Usando una extensión al problema de los Filósofos que Cenan para estudiar Sistemas de Tiempo Real Duro.

Gabriel A. Wainer
Departamento de Computación
Facultad de Ciencias Exactas y Naturales.
Universidad de Buenos Aires.
Charcas 3960 7mo. 44. (1425). Capital Federal.
gabrielw@dc.uba.ar

Resumen

En este trabajo se presentan los resultados del uso de un problema clásico para estudiar algunas dificultades existentes en el desarrollo de Sistemas de Tiempo Real Duro. Se analiza una extensión a un problema usado para analizar sistemas concurrentes y se adapta para estudiar algoritmos tradicionales de planificación en tiempo real. Los resultados obtenidos muestran que su uso permite analizar el comportamiento de algoritmos de planificación tradicionales, comprender los servicios que debería proveer un entorno de programación para tiempo real, y adquirir alguna experiencia de implementación de sistemas de tiempo real usando entornos de programación tradicionales.

Palabras clave: tiempo real, planificación de procesos, programación concurrente.

1. INTRODUCCION

Cada vez es más común el uso de computadoras para hacer control directo de dispositivos del ambiente que las rodea. En un extremo, hay dispositivos sencillos como el controlador de un lavarropas; en el otro extremo hay, por ejemplo, sistemas de control y seguridad de una planta nuclear, pasando por fábricas automatizadas, aplicaciones de aviónica, exploración submarina, sistemas de robots y visión, así como aplicaciones militares complejas.

El funcionamiento correcto de estos sistemas no sólo depende del resultado de los cálculos, sino también del momento en que se producen (resultados con retraso pueden no ser válidos en el momento de utilizarlos). Por este motivo se dice que estos sistemas deben actuar en "tiempo real". Otra característica de estos sistemas es que deben responder a los eventos del ambiente a medida que ocurren.

Hay casos en los que la respuesta a tiempo es importante pero se pueden tolerar pequeños atrasos, y se conocen como ambientes de tiempo real "blando" (cajeros automáticos, sistemas de reserva de pasajes, etc.). En otros casos, si no se cumplen estrictamente las restricciones de tiempo, puede haber consecuencias catastrófi-cas: es el caso de los ambientes de tiempo real "duro" (sistemas para control industrial, tráfico aéreo, defensa, etc.). En estos, correctitud y desempeño están fuertemente relacionados.

Estas características hacen que, para este tipo de aplicaciones el uso de programación secuencial no sea adecuado, ya que las aplicaciones son inherentemente paralelas. Un programa secuencial debería diseñarse cuidadosamente para cumplir las restricciones de tiempo, lo cual es muy complejo debido a las diferencias de tiempo entre los componentes del sistema. Esta complejidad suele acotarse dividiendo el problema en un conjunto de tareas que se ejecutan concurrentemente. También es bueno disponer de mecanismos de planificación por prioridades para reflejar la importancia de las distintas tareas.

En la actualidad existe gran variedad de mecanismos para programación concurrente, pero estos no son del todo adecuados para el desarrollo de sistemas de tiempo real duro, ya que la semántica de las construcciones suele considerar que los eventos ocurren en algún momen-to, sin considerar su orden exacto de ejecu-ción. El panorama es aún más complejo en sis-temas de multiprocesamiento, y distribuidos.

En este trabajo mostramos los resultados de una experiencia desarrollada desde 1993 en un curso de Sistemas de Tiempo Real en nuestro Departamento, usando un problema clásico de programación en tiempo real. El problema es-

tudiado permite que programadores no expertos puedan analizar diversas características de sistemas de tiempo real complejos, considerando distintas formas cumplir las metas de las tareas. También nos permite enfrentar el problema de desarrollo de estos sistemas usando servicios de programación tradicionales, facilitando el estudio de nuevas soluciones.

2. ALGUNOS PROBLEMAS GENERALES

Como dijimos, una característica primordial de los sistemas de tiempo real es que cumplan las restricciones de tiempo de las tareas. El procesamiento concurrente con prioridades parece ser la base para lograrlo, pero con estos servicios no alcanza, ya que además una tarea también puede tener otras restricciones que deben considerarse al analizar la funcionalidad del sistema: restricciones de recursos, de precedencia, concurrencia, comunicación, ubicación y criticidad [Sta93].

Es claro que el uso de procesamiento concurrente con prioridades no basta para cumplir con estas restricciones. Todas ellas deberían mapearse en un único valor: la prioridad de la tarea. A pesar de esto, muchos diseñadores siguen usando esta aproximación debido a que la mayoría de los entornos de programación no proveen otros servicios. En estos casos, el cumplimiento de las restricciones del sistema se logra por medio de simulaciones y chequeo exhaustivo, lo que trae acarreado problemas de mantenibilidad y aumento de costos (ya que cualquier cambio en el sistema implica una nueva ronda de chequeos). Se requieren soluciones generales que permitan evitar estos problemas, y permitan asegurar la planificabilidad de un conjunto de tareas [Wai95b].

Nuestro objetivo es lograr que individuos no expertos en la materia estudien algunas soluciones existentes con esta finalidad, reconozcan las características generales del problema (y de estas soluciones), y producir interés para plantear nuevas propuestas. Para ello tomamos como base un trabajo clásico de Dijkstra [Dij65], que plantea un problema general de sistemas concurrentes desde un punto de vista teórico. El problema, conocido como “Los Filósofos que Cenan” puede plantearse como sigue:

Hay cinco filósofos chinos que pasan sus vidas pensando y comiendo. Comparten una mesa circular, alrededor de la cual se sientan. En su centro hay un tazón con una provisión infinita de arroz, y sobre ella hay cinco palitos, uno a cada lado de cada filósofo. Cuando un filósofo piensa, no interactúa con sus colegas. De vez en cuando, un filósofo tiene hambre y trata de levantar los dos palitos más cercanos a él. Un filósofo puede levantar un palito a la vez, y no puede tomar un palito que ya está en la mano de un vecino. Cuando un filósofo tiene ambos palitos, puede

comer. Cuando terminó de hacerlo, deja sus dos palitos y comienza a pensar de vuelta.

Este problema tiene variedad de soluciones con el fin de evitar problemas tales como deadlock, inanición y livelock de los procesos que cooperan (los "filósofos"). Nuestro trabajo se basa en una extensión a este problema, llamada "Los Filósofos que Cenan y Mueren" [Sta93], que permite análizar procesos con restricciones de tiempo, y comprender el problema de asignación de recursos en sistemas de tiempo real. Aquí cada filósofo tiene un momento en el cual debe comenzar a comer, y si no lo hace, muere de hambre. Existe un "maitre" que, cuando esto ocurre, sienta automáticamente a otro filósofo.

Ninguna de las soluciones propuestas para el caso original sirve para esta extensión, ya que este problema es completamente diferente del anterior. El problema original sólo estudia la asignación de recursos (seguridad, correctitud), mientras que la nueva versión debe considerar la asignación a tiempo de los mismos (temporalidad). Si hay n filósofos en la mesa, existen n! formas de levantar los palitos, algunas de los cuales pueden provocar que uno o más filósofos mueran, y que los otros vivan. Por ende, es necesario establecer un orden específico para comer. El problema original no considera el orden en el cual las cosas ocurren, sino que sólo le importa la noción de que los recursos se asignen en algún momento. En cambio, la solución al nuevo problema debe considerar el orden específico de eventos y su tiempo absoluto.

Nuestra idea fue usar el problema recién planteado desde un punto de vista experimental, y adaptarlo para analizar algunas técnicas de planificación conocidas. En lugar de plantear la solución directa al problema se diseñará e implementará un "ejecutivo" que haga planificación de procesos (corriendo sobre algún entorno de programación), y se lo usará para implementar distintas soluciones al problema. Esta es una aproximación muy utilizada, sugerida, por ejemplo, en [Bak86], y revisitada recientemente en [Ade94]. De esta forma también puede hacerse un primer acercamiento al diseño de núcleos multitarea para tiempo real.

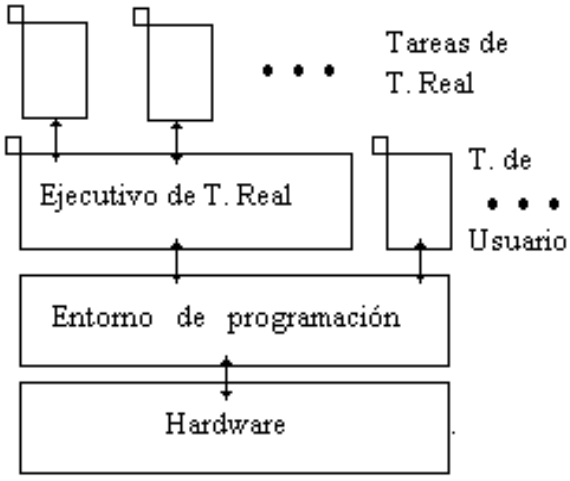

Figura 1. Esquema propuesto

Al lanzar un filósofo, se debe informar un conjunto mínimo de datos sobre el mismo, de forma tal que al comenzar la ejecución del conjunto de tareas, el planificador tome esta información y ejecute rutinas de estudio de garantía

para asegurar si las tareas pueden ejecutar de forma predecible.

La misión primordial del ejecutivo será controlar el orden de ejecución de los filósofos, de acuerdo con algún algoritmo elegido. Debe-rá utilizar los servicios del entorno de programación para demorar las tareas, sincronizarlas, y determinar su orden adecuado de ejecución. Además, será el encargado de detectar si una tarea ha perdido su meta, y en tal caso lanzar un nuevo filósofo, recolectando información sobre distintas métricas. El ejecutivo deberá proveer servicios extendidos a los del entorno de programación, que mostramos a continuación:

```
lanzar_filósofo(número, info_filósofo);
inicializar_métricas();
metricas = recuperar_métricas();
demorar_filósofo (número);
reiniciar_filósofo(número);
seleccionar_algoritmo(cod_planif);
```

Tabla 1. Algunos servicios del ejecutivo

Como en el problema original sólo se considera asignación de recursos, el tiempo demorado por cada filósofo para pensar no tiene especial importancia, a no ser para analizar cuestiones de seguridad y correctitud. Esto no es válido para tareas de tiempo real, ya que el tiempo demorado por cada filósofo puede influir en la adquisición a tiempo de los recursos necesarios. También difieren los casos de plantear el problema para uno o varios procesadores. Para poder analizar con detalle el caso de planificación centralizada y establecer la influencia de usar un único procesador, consideramos que cada filósofo, mientras piensa, toma notas en un anotador compartido por todos y que los filósofos no pueden filosofar sin tomar nota de sus ideas (en el caso de tener que analizar el caso de múltiples procesadores, podemos considerar que existen tantos anotadores como procesadores). El planificador es quien debe decidir qué filósofo toma el anotador en cada momento.

Habiendo construido el ejecutivo, deberá usarse para programar los filósofos. Para ello consideramos que cada filósofo ejecuta siguiendo un modelo tradicional de planificación de tareas en tiempo real, conocido como modelo de tareas periódicas. En éste se considera la existencia de dos tipos de tareas: las periódicas y las aperiódicas. Las tareas periódicas tienen múltiples instancias de activación a una frecuencia dada (el período de la tarea). Suelen tener meta al comienzo de su próximo período (aunque en otros casos se aceptan metas anteriores), y un peor caso de ejecución (T1 y T2 en la figura). Las tareas esporádicas son de única instancia, y con activación aleatoria. Tienen una meta y un peor caso de ejecución asociados (E1 en la figura).

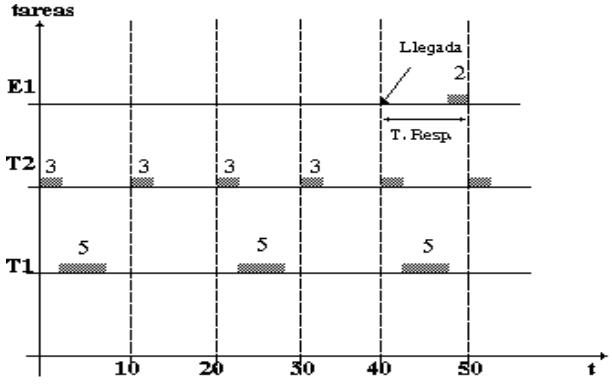

Figura 2. Modelo de tareas periódicas

En nuestro caso se sugiere considerar que cada filósofo ejecutará como una tarea periódica, con meta al comienzo de "hora de comer", construído como un servidor que piensa, come, y se va a "dormir", esperando que el ejecutivo lo despierte para comenzar un nuevo ciclo.

Además se pide considerar la existencia de un "mozo", que ejecuta como una tarea aperiódica, levantando uno por uno los palitos y limpiándolos. El mozo también usa el anotador (para apuntar qué palitos ha limpiado y cuales no), y tiene una restricción de dura tiempo para hacer su tarea (si no la cumple, es despedido).

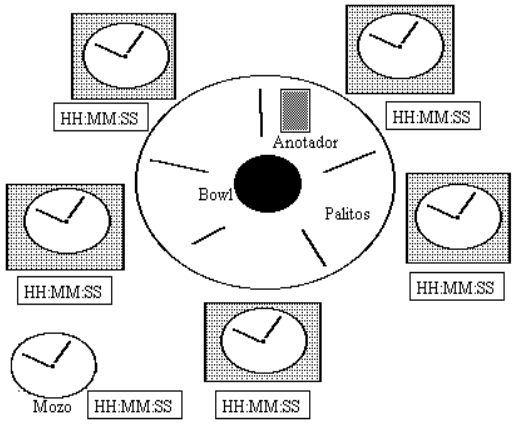

Figura 3. Esquema del problema extendido

Habiendo construido el sistema completo, se estudiará su comportamiento, poniendo énfasis en métricas significativas (porcentaje de éxito de metas cumplidas, análisis de sistemas con sobrecargas, estabilidad, tiempos ociosos). También se estudiará la influencia del uso de los distintos planificadores para tiempo real, y de cuestiones de implementación (qué problemas se pueden solucionar y cuáles no con un entorno multitarea tradicional).

3. ALGUNOS RESULTADOS

Hasta el momento se han realizado variedad de implementaciones del problema, usando diversos entornos (OS/2, QNX y varias versiones de Unix, incluyendo Xenix, Minix, Linux, SCO, AIX y Solaris), usando diferentes servicios de cada uno de ellos. También se han realizado experiencias con una versión en tiempo real del sistema operativo Minix [Wai95a], con la cual no es necesaria la construcción del ejecutivo, ya que se proveen primitivas de tiempo real. Por ende, la discusión a continuación sólo tiene sentido en algunos de los casos analizados a continuación. Pudo mostrarse que el desarrollo de este problema usando un sistema operativo para tiempo real trae aparejada una reducción en los tiempos de desarrollo (especialmente en la fase de chequeo) y mantenimiento.

Una de las primeras cuestiones a considerar es cómo implementar la mesa con los palitos, y cómo lograr sincronización en su acceso. Se han usado diversas aproximaciones, incluyendo el uso de semáforos con memoria compartida, pasaje de mensajes y pipes. Otras cuestiones se relacionan con implementación de los filósofos, y su sincronización. En general las tareas (ya sean procesos independientes o procesos livianos -threads-), son sincronizados por medio de pasaje de mensajes, pipes o estimulación cruzada (señales). Para implementar el ejecutivo se han usado varios mecanismos de comunicación y sincronización (mensajes, estimulación cruzada, pipes), y diversos servicios de tiempo (clock, alarmas, demoras y servicios similares), así como servicios relacionados con el planificador de procesos del sistema operativo (cambio de prioridad, variación del quantum del sistema, etc.). En general el planificador utiliza una tabla de control de procesos, en la que se controla la información básica de cada filósofo: hora de comienzo de la instancia actual, meta, estado (comiendo, pensando, durmiendo), hora de próxima comida, y la hora en la que debe activarse su próxima instancia, entre otros.

Hemos puesto énfasis en la implementación de algunos algoritmos clásicos para hacer planificación de tareas, seleccionables en tiempo de ejecución, por ejemplo Tasa Monotónica (TM), Meta Anterior Primero (MAP) [Liu73] y Menor Flexibilidad Primero (MFP) [Mok83] entre otros

En cuanto a los análisis realizados, el prin-cipal foco está puesto en estudiar las pérdidas de metas de las tareas. Los resultados han sido muy variados debido a la diversidad de implementaciones, pero siempre se ha podido comprobar empíricamente resultados teóricos conocidos. A continuación resumimos algunos de los resultados obtenidos (los interesados en obtener mayor información de los resultados de versiones en particular, o la implementación de algún caso, pueden contactar al autor).

En un primer caso se estudian los filósofos considerando que su tiempo para comer es cero, lo que neutraliza los problemas existentes por demoras en el uso de los recursos (palitos), y el problema es el de planificar un modelo de tareas periódicas tradicional. La idea es hacer estudio exhaustivo de los algoritmos recién mencionados (que consideran que las tareas son independientes entre sí, y que no tie-nen restricciones de precedencia). Pudo verse que, en el caso de que se cumplan las cotas teóricas de los algoritmos, todas las tareas cumplen sus metas.

El panorama cambia totalmente cuando hay dependencias entre las tareas, ya que ninguno de estos algoritmos están preparados para soportarlas. Lo mismo ocurre al considerar sistemas sobrecargados.

En este caso la detección de las pérdidas de metas puede ocurrir en dos momentos: al comer o al pensar. La detección en el momento de pensar ocurre cuando el conjunto de tareas no es planificable (las entradas/salidas no influyen) o cuando una entrada/salida provoca retención de los palitos, demorando el comienzo de una instancia. Estas pérdidas de metas también ocurren cuando el planificador del procesador del entorno de programación provoca un orden de ejecución diferente al ejecutado por el planificador de alto nivel (lo que ocurre, por ejemplo, en todos los sistemas con remoción por reloj en los casos en que el peor caso de ejecución de las tareas es mayor al quantum). Las pérdidas de metas ocurren al comer provocadas por las dependencias entre las tareas, ya que el modelo de tareas elegido no considera la asignación de recursos. Pudo detectarse variedad de casos en los que el anotador queda libre, pero ningún filósofo lo usa, bloqueados en espera de obtener un palito o esperando su próximo período. Esto produce pérdidas de metas aunque haya tiempo disponible de procesador.

También es interesante analizar la influencia de los algoritmos de planificación elegidos, ya que estos cumplen o no las metas dependiendo del conjunto de tareas que se quiera ejecutar. Se pudo detectar variedad de casos en los que MFP/MAP son mejores que TM (debido a la existencia de tareas que se comportan mejor con cálculo dinámico de prioridades), mientras que en muchos otros casos se pudo ver que algoritmos como MFP/MAP producen pérdidas de metas en cascada (y que MFP permite lanzar tareas alternativas de forma más simple, ya que si una tarea tiene flexibilidad negativa, sabemos que no podrá cumplir su meta). También pudo estudiarse con detalle el overhead extra en los algoritmos con prioridades dinámicas que, en algunos casos (con tareas con metas muy duras) provocan pérdidas de metas.

Pudo analizarse cómo las tareas esporádicas alteran la ejecución de las periódicas, así como algunos problemas al compartir recursos. Para los análisis de planificabilidad, el mozo se analiza como si fuera una periódica más, tomando su meta como si fuera su período (y haciendo cálculos de planificabilidad on-line). Los resultados son esencialmente similares a los presentados en [Wai95c] y [Wai96].

Como en el caso tradicional, puede haber problemas de deadlock, livelock o inanición. Si el algoritmo elegido permite encontrar un orden de planificación estático y predecible que considere la asignación de recursos, no será necesario estudiar estos problemas de seguridad y correctitud, ya que el orden de ejecución elegido asegura la planificabilidad del conjunto de tareas, y establece un orden completo para hacerlo. Como los planificadores seleccionados no usan esta aproximación, estos problemas fueron solucionados con sus aproximaciones tradicionales (asignación de recursos en orden, uso de semáforos de exclusión mutua y pedido simultáneo de

recursos, protocolos de asignación que rompan la espera circular, etc.), lo que altera el orden de ejecución de los planificadores utilizados.

Una última cuestión se relaciona con el uso de un entorno de programación tradicional para resolver estos problemas. En nuestro caso encaramos el uso de un sistema tradicional de tiempo compartido para el desarrollo de la experiencia, ya que en muchos casos siguen usándose estos entornos, y la resolución de este problema usando este enfoque puede otorgar alguna habilidad de desarrollo. Esta aproximación nos permite identificar las facilidades que debería proveer un sistema operativo para tiempo real, y analizar cómo hacer para cumplir las restricciones de tiempo con un sistema operativo tradicional.

Aunque en la actualidad existe una gran variedad de sistemas operativos, estos suelen ser de propósito demasiado general, y demasiado complejos para ser útiles en aplicaciones de tiempo real. Nuestra aproximación nos permite estudiar algunas de estas falencias: el overhead impuesto por el sistema, la forma en que este elige las tareas (en general, distinta a la del algoritmo de tiempo real elegido para el ejecutivo), y la influencia de las interrupciones y llamadas al sistema.

Se pudo estudiar con detalle muchas características de los problemas a resolver, acotando las dificultades mediante el uso adecuado de dicho entorno. Una conclusión obvia es que los entornos con prioridades con remoción son los mejores para construir el ejecutivo (ya que muchas de las aproximaciones usadas se basan en algoritmos por prioridades). Existen problemas especiales con respecto a los timeouts existentes debido a su utilización para tiempo compartido. Estos timeouts pueden provocar inversiones de prioridades ya que una tarea de alta prioridad puede ser desalojada por el reloj. En el mejor de los casos, si el algoritmo de planificación del sistema operativo usa prioridades y la tarea vuelve a ejecutar luego de ser desalojada, se introduce un overhead que puede provocar pérdidas de metas. Por ende, en estos casos es bueno que el sistema operativo provea la facilidad de variar la granularidad del reloj.

En todos los casos es imperativo no incluir más tareas que las del sistema a desarrollar (la máquina debe estar dedicada), ya que en caso contrario las teorías sobre las cuales se basa la construcción del ejecutivo no serían útiles. Esto, en la mayoría de los casos es muy complejo, ya que debe asegurarse que no haya corriendo tareas en background, ni otros usuarios en el sistema. Si es necesario que existan más tareas, es indispensable tener un mecanismo de planificación basado en prioridades, y dar a estas tareas prioridad mas baja que a las tareas de tiempo real (para que ejecuten en los tiempos ociosos de procesador). Otro problema está relacionado

con remoción del núcleo: muchos sistemas no liberan el procesador si una tarea está ejecutando una llamada al sistema o una rutina de atención de interrupciones, lo que puede provocar inversiones de prioridades. Una tarea de alta prioridad puede verse demorada por una tarea de muy baja prioridad que pide un servicio al Sistema Operativo (o si se está atendiendo una interrupción con baja prioridad con respecto a las tareas de tiempo real), y provocar una demora significativa.

Un último problema está relacionado con la imposibilidad de relacionar restricciones de tiempo con las primitivas de comunicación y sincronización entre procesos; el uso de las mismas puede provocar inversiones ilimitadas de prioridades, por lo que se requieren otras soluciones: semáforos o monitores con tiempo, o el uso de soluciones de planificación del procesador que eviten inversiones de prioridades [Dav92].

5. CONCLUSIONES

En este trabajo mostramos el uso de un problema clásico para analizar algunos problemas existentes en el desarrollo de sistemas de tiempo real. Se consideró la extensión de dicho problema para planificar tareas periódicas y esporádicas, así como la existencia de un único procesador en el sistema. En base a esto pudo estudiarse la influencia de distintos planificadores de procesos en tiempo real centralizados.

Mostramos que el problema puede usarse para que diseñadores no expertos puedan aprender algunas técnicas para el desarrollo de estos sistemas, poniendo especial énfasis en el uso de un entorno de programación tradicional. De esta forma, quienes hayan pasado por esta experiencia tendrán facilitado su trabajo cuando dispongan de un entorno de desarrollo especialmente orientado para la resolución de estos problemas. A pesar de no tener servicios complejos para tiempo real, el uso de un ejecutivo con servicios extendidos, usando algoritmos de planificación conocidos, permite mejorar los resultados (en términos de tiempo de desarrollo y mantenibilidad).

También pudo comprenderse el desarrollo de ejecutivos corriendo por sobre el núcleo del sistema operativo, y cómo lograr cumplir las metas de las tareas con esta aproximación, poniendo especial énfasis en el overhead y las restricciones impuestos por el núcleo. Esta aproximación tiene, a su vez, gran utilidad si consideramos los casos reales en los que se dispone de un entorno de desarrollo como el usado en nuestro caso.

Finalmente se pudieron estudiar problemas de sincronización con tiempo, y proponer algunas soluciones simples. Este problema nos muestra que pueden encararse estudios relacionados con sistemas de tiempo real, enfrentando problemas complejos, y atacar algunos de ellos, sin que sea necesario disponer de instalaciones caras ni de un laboratorio

complejo; tampoco es necesario el uso de herramientas onerosas. Los resultados han permitido que los desarrolladores verifiquen empíricamente variedfad de resultados conocidos.

En la actualidad estamos tratando de implementar algoritmos más complejos ([Zha87], [Sha90], [Sha90b]) que consideren las restricciones de recursos desde un primer momento. También se está estudiando la implementación de una nueva extensión para estudiar el comportamiento de soluciones distribuidas (ya que el problema estudiado considera la existencia de un mecanismo centralizado que ordena la actividad de los filósofos). Se puede considerar una extensión al problema en la cual cada uno de los filósofos debe comunicarse con los otros de forma descentralizada para lograr el orden de ejecución apropiado. En el caso original, no importa cuantas rondas de mensajes de negociación sean necesarios, sino que sólo importa el resultado de estar habilitado para comer en algún momento. En cambio, el caso es distinto para los sistemas de tiempo real, ya que esta negociación descentralizada consumirá tiempo y afectará el cumplimiento de tiempos de comienzo o de las metas.

A pesar que un ejemplo teórico como este nos muestra que la utilización de un conjunto de tareas concurrentes nunca puede garantizar determinísticamente la ejecución de un conjunto de tareas de tiempo real, existen muchas aplicaciones que siguen usando soluciones de este tipo. Para facilitar el desarrollo es necesario que los sistemas operativos para tiempo real incluyan conjuntos de primitivas especializadas, como ser semáforos con tiempo, monitores con tiempo, datagramas de tiempo real, circuitos virtuales de tiempo real, y transacciones de tiempo real. El desarrollo de trabajos de este tipo pueden servir como primer paso para comprender el problema y proponer nuevas soluciones.

6. AGRADECIMIENTOS

Al Lic. Roberto J.G. Bevilacqua, a los jurados (por sugerencia de ellos estamos desarrollando una especificación del problema en Redes de Petri con tiempo, que estará disponible en http://www.dc.uba.ar/) y a todos los alumnos del curso de Sistemas de Tiempo Real de nuestro departamento de 1993 a la fecha.

7. REFERENCIAS

[Ade94] ADELBERG, B.; GARCIA-MOLINA, H.; KAO, B. "Emulating soft real-time scheduling using traditional operating system schedulers". Technical Report, Stanford University. 1994.

[Bak86] BAKER, T., SCALLON, G. "An Architecture for Real-Time Software Systems". IEEE Software, May 1986, pp. 50-58.

[Dav92] DAVARI, S.; SHA, L. "Sources of Unbounded priority inversions in Real-Time systems and a comparative study of possible solutions". ACM Operating Systems Review, Octubre 1992. pp. 110-120.

[Dij65] DIJKSTRA, E.W. "Cooperating Sequential processes". Technical report EWD-123, Technological University, Eindhoven, The Netherlands. (1965). En "Programming Languages", Genuys, F. (Ed.), London: Academic Press, 1965.

[Liu73] LIU, C.; LAYLAND, J. "Scheduling algorithms for multiprogramming in a Hard Real Time System Environment". Journal of the ACM, Vol. 20, No. 1, 1973, pp 46-61.

[Mok83] MOK, A. "Fundamental design problems of distributed systems for the hard-real-time environment". Ph.D thesis. MIT. Cambridge, Mass. May 1983.

[Ram94] RAMAMRITHAM, K.; STANKOVIC, J. "Scheduling algorithms and operating systems support for real-time systems". Proceedings of the IEEE, Vol. 82, No. 1, pp. 55-67, January 1994.

[Sha90] SHA, L.; GOODENOUGH, J. "Real-Time Scheduling Theory and Ada". IEEE Computer, April 1990. pp 53-62.

[Sha90b] SHA, L. et al. "Priority inheritance protocols: an approach to real-time syn-cronization". IEEE TOCS. 39 (9). 1990.

[Sta93] STANKOVIC, J., RAMAMRITHAM, K. "Hard Real-Time Systems". IEEE Press. 1993.

[Wai95a] WAINER, G. "Implementing Real-Time services in MINIX". ACM Operating Systems Review. Julio de 1995.

[Wai95b] WAINER, G. "Algunos resultados de planificación centralizada en tiempo real". En Anales de las 24 Jornadas Argentinas de Informática e Investigación Operativa. Agosto de 1995.

[Wai95c] WAINER, G. "Some Results on Experimental Evaluation of Real-Time Scheduling". En Anales de la XXI Conferencia Latinoamericana de Informática (CLEI) Panel '95. Agosto de 1995.

[Wai96] WAINER, G. "AgaPé-TR: una herramienta para simulación de planificación local de procesos en tiempo real". Informe Interno FCEN-UBA. En Anales de la XXII Conferencia Latinoamericana de Informática. 1996.

[Zha87] ZHAO, W. et al. "Preemptive scheduling under time and resource constraints". IEEE Transactions on Computer Systems. Agosto 1987. pp. 949-960.