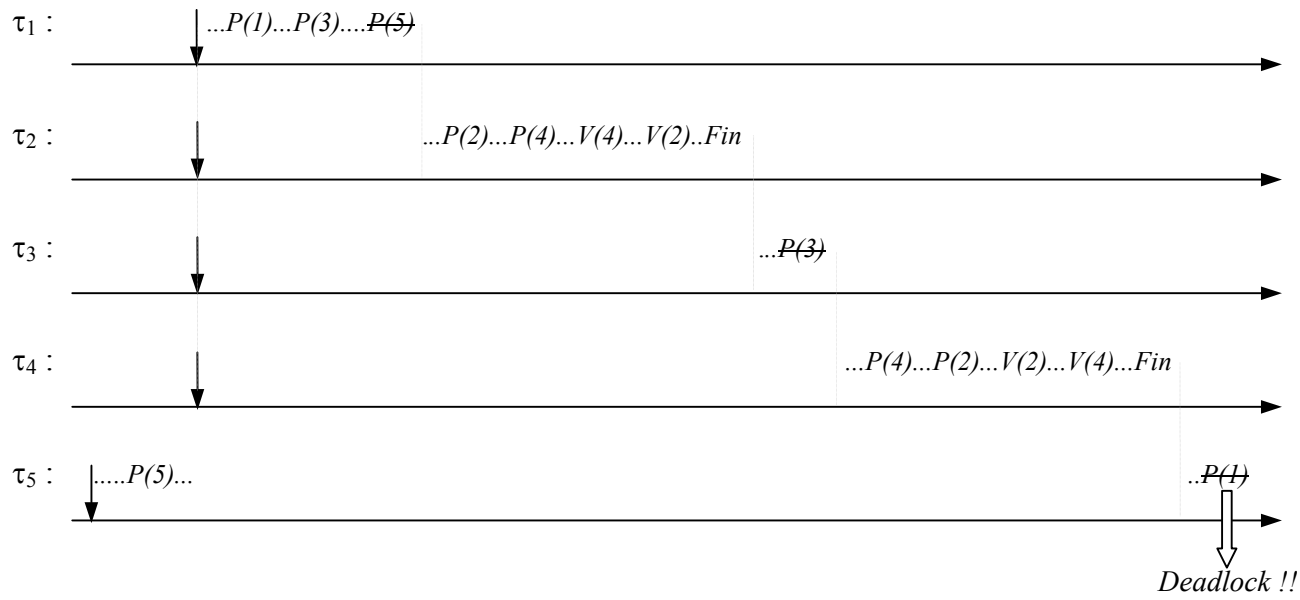
P(n) : bloqueo del semáforo *n*

~~*P(n)*~~ : intento fallido de bloquear el semáforo *n*

V(n) : liberación del semáforo *n*

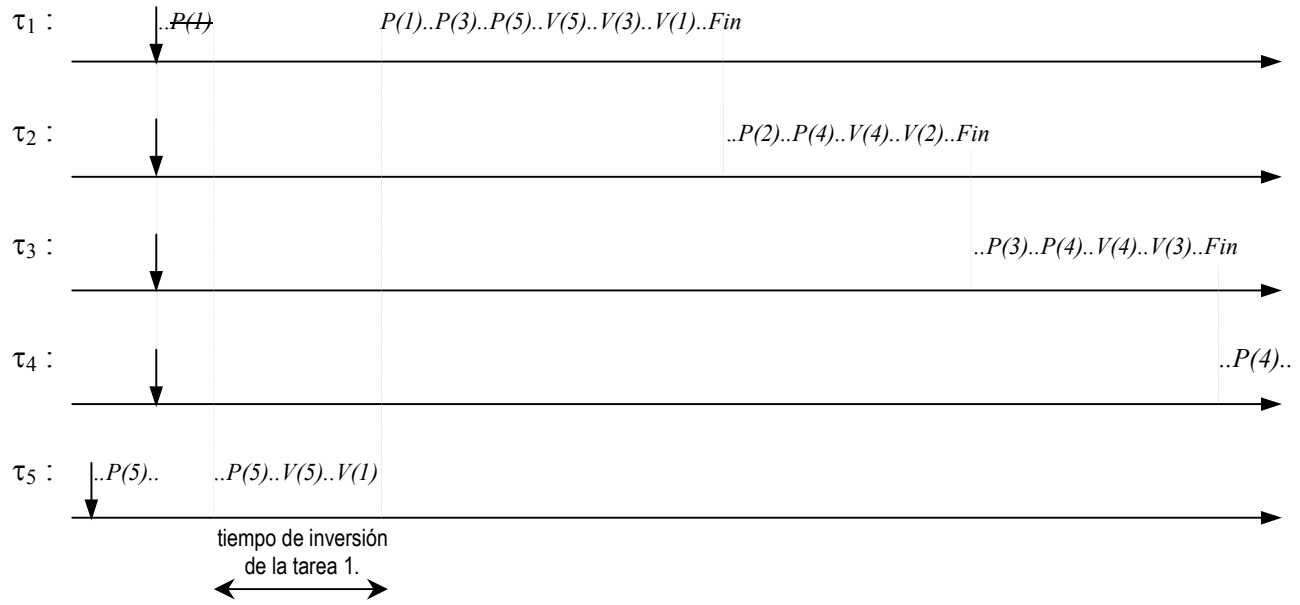
↓ : llegada de un requerimiento para una tarea

➢ Caso 1: Usando los semáforos normales de QNX.



Como se puede apreciar, el resultado de ejecutar este conjunto de tareas con esta secuencia de requerimientos no sólo ha dado lugar a una inversión de prioridades descontrolada (porque estos semáforos no manejan herencia de prioridad), sino que además ha dado lugar a un interbloqueo entre las tareas τ_1 y τ_5 (debido a no haberse adoptado un orden global para el anidamiento de las secciones críticas). Por otra parte, en el siguiente gráfico se puede apreciar el resultado obtenido usando los semáforos PCP.

➢ Caso 2: Usando semáforos PCP.



Si bien, por razones de espacio, en el gráfico anterior no se alcanza a visualizar el final de la ejecución de las τ_4 y τ_5, se puede apreciar que los semáforos PCP han evitado la ocurrencia de interbloqueos, y han reducido al mínimo posible el tiempo de inversión sufrido por la tarea 1.

5. Conclusiones.

Los sistemas de tiempo real pueden requerir un manejo de sus secciones críticas que permita predecir el tiempo de bloqueo por inversión de prioridad que puede sufrir cada tarea, al mismo tiempo que permita mantener dichos tiempos de inversión lo más breves posibles. Un esquema eficaz para resolver este problema es el protocolo de techo de prioridad. En el presente trabajo se ha descripto un par de módulos de software que han sido desarrollado como una extensión al manejo básico de semáforos provisto por el sistema operativo. Dichos módulos le brindan a las aplicaciones la posibilidad de usar un conjunto de semáforos manejados por techo de prioridad (semáforos PCP). Uno de estos módulos es el encargado de la creación y gestión de los recursos necesarios, y el otro es un módulo que se linkea con los programas que lo requieran. Este segundo módulo dispone de una función para declarar los semáforos PCP que la tarea va a usar, y las funciones para el bloqueo y liberación de los mismos, a la entrada y salida de una sección crítica, respectivamente. Se ha realizado y probado una versión experimental en lenguaje C, sobre QNX Neutrino, aunque la misma puede ser fácilmente adaptada a otros sistemas operativos.

Referencias

[1] Sha, Rajkumar y Lehoczky, "Priority Inheritance Protocols", IEEE Transactions on Computers, vol. 39, Nº 9, Sept. 1990, págs. 1175-1184.

[2] Liu & Layland, “Scheduling algorithms for multiprogramming in a hard real-time environment”, Journal of the ACM, Vol.20 Nº 1, Jan. 1973, pp. 46-61

[3] Lehoczky, Sha & Ding, “The rate monotonic scheduling: Exact Characterization And Average Case Behavior”, Proceedings IEEE, May 1989, pp. 166-171.

[4] QNX Software System Ltd, "QNX OS, ver. 6.1.0 System Architecture", accesible en formato electrónico en: http://www.qnx.com/developer/docs/qnx_6.1_docs/neutrino/sys_arch/about.html