Prototipo de Simulador de Tareas de Tiempo Real en Procesadores Heterogéneos

Guillermo Arispe, Martín Beltrame, Martín Rodríguez, Claudio Aciti[1]

Universidad Nacional de Tres de Febrero,
Valentín Gomez 4752 - Caseros - Buenos Aires - Argentina
{guillermo.arispe,tinchobeltrame,martin.roosp,
claudioaciti}@gmail.com

Resumen. Se diseñó y se desarrolló un prototipo de un simulador de tareas de tiempo real que se pueden ejecutar en procesadores heterogéneos. Dichas actividades se enmarcan dentro del proyecto de investigación 161/14 "Técnicas de aceleración en Arquitecturas Heterogéneas de Computación", aprobado y ejecutado en UNTREF. El trabajo se desarrolló en Java y permite simular la asignación de tareas de tiempo-real periódicas a un grupo de procesadores heterogéneos, basándose en los conceptos aplicados al estudio de sistemas de tiempo real, tales como tiempo de inicio, tiempo de período, tiempo de cómputo y plazo de finalización. El simulador permite optar por tareas con o sin desalojo, ya que tienen una prioridad asociada. Dentro de los objetivos parciales se hizo la definición de entradas, el algoritmo de simulación y la visualización del resultado, como módulos separados desarrollados independientemente, que mediante un lenguaje de intercomunicación permitan la ejecución de la simulación en su totalidad.

1 Introducción

En la actualidad, las aplicaciones en sistemas embebidos crecen en complejidad y son, cada vez más, usados en situaciones donde se requiere control del tiempo. Es por esto que las arquitecturas de hardware y las herramientas de planificación de tareas para sistemas embebidos deben evolucionar con el fin de resolver los requerimientos. Históricamente, muchos sistemas embebidos de tiempo-real han contado con una sola CPU, así que los diseñadores han tenido que apoyarse en mejoras en la velocidad del reloj del CPU, en el cambio a la computación con multiprocesadores, y otras innovaciones para obtener el rendimiento de procesamiento requerido por las aplicaciones complejas. Sin embargo, cada vez más los sistemas embebidos de tiempo-real están migrando a arquitecturas heterogéneas, es decir arquitecturas de computación que cuentan con múltiples elementos de procesamiento para proporcionar un balance más óptimo entre rendimiento, latencia, flexibilidad, costo, y otros factores.

[1] Director del trabajo de investigación.

Uno de los principales retos que surge es la planificación de las tareas de tiempo-real en multiprocesadores heterogéneos. Los sistemas de tiempo-real (STR), por lo general, está constituidos por tareas periódicas que incluyen, entre sus parámetros, los instantes máximos en que las mismas deben finalizar su ejecución. Este parámetro extra se denomina vencimiento. Si una tarea finaliza después de este tiempo, se dice que ha perdido su vencimiento.

Los sistemas embebidos por lo general, cuentan con un Sistema Operativo de Tiempo Real (RTOS), donde una de las principales funciones es la planificación de las tareas a ejecutar. Para esto, el planificador debe elegir a qué tarea otorgar el derecho de ejecución. Cada vez que una tarea se instancia o termina, el planificador debe examinar la cola de tareas listas y, dependiendo de la política de ejecución implementada, elegir una para ejecutarla. Al planificar tareas en sistemas heterogéneos, la complejidad aumenta notablemente, por que no solo hay que decidir en qué procesador se va a ejecutar la tarea siguiente, como ocurre en un sistema de multiprocesadores homogéneos, sino que además el RTOS debe decidir en cuáles de los procesadores se puede realizar la ejecución. Es por esto que, de antemano, el RTOS debe saber en qué tipo de procesadores se puede ejecutar cada una de las tareas que integra el sistema.

Por otra parte, los RTOS implementan diversas técnicas y métodos como: planificación basadas en prioridades, manejo de tareas que no cumplen los tiempos preestablecidos, el desarrollo de políticas de tolerancia a las fallas, atención de tareas esporádicas y aperiódicas, atención de tareas mandatorias / opcionales, calidad de servicio (QoS), por mencionar solamente algunas.

2 Objetivo

El presente trabajo de investigación, con desarrollo experimental en las áreas de Sistemas de Tiempo Real y Computación Paralela, busca desarrollar un prototipo de simulador de planificación de tareas de tiempo-real con multiprocesadores heterogéneos, que permita mejorar las técnicas de planificación, simular el manejo de plazos perdidos, manejar tareas esporádicas y periódicas, realizar estadísticas de las ejecuciones, entre otras.

3 Definición del problema

Definición de Sistemas de Tiempo-Real: Un sistema de tiempo-real consiste en un conjunto P de N tareas periódicas. Cada tarea $\tau i \in \Pi$ está caracterizada por su período, Ti, el tiempo límite para ejecutarse, Di, el peor tiempo de ejecución, Ci, el tiempo de inicio (offset), fi y una prioridad, Pi.

$$\Pi = \{\tau i = (Ti , Di , Ci , fi , Pi) \}$$

La unidad mínima de cómputo de una tarea Ti es una acción J. Una tarea es un conjunto de acciones similares que se repiten a lo largo del tiempo.

$$\tau i = Ji,1 , Ji,2 , . . . , Ji,q$$

Se asume que

$$\tau i,q = Ji,q$$

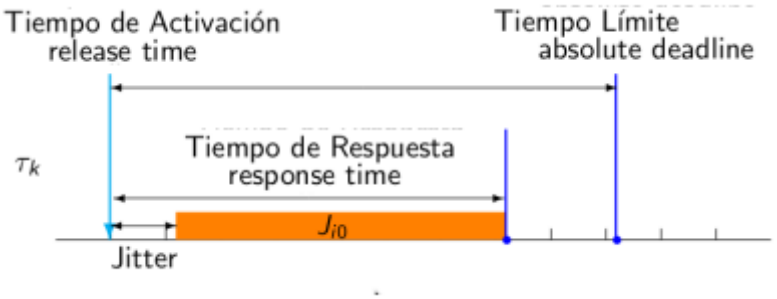

Fig 1. Se observa una acción $J_{i,0}$ de una tarea τ_i.

Definición de Sistemas Heterogéneos: Un sistema con multiprocesadores heterogéneos está compuesto por un conjunto Φ de tipos de procesadores diferentes φx.

$$\Phi = \{ \varphi x,y , 1 \leq x \leq M, y \geq 0 \}$$

El subíndice x indica el tipo de procesador y el subíndice y indica la cantidad.

3.1 Sistema de tiempo-real de tareas periódicas en procesadores heterogéneos

Un sistema de tiempo-real de tareas periódicas en procesadores heterogéneos es un sistema compuesto por:

- Un conjunto de tareas $\tau_i \in \Pi$.
- Un conjunto de procesadores $\varphi_{x,y} \in \Phi$
- Una lista Υ de procesadores y tareas definida por pares ordenados $< \varphi_{x,y} , \tau_i >$

y sigue las condiciones que se especifican a continuación:

- Cada tarea τ_i tiene asignada una prioridad. La tarea que tiene mayor frecuencia es la que tiene mayor prioridad. Y la que tiene menor frecuencia es la que tiene menor prioridad.
- Una tarea τ_i puede ser ejecutada en uno o más tipos de procesadores $\varphi_{x,y}$.

- Una vez que una instancia q representada por $\tau_{i,q}$ de una tarea τ_i inició su ejecución, sólo puede continuar en procesadores del mismo tipo.
- Cada tarea deberá tener una definición para cada tipo de procesador en la que pueda ser ejecutada.

3.2 Mecanismos para la planificación de tareas

Los mecanismos para planificar las tareas son en sí la forma en que se le asignan las prioridades a las tareas. Los más usados son:

- Prioridades fijas (PF): A cada tarea se le asigna una prioridad al iniciar la ejecución y esta no cambia. Para ello, se ordenan las tareas por su período T en orden creciente, y a cada una se le asigna una prioridad empezando de 0.
- Prioridades dinámicas (EDF): Inicialmente es igual a PF, pero a medida que avanza el tiempo t, en cada instante se recalcula la prioridad $P_{i,q} = q * Ti - t$

3.3 Mecanismos para manejo de tareas no cumplidas por sobrecarga

Cuando una tarea llega a su tiempo de finalización y no terminó la ejecución, el scheduler debe decidir qué hacer con esa instancia. Los mecanismos más usados son:

- Abortion: Abortar la instancia de la tarea en curso y darle lugar a la instancia nueva.
- Delayed Completition: Esperar a que la instancia termine en algún momento y abortar las instancias siguientes que van apareciendo.

4 Solución propuesta

La arquitectura de la aplicación está compuesta por cuatro clases con responsabilidades bien definidas.

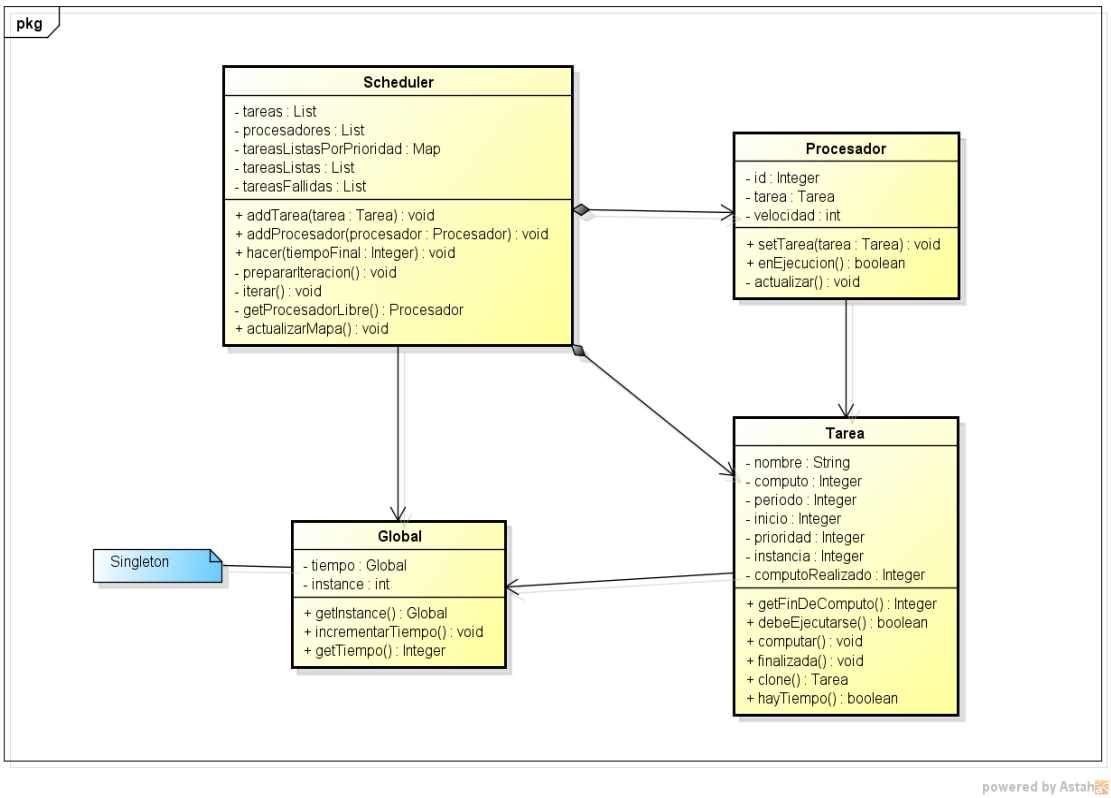

Fig 2. Diagrama de clases.

4.1 Clase Tarea

Modela el concepto de Tarea estudiado. Una tarea tiene cinco atributos que la definen:

- Tiempo de activación / Tiempo de inicio: Momento en que la tarea empieza a ejecutarse.
- Deadline: En este punto, la tarea debe haber finalizado. En el simulador siempre es igual al valor del período.
- Tiempo de cómputo: Representa el tiempo de procesador necesario para ejecutar la tarea.
- Período: Intervalo de tiempo entre el tiempo de inicio y el deadline.
- Prioridad: Durante la fase de diseño a cada tarea se le asigna una prioridad, la cual es denominada Prioridad Base. Las tareas con mayor prioridad son las que se ejecutan primero. Este atributo, es utilizado en tácticas como la de cambio de contexto. Las prioridades se definen de 0 a 99 usualmente, siendo 0 la mayor prioridad. En el simulador utilizamos prioridades fijas.

4.2 Clase Procesador

Cada procesador sólo puede computar una tarea a la vez. Que el procesador tenga asignada una tarea significa que provocará un incremento en su tiempo computado pasada la unidad de tiempo actual.

4.3 Clase Global

Esta clase se encarga únicamente de llevar el control del tiempo durante la ejecución del programa. El tiempo es representado por un valor entero el cual se incrementa en una unidad luego de cada iteración. Una unidad de tiempo es el valor mínimo que puede representar el simulador, por lo tanto los atributos de las tareas construidas, como período y cómputo, deberán ser definidos con un valor mayor o igual para que la simulación sea coherente.

El manejo del tiempo no permite valores fraccionarios. En consecuencia, un procesador con velocidad de cómputo 1 por unidad de tiempo, representa la mayor velocidad posible. Si se quisiera ralentizar los procesadores se deberá hacer a razón de números enteros, incrementando su variable de velocidad de cómputo.

Los atributos y los métodos de esta clase son estáticos, por lo que pueden ser accedidos desde cualquier otra clase sin necesidad de instanciarla, lo que asegura que la variable tiempo sea única a lo largo de la ejecución.

4.4 Clase Scheduler

Esta clase modela el planificador encargado de distribuir, entre los procesadores disponibles, las tareas que están listas para ejecutarse. El Scheduler considera que estas tareas consumen cierta cantidad de tiempo del procesador, por lo que su objetivo es asignarles dicha cantidad de tiempo.

Estos procesos son ejecutados de forma preemptiva o apropiativa (con desalojo), es decir que, teniendo en cuenta las prioridades de las tareas que están en ejecución y las tareas a ejecutar, puede realizar un **cambio de contexto** en caso de ser necesario. De esta manera, la tarea en ejecución pasa al estado de suspendida, liberando el procesador para ejecutar una tarea con mayor prioridad. Una vez finalizada la ejecución de la tarea prioritaria, la suspendida se reanuda para su finalización.

5 Implementación de procesadores heterogéneos

Se modificó la arquitectura de la aplicación para hacer posible la existencia de procesadores heterogéneos, es decir, procesadores que solo pueden computar tareas de un tipo determinado. A su vez, los tipos de procesador pueden diferir en su velocidad de procesamiento, haciendo que algunos computen más rápido que otros.

En principio se definieron las clases Procesador y Tarea como clases abstractas, de manera que no puedan instanciarse. Obligatoriamente los tipos de procesadores y tareas nuevas deben heredar de estas clases respectivamente.

5.1 Clase Procesador

El atributo ya existente en la clase Procesador denominado velocidad, con valor por defecto 1 ahora es denominado clock y representa cuántas unidades de tiempo se necesitan para completar un cómputo de tarea. De esta manera, el valor 1 representa la velocidad más rápida, comportamiento esperado en la versión con procesadores homogéneos.

Para que el procesador pueda saber qué cantidad de fracciones de cómputo ha realizado, se utiliza la variable razonDeClock.

Para crear un tipo de procesador en particular se debe heredar de la clase Procesador. En su constructor se pasa además de su id, su velocidad de clock.

Es responsabilidad de la clase hijo sobreescribir el método setTarea() para restringir el tipo de tareas que puede recibir el nuevo tipo de procesador.

Si este método no es sobreescrito, se asume que se puede recibir cualquier tipo de tarea, como lo establece el comportamiento de la clase padre.

El método retorna un boolean que confirma la asignación de la tarea. Esto es usado por el Scheduler para verificar si el procesador acepta el tipo de tarea que se le asigna. El Scheduler desconoce los tipos de tareas y procesadores que tiene disponibles y se utiliza el polimorfismo para poder tratarlos por igual independientemente de su subtipo.

Se modificó la clase Scheduler para que, en caso de no poder asignarle una tarea a un procesador, se le intente asignar otra.

En conclusión, heredando de esta clase se pueden crear procesadores con distintas velocidades de cómputo y que acepten sólo un conjunto de tareas específicas. Por ejemplo: ProcesadorGráfico sólo admite TareaGráfica.

También procesadores del mismo tipo pueden ser instanciados con velocidades distintas. Por ejemplo: procesadores gráficos de distinta gama:

```
Procesador NVIDIA8400 = new ProcesadorGrafico(1, 400);
Procesador NVIDIA9800 = new ProcesadorGrafico(2, 250);
```

5.2 Clase Tarea

De forma similar a lo implementado con los procesadores, se define la clase Tarea como una clase abstracta de la cual se debe heredar para crear nuevos tipos de tareas.

Es obligación de la clase heredada implementar el método nuevaInstancia() ya que es definido como abstracto por la clase padre. Esto es necesario para que la nueva instancia de la tarea guarde su nombre de clase original y sea tratada como tal.

5.3 Paso a paso de la Planificación de tareas en varios procesadores

De forma similar a lo implementado con los procesadores, se define la clase Tarea como una clase abstracta de la cual se debe heredar para crear nuevos tipos de tareas.

Es obligación de la clase heredada implementar el método nuevaInstancia() ya que es definido como abstracto por la clase padre. Esto es necesario para que la nueva instancia de la tarea guarde su nombre de clase original y sea tratada como tal.

1. El scheduler se fija, en cada instante de tiempo t, si una tarea τ_i inicia un período nuevo (una instancia nueva) en la estructura T. En ese caso, $\tau_{i,\,q}$ debería agregarse al conjunto de tareas LISTAS (L). Esto significa que la instancia anterior $\tau_{i,\,q-1}$ llegó a su tiempo de finalización.
    a. Si la instancia $\tau_{i,\,q-1}$ completó su ejecución. Entonces $\tau_{i,\,q}$ es agregada al conjunto de tareas LISTAS (L).
    b. Si la instancia $\tau_{i,q-1}$ empezó su ejecución pero no la completó su ejecución. Entonces $\tau_{i,\,q-1}$ sigue a la espera en el conjunto de tareas LISTAS (L) y $\tau_{i,\,q-1}$ es descartada.
2. El scheduler debe decidir qué tarea τ_i se va a ejecutar en el instante t siguiente.
    a. Para el primer procesador sin ninguna tarea en ejecución se busca en las tareas LISTAS la de mayor prioridad que coincida con el tipo de procesador y la ejecuta.
    b. Si no hay procesadores libres disponibles, para cada procesador compara su tarea en ejecución con la de mayor prioridad de las tareas LISTAS del mismo tipo del procesador. Si la que está en ejecución tiene mayor prioridad o igual, entonces sigue. Caso contrario

realiza un cambio de contexto.

3. Ejecuta la tarea.
4. Vuelve a 1.

5.4 Ejemplo

En este ejemplo se desea simular la ejecución de 3 tareas de distintas propiedades en 2 CPUs independientes. Para ello es necesario primero definir las tareas.

Tabla 1. Definición de tareas.

Id	Cómputo	Deadline = Período
1	5	10
2	5	20
3	2	10

Luego de la ejecución el simulador muestra el resultado mediante el siguiente gráfico:

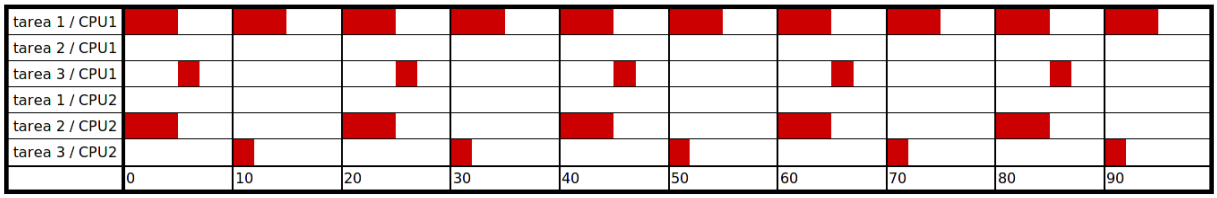

Fig 3. Resultado de 100 iteraciones.

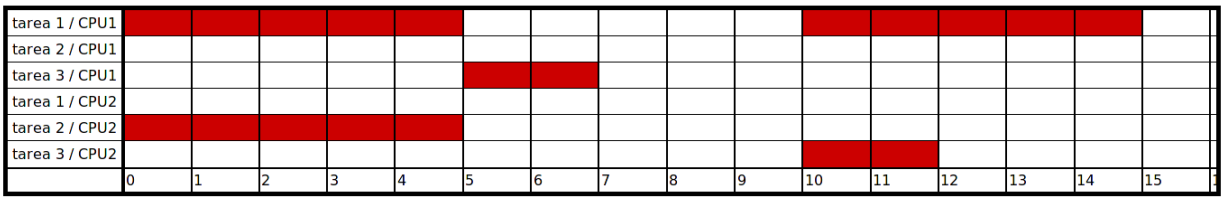

Fig 4. Detalle de la ejecución de 15 iteraciones.

6 Conclusiones y Trabajos Futuros

Se desarrolló un prototipo de simulador de planificación de tareas de tiempo-real con multiprocesadores heterogéneos, que permite simular diferentes técnicas de planificación, desde las más conocidas con Prioridades Fijas o EDF, hasta cualquier otra que se desee. Para situaciones de sobrecarga también se puede elegir el mecanismo para el manejo de plazos perdidos.

El simulador permite analizar la cantidad de plazos cumplidos, la cantidad de fallos y los cambios de contexto por cada tarea y para todo el sistema.

Entre los trabajos futuros, se propone probar algoritmos para la asignación de los procesadores cada vez que una tarea lo requiera. También se espera comparar rendimientos de diferentes arquitecturas de procesadores.

Referencias

[1] P. A. Laplante, Real-Time Systems Design & Analysis. Wiley India Pvt. Ltd. (2006)
[2] Sistemas de Tiempo Real y Lenguajes de Programación, Burns A., Wellings A., 3ra Edición, Addison Wesley, ISBN 84-7829-058-3. (2003)
[3] Real-Time Systems, Liu, J.W.S., Prentice Hall. ISBN 0-13-099651-3. (2000)
[4] Real-Time Computer Control. An Introduction, Bennet S., 2da Edición, Prentice Hall, ISBN 0-13-764176-1. (1994)
[5] G. Buttazzo. Rate Monotonic vs. EDF: Judgment Day. Real-Time Systems, Vol. 29, Issue 1, pp. 5-26. (2005)
[6] Sha, L., Abdelzaher, T., Årzén, K-E., Cervin, A., Baker, T., Burns, A., Caccamo, M., Lehoczky, J. and Mok, A. K. "Real Time Scheduling Theory: A Historical Perspective". Real-Time Systems 28(2-3):101-155. (2004)
[7] G. Bernat, and R. Cayssials. "Guaranteed on-line weakly-hard realtime systems". In proc. 22nd IEEE Real-Time Systems Symposium, pages 25-35, London. (2001)
[8] G. Buttazzo, M. Bertogna, and G. Yao. "Limited Preemptive Scheduling for Real-Time Systems. A Survey". IEEE Transactions o Industrial Informatics, Vol. 9, No. 1, (2012)
[9] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority inheritance protocols. An approach to real-time synchronization". IEEE Trans. Comput., vol. 39, no. 9, pp. 1175–1185. (1990)