AHORRO ENERGÉTICO EN LA PLANIFICACIÓN DE SISTEMAS EN TIEMPO REAL

ARQUITECTURA I TECNOLOGIA DE COMPUTADORS

UNIVERSITAT POLITÈCNICA DE CATALUNYA

DOCTORANDO: M. ÀNGELS MONCUSÍ

CO-DIRECTORES: ALEJANDRO ARENAS

JESÚS LABARTA

AGRADECIMIENTOS

Aquesta tesis, la dedico a tota la meva família especialment a la meva mare i sogra que han tingut cura dels nens (Carla i Dídac) per a què jo tingués temps de treballar, al meu marit, que tot i estar sempre treballant ha tingut sempre temps per animar-me, també la vull dedicar al meu cunyat Joaquim, que no ha pogut veure-la finalitzada però segur que n'hagués fet broma.

Vull agrair al Jesús Labarta i a l'Alex Arenas que van acceptar dirigir-me la tesis, així com la paciència, consells i les moltíssimes revisions del document que m'han fet.

També vull agrair al meu company de fatigues Josep M. Banús, la companyia en els viatges a Barcelona, i l'ajuda que m'ha ofert en la cerca de informació, la generació dels jocs de proves sintètics, les discussions que hem fet i els consells que m'ha donat.

ÍNDICE

ÍNDICE DE FIGURAS

Capítulo 1

AHORRO ENERGÉTICO EN SISTEMAS DE TIEMPO REAL

El objetivo de este capítulo es dar una visión general de los sistemas informáticos de tiempo real y del problema del consumo energético en los mismos, realizando un énfasis especial en la problemática propia de la planificación con ahorro energético de las tareas de tiempo real.

1.1 Introducción

En la actualidad el uso de los procesadores se ha extendido considerablemente y se hallan en todos los campos, desde aparatos electrodomésticos, periféricos y controladores hasta llegar a los grandes equipos computacionales que requieren del uso de aplicaciones informáticas de gran complejidad. Los procesadores están evolucionando continuamente, ofreciendo configuraciones hardware más pequeñas, rápidas, fiables, y baratas, permitiendo con ello generalizar su uso. Ejemplos obvios de esta evolución se dan en el uso cada vez mayor de los teléfonos móviles, los PDA's, las video-consolas, los ordenadores portátiles, y los no tan obvios como los procesadores empotrados que se encuentran en la mayoría de electrodomésticos modernos. A este tipo de aplicaciones se las denomina genéricamente aplicaciones de tiempo real o empotradas [14]. Siguiendo esta evolución de los procesadores, las aplicaciones que ejecutan los sistemas de tiempo real pueden ser mas complejas y precisas puesto que se dispone de una mayor velocidad de proceso.

El desarrollo de estos procesadores más complejos y potentes, precisarían, a su vez, de un consumo energético mayor, provocando una mayor generación de calor y de mecanismos para su disipación. Esta disipación se hace más difícil conforme la densidad de los transistores en los chips y la de estos en los procesadores va incrementándose, lo que provoca problemas de durabilidad de los sistemas. Además, este mayor consumo energético es un gran inconveniente para aquellos sistemas que precisen del uso de baterías (sistemas móviles), y en menor medida para todos los sistemas en general.

Existen sistemas que pueden estar localizados en entornos donde el ahorro energético es esencial para asegurar la operabilidad y duración del sistema, por ejemplo en aplicaciones espaciales, donde un menor consumo se traduce en menores paneles solares y menores baterías, por tanto en menor peso y dimensión.

En la nota de prensa de la corporación de IBM[1] en el año 2001 hace referencia al problema del consumo energético en los siguientes términos:

[1] http://www-5.ibm.com/es/press/notas/2001/octubre/lowpower.html

> "Durante años, IBM ha ido reduciendo su propio gasto en el consumo de energía mediante avances en tecnologías de la información y otras acciones para el ahorro de energía. En el año 2000, estas acciones de IBM a nivel mundial tuvieron como resultado un recorte en el consumo de electricidad de 271 millones de kilovatios / hora -lo que equivale al suministro anual de 45.000 hogares-, y una reducción en el consumo de combustible equivalente a 5.300.000 litros de gasolina. Como resultado, IBM evitó la emisión de aproximadamente 157.500 toneladas de dioxinas procedentes del carbón y ahorró unos 15,2 millones de dólares. Durante los últimos 10 años (1991 - 2000) IBM ha ahorrado aproximadamente 527 millones de dólares en el consumo de electricidad y ha evitado la emisión de 5,6 millones de toneladas de CO2".

A nivel hardware, investigadores de distintas universidades [12],[19],[67],[85], y grandes compañías de desarrollo de procesadores han realizado una fuerte inversión en el desarrollo de procesadores de bajo consumo energético, por ejemplo:

En marzo del 2003 en una nota de prensa de Intel[2] se dice que:

> El procesador PXA255 de Intel, basado en la microarquitectura *Xscale*, disponible en las velocidades de 100, 200 y 400 MHz, aporta dos mejoras ya al precedente PXA250: mayor rendimiento general y menor consumo energético. Ha doblado la velocidad del bus interno, mejorando así el rendimiento de las aplicaciones del dispositivo y ha reducido el tensión de alimentación del procesador a 1,3 voltios a 400 MHz. Esta reducción se traduce en un 30 por ciento de ahorro energético, cuando se encuentra en funcionamiento, y a hasta un 60 por ciento, cuando está inactivo.

en diciembre del 2004 en una nota de prensa de AMD[3] se dice que:

> AMD e IBM anunciaron hoy que han desarrollado una nueva y excepcional tecnología de transistores de silicio tensado con el objetivo de mejorar el rendimiento de los procesadores y el ahorro de energía. El innovador proceso da como resultado un aumento de la velocidad del transistor de hasta un 24%,

[2] http://www.intel.com/pressroom
[3] http://www.amd.com/es-es/Corporate/VirtualPressRoom

a los mismos niveles de energía, comparado con transistores similares fabricados sin esta tecnología.

Estos transistores más rápidos y que utilizan la energía de forma más eficaz son la base de unos procesadores de mayor rendimiento y menor consumo energético. Al disminuir el tamaño de los transistores éstos funcionan más rápido, pero también se corre el riesgo de que aumente el nivel de calor y el consumo de energía debido a fugas eléctricas o a una conmutación ineficaz. El silicio tensado desarrollado conjuntamente por AMD e IBM ayuda a superar estas dificultades. Además, este proceso convierte a AMD e IBM en las primeras empresas en introducir el silicio tensado que funciona con tecnología de silicio sobre aislante (SOI), lo que resulta en una mejora del rendimiento y un mayor ahorro de energía.

A nivel software, también se han invertido grandes esfuerzos en la minimización energética global (hardware y software) [18], [20], a nivel de la BIOS [24], del sistema operativo [39] y de la compilación de las aplicaciones [62]. Se extenderán estos puntos en apartados posteriores.

La presente tesis, se centrará únicamente a nivel de sistema operativo, más concretamente en la planificación de tareas de tiempo real. En el siguiente punto se efectuará una breve descripción de las características fundamentales de la especificación de los sistemas de tiempo real, de la problemática de la reducción energética en la tecnología CMOS y de los algoritmos para la planificación de tareas de tiempo real con ahorro energético existentes. Para finalizar este capítulo se especificaran los objetivos concretos que se han planteado.

1.2 Sistemas de Tiempo Real

La característica fundamental de los sistemas de tiempo real es que su correcto funcionamiento no depende únicamente del resultado lógico de la computación sino también del tiempo en el que se producen los resultados. A veces, es más conveniente tener un resultado impreciso en un cierto instante a tenerlo con gran precisión pero demasiado tarde. Por tanto, el parámetro de rendimiento más importante para estos sistemas es el cumplimiento de las restricciones temporales

impuestas por la aplicación en tiempo de diseño. Otra característica importante de estos sistemas es que las aplicaciones deben interactuar con su entorno a través de sensores y actuadores. Deben ser, por tanto, buenos receptores de eventos externos proporcionándoles una buena calidad de servicio o tiempo de finalización adecuado.

Los sistemas de tiempo real, según el tipo de restricción temporal que poseen, pueden clasificarse en:

- Sistemas de tiempo real estricto (*Hard real time*)
- Sistemas de tiempo real firme (*Firm real time*)
- Sistemas de tiempo real no estricto (*Soft real time*)

Los sistemas de tiempo real estricto son aquellos en los que es absolutamente imperativo el cumplimiento de todos los plazos temporales, debido a que su incumplimiento puede dar lugar a una catástrofe, en vidas humanas o económicas. Básicamente, estos sistemas se hallan en aplicaciones de control industrial, aplicaciones aeronáuticas, etc. En los sistemas de tiempo real no estrictos, el tiempo de finalización es importante, pero el sistema seguirá funcionando aunque los plazos no se cumplan, simplemente habrá una degradación de la calidad de servicio, un ejemplo de estos sistemas son los sistemas multimedia en general. En la Figura 1 se pueden observar dos ejemplos de sistemas de tiempo real.

Los sistemas de tiempo real mixto o firme, son aquellos en los que si se pierde algún plazo de vez en cuando, hay una degradación del servicio que puede compensarse usando mecanismos de control de calidad, por ejemplo, en las aplicaciones de control industriales, como seria el caso de las cadenas de producción, el control de una cinta transportadora, etc.

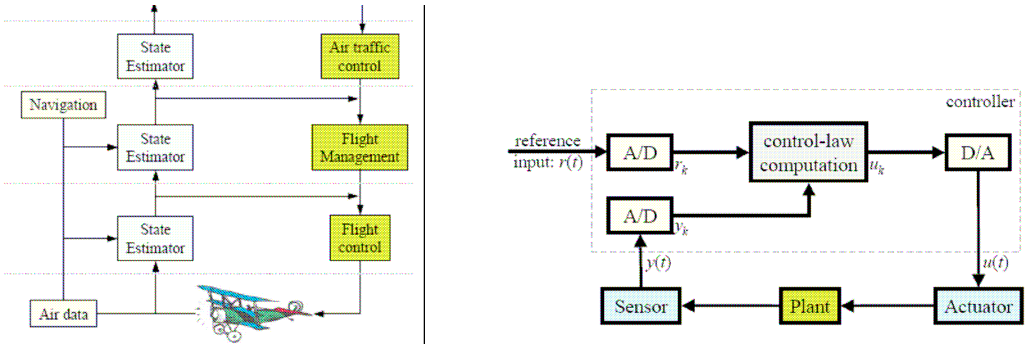

Figura 1: Dos ejemplos de sistemas de control en tiempo real, el de la izquierda de aviónica, y el de la derecha un control digital.

Las aplicaciones desarrolladas para los sistemas de tiempo real estricto se basan en una serie de sensores que reciben información del mundo exterior y unos actuadores que permitirán ejecutar acciones de control sobre el sistema. Las aplicaciones de este tipo, necesitan tener una habilidad especial para coordinar un gran número de actividades en paralelo, pero de complejidad baja. Cada una de estas actividades esta implementada en forma de tarea, que la mayoría de veces serán de ejecución cíclica o periódica, por ejemplo la evaluación del estado de un sensor cada cierto tiempo. Sin embargo, existen algunas tareas que no pueden ser periódicas por su propia naturaleza sino que deberán ser aperiódicas, tareas que se ejecutan como respuesta a un evento externo a la aplicación por ejemplo alarmas y/o peticiones de un operador de consola, o como respuesta a un evento interno a la propia aplicación (acciones asíncronas requeridas por el propio programa).

En general, las tareas implementadas para estas aplicaciones pueden tener plazos temporales estrictos (*hard deadlines*) o no estrictos (*soft deadlines*). A las tareas aperiódicas con plazos temporales se las llama tareas esporádicas, siendo las tareas aperiódicas aquellas que no los tienen. Normalmente, las aplicaciones de tiempo real tendrán una mezcla de todo tipo de tareas. A menudo las tareas esporádicas son tratadas como tareas periódicas en el que el periodo viene fijado por el tiempo mínimo entre llegadas de estas tareas, de manera que estas tareas serán siempre aceptadas. A continuación en la Figura 2 se muestra el modelo de sistema explicado.

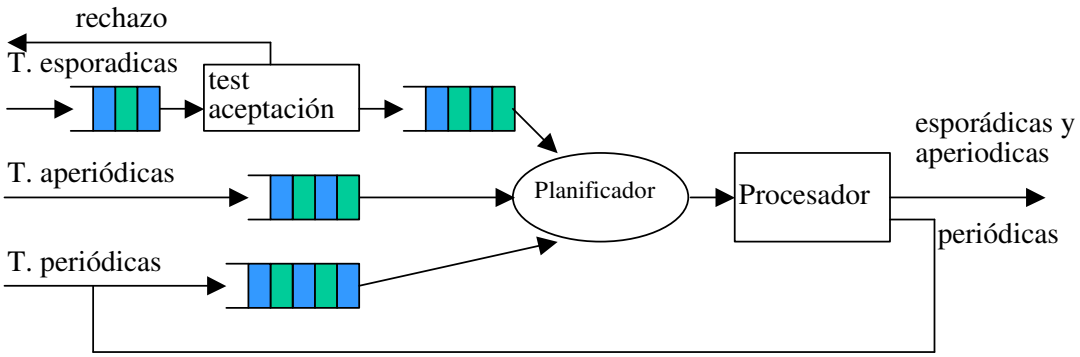

Figura 2: Modelo de sistema de tiempo real.

Últimamente, debido a los avances tecnológicos que se han producido en la electrónica en general y en la informática en particular, se ha aumentado la complejidad de estos los sistemas de tiempo real aumentado también la complejidad de las tareas. El diseño de este tipo de aplicaciones es más complejo si en lugar de

usar un único procesador se usan varios procesadores formando una aplicación distribuida, ya que presentan nuevos requerimientos físicos tales como canales de interconexión entre los procesadores, secuencialidad entre tareas de distintos procesadores, restricciones temporales globales (desde el momento en que se produce el evento en un procesador hasta la ejecución de la última tareas de una cadena de tareas que pueden hallarse en distintos procesadores), recursos globales compartidos entre los diversos procesadores, etc.

Realizando una simplificación importante, se puede decir que las tareas que ejecutan los sistemas de tiempo real tienen como características más importantes:

- Periodo de activación (T)
- Plazo temporal (D)
- Peor tiempo de ejecución (WCET)

El periodo de activación de una tarea es la separación temporal entre dos activaciones sucesivas de la tarea, el plazo temporal es el tiempo que tiene la tarea para emitir una respuesta una vez activada y el peor tiempo de ejecución es el máximo tiempo que tardará la tarea en realizar sus cálculos. Gráficamente, pueden verse representados estos tiempos en la Figura 3.

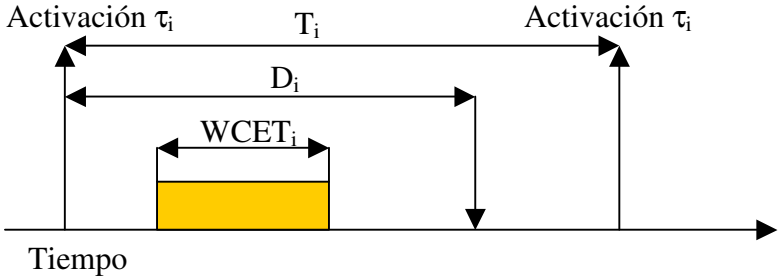

Figura 3: Características de la tarea τ_i

Dadas las anteriores características se deduce que $0 \leq WCET \leq D$. El porcentaje de utilización del procesador de una tarea determinada τ_i viene especificado por $WCET_i/T_i$, por tanto la suma del porcentaje de utilización de todas las tareas que se ejecutaran en un procesador determinado debe cumplir:

$$U = \sum_{i=1}^{n} \frac{WCET_i}{T_i} \leq 1 \qquad \text{(ec. 1)}$$

El periodo y el plazo temporal de las tareas viene impuesto por las características del sistema físico que se quiera controlar y se determina en tiempo de diseño de la aplicación de tiempo real, sin embargo, el cálculo del peor tiempo de finalización es más complejo. En el artículo de P. Puschner y A. Burns [71] se encuentran las referencias básicas de cómo se puede realizar este cálculo. El WCET se calcula en el peor de los casos, por tanto, es una cota superior del tiempo máximo de ejecución de una determinada tarea. Esta cota temporal es dependiente de la arquitectura de la máquina donde se ejecuta, y no debe tener en cuenta las posibles interferencias entre las distintas tareas del sistema, tales como apropiaciones del procesador por una tarea de mayor prioridad, tiempos de bloqueo en el acceso a un recurso compartido, y demás tiempos que dependen del tipo de planificador que se use.

A modo de resumen, el cálculo del WCET tiene que permitir:

- Proporcionar una cota superior segura para el tiempo que tardará en ejecutarse una tarea, el WCET se usará para validar la planificabilidad del sistema de tiempo real.
- Ajustarse al tiempo real de ejecución de la tarea, un tiempo muy pesimista dará lugar a una baja utilización real del procesador.

Los principales problemas que se pueden encontrar en el momento de realizar el cálculo del WCET son los siguientes:

- Caracterización de la secuencia de ejecución de la tarea, análisis de la semántica del código de las aplicaciones, por ejemplo, en cualquier sentencia de control, siempre se tiene que evaluar el camino más largo [2] de ejecución.
- Cambio de la representación del código en lenguaje fuente a la representación en código máquina. Los compiladores, especialmente los que optimizan código, pueden modificar considerablemente el resultado temporal de ejecución a la vez que hacen difícil analizar los posibles caminos de ejecución hallados anteriormente.
- Análisis temporal a nivel de arquitectura hardware. Este análisis debe tener en cuenta todos los tiempos de ejecución de cada posible secuencia de ejecución de código, para después calcular la cota superior. En los procesadores actuales, el tiempo de ejecución no depende exclusivamente de las instrucciones que ejecuta sino que también depende del estado del

hardware (por ejemplo el contenido de las caches, de los *pipelines*, etc.), S.S. Lim en [47] se hace el estudio para el cálculo del WCET en una arquitectura RISC.

Dado que este cálculo es muy complejo, las cotas halladas, suelen ser bastante pesimistas. Concretamente en [47] concluyen que las cotas tienen, en media, un 30% de sobreestimación del tiempo real ejecutado frente al tiempo calculado. Para una tarea periódica, cada instancia puede tener un comportamiento temporal distinto, dado que el número de cálculos a realizar dependerá normalmente del valor de la entrada, de la historia reciente, del estado de otras tareas del sistema. Por tanto en el funcionamiento real de la aplicación normalmente no se consumirá todo el WCET sino únicamente un porcentaje de este tiempo, dejando al procesador libre el resto del tiempo para poder ejecutar otras tareas si las hubiera.

En los sistemas de tiempo real estrictos, es importante poder asegurar, antes de poner en funcionamiento el sistema, que se respetarán todas las restricciones temporales de las tareas, es decir, que el sistema sea planificable. Dos de las medidas generales de planificabilidad de un sistema en tiempo real son la utilización del procesador (U) definida en la ec. 1 y el "*breakdown utilization"* (BU) que se calcula de la siguiente manera: Para todas las tareas del conjunto, el $WCET_i$ se multiplica por un mismo factor de escalado (α), sin modificar los periodos ni los plazos temporales. El factor de escalado se va incrementando hasta el instante en que si hubiera un incremento adicional se perdería el plazo temporal de alguna tarea del conjunto. La utilización del procesador en este punto se define como "*breakdown utilization*" [37], [42].

$$BU = \sum_{i=1}^{n} \frac{\alpha WCET_i}{T_i} = \alpha U \qquad \text{(ec. 2)}$$

Los algoritmos de planificación son los encargados de decidir qué tarea se tiene que ejecutar en cada instante. Los planificadores pueden ser *"planificadores off-line"* en los que previo al funcionamiento del sistema se ha establecido un plan de ejecución de las tareas o *"planificadores on-line"*, en los que la decisión de qué tarea se ejecuta se realiza durante el funcionamiento del sistema. Un esquema de los diferentes tipos de algoritmos básicos de planificación de sistemas de tiempo real puede verse en la Figura 4.

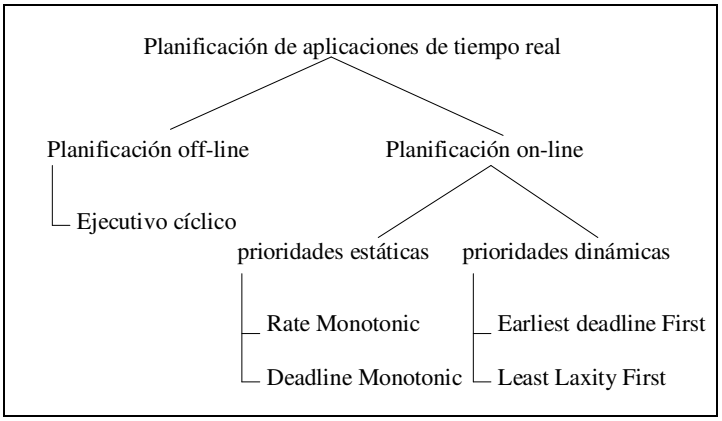

Figura 4: Clasificación de los planificadores clásicos de tiempo real.

Los algoritmos de "*planificación off-line*" son aquellos en los que el planificador debe conocer exactamente, antes de empezar la ejecución de la aplicación de tiempo real, qué tarea y cuándo debe ejecutarla. El conjunto de tareas tiene que ser estático, y difícilmente puede variar durante la ejecución del sistema. El planificador principal de este tipo es el ***ejecutivo cíclico*** que se basa en un plan realizado en la etapa de diseño. En éste se especifica, en cada instante, que tarea se debe ejecutar y la duración máxima de la misma. El planificador simplemente se encarga de seguir el plan en tiempo de ejecución del sistema. Las ventajas principales son: que los plazos se garantizan al diseñar el sistema, que es un método sencillo y robusto aunque de bajo nivel, no hay concurrencia en la ejecución, no hace falta usar mecanismos de exclusión mutua, no hace falta analizar el comportamiento temporal ya que el sistema es correcto por construcción. Sin embargo, los principales problemas que presenta son: que la elaboración del plan cíclico, que en el caso de armonicidad en los periodos de las tareas es NP-completo [14] y en el caso más general es NP-duro [25], el tiempo de generación del plan crece exponencialmente con el número de tareas. Además, las tareas esporádicas y aperiódicas son difíciles de tratar y es poco flexible, debe reservar para la ejecución de las tareas todo el WCET, es difícil de mantener, cuando se cambia la implementación de una tarea hay que rehacer toda la planificación, y por último, desde la perspectiva de la ingeniería del software, no es elegante, debido a que un programa con un tiempo de cálculo elevado normalmente tiene que ser dividido en procedimientos más pequeños, de tamaño fijo, lo que puede romper la estructura del código, por tanto puede ser más difícil de verificar y más propenso a errores.

Los *planificadores on-line* establecen en tiempo de ejecución del sistema que tarea se debe ejecutar en cada momento, por tanto son más flexibles al entorno de la aplicación, pudiéndose adaptar a la carga real del sistema. Estos planificadores normalmente basan la elección de la tarea a ejecutar en distintos niveles de prioridades, eligiendo siempre la tarea con mayor prioridad. Pueden ser expulsivos, de manera que si se activa una tarea de mayor prioridad que la tarea que se está ejecutando, la nueva tarea se apodera del procesador, o no expulsivos [81]. La asignación de prioridades puede ser estática, fija para cada tarea durante toda la ejecución del sistema, o dinámica, la prioridad que se asigna a una tarea va variando en función del estado de la tarea, del instante de activación o del estado del sistema. Los principales *planificadores on-line* con asignación estática de prioridades son: los planificadores expulsivos de prioridades fijas en los que la asignación de prioridades es *Rate Monotonic* o *Deadline Monotonic*. Los principales planificadores con asignación dinámica de prioridades son el *Earliest Deadline First* y el *Least Laxity First*. Un articulo reciente de G. Buttazzo [16] describe las ventajas, inconvenientes y falsos mitos entre ambos tipos de planificadores. A continuación se verá un breve resumen de estos planificadores.

- La asignación de prioridades con un criterio *Rate Monotonic* [48] asigna las prioridades a la tareas dependiendo de su periodo de activación, de manera que una tarea será más prioritaria cuanto más pequeño sea el periodo de activación. Mientras que la asignación de prioridades con criterio *Deadline Monotonic* [45] asigna las prioridades en función del plazo temporal, de manera que las tareas con menor plazo temporal tendrán una prioridad mayor que aquellas tareas con mayor plazo temporal. Se dice que ambas asignaciones con planificadores de prioridades fijas son óptimas en el sentido de la planificación, de manera que si se puede hallar una asignación de prioridades que haga que el conjunto de tareas sea planificable, usando la asignación *Rate Motononic* (con los plazos temporales iguales a los periodos) o *Deadline Monotonic* (con los plazos temporales menores o iguales a los periodos) también lo serán. Para este tipo de planificadores de prioridades fijas existen pruebas de planificabilidad basadas en la utilización del procesador que si superan implica que se puede garantizar la planificabilidad del sistema. La prueba de planificabilidad de C.L. Liu y J.W. Layland [48]

establece que la utilización mínima garantizada para n tareas viene dada por la siguiente ecuación:

$$U = n * (2^{\left(\frac{1}{n}\right)} - 1) \qquad \text{(ec. 3)}$$

conforme el número de tareas (n) del sistema va aumentando el limite tiende a "ln 2" aproximadamente el 69.3% de utilización total. Esta prueba es suficiente, pero no necesaria, lo que implica que se puede asegurar mayores utilizaciones del procesador, dependiendo de los periodos de las tareas. Otra prueba de planificabilidad necesaria y suficiente es la que esta basada en el cálculo del tiempo de finalización de las tareas en el instante crítico, si para todas ellas este tiempo es menor que el plazo temporal el sistema será planificable. Para calcular el tiempo exacto de finalización de las tareas puede usarse la siguiente iteración lineal definida por Joseph y Pandya en [36]:

$$W_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil \cdot C_j \qquad \text{(ec. 4)}$$

donde hp(i) es el conjunto de tareas con una prioridad superior a la tarea i, siendo un valor inicial aceptable

$$W_i^0 = C_i + \sum_{j \in hp(i)} C_j \qquad \text{(ec. 5)}$$

se termina cuando

$$W^{n+1} = W^n \qquad \text{(ec. 6)}$$

o bien cuando el valor calculado es mayor que el plazo, es decir, no cumpliría el plazo temporal

$$W^{n+1} > D_i \qquad \text{(ec. 7)}$$

- El planificador *Earliest Deadline First* [80] asigna las prioridades a las tareas dinámicamente en función de sus plazos temporales y del estado del sistema. De manera que la tarea con un plazo temporal absoluto más inmediato será la tarea que tenga mayor prioridad. Es decir, ejecuta siempre la tarea más urgente independientemente del tiempo de cálculo de esa tarea. En el momento de activación de una tarea, deben recalcularse las prioridades para dar cabida a la nueva tarea. Si esta nueva tarea tiene su plazo temporal más inmediato pasara a ser la tarea que se ejecute adueñándose en este momento

del procesador. En el caso de la asignación dinámica de prioridades, la prueba de planificabilidad suficiente y necesaria es la que viene dada por la ecuación (ec. 1) que indica que en EDF siempre que la utilización del procesador sea menor o igual que 1 el conjunto de tareas es planificable con los plazos temporales de las tareas igual a sus periodos.

- El planificador *Least Laxity First* [53] asigna las prioridades dinámicamente a las tareas dependiendo del tiempo que les sobra para alcanzar el plazo temporal después de descontar el tiempo que necesita para finalizar su ejecución ($Laxity_i = D_i - WCET_i$). Este planificador puede provocar excesivos cambios de contexto en el momento en que dos tareas tengan la misma "*laxity*". Las tareas se van alternando el tiempo de procesador entre ellas, ya que la tarea que espera a ser ejecutada va aumentando su prioridad, mientras que la tarea que se está ejecutando la va disminuyendo. Sin embargo existe una modificación de este algoritmo que evita este problema [64], permitiendo una inversión de *laxity*, en caso de empate entre dos tareas se escoge la tarea con menor plazo temporal hasta su finalización o hasta la llegada de una nueva tarea.

Todos los algoritmos explicados anteriormente son óptimos, en el sentido de que si el conjunto de tareas es planificable según algún otro criterio de asignación de prioridad también será planificable usando el algoritmo anteriormente citado. En su vertiente más básica, los algoritmos anteriores, están basados en conjuntos de tareas independientes. Las tareas implementadas para los sistemas de tiempo real no son independientes entre ellas, sino que cooperan entre ellas, pudiendo precisar el uso de recursos compartidos (acceso a dispositivos hardware, a zonas de memoria en las que se precisa acceso exclusivo), precedencias temporales entre las tareas (para la correcta ejecución de una tarea es preciso que haya finalizado la ejecución de otra ejecutada anteriormente), tratamiento de tareas aperiódicas, casos excepcionales de sobrecarga (*overload*) del sistema. En estos casos más reales y completos, el conjunto de tareas que se obtiene y su planificabilidad es más complejo.

1.3 Reducción del consumo energético.

A nivel de sistema operativo y de BIOS [4] [24] [51] existen dos estándares que se implementan en la actualidad y que permiten la reducción energética, estos son:

- APM (Advanced Power Management)
- ACPI (Advanced Configuration & Power Interface)

En la Figura 6 y en la Figura 5 se muestra la relación entre componentes software y hardware para un sistema que utiliza APM y ACPI respectivamente.

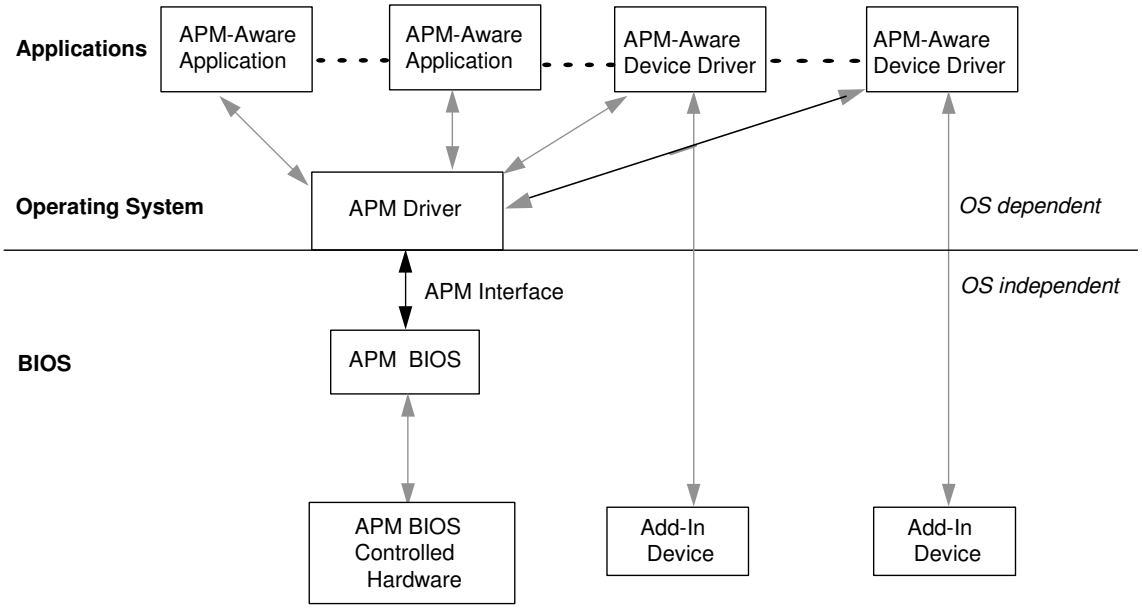

Figura 5: Relación entre los componentes software y hardware relevantes a APM [29].

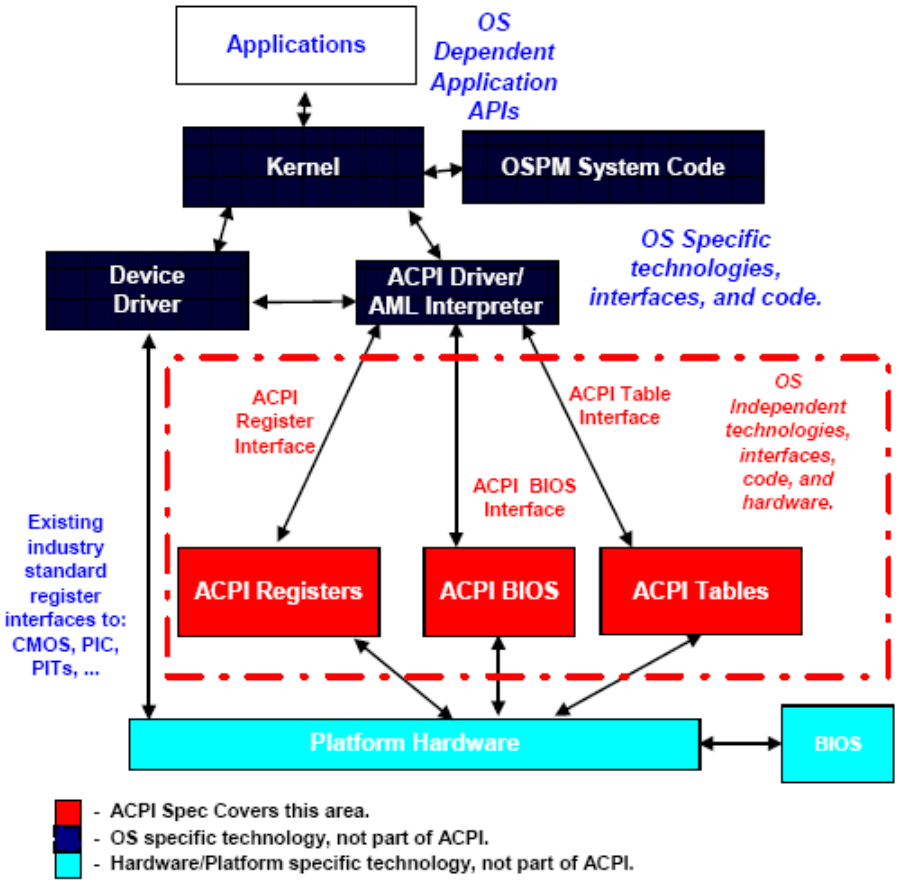

Figura 6: Relación entre los componentes software y hardware relevantes al ACPI. [28]

Estos estándares de la BIOS utilizan a nivel inferior los mecanismos de ahorro energético que proporcionan las arquitecturas de los procesadores actuales [3] [31] [69] [84] que son:

- DVS (Dynamic Voltage Scaling)[67]-[70]. Esta técnica, permite la reducción de la energía consumida por el procesador, a base de reducir el voltaje suministrado al procesador y al mismo tiempo, reduciendo proporcionalmente la velocidad de ejecución de las tareas.
- Diversos estados de alimentación energética[30]:
  - Preparado o activo: el equipo o el dispositivo está totalmente encendido y listo para su uso.
  - Inactividad: La inactividad es un estado intermedio dependiente del sistema que intenta ahorrar energía. Se entra en estado de inactividad cuando la Unidad central de proceso (CPU) está inactiva y se sabe que no

se ha producido ninguna actividad del dispositivo durante un período de tiempo especificado. El equipo no volverá al estado Activo hasta que se produzca una interrupción de hardware o se tenga acceso a cualquier dispositivo controlado. En el estado de inactividad se conservan todos los datos y parámetros operativos.

- Suspendido: El estado Suspendido de un equipo se define como el mínimo nivel de consumo de energía posible que conserva los datos y parámetros operativos. Cuando el equipo se encuentra en estado Suspendido, no se realiza ninguna operación hasta que se reanude la actividad normal. La reanudación de la actividad no se produce hasta que se produce un suceso externo, como presionar un botón, la alarma de un temporizador, etc.
- Hibernación: La hibernación guarda todo el estado del equipo y apaga la alimentación. Se trata del estado mínimo de suspensión de energía disponible y es seguro frente a cortes de alimentación eléctrica. Cuando reanuda la actividad desde un estado de suspensión hibernado, la BIOS realiza las pruebas POST normales y después lee el archivo de hibernación que se creó para guardar el estado del equipo. El equipo vuelve al último estado que tenía antes de que entrara en modo de hibernación. El modo de hibernación reduce el tiempo necesario para el inicio.
- Apagado: Cuando se encuentra en el estado Apagado, el equipo o el dispositivo no recibe alimentación eléctrica y está inactivo. En el estado Apagado puede que los datos y los parámetros operativos no se conserven.

La reducción de la energía que utiliza la BIOS y los sistema operativos vendrá dada por la detección de la inactividad del procesador así como de los diversos periféricos, sin embargo, las aplicaciones de control y algunas aplicaciones multimedia deben ejecutarse de manera continua, por ejemplo la compresión-descompresión de video o voz, y esta técnica por si sola no resulta útil. Por otra parte, estas aplicaciones no requieren un tiempo de finalización muy pequeño, sino que para obtener una buena calidad de servicio es suficiente con obtener el resultado antes de un determinado instante de tiempo, especificado durante el diseño de la aplicación. Para estas

aplicaciones resulta adecuado el uso del la técnica de DVS que consiste en reducir la tensión suministrada junto con la velocidad del procesador, siempre que la aplicación no requiera obtener el máximo rendimiento de este.

En los procesadores basados en la tecnología CMOS, la potencia consumida (P_{cmos}) depende de tres fuentes principales[19], tal como indica la siguiente formula:

$$P_{cmos} = p_t(C_L V_{dd}^2 f_{clk}) + I_{sc} V_{dd} + I_{leakage} V_{dd} \qquad \text{(ec. 8)}$$

El primer término es la componente dinámica del consumo de potencia, donde C_L es la capacitancia del procesador, f_{clk} es la frecuencia del reloj, p_t es la probabilidad de que se produzca una transición (factor de actividad), V_{dd} es el voltaje suministrado al procesador. El segundo termino es la potencia disipada en el transito desde la fuente al procesador. Finalmente el tercero es la corriente de fuga (*leakage current*) que viene determinada por consideraciones tecnológicas. La componente principal de disipación de potencia en un procesador, bien diseñado, es la componente dinámica, y es la que los diseñadores de circuitos de bajo consumo intentan minimizar, es decir, intentan disminuir el primer factor de la ecuación (ec. 8) manteniendo la funcionalidad del procesador en el sistema. En el articulo de Borkar S. [11] se especifican los retos actuales en el diseño de baja potencia, que básicamente consisten en reducir la potencia activa (utilizando técnicas de DVS o de reducción de la actividad del procesador) y la corriente de fuga.

Consideraciones generales sobre la latencia de los circuito CMOS implican una relación lineal entre el voltaje suministrado y la velocidad [77]. Por lo tanto, utilizando la técnica DVS, se puede reducir la energía consumida para ejecutar un determinado conjunto de tareas reduciendo la velocidad del procesador mediante el voltaje y consiguiendo así un decremento cuadrático en la potencia disipada. Esto implica que un pequeño decremento en la velocidad del procesador, tiene como consecuencia un importante ahorro energético. Sin embargo, al ejecutar a menor velocidad, las aplicaciones tardan más tiempo en finalizar. Teniendo en cuenta que en los sistemas con restricciones temporales, el correcto funcionamiento de la aplicación depende en gran medida del tiempo en el que se producen los resultados, se debe ser muy cuidadoso en el uso de esta técnica, utilizándola únicamente cuando el sistema no requiere la máxima capacidad del procesador.

A diferencia de los sistemas informáticos en general en los que las aplicaciones empiezan a ejecutarse y acaban, los sistemas informáticos de tiempo real, sistemas de control, tienen un funcionando periódico continuo, no acaba nunca. Si una aplicación informática se ejecuta a mayor velocidad termina antes que si se ejecuta a velocidad reducida. El consumo de energía es mayor a mayor velocidad de ejecución de las tareas, pero puede darse el caso que si la aplicación finaliza más rápidamente consuma menor cantidad de energía. Sin embargo, en las aplicaciones informáticas de tiempo real, al estar formadas por tareas periódicas, no finalizan, sino que al finalizar una instancia, deben esperar la siguiente instanciación de la tarea para continuar la ejecución, con lo que resultará energéticamente favorable ejecutar estas aplicaciones a la velocidad mínima que garantice los plazos temporales y minimice el tiempo libre de procesador.

Procesadores recientes para sistemas empotrados reducen la corriente de fuga a nivel hardware [5],[17],[31],[63] usando distintos niveles en alimentación energética: *standby*: congelado (*freeze*), hibernación (*hibernation*) y *cryo mode*; y usando técnicas *clock gating*[46] que consisten en deshabilitar el reloj de los circuitos que no se están usando, evitando de este modo la disipación de energía debida a la carga y descarga de los circuitos no usados. Sin embargo, estas técnicas necesitan conocer que circuitos deben suspender y durante cuanto tiempo, una suspensión y activación frecuente generan una sobrecarga que puede derivar en una disipación de potencia mayor que la que se hubiera tenido sin usar la técnica.

A nivel de planificación de tareas de tiempo real, también existen estudios recientes que reducen la corriente de fuga del procesador, la idea básica de estas técnicas es poner al procesador en *standby* el máximo tiempo posible, ejecutando las tareas de manera que se agrupe al máximo los intervalos en que el procesador queda libre para poder "dormirlo" el máximo tiempo posible disminuyendo el número de transiciones de potencia. Y.H. Lee [41] aplica la técnica de unión de los intervalos libres del procesador a los algoritmos de expulsivos de prioridades fijas y de prioridades dinámicas básicos ejecutándose las tareas a máxima velocidad y deteniendo el procesador cuando no existen tareas activas, en S. Irani [32] y R. Jejurikar [35] se combinan las dos técnicas: DVS y la unión de los intervalos libres de procesador.

1.4 Planificación con ahorro energético

La planificación de sistemas de tiempo real con ahorro energético se basa en la utilización de técnicas DVS, y más recientemente en algoritmos que unen los intervalos libres del procesador para reducir la corriente de fuga. Los algoritmos que utilizan las técnicas DVS, pueden calcular la velocidad del procesador estáticamente (off-line) o dinámicamente (on-line) (Figura 7). En el caso del cálculo estático, se calculan las velocidades óptimas a las que debe ir el procesador en cada momento para conseguir el mayor ahorro energético posible teniendo en cuenta el consumo de todo el WCET de las tareas y garantizando el cumplimiento de todos los plazos. Posteriormente, en tiempo de ejecución el planificador únicamente debe seguir los pasos marcados ejecutando cada tarea a la velocidad indicada. Los planificadores que usan el cálculo de velocidad del procesador estático, pueden ser planificadores estáticos, a los que se ha calculado el plan de ejecución de las tareas, o planificadores dinámicos, que se basan en la prueba de planificabilidad de los algoritmos de prioridades fijas (RMA, DMA) [36] o en la de prioridades dinámicas (EDF) [68]. En el caso de planificación con prioridades dinámicas se puede ahorrar más energía que con prioridades fijas debido a que el factor de utilización del procesador que garantizan las pruebas de planificabilidad es mayor.

Durante la ejecución de la aplicación de tiempo real, lo habitual es que no se llegue a consumir la totalidad del WCET de la tarea, generándose en este caso "*slack dinámico*", tiempo libre del procesador que se podría aprovechar para reducir la velocidad de ejecución de alguna tarea. Los planificadores que calculan la velocidad de manera estática, no permiten aprovechar el *slack dinámico*, ni permite fácilmente la ampliación del conjunto de tareas, como por ejemplo disponer de tareas aperiódicas. Sin embargo, existen planificadores mixtos, en los que el cálculo de la velocidad de proceso se realiza de manera estática, y durante la ejecución se puede reducir la velocidad para permitir aprovechar el *slack* dinámico producido por las tareas recientemente ejecutadas. Sin embargo, puede darse el caso que la tarea que ha generado el *slack* dinámico sea la última tarea que estaba lista para ser ejecutada. En este caso, el tiempo que ha dejado libre no podrá ser aprovechado por ninguna otra tarea, puesto que se deberá esperar que haya tareas en el sistema listas para ser ejecutadas.

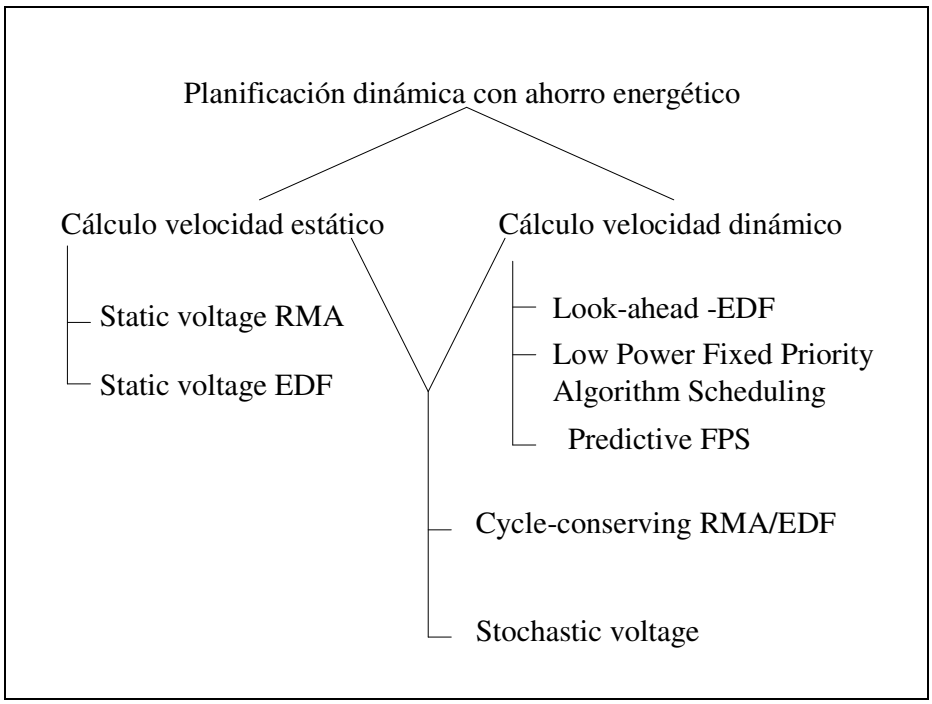

Figura 7: Clasificación de los planificadores de bajo consumo en sistemas Tiempo Real

Si el algoritmo de planificación tiene en cuenta el *slack* dinámico, otorgará cierta flexibilidad a este, permitiendo un mayor ahorro energético, en este caso el planificador reducirá "en tiempo de ejecución" la velocidad con la que ejecuta la tarea. Esta es la idea de la implementación del "*cycle conserving RMA o EDF*" [68]. Debe tenerse en cuenta que el *slack* dinámico no puede ser utilizado por cualquier tarea del sistema, sino que, ésta, debe cumplir unos determinados requisitos. El *slack* generado por una tarea de una determinada prioridad, únicamente puede ser aprovechado por las tareas de igual o menor prioridad, nunca por una tarea de mayor prioridad. Esto implica que el algoritmo deberá disponer de tantos contadores como posibles niveles de prioridad existan en el sistema. Cuando se ejecuta una tarea de una determinada prioridad, la reducción de la velocidad que se realizará será la calculada estáticamente más la que permite el *slack* dinámico de las tareas con una prioridad superior a la suya, siempre teniendo en cuenta que la reducción no provoque una perdida del plazo de la tarea que se ejecuta.

El planificador presentado por F. Gruian en [26] el "*stochastic voltage scheduler"* se basa en dividir la ejecución de las tareas en diversas fases, de manera que cada fase se ejecutará a distinta velocidad del procesador. Esta velocidad se calcula de manera estática siguiendo una distribución de probabilidades de la duración de la ejecución

de las diferentes instancias de la tarea. Las velocidades del procesador escogidas serán menores conforme haya una mayor probabilidad de ejecución y mayores conforme esta probabilidad sea menor, de manera que en caso que la instancia actual no consuma todo el WCET, la parte no consumida sea la que habría ido a mayor velocidad, y por tanto será menor el tiempo que deje libre. En este planificador tiene en cuenta el *slack* dinámico que se produce, usándolo para reducir la velocidad de la siguiente tarea de menor prioridad que se ejecutará.

Y por último, en el esquema del cálculo de velocidad totalmente dinámico la decisión sobre que tarea se ejecuta y a que velocidad de procesador, se calcula en tiempo de ejecución del sistema, consiguiendo con ello una mayor flexibilidad en la planificación de las tareas, y pudiendo en este caso añadir nuevas tareas al sistema. En este caso se deberá guardar información extra para ayudar al planificador a decidir en cada momento que tarea ejecuta y a que velocidad puede ejecutarla. Cuanta más información se guarde mayor podrá ser el refinamiento de la velocidad a la que se ejecute la tarea consiguiendo un mayor ahorro energético, sin embargo se tendrá un mayor consumo de energía debido al incremento de memoria necesaria y al algoritmo de planificación, que será más costoso en los cálculos.

En el articulo de P. Pillai [68] se desarrolla, también, un algoritmo de ahorro energético en el que el cálculo de la velocidad es totalmente dinámico, el "*Look-ahead EDF*". Este algoritmo se basa en determinar la necesidad de cálculo futuro y retardar la ejecución de las tareas al máximo. De manera que ejecuta a la mínima velocidad posible que permite ejecutar la mínima parte de tarea necesaria para asegurar el cumplimiento de todos los plazos temporales futuros. Esto requerirá que la ejecución del trabajo retardado se ejecutará a mayor velocidad consumiendo mayor energía, pero si la aplicación no consume todo el WCET, la parte que dejará de ejecutarse es la que debería ejecutar a mayor velocidad. El resultado final es un mayor ahorro energético global, no dejando tiempo libre al procesador. Una conclusión que se puede extraer de este último articulo es que los planificadores de prioridades fijas no pueden obtener un ahorro energético tan agresivo como con los planificadores de prioridades dinámicas.

A este tipo de planificadores que calculan la velocidad de manera totalmente dinámica, también, pertenece el "*Low Power Fixed Priority Algorithm Scheduling*"

desarrollado por Y. Shin y K. Choi en [77], este algoritmo es expulsivo con prioridades fijas, de manera que el planificador escoge siempre para ejecutar la tarea de mayor prioridad, reduciendo la velocidad del procesador únicamente cuando la tarea que va a ejecutarse es la única disponible para ser ejecutada. En el caso que esta tarea no consumiera todo el WCET, el procesador se quedaría libre, hasta la llegada de una nueva instancia al sistema. Este algoritmo es uno de los algoritmos base que se ha escogido para implementar nuestro algoritmo de planificación. Una posible implementación de este algoritmo se halla en el Anexo 1.

El algoritmo "*predictive FPS*" de P. Kumar [40], se basa en predecir el tiempo que consumirá la instancia que se va a ejecutar. De manera que se dice que la tarea consumirá la media aritmética de lo que han consumido las instancias ya ejecutadas de esa tarea. La reducción de velocidad del procesador se efectuará, al igual que en el algoritmo anterior, únicamente cuando la tarea es la única tarea del sistema lista para ser ejecutada. Se debe tener en cuenta que aunque con este algoritmo se logre un menor consumo energético que con el anterior algoritmo puede haber una perdida de las restricciones temporales de algunas tareas debido a que alguna instancia consuma más tiempo del anunciado por el WCET, lo que es inaceptable si el sistema es de tiempo real estricto.

Al igual que en los sistemas de tiempo real convencionales, estos planificadores se pueden ampliar para que tengan en cuenta los recursos compartidos y la secuenciación temporal entre tareas.

1.5 Objetivos de la tesis

El objetivo principal de esta tesis es el estudio de la planificación de las tareas de aplicaciones de tiempo real estricto que se ejecutaran en una arquitectura multiprocesador distribuida realizando especial énfasis en el coste energético de la aplicación. El conjunto de tareas que se estudiarán utilizan recursos compartidos y poseen dependencias temporales entre tareas ubicadas en el mismo o distinto procesador.

Para conseguir este objetivo, se han fijado distintas metas intermedias, empezando con un conjunto de tareas sencillo y añadiéndole complejidad con cada meta cumplida.

- La primera meta es la planificación de tareas que permita obtener el máximo ahorro energético posible en un procesador. Estas tareas serán de tiempo real estricto independientes, es decir, tareas que no usan recursos comunes, ni poseen dependencias temporales entre ellas. El capítulo 2 esta dedicado al desarrollo de esta meta.[56], [57],[59],[61]

- La segunda meta que se ha fijado es la de añadir recursos compartidos entre las tareas de tiempo real. Con el fin de alcanzar esta meta, se ha modificado convenientemente el algoritmo de planificación obtenido anteriormente y se realiza un estudio de como afecta dicha modificación a nivel de consumo energético. Se observará que en este caso no es posible llegar a obtener el mismo ahorro energético que se obtenía anteriormente. Esta meta es la que se ha desarrollado en el tercer capítulo del presente documento. [58]

- Y por último, la tercera y última meta será la de añadir dependencias temporales entre las tareas, de manera que una tarea no puede empezar su ejecución hasta que termina la ejecución de la tarea que la precede. En este caso, se tienen plazos temporales globales que van desde que se inicia la primera tarea de la cadena de precedencias hasta que finaliza la última. Para resolver este problema, se modificara el algoritmo de planificación obtenido en los puntos anteriores y se implementará un algoritmo de prioridades dinámicas basado en el EDF que puede adaptarse mejor a la carga del sistema

para comparar el rendimiento obtenido por el primer algoritmo respecto a la planificación dinámica con prioridades dinámicas. [60]

Tal como se ha comentado, los capítulos restantes están ordenados en función de las distintas metas planteadas. En todos los capítulos se empieza definiendo el conjunto de tareas que se quiere planificar, se presentan las soluciones de planificación que se han propuesto en la literatura y se plantea nuestra propuesta. A continuación se presenta una posible implementación de esta propuesta, y se evalúa su rendimiento. Finalizando el capítulo con las conclusiones particulares de éste.

Se ha añadido un capítulo en el que se realiza una reflexión sobre el dilema entre el ahorro energético y el tiempo de finalización a las tareas aperiódicas. En el último capítulo del documento, se extraen unas conclusiones generales del trabajo realizado y se comentan posibles líneas de investigación futuras.

En los anexos, se incluye una posible implementación realizada de los algoritmos de otros autores que se han usado a nivel de comparativa con nuestros algoritmos.

Capítulo 2

PLANIFICACIÓN CON AHORRO ENERGÉTICO EN SISTEMAS DE TIEMPO REAL

El objetivo de este capítulo es la implementación de un algoritmo en-línea de planificación que aporte ahorro energético a la ejecución de conjuntos de tareas independientes de tiempo real estrictos. En este capítulo se estudiará el consumo energético del algoritmo implementado, comparándolo con otros algoritmos de la literatura y con el consumo energético que se obtendría si se ejecutarán todas las tareas a velocidad máxima, deteniendo el procesador cuando no existen tareas listas para ser ejecutadas.

2.1 Introducción.

El objetivo principal de este capítulo es: el estudio de la planificación en-línea para la ejecución de tareas independientes en un sistema de tiempo real estricto que aporte ahorro energético a la ejecución de tareas independientes, y el desarrollo de un planificador que mejore el ahorro energético de los algoritmos existentes sin añadir complejidad de implementación ni de ejecución al algoritmo de planificación. Se centrará el estudio en los algoritmos de planificación en-línea, con asignación de prioridades fijas y dinámicas. La implementación del algoritmo de planificación tiene como requisito que sea lo más simple posible, tanto en el uso de la memoria como en el uso del procesador, dado que su ejecución también consume recursos del sistema, pudiéndose llegar al extremo que aún reduciendo la energía necesaria para ejecutar las tareas, se aumentára la energía utilizada para planificarlas, con lo que el resultado global a nivel energético no seria aceptable.

Tal como se expone en el capítulo de introducción, las tareas de tiempo real vienen especificadas principalmente por: el peor tiempo de ejecución de la tarea (WCET), el periodo (T) y el plazo temporal (D). El WCET, es el límite superior que tardará la tarea en ejecutarse a máxima velocidad del procesador sin tener en cuenta las posibles interferencias con otras tareas, debidas al algoritmo de planificación utilizado. Las interferencias entre las distintas tareas del conjunto serán mayores o menores dependiendo de la asignación de prioridades, de los periodos de las tareas, del instante actual y del algoritmo de planificación utilizado. Se define Umax como la utilización máxima de la tarea, es decir, WCET/T. Las tareas aperiódicas vienen especificadas por su tiempo de ejecución máximo WCET y por su intervalo de llegadas de media T.

Desde el punto de vista de la planificación de tareas, se ha comentado anteriormente que una de las técnicas que permite la reducción de la energía consumida por el procesador al ejecutar la aplicación de tiempo real es la aplicación de las técnicas del ajuste dinámico del voltaje *"Dynamic Voltage Scaling" (DVS)* [68]. Esta técnica, permite la reducción de la energía consumida por el procesador, reduciendo el voltaje suministrado al procesador y al mismo tiempo, reduciendo proporcionalmente la velocidad de ejecución de las tareas. De manera que si se reduce la velocidad de ejecución en un factor α ($0 < \alpha \leq 1$) se aumenta el tiempo necesario para ejecutar

todo WCET de la tarea en un factor de $1/\alpha$. Por tanto, en primera aproximación, se supondrá que el tiempo máximo necesario para ejecutar el WCET de la tareas será inversamente proporcional a la velocidad con la que se ejecutemos las tareas, mientras que esta reducción en la velocidad del procesador no afecta ni al periodo ni al plazo temporal de las mismas. Dado que el entorno en el que nos situamos es el de tiempo real estricto, se debe asegurar siempre el cumplimiento de todos los plazos temporales de todas las tareas del sistema, aun ejecutándose a velocidades reducidas del procesador. El análisis de planificabilidad realizado antes de poner en funcionamiento el sistema debe suponer que se consumirá todo el WCET. Aun así, durante la ejecución real del sistema se consumirá únicamente un porcentaje del WCET. El porcentaje de tiempo consumido variará de una activación de la tarea a la siguiente, dependiendo del estado del sistema en el momento de la ejecución de la tarea, de los datos de entrada, de la ejecución de tareas precedentes, o incluso de la ejecución de activaciones previas de esta misma tarea. Por esta razón, no se puede determinar con antelación a la ejecución, el porcentaje de WCET que consumirán las diversas activaciones de las tareas. Por otra parte, si el cálculo de la velocidad del procesador es dinámico, el tiempo reservado para la ejecución de una tarea no usado (*slack dinámico*) puede aprovecharse para la reducción de la velocidad de ejecución de las siguientes tareas, reduciendo la cantidad de energía total necesaria para la ejecución de la aplicación.

Los planificadores que aplican las técnicas DVS, pueden ejecutar las tareas periódicas a velocidad reducida, sin comprometer los plazos temporales. Esta reducción de la velocidad, provoca que las tareas periódicas retarden su finalización, ahorrando con ello energía. Por otra parte, en la literatura de tiempo real, existen otro tipo de planificadores que tienen el mismo efecto sobre este tipo de tareas, son los planificadores que aportan una buena calidad de servicio (en forma de tiempo de finalización) a las tareas aperiódicas. En este caso se retarda la ejecución de las tareas periódicas para permitir la ejecución de las tareas aperiódicas. Nuestra idea consiste en usar la combinación de los ambos tipos de algoritmos para que cuando no existan tareas aperiódicas listas para ser ejecutadas, reducir la velocidad de ejecución de las tareas periódicas, consiguiendo con ello una reducción de la energía necesaria para la ejecución de toda aplicación de tiempo real, además de conseguir una buena calidad de servicio a las tareas aperiódicas.

Cuando el planificador ha decidido que tarea tiene que ejecutar en un instante determinado, éste tiene que calcular, además, a que velocidad ejecutará esta tarea, para ello, se debe tener en cuenta que aun cuando el cálculo de la velocidad del procesador es continuo, las arquitecturas existentes [31] [84] en la actualidad, no abarcan todo el espacio continuo, sino un rango discreto de velocidades. Por tanto, la velocidad utilizada por el sistema será siempre igual o mayor que la velocidad calculada por el algoritmo de planificación, pudiendo así asegurar siempre, el cumplimiento de las restricciones temporales de todas las tareas.

A continuación, se presentarán brevemente los dos algoritmos en los que nos basamos para desarrollar nuestro propio algoritmo de planificación con ahorro energético. Los detalles de ambos algoritmos se hallan en los anexos de la presente tesis. El primer algoritmo se usa para calcular la velocidad del procesador mientras que el segundo es el que se usará para planificar el conjunto de tareas:

- El "*Low Power Fixed Priority Algorithm Scheduling*" *(LPFPS)* propuesto por Y. Shin y K. Choi [77] implementado para proporcionar ahorro energético aplicando las técnicas de DVS (Anexo 1).
- "*Dual Priority Scheduling Algorithm"(DPS)* propuesto originalmente por A. Burns y A.J. Wellings [13] para conseguir maximizar la planificación de un conjunto de tareas periódicas independientes, y modificado posteriormente por R. Davis y A.J. Wellings [23] para dar una buena calidad de servicio a las tareas aperiódicas (Anexo 2).

El nuevo algoritmo de planificación con ahorro energético que se busca es pues: un algoritmo de planificación expulsivo de tareas periódicas y aperiódicas de tiempo real enmarcado en el entorno de prioridades estáticas y cálculo dinámico de la velocidad del procesador, que ofrezca, un buen tiempo de servicio a las tareas aperiódicas. Una vez desarrollado el algoritmo, se realizaran una serie de consideraciones sobre la adecuación de la reducción de velocidad del procesador a la ejecución de las tareas, finalizando el capítulo con el desarrollo de una mejor elección de la velocidad de ejecución de las tareas por parte del algoritmo propuesto inicialmente. Para finalizar el capítulo se realizará una comparativa entre distintos algoritmos de planificación.

2.1.1 Low Power Fixed Priority Algorithm Scheduling

El algoritmo Low Power Fixed Priority Algorithm Scheduling (LPFPS) [77] es un planificador de tiempo real con ahorro energético del conjunto de planificadores en-línea con asignación de prioridades estáticas y cálculo dinámico de la velocidad de procesador. Este planificador es eficiente desde el punto de vista de uso de procesador, de memoria, siendo además su implementación muy simple. La reducción energética se basa en aplicar las técnicas del DVS **reduciendo la velocidad de ejecución del procesador siempre que haya una única tarea lista para ser ejecutada o bien detener el procesador cuando no hay tareas listas para su ejecución**. Los detalles de una posible implementación se hayan en el Anexo 1.

Los resultados obtenidos comparando el funcionamiento de LPFPS con el funcionamiento de "*Fixed Priority Scheduling*" FPS[4], llevan a la conclusión de que la reducción energética que se obtiene con el LPFPS, no depende de la utilización del procesador, sino de la relación entre los periodos de las tareas y de la distribución de la carga del procesador entre las tareas, de manera que se obtiene una mayor reducción cuando el procesador esta ocupado por pocas tareas de periodos grandes en comparación con el WCET.

Las ventajas que presenta el algoritmo LPFPS son:

- Implementación sencilla y eficiente en el uso de procesador y de memoria.
- Permite predecir la planificabilidad del conjunto de tareas de manera estática antes de iniciar la ejecución del sistema de tiempo real al igual que FPS, garantizando el cumplimiento de todos los plazos temporales de las tareas durante la ejecución del sistema.
- Es un algoritmo estable frente a situaciones de sobrecarga del sistema, es decir, en el caso extremo que alguna tarea consuma más tiempo que el *WCET* calculado, se puede asegurar que en el caso que no se puedan ejecutar todas las tareas cumpliendo sus plazos temporales, la tarea o tareas que perderán el plazo serás las que tengan una menor prioridad, nunca perderá el plazo una tarea de prioridad mayor a la que ha provocado la sobrecarga del procesador.

[4] Rate Monotonic es una de las asignaciones de prioridades más comunes para un FPS en el que las prioridades de las tareas vienen establecidas por su periodo. A menor periodo, mayor prioridad.

- Aprovechamiento implícito del *slack* dinámico, si una tarea finaliza sin haber consumido todo su WCET permite que la siguiente tarea empiece antes, con lo que también acabará antes y dejará al final, un intervalo mayor en el que si existe una única tarea activa en el sistema, se podrá reducir la velocidad del procesador durante más tiempo. Si los periodos del conjunto de tareas son armónicos, solo habrá una tarea que se ejecutará a velocidad reducida (habrá instantes en los que todas las tareas se activarán a la vez), pero el algoritmo agrupará todo el espacio libre del procesador, con lo que en este caso se reducirá la corriente de fuga. Si los periodos no son armónicos, los intervalos en los que el procesador puede aprovechar para reducir velocidad serán pequeños y esparcidos por todo el hiperperiodo. No obstante, en este caso, habrá más tareas que se ejecuten reduciendo su velocidad de proceso.

El inconveniente principal del algoritmo LPFPS es que:

- El cálculo de velocidad del procesador se realiza suponiendo que la última tarea consumirá todo el WCET, sin embargo esta suposición no siempre será cierta. Y en ese caso, no existirá ninguna otra tarea que pueda ejecutarse a velocidad reducida, dejando así libre al procesador, hasta que se active una nueva tarea.

Para intentar reducir el impacto de este inconveniente, aparecen algunos algoritmos de planificación que basados en heurísticas que intentan predecir el porcentaje de WCET que usará una tarea basándose en su ejecución histórica. A continuación se detallan unos ejemplo de este tipo de heurísticas.

La primera heurística consiste en guardar la historia de las diversas activaciones de las tareas, es decir, supone que si una tarea anteriormente ha consumido el 50% de su WCET, es muy probable que la siguiente activación de esa tarea, también consuma el 50% de su WCET. P. Kumar y M. Srivastava en [40] implementan esta tipo de heurística sobre el algoritmo de LPFPS. En el citado articulo el cálculo de la velocidad del procesador está basada en la media del tiempo de ejecución de las activaciones precedentes, no en el WCET. Este hecho implica que si la tarea consume más tiempo, con este algoritmo se pueda perder el plazo temporal, puesto que se habrá calculado una velocidad menor de procesador de la que hubiera sido necesaria para ejecutarse dentro de sus límites. La sobrecarga necesaria para la

implementación del algoritmo, consiste únicamente en mantener en memoria una variable extra por tarea que contenga la media de ejecución de las diversas activaciones de esa tarea, y el cálculo de esta media cuando la instancia en curso de la tarea ha finalizado. El articulo concluye que aunque se consigue una considerable mejora en el ahorro energético, existen algunas tareas que pierden su plazo temporal, siendo pues inaceptable en sistemas de tiempo real estricto. Estos problemas podrían minimizarse utilizando técnicas de inteligencia artificial, aunque estas serian más costosas de implementar.

En nuestro caso, esta heurística no nos interesa, puesto que estamos en un entorno de tiempo real estricto, pero si que podría resultar interesante su aplicación en un entorno multimedia o en un entorno donde sea suficiente asegurar el cumplimiento del plazo temporal de un cierto porcentaje de tareas, pudiéndose perder un porcentaje de plazos temporales cada cierto intervalo (*firm real time*).

La segunda heurística, es la que aplica F. Gruian en [26] y se divide en dos fases, una estática y otra dinámica. En la fase estática, antes de empezar la ejecución del sistema se calcula en cuanto puede incrementarse el tiempo necesario para ejecutar el WCET de las tareas sin perder ningún plazo temporal (a que velocidad mínima pueden ejecutarse las tareas sin perder ningún plazo temporal) . Después, se divide el tiempo disponible para ejecutar cada tarea en intervalos, de manera que el primer intervalo se ejecutará a una velocidad menor, y se irá aumentando gradualmente la velocidad conforme la tarea vaya avanzando en su ejecución, así, en el caso más realista de que la tarea no consuma todo el WCET, la parte no ejecutada será la que el procesador debería haber ejecutado a mayor velocidad y por tanto la energía final consumida será menor.

Dinámicamente, cuando el sistema ya está ejecutándose, se usará el *slack* dinámico que vayan produciendo las tareas que finalizan sin llegar a consumir la totalidad del WCET, para ejecutar tareas con una prioridad menor a la que ha generado el *slack*, con una reducción mayor de la velocidad del procesador que la indicada por el análisis estático. Con esta heurística, se obtiene una mayor reducción energética que usado el LPFPS sin comprometer ningún plazo temporal, pero la implementación del planificador requiere de tantos contadores como niveles de prioridad para poder contabilizar el *slack* generado en cada uno de los distintos niveles de prioridad. A si

mismo, en el momento de ejecutar la tarea el planificador debe recorrer todos los contadores de prioridad superior que puedan aportar *slack* a la tarea activa. Este algoritmo también presenta el problema de que no permite una ampliación del conjunto de tareas de una manera fácil, por ejemplo añadir tareas aperiódicas al sistema. En este caso, en el momento de la llegada de la tarea aperiódica se debe rehacer el análisis de planificabilidad para calcular la nueva velocidad de ejecución que asegure la planificabilidad.

2.1.2 Dual Priority Scheduling Algorithm.

El algoritmo de prioridad dual "*Dual Priority Scheduling Algorithm"(DPS)* (Prioridad Dual) propuesto originalmente por A. Burns y A.J. Wellings [13] para conseguir maximizar la planificación de tareas periódicas, y modificado posteriormente por R. Davis y A.J. Wellings [23] para mejorar el tiempo de finalización para las tareas aperiódicas. Este es el algoritmo que se usará como base para la implementación de nuestro planificador de ahorro energético que elige que tarea debe ejecutar.

La idea del algoritmo de prioridad dual consiste, tal como ya se ha indicado anteriormente, en retardar las tareas periódicas sin llegar a comprometer los plazos temporales, dando al mismo tiempo un buen servicio a las tareas aperiódicas que lleguen al sistema. Para eso, las tareas periódicas poseen dos niveles distintos de prioridades, en el nivel de prioridad superior "*Upper Run Queue*" (URQ), únicamente podrán recibir interferencias de tareas periódicas de mayor prioridad, y en el nivel de prioridad inferior "*Lower Run Queue*" (LRQ) podrán recibir interferencias de las tareas aperiódicas y de las tareas periódicas de mayor prioridad. Las tareas aperiódicas se hayan en un nivel intermedio (MRQ), ordenadas mediante un algoritmo FIFO (*First In First Out*), es decir el primero en entrar es el primer en salir. Los detalles de una posible implementación se hayan en el Anexo 2.

En resumen, el funcionamiento de este algoritmo de planificación permite que las tareas periódicas retrasen su ejecución el máximo tiempo posible (teniendo en consideración la máxima interferencia que pueden tener), siempre que exista alguna tarea aperiódica en el sistema.

La dificultad principal de este algoritmo radica en el cálculo del tiempo máximo de finalización (WCRT), calculado suponiendo la máxima interferencia de las tareas de mayor prioridad, por tanto será una cota superior. En el instante crítico, la tarea tendrá esta interferencia máxima, pero normalmente, la interferencia será siempre menor. Si se mejorará esta cota, acercándola más a la interferencia real, los resultados que se obtendrían a nivel de tiempo de finalización para las tareas aperiódicas mejorarían. Este mejora la ha desarrollado G. Bernat en [10] aplicada, entre otras, a mejorar el tiempo de finalización de las tareas aperiódicas.

2.2 Metodología y desarrollo.

En este punto se expondrá la idea básica de la implementación de nuestro algoritmo de ahorro energético. El algoritmo "*Low Power Fixed Priority Scheduling"* (LPFPS) propuesto por Y. Shin y K. Choi [77] tiene características similares de reducción de velocidad a las que deseamos. Este algoritmo, utiliza un esquema de planificación con prioridades fijas pero posee un factor limitante de la reducción energética, para garantizar el cumplimiento de los plazos temporales de todas las tareas sólo se puede reducir la velocidad del procesador cuando existe una única tarea en la cola de tareas listas para ser ejecutadas. El algoritmo "*Dual Priority Scheduling"* (DPS) propuesto R. Davis y A.J. Wellings [23], está pensado para ofrecer un tiempo de finalización corto para las tareas aperiódicas sin complicar el algoritmo de planificación en exceso. En el algoritmo, las tareas disponen de prioridad dual, y en su implementación se dispone de dos niveles de prioridades distintos. Combinando ambos algoritmos, y teniendo en cuenta que las tareas del nivel de prioridad menor no corren ningún riesgo de perdida de plazos temporales, se podrá reducir la velocidad del procesador siempre que exista una única tarea en el nivel superior de prioridades, pudiendo haber, varias tareas listas para su ejecución en el nivel inferior.

Para nuestro algoritmo, se usara la parte del algoritmo LPFPS que realiza el cálculo de la velocidad del procesador, y la parte de elección de tareas que utiliza el algoritmo DPS que permite retardar al máximo las tareas periódicas siempre que sea posible finalizando su ejecución lo más tarde posible sin complicar el algoritmo de planificación de las tareas. Por lo tanto el "*Power Low Modified Dual Priority Scheduling"* (PLMDP) consiste en ralentizar la ejecución de las tareas para permitir

obtener ahorro energético aplicando las técnicas del DVS en un entorno en el que las tareas disponen de prioridad dual.

En el apartado 2.2.1 se desarrolla y analiza en detalle la modificación que se propone del algoritmo DPS propuesto R. Davis y A.J. Wellings [23] para poder obtener ahorro energético, se utilizará el posible retardo en la ejecución de las tareas periódicas para ejecutarlas a la mínima velocidad posible, prolongando su ejecución al máximo sin perder ningún plazo temporal. Esta ralentización de tareas se enmarca directamente en el proceso de DVS, permitiendo así, el ahorro energético. El intervalo de tiempo del que se dispone para ejecutar a velocidad reducida viene determinado por el tiempo de promoción de las tareas periódicas. O sea, siempre que no haya tareas en la URQ o en la MRQ se podrá reducir la velocidad del procesador. En caso de que tampoco existen tareas en la LRQ entonces se podrá "desconectar" el procesador.

2.2.1 Power Low Modified Dual Priority Scheduling.

En esta sección se veran las modificaciones propuestas al algoritmo de prioridad dual para obtener ahorro energético aplicando las técnicas del DVS, siempre que sea posible sin comprometer los plazos temporales de las tareas del sistema. En el caso de que existan tareas aperiódicas se les intentará ofrecer una buena calidad de servicio.

Para obtener ahorro energético usando el algoritmo DPS, es necesario que cuando el planificador halla seleccionado la tarea activa, calcule la velocidad de procesador a la que debe ejecutar la tarea para respetar todos los plazos temporales. Recordemos que el planificador DPS dispone de tres niveles de prioridades URQ, MRQ y LRQ, y que la tarea activa será la primera del nivel de mayor prioridad. Con el fin de restringir las expulsiones (aumentando la sobrecarga del sistema) de tareas que se producen debido a las promociones de estas a las colas de un nivel superior, y con la finalidad de ganar eficiencia con el algoritmo de planificación, la cola LRQ se ordenará por tiempo de promoción en lugar de por prioridades como estaba en el algoritmo original.

El algoritmo "*Power Low Modified Dual Priority Scheduling Algorithm*" (PLMDP) es una combinación de los algoritmos de planificación explicados en los

subapartados anteriores: por una parte se usa los mismos criterios de planificación que el algoritmo del DPS [23] para escoger la tarea que se va a ejecutar, y por otra parte usa la fórmula del cálculo de la velocidad del procesador del LPFPS [77] para determinar la velocidad de proceso de la tarea activa. Se puede observar que con el algoritmo DPS en caso extremo que siempre haya tareas aperiódicas en la MRQ, las tareas periódicas se ejecutan siempre a partir de su tiempo de promoción, se retrasa, por tanto, la ejecución de las tareas periódicas hasta el momento de su promoción quedando garantizado el cumplimiento de todos los plazos. Este fenómeno es equivalente a reducir la velocidad de proceso de las tareas periódicas hasta la llegada de la promoción. A partir de la promoción de las tareas se aplicará el algoritmo LPFPS a la cola URQ, por lo que se podrá reducir la velocidad de proceso aunque existan varias tareas activas en la LRQ. Si existen tareas aperiódicas para ser ejecutadas, en la MRQ, se ejecutaran a máxima velocidad.

Las estructuras de datos necesarias para implementar el PLMDP son las mismas que se necesitan para implementar el DPS, un esquema de las cuales se haya en la Figura 79 del Anexo 2.

En la Figura 8 se muestra el pseudocódigo del algoritmo PLMDP. Este algoritmo, tiene en cuenta los siguientes casos:

- Si existen tareas en la URQ, la tarea activa es la primera de la cola URQ (τ_i):
- Si existen otras tareas en la cola URQ o en la MRQ se debe ejecutar τ_i a máxima velocidad.
- Si es la única tarea de la cola URQ y no hay tareas en la MRQ entonces
  - Se puede ralentizar la ejecución de la tarea τ_i hasta la llegada de otra tarea τ_k vía promoción de la LRQ, hasta completar la ejecución de τ_i o hasta la llegada del plazo temporal de τ_i.
  - La velocidad de ejecución se calcula mediante la siguiente fórmula:

$$Velocidad = trabajo / tiempo\ disponible \qquad \text{(ec. 9)}$$

  - Si se denota Tp_k como el tiempo de promoción de cualquier tarea τ_k ($\tau_k \neq \tau_i$) más próximo al instante actual (tc), Td_i el plazo temporal de τ_i, R_i el remanente de la ejecución de τ_i, entonces la formula de la velocidad quedaría:

$$Velocidad = \frac{\min(Tp_k - tc, R_i)}{\min(Tp_k, Td_i) - tc} \qquad \text{(ec. 10)}$$

- Si existen tareas en la MRQ y no en la URQ se activa la primera de la cola MRQ (tarea aperiódica).
  - La tarea activa se ejecuta a máxima velocidad hasta que promocione una tarea τ_k desde la LRQ o hasta completar la ejecución de la tarea aperiódica.
- Si existen tareas en la LRQ pero no en la URQ ni en la MRQ se activa la primera de la cola LRQ (τ_i):
  - Si el tiempo de promoción Tp_k es inferior al tiempo de promoción de la tarea activa Tp_i entonces el tiempo disponible para hacer parte del trabajo es el intervalo comprendido desde instante actual *tc* y el instante en que llega la tarea k (Ta_k), es decir Ta_k - *tc*. Esto es debido a que la LRQ está ordenada siguiendo los instantes de promoción. Por tanto, si llega al sistema una tarea con menor tiempo de promoción que la tarea activa, se producirá una expulsión. Y dado que no se sabe si se tendrá oportunidad de volver a reducir la velocidad de ejecución, la mejor alternativa consiste en que esta reducción sea lo más drástica posible, por lo tanto se ejecutará a la mínima velocidad que sea eficiente [52].
  - Si por el contrario, el tiempo de promoción Tp_k es superior al tiempo de promoción de la tarea activa Tp_i, se puede ralentizar la ejecución de τ_i como si se tratase la última tarea en la URQ, por lo tanto el cálculo de la velocidad es:

$$Velocidad = \frac{\min(Tp_k - Tp_i, R_i)}{\min(Tp_k, Td_i) - tc} \qquad \text{(ec. 11)}$$

- Por último, si no existen tareas ni en la URQ, ni el la LRQ, "se desconectará" el procesador y se programará un temporizador para activar al procesador cuando se active una nueva instancia de alguna tarea.

```
L1    si no vacia(URQ) entonces
L2        Tarea activa = URQ.head;
L3        si URQ.head.next = NIL y empty(MRQ) entonces
L4            Velocidad = min(Tp_k − tc, R_i) / min(Tp_k, Td_i) − tc ;
L5        sino
L6            Velocidad = Maxima velocidad;
L7        finsi
L8    sino
L9        si no vacio(MRQ) entonces
L10           Tarea activa = MRQ.head;  Velocidad = Maxima velocidad
L11       sino
L12           si no vacia(LRQ) entonces
L13               Tarea activa = LRQ.head;
L14               si Tp_k < Tp_i entonces
L15                   Velocidad = Mínima velocidad;
L16               sino
L17                   Velocidad = min(Tp_k − Tp_i, R_i) / min(Tp_k, Td_i) − tc ;
L18               finsi
L19           sino
L20            progr. temporizador a (next Ta_i − wake up delay);
L21            Ponerse en modo power-down;
L22           finsi
L23       finsi
L24   finsi
L25   ejecutar la tarea activa a la velocidad calculada;
```

Figura 8: Pseudocódigo del algoritmo de planificación *Power Low Modified Dual-Priority* (PLMDP).

2.2.2 Ejemplo de funcionamiento del PLMDP.

A continuación se muestra un ejemplo de comportamiento del PLMDP usando el conjunto de tareas que se detalla en la Tabla 1. En el ejemplo, se describe como será el comportamiento de las tareas a nivel de planificación y a nivel energético (ver Figura 9) suponiendo que las tareas consumen el 100% de su WCET, aunque en la ejecución normal únicamente consumirán un porcentaje de este tiempo. En este primer ejemplo, para simplificar el comportamiento del algoritmo se supone que no llega ninguna tarea aperiódica al sistema.

Task	T	D	WCET	WCRT	D-WCRT	P
t1	50	50	10	10	40	1
t2	80	80	20	30	50	2
t3	100	100	40	80	20	3

Tabla 1: "Benchmark" propuesto por Y. Shin y K. Choi [77] para la evaluación del ahorro energético. La notación correspondiente es: T = periodo, D = plazo temporal, WCET = peor tiempo de cálculo, WCRT = peor tiempo de finalización, P = prioridad, a menor número mayor prioridad.

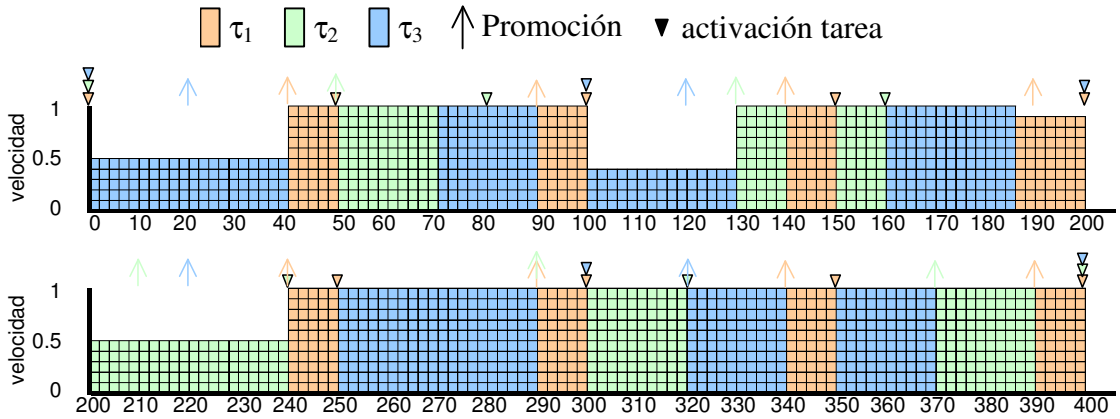

Figura 9: Diagrama de planificación de procesos usando el algoritmo PLMDP cuando las tareas consumen el 100% del WCET.

En la Figura 9 se representa el resultado de la planificación PLMDP suponiendo que todas las tareas consumen el 100% de su peor tiempo de cálculo y que no existen tareas aperiódicas. Esta situación permite explicar claramente el funcionamiento básico del algoritmo.

En el instante tc=0, se activan las tres tareas τ_1, τ_2 y τ_3 que se encolaran en la LRQ (cola de tareas listas para ejecutar de menor prioridad) ordenadas según su tiempo de promoción. En este caso τ_3 es la que está en primer lugar, le sigue τ_1 y finalmente τ_2.

En tc=0 empieza la ejecución de la tarea τ_3. En este instante no hay tareas en la URQ (cola de tareas listas para la ejecución de mayor prioridad) por lo tanto se puede reducir la velocidad del procesador. Se aplica la fórmula (9), la primera promoción es la de τ_3 que se producirá en t=20 pero esta es la promoción de la tarea activa $Tp_i=Tp_3=20$, por tanto se debe mirar cuando se producirá el siguiente instante de promoción que será el de τ_1 en t=40, por tanto, $Tp_k=Tp_1=40$, el tiempo actual (tc) es igual a 0, y el remanente $(R_3)=40$, el mínimo entre Tp_k-Tp_i y R_i será de 20 unidades temporales. Se ejecutan 20 unidades de la tarea τ_3. En el denominador de la ecuación hay $Tp_k=40$ y $Td_i=100$, por tanto el mínimo es 40. Si se resta tc=0 son 40, por lo que la tarea puede ejecutarse durante un tiempo de 40 unidades sin recibir interferencias

de otras tareas. Resumiendo, se ejecutan 20 unidades útiles de trabajo durante 40 unidades de tiempo, la velocidad del procesador será el 50% de la velocidad máxima.

En t=40, promociona la tarea τ_1 de mayor prioridad, τ_3 no ha finalizado y le ha llegado su promoción, por tanto en la URQ hay τ_1 y τ_3, con R_1=10 y R_3=20 respectivamente. Mientras exista más de una tarea en la URQ se deberán ejecutar a máxima velocidad del procesador. Al promocionar, τ_1 ha expulsado τ_3 del procesador. Se ejecuta τ_1 a máxima velocidad y finaliza cuando tc=50.

En t=50, τ_1 termina su ejecución, le llega su plazo temporal y se activa una nueva instancia de τ_1 que se que se encola en la LRQ. En este instante también promociona τ_2 que pasa a la cola URQ. En la URQ estan τ_2 y τ_3. El planificador escoge a τ_2 asignándole el procesador. Esta tarea debe ejecutarse también a máxima velocidad. La ejecución de todas las tareas continua a máxima velocidad hasta el instante t=100.

En t=100, el escenario es exactamente el mismo que en t=0, se permite la reducción de la velocidad del procesador para ejecutar la tarea τ_3 mientras no promocione ninguna tarea a la URQ.

En t=200, la tarea de mayor prioridad de la LRQ es la τ_2, y por tanto, será la que se ejecutará a velocidad reducida, por último se podrá volver a reducir la velocidad del procesador en t=270.

Al igual que en la planificación LPFPS de la Figura 78 el procesador no tiene tiempo libre de procesador, con lo que se ha ejecutado el conjunto de tareas a la mínima velocidad promedio, obteniendo una reducción en la energía consumida por el sistema sin comprometer ningún plazo temporal de las tareas.

Para finalizar la comparación, se analiza el rendimiento energético del planificador PLMDP con el LPFPS. Al ejecutar todas las tareas el 100% del WCET, ambos algoritmos consumen prácticamente lo mismo dado que el esquema final de la ejecución de las tareas no dispone de intervalos de tiempo libres (el sistema está siempre ejecutando alguna tarea). En las Figura 11 y en la Figura 12 se muestra la ejecución de ambos algoritmos cuando las tareas consumen siempre el 50 % de su WCET, para que sin complicar en exceso el esquema de planificación, se pueda comparar el comportamiento de ambos algoritmos en un escenario más adecuado para observar la diferencia de ahorro energético.

En la Figura 11 se muestra como sería la ejecución de las tareas de la Tabla 1 siguiendo el algoritmo FPS cuando todas las tareas consumen siempre el 50% de su peor tiempo de ejecución, y en la Figura 11 se muestra el resultado de la ejecución con el LPFPS con un consumo del 50% del WCET.

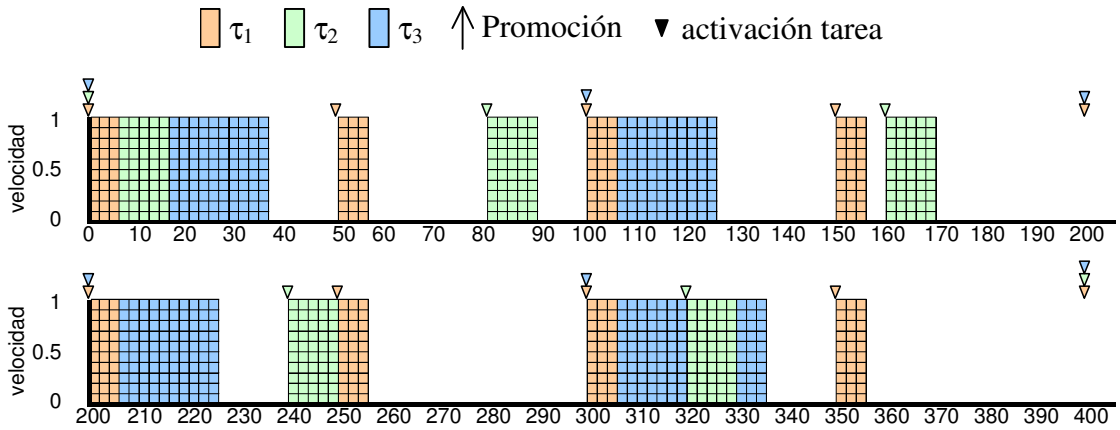

Figura 10: Diagrama de planificación de procesos usando el algoritmo FPS cuando todas las tareas consumen el 50 % de su WCET.

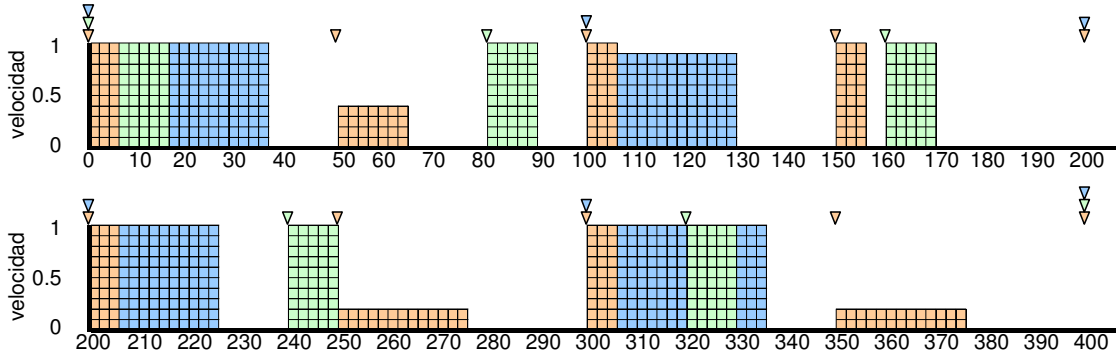

Figura 11: Diagrama de planificación de procesos usando el algoritmo LPFPS cuando todas las tareas consumen el 50 % de su WCET.

A continuación se analizará el diagrama de la Figura 11. En t=0, llegan al sistema las tareas τ_1, τ_2 y τ_3. Empieza a ejecutarse la tarea τ_1 debido a que es la tarea con mayor prioridad, esta se ejecuta durante 5 unidades de tiempo y cuando finaliza, el algoritmo elige la τ_2 para su ejecución. τ_2 consume 10 unidades de tiempo.

En t=15, finaliza τ_2, el planificador elige τ_3 para su ejecución. Al ser esta la única tarea que hay lista para ser ejecutada, se podrá reducir su velocidad de ejecución. La siguiente tarea que llegará para ser ejecutada será τ_1 en el instante t=50, esto implica que τ_3 estará sola en el sistema durante 35 unidades de tiempo, como el WCET es de 40 unidades de tiempo, no puede reducir su velocidad de ejecución, y se ejecutará a

velocidad máxima. Pero τ_3 sólo consume el 50% de su WCET, con lo que finaliza en t=35.

En t=35, el procesador se quedará libre durante 15 unidades de tiempo. Este tiempo no puede ser aprovechado por ninguna tarea, ya que el planificador le ha calculado la velocidad de ejecución de la tarea τ_3 para que pudiera ejecutar todo el WCET. Al llegar τ_1 y ser esta la única tarea lista para ser ejecutada, el planificador se plantea una nueva reducción de velocidad del procesador.

En t=80 llega la tarea τ_2. El WCET de τ_1 es de 10 unidades, por tanto hay 30 unidades de tiempo para ejecutar 10, se puede establecer que la velocidad del procesador sea 10/30 de la velocidad máxima. Pero nuevamente la tarea τ_1 sólo ejecuta el 50% del WCET, por tanto esta consume únicamente 5 unidades de cálculo en 15 unidades de tiempo, dejando al procesador libre durante otras 15 unidades de tiempo.

En t=80 cuando llega la tarea τ_2 se plantea una nueva reducción de velocidad del procesador.

En t=100, llega la tarea τ_2 con WCET de 20 unidades, que son precisamente las que dispone el planificador para ejecutar la tarea, por lo tanto hay que ejecutar τ_2 a velocidad máxima, pero al finalizar antes de lo esperado, deja nuevamente libre el procesador. La ejecución del resto del hiperperiodo continuaría de manera equivalente.

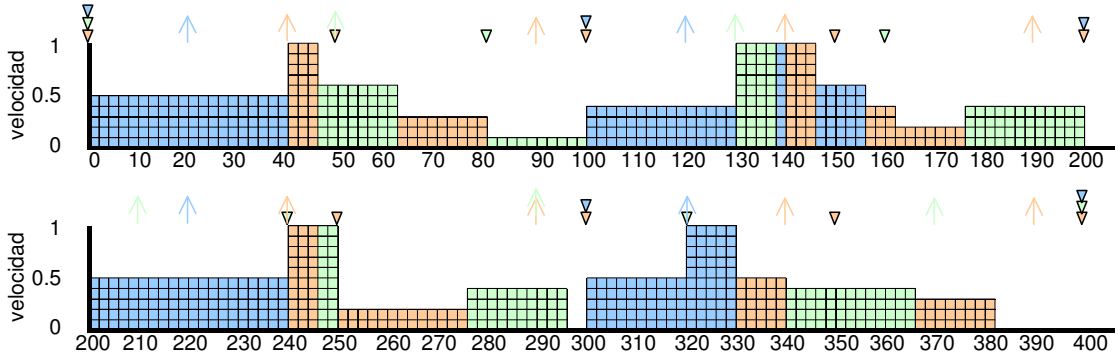

Figura 12: Diagrama de procesos del algoritmo de ahorro energético PLMDP, cuando las tareas consumen el 50% WCET.

En la Figura 12 se muestra como sería la ejecución de las tareas siguiendo el algoritmo PLMDP cuando estas consumen el 50% de su peor tiempo de ejecución.

En t=0, llegan al sistema las tareas τ_1, τ_2 y τ_3 que se encolan en la LRQ. Empieza la ejecución de la tarea τ_3 que es la que tiene el instante de promoción más próximo. En este instante no hay tareas en la URQ (cola de preparados de mayor prioridad) por lo tanto se puede reducir la velocidad del procesador.

En t=20, se producirá la primera promoción de la τ_3, y la siguiente promoción de alguna tarea será en t=40. La τ_3 tiene un WCET de 40 unidades de tiempo, no se puede reducir la velocidad, pero como la URQ ha estado libre durante 20 unidades de tiempo se puede reducir la velocidad en estas 20 unidades ya que la tarea τ_3 ha avanzado trabajo. Si la tarea τ_3 empezará su ejecución a partir del instante de promoción se ejecutaría durante 20 unidades de tiempo antes de ser expulsada por la tarea τ_1. La tarea τ_3 tiene 40 unidades de tiempo para ejecutar 20, puede ejecutarse al 50% de la velocidad máxima.

En t=40 la tarea τ_3 ha ejecutado el 50% del WCET al 50% de la velocidad máxima, esta finaliza y es en este momento en que llega la promoción de τ_1. Ahora esta, está sola en la URQ, se podría reducir la velocidad, pero como que el plazo temporal de τ_1 es 50 y el WCET de τ_1 es de 10, se tiene que ejecutar a velocidad máxima, terminando en t=45 debido a que consume el 50 % del WCET. El tiempo no consumido por esta tarea, será automáticamente utilizado por la tarea τ_2 que al ejecutarse sola en la URQ podrá ejecutarse reduciendo su velocidad. Se continuará de manera equivalente durante el resto del hiperperiodo.

Se debe remarcar que en este esquema (Figura 12) prácticamente solo hay tiempo libre de procesador al final del hiperperiodo, reduciendo de manera considerable el consumo energético que se produce en la ejecución de las tareas. Esto es debido a que en lugar de reducir la velocidad del procesador únicamente cuando existe una única tarea en el sistema, se reduce la velocidad cuando únicamente existe una o ninguna tarea en la URQ, pero pueden haber tareas activas en la LRQ que podrán aprovechar el tiempo no consumido por las tareas que han finalizado sin llegar a consumir su WCET. El LPFPS deja la reducción de velocidad para el final mientras que el algoritmo PLMDP se aplica la reducción de velocidad lo antes posible, haciendo que las tareas periódicas se ejecuten a partir del instante más próximo al plazo temporal, con lo que se puede aprovechar mejor el tiempo no consumido por las tareas recién finalizadas.

2.2.3 Consideraciones generales sobre la velocidad del procesador

En este punto se realiza una reflexión sobre las posibles velocidades de procesador válidas para ejecutar un determinado conjunto de tareas cumpliendo todos los plazos temporales. Recordemos que la formula que calcula la parte dinámica de la potencia consumida por un sistema con tecnología CMOS es $P \cong p_t \, C_L \, V_{DD}^2 \, f$, donde P es la potencia consumida, p_t es la probabilidad de cambio en una transición, C_L es la capacitancia, V_{DD} es el voltaje suministrado y finalmente f es la frecuencia de reloj del procesador. Si el procesador puede aplicar las técnicas del escalado del voltaje para escalar la frecuencia, entonces $P(f) \propto C_L \, V_{DD}^2 \, f \approx k \, f^3$ [75] [19]. Por tanto, la energía consumida para completar t unidades de tiempo a velocidad del procesador v será: $E \approx tv^3$.

El algoritmo de planificación del procesador, debe fijar la velocidad del procesador antes de empezar o reanudar la ejecución de toda tarea. El planificador tendrá varias opciones válidas de velocidad en las que el sistema es planificable, desde una reducción máxima, hasta la no reducción, pasando por reducciones intermedias. En este apartado, se van a considerar las diversas alternativas de que se dispone. Además, se debe considerar que los procesadores actuales disponen de un número finito de velocidades de trabajo [75], por lo que para garantizar la planificabilidad del sistema, el planificador deberá escoger una velocidad superior a la calculada.

En las tres subtablas de la Tabla 2 muestran la energía consumida por el conjunto de tareas de la Tabla 1, teniendo en cuenta distintos niveles de velocidad del procesador: 3 niveles en la primera subtabla, 10 niveles en la segunda y 100 niveles en la tercera. Suponiendo siempre que el consumo energético a velocidad 0 es despreciable, el consumo energético ejecutando las tareas durante un total de 40000 unidades de tiempo es de 29987,5 con tres niveles posibles de velocidad, de 29088,6 si se dispusiéra de una reducción de 10 niveles de velocidad posibles, y de 29515,9 con una reducción de 100 posibles niveles. En esta tabla se puede observar que el cálculo de la energía consumida va variando dependiendo de los niveles en que se divide la velocidad de proceso, sin embargo esta variación es de 0,02 por unidad de tiempo entre disponer de 3 niveles o 10 niveles y de 0,01 en el caso de disponer de 10 o 100 niveles.

V	tiempo	*E*
0	650	
0,5	10700	29987.5
1	28650	

a) 3 niveles de velocidad

V	tiempo	*E*
0	70	
0,4	3000	
0,5	6080	29088,6
0,8	3700	
0,9	3350	
1	23800	

b) 10 niveles de velocidad

V	tiempo	*E*
0	6	
0,34	3000	
0,5	6008	29515.9
0,75	3994	
0,99	1011	
1	25981	

c) 100 niveles de velocidad

Tabla 2: energía consumida según los niveles posibles de velocidad de ejecución del procesador. En la primera columna se muestra la velocidad escogida, en la segunda el tiempo ejecutado a la velocidad escogida, y por último en la tercera la energía consumida en un hiperperiodo.

En el artículo [75] de S. Saewong y R. Rajkumar se ha realizado un estudio en el que se llega a la conclusión, de que si el procesador dispone de un mínimo de 10 niveles de velocidad, tendrá como máximo un 10% de perdida de energía en el error cometido frente a disponer de un procesador ideal, en el que el planificador puede escoger entre un rango lineal de velocidades. Sin embargo, A. Miyoshi [52] demuestra que existen unas velocidades del procesador que son energéticamente ineficientes. En contra de la intuición general, si se ejecuta a menor velocidad en estos intervalos ineficientes se consume mayor cantidad de energía,.que ejecutando a una velocidad superior. Esto implica que existen unas velocidades del procesador que se deben evitarse a toda costa. En nuestro caso, se considerará que en el procesador no existen estos estados, y que la asignación de velocidad es discreta, en 10 o 100 intervalos, entre los que siempre hay la misma reducción de velocidad y energía entre dos puntos consecutivos de la división.

La implementación del planificador PLMDP supone que el procesador puede escalar la velocidad de manera discreta realizando particiones lineales de la velocidad, por ejemplo, en el caso de que se tuvieran 10 posibles velocidades de ejecución del procesador, los incrementos de velocidad serian de 0.1. El planificador escoge siempre la mínima velocidad posible que asegura el cumplimiento de todos los plazos temporales. Sin embargo, a continuación, se demuestra que el hecho de escoger siempre la mínima velocidad posible, no asegura siempre el máximo ahorro energético. Reduciendo la velocidad de ejecución de la tarea a una velocidad intermedia, la siguiente tarea quizás pueda reducir también la velocidad de ejecución.

Si la tarea reduce la velocidad de ejecución al máximo, obliga a las tareas restantes a ejecutarse a máxima velocidad, resultando con ello un mayor gasto energético.

Imaginemos, ahora, el caso límite en el que se tiene un conjunto de tareas formado por una única tarea con una utilización total del 60 %, WCET=60, T=D=100, esto implica que se dispone de un 40% de tiempo libre. Este tiempo libre, que queda después de ejecutar la tarea puede usarse para reducir la velocidad del procesador ahorrando de esta manera energía. Si se calcula la velocidad óptima de ejecución de esta tarea será de 60/100, es decir, si la tarea se ejecutase al 60% de la velocidad máxima se ocuparía todo el tiempo libre disponible, y la energía consumida por el sistema sería mínima y aplicando la formula $E = tv^3$ seria 100*0.216=21.6 en lugar de 60*1=60 que se tendría ejecutando a máxima velocidad, v=1. Si se ejecutara a menor velocidad, supongamos al 0,5 la energía necesaria sería 120*0.125=15, se consumiría menor energía, pero la tarea finalizaría en t=120, siendo su plazo temporal de 100, por tanto se perdería el plazo temporal. Se podría ejecutar a mayor velocidad, por ejemplo a 0,7 en este caso la tarea finalizaría en t=86, cumpliendo el plazo, pero ahora la energía consumida sería de 29.5, por tanto mayor que la energía ejecutando a 0.6. Imaginemos que se ejecuta la mitad de la tarea a 0.75% de velocidad usando 40 unidades de tiempo para realizar 30 de cálculo y el resto 30 unidades de cálculo en 60 unidades de tiempo resultando una velocidad de 0.5, i se finalizará, por tanto, en t=100 cumpliendo el plazo temporal, y se habrán consumido 0.422*40+0.125*60=24.38 unidades de energía, siendo este caso superior a la energía consumida a velocidad constante del 60%.

Recapitulando, se puede observar que, ejecutando a velocidades inferiores al 60%, se necesita más tiempo para ejecutar las mismas unidades de tiempo, por tanto, al final de la ejecución de la tarea si se quiere cumplir el plazo temporal, se necesita ejecutar a mayor velocidad, siendo el 100% la velocidad máxima posible. Ejecutando a velocidades superiores se finaliza con antelación al plazo temporal, pero se consume mayor energía. Esto implica que la velocidad mínima en este caso es el 60% que concuerda con la utilización que realiza la tarea del procesador.

Utilizando un planificador basado en EDF donde un conjunto de tareas es planificable si y solo si la utilización del procesador es menor o igual a 1. H. Aydin, R. Melhem, D. Mosse y P. Mejia-Alvarez proponen en [7] una reducción de la

velocidad en función de la utilización del procesador, de manera que la utilización final del procesador sea igual a 1.

Generalizando este caso a cualquier conjunto de tareas, se puede observar que la utilización del procesador condicionará la velocidad de ejecución mínima del conjunto de tareas. Sin embargo, con asignación estática de prioridades, se debe recordar que no es suficiente con tener una utilización del procesador menor que el 100% para garantizar la planificabilidad del sistema. Un parámetro que indica mejor el grado de planificabilidad de un conjunto de tareas es el "*Breakdown utilization*" (BU)[37], que se calcula de la siguiente forma: Cada $WCET_i$ del conjunto de tareas es multiplicado por un factor de escala, mientras que los periodos y plazos temporales no se modifican. Todo el conjunto se escala hasta el instante en que si hubiera un incremento adicional se perdería algún plazo temporal. El *Breakdown utilization* es la utilización del procesador en este punto.

$$BU=\Sigma_i ((\alpha\, WCET_i)/T_i) \qquad \text{(ec. 12)}$$

donde α es el factor de escala y el sumatorio es para todas las tareas del sistema. Se puede observar, que el factor d'escala es equivalente a la reducción de velocidad por este factor a todas las tareas del conjunto, por tanto, ejecutando todas las tareas con una reducción de α el conjunto es planificable y la energía utilizada es la mínima (Sin tener en consideración el *slack* dinámico generado por las tareas).

Para comprobar este punto se ha utilizado un algoritmo FPS (*Fixed Priority Scheduling*) con asignación de prioridades estáticas según "*Rate Monotonic"* (a menor periodo, mayor prioridad) en el que se ha ido reduciendo gradualmente la velocidad del procesador hasta que alguna tarea del conjunto de tareas pierda algún plazo temporal, representando en la gráfica la velocidad inmediatamente superior. En el experimento se han variado los siguientes parámetros de caracterización del conjunto de tareas: armonicidad de los periodos de las tareas, utilización máxima individual de una tarea del conjunto (Umax), utilización total del procesador (U) y por último el "*breakdown utilization"* (BU) [37] que es la medida que indica el grado de planificabilidad de un conjunto de tareas. Los resultados se hallan en la Figura 13 y en la Figura 14. Cada punto en las gráficas es la media aritmética del consumo energético normalizado de 100 conjuntos de tareas aleatorios. La recta se corresponde a los resultados exactos que se obtienen de dividir la utilización total del

procesador entre el BU (U/BU). En sistemas con un gran número de tareas con periodos no armónicos es muy difícil encontrar conjuntos de tareas con un BU del 100%, por lo que en la Figura 14 el simulador no fue capaz de hallar un número razonable de conjuntos que permitan realizar una media aritmética estadísticamente significativa.

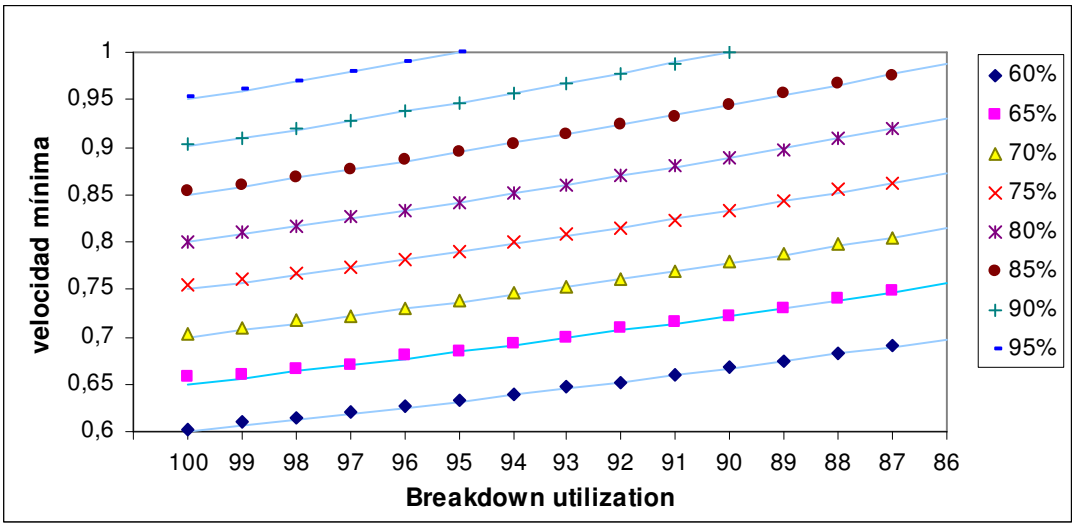

Figura 13: Velocidad mínima necesaria para planificar un conjunto de tareas cuando se varia la utilización del procesador y *el breakdown utilization* de los conjuntos de tareas. El Umax está fijado al 20%, los periodos son armónicos, y el conjunto de tareas está compuesto por 8 tareas independientes.

En la Figura 13, se puede observar que la reducción de velocidad factible es la resultante de aplicar la formula U/BU (líneas azules), si la reducción es menor que la máxima reducción posible, la energía consumida será mayor, mientras que si la reducción de velocidad es mayor, se consumiría menor energía, pero se podría perder el plazo temporal de alguna de las tareas del conjunto.

En las Figura 13 y Figura 14 se observa que la variación del número de tareas del conjunto no afecta significativamente al comportamiento de la reducción energética, aunque si afecta al hecho de encontrar conjuntos planificables con un BU determinado. Si los periodos no son armónicos, cuantas más tareas tenga el conjunto, menor será el BU obtenido.

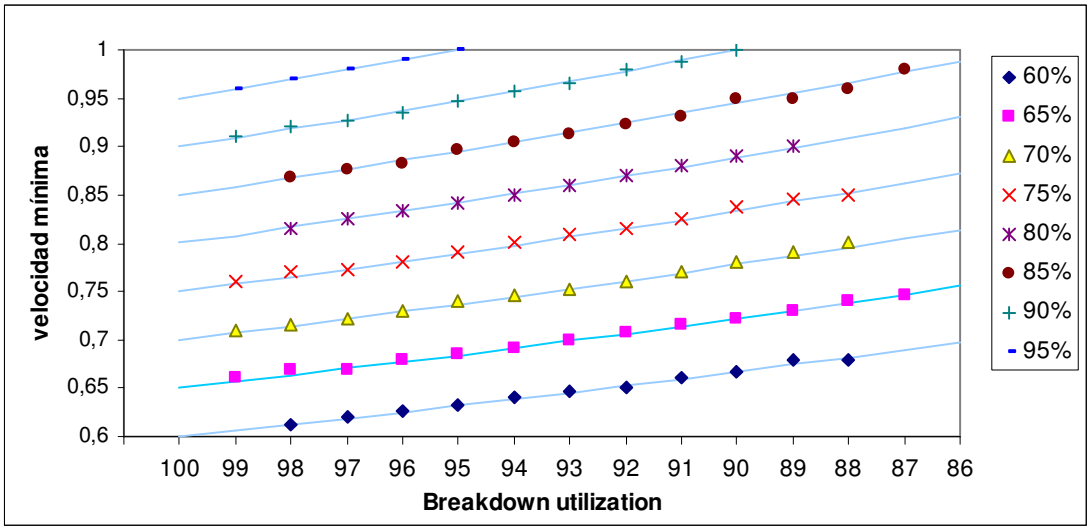

Figura 14: Velocidad mínima necesaria para planificar un conjunto de tareas cuando se varia la utilización del procesador y el *breakdown utilization* de los conjuntos de tareas. El Umax está fijado al 20%, los periodos no son armónicos, y el conjunto de tareas está compuesto por 16 tareas.

A pesar de los resultados mostrados, se debe tener en cuenta que este algoritmo no es apropiado para sistemas de tiempo real, puesto que esta reducción energética obtenida a base de reducir la velocidad de ejecución por igual a todas las tareas del conjunto no permite sacar provecho del hecho que las tareas finalicen sin llegar a consumir todo el WCET. De igual manera, este planificador no podrá tratar tareas aperiódicas puesto que como tareas menos prioritarias no se ejecutarían nunca, ya que la reducción de velocidad propuesta provoca que el procesador este ocupado durante todo el hiperperiodo planificable. Recordemos, que el FPS al tener prioridades estáticas no garantiza la planificabilidad al 100% de utilización del procesador.

2.2.4 Enhanced Power Low Dual Priority Scheduling

En este punto se propone una mejora al planificador PLMDP aplicando los resultados obtenidos en el apartado anterior. De manera que no haya una tarea que reduzca en exceso su velocidad de ejecución provocando que el resto de tareas deban ejecutarse a máxima velocidad. Inicialmente, se calculará el BU y la U del conjunto de tareas. Estos dos datos son estáticos (no se modifican durante la ejecución del sistema) y por lo tanto, pueden calcularse con anterioridad a la ejecución del conjunto de tareas. Cuando el sistema está en funcionamiento, la utilización real del

sistema, va variando en función del comportamiento de las tareas que se ejecutan. Inicialmente se debe suponer que las tareas consumirán el 100% de su WCET, y por tanto la mínima velocidad a la que se ejecutaran las tareas será la utilización calculada inicialmente del conjunto de tareas, dividido por su BU (U/BU). A medida que las tareas se vayan ejecutando, se pueden producir varios casos:

- La tarea se ha ejecutado a mayor velocidad que la calculada según la fórmula U/BU, con el fin de cumplir los plazos temporales.
- La tarea ha finalizado sin llegar a consumir todo su WCET.
- La tarea se ha ejecutado a menor velocidad que la calculada por U/BU con el fin de minimizar el tiempo libre de procesador.

En los dos primeros casos, la tarea finalizará sin llegar a utilizar todo el tiempo que tenia disponible para su ejecución, por tanto velocidad de ejecución calculada para el resto de las tareas será menor que la que seria si se hubiera ocupado todo el tiempo disponible, es decir, la ejecución de la tarea recién finalizada ha proporcionado tiempo extra para la ejecución del resto de tareas.

En el tercer caso, la tarea se ha ejecutado a menor velocidad que la previamente calculada, por tanto la tarea ha utilizado un tiempo mayor que el previsto inicialmente, sin embargo de no haberse utilizado, este tiempo se habría perdido debido a la ausencia de tareas listas para su ejecución. Esto implica que puede darse el caso que las siguientes tareas que se ejecutaran deberán hacerlo a una velocidad mayor.

Resumiendo, la mejora que se propone, con el algoritmo EPLDP, consiste en conseguir que el algoritmo de planificación **reparta el tiempo disponible** para reducir la velocidad del ejecución de las tareas entre las máximas tareas posibles, es decir, se intentará que las tareas no deban ejecutarse a nunca a velocidad máxima.

Antes de empezar la ejecución del sistema, de manera estática se calcularán:

- BU, que depende únicamente de las características físicas de las tareas (periodo y WCET).
- Up (La utilización pendiente de ser ejecutada) suponiendo que las tareas se ejecutarán a máxima velocidad y consumirán todo el WCET.

Los cambios que se deben realizar en el tratamiento de los eventos son los siguientes:

- Al inicio de un hiperperiodo, inicializar la variable W_{rem} con el trabajo que se deberá ejecutar en ese hiperperiodo.
- Cuando se produce una expulsión de la tarea (se activa una tarea de mayor prioridad) la variable W_{rem} debe reducirse según el trabajo recién realizado
- Cuando se produce la finalización la variable W_{rem} debe reducirse según el trabajo recién realizado más el tiempo que le ha sobrado.
- El planificador calcula la velocidad de ejecución de la tarea de manera individual antes de iniciar su ejecución, siendo esta igual al valor máximo entre la velocidad calculada por el PLMDP y el cociente entre la utilización remanente (U_{rem}) y el BU. La U_{rem} se calcula de la siguiente forma:

$$U_{rem} = \frac{W_{rem}}{hiperperiodo - tc} \qquad \text{(ec. 13)}$$

En la Figura 15 se presenta el pseudo-código del algoritmo EPLDP. Nótese, que la energía obtenida será la mínima posible cuanto más cerca se este de conseguir la velocidad media deseada, mientras que el algoritmo se alejará de esta optimalidad cuando más se alejen las tareas de la media deseada.

El hecho de que las tareas no consuman todo el WCET propuesto, también provocará que la energía consumida se aleje del mínimo posible, debido a que el cálculo de la utilización pendiente (Up) se alejará de la real. Pero, el planificador en ningún caso puede suponer una ejecución de las tareas por debajo del WCET, ya que de otro modo podría comprometer los plazos.

```
L1   si no vacia(URQ) entonces
L2        Active Task = URQ.head;
L3        si URQ.head.next = NIL y empty(ARQ) entonces
L4             velocidad = max( min(Tp_k − tc, remaining_i) / min(Tp_k, Td_i) − tc , U_rem / BU )
L5        sino
L6             velocidad = MAX_FREQ;
L7        finsi
L8   sino
L9        si no vacia(ARQ) entonces
L10            Active task = ARQ.head; velocidad = MAX_FREQ;
L11       sino
L12            si no empty(LRQ) entonces
L13                 Active task = LRQ.head;
L14                 si Tp_k < Tp_i entonces
L15                      velocidad = ( U_rem / BU )
L16                 sino
L17                      velocidad = max( min(Tp_k − Tp_i, remaining_i) / min(Tp_k, Td_i) − tc , U_rem / BU )
L18                 finsi
L19            sino
L20               progr. temporizador a (next Tai - wake up delay);
L21               Ponerse en modo power-down;
L22            finsi
L23       finsi
L24  finsi
L25  Ejecutar la tarea activa a la velocidad calculada;
```

Figura 15: Pseudocódigo del Enhanced Power Low Dual-Priority (EPLDP) Scheduling.

A continuación se establecerá en que casos funciona mejor el PLMDP que el EPLDP. El algoritmo EPLDP funciona siempre bien cuando se consume el 100% del WCET debido a que en este caso la velocidad estimada es la exacta. Pero conforme el consumo real del WCET se aleja del 100% la velocidad calculada también difiere más, llegando a la máxima divergencia cuando el consumo real del WCET es mínimo. En cambio, el algoritmo PLMDP, reduce siempre que puede al máximo su velocidad, de esta manera se adapta perfectamente al hecho que el porcentaje de WCET usado sea mínimo.

Las gráficas (Figura 16 a Figura 18) muestran el porcentaje de WCET mínimo que deben consumir las tareas para que el planificador EPLDP mejore el ahorro energético obtenido por el planificador PLMDP. Se han realizado distintos experimentos variando:

- Número de tareas: de 8 y 16 tareas.
- Armonicidad de los periodos: armónicas y no armónicas
- Utilización máxima de las tareas del conjunto (Umax): entre 10 y 40%
- Utilizaciones máximas del procesador(U): entre el 60% y el 95%.

En las gráficas se ha representado la media de 100 conjuntos para cada porcentaje de utilización del procesador representando en el eje de las X, y el % de WCET consumido en el que cambia el planificador mejor en el eje de las Y.

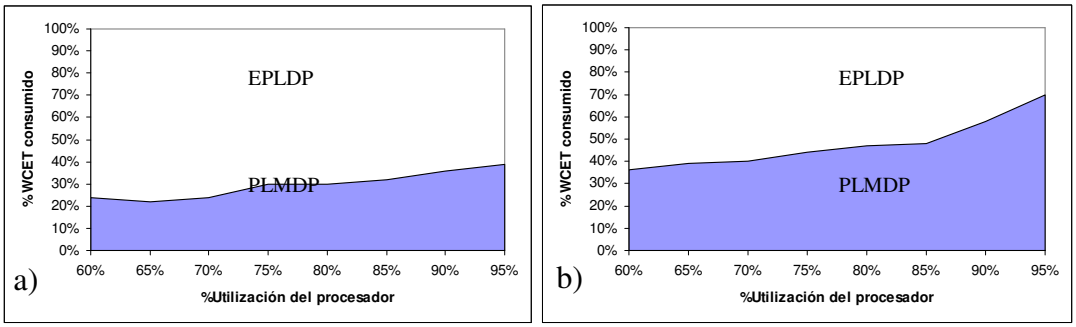

Figura 16: Representación de la curva de transición entre el algoritmo EPLDP y el PLMDP en conjuntos de 8 tareas con una carga máxima de las tareas del 20%. En a) las tareas tienen los periodos armónicos y en b) son no armónicos.

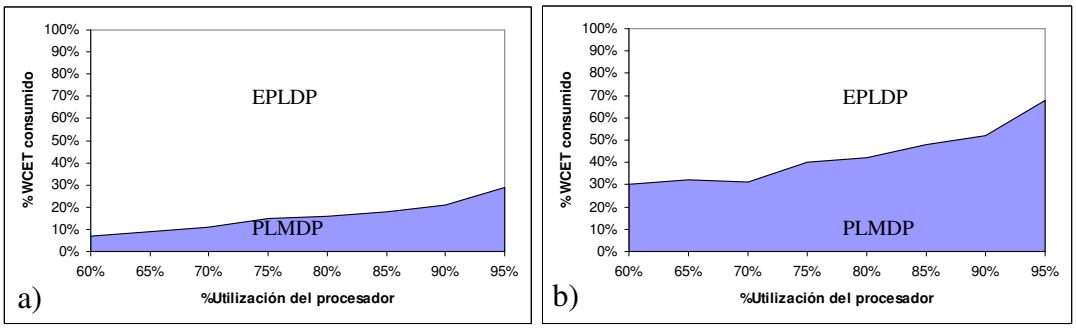

Figura 17: Representación de la curva de transición entre el algoritmo EPLDP y el PLMDP en conjuntos de 8 tareas con una carga máxima de las tareas del 40%. En a) las tareas tienen los periodos armónicos y en b) son no armónicos.

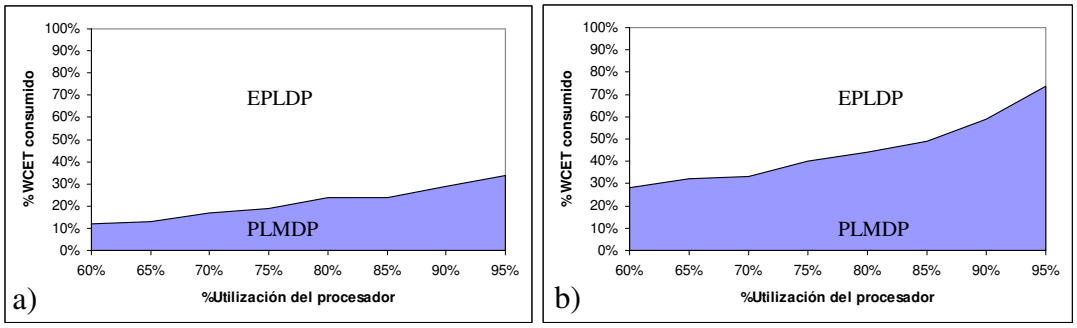

Figura 18: Representación de la curva de transición entre el algoritmo EPLDP y el PLMDP en conjuntos de 16 tareas con una carga máxima de las tareas del 20%. En a) las tareas tienen los periodos armónicos y en b) son no armónicos.

De todos los experimentos realizados se han escogido las anteriores por ser los más representativos. Si se comparan las gráficas con periodos armónicos se observa: que el planificador EPLDP tiene un mayor ahorro energético que el PLMDP con consumos menores de porcentajes de WCET, esto es debido principalmente a que con los periodos armónicos es más probable que coincidan diversas tareas en las distintas colas que con periodos no armónicos, y en este caso el PLMDP asigna el tiempo libre a una única tarea, ejecutando esa tarea a una velocidad muy reducida, pero la mayoría de tareas se ejecutan a velocidad máxima, mientras que el EPLDP reparte la velocidad de las tareas de manera más equitativa. En el caso de tareas no armónicas, el planificador PLMDP puede distribuir mejor, que con tareas armónicas, la reducción entre las distintas tareas dado que la activación de las tareas ocurre en distintos instantes, y así conseguir mejor ahorro energético con porcentajes consumidos de WCET mayores.

Si se considera ahora la carga máxima teórica del procesador, se observa un comportamiento creciente, lo que implica que con cargas de utilización alta, el algoritmo PLMDP se adapta mejor a los distintos porcentajes de WCET que con utilizaciones menores de procesador.

Si se dispone de mayor número de tareas para igual carga de procesador, se observa que la transición entre PLMDP y EPLDP de porcentaje de WCET consumido es menor que con menos tareas. Esto es debido a que dado que el rango de periodos es el mismo, la mayoría de tareas tendrán una carga menor, por tanto se adapta mejor el EPLDP ya que al existir mayor numero de tareas, la probabilidad de coincidencia en una cola es mayor.

Resumiendo, si las tareas de la aplicación tienen el cálculo del WCET bien estimado, asegurando que un consumo medio del WCET superior al 50% en utilizaciones del procesador por debajo del 85% con periodos no armónicos y por debajo del 95% en el caso de periodos armónicos, se puede asegurar que el ahorro energético conseguido con el EPLDP será siempre mejor que el que se consigue con el PLMDP.

2.2.5 Ejemplo de funcionamiento algoritmo EPLDP.

Al igual que para el algoritmo anterior, a continuación se muestra un ejemplo del comportamiento del planificador EPLDP (*Enhanced Power Low Dual Priority*) ejecutando el conjunto de tareas que se detalla en la Tabla 1. En la Figura 19 se muestra el comportamiento de las tareas a nivel de planificación y a nivel energético suponiendo que las tareas consumen siempre el 100% de su WCET. Y en la Figura 20 se muestra el comportamiento suponiendo que las tareas consumen únicamente el 50% de su WCET. Al igual que para los planificadores anteriores, para simplificar la problemática, se supone que no llegan tareas aperiódicas al sistema. Esta situación permite explicar claramente el funcionamiento básico del algoritmo. Como característica principal en la Figura 19, se puede observar que no existe tiempo de procesador libre, y que hay menores intervalos de ejecución de las tareas a máxima velocidad que con el algoritmo PLMDP. Otra característica que se puede observar es que la velocidad de ejecución de las tareas es más uniforme, es decir, no existen variaciones muy grandes en la velocidad de ejecución del procesador.

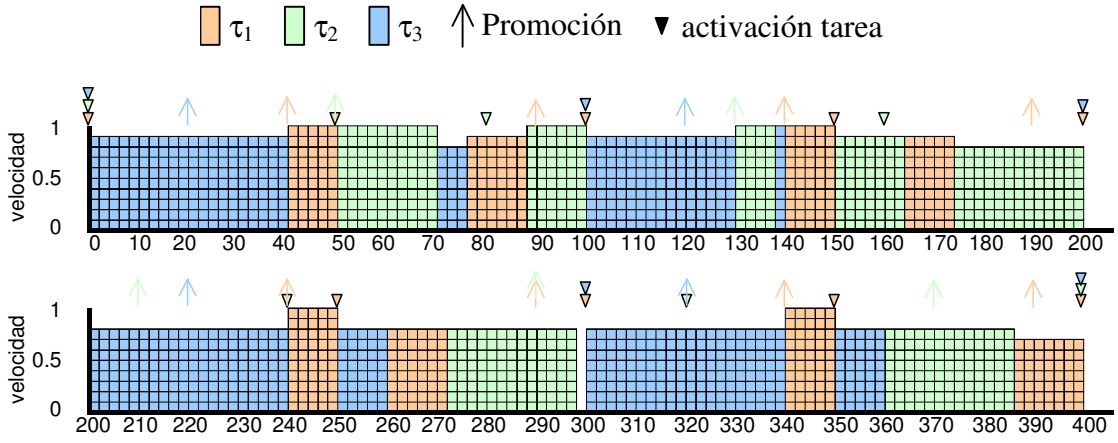

Figura 19: Diagrama de planificación de procesos usando el algoritmo EPLDP cuando se consume el 100% del WCET.

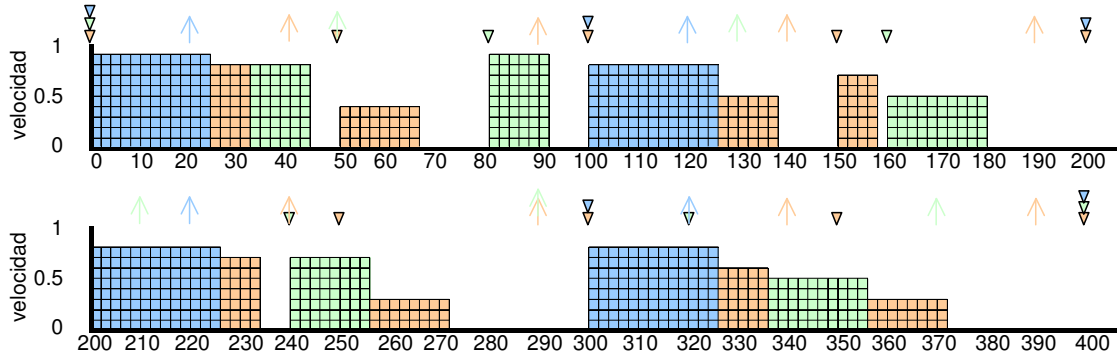

Figura 20: Diagrama de procesos del algoritmo de ahorro energético EPLDP, cuando todas las tareas consumen el 50% WCET.

En el caso de la Figura 20, se observa que aunque el algoritmo intenta ejecutar las tareas siguiendo la desviación de la ejecución real no lo consigue. Si se compara con el planificador PLMDP, el EPLDP es mejor a nivel energético siempre que se pueda estimar el promedio de carga del sistema de manera precisa. En este caso se hubiera podido determinar que las tareas consumirán aproximadamente el 50% del WCET, de manera que en el cálculo del límite de la velocidad ya se pudiera tener en cuenta. El resultado obtenido en este caso seria el que se muestra en la Figura 21, resultado que se halla más próximo al PLMDP. Se debe hacer constar, que si por alguna razón, las tareas ahora consumieran el 100% del WCET, se continuarían cumpliendo los plazos temporales, puesto que en este caso entraría en funcionamiento el planificador *dual priority* tal cual.

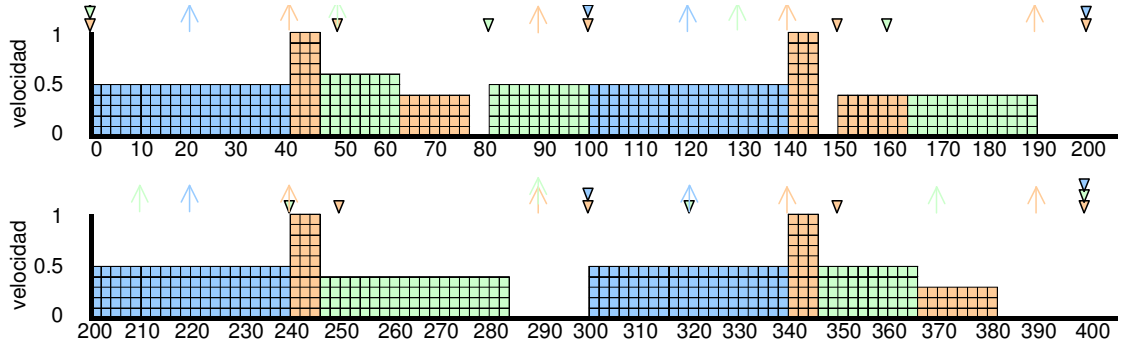

Figura 21: Diagrama de procesos del algoritmo de ahorro energético EPLDP, cuando las tareas consumen el 50% WCET.

2.2.6 Estimación dinámica de la utilización del procesador.

El WCET de una tarea depende no solo del flujo del programa sino de factores arquitectónicos como los *pipelines* y las caches. Este tiempo debe ser una cota superior de la utilización de la tarea, por lo tanto, normalmente la tarea no consumirá la totalidad de este tiempo. Para alcanzar la máxima eficiencia en el uso del procesador el sobredimensionado debe ser mínimo. Sin embargo, conforme la arquitectura se hace más compleja, la estimación se complica, siendo el sobredimensionado del WCET mayor.

Nosotros proponemos una estimación dinámica (EU) de este sobredimensionado utilizando la historia del sobredimensionado de las últimas ejecuciones ocurridas en los últimos hiperperiodos (U_m) donde *m* se refiere a los distintos hiperperiodos ejecutados. A continuación se presenta el algoritmo utilizado para determinar la reducción de velocidad que se adaptará mejor al cálculo real de las tareas (U_i) (Figura 22) Inicialmente se estima la utilización (EU) con los WCET propuestos por los diseñadores de la aplicación.

- Cada hiperperiodo será ejecutado utilizando la EU actual. Al finalizar cada tarea se debe contabilizar la utilización real de esta.
- Al final de cada hiperperiodo la EU es actualizada conforme la historia disponible usando una media móvil.

L1 $U_o = \sum_{\forall \tau_i} C_i \qquad EU = U_o$

L2 i=1

L3 mientras no termine la aplicación de tiempo real hacer

L4 ejecutar el hiperperiodo i con EU i actualizar U_i

L5 $U_m = \frac{(U_{m-1} * i) + U_i}{i+1} \qquad EU = U_i - U_m$

L6 finmientras

Figura 22**:** Estimación dinámica de la utilización empírica del procesador

En la Figura 23 se presenta la evolución del EPLDP usando la estimación dinámica de la sobrecarga sobre diferentes hiperperiodos (EPLDP-m) para un conjunto de tareas aleatorias cuya utilización real es exactamente el 50% del WCET. Se debe notar que esta estimación es una cota inferior para la reducción de velocidad, la cota superior viene proporcionada por el algoritmo de planificación. La energía

consumida que se obtiene con el EPLDP-m tiende a ser la misma que la energía consumida por el algoritmo teórico EPLDP-f, que es el comportamiento que tendría el EPLDP si se conociera exactamente el consumo real de WCET. Las pequeñas divergencias entre ambos algoritmos son consecuencia de las variaciones del uso real del WCET cuando este consume el 10% del WCET.

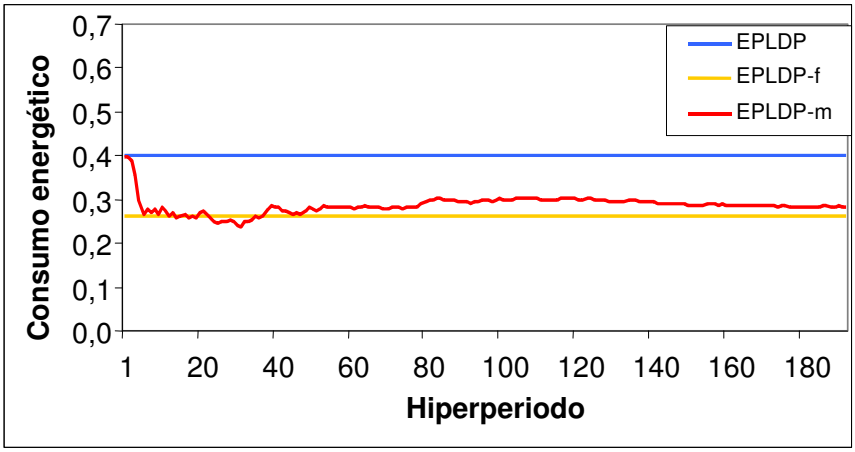

Figura 23: Evolución del EPLDP-m comparado con el EPLDP y el EPLDP-f. Los resultados mostrados son de 100 conjunto de tareas aleatorios con una utilización máxima del procesador del 80% y una utilización máxima de las tareas del 20%. El consumo real de las tareas se ha obtenido con una distribución Gaussiana con un consumo medio del 50% y una desviación del 10%.

La mejora en el consumo energético que se obtiene con EPLDP-m es contrastable, y se hace mas evidente cuanto mayor es la sobreestimación del WCET. En la Figura 24 se muestra la mejoría en función del porcentaje de WCET realmente consumido.

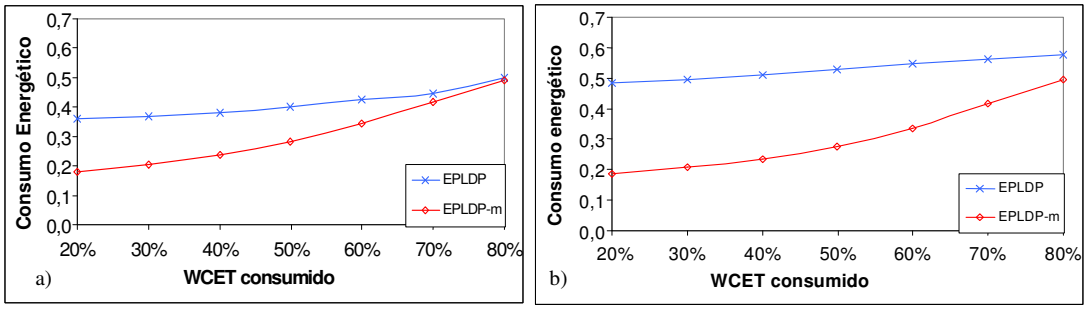

Figura 24: Comportamiento del EPLDP y del EPLDP-m variando el porcentaje de WCET consumido. Los resultados son los obtenidos por 100 conjuntos de tareas periódicos con una utilización del procesador del 80% y una utilización máxima de las tareas del 20%. En a) los periodos de las tareas son armónicos, y en b) los periodos son no armónicos.

2.3 Comparativa entre los algoritmos descritos

En este punto se evaluaran los distintos algoritmos presentados confrontándolos entre ellos, para observar las distintas características energéticas de cada uno, verificando que realmente se obtiene un ahorro energético significativo. En las gráficas de resultado, se incluye un planificador con ahorro energético (LPEDF) basado en una asignación de prioridades dinámicas (tipo EDF) y cálculo de la velocidad del procesador dinámica (Ver anexo 3) dado que el algoritmo del Dual Priority que se ha escogido como base para posteriores modificaciones posee una asignación de prioridades estáticas, pero con un comportamiento dinámico, al disponer de dos niveles de prioridades para las tareas periódicas, que le permite obtener mejores resultados que los planificadores totalmente estáticos.

Cada experimento constará de un conjunto de tareas independientes. Cada tarea está caracterizada por un periodo (T), un plazo temporal (D) y un peor tiempo de cálculo (WCET) y una prioridad relativa a las otras tareas del conjunto (P). En los experimentos que se han realizado se ha variado el porcentaje de WCET consumido por las distintas tareas, para comprobar si el algoritmo de planificación se adapta a esta característica dinámica en el comportamiento real de las tareas. La variación definida va desde el 10% hasta el 100% de su WCET. La duración de cada experimento será de un hiperperiodo (mínimo común múltiplo de los periodos de las tareas que forman un conjunto de pruebas) en el caso que el comportamiento del consumo del WCET sea siempre el mismo para todas las activaciones de las tareas, y sobre 100 hiperperiodos en el caso de que la variación del porcentaje de WCET consumido sea variable. En este último caso, el porcentaje de WCET representado en la gráfica corresponde al valor promedio.

Todos los resultados se muestran normalizados respecto a uno de los algoritmos de planificación, en particular el que se ejecute a la máxima velocidad del procesador.

Los experimentos que se van a realizar pueden dividirse en dos grandes bloques:

- Conjuntos individuales de tareas: Benchmark propuesto por Shin y Choi en [77] las características de las tareas se hallan en la Tabla 1 y cuyo resultado de planificación de las tareas se ha presentado de la Figura 9 a la Figura 12. Después se verán varios conjuntos de tareas obtenidos de tres aplicaciones

reales de tiempo real: un caso de aviónica [50], un caso de navegación [15] y por último un caso de control automático [37].

- Conjuntos de tareas sintéticos para poder analizar como afectan las variaciones de los distintos parámetros que caracterizan las tareas en el conjunto de la planificación. Para cada posible variación se han generado 100 conjuntos de tareas planificables. Las variaciones estudiadas son las que siguen:
  - Variación de la carga periódica total del sistema.
  - Variación del ratio entre el período mínimo (Tmin) y el máximo (Tmax)
  - Variación de la utilización máxima individual de las tareas
  - Variación en el número de tareas que componen el conjunto.

Para cada uno de estos experimentos, se ha realizado una gráfica mostrando el consumo energético normalizado, que se consigue con los distintos algoritmos de planificación según se va variando el porcentaje de WCET consumido por las tareas, y otra gráfica en el que se muestra la diferencia en consumo energético de cada punto referido al consumo energético cuando el procesador ejecuta siempre a la máxima velocidad, variando la carga periódica total del sistema. Al final de cada experimento se expondrán las conclusiones obtenidas individualmente.

2.3.1 Conjunto individual de tareas simple.

Inicialmente se estudiará el conjunto de tareas propuestas en la Tabla 1. Se mostrará su comportamiento según la variación del porcentaje de WCET consumido por las distintas activaciones de las tareas y su repercusión en el ahorro energético obtenido. Este conjunto tiene definidas 3 tareas con una utilización del procesador del 85 %. La variación entre el periodo mínimo y el máximo es de 0.5, y la utilización individual máxima (Umax) de las tareas es del 40%, y la utilización individual mínima (Umin) es del 20%.

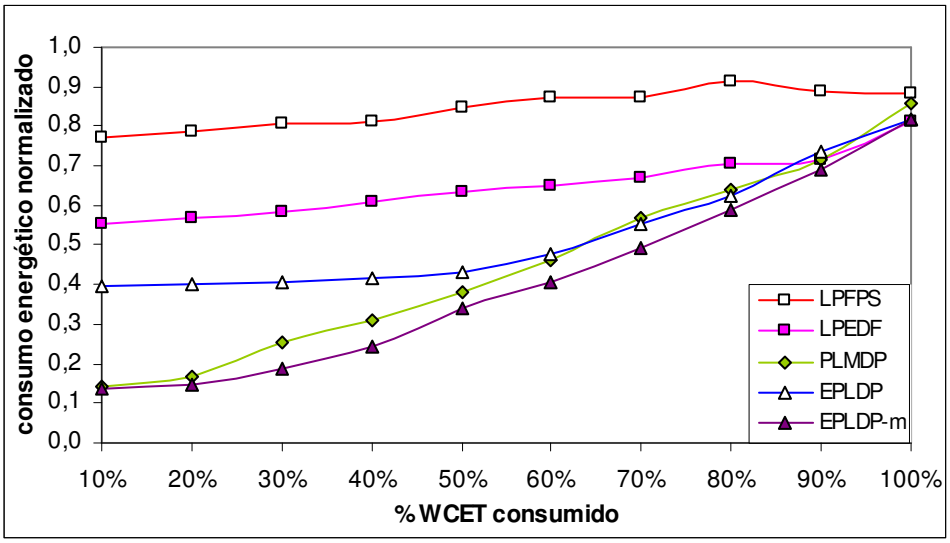

Figura 25: Comparación del consumo energético entre los algoritmos analizados usando el conjunto de tareas propuesto en Tabla 1 según el porcentaje de WCET consumido.

En la gráfica de la Figura 25 se muestran los resultados de la energía consumida normalizada respecto a la ejecución a máxima velocidad, variando en incrementos de 10% desde el 10% hasta el 100% el porcentaje de WCET consumido por todas las tareas del sistema. En este ejemplo, el promedio de consumo energético normalizado (área de cada gráfica) según el consumo con una ejecución de las tareas a máxima velocidad para cada uno de los algoritmos de planificación es: del 85% para el LPFPS, del 65% para el LPEDF, del 45% para el PLMDP y por último del 53% para el EPLDP, siendo 41% para el EPLDP-m.

Caso de aviónica

Las características del conjunto de tareas para el caso de aviónica [50] se muestran en la Tabla 3. Este conjunto tiene definidas 17 tareas con una utilización del procesador muy alta, del 90 %. La variación entre el periodo mínimo y el máximo es de 1000, y la carga individual máxima de las tareas es del 20%, y la carga individual mínima es de 10%. La característica principal de este conjunto es que hay varias tareas con el mismo periodo, y utilización individual muy pequeña. Otra característica a tener en cuenta es que la tarea que fija la carga mínima individual del conjunto de tareas es la que posee un periodo y un plazo temporal mayor.

Tarea	T	D	WCET
T1	100	100	5,1
T2	20000	20000	300
T3	2500	2500	200
T4	2500	2500	500
T5	4000	4000	100
T6	5000	5000	300
T7	5000	5000	500
T8	5900	5900	800
T9	8000	8000	900
T10	8000	8000	200
T11	10000	10000	500
T12	20000	20000	300
T13	20000	200000	100
T14	20000	20000	100
T15	20000	20000	300
T16	100000	100000	100
T17	100000	100000	100

Tabla 3:Conjunto de tareas de aviónica [50]

En la Figura 26 se muestra el consumo energético normalizado de los diferentes algoritmos. Observemos que, debido a que el sistema tiene una utilización muy alta, la posibilidad de reducir la velocidad del procesador es pequeña, con lo que apenas hay ahorro energético cuando el consumo de las tareas es el 100% del WCET. Conforme el porcentaje de WCET consumido disminuye, podría mejorar el ahorro, sin embargo, dado que los algoritmos de planificación LPFPS y LPEDF se basan en reducir la velocidad de la última tarea activa y ésta representa un porcentaje muy pequeño en la utilización total, el ahorro que se obtiene es muy pequeño. En el caso del LPEDF, lo que no consume una tarea lo aporta directamente a la reducción de velocidad y por tanto energética de la siguiente, con lo que independientemente de la utilización individual de cada tarea se consiguen resultados de ahorro energético muy buenos. Con el planificador PLMDP, ocurre algo similar a los dos primeros, solo se reducirá la velocidad del procesador cuando en la cola de mayor prioridad haya un sola tarea, si dos tareas tienen el mismo comportamiento de promociones y además la tarea de mayor periodo / plazo temporal es la que aporta la mínima utilización individual.

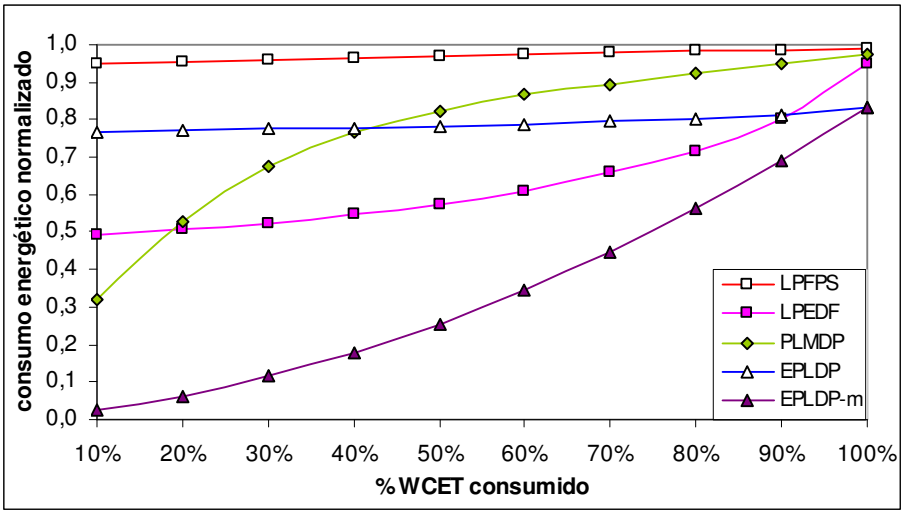

Figura 26: Consumo energético normalizado para el conjunto de tareas de aviónica [50] según el porcentaje de WCET consumido por las tareas.

En el ejemplo de aviónica, el promedio de consumo energético normalizado según el consumo con una ejecución de las tareas a máxima velocidad para cada uno de los algoritmos de planificación es: del 97% para el LPFPS, del 64% para el LPEDF, del 77% para el PLMDP y por último del 79% para el EPLDP, siendo 35% para el EPLDP-m.

Caso de navegación

Las características del conjunto de tareas para el caso de navegación se muestran a continuación en la Tabla 4. En este el conjunto de tareas tiene definidas 6 tareas con una utilización del procesador del 74 %. La variación entre el periodo mínimo y el máximo es de 2000, y la carga individual máxima de las tareas es de 47%, y la carga individual mínima es de 16%. La característica principal de este conjunto es que hay dos tareas (T4,T5) con el mismo periodo, y utilización individual muy pequeña. Otra característica importante es que la tarea que fija la carga máxima individual del conjunto de tareas es la que posee un periodo y un plazo temporal menor.

Tarea	T	D	WCET
T1	250	250	118
T2	4000	4000	428
T3	62500	62500	1028
T4	100000	100000	2028
T5	100000	100000	10028
T6	125000	125000	2500

Tabla 4: Conjunto de tareas de navegación (INS) [15]

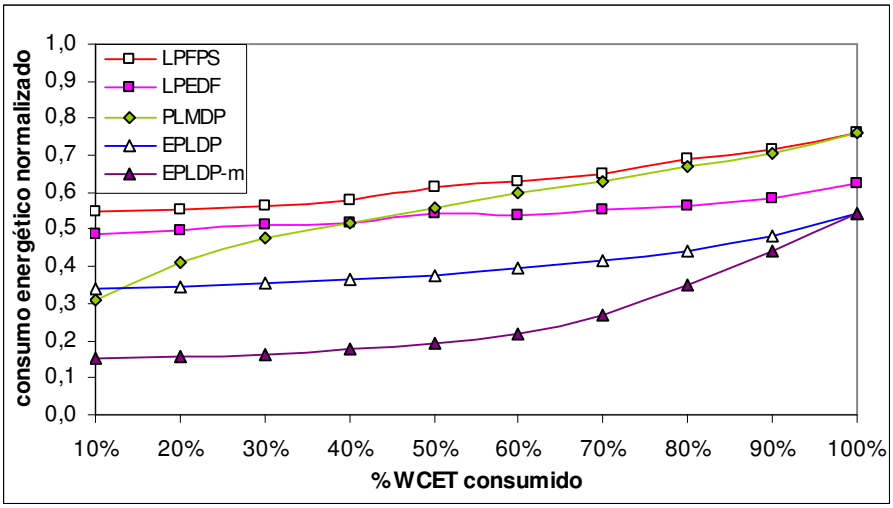

Figura 27: Consumo energético normalizado para el conjunto de tareas de navegación (INS) [15] según el porcentaje de WCET consumido por las tareas.

En el ejemplo de navegación, el promedio de consumo energético normalizado según el consumo con una ejecución de las tareas a máxima velocidad para cada uno de los algoritmos de planificación es: del 63% para el LPFPS, del 54% para el LPEDF, del 56% para el PLMDP y por último del 41% para el EPLDP, siendo 27% para el EPLDP-m.

Caso de control automático (CNC).

Las características del conjunto de tareas para el caso de control automático [38] se muestran a continuación en la Tabla 5. En este el conjunto de tareas tiene definidas 8 tareas con una utilización del procesador muy baja, del 48 %. La variación entre el periodo mínimo y el máximo es de 0.25, y la carga individual máxima de las tareas es de 15%, y la carga individual mínima es de 1,5%. La característica principal de este conjunto es que hay una tarea con el mínimo periodo y el plazo temporal mayor, siendo esta la tarea que aporta la utilización individual mínima.

Tarea	T	D	WCET
T1	2400	2400	35
T2	2400	24000	40
T3	4800	4800	180
T4	4800	4800	720
T5	2400	2400	165
T6	2400	2400	165
T7	9600	9600	570
T8	7800	7800	570

Tabla 5: Conjunto de tareas de control automático (CNC) [38]

En la Figura 28 se ve que en este caso, los planificadores LPFPS y el LPEDF aportan un bajo rendimiento debido a la tarea τ_2 que está la mayor parte del tiempo activa, no permitiendo la reducción de velocidad a las demás tareas. En el caso del PLMDP, esta tarea no molesta tanto debido a que se halla en la cola de menor prioridad, y cuando promociona finaliza rápidamente debido a que su carga individual es la mínima posible. En el caso del planificador LPEDF se obtiene ahorro energético debido a que al tener una utilización baja, se consigue la reducción de la velocidad de ejecución de las tareas.

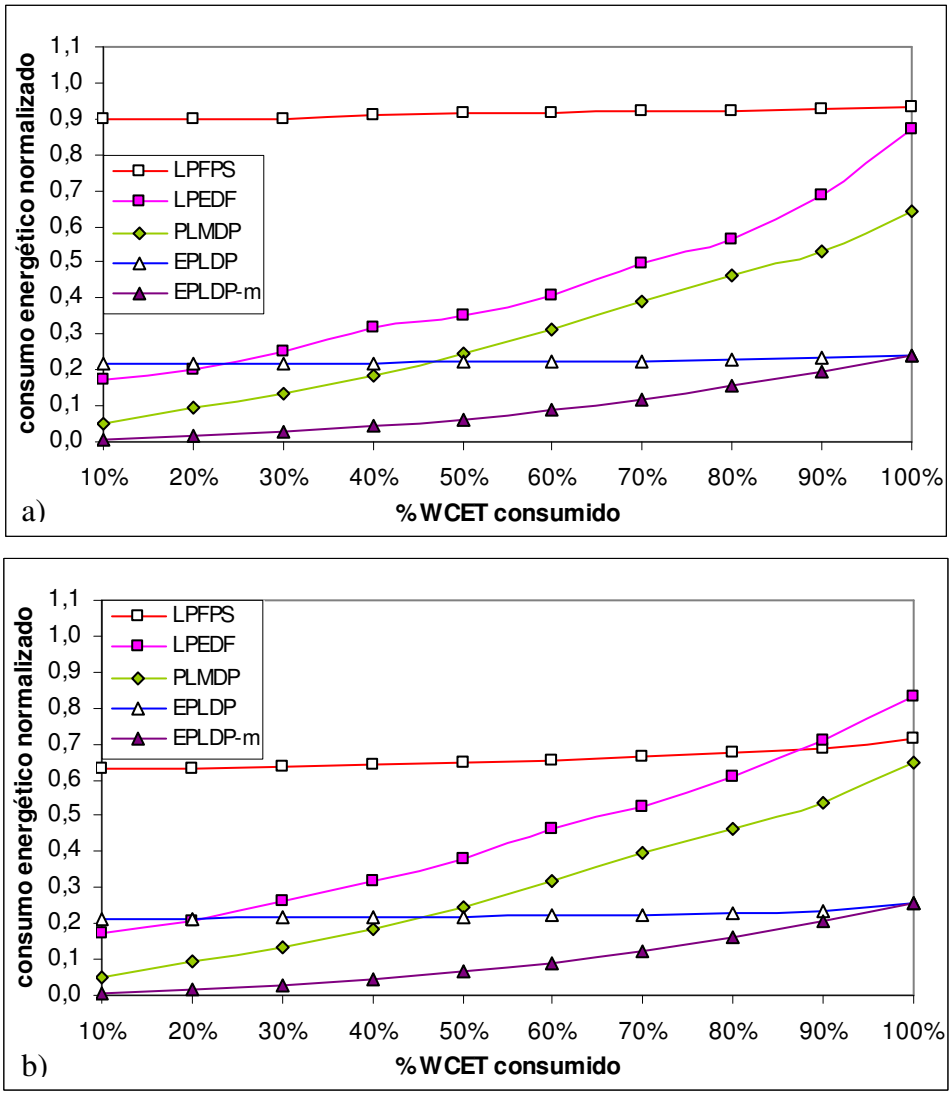

Figura 28: En a) Consumo energético normalizado para el conjunto de tareas de control automático [38] según el porcentaje de WCET consumido por las tareas. En b) todas las tareas tienen el plazo igual al periodo.

En este caso el LPFPS tiene un promedio de consumo del 91% si los periodos son distintos a los plazos y del 66% en caso de igualdad. Para el resto de planificadores no existe una diferenciación de promedios importante, siendo de 43% y 45% para el planificador LPEDF, de 30% y 31% para el PLMDP, de 22% en los dos casos para el EPLDP y finalmente de 9% y 10% para el EPLDP-m.

Comentarios generales de los casos analizados anteriormente.

Los resultados del consumo energético normalizado para cada una de las aplicaciones está mostrado desde la Figura 25 a la Figura 28. El resumen de las principales características de las aplicaciones se hayan en la Tabla 6, y la media normalizada consumida por los distintos planificadores se halla en la Tabla 7.

	armonicidad	N tareas	U	Umax	Umin	Tmin/Tmax
Ejemplo	poco	3	85%	40%	20%	0.5
Aviónica	Tareas con T iguales	17	90%	20%	10%	0.001
INS	Tareas con T múltiples	6	74%	47%	16%	0.002
CNC	totalmente	8	48%	15%	1,5%	0.25

Tabla 6: Características principales de las aplicaciones individuales.

	Ejemplo Tabla 1	Aviónica	INS	CNC	CNC con D=T
LPFPS	0,85	0,97	0,63	0,91	0,66
LPEDF	0,65	0,64	0,54	0,43	0,45
PLMDP	0,45	0,77	0,56	0,30	0,31
EPLDP	0,53	0,79	0,41	0,22	0,22
EPLDP-m	0,41	0,35	0,27	0,09	0,10

Tabla 7: Resumen de la media de los conjuntos de pruebas individuales normalizado de las tareas consumiendo entre el 10% y el 100% del WCET con los distintos algoritmos estudiados.

- LPFPS: es el planificador con un rendimiento energético más desfavorable. Siendo este rendimiento peor en el caso de conjuntos de tareas con periodos armónicos debido a que hay varias tareas que se activan simultáneamente, y solo se podrá disminuir la velocidad de ejecución de la última tarea de la cola de ejecución. El planificador es muy sensible a la asignación de prioridades, de manera que si una tarea con poca prioridad se halla en el sistema, provoca

que el resto de tareas no puedan disminuir la velocidad de ejecución, utilizando al final mayor porcentaje de energía.

- LPEDF: obtiene mejor rendimiento cuanto más armónicos son los periodos de las tareas, debido a la existencia de un servidor de disminución de la velocidad cuya capacidad se recarga con el periodo de la tarea más frecuente. Si este periodo es muy pequeño puede ser que no se utilice totalmente el tiempo, mientras que si es muy grande, las tareas se ejecutan primero muy despacio, para ejecutarse después a máxima velocidad.
- PLMDP: el rendimiento del algoritmo va en función del tamaño de las tareas, si estas son pequeñas se obtiene un mejor rendimiento que si las tareas son grandes. Además, si el procesador tiene una utilización baja, la disminución de la velocidad de proceso se realiza al principio del hiperperiodo, y esto permite que si la tarea finaliza sin consumir la totalidad del WCET este tiempo pueda ser aprovechado por las siguientes tareas.
- EPLDP: Al estar basado en el algoritmo anterior, obtiene mejor rendimiento cuando el PLMDP lo obtiene, sin embargo, si el procesador tiene una utilización baja, la disminución de la velocidad de proceso puede ser más uniforme, quedando esta más repartida entre todas las distintas tareas mejorando así el resultado obtenido por el algoritmo.
- EPLDP-m: Consigue mejores resultados que el anterior, debido a que se basa en la ejecución histórica para calcular la tasa de sobredimensionado del WCET. Provocando el reparto del tiempo no consumido entre todas las tareas del sistema, consiguiendo que estas no se ejecuten a máxima velocidad. Este algoritmo consigue aprovechar prácticamente la totalidad de tiempo libre del procesador.

2.3.2 Conjuntos de tareas sintéticos.

Para los siguientes experimentos se han generado 100 conjuntos de tareas sintéticas para poder observar la influencia de las distintas caracterizaciones de las tareas en el ahorro energético según el algoritmo empleado. La generación y ejecución de las tareas se ha realizado como sigue:

- Durante la simulación de la ejecución se ha variado del 10% al 100% el porcentaje de WCET consumido por las tareas, en incrementos del 10%.
- Se simula durante un hiperperiodo y los valores de energía obtenidos los normalizamos conjunto a conjunto, debido a que los hiperperiodos obtenidos por el conjunto de tareas pueden ser distintos (depende de los periodos de las tareas que forman parte del conjunto) y por tanto la energía consumida por los distintos conjuntos no seria comparable, aun usando el mismo planificador. En las gráficas se muestra la media aritmética normalizada del consumo de los cien conjuntos generados aleatoriamente con características similares.

Variación del número de tareas y armonicidad de los periodos.

Se han realizado experimentos variando la armonicidad de los periodos y el número de tareas del conjunto. Los periodos varían entre 1000 y 70000 unidades de tiempo, para los conjuntos de tareas con periodos no armónicos y entre 1024 y 65536 para los conjuntos de tareas con periodos armónicos. Las figuras siguientes muestran el consumo energético normalizado cuando la utilización del procesador es del 75%, la carga máxima individual de las tareas del 20% y se ha variado el número de tareas entre 8 y 16 tareas. Únicamente se han tenido en cuenta conjuntos planificables de tareas, es decir, los conjuntos que cumplen que $\forall\tau_i$ $WCRT_i \leq D_i$. La Figura 29 para muestra conjuntos de tareas con periodos armónicos y la Figura 30 muestra conjuntos de tareas con periodos no armónicos.

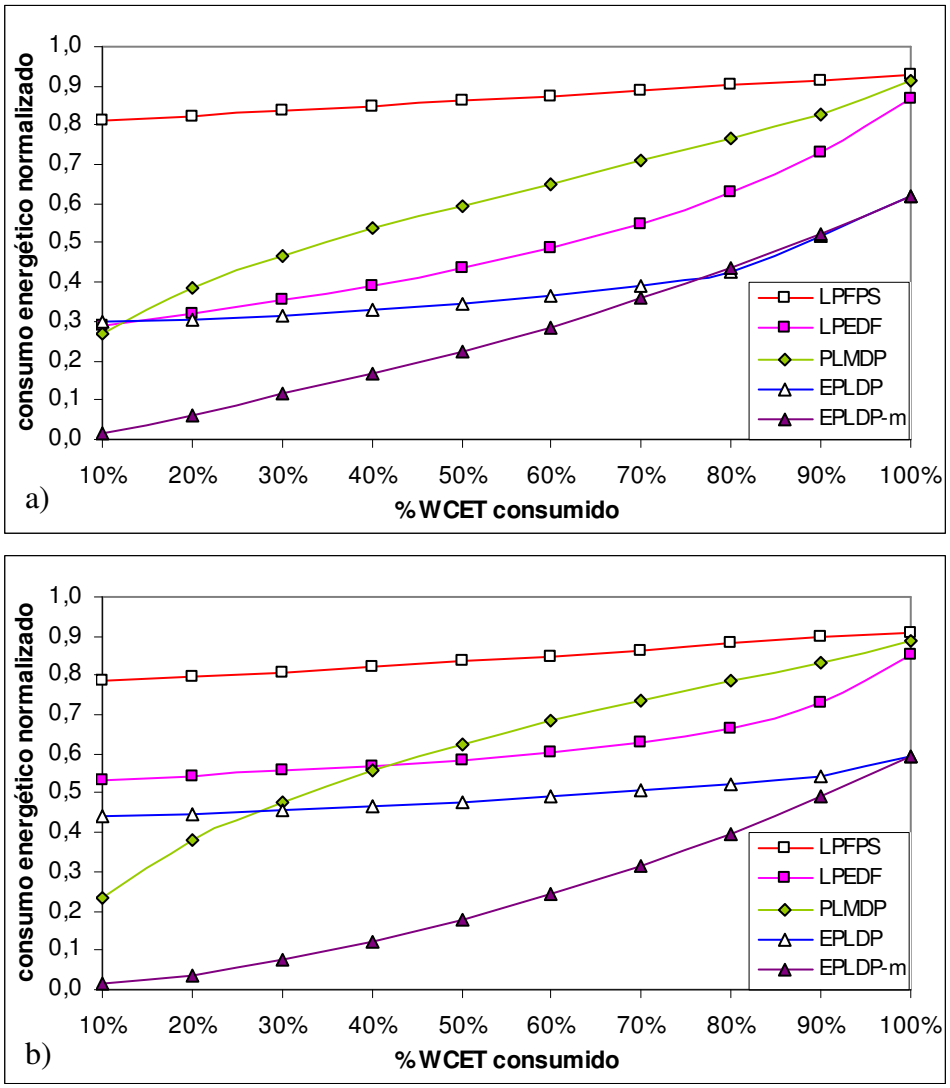

Figura 29: 100 Conjuntos de 8 tareas con una carga máxima de las tareas del 20% y una utilización máxima del procesador del 75%. En a) los periodos de las tareas son armónicos y en b) los periodos son no armónicos.

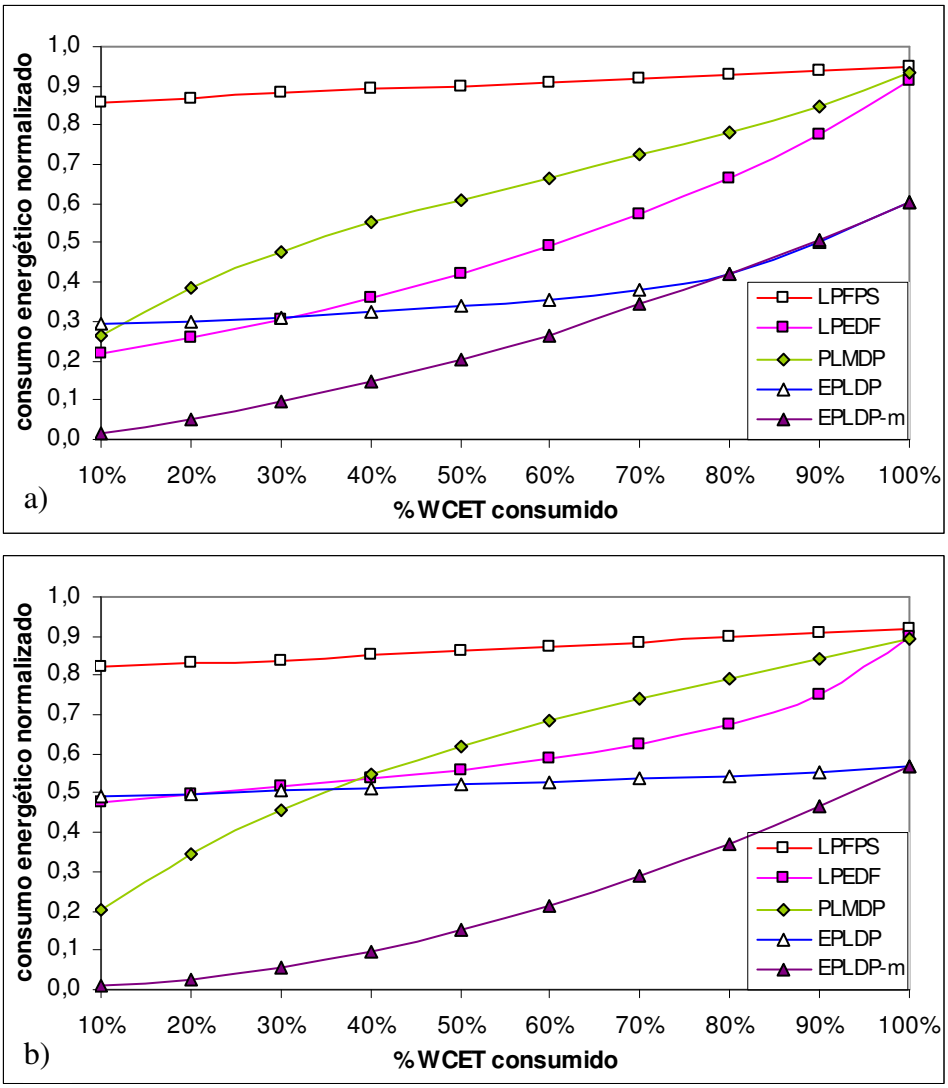

Figura 30: 100 Conjuntos de 16 tareas con una carga máxima de las tareas del 20% y una utilización máxima del procesador del 75%. En a) los periodos de las tareas son armónicos y en b) los periodos son no armónicos.

En las cuatro figuras anteriores se puede observar que el algoritmo EPLDP-m es el que funciona mejor. La variación en el número de tareas no presenta diferencias estadísticamente significativas. (Tabla 8)

	LPFPS	LPEDF	PLMDP	EPLDP	EPLDP-m
Conjunto armónico 8 tareas	87%	51%	61%	39%	28%
Conjunto no armónico 8 tareas	85%	63%	62%	49%	25%
Conjunto armónico 16 tareas	90%	50%	62%	38%	27%
Conjunto no armónico 16 tareas	87%	61%	61%	53%	23%

Tabla 8: Consumo medio normalizado de los planificadores de referencia según la armonicidad de los periodos y el número de tareas. La utilización del procesador máxima elegida es del 75%, y la carga máxima de las tareas del conjunto es del 20%

De las figuras anteriores también se puede observar que el EPLDP y el LPFPS, son los planificadores que poseen un consumo energético normalizado más estable frente al consumo del WCET, es decir, el consumo energético obtenido no depende significativamente del porcentaje de WCET consumido. Este hecho es debido a que la velocidad de ejecución de las tareas que se asigna intenta ocupar todo el tiempo libre disponible con los datos disponibles inicialmente, pero si las tareas no consumen todo el WCET, este no puede ser aprovechado por ninguna tarea dejando libre el procesador. El planificador ejecuta las tareas a una velocidad superior a la necesaria, creando, con ello, espacios de procesador libres, y consumiendo mayor cantidad de energía que la que usan otros planificadores que se adaptan más fácilmente a los cambios de consumo del WCET.

Si se utiliza una media móvil para calcular el sobredimensionado del WCET, conforme evoluciona el sistema se llega a disponer de un sobredimensionado muy aproximado al real, con lo que la velocidad de ejecución de las tareas es la más adecuada para ocupar todo el espacio de procesador disponible. El algoritmo EPLDP está preparado para que en caso de un consumo superior no haya perdidas de plazos, interviniendo el planificador DP de base. Pero, en caso de un consumo inferior, la media móvil va a continuar posicionándose en la velocidad necesaria.

Variación de la carga del sistema entre el 60% y el 95%.

La máxima utilización de las tareas del conjunto está fijada al 20%, los periodos varían entre 1000 y 70000 unidades de tiempo, para los conjuntos de tareas con periodos no armónicos y entre 1024 y 65536 para los conjuntos de tareas con

periodos armónicos. Se ha fijado el número de tareas en 8. Los resultados se hayan de la Figura 31a la Figura 33.

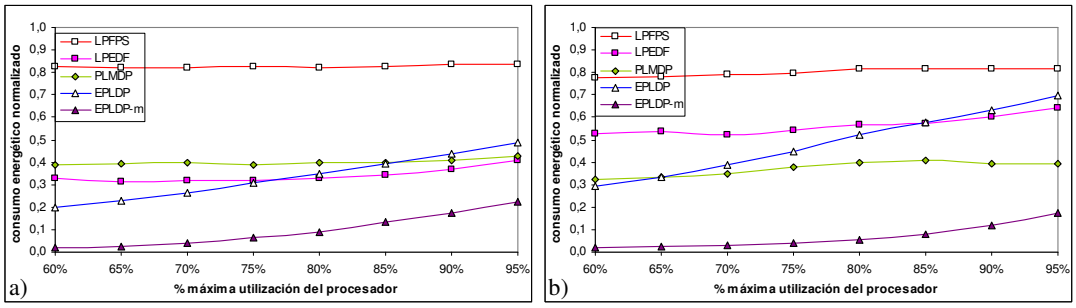

Figura 31: Consumo energético normalizado para distintas utilizaciones de procesador con un 20% de consumo de WCET en a) los periodos de las tareas son armónicos y en b) los periodos no son armónicos. La carga máxima por tarea es del 20%.

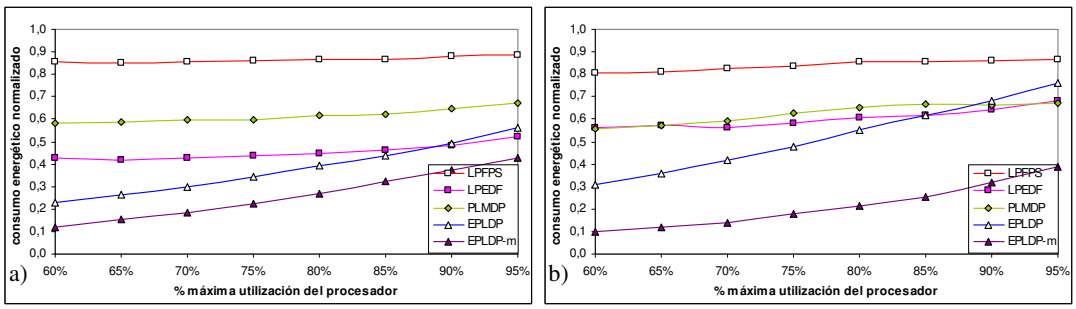

Figura 32: Consumo energético normalizado para distintas utilizaciones de procesador con un 50% de consumo de WCET en a) los periodos de las tareas son armónicos y en b) los periodos no son armónicos. La carga máxima por tarea es del 20%.

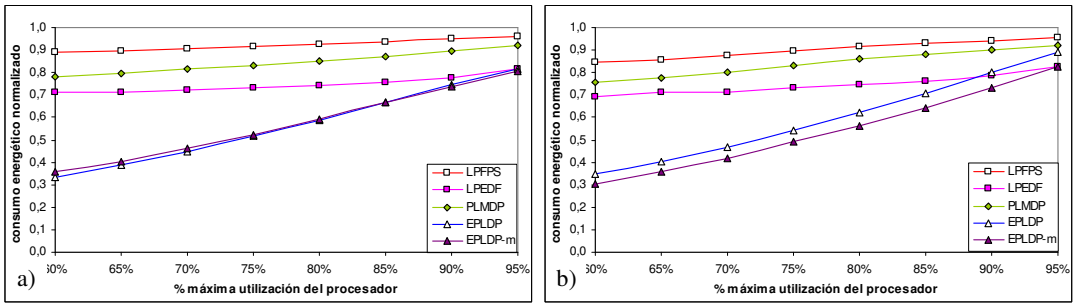

Figura 33: Consumo energético normalizado para distintas utilizaciones de procesador con un 90% de consumo de WCET en a) los periodos de las tareas son armónicos y en b) los periodos no son armónicos. La carga máxima por tarea es del 20%.

De la Figura 31 hasta la Figura 33 se puede observar la influencia que tiene la variación de carga del sistema cuando todas las tareas consumen el 20% del WCET, cuando estas consumen el 50% del WCET, y un consumo del 90% del WCET respectivamente.

% WCET consumido	LPFPS	LPEDF	PLMDP	EPLDP	EPLDP-m
20%	83%	34%	40%	33%	10%
50%	87%	45%	61%	38%	26%
90%	92%	74%	84%	56%	57%

Tabla 9: consumos medios normalizados variando la utilización máxima teórica del procesador, en conjuntos de tareas con periodos armónicos

% WCET consumido	LPFPS	LPEDF	PLMDP	EPLDP	EPLDP-m
20%	80%	57%	37%	49%	7%
50%	84%	60%	63%	52%	21%
90%	90%	75%	84%	60%	54%

Tabla 10: consumos medios normalizados variando la utilización máxima teórica del procesador, en conjuntos de tareas con periodos no armónicos

En el conjunto de las últimas 6 gráficas, se observa que el planificador EPLDP-m es el que tiene el mejor comportamiento en cualquier situación, mientras que LPFPS es el que tiene un consumo mayor. Los planificadores EPLDP y el EPLDP-m son los planificadores que reciben una mayor influencia de la variación de la utilización máxima teórica del procesador, a medida que las especificaciones se alejan de la realidad (menor consumo del WCET), el algoritmo EPLDP-m mejora el consumo energético del EPLDP. En bajo consumo del WCET, el PLMDP tiene un comportamiento mejor que el EPLDP debido a que el PLMDP es más agresivo y siempre reduce primero lo máximo posible, al tener la suerte que la tarea finaliza enseguida, permite a la siguiente tarea reducir su velocidad. En el caso del LPEDF, pasa lo mismo, la tarea intenta utilizar todo el *slack* disponible para reducir velocidad. Con consumos de WCET bajos, la tarea finaliza antes de lo previsto, añadiendo *slack* dinámico que es aprovechado por la siguiente tarea para reducir velocidad. En el caso de EPLDP lo que ocurre es que se está en el caso opuesto, este algoritmo tiene un comportamiento muy conservativo, usando lo previsto suponiendo que las especificaciones de las tareas son fiables, al desviarse la ejecución mucho de estas especificaciones, no se aprovecha totalmente el tiempo disponible para reducir energía. En el caso del LPFPS, se decide demasiado tarde reducir la velocidad del procesador, con lo que si esta tarea termina antes de lo previsto, no hay ninguna otra tarea que pueda usar el tiempo no consumido por la tarea recién finalizada.

Variación de la carga máxima de las tareas entre el 10% y el 40%.

La carga del sistema se ha fijado al 60%, 75% y 90% y los periodos varían entre 1000 y 70000 unidades de tiempo, para los conjuntos de tareas con periodos no armónicos. El número de tareas se ha establecido en 8. En las gráficas que hay a continuación se ha fijado la utilización del procesador al 75%, y se ha variado la máxima utilización de las tareas entre el 10% y el 40%. En las gráficas se ha representado el consumo del WCET igual al 10%, al 50% y finalmente al 100%.

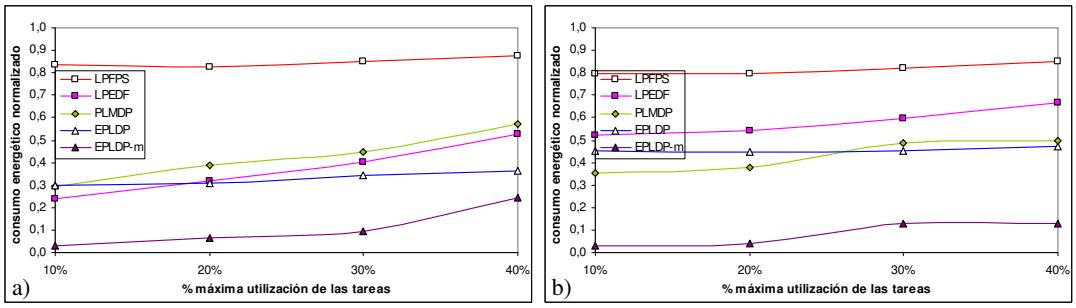

Figura 34: Variación de la carga del sistema cuando todas las tareas consumen el 20% del WCET. En a) las tareas tienen periodos armónicos y en b) los periodos no son armónicos.

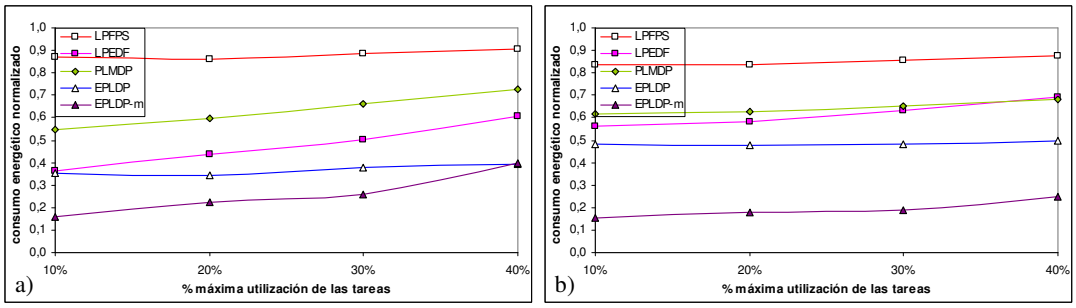

Figura 35: Variación de la carga del sistema cuando todas las tareas consumen el 50% del WCET. En a) las tareas tienen periodos armónicos y en b) los periodos no son armónicos.

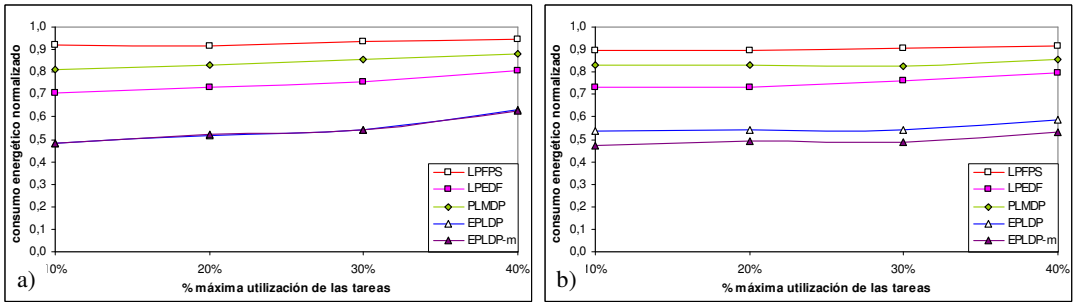

Figura 36: Variación de la carga del sistema cuando todas las tareas consumen el 90% del WCET. En a) las tareas tienen periodos armónicos y en b) los periodos son no armónicos.

En general, la conclusión que se puede sacar de estas últimas gráficas es que la carga máxima de una tarea afecta más a los algoritmos conforme estos consumen realmente todo el WCET, dándose las mayores diferencias entre los algoritmos cuando el consumo del WCET es menor. En los planificadores no se observa una diferencia significativa según sea la máxima utilización de la tarea.

% WCET consumido	LPFPS	LPEDF	PLMDP	EPLDP	EPLDP-m
20%	85%	37%	42%	33%	11%
50%	88%	48%	63%	37%	26%
90%	93%	75%	84%	54%	54%

Tabla 11: consumos medios normalizados variando la utilización máxima teórica del procesador, en conjuntos de tareas con periodos armónicos

% WCET consumido	LPFPS	LPEDF	PLMDP	EPLDP	EPLDP-m
20%	82%	58%	41%	46%	8%
50%	85%	62%	64%	49%	19%
90%	90%	75%	84%	55%	50%

Tabla 12: consumos medios normalizados variando la utilización máxima teórica del procesador, en conjuntos de tareas con periodos no armónicos

Variación del ratio entre el período máximo (Tmax) y el mínimo (Tmin) desde 0,1 a 0,00001.

La carga del sistema se ha variado entre el 60%, el 75% y el 90% y la carga máxima por tarea al 20%. A nivel de comparativa no se puede apreciar una variación importante en el comportamiento de los distintos algoritmos al variar la utilización máxima de las tareas del sistema. En las diferentes gráficas se ha variado el porcentaje de WCET consumido desde el 10% al 100% fijando la máxima utilización de las tareas al 20%. Como muestra del comportamiento se ha escogido el consumo del 20% (Figura 37), 50% (Figura 38) y 90 % (Figura 39).

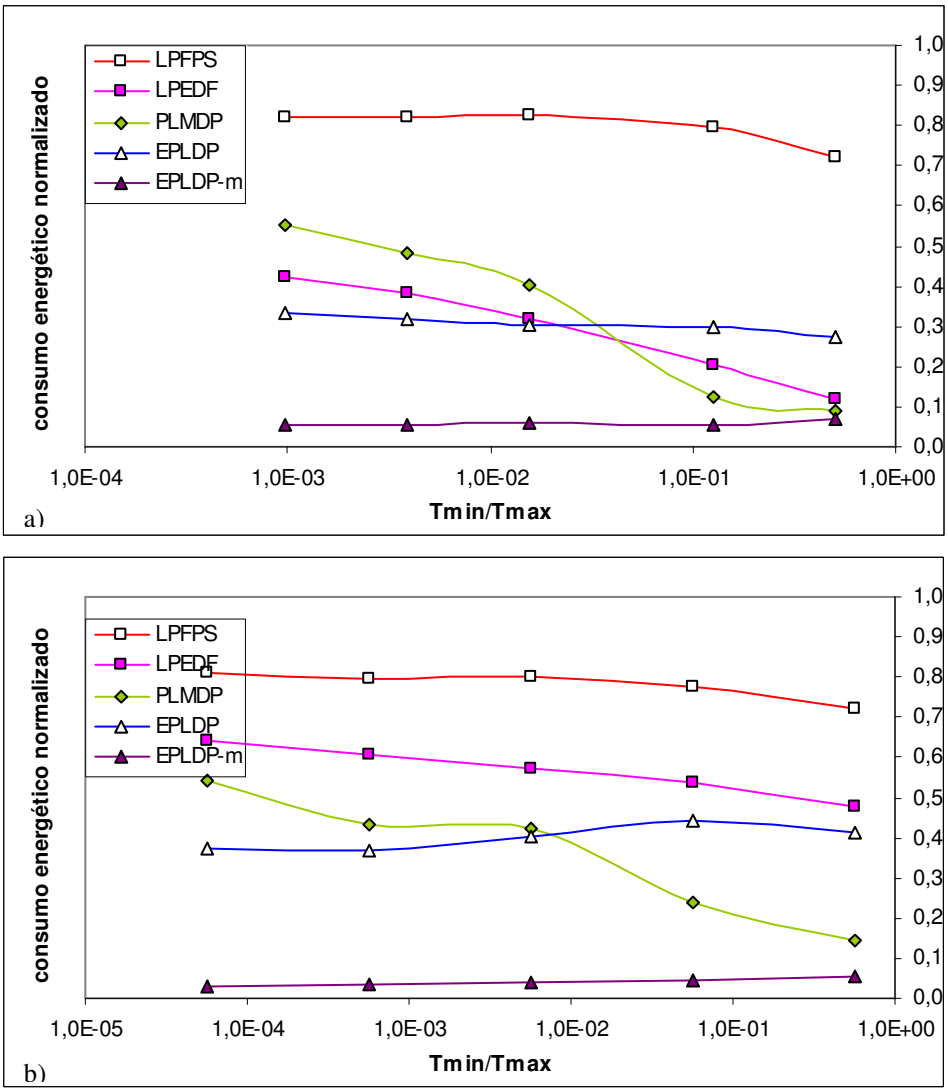

Figura 37: Variación del ratio entre Tmin/Tmax fijando la utilización del procesador al 75% y WCET consumido al 20%. La utilización máxima de las tareas está fijada al 20%. En a) los periodos de las tareas son armónicos y en b) son no armónicos.

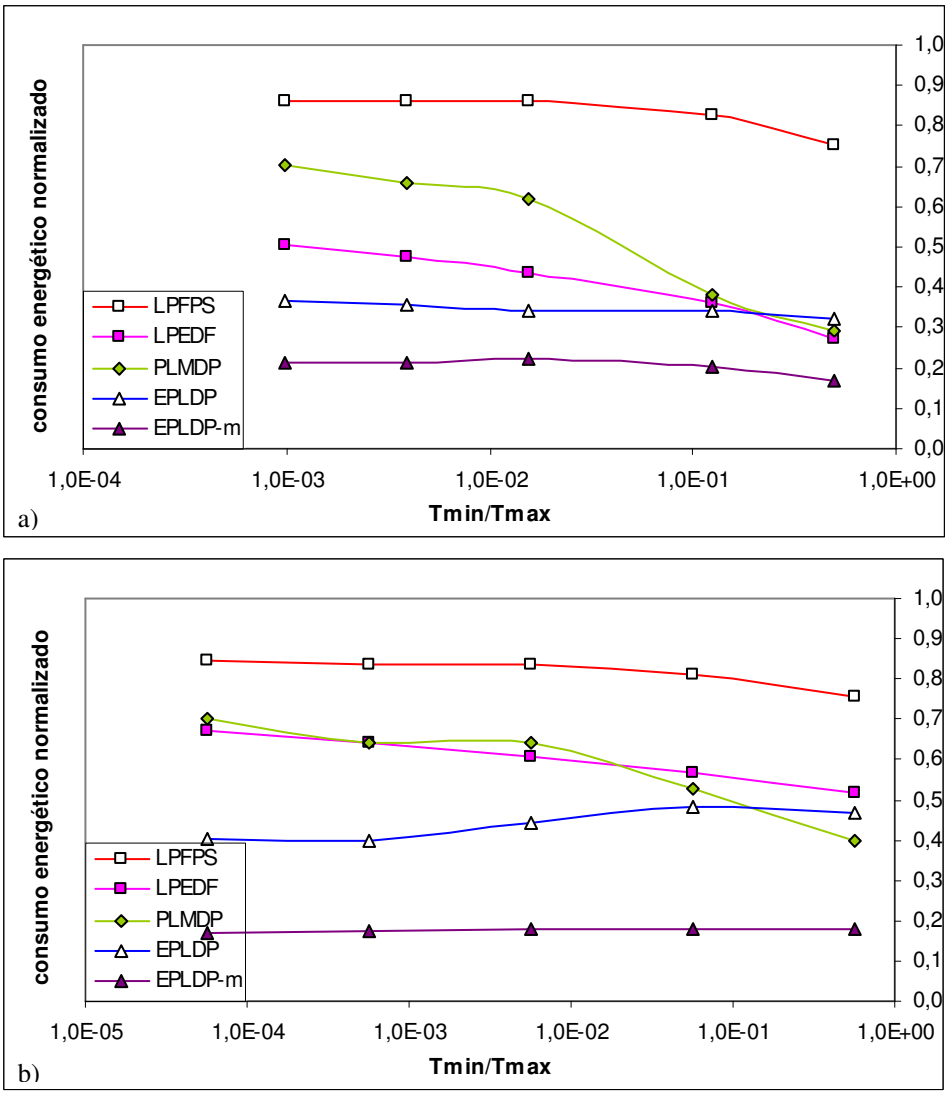

Figura 38: Variación del ratio entre Tmin/Tmax fijando la utilización del procesador al 75% y WCET consumido al 50%. La utilización máxima de las tareas está fijada al 20%. En a) los periodos de las tareas son armónicos y en b) son no armónicos.

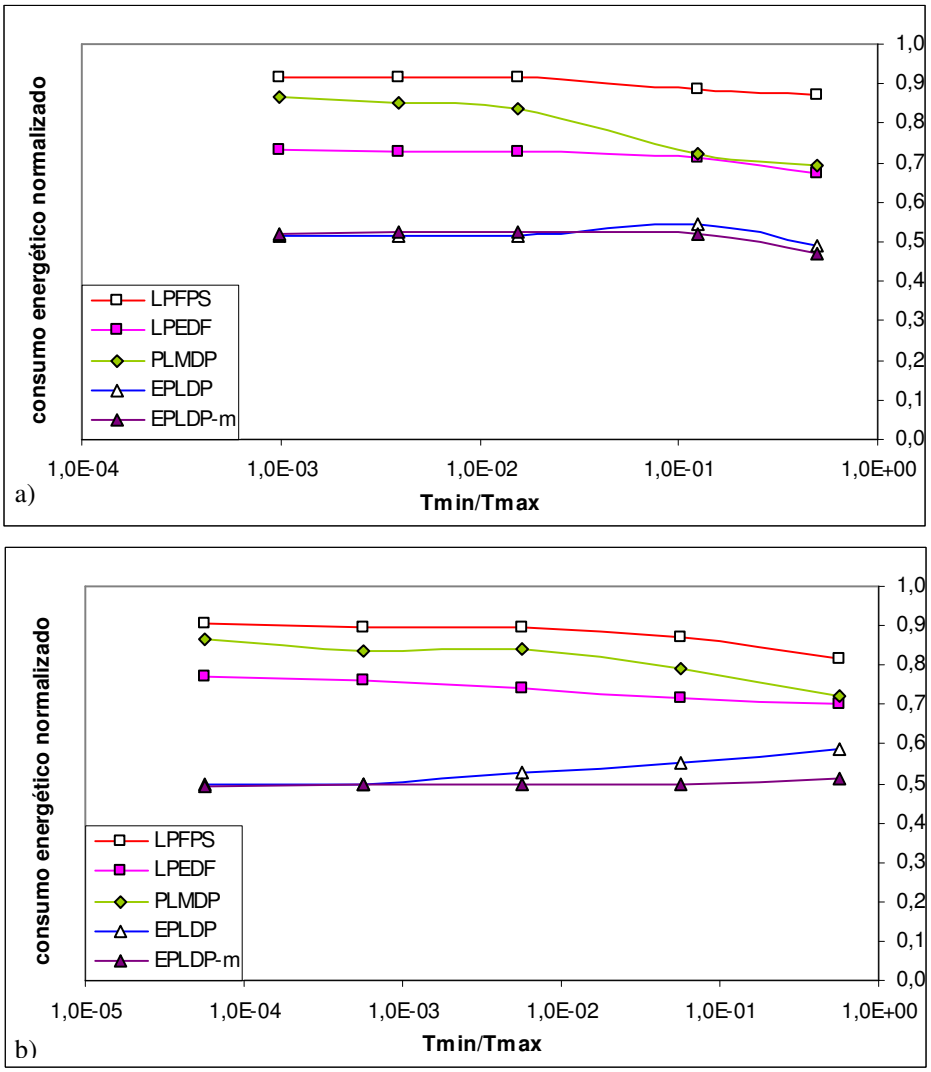

Figura 39: Variación del ratio entre Tmin/Tmax fijando la utilización del procesador al 75% y WCET consumido al 90%. La utilización máxima de las tareas está fijada al 20%. En a) los periodos de las tareas son armónicos y en b) son no armónicos

Con estas últimas figuras, se puede concluir que las diferencias de consumos energéticos entre los distintos algoritmos de planificación es mas evidente cuanto mayor es el ratio entre los periodos.

%WCET consumido	LPFPS	LPEDF	PLMDP	EPLDP	EPLDP-m
20%	80%	29%	33%	31%	6%
50%	83%	41%	53%	34%	20%
90%	90%	71%	79%	52%	51%

Tabla 13: Consumos medios normalizados variando la utilización máxima teórica del procesador, en conjuntos de tareas con periodos armónicos

%WCET consumido	LPFPS	LPEDF	PLMDP	EPLDP	EPLDP-m
20%	0,78	0,57	0,36	0,40	0,04
50%	0,82	0,60	0,58	0,44	0,18
90%	0,88	0,74	0,81	0,53	0,50

Tabla 14: Consumos medios normalizados variando la utilización máxima teórica del procesador, en conjuntos de tareas con periodos no armónicos

Con las medias entre los distintos planificadores, se puede observar que el planificador EPLDP-m es el que posee un mejor ahorro energético, seguido del EPLDP. Existe una gran diferencia a nivel de consumo energético para el planificador LPEDF según las tareas tengan un periodo armónico o no. Y las diferencias no son tan evidentes para el resto de los planificadores.

2.4 Conclusiones

En este capítulo se han desarrollado algoritmos de planificación para sistemas de tiempo real estricto que aportan un ahorro energético superior a los algoritmos simples de prioridades estáticas o dinámicas.

La implementación de los algoritmos de planificación aportan una mínima sobrecarga respecto a un algoritmo simple de prioridades estáticas. En el cálculo de la energía consumida no se ha tenido en cuenta las sobrecargas debidas al algoritmo de planificación, ni a la sobrecarga y tiempo que representa el cambio de la velocidad del procesador. Sin embargo, si que se ha tenido en cuenta que la velocidad del procesador no es lineal sino discreta, usando en este caso la velocidad mayor más próxima.

Para el análisis del algoritmo se han realizado varios casos de pruebas, tanto reales como sintéticos, con el objetivo de poder hallar un patrón entre los parámetros de las tareas del ahorro energético que se obtiene. Se han comparado nuestros algoritmos EPLDP-m, EPLDP y PLMDP con el LPEDF y el LPFPS. Todos los consumos han sido normalizados respecto a la ejecución a máxima velocidad del procesador.

En los conjuntos de tareas escogidos se ha ido variando: la utilización máxima del procesador, el número de tareas del sistema, la armonicidad en los periodos de las tareas, la utilización máxima de las tareas del conjunto, el rango entre el periodo

máximo y el mínimo. También se ha variado el porcentaje de WCET consumido entre el 10% y el 100% de consumo. Se ha comprobado que nuestros planificadores se comportan bien en cualquier situación, no siendo especialmente sensibles a la armonicidad de los periodos. El planificador EPLDP no se adapta con suficiente rapidez a bajos consumos de WCET, pero el planificador EPLDP-m soluciona este problema. El EPLDP-m calcula de manera dinámica el grado de consumo medio de las tareas, adecuándose al consumo real de las tareas y consiguiendo consumos energéticos mínimos en cualquier situación.

Además, de las anteriores características, debido a la propia naturaleza del algoritmo base escogido, se puede asegurar que nuestros algoritmos se adaptaran bien a la llegada de tareas aperiódicas y debido a que posee prioridades estáticas, será estable en caso de sobrecarga de alguna de sus tareas, de manera que en caso de que en alguna tarea se produzca un sobrecarga del procesador las posibles tareas que pueden perder el plazo son las de prioridad igual o inferior a la que ha producido el sobrecarga.

Capítulo 3

RECURSOS COMPARTIDOS.

El objetivo de este capítulo es la planificación con ahorro energético de un sistema de tiempo real estricto en el que diversas tareas del sistema comparten recursos software (estructuras compartidas) o recursos hardware (dispositivos compartidos), ambos en forma de secciones críticas. Para ello se ha introducido un protocolo que minimice los problemas de la inversión de prioridades que puede presentarse en el acceso con exclusión mutua a los recursos.

3.1 Introducción

Los sistemas de tiempo real están formados por múltiples tareas que se ejecutan de manera concurrente, o en paralelo en el caso de las arquitecturas con múltiples procesadores. Si estas tareas no son independientes sino que interaccionan entre si la planificación resulta mucho más compleja. Cuando las tareas cooperan entre si, es imprescindible la sincronización entre ellas, ya sea mediante sincronismo en el uso de variables compartidas, dispositivos o zonas de exclusión mutua entre las distintas tareas, ya sea mediante la precedencia temporal, una tarea debe finalizar antes del inicio de la siguiente. Sin embargo, en la mayoría de estudios de planificación, se ignoran estas especificaciones de sincronismo entre las tareas. El algoritmo de planificación debe asegurar la consistencia de los recursos compartidos, por lo tanto, se debe garantizar la exclusión mutua entre las tareas que compiten por el uso de estos recursos (el código que debe ejecutase en exclusión mutua se denomina sección crítica).

El modelo de tareas que se usará es el siguiente: Tareas periódicas independientes sin acceso a los recursos compartidos, tareas periódicas que acceden a los recursos compartidos en algún instante de su ejecución. El tiempo máximo de acceso al recurso compartido es conocido y se halla incluido en el WCET de la tarea periódica.

Cuando se utilizan algoritmos de planificación que utilizan distintas prioridades para las tareas junto con el acceso a recursos compartidos, se puede presentar el problema de la inversión de prioridades que tiene lugar cuando una tarea de una cierta prioridad esta forzada a posponer su ejecución por otra tarea de prioridad menor con el fin de preservar la exclusión mutua. (Se dice que la tarea de menor prioridad bloquea a la tarea de mayor prioridad.) Además, la determinación del tiempo de bloqueo (tiempo en que la tarea de mayor prioridad esta pendiente de la finalización de la tarea de menor prioridad) en el peor caso no está acotado puesto que otras tareas de prioridad intermedia pueden, a su vez, retardar la ejecución de la tarea de prioridad inferior que posee el recurso. Este tiempo de bloqueo no acotado, afecta seriamente la predecibilidad del sistema de tiempo real [14].

Un mecanismo de coordinación en el acceso a los recursos compartidos en la planificación con sistema de prioridades estáticas, consiste en el uso de protocolos de

cambio de prioridad inducido por el acceso al recurso compartido. Por ejemplo, los protocolos definidos por de Sha, Rajkumar y Lehoczky en [76] que son el protocolo de herencia de prioridad (*Priority Inheritance Protocol*), el protocolo de techo de prioridad original (*Priority Ceiling Protocol*) y el protocolo de techo de prioridad inmediato (*Immediate Ceiling Priority Inheritance Protocol*). En los tres protocolos se produce un cambio en la prioridad de la tarea que accede o posee el recurso, pero con propiedades diversas al producirse el cambio debido a distintos motivos:

- En el protocolo de herencia de prioridad, el cambio de prioridad se produce en el instante en que una tarea solicita un recurso ocupado por otra tarea de menor prioridad, la tarea que posee el recurso hereda la prioridad de la tarea que lo solicita. Al abandonar el recurso la tarea recupera la prioridad que tenia inicialmente. Este protocolo acota el tiempo de bloqueo, siendo este la el mínimo entre: la suma de tiempo que todas las tareas de menor prioridad ocupan el recurso y la suma del acceso a todos los recursos compartidos del sistema. Un problema de este protocolo es que si una tarea utiliza diferentes recursos, puede estar bloqueada en todos ellos. Este fenómeno se conoce como bloqueo en cadena y ocasiona un tiempo de bloqueo excesivo no acotado. Una tarea puede verse obligada a modificar su prioridad varias veces durante la ocupación de un recurso. Este protocolo no evita el interbloqueo entre las tareas. (Un interbloqueo se produce por ejemplo en la siguiente situación: una tarea posee un recurso x y esta bloqueada esperando la liberación de un recurso y. La tarea que posee el recurso y también esta bloqueada esperando el recurso x. Ambas tareas se encuentran en un interbloqueo, no pudiendo progresar ninguna de las dos.)
- En el protocolo de techo de prioridad original, al igual que en el anterior, el cambio de prioridad se produce en el instante en que una tarea solicita un recurso ocupado por otra tarea de menor prioridad. En este caso la prioridad que obtendrá será la máxima prioridad de todas las tareas que puedan acceder al recurso (techo de prioridad). El techo de prioridad de un recurso puede calcularse de manera estática (antes de iniciar la ejecución de la aplicación). Una tarea solo puede bloquear un recurso si su prioridad es estrictamente mayor que el techo de prioridad de todos los recursos que en ese momento estén ocupados. Al abandonar el recurso recupera la prioridad inicial. Este

protocolo acota el tiempo de bloqueo, siendo este el máximo tiempo que pueden ocupar el recurso las tareas de menor prioridad. Este protocolo impide los bloqueos en cadena y el interbloqueo. Un problema de este protocolo es que puede bloquear tareas sin necesidad, debido a la ocupación de un recurso con un techo de prioridad mayor que la tarea no necesitará. La implementación de este protocolo es compleja, debido a que se necesita un control estricto de los recursos que están ocupados en cada instante.

- En el protocolo de techo de prioridad inmediato, el cambio de prioridad se produce en el instante en que la tarea bloquea el recurso. En este caso la prioridad que obtendrá será el techo de prioridad del recurso al que accede. Este protocolo tiene las mismas ventajas que el anterior, pero su implementación es más sencilla, incluyéndose en el estándar POSIX 1000.1c (*Pthreads*) bajo el nombre de *Priority Protected Protocol*.

Un refinamiento de los protocolos anteriores de cambio de prioridad es el protocolo de *Stack-based* diseñado por Baker en [9] que permite la planificación con prioridades dinámicas y estáticas.

En el momento de diseñar nuestro algoritmo no existían (según nuestro conocimiento) otros algoritmos de planificación con ahorro energético que permitieran el uso de recursos compartidos. En las mismas fechas, R. Jejurikar y R. Gupta en [33] y [34] publicaron una serie de algoritmos basado en EDF y en RMS respectivamente. Sin embargo ambos algoritmos el computo de la velocidad del procesador se realiza de manera estática, sin poder aprovechar el WCET no consumido por la tarea. En sus artículos presentan cuatro posibilidades:

- Calcular la velocidad de ejecución considerando que las secciones críticas se ejecutaran a máxima velocidad, con lo que se reduce el tiempo de bloqueo de las tareas de mayor prioridad.
- Calcular la velocidad de ejecución constante durante toda la ejecución de la tarea, incrementando con ello el tiempo de bloqueo de las tareas de mayor prioridad.
- Incrementar el WCET de las tareas candidatas a ser bloqueadas, y considerar a partir de este punto que las tareas son independientes.

- Añadir una nueva tarea de máxima prioridad con WCET igual al máximo tiempo de bloqueo que se pueda producir con periodo el máximo del periodo de las tareas del sistema. A partir de este punto se consideraran las tareas como independientes.

Estas cuatro aproximaciones las comparan con el planificador FPS de Shin y Choi, y se demuestra que existe un ahorro de energía promedio del 25-30%. Donde los algoritmos son de complejidad polinómica. En todas sus aproximaciones se considera una variación continua de la velocidad del procesador.

La consecución del objetivo del presente capítulo vendrá dado por la inmersión del protocolo de techo de prioridades inmediato en los algoritmos para ahorro energético de prioridades estáticas propuestos anteriormente, con lo que se permitirá que el conjunto de tareas utilice recursos compartidos. Para finalizar el capítulo se realizará la evaluación de los resultados obtenidos en los diversos algoritmos.

3.2 Metodología y desarrollo.

En este punto se expondrán las ideas básicas de la modificación que se realizará de los algoritmos desarrollados en los capítulos anteriores, con el fin de permitir la sincronización de los procesos. La idea principal de la modificación es la inmersión de protocolos de cambio de prioridad en los algoritmos que se utilizan.

El algoritmo de planificación que se usará es una modificación del algoritmo *Power Low Modified Dual Priority Scheduling* desarrollado en el capítulo 2 de la presente tesis. Este algoritmo, tal como se ha comentado anteriormente, se basa por una parte en el algoritmo de prioridad dual, propuesto originalmente por Burns y Wellings [13] y modificado por Davis y Wellings [23], y por otra en el algoritmo de ahorro energético *Fixed Priority Scheduling Algorithm* propuesto por Shin y Choi [77].

Nosotros proponemos una modificación de estos algoritmos de planificación con ahorro energético para imbuir el protocolo de techo de prioridades inmediato que Sha, Rajkumar y Lehoczky en [76] proponen como solución al problema de inversión de prioridades.

Las modificaciones que deben realizarse en el algoritmo *Power Low Modified Dual Priority Scheduling* son las siguientes:

- Aumento del WCRT (*Worst Case Response Time*) de la tarea i (τ_i) debido a los bloqueos que pueden producir las tareas de menor prioridad que usan los recursos compartidos a los que quiere acceder la tarea τ_i. El WCRT se calcula según la siguiente formula recursiva :

$$WCRT_i^{n+1} = B_i + C_i + \sum_{j \in hp(i)} \left\lceil \frac{WCRT_i^n}{T_j} \right\rceil * C_j \qquad \text{(ec. 14)}$$

  donde `Bi` es el tiempo máximo de bloqueo que se determinará mediante el siguiente algoritmo:

$$B_i = \max_{k=1}^{K} \left(utilización(k,i) * C(k) \right) \qquad \text{(ec. 15)}$$

  donde utilización(k,i) es la fracción de tiempo de computo de la tarea τ_k en su interferencia con la tarea τ_i en el acceso a algún recurso compartido.

- Como consecuencia del punto anterior, el tiempo de promoción (Tp) disminuye, dado que éste se calcula como Tp=(D-WCRT) Este hecho repercutirá de forma negativa en el ahorro energético, dado que las tareas promocionarán más rápido a la URQ, siendo en el peor de los casos (tiempo de promoción igual al plazo temporal) un esquema equivalente al algoritmo LPFPS, en el que la única posibilidad de reducir la velocidad del procesador está ligada a la existencia de una única tarea en la cola de preparados.
- Cambio de la prioridad de las tareas durante el tiempo de acceso a los recursos compartidos, incrementando momentáneamente la prioridad la tarea al techo del recurso. En el algoritmo PLMDP las tareas disponen de dos niveles de prioridades según se hallen en la URQ o en la LRQ. Recordemos que las tareas de la URQ son aquellas en las que las tareas no permiten ningún retraso causado por tareas no previstas. En nuestro caso, el techo de prioridad de los recursos será igual a la prioridad de la tarea de mayor prioridad en la URQ que utilice el recurso.

Para poder evaluar la eficiencia de nuestro algoritmo y al no disponer de ningún algoritmo de planificación *on-line* con prioridades dinámicas, ahorro energético y acceso a recursos compartidos, se ha modificado el algoritmo propuesto por Shin y

Choi [77] insertándole el mismo protocolo de acceso a los recursos compartidos de manera que se pueda comprobar con las mismas condiciones si la mejora energética obtenida es aún aceptable.

3.2.1 Ejemplo de funcionamiento

A continuación en la Figura 40 y en la Figura 41 se muestra la ejecución de las tareas a velocidad reducida según el algoritmo de Shin y Choi modificado para trabajar con recursos compartidos (LPFPSR) y el nuestro (PLMDPR). En este ejemplo, para simplificar la casuística, se considerará inicialmente que si una tarea usa el recurso compartido X, lo utilizará durante toda su ejecución (ver Tabla 15).

Tarea	Periodo	Plazo temporal	WCET	Tp	Recurso compartido
T1	50	50	10	0	X
T2	80	80	20	0	
T3	100	100	40	20	X

Tabla 15.Caracterización del conjunto de tareas de prueba.

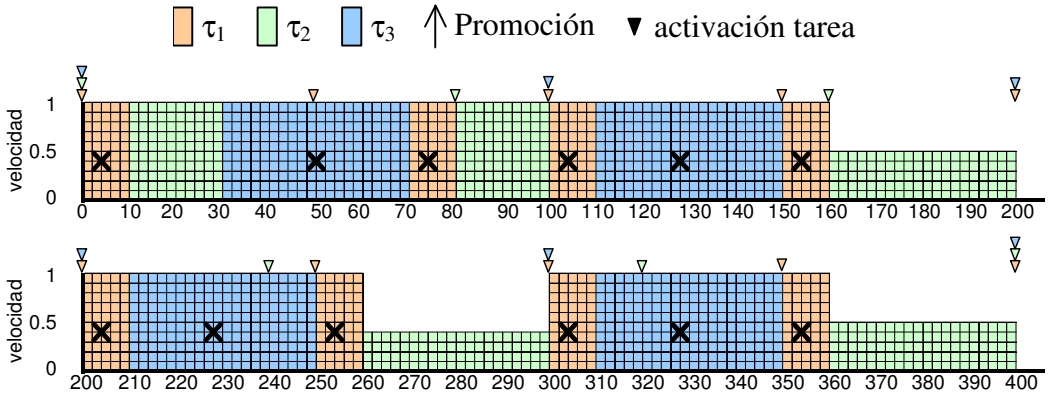

Figura 40: Planificación LPFPSR cuando todas las tareas usan el 100 % del WCET. Una X significa que la tarea se halla en sección crítica.

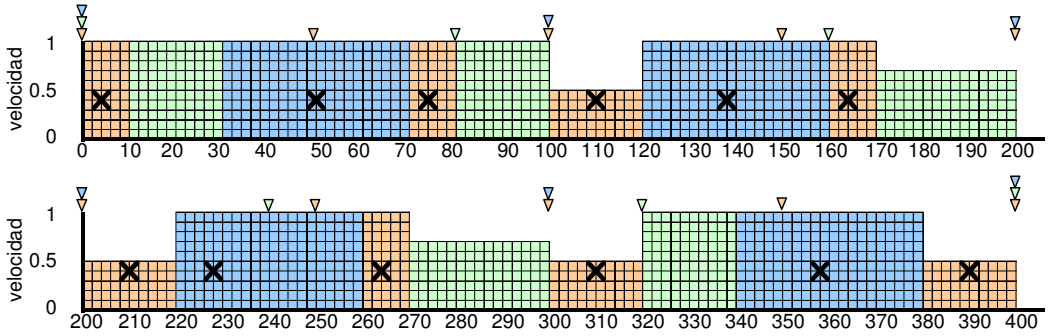

Figura 41: Planificación PLMDPR cuando todas las tareas usan el 100 % del WCET. Una X significa que la tarea se halla en sección crítica.

Si se observa como se produce la ejecución de las distintas tareas durante un hiperperiodo, se puede ver que en la Figura 40 existen tres intervalos de tiempo en los que el algoritmo LPFPSR puede reducir velocidad, coincidiendo con la existencia de una única tarea en la cola de tareas listas para ejecutarse. En cambio, con el algoritmo PLMDPR (Figura 41) se puede reducir velocidad en seis posibles intervalos. Esto es debido a la doble capa de prioridades. Observando el funcionamiento de ambos algoritmos se puede distinguir las siguientes diferencias:

En el instante t=100. Llegan al sistema τ_1 y τ_3.

- Con el algoritmo LPFPSR ambas tareas se hallan en la cola de listos para ejecutarse con lo que τ_1 empieza a ejecutar a máxima velocidad. Cuando finaliza empieza la ejecución de τ_3, que al ser la única tarea de la cola podría reducir la velocidad, pero no en el instante t=150 llegará una nueva instancia de τ_1 con lo que τ_3 debe ejecutarse a máxima velocidad.
- Con el algoritmo PLMDPR τ_1 entra en la URQ debido a que su tiempo de promoción es 0 desde el instante de llegada, mientras que τ_3 pasa a la LRQ y no promocionará hasta dentro de 20 unidades de tiempo. Por tanto τ_1, puede ejecutarse a velocidad reducida durante estas 20 unidades.

En el instante t=160. Llega al sistema τ_2.

- Con el algoritmo LPFPSR ésta es la única tarea lista para ser ejecutada, por lo que puede reducir la velocidad de ejecución hasta la llegada de la siguiente tarea en t=200.
- Con el algoritmo PLMDPR al haber reducido anteriormente, en la cola lista para ser ejecutada aun hay la tarea τ_1, con lo que es esta tarea la que se ejecutará. Al finalizar queda una única tarea en la URQ que puede reducir su velocidad de ejecución.

En el instante t=200. Llegan al sistema τ_1 y τ_3, y se encuentra la misma situación que en el instante t=100, al igual que en instante t=300.

3.2.2 Planificabilidad del sistema.

En este punto se definirá la caracterización general de las tareas que se pueden tratar con los algoritmos que se describirán a continuación. El sistema de tiempo real dispondrá de tareas periódicas independientes (especificadas por su periodo, peor tiempo de cálculo y plazo temporal) y tareas periódicas con acceso a recursos compartidos (especificadas por periodo, peor tiempo de cálculo, plazo temporal, tiempo máximo de acceso a los recursos).

Cuando una tarea accede a un recurso compartido incrementa su prioridad a la del techo del recurso al que accede, retornando a su prioridad original cuando la tarea abandona el recurso. Este hecho se simulará mediante la división conceptual de la tarea en subtareas. De manera que la tarea se comportará como una cadena de precedencia (ver Capítulo 4), donde la suma de los peores tiempos de cálculo de las tareas será igual al peor tiempo de cálculo de la tarea original, el periodo de la primera tarea de la cadena será el periodo de la tarea original, y finalmente el plazo temporal de la última tarea de la cadena será también el plazo temporal de la tarea original. Con esta división de la tarea, se podrá conseguir que la subtarea que acceda a un recurso compartido lo haga durante toda su ejecución.

Veamos un ejemplo de esta división en subtareas Figura 42. Supongamos un tarea τ_1 con periodo T=24 unidades temporales, un plazo temporal D=20 unidades y un WCET=7 con una prioridad de 5 (a menor número, mayor prioridad). Esta tarea usará, después de una ejecución máxima de 2 unidades, un recurso compartido σ_1 durante un tiempo máximo de 4 unidades, después liberará el recurso hasta la siguiente instanciación de la tarea. Tal como se puede observar en la Figura 42, la tarea τ_1 se dividirá en tres subtareas: En la primera y tercera subtarea no hay ningún acceso a ningún recurso compartido, mientras que en la segunda subtarea se accede durante todo su ejecución a σ_1 (sección crítica). Para asegurar la precedencia, tan pronto acabe la primera subtarea enviará un mensaje a la segunda, provocando su inicio. Al necesitar esta subtarea acceder al recurso σ_1 se aplicará el protocolo de techo de prioridad inmediato, ejecutándose a la prioridad del techo del recurso, que en este caso se ha calculado que es 3. Tan pronto finalice enviará un mensaje a la tercera subtarea que se ejecutará a su prioridad base. Para simplificar se supone que el envío del mensaje es instantáneo. Asimismo se han considerado despreciables los

tiempos de cambio de contexto, de planificación y de cambios de la velocidad de ejecución del procesador. Estas simplificaciones son validas debido por una parte a que los cambios de contexto y ejecución del planificador puede ser incorporado al WCET que se haya sobredimensionado, y por otra a que la velocidad del procesador que se escoge siempre es una velocidad mayor a la necesaria (por ser el rango de velocidades discretas en lugar de lineales).

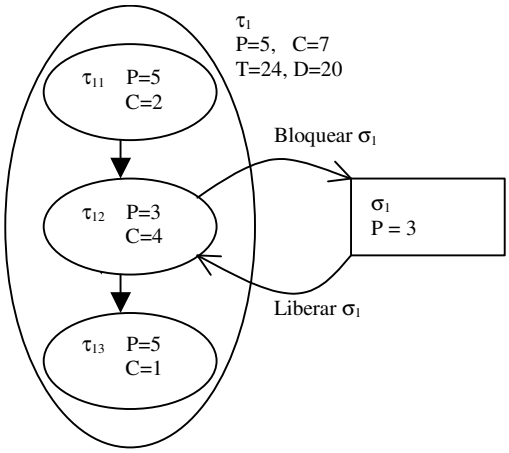

Figura 42: Ejemplo de la división de una tarea con acceso a un recurso compartido σ_1 en subtareas.

Para aplicar el protocolo de techo de prioridad inmediato, inicialmente se tiene que calcular para cada recurso compartido su techo de prioridad que será la mayor prioridad de todas las tareas que puedan acceder a este recurso. Recordemos que nuestro algoritmo posee prioridades estáticas. El peor tiempo de bloqueo que una tarea τ_i puede experimentar debido al protocolo está determinado por la sección crítica mayor de las tareas de menor prioridad que accedan a recursos con techos iguales o mayores que la prioridad de la tarea τ_i.

La planificabilidad del sistema se puede asegurar de manera estática utilizando la técnica propuesta por Joseph y Pandya [36], extendida teniendo en cuenta los recursos compartidos por Audsley en [4]. En los artículos se propone una formula iterativa para calcular el WCRT, teniendo en cuenta los posibles bloqueos de las tareas de menor prioridad, que será comparado con el plazo temporal para verificar la planificabilidad del sistema. El WCRT inicial es 0, y la iteración n+1 se calcula de la siguiente manera:

$$WCRT_i^{n+1} = B_i + C_i + \sum_{j \in hp(i)} \left\lceil \frac{WCRT_i^n}{T_j} \right\rceil * C_j \qquad \text{(ec. 16)}$$

donde B_i es el máximo tiempo de bloqueo debido a tareas de menor prioridad y hp(i) es el conjunto de tareas que pueden expulsar del procesador a la tarea τ_i, es decir, es el conjunto de tareas con una prioridad mayor a la tarea τ_i.

El proceso iterativo empieza con $WCRT_i=0$ y finaliza cuando $WCRT_i^{n+1}=WCRT_i^n$ o bien cuando $WCRT_i^{n+1}>D_i$. Sin embargo en este último caso, el sistema no será planificable, ya que el tiempo de finalización calculado hasta el momento es mayor que el plazo temporal.

```
L1    Bi=0
L2    Para todas las tareas τj con menor prioridad que τi
L3        Para todas las subtareas τjk ∈ τj
L4            Si τjk accede a algún recurso σp y
L5               el techo de σp ≤ prioridad que τi
L6            Entonces
L7                Bi=max(Bi,WCET τjk)
L8            Fin si
L9        Fin para
L10   Fin para
```

Figura 43: Algoritmo para la determinación del peor tiempo de bloqueo de la tarea τ_i.

En nuestro sistema, el análisis de la planificación de las tareas que tienen precedencias en un mismo procesador se realizará sin tener en cuenta la división en subtareas, puesto que el planificador empieza la tarea posterior en la cadena solo cuando la subtarea precedente ha finalizado, y se ha supuesto que el tiempo de envío del mensaje es despreciable debido a que ambas tareas se ejecutan en el mismo procesador. El cálculo del máximo tiempo de bloqueo debido a tareas de menor prioridad (B_i) para una tarea τ_i se calcula utilizando el algoritmo de la Figura 43, basado en [76].

3.2.3 Aplicación del protocolo a los algoritmos de planificación.

La idea principal de cualquiera de nuestros algoritmos de planificación es el de planificar las subtareas en lugar de planificar la tarea completa, ya que de esta forma es posible reasignar las prioridades con mayor sencillez aplicando el protocolo de techo de prioridad inmediato cuando sea necesario.

Inicialmente las prioridades estáticas están asignadas de manera que cada tarea tenga una prioridad distinta. Sin embargo todas las subtareas de una misma tarea tendrán la misma prioridad. El planificador aumentará la prioridad de la subtarea en el momento en que esta acceda a un recurso compartido según indica el protocolo, asignándole la prioridad que indica el techo del recurso al que accede.

El planificador *Power Low Modified Dual Priority* con recursos compartidos (PLMDPR) define para todas las tareas periódicas dos niveles de prioridad, tal como se ha visto en el punto 2.2.4. Un nivel de mayor prioridad (URQ) y uno de menor prioridad (LRQ). El techo del recurso se corresponderá a la prioridad de la tarea de mayor prioridad que acceda al recurso. De las dos posible prioridades, se escogerá la de mayor prioridad, es decir, la prioridad de la tarea en la URQ.

El algoritmo de planificación PLMDPR se basará en los siguientes eventos:

- Activación de una tarea periódica τ_i : Se encolará la tarea τ_i en la LRQ ordenada según el instante de promoción, que se corresponde a $WCRT_i - D_i$ calculado de manera estático antes de iniciar la aplicación.
- Llegada del instante de promoción de la tarea τ_i : La tarea τ_i promociona desde la LRQ a la URQ.
- Adquisición de un recurso compartido: La tarea incrementa su prioridad al máximo entre la prioridad actual y el techo del recurso.
- Liberación de un recurso compartido: La tarea vuelve a la prioridad original que tenia antes de bloquear el recurso.
- Finalización de una subtarea: Envío instantáneo del mensaje de inicialización de la siguiente subtarea, y activación de ésta, con la misma prioridad que la tarea precedente. Si la subtarea no tiene sucesora, el planificador escogerá la

siguiente tarea para ser ejecutada seleccionando la primera disponible desde el nivel de prioridad mayor al nivel de prioridad menor (URQ, MRQ, LRQ).

Cuando el planificador ha decidido que tarea ejecutará, antes de empezar la ejecución de ésta, deberá calcular la velocidad del procesador a la que ejecutará la tarea, teniendo en cuenta el tiempo máximo de ejecución posible de las tareas según se vio en el punto 2.2.4. El cálculo de la velocidad se halla explicado en capítulo anterior.

3.2.4 Cálculo de la velocidad de ejecución del procesador.

La principal diferencia entre los algoritmos PLMDP y EPLDP, tal como se vio en el Capítulo 2, se basa en el establecimiento de una cota en la máxima reducción de la velocidad. Esta cota será variable en función del comportamiento del sistema, intentando monitorizar la utilización total del procesador.

En este punto se va a realizar una reflexión sobre este cálculo de velocidad. Se ha visto que la inmersión del protocolo de techo de prioridad inmediato en el algoritmo de planificación provoca un aumento del WCRT de las tareas provocado a su vez por el tiempo de bloqueo que producen las tareas de menor prioridad que usan los recursos compartidos, que conlleva una disminución del tiempo de promoción. Esto implica que las tareas promocionaran antes y por lo tanto habrá menores oportunidades para reducir la velocidad del procesador.

El tiempo de bloqueo está calculado suponiendo que la tarea se ejecuta a máxima velocidad. En nuestros planificadores, la reducción de velocidad es independiente del uso de los recursos, esto implica que una tarea puede reducir la velocidad de ejecución aunque tenga asignado un recurso, por lo tanto, se debe asegurar que la reducción de la velocidad cuando la tarea usa un recurso no pueda provocar la perdida de ningún plazo temporal. El problema puede darse cuando el tiempo de bloqueo real sea mayor que el tiempo de bloqueo calculado. El tiempo de bloqueo real de la tarea de mayor prioridad es el tiempo que pasa desde que se activa, sin poder ejecutarse debido a la ejecución de una tarea de menor prioridad con un recurso, hasta que empieza la ejecución de esta. A continuación se realizará un análisis de los posibles casos: Imaginemos que en el sistema hay dos tareas: τ_1 y τ_2. τ_2 en ejecución, τ_1 mayor prioridad que τ_2, esto implica que τ_2 esta bloqueando el recurso.

- τ_1 en LRQ. La tarea puede retrasarse sin causar ningún problema, tal como demuestra el planificador Dual Priority. Sin embargo debido al protocolo de techo de prioridad inmediata la tarea τ_2 se esta ejecutando a la prioridad del techo del recurso (URQ) y por tanto momentáneamente tiene mayor prioridad que τ_1. τ_1 puede retardarse sin problemas hasta su instante de promoción.
- τ_1 promociona a URQ. En URQ hay como mínimo τ_1 y τ_2 se esta ejecutando, por tanto no se puede reducir la velocidad del procesador, ejecutándose τ_2 a máxima velocidad, por lo que el tiempo de bloqueo real será igual o menor que el calculado.

3.3 Comparativa entre los algoritmos.

A continuación, se evaluaran únicamente los algoritmos PLMDP y el LPFPS contra ejecutar siempre a máxima velocidad, con el fin de observar las distintas características energéticas de cada uno. En este punto no se compararan nuestros algoritmos con ningún planificador con asignación de prioridades dinámicas tipo EDF, debido a que el protocolo que se usa en el acceso a los recursos compartidos es específico para planificadores con prioridades estáticas. Ni se comparará con el EPLDP, que consiste en limitar la reducción de velocidad del procesador, adecuándose a la carga del sistema pendiente de realizarse.

Cada experimento constará de un conjunto de tareas independientes que pueden o no acceder a recursos compartidos. Cada tarea está caracterizada por un periodo (T), un plazo temporal (D) y un peor tiempo de cálculo (WCET) y una prioridad relativa a las otras tareas del conjunto (P). Las tareas con acceso a recursos compartidos están especificadas por el instante de acceso al recurso, y el WCET que consumirán dentro del recurso. En los experimentos que se han realizado se ha variado el porcentaje de WCET consumido por las distintas tareas y subtareas, para comprobar si el algoritmo de planificación se adapta a esta característica dinámica en el comportamiento real de las tareas. La variación que se ha tenido en cuenta va desde el 10% hasta el 100% del WCET. La duración de cada experimentos normalmente será de un hiperperiodo (mínimo común múltiplo de los periodos de las tareas que forman un conjunto de pruebas) en el caso que el comportamiento del consumo del WCET sea siempre el

mismo para todas las activaciones de las tareas, y sobre 100 hiperperiodos en el caso de que la variación del porcentaje de WCET consumido sea variable, de media el porcentaje de WCET representado en la gráfica.

Todos los resultados se muestran normalizadas respecto a uno de los algoritmos de planificación, normalmente será el que se ejecute a la máxima velocidad del procesador, que será el que tendrá un mayor consumo energético, por lo tanto, se representarán valores relativos entre 0 y 1, para evitar así las posibles diferencias entre los distintos conjuntos de prueba.

En los casos conocidos no existe constancia ni del momento ni de la duración en el acceso a los recursos compartidos, por este motivo, se analizará únicamente el caso de control automático (CNC) [38] como caso real, dado que para los otros casos, aviónica [50] y navegación [15], no se ha encontrado un caso general planificable en el acceso a los recursos por parte de las tareas.

3.3.1 Caso real: Caso de control automático.

Las características del conjunto de tareas para el caso de CNC [38] se muestran en la Tabla 5. En la Figura 44 se muestra el diagrama de acceso a los recursos compartidos por parte de las tareas del sistema. En esta figura, los óvalos representan las tareas periódicas y los cuadrados los recursos compartidos. Las direcciones de las flechas significan el uso que se hace del recurso, si se consulta o si se modifica. Al no estar especificado el cuando se accede al recurso, ni el tiempo de acceso a este. Se define, como es habitual, la sección crítica (cs) el intervalo de tiempo en que la tarea bloquea el recurso y la sección no crítica (ncs) el intervalo de tiempo en que la tarea no utiliza el recurso. En este caso, según las tareas bloqueen o liberen los recursos se pueden observar los siguientes casos:

- cs-ncs. Bloqueo del recurso al inicio de la tarea y después se libera continuándose la ejecución de esta.
- ncs-cs. Primero no se bloquea el recurso, se accede a él al cabo de un intervalo de tiempo, liberándose al final de la ejecución de la tarea.
- cs-ncs-cs. Bloqueo inicial del recurso, liberación del recurso, finalizando la tarea con otro bloqueo del recurso.

Con estas tres posibilidades, se han caracterizado todos los casos posibles en los que la tarea puede acceder a un recurso compartido.

En las simulaciones se ha variado el tiempo en el que una tarea esta ejecutando una sección crítica, es decir, está usando un recurso con exclusividad. Este tiempo se ha variado des de el 0% (no usa el recurso) al 100% del WCET (usa el recurso durante toda la ejecución) para tratar todos los casos posibles.

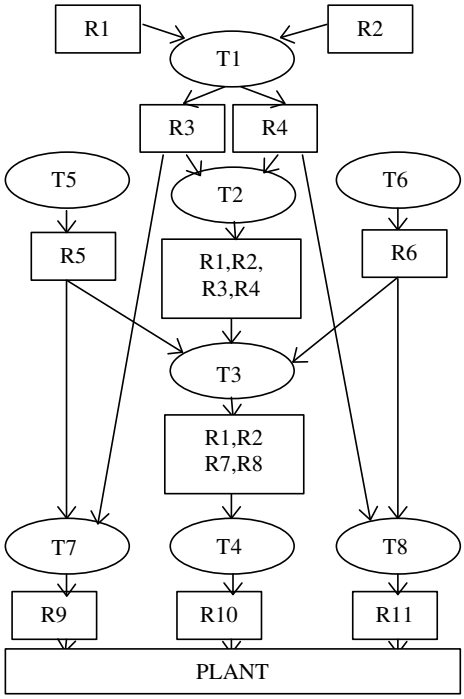

Figura 44: Diagrama del acceso a los recursos compartidos por parte de las tareas en el caso de control automático [38].

En las gráficas siguientes, se han representado los resultados en términos del promedio de energía consumida por el procesador. Todas las simulaciones cubren un hiperperiodo, y las tareas consumen siempre el mismo porcentaje de WCET. Tal como se ha visto en el Capítulo 2, es el mismo resultado que se obtendría si la simulación se hubiera realizado con las tareas consumiendo un WCET variable de promedio el indicado, y haciendo durar la simulación varios hiperperiodos.

De la Figura 45 a la Figura 47, se muestra el comportamiento de los dos algoritmos cuando se varia el tiempo de ejecución en la sección crítica desde el 0% al 100%. En la Figura 45, las tareas consumen el 100% de todo el WCET, en la Figura 46 las

tareas consumen el 60% del WCET y en la Figura 47 las tareas solamente consumen el 20% del WCET. La mejora obtenida por nuestro algoritmo varia del 15% si todas las tareas consumen el 100% del WCET al 48% si las tareas consumen el 60% del WCET y el 75% si las tareas consumen únicamente el 20% del WCET.

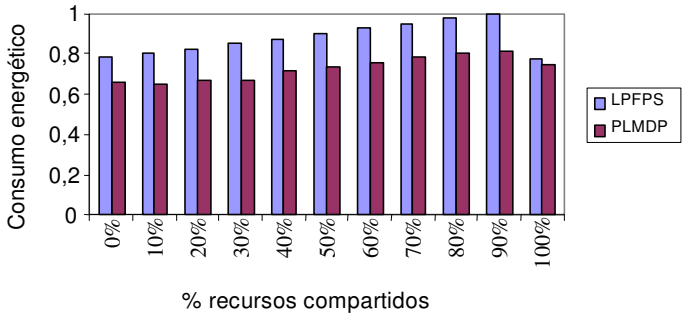

Figura 45: Consumo energético normalizado variando el tiempo ejecutado en sección crítica si la tareas consumen el 100% del WCET.

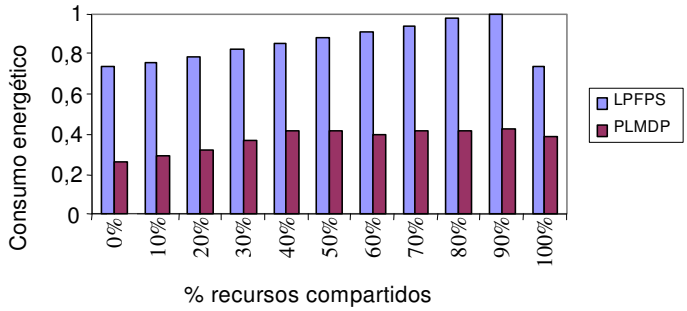

Figura 46: Consumo energético normalizado variando el tiempo ejecutado en sección crítica si la tareas consumen el 60% del WCET.

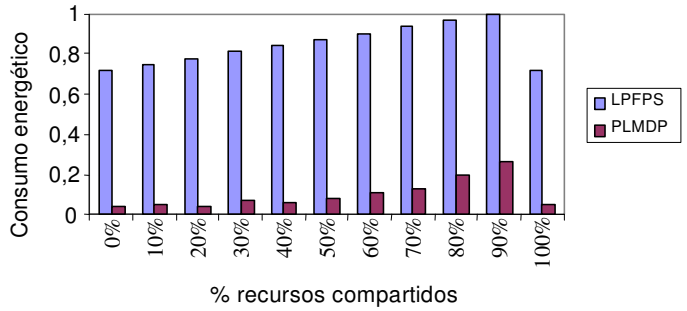

Figura 47: Consumo energético normalizado variando el tiempo ejecutado en sección crítica si la tareas consumen el 20% del WCET.

En las figuras anteriores se puede observar un descenso significativo de la energía consumida cuando la ejecución de la tarea en la sección crítica es del 100%. A continuación se muestra un análisis individual de está situación particular (Figura 48, Figura 49).

Supongamos que se dispone de tres tareas τ_1, τ_2 y τ_3. La prioridad de las tareas es de 3,2 y 1 respectivamente (a menor número mayor prioridad). τ_1 y τ_2 acceden al mismo recurso R_1, por lo tanto, el techo de prioridad es la prioridad de τ_2. El instante de promoción de la tarea τ_1 es menor que le instante de promoción de la tarea τ_2, y es menor que el instante de promoción de la tarea τ_3. En la cola LRQ el orden de las tareas es τ_1, τ_2 y τ_3.

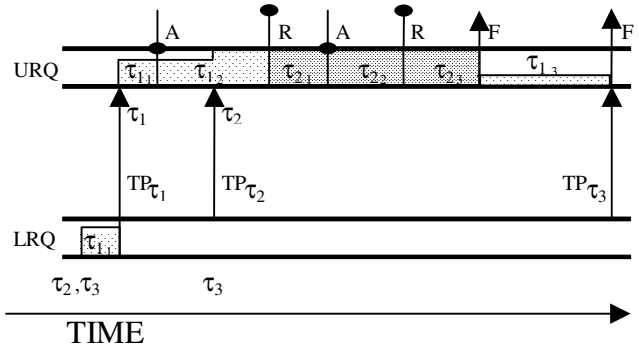

Figura 48: Las tareas no consumen todo el WCET en la sección crítica.

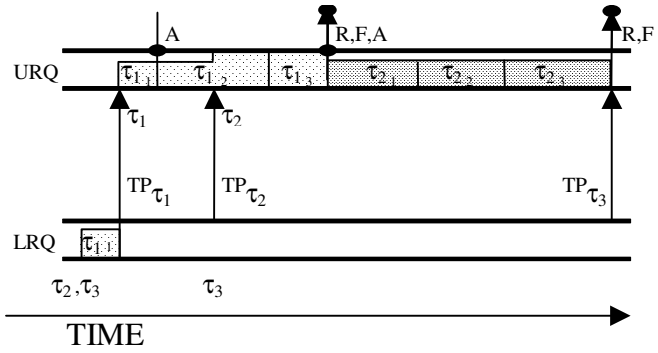

Figura 49: Las tareas consumen el WCET en la sección crítica.

En la Figura 48, inicialmente la tarea τ_1 se está ejecutando en la LRQ, en el instante de promoción TPτ_1, la tarea τ_1 promociona, y continua su ejecución. En el instante de promoción TPτ_2, la tarea τ_2 promociona a la URQ. Como τ_1 se halla en sección crítica, su prioridad es la del techo del recurso, por lo tanto τ_2 no puede expulsarla del procesador, continuando la ejecución de τ_1 hasta de τ_1 abandone la sección

crítica y recupere su prioridad base. Cuando τ_1 libera el recurso, τ_2 expulsa a τ_1, que se encola en la URQ. En este momento existen dos tareas en la URQ, por la que no se puede reducir la velocidad de ejecución. Cuando finaliza τ_2, τ_1 puede ejecutarse a velocidad reducida, ahorrando energía, pero el tiempo de ejecución que le queda es pequeño, reduce mucho la velocidad, pero ha obligado al procesador a ejecutar τ_2 a máxima velocidad. En la Figura 49, la tarea τ_1 finaliza al liberar el recurso, por lo tanto τ_2 esta sola en la URQ y puede por tanto reducir la velocidad de ejecución durante toda su ejecución.

Finalmente, en la Figura 50 se representa el porcentaje de ahorro energético en las tres posibles combinaciones analizadas cuando el WCET consumido total varia des del 10% al 100%. Cada punto del gráfico representa la suma de la energía consumida variando el porcentaje de WCET consumido en la sección crítica desde el 0% al 100%. Se puede observar que en el caso cs-ncs-cs, en que el bloqueo causado por la sección crítica es la mitad del bloqueo causado por cs-ncs y por ncs-cs, y ocurre dos veces durante la ejecución de una instancia de la tarea, el algoritmo PLMDPR mejora al algoritmo LPFPSE entre el 23% y el 62%. En el caso de cs-ncs la mejora obtenida varia entre el 21% y el 90%. Una razón posible del distinto comportamiento entre los casos es que en el cs-ncs-cs, la tarea que abandona la sección crítica aun no ha finalizado, obligando a cualquier otra tarea de mayor prioridad que se ejecute a ejecutarse a máxima velocidad. Si existe una única sección crítica, los resultados son equivalentes, ya sea ejecutando primero la sección crítica o primero la no sección crítica finalizando la tarea con la ejecución en sección crítica.

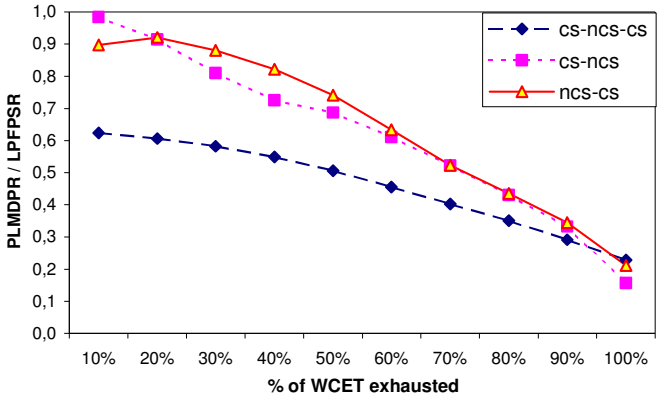

Figura 50: Ratio PLMDPR/LPFPSR de energía consumida en función del tiempo ejecutado en sección crítica para distintas ubicaciones de ésta.

3.4 Conclusiones

Se ha presentado una extensión del planificador Power Low Modified Dual Priority que permite a los sistemas de tiempo real usar recursos compartidos sin comprometer el cumplimiento de los plazos temporales consiguiendo un menor consumo energético. Se ha demostrado que esta aproximación continua mejorando al ahorro energético obtenido en media por el LPFPR. En el caso del conjunto de tareas del CNC, el rango de mejoría va desde el 16% al 98% dependiendo del uso de los recursos compartidos que realicen las tareas y el porcentaje de consumo del WCET que se produzca.

El algoritmo presentado no incrementa la complejidad del LPFPR con lo que puede ser implementado fácilmente en la gran mayoría de núcleos de sistemas operativos, como demuestra por una parte que el protocolo techo de prioridad inmediato sea considerado estándar POSIX, y por otra que el cálculo de la velocidad requiere únicamente conocer el instante de llegada de las tareas, el instante de promoción y el plazo temporal de las tareas, requerimientos necesarios para la implementación del algoritmo de *Dual Priority* [86].

Capítulo 4

PLAZOS GLOBALES Y PRECEDENCIAS ENTRE TAREAS

En este capítulo se planificará un sistema de tiempo real estricto en el que existan restricciones temporales globales que afecten a una secuencia de tareas, que han de ejecutarse en el mismo o en distintos procesadores siguiendo una ordenación temporal. La planificación de la secuencia de tareas deberá garantizar todas las restricciones temporales de estas, que pueden ser individuales a las tareas o globales al conjunto, con el mínimo coste energético posible.

4.1 Introducción

El objetivo principal de este capítulo es el estudio del ahorro energético que se puede obtener en sistemas distribuidos de tiempo real estricto en los que existen plazos temporales globales que implican la existencia de precedencias temporales entre las tareas que deben ejecutarse en el mismo o distinto procesador. Para lograr este objetivo, se realizará una ampliación de los algoritmos de planificación utilizados en el capitulo anterior.

El bajo coste y la fiabilidad de las redes de interconexión, existentes en la actualidad, permiten la conexión de múltiples dispositivos y controladores en un único sistema de una manera predecible. Este avance ha contribuido al incremento que se ha producido en el uso de las arquitecturas distribuidas en sistemas de tiempo real [27]. El uso de estos sistemas distribuidos, permite la implementación de sistemas de tiempo real más complejos. Las tareas ya no son independientes sino que cooperaran entre si para realizar una determinada función, necesitando la definición de una ordenación temporal para su correcta ejecución. Es decir, las tareas se diseñarán siguiendo una cadena de precedencias que debe respetarse durante su ejecución. De manera que primero se tiene que ejecutar la primera tarea de la cadena de precedencias, después la segunda, la tercera y así hasta la última tarea de la cadena. Una vez definida esta ordenación temporal, se puede hablar de tareas precedentes en la cadena y de tareas sucesoras. A nivel general, una cadena de precedencias puede tratarse como si de una única tarea se tratase, teniendo un periodo de activación global (Tg), un plazo temporal global (Dg) y un peor tiempo de ejecución global (WCETg) que será igual al sumatorio de los peores tiempos de ejecución de las tareas que forman parte de la cadena de precedencias que se define. El plazo temporal global estará definido por el tiempo comprendido desde que se genera el evento que provoca la activación de la primera tarea hasta que finaliza la ejecución de la última tarea de la cadena. En nuestro caso, se considerará que el evento que activa la primera tarea de la cadena es periódico con periodo igual a Tg.

La arquitectura del sistema distribuido que se tratará estará formada por varios procesadores unidos por una red de interconexión de tiempo real. Para limitar la casuística, no se analizarán los posibles casos de las migraciones entre los procesadores, considerando que en tiempo de diseño de la aplicación, se ha definido

el conjunto de tareas que se ejecutarán en cada uno de los procesadores disponibles. Esto implica que no se distinguirán entre procesadores homogéneos (con iguales características físicas) y procesadores heterogéneos (con características físicas distintas, pudiendo variar el tiempo de ejecución de las tareas según el procesador que la ejecute). Sin embargo, el hecho de tener variaciones en el tiempo de ejecución de las tareas debido a la heterogeneidad de los procesadores, se trata considerando como el WCET de la tarea, el peor tiempo de finalización de entre todos los posibles procesadores, y en tiempo de ejecución, las tareas pueden usar un porcentaje de este tiempo máximo. La red de interconexión se definirá con un WCETx (tiempo máximo de circulación del mensaje por la red), un plazo temporal Dx (plazo temporal de envío del mensaje a través de la red), y por un Tax (instante de activación del mensaje).

En una aplicación distribuida de tiempo real, se dispondrá de un conjunto de tareas independientes con plazos temporales individuales, y un conjunto de cadenas de precedencia con periodos y plazos temporales globales que se comunicaran a través de la red de interconexión. Las tareas independientes están caracterizadas por el procesador en el que deben ejecutarse (P), el periodo (T), el plazo temporal (D) y el peor tiempo de ejecución de la tarea (WCET). Las cadenas de precedencia están definidas por la secuencia de tareas que la forman, junto con el WCET de cada tarea, el periodo de activación de la primera tarea de la cadena (Tg) y el plazo temporal global (Dg). Al finalizar cualquier tarea que posea una tarea sucesora en una cadena de precedencia, se activará la tarea sucesora mediante un mensaje. Si la tarea sucesora se halla en un procesador distinto de la predecesora, el mensaje circulará por la red de interconexión, en caso contrario, se supondrá que el tiempo de circulación del mensaje es despreciable.

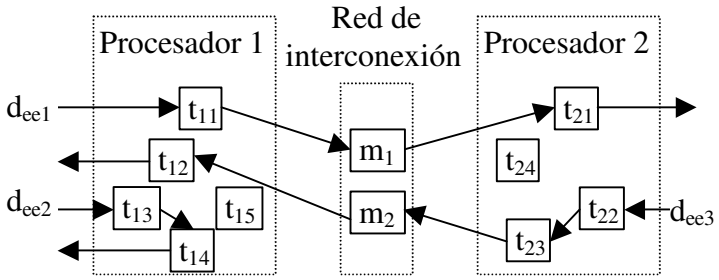

Figura 51: Esquema de un sistema distribuido de tiempo real.

La planificación de sistemas distribuidos consiste en planificar las tareas de cada procesador junto con los mensajes de las redes de interconexión, asegurando el cumplimiento de los plazos temporales de todas las tareas, la precedencia entre tareas y los plazos globales de las cadenas. La manera más general de garantizar los plazos temporales globales, consiste en la asignación de un plazo temporal local a cada una de las tareas del sistema, tal que la suma de todos los plazos temporales de la cadena de precedencias sea menor o igual al plazo temporal global. Para garantizar la precedencia entre las tareas a cada tarea perteneciente a la cadena de precedencias se le asigne un desfase inicial igual al plazo temporal de la tarea que la precede, esto obliga a que la tarea sucesora en la cadena empiece su ejecución cuando es seguro que la tarea predecesora ha finalizado, ésta tiene que cumplir su plazo temporal. Con esta repartición del plazo temporal global se consigue que el estudio del sistema distribuido sea equivalente al estudio de una conjunto de procesadores independientes con tareas independientes entre si [83], [66]. Para utilizar este modelo, es necesario: la asignación de un plazo temporal mínimo para todas las tarea de la cadena de precedencia que viene determinado por el peor tiempo de finalización (WCRT) de la tarea y la generación de desfases iniciales del instante de inicio de las tareas sucesoras tales que se garantice la precedencia entre las tareas, este desfase inicial será igual al plazo temporal de la tarea predecesora. Una vez realizadas estas asignaciones, se podrá aplicar las técnicas de planificación con ahorro energético ampliamente estudiadas para sistemas monoprocesadores [87].

Sin embargo, el uso de los planificadores para sistemas monoprocesador en la planificación de sistemas distribuidos tiene como principal problema que, en sistemas con una carga de procesador teórica alta, es imposible encontrar una distribución estática del plazo temporal global que satisfaga las restricciones temporales del sistema distribuido. Por ejemplo, si se considera una restricción temporal de un sistema distribuido con una utilización global del procesador alta (80%), y una distribución de periodos de las tareas muy heterogénea, la determinación del WCRT de las tareas periódicas realizada por cualquier planificador de prioridad estática será muy pesimista debido a que estos normalmente no tienen en cuenta el desfase entre las tareas, y por tanto contaran con interferencias entre las tareas pertenecientes a las cadenas de precedencia que realmente no se dará nunca. Una aproximación a la solución de este problema para planificadores

dinámicos de prioridades estáticas fue propuesto por el grupo de K. Tindell en [15], y mejorado por el grupo de J.C. Palencia en [66], determinando el instante crítico para un sistema distribuido teniendo en cuenta la interferencia real de las tareas en las cadenas de precedencias. Sin embargo, esta aproximación no es del todo exacta representando únicamente una cota superior del peor tiempo de finalización de las tareas periódicas, que descarta como planificables algunos sistemas distribuidos cuando la carga es alta. Otra aproximación similar para planificadores dinámicos de prioridades dinámicas es la de M. Spuri en [78] y la de J.C. Palencia y M.González-Harbour en [65]. La idea principal de M. Spuri es adaptar el análisis propuesto por K. Tindell al planificador EDF. Por otra parte, J.C. Palencia y M.González-Harbour consiguen una importante mejora considerando que los desplazamientos son dinámicos resultando una cota superior mejor para el análisis del peor caso.

La diferencia principal de nuestro trabajo con los anteriormente mencionados, es que nosotros no buscaremos cotas para la planificación de las tareas, sino que lo que nos interesa es el ahorro energético sin comprometer en ningún caso los plazos temporales de las tareas. De manera que el conjunto de tareas que nosotros trataremos tiene que ser planificable mediante simulación durante todo el hiperperiodo asumiendo que las tareas consumen todo su WCET. En caso que se pierda algún plazo temporal durante la simulación inicial, se descartará el conjunto de tareas. Otra posible solución al análisis de la planificabilidad para este escenario se podría haber realizado antes de la ejecución del sistema usando el algoritmo de J.C. Palencia y M.González-Harbour descrito en [65] si la restricción temporal estuviese localmente distribuida, sin embargo en el momento del presente trabajo desconocíamos dicha fuente.

Proponemos un nuevo escenario en el que el plazo temporal global no está distribuido entre las tareas que forman la cadena de precedencias sino que se mantiene global hasta el momento de ejecución del sistema. Las tareas se activarán individualmente debido a la finalización de la tarea precedente o debido al periodo de activación. Al finalizar una tarea que posea una tarea sucesora, enviará un mensaje que provocará la activación de esa tarea sucesora. El ahorro energético que se conseguirá será, como en los capítulos precedentes, debido a la aplicación de las técnicas del DVS en la planificación de las tareas, o al hecho de detener un procesador cuando no existan tareas para ser ejecutadas.

4.2 Metodología y desarrollo: Análisis del plazo global.

En este punto, se analizará el tratamiento del plazo global en la planificación de un sistema distribuido con plazos temporales que abarcan diversas tareas que se hallan ubicadas en diversos procesadores. La planificación debe asegurar el cumplimiento de todos los requisitos temporales que consistirán, no solo en asegurar la secuencialidad temporal de las tareas pertenecientes a las cadenas de precedencia, sino que además se han de cumplir los plazos temporales globales e individuales de todas las tareas del sistema. La solución que se propone consistirá en no distribuir todo el plazo global entre las diversas tareas, sino que asegurando que cada tarea tenga el mínimo tiempo posible, use el resto únicamente cuando sea preciso, dinámicamente en función de la carga instantánea del sistema. Esta solución nos conducirá a una opción más flexible que una repartición estática, pudiéndose dar el caso, de que en una activación dada, una tarea concreta de una cadena de precedencias pueda terminar muy pronto, por no haber, en ese momento, otras tareas activas en el procesador, y que la tarea que la suceda finalice muy tarde por haberse activado en ese instante otras tareas de mayor prioridad, y en cambio en la siguiente activación de la cadena ocurra justamente lo contrario.

A continuación, se analizaran las distintas maneras de dividir el plazo temporal global entre varias tareas $<\tau_1,..,\tau_n>$ que se ejecutan en el mismo o en distinto procesador y las repercusiones que se obtienen según la división de plazo que se realice:

a) Repartiendo la totalidad del plazo global entre todas las tareas de la cadena. τ_1 se activará en el periodo la cadena. Su tarea sucesora (τ_2) se activará en el plazo temporal de τ_1 asegurando así la precedencia entre ambas tareas, y así sucesivamente. Este caso se puede analizar estáticamente usando alguna prueba de planificabilidad, asegurando así, la planificabilidad del sistema. Después de repartir el plazo global, nos hallamos ante un problema de planificación estática con desplazamiento inicial (*offset*) de las tareas sucesoras igual al plazo temporal de la tarea predecesora.

   Ambas tareas pueden ejecutarse a velocidad reducida dado que la segunda tarea se activará cuando la primera haya finalizado, pudiendo, según la repartición del

plazo que se haya establecido reducir velocidad de ejecución, obteniendo reducción energética. Sin embargo, si dos tareas consecutivas en la cadena de precedencias se hallan en procesadores distintos, se debe tener en cuenta que el procesador puede estar libre a la espera de las activaciones de las tareas sucesoras.

Cada procesador se puede analizar individualmente, ya que las precedencias se aseguran en el momento de repartir el plazo global, fijando un desplazamiento inicial de las tareas sucesoras igual al plazo temporal de su tarea predecesora. El periodo de activación de todas las tareas será igual al periodo de la cadena.

b) No repartiendo la totalidad del plazo global entre todas las tareas de la cadena. Si no se fija un plazo local a las tareas, se permitirá que estas usen el necesario según la carga existente en el sistema a cada momento, otorgando flexibilidad al sistema.

   La primera tarea de la cadena (τ_1) se activará en el periodo la cadena mientras que las tareas sucesoras no tienen instante de activación sino que se activan con la finalización de sus tareas predecesoras. Esto implica que la tarea predecesora no puede ejecutarse a velocidad reducida, puesto que no tiene previsto un instante de finalización máximo distinto del plazo temporal global, instante en el que deben haber finalizado todas las tareas que forman parte de la cadena de precedencias.

   Esta política es la más flexible, sin embargo a nivel energético dará poco ahorro puesto que únicamente se podrá reducir la velocidad de ejecución de la última tarea de la cadena de precedencias y de las tareas independientes. Usando esta política, es muy difícil la asignación automática de prioridades a las tareas, puesto que como que el plazo global acostumbra a ser grande, las tareas de la cadena de precedencia tendrán poca prioridad.

c) Repartiendo una parte del plazo global entre todas las tareas de la cadena. A cada tarea se le asigna el mínimo plazo que asegure la planificabilidad del sistema, y se usa el resto del plazo según las necesidades instantáneas de este. Este caso es parecido al del punto a), pero con el plazo temporal de las tareas variable.

Si se usa un algoritmo de planificación con prioridades estáticas, el hecho de disponer de un plazo mayor puede modificar la prioridad de la tarea según la asignación de prioridades que se utilice. Sin embargo, al ser las prioridades estáticas cuando se ha decidido el orden, las tareas se ejecutaran siempre en este orden fijado. Disponer de un plazo mayor al previsto, solo será útil en el momento en que la tarea pueda reducir la velocidad de ejecución ahorrando energía. Esta división es la que se realiza en el artículo [55]. El hecho de disponer de prioridades estáticas permite que, antes de iniciar el sistema, se pueda asegurar su planificabilidad ejecutándose a máxima velocidad.

Si el planificador es de prioridades dinámicas, el orden de ejecución de las tareas será variable dependiendo de la asignación del plazo que se realice. Al reducir la velocidad de ejecución de las tareas debe tenerse en cuenta que la utilización del procesador no supere el 100%, de lo contrario se tendría un sistema que no seria planificable. Para asegurar que esto no ocurre, se utilizará un planificador con servidor de tiempo disponible. Este tiempo se usará para ralentizar la ejecución de las tareas [87], ahorrando energía.

Analizadas las distintas alternativas, la solución que se propone en esta tesis, consiste en no distribuir la totalidad del plazo temporal global, permitiendo una cierta flexibilidad en la ejecución de las tareas. Para realizar la planificación del sistema distribuido, se dispondrá de un planificador local en cada procesador. Las activaciones de la primera tarea de cada cadena de precedencias serán periódicas con un periodo igual al de la cadena, mientras que las activaciones de las tareas sucesoras, serán dependientes de la finalización de la tarea que la precede en la cadena. Cuando una tarea finalice su ejecución, generará un mensaje que activará su tarea sucesora.

Cuando en el conjunto de tareas existen precedencias se pueden dar varias opciones:

- La última tarea lista para ser ejecutada no dispone de tarea sucesora. En este caso, se puede reducir la velocidad de ejecución para finalizar antes de la activación de la siguiente tarea o antes del cumplimiento del plazo temporal.
- La última tarea dispone de tarea sucesora. En este caso, existen dos opciones, todas la tarea sucesora se ejecute en el propio procesador o en otro procesador distinto.

- **Mismo procesador**: en este caso, se podría llegar a reducir la velocidad de ejecución de ambas tareas hasta el cumplimiento de su plazo o hasta la activación de otra tarea. O bien se podría ejecutar la tarea predecesora a máxima velocidad y dejar la posible reducción de velocidad para la tarea sucesora.
- **Distinto procesador**: en este caso, la reducción de la tarea en curso (finalización posterior), puede provocar que el procesador en el que se ejecute alguna tarea sucesora quede libre a la espera de l'activación de esta. Esto implica, que la interferencia que produce esta tarea sobre las otras tareas del sistema seria mayor que si la tarea se hubiera activado a la finalización de la tarea predecesora ejecutada a máxima velocidad.

Aunque como se ha visto en el capítulo 2, la mejor opción seria ejecutar ambas tareas a velocidad reducida, el algoritmo implementado reduce únicamente la velocidad de la última tarea de la cadena, reduciendo de este modo la complejidad de la solución y unificando los casos en que la tarea sucesora pertenezca al mismo o distinto procesador.

4.2.1 Adecuación de los algoritmos de planificación.

A continuación se verá un ejemplo para comprobar como seria la planificación con los distintos algoritmos estudiados en los capítulos anteriores para tratar el problema de las precedencias y de los plazos globales.

Supongamos que se dispone de los siguientes conjuntos ejemplos:

- Tabla 16: En el conjunto de tareas existe un único plazo temporal global.
- Tabla 17: En el conjunto de tareas existen dos plazos temporales globales.
- Observemos que en el sistema de la Tabla 16 el plazo temporal global no es planificable directamente, puesto que si se suma el tiempo de finalización de la tarea t_{13} con el de la tarea τ_{23} se obtienen 43 unidades de tiempo, que es superior al plazo temporal disponible (36 unidades). Sin embargo, esta situación es demasiado pesimista, ya que la tarea τ_{23} únicamente debe esperar a que finalice la tarea τ_{13}. Esta tarea no siempre finalizará usando su tiempo peor de finalización, y aunque ésta lo haga, la tarea τ_{23} puede que no reciba

toda la interferencia posible de las tareas de mayor prioridad, finalizando siempre antes que el peor tiempo de finalización.

Procesador	Tarea	Periodo	Plazo Temporal	WCET	WCRT independiente
Procesador 1	τ_{11}	10	10	4	4
	τ_{12}	18	18	4	8
	τ_{13}	36	36	8	28
Procesador 2	τ_{21}	10	10	4	4
	τ_{22}	18	18	4	8
	τ_{23}	36	36	3	15
Cadena de Precedencias	Tareas	Periodo	Plazo temporal	Tiempo envío del mensaje	WCRT de la cadena de precedencias
Dee_1	τ_{13}	36	36	1	36
	τ_{23}				

Tabla 16: "Benchmark" con una cadena de precedencias para la evaluación del ahorro energético con precedencias entre las tareas. WCET=peor tiempo de ejecución, WCRT=peor tiempo de finalización suponiendo procesadores independientes.

Para realizar el cálculo exacto del peor tiempo de finalización de las cadenas de precedencia debe calcularse el tiempo de finalización de cada una de las instancias de esas tareas en todo el hiperperiodo. Este cálculo puede hacerse off-line, para asegurar que el conjunto de tareas que se propone es planificable. Estudiando el peor tiempo de finalización obtenido por las distintas instancias de la cadena de precedencias en un hiperperiodo, se obtiene que el peor tiempo de finalización es de 36 unidades de tiempo, lo que implica que el sistema es planificable. El resultado del peor tiempo de finalización podría guardarse en un vector del planificador para realizar una previsión fiable del tiempo de llegada de los mensajes de las tareas predecesoras, sin embargo, si el hiperperiodo es muy grande (acostumbra a ocurrir en tareas no armónicas), el uso de memoria seria grande, cosa que se quiere evitar.

En nuestros algoritmos se realizará una previsión muy optimista del tiempo de finalización de las tareas de la cadena de precedencias, previendo que tendrán una única interferencia de las tareas de mayor prioridad. Este hecho provocará que si la tarea tiene una interferencia mayor de la prevista, no se pueda reducir la velocidad de ejecución de tareas que serian factibles de ser reducidas, gastando mayor cantidad de energía y dejando libre al procesador.

En la Figura 52 se puede observar la ejecución de las tareas del ejemplo (Tabla 16) suponiendo una planificación de prioridades fijas con asignación Rate Monotonic a velocidad máxima. En esta ejecución se observa que la primera instancia de la tarea τ_{13} se ejecuta con una interferencia máxima (instante crítico) finalizando en t=28. La

instancia de la tarea τ_{23} que la sucede no tiene la máxima interferencia finalizando en el instante 36, cumpliendo así el plazo temporal global. La tarea τ_{23} tiene su instante crítico en la tercera instancia con un tiempo de finalización de 14 unidades de tiempo, ya que a su llegada, una tarea de mayor prioridad ha ejecutado una unidad de cálculo. Pero en este caso su instancia precedente ha finalizado en 20 unidades de tiempo, no ha hallado la interferencia máxima. La tercera instancia, ha finalizado en 35 unidades de tiempo. El instante crítico de las tareas que forman parte de una cadena de precedencias, normalmente, no coincidirán, por lo que es demasiado pesimista considerar que se da esta circunstancia. En el caso concreto del ejemplo, se observa que no hay perdidas de plazo, y por lo tanto, el sistema es planificable, tal y como se había supuesto inicialmente. Con esta planificación existen 28 unidades libres en el primer procesador, y 53 en el segundo.

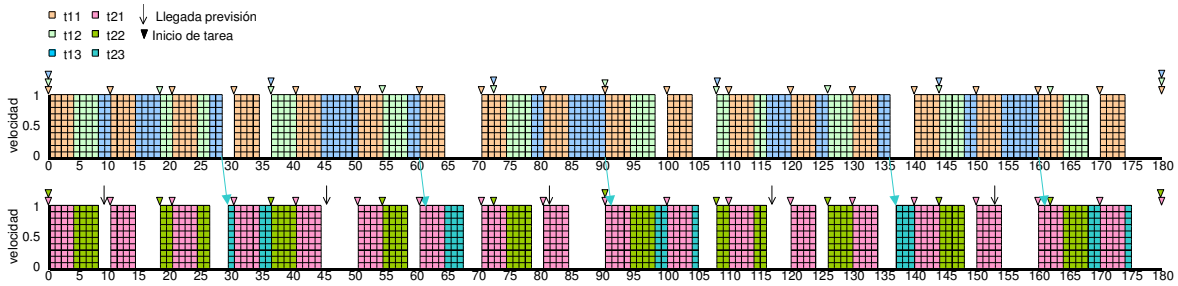

Figura 52: Ejecución a máxima velocidad de los procesos de la Tabla 16 en un sistema distribuido con un único plazo temporal global usando offset dinámico.

En el siguiente ejemplo (Tabla 17), se ha añadido otro plazo global en las tareas del conjunto anterior, de manera que existirán, en el sistema, dos cadenas de precedencia o plazos globales.

Procesador	Tarea	Periodo	Plazo Temporal	WCET	WCRT independiente
Procesador 1	τ_{11}	10	10	4	4
	τ_{12}	18	18	4	8
	τ_{13}	36	36	8	28
Procesador 2	τ_{21}	10	10	4	4
	τ_{22}	18	18	4	8
	τ_{23}	36	36	3	15
Cadena de Precedencias	Tareas	Periodo	Plazo temporal	Tiempo envío del mensaje	WCRT de la cadena de precedencias
Dee_2	τ_{12} / τ_{22}	18	18	1	18
Dee_1	τ_{13} / τ_{23}	36	36	1	36

Tabla 17: Conjunto de tareas con dos cadenas de precedencias para la evaluación del ahorro energético.

En este sistema se puede observar, que la suma de los peores tiempos de finalización de las tareas del Dee_2 es 16, por lo tanto el sistema sería planificable repartiendo de manera estática el plazo global a cada una de las tareas de la cadena de precedencias. En el caso de distribución estática, la tarea τ_{22} tiene un desplazamiento inicial en su ejecución igual al plazo de la tarea τ_{12} no pudiendo empezar hasta el instante 8 si la repartición es estática o hasta el instante 9 en caso de activación por mensaje. Finalizando esta tarea en el instante 16 o 17, respectivamente, siendo por tanto planificable. Sin embargo, la cadena Dee_1 tiene el mismo problema que en el ejemplo de la Tabla 16, no siendo planificable si se supone una repartición estática.

En la Figura 53 se presenta la ejecución del sistema de la Tabla 17 en el caso de activación de la tarea sucesora mediante mensaje. En el segundo procesador, se puede observar que en el instante 4, las tarea τ_{21} ha finalizado, pero la tarea τ_{22} no se haya activa, dejando, por tanto, libre el procesador. La interferencia que tiene la primera instancia de la tarea τ_{13} provoca que esta finalice, en t=28 (ha hallado su máxima interferencia). En este ejemplo, a diferencia del anterior, la segunda instancia de la tarea τ_{22} no ha finalizado su ejecución, aumentando así, el tiempo de finalización de la primera instancia de la tarea τ_{23}, provocando una perdida de plazo temporal. En este caso, el sistema con la asignación de prioridades *Rate Monotonic*, no es planificable.

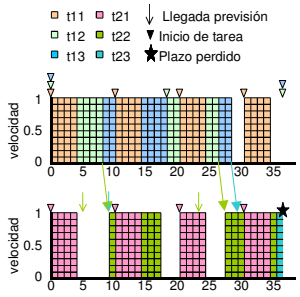

Figura 53: Ejecución a máxima velocidad de los procesos de la Tabla 17 en un sistema distribuido con un único plazo temporal global usando offset dinámico.

A continuación se estudiarán estos dos ejemplos aplicados a los algoritmos propuestos en el capítulo anterior, teniendo en cuenta que las únicas tareas que reducirán la velocidad del procesador serán las que no tengan tareas sucesoras, y que se estimará la llegada de las tareas sucesoras (previsión llegada), de manera que a partir de su llegada estimada ya no será posible disminuir la velocidad del procesador

aunque la tarea sea la única del sistema. A si mismo, las tareas sucesoras se activarán por mensaje al finalizar su tarea precedente.

Low Power Fixed Priority Scheduling Algorithm.

Este algoritmo, como se ha visto en el Capítulo 2, se basa en ahorrar energía reduciendo la velocidad de ejecución de la última tarea existente en la cola de preparados.

Las modificaciones realizadas en este planificador son las siguientes:

- No reducción de la velocidad del procesador si la última tarea activa tiene una tarea sucesora.
- Tener en cuenta la previsión de la llegada de una tarea sucesora. No pudiéndose reducir la velocidad aunque exista, únicamente, una tarea activa, cuando se prevea la activación de una tarea sucesora. La reducción de la velocidad de proceso, podría derivar en una perdida del plazo, ya que se producirá una activación de una tarea cuya llegada no estaba contemplada en el momento del cálculo de la velocidad del procesador.
- Envío de un mensaje de activación de las tareas sucesoras.

A continuación se analizará el resultado de la planificación LPFPS en los dos ejemplos analizados en el apartado anterior.

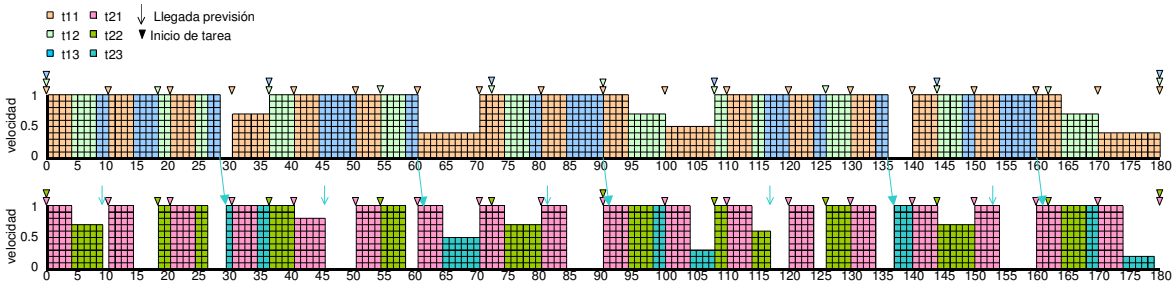

Figura 54: Ejecución usando la planificación LPFPS de los procesos de la Tabla 16 en un sistema distribuido con un único plazo temporal global usando offset dinámico.

En la Figura 54 se ha realizado el diagrama de ejecución del conjunto de tareas de la Tabla 16. En este caso se puede observar que el sistema es planificable, ya que se cumplen todos los plazos. El consumo energético obtenido frente a la planificación a máxima velocidad es del 90%. Dejando 42 unidades de tiempo libres.

A continuación, se remarcan los instantes en los que se aplica la modificación del algoritmo de planificación y la consecuencias que ello comporta:

- t=5. En el segundo procesador, la tarea τ_{22} empieza su ejecución, es este instante el algoritmo de planificación calcula la velocidad de ejecución de esa tarea. Es la única tarea activa en el sistema, por lo que se puede alargar su ejecución reduciendo la velocidad del procesador. La siguiente tarea que se activará lo hará en el instante 10, pero se tiene prevista la llegada de la tarea τ_{23} en el instante 9, por lo que solo se tiene disponibilidad temporal hasta el instante 9, mínimo entre la llegada de una tarea y la previsión de llegada. Este hecho comporta que el procesador quede una unidad de tiempo libre.
- t=26. En el primer procesador, la tarea τ_{13} reanuda su ejecución. En el momento de calcular la velocidad de ejecución, se ve que aun siendo la única tarea activa en el momento, tiene una tarea sucesora, por lo tanto, se ejecutará a velocidad de procesador máxima. Este hecho comportará que al procesador le queden dos unidades de tiempo libre, hasta la activación de la tarea τ_{11}.
- t=28. En el primer procesador finaliza la ejecución la tarea τ_{13} que dispone de tarea sucesora, enviando un mensaje que provocará la activación en t=29 de la tarea τ_{23} en el segundo procesador.

En la Figura 55 se ha realizado el diagrama de ejecución del conjunto de tareas de la Tabla 17. En este caso se puede observar que el sistema no es planificable, ya que no se cumple el plazo global de la cadena de precedencias Dee_1.

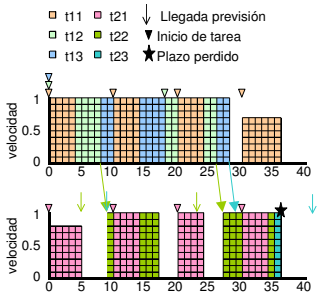

Figura 55: Ejecución usando la planificación LPFPS de los procesos de la Tabla 17 en un sistema distribuido con un único plazo temporal global usando offset dinámico.

A continuación, se remarcan los instantes en los que se aplica la modificación del algoritmo de planificación y la consecuencias que ello comporta:

- t=0. En el segundo procesador, la tarea τ_{21} empieza su ejecución, es este instante el algoritmo de planificación calcula la velocidad de ejecución de esa tarea. Es la única tarea activa en el sistema, por lo que se puede alargar su ejecución reduciendo la velocidad del procesador. Su plazo temporal es en el instante 10, pero se tiene prevista la llegada de la tarea τ_{22} en el instante 5, por lo que solo se tiene disponibilidad temporal asegurada hasta el instante 5, mínimo entre el plazo temporal y la previsión de llegada de un mensaje activador de una tarea sucesora. El planificador alargará la ejecución de la tarea hasta el instante 5. Este hecho comportará que el procesador quede nueve unidades de tiempo libre.
- t=8. En el primer procesador, la tarea τ_{12} finaliza su ejecución, enviando un mensaje que llegará en el instante t=9 al segundo procesador, activando la tarea τ_{13}. Que se ejecutará una unidad hasta la activación de la tarea τ_{12}. En el momento de calcular la velocidad de ejecución, se ve que aun siendo la única tarea activa en el momento, no se puede ejecutar a velocidad reducida debido a que se espera la llegada de la tarea τ_{23}. tiene una tarea sucesora, por lo tanto, se ejecuta a velocidad de procesador máxima.
- t=10. La activación de la tarea τ_{23} se apropia del procesador, por ser una tarea de mayor prioridad.
- t=28. Finaliza la ejecución de la tarea τ_{13} enviando un mensaje que activará la tarea τ_{23} que al ser la tarea de menor prioridad, debe esperar la finalización de las tareas de mayor prioridad en t=35, empieza la ejecución, pero en t=36 le llega el plazo temporal si haber finalizado la ejecución.

Low Power Earliest Deadline First Scheduling Algorithm.

Este algoritmo, como se ha visto en el Capítulo 2, consiste en ahorrar energía utilizando el *slack* estático hasta conseguir una utilización del procesador del 100%, pudiendo reducir la velocidad de ejecución de cualquier tarea.

Las modificaciones realizadas en este planificador son las siguientes:

- No reducción de la velocidad del procesador si la tarea que se ejecuta tiene una tarea sucesora.

- Servidor con periodo igual al de la tarea más frecuente y tiempo de cálculo el máximo posible que mantenga la utilización del procesador por debajo del 100%. Al calcular la capacidad del servidor, ya se ha tenido en cuenta la utilización que usaran las tareas pertenecientes a las cadenas de precedencias.
- Las tareas pertenecientes a una cadena de precedencias tendrán la siguiente asignación de plazo:

$$D_i = Dee_k - \sum_{\forall\ j \in succ_i} C_j - C_{mess} \quad \tau_i \in \Gamma_{Deek} \qquad \text{(ec. 17)}$$

  donde D_i es el plazo temporal de la tarea τ_i, Dee_k es el plazo global de la cadena de precedencias k y C_{mess} es el tiempo de circulación de los mensajes. Esta asignación implica que las tareas del inicio de la cadena dispondrán de un mayor plazo, por tanto menor prioridad. Mientras que las tareas del final el plazo si las iniciales lo han usado será menor, teniendo por lo tanto mayor prioridad.
- Envío de un mensaje de activación de las tareas sucesoras.

A continuación, se analizará el resultado de la planificación LPEDF usando los dos ejemplos analizados anteriormente.

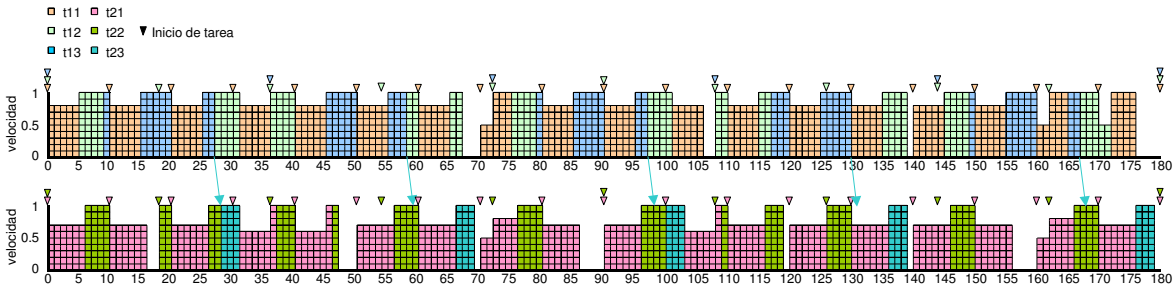

Figura 56: Ejecución usando la planificación LPEDF de los procesos de la Tabla 16 en un sistema distribuido con un único plazo temporal global usando offset dinámico.

En la Figura 56 se ha realizado el diagrama de ejecución del conjunto de tareas de la Tabla 16. En este caso se puede observar que el sistema es planificable. El consumo energético obtenido frente a la planificación a máxima velocidad es del 87%. Dejando 27 unidades de tiempo libres.

En este sistema, la capacidad del servidor es de 1 unidad de tiempo en el primer procesador con periodo cada 10 unidades de tiempo y de 2 unidades de tiempo en el segundo procesador, con periodo igual a 10.

- En el instante t=0. En el primer procesador, la tarea τ_{11} empieza su ejecución, en este instante el algoritmo de planificación calcula la velocidad de ejecución de esa tarea. La capacidad del servidor es 1, reduciendo la velocidad de ejecución de la tarea. En t=5 la capacidad del servidor esta vacía, por lo tanto, la tarea debe ejecutarse a velocidad máxima.
- t=10. En el primer procesador, se activa la tarea τ_{11} que expulsa a la tarea τ_{13}. En este instante se rellena la capacidad del servido, pudiendo la tarea τ_{11} reducir su velocidad de ejecución.
- t=27. En el primer procesador finaliza la tarea τ_{13} enviando un mensaje que activará al cabo de una unidad de tiempo a la tarea τ_{23}.
- t=29. En el segundo procesador se activa la tarea τ_{23} con un plazo de 36. Al ser la única tarea se empieza su ejecución sin disminuir su velocidad ya que pertenece a una cadena de precedencias. Al activarse τ_{21} no puede apoderarse del procesador ya que su plazo es 40 que es mayor al plazo de τ_{23}.

En la Figura 57 se ha realizado el diagrama de ejecución del conjunto de tareas de la Tabla 17. En este caso se puede observar que el sistema no es planificable, ya que no se cumple el plazo global de la cadena de precedencias Dee_1.

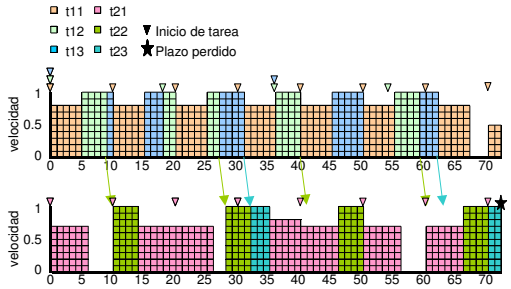

Figura 57: Ejecución usando la planificación LPEDF de los procesos de la Tabla 17 en un sistema distribuido con dos plazos temporales globales usando offset dinámico.

A continuación, se remarcan los instantes en los que se aplica la modificación del algoritmo de planificación y la consecuencias que ello comporta:

- En el instante t=0. En el segundo procesador, la tarea τ_{21} empieza su ejecución, en este instante el algoritmo de planificación calcula la velocidad

de ejecución de esa tarea. El servidor dispone de capacidad, por lo que se puede alargar su ejecución reduciendo la velocidad del procesador. La capacidad del servidor es 2, que es el tiempo que reduce su velocidad de ejecución, lo que comportará que el procesador quede cuatro unidades de tiempo libre.

- t=10. En el segundo procesador se activa la tarea t_{21} con un plazo de 20 y la tarea t_{22} con un plazo de 18. La tarea con mayor prioridad es la τ_{22} que no reduce su velocidad de ejecución por pertenecer a una cadena de precedencias.
- t=35. En el segundo procesador, empieza su ejecución la tarea τ_{21} con plazo igual a 40 y tiempo de cálculo de 4. El servidor tiene capacidad de 2 unidades, sin embargo si la tarea solo puede usar una unidad ya que si usara las dos perdería el plazo. La unidad de capacidad sin usar se pierde.
- t=60. En el segundo procesador se activa la tarea τ_{21} con plazo 70, la tarea τ_{22} con plazo 72. Por lo que la tarea t_{21} empieza si ejecución. En este momento la capacidad del servidor es 2 con lo que la tarea se ejecuta a velocidad reducida consumiendo la totalidad de ella.
- t=66. En el segundo procesador empieza la ejecución de la tarea τ_{22} que finaliza en t=70, instante en que empieza la ejecución la tarea τ_{23} con plazo 72, que empieza su ejecución pero no la finaliza dentro del plazo temporal. El resultando es que el sistema es no planificable.

Power Low Modified Dual Priority Scheduling Algorithm.

Este algoritmo, como se ha visto en el Capítulo 2, se basa en ahorrar energía reduciendo la velocidad de ejecución cuando exista una o ninguna tarea en la cola de mayor prioridad.

Las modificaciones realizadas en este planificador son las siguientes:

- Las tareas que pertenecen a una cadena de precedencias entraran directamente a la URQ, en lugar de entrar en la LRQ y esperar que le llegue el instante de promoción.
- No reducción de la velocidad del procesador si la última tarea activa tiene una tarea sucesora.

- Tener en cuenta la previsión de la llegada de una tarea sucesora. No pudiéndose reducir la velocidad aunque exista, únicamente, una tarea activa, cuando se prevea la activación de una tarea sucesora. La reducción de la velocidad de proceso, podría derivar en una perdida del plazo, ya que se producirá una activación de una tarea cuya llegada no estaba contemplada en el momento del cálculo de la velocidad del procesador.
- Envío de un mensaje de activación de las tareas sucesoras.

A continuación se analizará el resultado de la planificación PLMDP en los dos ejemplos analizados en el apartado anterior.

En la Figura 58 se ha realizado el diagrama de ejecución del conjunto de tareas de la Tabla 16. En este caso se puede observar que el sistema es planificable. El consumo energético obtenido frente a la planificación a máxima velocidad es del 79%, y en el grafico de planificación se puede observar que no existen espacios libres del procesador.

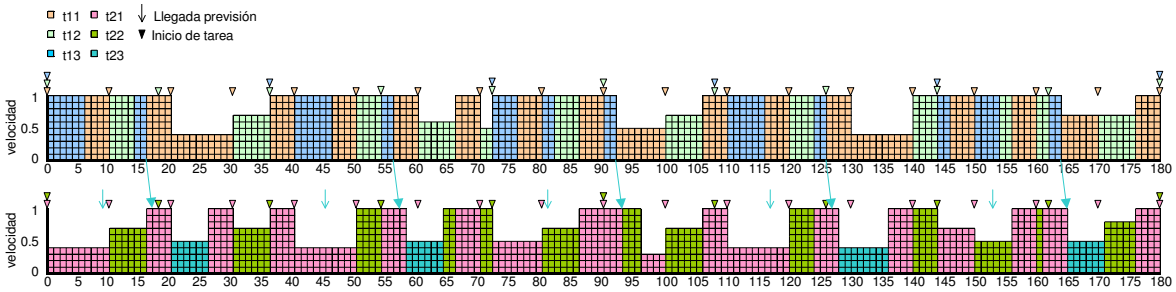

Figura 58: Ejecución usando la planificación PLMDP de los procesos de la Tabla 16 en un sistema distribuido con un único plazo temporal global usando offset dinámico.

A continuación, se remarcan los instantes en los que se aplica la modificación del algoritmo de planificación y la consecuencias que ello comporta:

- En el instante t=0. En el primer procesador, la tarea τ_{13} empieza su ejecución, como que esta tarea pertenece a la cadena de precedencias Dee_1, entra en la URQ ejecutándose a máxima velocidad.
- t=0. En el segundo procesador, no existen tareas en la URQ, por lo que la tarea τ_{21} puede ejecutarse a velocidad reducida hasta la previsión de la llegada de un mensaje indicando que una tarea que pertenece a una cadena de precedencias ha finalizado su ejecución.

- t=6. En el primer procesador, la tarea τ_{11} promociona que expulsa a la tarea τ_{13} pero al haber en la URQ dos tareas, ésta se ejecutará a máxima velocidad.
- t=10.En el segundo procesador, el mensaje previsto no ha llegado, con lo que la tarea activa puede reducir la velocidad de ejecución.
- t=16. En el primer procesador, finaliza la tarea τ_{13} que envia un mensaje al segundo procesador, llegando este en el instante t=17 y provocando con ello la activación de la tarea τ_{23}.

En la Figura 59 se ha realizado el diagrama de ejecución del conjunto de tareas de la Tabla 17. En este caso se puede observar que el sistema es planificable. El consumo energético obtenido frente a la planificación a máxima velocidad es del 79%. Dejando 0 unidades de tiempo libre en el primer procesador y 12 unidades de tiempo libre en el segundo procesador.

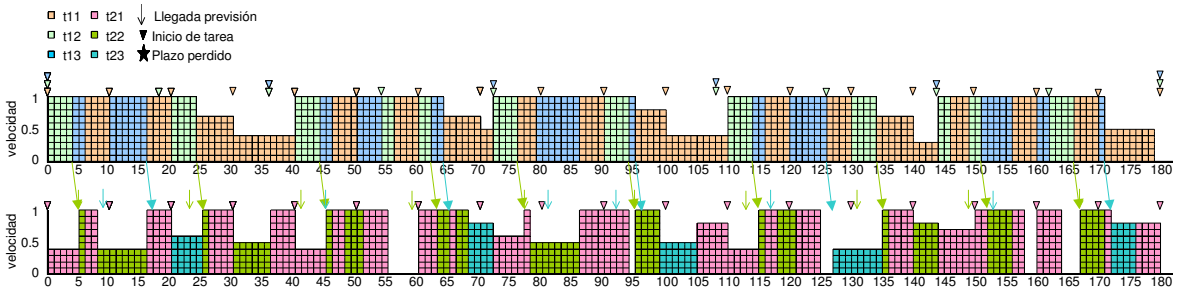

Figura 59: Ejecución usando la planificación PLMDP de los procesos de la Tabla 17 en un sistema distribuido con dos plazos temporales globales usando offset dinámico.

A continuación, se remarcan los instantes de la Figura 59 en los que se aplica la modificación del algoritmo de planificación y la consecuencias que ello comporta:

- En el instante t=0. En el primer procesador, en la URQ hay las tareas τ_{12} y τ_{13} ya que ambas pertenecen a cadenas de precedencias. En la LRQ hay únicamente la tarea τ_{11} hasta que esta promocione. La tarea τ_{12} empieza su ejecución por ser la de mayor prioridad de la URQ. Al finalizar en el instante t=4 envia un mensaje al segundo procesador.
- t=5, en el segundo procesador llega en mensaje enviado por la tarea τ_{12} que provoca la activación de la tarea τ_{22} provocando la expulsión de τ_{21}.
- t=9. en el segundo procesador se activa la previsión de la llegada del mensaje que envía la tarea τ_{13}.

Enhanced Power Low Modified Dual Priority Scheduling Algorithm.

Este algoritmo, como se ha visto en el Capítulo 2, se basa en ahorrar energía reduciendo la velocidad de ejecución cuando exista una o ninguna tarea en la cola de mayor prioridad, pero intentando reducir la velocidad de ejecución del mayor número de tareas posibles, con el fin de tener la mínima cantidad de ejecución a máxima velocidad del procesador.

Las modificaciones realizadas en este planificador son las mismas que para el PLMDP, pero reduciendo la velocidad de ejecución según los parámetros de planificabilidad del conjunto de tareas, utilización máxima del procesador y *breakdown utilization*.

- Fijar la velocidad mínima del procesador en función de la utilización pendiente de ser ejecutada y del *breakdown utilization*.
- Las tareas que pertenecen a una cadena de precedencias entraran directamente a la URQ, en lugar de entrar en la LRQ y esperar que le llegue el instante de promoción.
- No reducción de la velocidad del procesador si la última tarea activa tiene una tarea sucesora.
- Tener en cuenta la previsión de la llegada de una tarea sucesora. No pudiéndose reducir la velocidad aunque exista, únicamente, una tarea activa, cuando se prevea la activación de una tarea sucesora. La reducción de la velocidad de proceso, podría derivar en una perdida del plazo, ya que se producirá una activación de una tarea cuya llegada no estaba contemplada en el momento del cálculo de la velocidad del procesador.
- Envío de un mensaje de activación de las tareas sucesoras.

A continuación se analizará el resultado de la planificación EPLDP en los dos ejemplos analizados en el apartado anterior.

En la Figura 60 se ha realizado el diagrama de ejecución del conjunto de tareas de la Tabla 16. En este caso se puede observar que el sistema es planificable. El consumo energético obtenido frente a la planificación a máxima velocidad es del 68% y

existen 3 unidades de espacios libre del procesador. En este esquema se puede observar que las tareas se ejecutan a una velocidad muy uniforme.

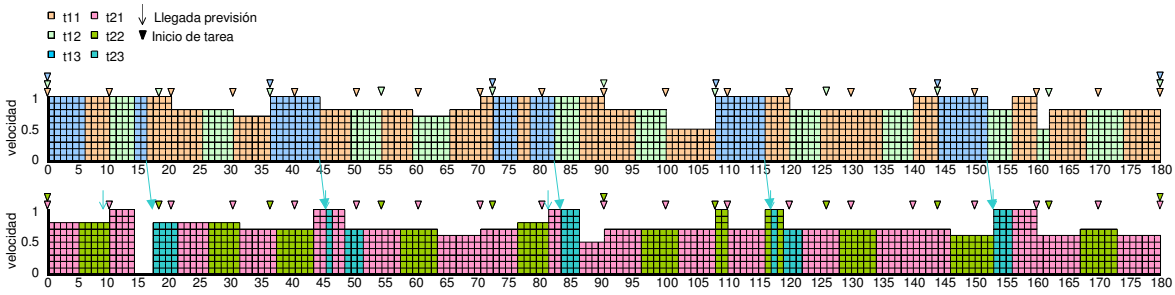

Figura 60: Ejecución usando la planificación EPLDP de los procesos de la Tabla 16 en un sistema distribuido con dos plazos temporales globales usando offset dinámico.

En la Figura 61 se ha realizado el diagrama de ejecución del conjunto de tareas de la Tabla 17. En este caso se puede observar que el sistema es planificable, minimizando el número de intervalos de procesador libre, que es zero en el primer procesador y cuatro unidades en el segundo procesador. La reducción de velocidad en este caso no es tan uniforme como en la figura anterior, debido a que en el segundo procesador normalmente existe una única tarea la mayor parte del tiempo, ya que las demás tareas se activan con la finalización de las tareas precedentes.

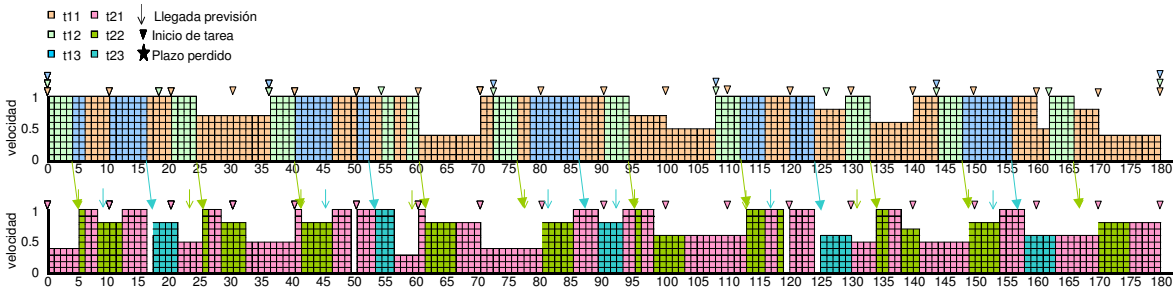

Figura 61: Ejecución usando la planificación EPLDP de los procesos de la Tabla 17 en un sistema distribuido con dos plazos temporales globales usando offset dinámico.

4.3 Conclusiones

En este capítulo se han modificado los algoritmos de planificación propios para tratar el problema de las precedencias y los plazos globales entre tareas del mismo o de distinto procesador. Usualmente, el problema se aborda repartiendo el plazo global entre las tareas que forman la cadena de precedencias, y asignando un offset inicial

igual al plazo temporal de la tarea precedente. Sin embargo este método disminuye la planificabilidad del conjunto.

Nosotros hemos propuesto no repartir la totalidad del plazo global, sino que dejar cierta flexibilidad al sistema para usarlo en las tareas que realmente lo necesitan. Pero dado que los algoritmos están enfocados al ahorro energético utilizando las técnicas de DVS, se ha hecho necesaria la previsión de un tiempo de llegada de las tareas sucesoras, ya que de otra manera, las tareas que se ejecutaban no contaban con un límite temporal de ejecución, pudiendo resultar en la perdida de algún plazo.

Se ha propuesto una predicción conservativa del tiempo de llegada de las tareas que garantiza el cumplimiento de los plazos ahorrando energía. Los resultados muestran que los algoritmos propuestos sin la repartición estática de los plazos, son capaces de planificar un mayor conjunto de sistemas que los que utilizan una repartición estática. Así mismo se ha observado que los planificadores basados en el *Dual Priority* son capaces de planificar conjuntos que no lo son utilizando los planificadores LPFPS o LPEDF.

En los ejemplos que se han desarrollado, se puede observar:

- Cuando el conjunto es planificable con todos los algoritmos estudiados, el algoritmo que aporta un mayor ahorro energético es el EPLDP con un 32%, seguido del PLMDP con un 21%, el LPEDF con un 13% y finalmente el LPFPS con un 10% de ahorro energético.
- Existen conjuntos que únicamente son planificables con nuestros algoritmos (PLMDP, EPLDP). Con los otros algoritmos estudiados existe una perdida de un plazo temporal.
- En los capítulos anteriores con sistemas monoprocesadores se ha observado que el algoritmo que aporta mayor ahorro energético es el EPLDP, con lo que estos resultados se podrían extrapolar al caso de sistemas distribuidos, como se ha observado con el conjunto evaluado.

Capítulo 5

DUALIDAD ENTRE AHORRO DE ENERGÍA Y TIEMPO DE FINALIZACIÓN

El objetivo de este capítulo es analizar la planificación de las tareas aperiódicas, siendo el parámetro de rendimiento usual del planificador, el tiempo de finalización medio que proporciona a estas. Se debe tener en cuenta que la idea de las técnicas del DVS de ralentizar la velocidad de ejecución de las tareas se contrapone con la idea de obtener una mayor productividad del procesador objetivo perseguido hasta la actualidad y que ha provocado parte de los grandes avances tecnológicos actuales.

5.1 Introducción

Tal como se ha expuesto en el Capítulo 1, los sistemas de tiempo real usualmente están formados por tareas periódicas, tareas asociadas a eventos que tienen un comportamiento cíclico, la lectura de un sensor cada cierto intervalo de tiempo, recepción de los datos de un radar,... Sin embargo también existen componentes no periódicas, eventos externos que se activan en instantes aleatorios, como podrían ser las ordenes de un operador, las alarmas que se producen en un sistema de control,... En estos sistemas mixtos, el planificador tiene que ser capaz de integrar las tareas periódicas con las tareas aperiódicas, ofreciendo a estas últimas el menor tiempo de finalización posible. En la Figura 62 se puede observar el esquema de ejecución de las tareas periódicas y aperiódicas.

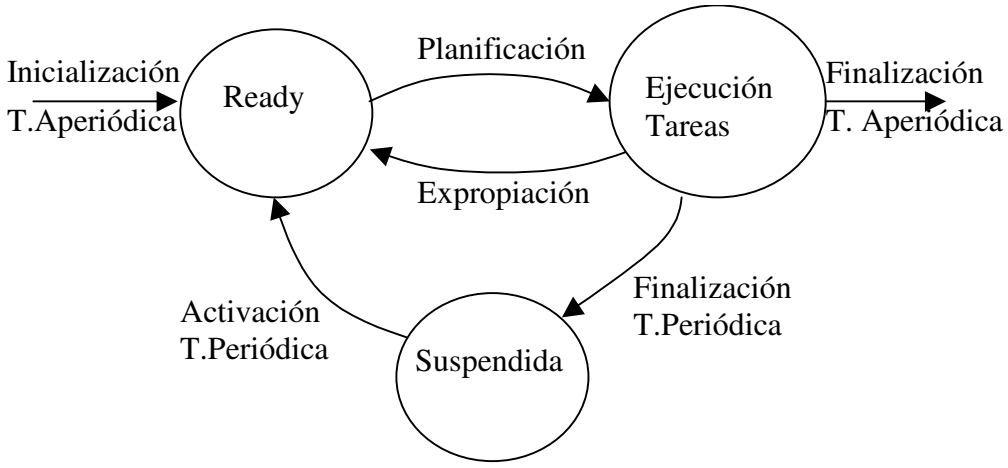

Figura 62: Esquema de ejecución de las tareas periódicas y aperiódicas

En los sistemas reales existen diversos tipos de tareas aperiódicas:

- Las tareas esporádicas, aquellas tareas que tienen un plazo temporal estricto y tienen normalmente un intervalo mínimo entre las activaciones. Para este tipo de tareas, lo primordial es el cumplimiento del plazo temporal en lugar del tiempo de finalización. Estas tareas esporádicas deben garantizarse siempre y una práctica común es analizar la planificación de estas tareas como si fueran periódicas con periodo igual al mínimo intervalo de tiempo entre las activaciones.

- Las tareas *firm* aperiódicas, son tareas que también tienen un plazo temporal, pero estas tareas pueden rechazarse. Si la tarea se acepta, esta debe finalizar con anterioridad al plazo temporal. Si se rechaza, disminuye la calidad de servicio del sistema.
- Las tareas aperiódicas, estas tareas se aceptan siempre y no disponen de plazo temporal estricto, como máximo pueden tenerlo flexible. Para este tipo de tareas, lo importante es el tiempo finalización medio.

En este capítulo se analizaran únicamente las tareas aperiódicas sin plazo temporal, sin entrar en el tratamiento de los demás tipos. Para ello, se comentaran qué planificadores específicos para el tratamiento de las tareas aperiódicas existen, analizando los tres mejores de ellos (uno de prioridad estática, uno de prioridades dinámicas y uno mixto). A continuación se realizará el estudio de la planificación de las tareas aperiódicas en los planificadores utilizados en la presente tesis. Para finalizar con la unión de la planificación de las tareas aperiódicas junto al ahorro energético y se finalizará el capítulo con las conclusiones obtenidas.

5.2 Planificación de las tareas aperiódicas

En la planificación estática, la manera más simple de planificar las tareas aperiódicas sin interferir en la planificación de las tareas periódicas es asignándoles una prioridad inferior a estas y igual a todas ellas, utilizando un esquema FIFO entre todas las tareas aperiódicas, de manera que la carga aperiódica únicamente se ejecutaran cuando no exista carga periódica. Este esquema de planificación se denomina *background*, y el tiempo de finalización que obtienen las tareas aperiódicas es demasiado grande, de manera que este tiempo de finalización se podría considerar como una cota superior, cualquier esquema de planificación que se use, tiene que mejorar el tiempo de finalización de este tipo de tareas. Una mejora de estos métodos consiste en el uso de los servidores [42]. Estos servidores son tareas periódicas artificiales que disponen de una capacidad de cálculo y que cuando les toca ejecutarse, ejecutan la carga aperiódica en su lugar, de manera que se puede asegurar que no interferirán con la ejecución de las tareas periódicas, pero ofrecerán un mejor tiempo de finalización a las tareas aperiódicas. Cuando se ejecuta carga aperiódica la

capacidad se va utilizando. Esta capacidad se rellenará al máximo al inicio de cada periodo de la tarea servidora.

Existen varias clases de servidores según el relleno y uso de la capacidad, el primero que apareció y el mas sencillo es el "*polling server"* (PS). Es una tarea periódica con un tiempo de cálculo, un periodo y con la mayor prioridad del sistema. Al ejecutarse mira en la cola de aperiódicas si existe trabajo aperiódico, si existe, se ejecuta durante el tiempo de cálculo asignado al servidor, si no existe el servidor se suspende hasta el siguiente periodo, perdiendo la capacidad no utilizada. El problema de este servidor, es que si la tarea aperiódica llega inmediatamente después de la activación del servidor, esta debe esperarse a la siguiente activación, aunque la capacidad del servidor no halla sido utilizada. Los servidores que conservan la capacidad no utilizada son los llamados servidores que preservan la capacidad (*bandwidth-perserving servers*), entre ellos se hallan los *Deferrable servers*, los *Sporadic server*, los *Constant utilization servers*, los *Total bandwidth servers*, *losWeighted fair queuing servers*, etc.

El "*Deferrable Server"* (DS) (Servidor diferido) [42] es el más simple de implementar de todos los servidores que preservan la capacidad, en el se define un periodo y una capacidad. Estos valores son los máximos posibles que continúan ofreciendo planificabilidad a las tareas periódicas. El funcionamiento de este servidor es el siguiente: En tiempo de ejecución cuando llega una tarea aperiódica si existe capacidad disponible, empieza a ejecutarse inmediatamente y continua hasta que finaliza o se agota la capacidad del servidor. La capacidad del servidor se decrementa conforme se ejecutan las tareas aperiódicas. Si se agota la capacidad del servidor, se suspende la tarea aperiódica o se envía en background. Cada periodo de tiempo se rellena la capacidad del servidor. Si en el instante de activación del servidor no existen tareas aperiódicas, la capacidad del servidor no se pierde si no que se mantiene durante todo el periodo del servidor.

El "*Sporadic Server"* (SS) (Servidor esporádico) [42] se diferencia del anterior en el relleno de la capacidad. Si un proceso en el instante t utiliza una capacidad C, el servidor rellenará esta capacidad C después de T unidades de tiempo. En general, el SS proporciona una capacidad mayor que el DS, pero tiene una implementación más costosa.

Hay otros métodos que no utilizan servidores para planificar tareas aperiódicas, estos serian el "*Slack Stealing"* [44], [22] que Tia et al. demostró que no era óptimo [82] y el "*Dual Priority"* (Anexo 2) entre otros. Todos estos algoritmos están basados en la idea de robar todo el tiempo posible a las tareas periódicas, sin causarles ninguna perdida de plazos temporales retardando la ejecución de las tareas periódicas al máximo para obtener un buen tiempo de finalización a las tareas aperiódicas [21]. Lo que varía de un algoritmo a otro es la forma de calcular el retardo de las tareas periódicas.

El "*Slack Stealing"* (Extractor de holgura) aprovecha el *slack* (tiempo disponible entre la finalización de una tarea y el plazo temporal) moviéndolo antes de la ejecución de la tarea periódica. Este algoritmo requiere tener computado de manera exacta todos los intervalos libres que dejan las tareas periódicas a lo largo del hiperperiodo, de manera que la planificación requiere del uso de tablas muy grandes (longitud del hiperperiodo) o calcular los intervalos libres en tiempo de ejecución (sobrecarga del procesador).

El planificador "*Dual Priority"* (Prioridad dual) (Anexo 2), obtiene tiempos de finalización parecidos al "*Slack Stealing"*, sin tener la sobrecarga de memoria ni de cálculo [54]. En este caso, las tareas aperiódicas se ejecutan en un nivel de prioridad intermedio, de manera que ante la presencia de tareas aperiódicas únicamente se ejecutaran aquellas tareas periódicas que han promocionado, es decir, aquellas tareas que no pueden retardarse mas. En [54] se realiza una modificación del *Dual Priority* para mejorar el tiempo de finalización de las aperiódicas. La modificación consiste en tener en cuenta el tiempo que las tareas periódicas se han ejecutado antes de ser promocionadas. Este tiempo ejecutado en el nivel inferior de prioridad puede ser usado por las tareas aperiódicas hasta la llegada del plazo de la tarea.

Spuri and Buttazzo en [79] presenta unos algoritmos óptimos para prioridades dinámicas usando el planificador EDF de base. Presenta el "*Earliest Deadline Last"* (EDL) y el "*Improve Priority Exchange"* (IPE) que es una aproximación de éste último. Este algoritmo asigna las prioridades usando EDL y calcula el tiempo que queda libre el procesador, el instante en que se produce este hueco y la longitud del mismo y lo almacena en un vector. Si en el instante que hay hueco no existe ninguna aperiódica, se ejecuta una tarea periódica de menor prioridad, intercambiando su

prioridad con la del servidor, de esta manera, el hueco no se pierde y queda disponible para la llegada de tareas aperiódicas.

A continuación se compararan el *Background*, el *Dual Priority*, el *Slack Stealing*, y el *Improve Priority Exchange*, a nivel de tiempo de finalización que se obtiene de las tareas aperiódicas en los distintos casos. El hecho que únicamente se compare con estos algoritmos es debido a que estos son los considerados mejores con asignación de prioridades estática (SS), el mejor con asignación de prioridades dinámicas (IPE) y el *Dual Priority* que es nuestro algoritmo base.

En las simulaciones se han generado conjuntos de tareas con una utilización periódica y aperiódica escogidas aleatoriamente de la siguiente manera:

- Número de tareas periódicas entre 4 y 10.
- Periodos entre 5 y 500.
- Tiempo de cálculo escogidos aleatoriamente para que las tareas sean planificables.
- La llegada de las tareas aperiódicas sigue una distribución de Poisson y su requerimiento de cálculo varia entre 1 y 500.
- Finalmente, se ha variado la utilización del procesador del 70% al 95 %, descartando los conjuntos con una diferencia de utilización mayor al 1% de la utilización demandada. La manera de variar la utilización del procesador dependerá del experimento realizado.

Todas las simulaciones se han realizado hasta la finalización de 100.000 tareas aperiódicas. Cada conjunto se ha simulado varias veces hasta obtener un tiempo de finalización medio con un intervalo de confianza del 95%.

En la Figura 63, se ha simulado 100 conjuntos de tareas aleatorias variando la utilización periódica entre un 60 y un 80%. En todos ellos se ha fijado una utilización aperiódica del 20%. Los puntos representados en el gráfico es la media aritmética del tiempo medio de finalización de las tareas aperiódicas dividido por el tiempo de ejecución de las tareas aperiódicas en cada conjunto.

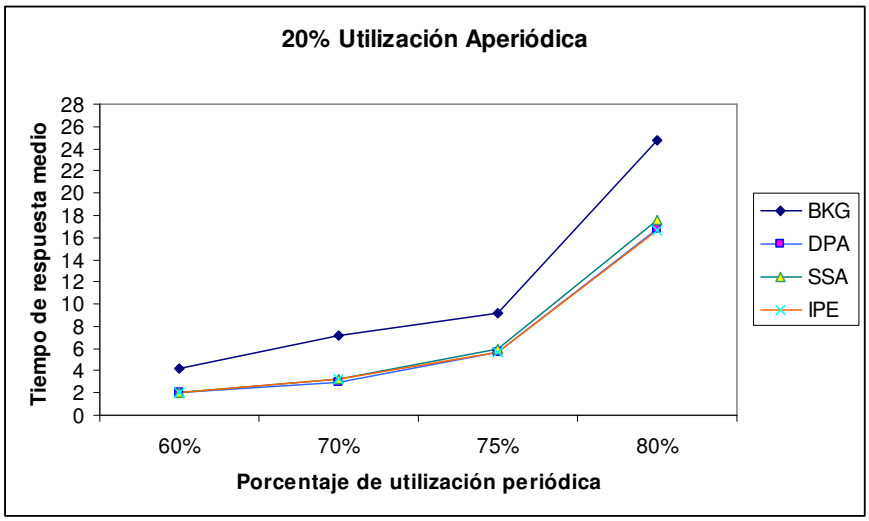

Figura 63: Tiempo de finalización medio variando la utilización periódica del procesador.

En la Figura 63, se puede observar que conforme la carga periódica aumenta, el *Dual priority* (DPA) y el *Improve Priority Exchange* (IPE) obtienen mejores resultados medios que el *Slack Stealing* (SSA). Esto es debido a que las prioridades dinámicas de las tareas permiten una mayor flexibilidad para permitir la ejecución de las tareas aperiódicas.

En la siguiente simulación se ha variado el tiempo de ejecución de las tareas aperiódicas fijándolas a 1. La utilización aperiódica de cada conjunto se ha variado entre un 20 y un 33%. La utilización periódica se ha fijado a un 60%.

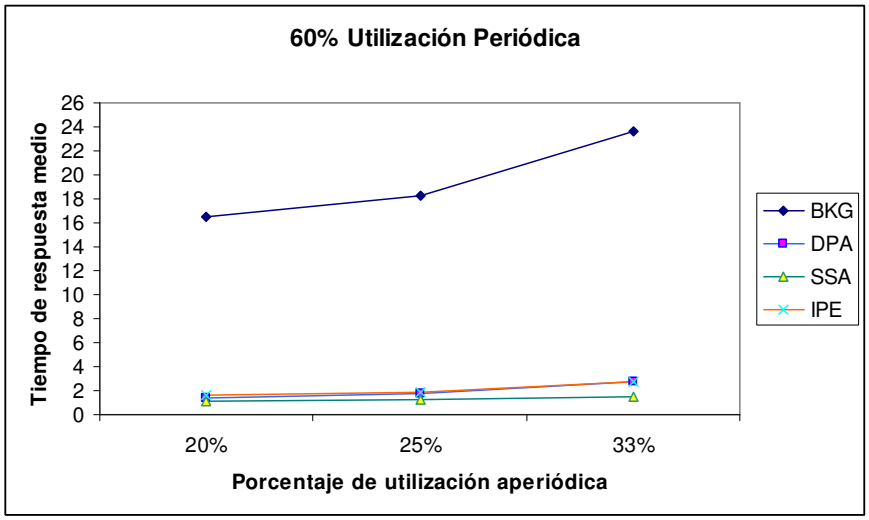

Figura 64: Tiempo de finalización medio variando la utilización aperiódica del procesador.

En la Figura 64 se puede observar que con utilizaciones periódicas bajas del procesador el *Improve Priority Exchange* (IPE) y el *Dual Priority* (DPA) tienen el mismo comportamiento a nivel de tiempo de finalización de la tareas aperiódicas, mientras que el *Slack Stealing* (SS) mejora un poco.

Observando los resultados obtenidos anteriormente, se puede concluir que el planificador más adecuado para el tratamiento de las tareas aperiódicas y ahorro energético es el *Dual Priority* dado que los tiempos de finalización que obtienen las tareas aperiódicas son aceptables siendo además el planificador con una menor sobrecarga de implementación y de utilización de memoria. En el siguiente punto se realizará el estudio combinado de ahorro energético combinado con el tratamiento de las tareas aperiódicas.

5.3 Tareas aperiódicas y ahorro energético

Si se observa el comportamiento de las tareas periódicas cuando se usan los algoritmos de planificación orientados al tratamiento de las tareas aperiódicas, y se comparará con el comportamiento de este mismo tipo de tareas usando los algoritmos de planificación con ahorro energético que usan las técnicas del DVS, se observa lo siguiente:

- La planificación de tareas background es equivalente a la planificación con ahorro energético LPFPS, se reduce la velocidad del procesador cuando existe una única tarea lista para ser ejecutada. En el caso de la planificación background, las tareas aperiódicas se ejecutaran únicamente cuando no hay ninguna tarea periódica lista para ser ejecutada. Si se traduce a la planificación con ahorro energético, la última tarea periódica reducirá la velocidad del procesador, que sería el momento en que se ejecutarían las tareas aperiódicas activas en el sistema.

- La planificación de tareas aperiódicas que se obtiene usando el algoritmo del *Dual Priority* el tiempo de finalización de las tareas es mejor que el que se obtiene con el algoritmo background. Las tareas aperiódicas se ejecutaran cuando no existan tareas periódicas en la cola de mayor prioridad, momento en el que el PLMDP reduce la velocidad de ejecución de las tareas del

procesador. Si se compara la planificación resultante del *Dual Priority* con el PLMDP se observa que la reducción energética se obtiene en el momento en que se ejecutarían las tareas aperiódicas.

Resumiendo, se puede afirmar que en los algoritmos de planificación con tareas aperiódicas que se basan en retardar las tareas periódicas se puede obtener ahorro energético usando las técnicas del DVS, ya que estas implican provocar una finalización tardía de las tareas periódicas a base de ejecutar las a una velocidad reducida. Por lo tanto las técnicas DVS provocaran el mismo esquema de ejecución de tareas periódicas que se obtendría si siempre existieran tareas aperiódicas listas para ser ejecutadas.

Generalizando se podría afirmar que cualquier algoritmo de planificación pensado para dar servicio a las tareas aperiódicas se puede usar para reducir la velocidad del procesador, ahorrando, de este modo, energía. De manera que, intuitivamente, se podría pensar que los planificadores que más eficientemente tratan a las tareas aperiódicas, es decir, obtienen un tiempo medio de finalización menor, serán aquellos que permitirán, también, obtener un mayor ahorro energético. Sin embargo esta intuición no es del todo cierta, ya que en un conjunto de tareas real, no siempre habrá carga aperiódica lista para ser ejecutada, lo que realmente importa es la disponibilidad de retardar las tareas periódicas en el momento de llegada de una tarea aperiódica. Y en los planificadores con ahorro energético, lo que importa es el poder reducir la velocidad de proceso el máximo tiempo posible, de manera que no se tengan que ejecutar tareas a máxima velocidad, cosa que se conseguirá si se puede distribuir de manera uniforme las tareas periódicas.

Los algoritmos implementados, solo reducirán la velocidad de ejecución de las tareas periódicas cuando no existan tareas aperiódicas, puesto que se intenta obtener el menor tiempo de finalización medio para estas. En este caso, se observa que en el algoritmo LPFPS no se podrá obtener ahorro energético ante la presencia de tareas aperiódicas, puesto que estas se ejecutaran cuando no exista carga aperiódica y el algoritmo únicamente reducirá la velocidad de ejecución de la última tarea que se ejecuta.

A continuación se compararán los distintos algoritmos de planificación, tanto a nivel de tiempo de finalización que ofrecen a las tareas aperiódicas como de energía

consumida. Los planificadores reducirán la velocidad de las tareas periódicas siempre que sea posible. Las tareas aperiódicas se ejecutarán siempre a máxima velocidad.

Los conjuntos de prueba se han generado de la siguiente manera:

- Los conjuntos son de 8 tareas periódicas con periodos armónicos generados aleatoriamente entre 1024 y 65536. La carga máxima de las tareas se ha fijado al 20% y se ha variado la utilización periódica entre un 60 y un 85%.
- La carga aperiódica se ha fijado en un 10%, en el que la llegada de las tareas aperiódicas sigue una distribución de Poisson y su requerimiento de cálculo se ha fijado a 1 o se ha fijado según la media de la distribución escogida.
- Cada punto representado en el gráfico es el valores promedio de 100 conjuntos de pruebas. La duración de las simulaciones se ha fijado en 100 hiperperiodos.

Las siguientes figuras van en parejas, en la primera se muestra la energía consumida normalizada respecto a la ejecución a máxima velocidad del procesador, y en la siguiente se muestra el tiempo de finalización medio obtenido por las tareas aperiódicas, normalizado respecto al WCET de la tarea aperiódica.

En las Figura 65 y Figura 66, se muestra el resultado de las simulaciones a nivel de consumo energético normalizado y del tiempo de finalización obtenido por las tareas aperiódicas. En este experimento con cálculo aperiódico igual a 1 y por lo tanto se muestra la disponibilidad del algoritmo de planificación a atender tareas aperiódicas, en la Figura 65 se puede observar que el algoritmo que consume menor energía es el EPLDP-m, en cualquier consumo de WCET de las tareas periódicas. Sin embargo, en la Figura 66 se puede observar que el algoritmo que proporciona una mayor disponibilidad a las tareas aperiódicas es el LPEDF, debido a la presencia de un servidor.

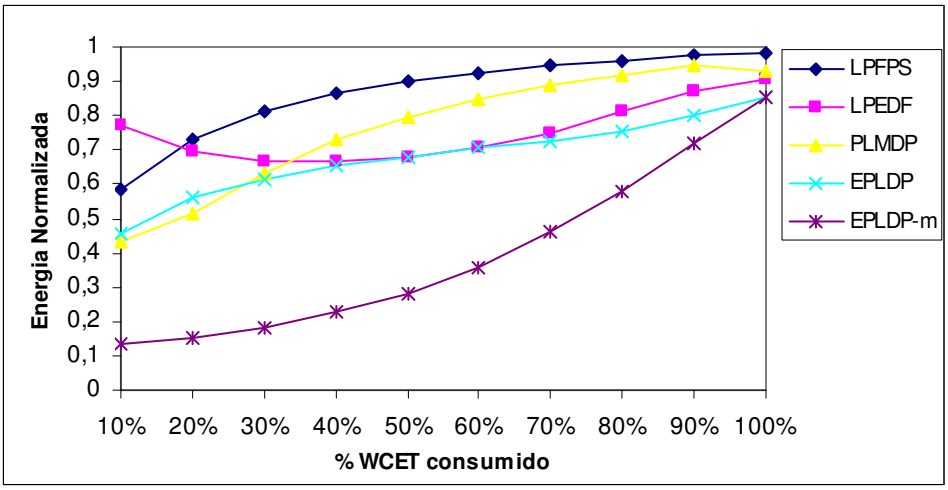

Figura 65: Gráfica de consumo energético de los distintos planificadores con una utilización periódica del 75% con periodos armónicos y una utilización aperiódica del 10% con un tiempo de cálculo aperiodico de 1.

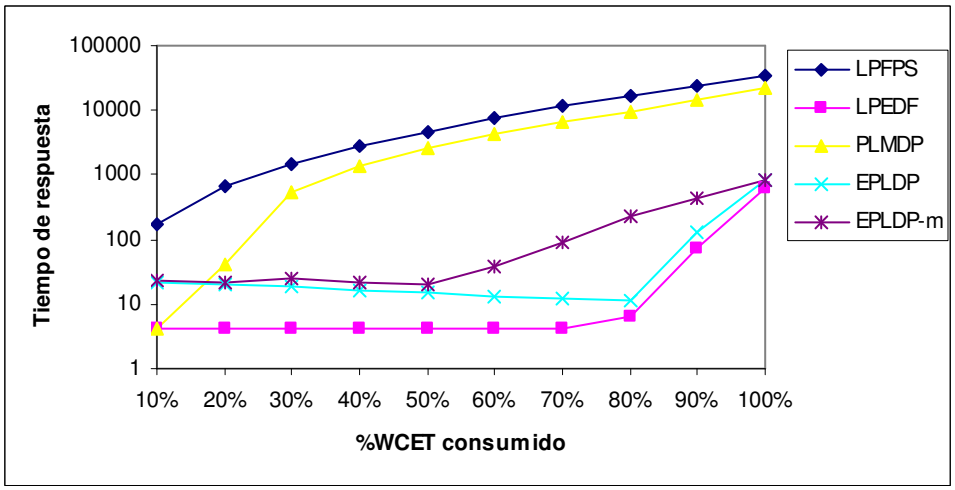

Figura 66: Gráfica de tiempo de finalización normalizado de los distintos planificadores con una utilización periódica del 75% con periodos armónicos y una utilización aperiódica del 10% con un tiempo de cálculo aperiódico de 1.

En las Figura 67 y Figura 68 el tiempo de cálculo de las tareas aperiódicas es mayor, pero la llegada de esta es menos frecuente, manteniendo la utilización aperiódica en el 10%. En este caso, el EPLDP-m obtiene un mejor rendimiento energético, con un tiempo de finalización de las tareas aperiódicas similar al que se obtiene con el LPEDF. En la Figura 68 también se muestra el tiempo de finalización de las tareas aperiódicas que se obtiene ejecutando a máxima velocidad con las técnicas de background, que se puede observar que es equivalente al que se obtiene con el LPFPS.

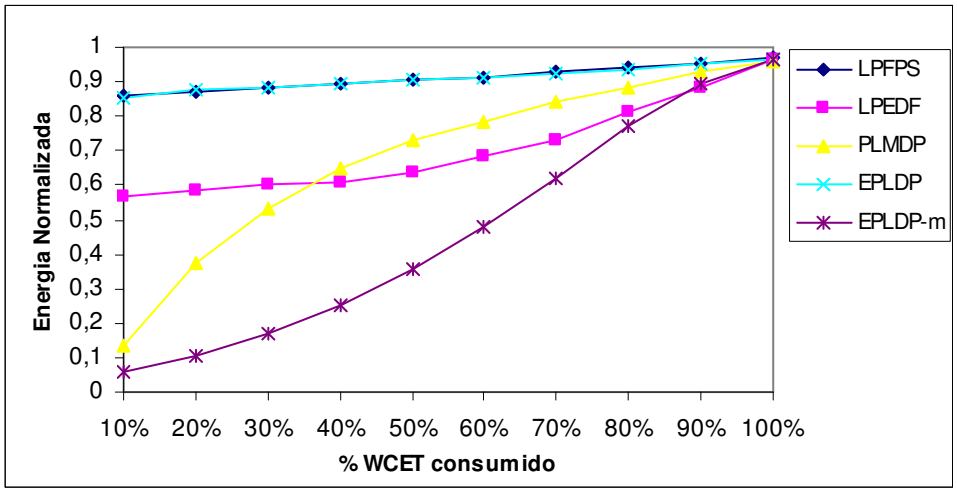

Figura 67: Gráfica de consumo energético de los distintos planificadores con una utilización periódica del 75% con periodos armónicos y una utilización aperiódica del 10% con una distribución de poisson de media el periodo de la tarea de menor periodo.

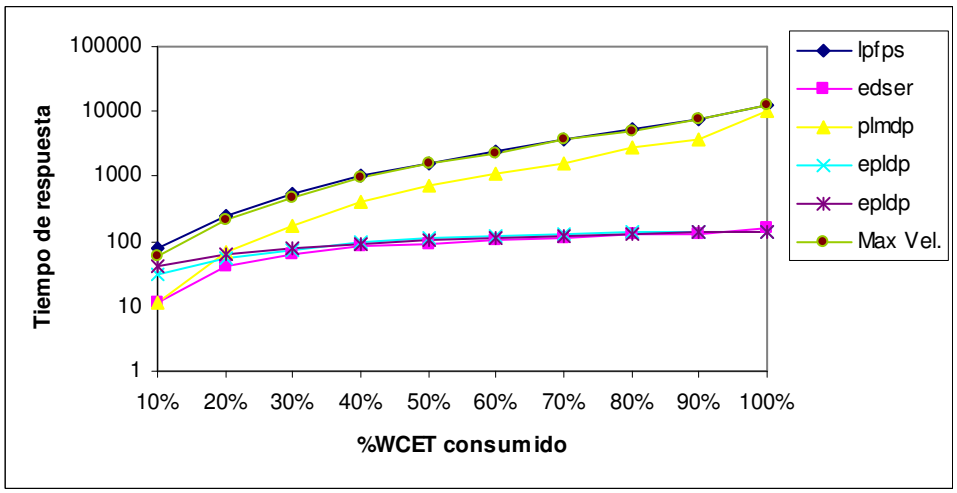

Figura 68: Gráfica de tiempo de finalización normalizado de los distintos planificadores con una utilización periódica del 75% con periodos armónicos y una utilización aperiódica del 10% con una distribución de poisson de media el periodo de la tarea de menor periodo.

De la Figura 69 a la Figura 74, se muestra las gráficas variando la utilización periódica del procesador. En las dos primeras el WCET consumido por las tareas periódicas es del 20%, en las dos siguientes es del 50% y por último en las dos siguientes es del 90%. La carga aperiódica se mantiene al 10% con un tiempo de cálculo de una unidad.

En la Figura 69 y Figura 70, se ve que el ahorro energético y el tiempo de finalización obtenido no depende de la utilización de las tareas periódicas, puesto que

el consumo real del procesador es muy bajo, varia del 20% de 60 al 20% de 85. Sin embargo, el algoritmo de planificación utilizado hace variar en gran medida los resultados obtenidos. A nivel energético el mejor es el EPLDP-m mientras que a nivel de tiempo de finalización el mejor es el LPEDF.

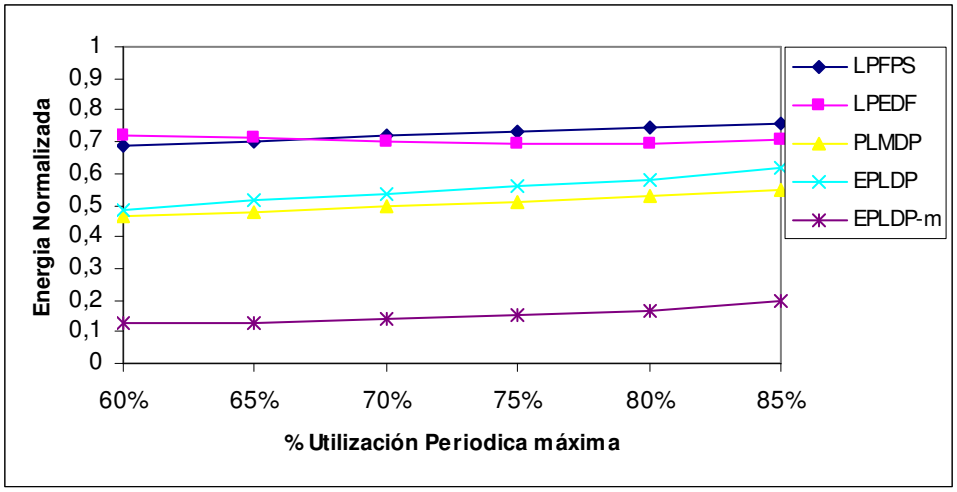

Figura 69: Gráfica de consumo energético normalizado de los distintos planificadores con un consumo del 20% del WCET en las tareas periódicas con periodos armónicos y una utilización aperiódica del 10% con un tiempo de cálculo de las tareas aperiódicas de 1.

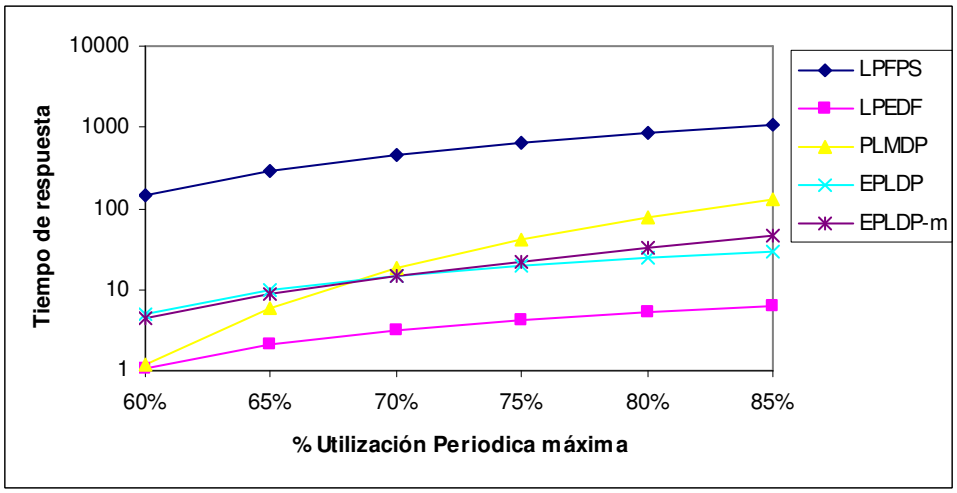

Figura 70: Gráfica de tiempo de finalización normalizado de los distintos planificadores con un consumo del 20% del WCET en las tareas periódicas con periodos armónicos y una utilización aperiódica del 10% con un tiempo de cálculo de las tareas aperiódicas de 1.

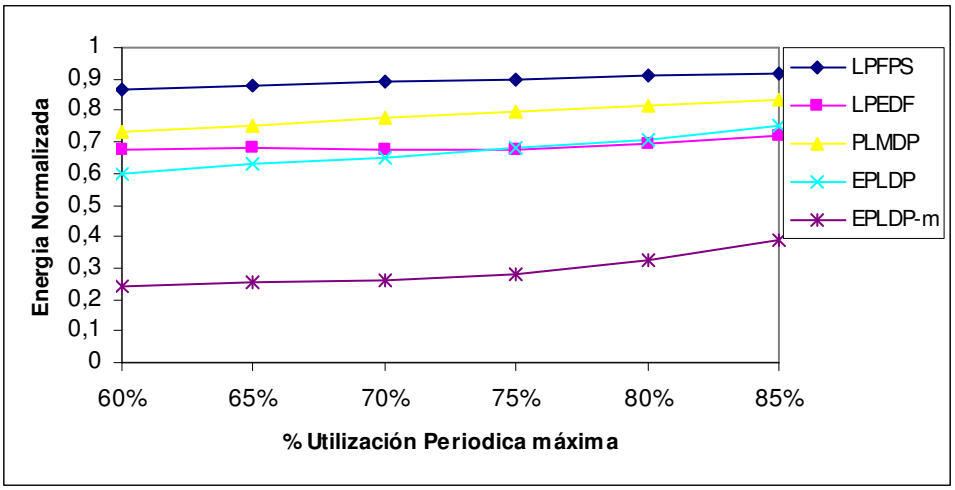

Figura 71: Gráfica de consumo energético normalizado de los distintos planificadores con un consumo del 50% del WCET en las tareas periódicas con periodos armónicos y una utilización aperiódica del 10% con un tiempo de cálculo de las tareas aperiódicas de 1.

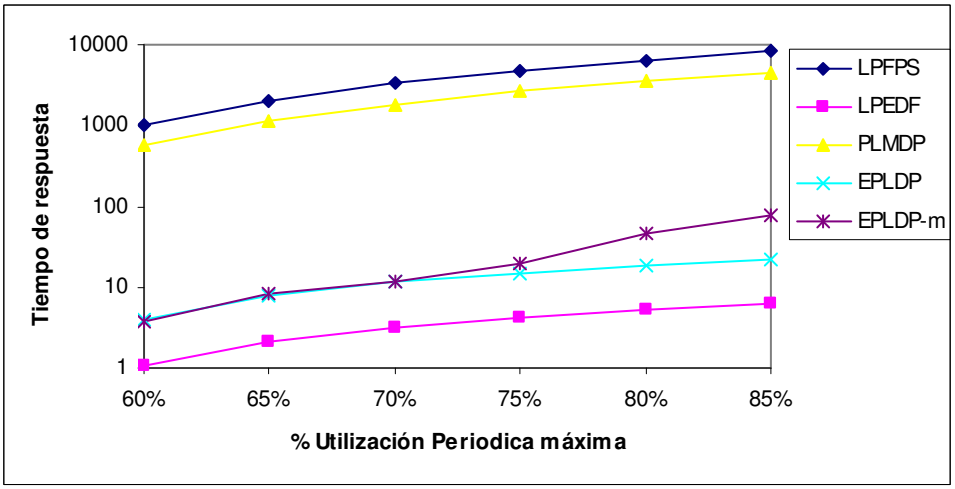

Figura 72: Gráfica de tiempo de finalización normalizado de los distintos planificadores con un consumo del 50% del WCET en las tareas periódicas con periodos armónicos y una utilización aperiódica del 10% con un tiempo de cálculo de las tareas aperiódicas de 1.

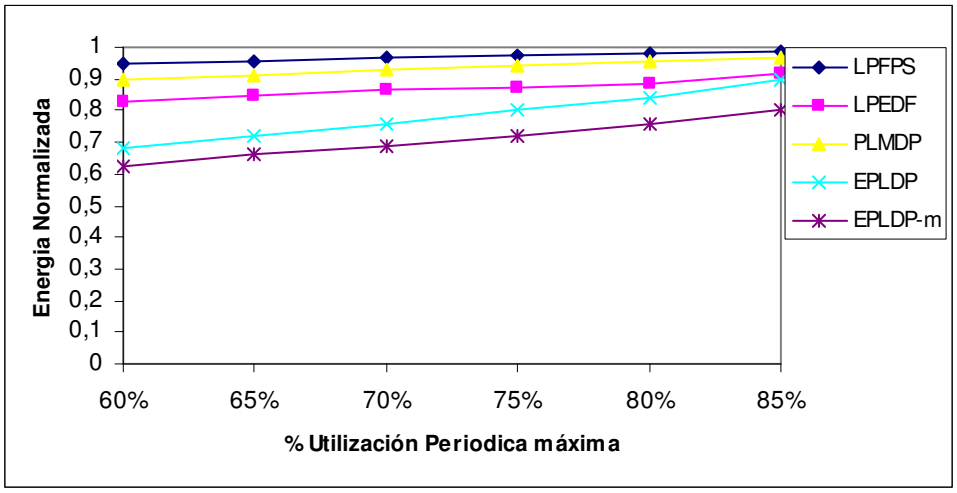

Figura 73: Gráfica de consumo energético normalizado de los distintos planificadores con un consumo del 90% del WCET en las tareas periódicas con periodos armónicos y una utilización aperiódica del 10% con un tiempo de cálculo de las tareas aperiódicas de 1.

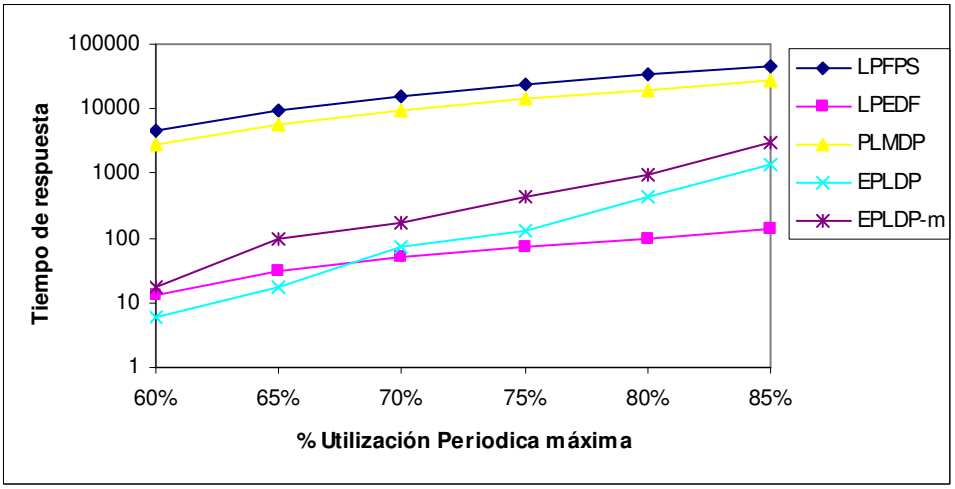

Figura 74: Gráfica de tiempo de finalización normalizado de los distintos planificadores con un consumo del 90% del WCET en las tareas periódicas con periodos armónicos y una utilización aperiódica del 10% con un tiempo de cálculo de las tareas aperiódicas de 1.

En las cuatro figuras anteriores, se puede observar en que cuanto más cargado está el procesador con tareas periódicas, el consumo energético de los distintos algoritmos es más equivalente, mientras que el comportamiento a nivel de tiempo de finalización de tares aperiódicas no depende tanto de la utilización, sino que depende en mayor medida de la flexibilidad del planificador en retardar la ejecución de las tareas periódicas en el instante en que las tareas aperiódicas lleguen al sistema.

5.4 Conclusiones

En este capítulo se ha reflexionado sobre el problema conjunto de la obtención de un tiempo de finalización aceptable para las tareas aperiódicas y el consumo energético que se obtiene en su ejecución. Ambos conceptos están claramente enfrentados, puesto que si se ejecutan las tareas periódicas a menor velocidad, dejaran menor disponibilidad para la ejecución inmediata de las tareas aperiódicas.

En las distintas figuras que se han presentado, se ha podido observar que el planificador EPLDP-m, es el que presenta una mejor relación entre consumo energético y tiempo de finalización a las tareas aperiódicas, siendo el LPEDF el que presenta un mejor tiempo de finalización a las tareas aperiódicas.

CONCLUSIONES Y TRABAJO FUTURO

La evolución de los procesadores siempre ha consistido en ir aumentando el rendimiento de estos, fijando como medida de este aumento de rendimiento la velocidad de proceso en la ejecución de las distintas aplicaciones. Sin embargo esta mayor velocidad implica también un mayor consumo energético. En la era de la informática móvil, un mayor consumo energético comporta el uso de mayores baterías y una disminución del tiempo útil de trabajo de estas. A su vez, el usuario común de esta informática móvil, puede que no necesita una mayor velocidad de proceso, puesto que la limitación de velocidad de las aplicaciones más comunes que usará, aplicaciones multimedia, procesadores de texto, viene impuesta por cualidades físicas (velocidad de mecanografía, frecuencia de visualización en pantalla, frecuencia de emisión del sonido,..). En estos casos, por más rápido que sea el procesador, la aplicación no puede ejecutarse a mayor velocidad, resultando en un mayor consumo energético pero sin la obtención de mejores resultados.

En los sistemas de tiempo real, estas restricciones en la velocidad del procesador también vienen impuestas por las características físicas del medio, por ejemplo, la frecuencia de muestreo de un sensor de temperatura, debe ser la necesaria, de nada sirve que se tomen medidas al doble de su frecuencia, puesto que la temperatura no cambiará tan rápidamente.

Para evitar el problema de sobredimensionado de la velocidad del procesador que requiere a su vez, de un mayor consumo energético, en la última década, el estudio del rendimiento de los procesadores (velocidad del procesador) se ha unido al estudio del consumo energético y de la disipación del calor. De manera que ya no puede haber evolución únicamente a nivel de velocidad del procesador sin tener en cuenta el consumo de este y la disipación de energía [85].

En el campo de la reducción energética se ha observado que la técnica del DVS (reducción dinámica de la velocidad del procesador junto con la reducción del voltaje suministrado) permite al algoritmo de planificación de las tareas ahorrar energía. Si se observan atentamente los algoritmos de planificación que se han usado en los entornos de tiempo real, se puede observar la siguiente evolución temporal:

- Algoritmos de planificación cíclicos.
- Algoritmos de planificación en-línea para conjuntos de tareas periódicas independientes.

- Algoritmos de planificación en-línea para conjuntos de tareas periódicas independientes y para tareas aperiódicas y/o esporádicas
- Algoritmos de planificación en-línea para conjuntos de tareas no independientes entre si, es decir tareas con recursos compartidos y/o precedencias entre tareas.
- Algoritmos de planificación para arquitecturas multiprocesadores.

Una evolución parecida se ha producido en el estudio de los planificadores con ahorro energético, iniciando el estudio con la planificación estática y finalizando con los algoritmos de planificación dinámica, que ofrecen mayor flexibilidad a los programadores de las aplicaciones.

En la presente tesis, se ha seguido esta misma evolución, estudiando y proponiendo un algoritmo de planificación con ahorro energético para un sistema monoprocesador con tareas independientes (Capítulo 2), inicialmente, suponiendo correctos todos los parámetros de diseño del sistema (algoritmo PLMDP) para finalmente aplicar de manera automática una corrección a un posible sobredimensionado del WCET (algoritmo EPLDP). Estudiando las diferencias de ahorro energético que se obtienen con estos dos algoritmos, se ha constatado que a nivel energético es siempre favorable ejecutar a una velocidad de proceso media en lugar de ejecutar a velocidades extremas (primero a muy poca velocidad, para después tener que ejecutar a velocidades más altas).

En el Capítulo 3 se añade a los planificadores la posibilidad de que las tareas puedan acceder a recursos compartidos entre estas. Implementando en los planificadores el protocolo de techo de prioridad. A pesar del uso de este protocolo, se puede observar que nuestros algoritmos continúan aportando ahorro energético, y pudiendo garantizar la planificabilidad de las tareas.

A continuación se ha añadido a todos los planificadores implementados, la posibilidad de tener precedencias entre las tareas y plazos globales entre una cadena de precedencias (Capítulo 4). En este capítulo se ha visto que la dificultad principal del algoritmo de planificación es el hecho de no saber con exactitud el instante de llegada de estas tareas, disminuyendo de esta manera la utilización del procesador que se puede planificar de una manera garantizada. A pesar de todo, se observa que

los algoritmos de planificación PLMDP y EPLDP son los que mejor se comportan en términos de planificación de tareas.

Para finalizar en el Capítulo 5 se ha realizado el estudio del tiempo de finalización de las tareas aperiódicas junto al ahorro energético de los procesadores. En principio ambos conceptos están enfrentados, puesto que para obtener un tiempo de finalización menor, se debe ejecutar la tarea a máxima velocidad, asegurando que en el procesador esté disponible a la llegada de las tareas. Y por el contrario, para tener ahorro energético se debe ejecutar a una velocidad reducida la mayor parte del tiempo.

Observando en conjunto todos los capítulos de la tesis, se ha visto que los planificadores PLMDP y EPLDP, obtienen un buen rendimiento, siendo de fácil implementación y requiriendo al mismo tiempo poca cantidad de memoria para su funcionamiento. Sin embargo, todos los resultados obtenidos se han realizado bajo simulaciones con conjuntos de tareas sintéticos. Ahora seria conveniente la implementación de estos planificadores en un núcleo real en una arquitectura que soporte las técnicas del DVS para poder contrastar los resultados que se obtienen con los obtenidos mediante las simulaciones.

Estas técnicas DVS y las modificaciones que se han realizado en los algoritmos de planificación para ahorro energético, son fácilmente aplicables a otros algoritmos de planificación, en concreto se podrían aplicar sin demasiada complicación a sistemas multiprocesadores con una planificación global de las tareas. Los resultados obtenidos en el capítulo 2 con el algoritmo EPLDP demuestran que en las aplicaciones de tiempo real es energéticamente económico ejecutar a una velocidad reducida el máximo tiempo posible en lugar de ejecutar a mayor velocidad para después tener el procesador parado. Este resultado, junto con los resultados que reducen la corriente de fuga a base de agrupar los espacios libres del procesador, es fácilmente aplicable a los algoritmos de planificación globales, en los que las tareas pueden ejecutarse en cualquier procesador. De manera que las tareas se deben ejecutar a una velocidad lo más uniformemente posible para tener al procesador ocupado el máximo tiempo posible.

Para concluir el presente trabajo, y como en cualquier desarrollo de investigación, somos conscientes de que todavía queda mucho por hacer en este ámbito. No se puede así concluir mas que comentando algunas de las posibles líneas futuras que se han dejado abiertas en esta tesis y que esperamos atacar en un futuro próximo:

1. Implementación de nuestros algoritmos en un núcleo real. Se ha realizado una primera aproximación modificando el RTLinux 3.1 para que la planificación se realice siguiendo el algoritmo PLMDP. Sin embargo, al no disponer de un sistema informático de tiempo real, se ejecutaron tareas sintéticas simuladas. El resultado fue satisfactorio, quedando pendiente la prueba con procesos reales.

2. Completar el estudio de la planificación teniendo en cuenta la sobrecarga debida a los algoritmos de planificación y la sobrecarga debida al cambio de velocidad del procesador. Ambas sobrecargas se han supuesto despreciables o que estaban incorporadas en el cálculo del peor tiempo de ejecución, aunque seria conveniente un análisis más exhaustivo de ambos efectos.

Asimismo, en el estudio de las sistemas distribuidos se debería tener en cuenta las sobrecargas debido a la red de interconexión ya que pueden llegar a ser importantes.

3. También sería interesante, realizar una planificación global incorporando al estudio general realizado, el estudio de la memoria caché y memoria principal, así como otros avances tecnológicos que se han desarrollado a nivel de la arquitectura del procesador.

4. Y por último, todo el estudio se ha realizado teniendo presente que el sistema informático a planificar son de tiempo real estricto, sin embargo, actualmente, los sistemas móviles ejecutan mayoritariamente sistemas multimedia, por lo que la mayoría de restricciones temporales no son estrictas. El problema principal que hallaríamos en este entorno seria la acotación no precisa de la utilización del sistema, ya que esta en los sistemas multimedia es variable, asimilándose a la presencia de tareas aperiódicas. En el capítulo 5 se ha visto que el tiempo de finalización a las tareas aperiódicas es aceptable consiguiendo ahorro energético, con lo que es previsible pensar que nuestros algoritmos podrían funcionar correctamente.

Se podría estudiar la adaptación a este tipo de sistemas, estudiando el porcentaje de restricciones temporales que no se cumplen, y que control de calidad obtendríamos.

ANEXOS

Anexo 1 Low Power Fixed Priority Algorithm Scheduling

El algoritmo "*Low Power Fixed Priority Algorithm Scheduling*" *(LPFPS)* propuesto por Shin y Choi [77] es un planificador de tiempo real con ahorro energético que se puede enmarcar en el conjunto de planificadores on-line con asignación de prioridades estáticas y cálculo de la velocidad de procesador dinámico. Este planificador es eficiente des del punto de vista de uso de procesador ,de memoria, y además su implementación es muy simple. La reducción energética se basa en utilizar las técnicas del DVS reduciendo la velocidad de ejecución del procesador cuando haya una única tarea lista para ser ejecutada o bien en detener el procesador cuando no hay tareas listas para su ejecución.

La implementación propuesta por Shin y Choi está basada en la implementación de los algoritmos de planificación con prioridades fijas propuestos por [15] [37] que consiste básicamente en usar dos colas de tareas (ver el esquema en la Figura 75): la "Cola de tareas ejecutables"(RDQ) ordenada según las prioridades estáticas, y la "Cola de tareas suspendidas" (DLQ) ordenada por el tiempo de activación de las tareas. En esta última cola estarán las tareas que están pendientes de la llegada de su periodo para ser activadas.

La información mínima necesaria, de cada tarea, que se requiere para la correcta implementación y ejecución del algoritmo es:

- Periodo de activación (T): separación temporal entre dos activaciones sucesivas de una tarea
- Plazo temporal relativo (D): tiempo máximo de finalización de una tarea
- Peor tiempo de ejecución (WCET) máximo tiempo que tardará la tarea en realizar sus cálculos
- Prioridad: escogida por el diseñador de la aplicación o el resultado de la aplicación de los algoritmos *Rate Monotonic* o *Deadline Monotonic*.
- Tiempo de activación: instante de llegada del próximo periodo
- Tiempo ya ejecutado: tiempo que la tarea se ha estado ejecutando: este tiempo será útil para poder calcular la capacidad de proceso que aun requiere la tarea.

El periodo, el plazo temporal, la prioridad base y el peor tiempo de ejecución vienen dados por las características propias de cada tarea. El tiempo de activación, se calculará dinámicamente sumando al periodo el tiempo de activación anterior, y siendo igual a cero en el instante inicial. El plazo temporal absoluto, será el tiempo de activación más el plazo temporal relativo, y por último, el cálculo del tiempo que lleva ejecutado la tarea activa debe actualizarse siempre que haya una expulsión de esta tarea, o bien cuando se active la tarea que será igualado a cero.

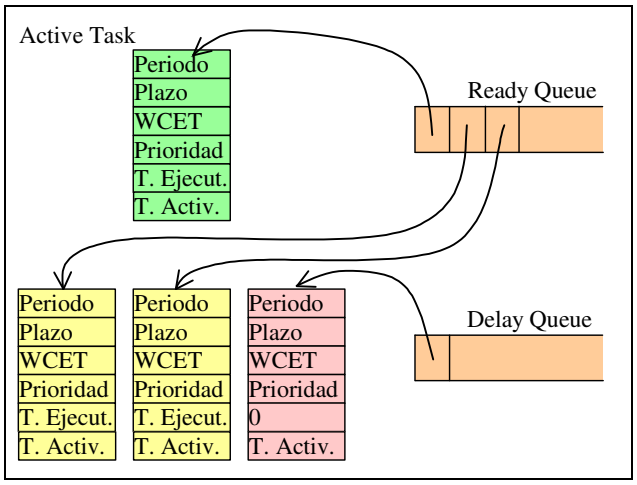

Figura 75: Esquema de las estructuras de datos necesarios para implementar el *Low Power Fixed Priority Algorithm Scheduling* (LPFPS).

El algoritmo de planificación puede implementarse en dos fases, la primera fase consiste en la elección de la tarea a ejecutar (tarea activa) y en la segunda fase, se realiza el cálculo de la velocidad de ejecución de la tarea activa con el consiguiente ahorro a nivel energético:

- La primera fase es exactamente la misma que podría usarse para implementar cualquier planificador expulsivo de prioridades fijas.
  - **Elección**. La tarea que se elegirá para ser ejecutada (tarea activa) será la primera de la cola de tareas ejecutables (RDQ). La activación de una tarea τk consiste en mover τk de la DLQ a la RDQ. Siendo a partir de este instante elegible para ser ejecutada.
  - **Apropiación**. Al activarse una tarea τ_k el algoritmo de planificación tiene que comprobar la posible apropiación del procesador comparando la prioridad de τ_k con la prioridad de la tarea activa τ_i. Si la prioridad de la tarea recién activada τ_k es mayor, esta se apropiará del procesador

expulsando del procesador a τ_i. A partir de este instante la tarea activa pasará a ser τ_k.

- La segunda fase es la que nos aporta el ahorro energético. El pseudocódigo del planificador se haya en la Figura 76.
  - Cuando en la RDQ exista únicamente la tarea activa, el planificador, antes de iniciar o continuar su ejecución se deberá calcular a que velocidad mínima de procesador puede ejecutarla sin comprometer ningún plazo con el fin de reducir la energía total consumida. En este caso, para el cálculo de la velocidad del procesador se empleará la siguiente formula:

$$Velocidad = \frac{min(Ta_k - tc, R_i)}{min(Ta_k, Td_i) - tc} \qquad \text{(ec. 18)}$$

donde Ta_k es el tiempo de activación de la primera tarea de la DLQ, tc es el tiempo actual, Td_i es el plazo de la tarea activa (tarea que se va a ejecutar) y por última R_i es el tiempo máximo de procesador a velocidad máxima que le falta a la tarea activa para finalizar su ejecución (WCET-tiempo ya ejecutado por la tarea).

  - Cuando existan varias tareas en la RDQ listas para ser ejecutadas, la tarea activa deberá ejecutarse a velocidad máxima, puesto que es la manera más simple de garantizar que todas las tareas del sistema cumplirán su plazo temporal.
  - Cuando no existan tareas listas para ser ejecutadas (la RDQ está vacía) se detendrá el procesador hasta el instante de activación de alguna tarea menos el tiempo necesario para que el procesador se active a su máxima velocidad. Dado que en el modelo de tiempo real propuesto todas las tareas son periódicas, se puede saber el momento en que se activará la primera tarea de la DLQ.

```
L1 si no vacia(RDQ) entonces
L2      Tarea activa = RDQ.head; // active task=τi
L3      si RDQ.head.next == NIL entonces   // RDQ.head.next=τk
L4              velocidad = min(Tak − tc, Ri) / min(Tak, Tdi) − tc
L5      sino
L6              velocidad = Máxima velocidad;
L7      finsi
L8 sino
L9      prog. Temporizador a (next Tak – wake up delay);
L10     Ponerse en modo power-down;
L11 finsi
L12 ejecutar la tarea activa a la velocidad calculada;
```

Figura 76: Pseudocódigo para el cálculo de la velocidad del procesador con el algoritmo de Low Power Fixed Priority Scheduling (LPFPS).

A continuación se muestra en un ejemplo como se planificarían las tareas de un conjunto de tareas muy simple especificado en la Tabla 1 usando el planificador *Fixed Priority Scheduling Algorithm* (FPS), y el planificador *Low Power Fixed Priority Scheduling* (LPFPS).

La representación de la planificación se realizará durante todo un hiperperiodo (mínimo común múltiplo de todos los periodos de las tareas del sistema) y suponiendo que todas las tareas consumen el 100% de su WCET. Las flechas hacia arriba indican el instante de activación de cada tarea, las flechas hacia abajo indican el plazo temporal de la tarea. En la Figura 77 se muestra el comportamiento del FPS, y en la Figura 78 se muestra el comportamiento del LPFPS. En las figuras, los triángulos hacia abajo representan la activación de las tareas en el sistema y, al ser el periodo igual al plazo temporal, también representa el plazo temporal de la tarea activada con anterioridad, cada rectángulo representa la ejecución de 5 unidades de tiempo, y por último el espacio en blanco representa el tiempo libre del procesador. En ambos casos, se ha supuesto que en el instante 0 (inicial) se activan todas las tareas del conjunto es por tanto el instante crítico del sistema.

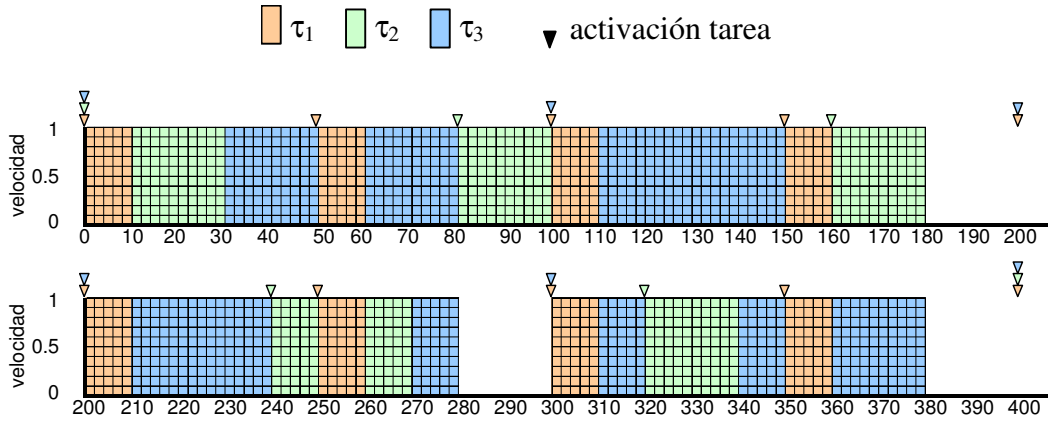

Figura 77: Diagrama de planificación de procesos usando el algoritmo FPS consumiendo el 100% del WCET.

En la Figura 77 se puede observar el resultado de la planificación del conjunto de tareas que se muestra en la Tabla 1 En el instante t=0 se activan todas las tareas del sistema (instante crítico) encolándose todas ellas en la cola de preparados (RDQ) ordenadas según su prioridad. El planificador escoge la primera tarea de la cola τ_1 y le pasa el control para que se ejecute terminando su ejecución en t=10 consumiendo todo el $WCET_1$. Cuando ha finalizado, se inserta en la cola de suspendidos (DLQ) ordenada según el tiempo de activación absoluto calculado como la suma del tiempo de activación anterior más el periodo de la tarea. A continuación el planificador escoge a τ_2, que también se ejecuta hasta el final de su WCET, se encola en la DLQ y el planificador escoge a τ_3 que se ejecuta hasta que llega la activación de τ_1 que al ser de mayor prioridad que la tarea activa (τ_3) le expulsa del procesador y se pasa a ejecutar τ_1 hasta que esta finaliza y se reanuda la ejecución de τ_3. En t=180 la RDQ está vacía, con lo que se deja libre al procesador hasta la activación de la τ_1 y de la τ_3. El procesador vuelve a quedarse libre en t=280 y en t=380. En total, el procesador ha estado libre durante 60 unidades de tiempo. El hiperperiodo del conjunto de tareas es 400, esto implica que en t=400 vuelven a activarse todas las tareas del conjunto repitiéndose el mismo esquema de planificación de tareas.

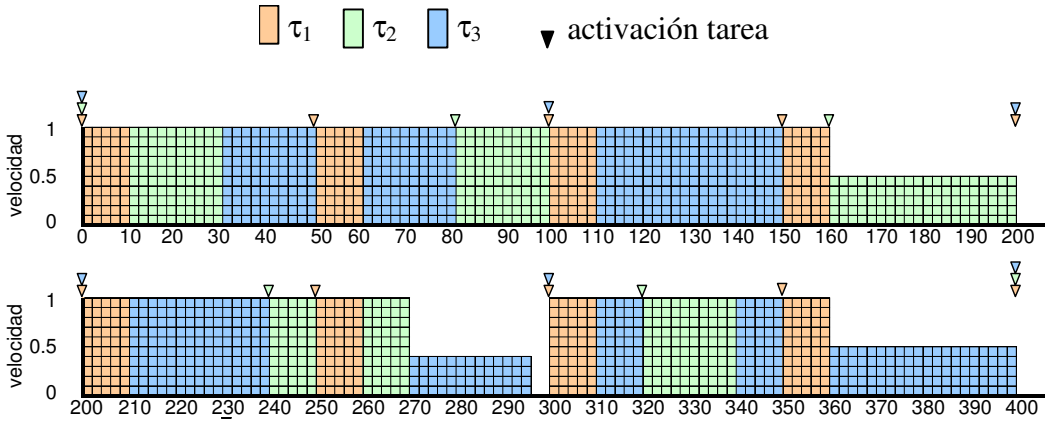

Figura 78: Diagrama de planificación de procesos usando el algoritmo LPFPS.

En la Figura 78 se muestra como sería la ejecución de las tareas siguiendo el algoritmo LPFPS cuando estas consumen el 100% de su peor tiempo de ejecución. En el instante t=0, llegan al sistema las tareas τ_1, τ_2 y τ_3. Empieza a ejecutarse la tarea τ_1 debido a que es la tarea con mayor prioridad, esta se ejecuta durante 10 unidades de tiempo y el algoritmo elige la τ_2 para su ejecución. τ_2 consume 20 unidades de tiempo y al finalizar el planificador elige τ_3 para su ejecución. Al ser esta la única tarea que se tiene lista para ser ejecutada, se intenta reducir la velocidad de ejecución del procesador aplicando la ecuación (ec. 18) La primera tarea que se activará a partir de tc=30, será τ_1 en el instante t=50, por tanto Ta_1=50, Ta_1-tc = 20, lo que implica que τ_3 estará sola en la RDQ durante 20 unidades de tiempo, pero como R = 40, el mínimo será 20, en el denominador de la formula aparece Td_1= 100, por tanto el mínimo es 50, si se resta tc queda 20. 20 dividido entre 20 es 1 por tanto no se puede reducir la velocidad de ejecución de τ_3, y se ejecuta a velocidad máxima. Al llegar τ_1 y ser de mayor prioridad que τ_3 se apodera del procesador produciéndose una expulsión. Al finalizar τ_1 en el instante t=60, reanuda la ejecución de τ_3. Aquí tampoco se puede ejecutar a menor velocidad, puesto que llegará τ_2 justo cuando τ_3 acabe su ejecución. Se continua ejecutando de la misma manera hasta el instante t=160 en que nos llega τ_2, al ser la única tarea en la RDQ hasta la llegada de la siguiente tarea τ_1, en t=200 se realiza de nuevo el cálculo del la velocidad del procesador. Entre tc=160 y ta_1=200 hay 40 unidades de tiempo en que en la RDQ habrá únicamente τ_2 lista para ser ejecutada. τ_2 necesita como máximo 20 unidades de tiempo para finalizar su WCET, por tanto se puede ejecutar τ_2 al 50% de la velocidad máxima sin comprometer ninguna restricción temporal y sin dejar tiempo

libre al procesador con lo que se consigue ahorrar energía respecto a la ejecución a máxima velocidad. Este mismo escenario se tendrá en t=270 y en t=360. Reduciendo en estos casos la velocidad de τ_3.

Para comprobar el ahorro energético que se consigue con este planificador, se compara la energía necesaria para ejecutar varios conjuntos de tareas planificadas con el LFPS con la energía necesaria para ejecutar estos conjuntos de tareas planificados con el *"Fixed Priority Scheduling Algorithm" (FPS)*, es decir, ejecutar siempre a máxima velocidad del procesador cuando haya tareas a ejecutar, y deteniendo al procesador cuando no exista ninguna tarea lista para ser ejecutada. En el caso del conjunto de tareas de la Tabla 1 ejecutando siempre la totalidad del WCET, comparando la energía utilizada por las tareas según la planificación LPFPS (295.393) frente a la planificación FPS (340) se obtiene un ahorro energético del 13,12%. Los resultados obtenidos llevan a la conclusión de que la reducción energética que se obtiene con este algoritmo, no depende de la utilización del procesador, sino más bien de la relación entre los periodos de las tareas y de la distribución de la carga del procesador entre las tareas, de manera que se obtiene una mayor reducción cuando el procesador esta ocupado por pocas tareas con periodos grandes en comparación con el WCET.

Las ventajas que presenta el algoritmo "*Low Power Fixed Priority Scheduling"* son:

- Implementación sencilla y eficiente en el uso de procesador y de memoria.
- Permite predecir la planificabilidad del conjunto de tareas de manera estática antes de iniciar la ejecución del sistema de tiempo real, garantizando el cumplimiento de todos los plazos temporales de las tareas durante la ejecución del sistema.
- Es un algoritmo estable frente a situaciones de sobrecarga del sistema, es decir, en el caso extremo que alguna tarea consuma más tiempo que el *WCET* calculado, se puede asegurar que en el caso que no se puedan ejecutar todas las tareas cumpliendo sus plazos temporales, la tarea o tareas que perderán el plazo serás las que tengan una menor prioridad, nunca perderá el plazo una tarea de prioridad mayor a la que ha provocado la sobrecarga del procesador.
- Aprovechamiento implícito del *slack* dinámico, si una tarea finaliza sin haber consumido todo su WCET permite de manera implícita que la siguiente tarea

pueda empezar con anterioridad, con lo que acabara antes y dejara durante más tiempo una única tarea activa, con lo que se podrá reducir la velocidad del procesador.

El inconveniente principal que presenta el algoritmo es que e cálculo de velocidad del procesador se realiza suponiendo que la última tarea consumirá todo el WCET, sin embargo esta suposición no siempre será cierta. Si está tarea finaliza sin llegar a consumir todo su WCET, como no existirá ninguna tarea que pueda ejecutarse el procesador quedará libre y se detendrá, hasta la activación de una nueva tarea

.

Anexo 2 Dual Priority Algorithm.

El algoritmo de prioridad dual "*Dual Priority Scheduling Algorithm"(DPS)* propuesto originalmente por Burns y Wellings [11] para conseguir maximizar la planificación de tareas periódicas, y modificado posteriormente por Davis y Wellings [12] para mejorar el tiempo de finalización para las tareas aperiódicas. Es el algoritmo base que se usa para la implementación de nuestro planificador de ahorro energético.

La idea del algoritmo de prioridad dual consiste, tal como ya se ha indicado anteriormente, en retardar las tareas periódicas sin llegar a comprometer los plazos temporales, dando al mismo tiempo un buen servicio a las tareas aperiódicas que lleguen al sistema. Para eso, las tareas periódicas constan, como su nombre indica, de dos niveles distintos de prioridades, en el nivel de prioridad superior "*Upper Run Queue*" (URQ), únicamente podrán recibir interferencias de tareas periódicas de mayor prioridad, y en el nivel de prioridad inferior "*Lower Run Queue*" (LRQ) podrán recibir interferencias de las tareas aperiódicas y de las tareas periódicas de mayor prioridad. Las tareas aperiódicas se hayan en un nivel intermedio (MRQ), ordenadas mediante un algoritmo FIFO (*First In First Out*), es decir el primero en entrar es el primer en salir. En la Figura 79 se ve la estructura de datos mínima que se propone para la implementación del *Dual Priority Scheduling Algorithm*.

La información mínima para cada tarea periódica que necesita este algoritmo de planificación es la siguiente:

- Periodo de activación (T): separación temporal entre dos activaciones sucesivas de una tarea
- Plazo temporal (D): tiempo máximo de finalización de una tarea.
- Peor tiempo de ejecución (WCET) máximo tiempo que tardará una tarea en realizar sus cálculos
- Peor tiempo de finalización (WCRT): Tiempo máximo de finalización teniendo en cuenta la máxima interferencia de todas las tareas periódicas de mayor prioridad.

- Prioridad: escogida por el diseñador de la aplicación o el resultado de la aplicación de los algoritmos *Rate Monotonic* o *Deadline Monotonic*.
- Tiempo de activación: instante de llegada del próximo periodo
- Tiempo ejecutado: tiempo que la tarea se ha estado ejecutando: este tiempo será útil para poder calcular la capacidad de procesador que aun requiere la tarea.
- Promoción, instante en que la tarea cambiará de cola desde LRQ a URQ. Se puede calcular off-line y es plazo temporal absoluto menos el peor tiempo de finalización.

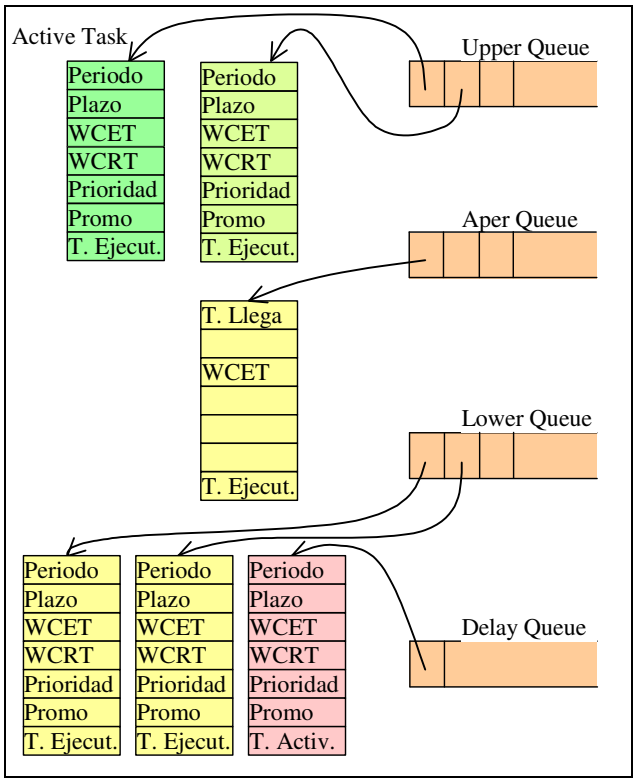

Figura 79: Esquema de las estructuras de datos necesarios para implementar el *Dual Priority Scheduling Algorithm*.

El periodo, el plazo temporal, la prioridad base y el peor tiempo de ejecución viene dado por las características propias de cada tarea. Para calcular el tiempo máximo de finalización de una tarea i τ_i (*WCRTi*) se utilizará la formula recursiva de Joseph y Pandya [36].

El tiempo que lleva ejecutado la tarea activa debe actualizarse siempre que haya una expulsión de esta tarea, o bien cuando se active la tarea que será igualado a cero. Y por último, el tiempo de promoción es el instante en que una tarea cambiará de la LRQ a la URQ, el cálculo se puede realizar antes de iniciar la aplicación siendo su valor igual al plazo temporal menos el WCRT de la tarea en cuestión.

La implementación de este algoritmo al igual que la implementación del FPS también es simple y eficiente en el uso de memoria y de procesador. Para su implementación hay que establecer tres capas de prioridades URQ (*Upper Run Queue*), MRQ (*Middle Run Queue*) y LRQ (*Lower Run Queue*). En cada una de estas capas se emplazan las instancias de las tareas siguiendo un orden de prioridades fijo para las tareas periódicas y FIFO para las tareas aperiódicas. El planificador escogerá para que se ejecute la primera tarea de la capa URQ, o si esta está vacía de la MRQ, o si también esta vacía de la LRQ en este orden. En la capa superior, tal como se ha especificado anteriormente, se encuentran aquellas tareas periódicas que no pueden ser retardadas sin comprometer su plazo temporal, en la inferior el resto de tareas periódicas que si pueden esperar y en la capa intermedia entraran las tareas aperiódicas. Las tareas periódicas cuando se activan se encolan en la LRQ y promocionan a la URQ en el instante de promoción (plazo temporal menos el peor tiempo de finalización de la tarea), instante en el que la ejecución de la tarea no puede verse retardada por tareas que no se hallen en su nivel de prioridad, para poder garantizar el cumplimiento de su plazo temporal, ya que puede darse el caso de llegar a tener la máxima interferencia por parte de las tareas de mayor prioridad de la aplicación. Cuando se activa una tarea aperiódica se encola según el orden FFIO en la MRQ. Por tanto, siempre que no existan tareas periódicas en la URQ se ejecutaran (si existen) tareas aperiódicas, retardando así la ejecución de las tareas periódicas que se hallen en la LRQ que son las que no tienen prisa para ser ejecutadas.

El esquema del algoritmo de planificación DP se muestra en la Figura 80. Este algoritmo es el que decide que tarea se debe ejecutar en cada momento, teniendo como característica más relevante el uso del tiempo de promoción de la LRQ a la URQ como parámetro clave de su funcionamiento.

```
L1  si no vacia(URQ) entonces
L2       Tarea activa ← URQ.head;
L3  sino
L4       si no vacia(MRQ) entonces
L5            tarea activa ← MRQ.head;
L6       sino
L7            si no vacia(LRQ) entonces
L8                 tarea activa ← LRQ.head;
L9                 Tpi ← Tai + Promoi;        //Promoi=Di-WCRTi
L10           sino
L11                tarea activa ← null;
L12           finsi;
L13      finsi;
L14 finsi;
L15 Ejecutar la tarea activa
```

Figura 80: Pseudocódigo del planificador Dual Priority de [23].

Cuando una tarea periódica llega al sistema tiene el nivel menor de prioridad posible, y promociona al nivel superior, cuando de no empezar su ejecución pondría en peligro la consecución de su plazo temporal. Si existe alguna tarea periódica promocionada al nivel de prioridades superior, no se permite la ejecución de ninguna tarea aperiódica, ni periódica de nivel inferior, de no existir tareas periódicas en el nivel de prioridad superior, serán las tareas aperiódicas las que se ejecutarán y si tampoco las hay, se ejecutaran las tareas periódicas del nivel inferior de prioridades.

La implementación del planificador está dirigida por los siguientes eventos:

- Activación de una tarea τ_i al sistema (Ta_i):
  - si la tarea es periódica se emplaza en la LRQ siguiendo el orden de prioridad fijado durante el diseño de la aplicación, se calcula el instante de promoción (Tp_i), y finalmente se programa la llegada de la siguiente activación, sumando al Ta_i el periodo de la tarea τ_i.
  - si la tarea es aperiódica se emplaza en la MRQ en orden FIFO, orden de activación de la tarea al sistema.

    En ambos casos puede tener lugar la apropiación del procesador por una de estas nuevas tareas debido a la prioridad de la tarea recién activada respecto a la tarea activa.
- Llegada de la promoción de una tarea τ_i (Tp_i):

La tarea en cuestión, promociona de la LRQ a la URQ. En este caso también se puede producir la expulsión de la tarea activa del procesador.

- Llegada del plazo temporal de una tarea τ_i (Td_i):

Si la tarea no ha finalizado el sistema la aborta y muestra un aviso de violación de restricción temporal dando lugar a las acciones oportunas. Al ser el sistema de tiempo real estricto, cuando se produzca este evento la instancia de la tarea que lo ha generado tiene que haber finalizado. Este caso podría darse en casos de sobrecarga del sistema, pero al ser este un algoritmo estable, las tareas que podrían perder el plazo serían las que posean menos prioridad que la tarea que ha generado la sobrecarga.

A continuación se verá un ejemplo de su funcionamiento usando el conjunto de tareas periódicas definido en la Tabla 1 más cuatro tareas aperiódicas de WCET=1 y tiempo de llegada al sistema de t=0, t=60, t=190 y finalmente t=260.

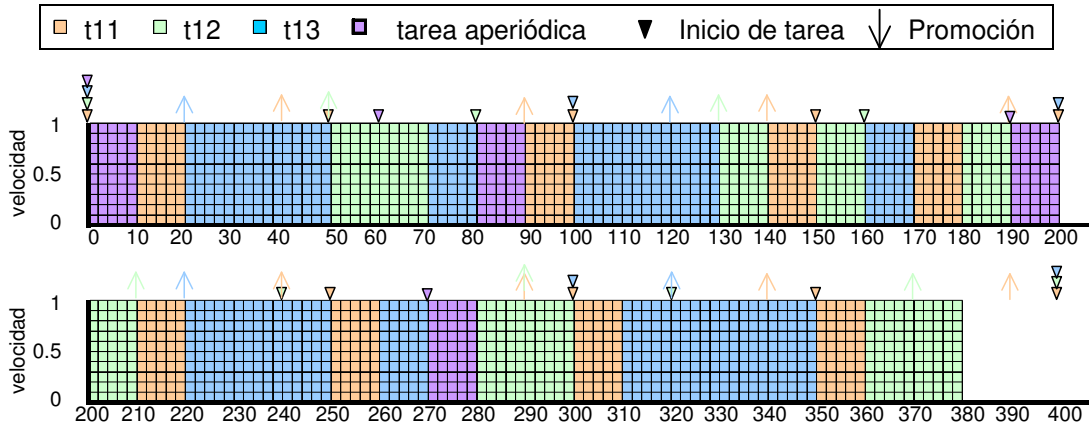

Figura 81: Diagrama de planificación de procesos usando el algoritmo DPS.

En la Figura 81 se muestra como sería la ejecución de las tareas siguiendo el algoritmo DPS cuando estas consumen el 100% de su peor tiempo de ejecución. En el instante t=0, llegan al sistema las tareas τ_1, τ_2, τ_3 y α_1. Las tareas periódicas se encolan en la URQ y la tarea aperiódica en la MRQ. Empieza a ejecutarse la tarea α_1 debido a que es la tarea que se halla en la cola de prioridad media y la URQ está vacía. Esta tarea consume diez unidades de tiempo y finaliza. A continuación, como que solo hay tareas en la LRQ, se escoge la tarea de mayor prioridad de esta cola que es τ_1. Esta tarea se ejecuta durante 10 unidades de tiempo, y también finaliza. En t=20, llega el instante de promoción de τ_3 que promociona a la URQ. En este

momento el algoritmo escoge esta tarea para ser ejecutada, cuando en t=40 llega la promoción de τ_1 está ya ha finalizado, por tanto no promociona y continua ejecutándose τ_3, en t=50, se activa la tarea τ_1 que se encola en la LRQ y promociona τ_2 que se apropia del procesador, ya que su prioridad es mayor que la prioridad de τ_3. En t=60 llega otra aperiódica al sistema, pero debido a que en la URQ hay τ_2 y τ_3 esta no puede ejecutar-se hasta que estas tareas finalicen en t=80 resultando un tiempo de finalización de esta tarea aperiódica de 30 unidades de tiempo. En t=120, ya se estaba ejecutando τ_3 cuando le llega su instante de promoción, se la promociona y continua con su ejecución. En t=130, promociona τ_2 que pasa a ser la tarea activa, y cuando en t=140 promociona τ_1 expulsa a τ_2 del procesador, pasando a ser ella la tarea activa hasta que finaliza τ_1 y continua con la ejecución de τ_2. En t=190 llega otra tarea aperiódica y como no hay tareas en la URQ puede pasar a ejecutarse inmediatamente, obteniendo un tiempo de finalización de 10 unidades. Lo mismo que en t=270. En t=380 finaliza τ_2, que era la única tarea activa y por tanto se deja libre al procesador hasta t=400, en que se vuelven a activar todas las tareas periódicas, empezando otro hiperperiodo.

La ventaja principal del funcionamiento de este algoritmo de planificación es que permite que las tareas periódicas retrasen su ejecución el máximo tiempo posible (teniendo en consideración la máxima interferencia que pueden tener estas tareas debido a las tareas de mayor prioridad), siempre que exista alguna tarea aperiódica en el sistema. De manera que si no es imprescindible la ejecución de tareas periódicas, se puede ejecutar tareas aperiódicas ofreciéndoles una mayor calidad de servicio. En el momento en que no existen tareas aperiódicas para ser ejecutadas, se van ejecutando las tareas periódicas, sin ser necesario retrasar su ejecución.

La dificultad principal de este algoritmo es el cálculo del tiempo máximo de finalización, que está calculado como una cota superior, en el peor de los casos, una tarea tendrá esta interferencia máxima, pero normalmente, la interferencia es siempre menor. Si se mejora esta cota, acercándola más a la interferencia real, los resultados que se obtendría a nivel de tiempo de finalización a las tareas aperiódicas mejorarían.

Anexo 3 Low Power EDF

El algoritmo "Low Power EDF." (LPEDF) está basado en el "*Feedback DVS-EDF*" *(FEDF)* propuesto por Zhu and Mueller [87] pero sin realimentación, es un planificador de tiempo real con ahorro energético que se puede enmarcar en el conjunto de planificadores *on-line* con asignación de prioridades dinámicas y cálculo de la velocidad de procesador dinámico. Este planificador es eficiente des del punto de vista de uso de procesador, de memoria, y además su implementación es simple. La reducción energética se basa en utilizar las técnicas del DVS reduciendo la velocidad de ejecución del procesador al máximo siempre que se disponga de tiempo libre. Satisfaciendo que para que un sistema basado en EDF sea planificable es suficiente con que la utilización del procesador sea menor o igual a 1. El hecho de reducir al máximo la tarea actual se basa en la idea en que el WCET no se consume en su totalidad, por tanto la tarea generara tiempo libre que podrá ser aprovechado para reducir la velocidad de la siguiente tarea que se ejecute.

En esta implementación se intentará usar todo el tiempo libre para reducir velocidad de proceso. De tiempo libre hay de dos tipos:

- Estático, el que no dejan las tareas debido a la no utilización total del procesador.
- Dinámico, el que se genera debido a la no utilización del WCET por parte de las tareas. (*slack dinámico*)
- Para implementar este algoritmo se creará una nueva tarea que será la encargada de repartir el tiempo libre disponible entre las tareas que lo necesiten para reducir velocidad. La nueva tarea tendrá el periodo igual a la tarea de menor periodo y el WCET necesario para llegar a que el procesador tenga el 100% de utilización. Sin embargo el tiempo de cálculo de esta tarea será siempre 0 de manera que únicamente producirá *slack dinámico* para el uso de las tareas reales.
- El *slack* no usado por una tarea puede pasarse a la siguiente tarea, siempre que la utilización de este por parte de la segunda tarea no exceda el plazo temporal del que lo ha generado.

- En el algoritmo original, la tarea se ejecuta en dos partes, una a velocidad reducida, haciendo uso del máximo *slack* dinámico generado por la tarea posterior, y el resto a máxima velocidad, de manera que si se pudiera adivinar el porcentaje de WCET que se consume por la tarea, esta podría ejecutarse a velocidad reducida, generando como *slack* dinámico la parte no consumida a máxima velocidad. La forma de dividir la tarea en dos partes, se retro-alimenta, de manera que si en la ejecución anterior la tarea ha finalizado a velocidad reducida, se disminuye la primera parte, y si la tarea finaliza cuando se ejecuta a máxima velocidad se aumenta la primera parte. De esta manera, si se tienen las tareas bien calibradas, la primera tarea que tenga *slack* disponible lo utilizará todo, y nunca llegará a ejecutarse a máxima velocidad. Ella, a su vez generará nuevo *slack* para la siguiente tarea que se ejecute.

La implementación propuesta por está basada en la implementación de los algoritmos de planificación con prioridades dinámicas, que consiste al igual que el LPFPS básicamente en usar dos colas de tareas (ver el esquema en la Figura 75): la "Cola de tareas ejecutables"(RDQ) ordenada según el plazo temporal absoluto, y la "Cola de tareas suspendidas" (DLQ) ordenada por el tiempo de activación de las tareas. En esta última cola estarán las tareas que están pendientes de la llegada de su periodo para ser activadas.

La información mínima necesaria, de cada tarea, que se requiere para la correcta implementación y ejecución del algoritmo es:

- Periodo de activación (T): separación temporal entre dos activaciones sucesivas de una tarea
- Plazo temporal relativo (D): tiempo máximo de finalización de una tarea
- Peor tiempo de ejecución (WCET) máximo tiempo que tardará la tarea en realizar sus cálculos
- Prioridad: asignada dinámicamente en función del plazo temporal absoluto.
- Tiempo de activación: instante de llegada del próximo periodo
- Tiempo ya ejecutado: tiempo que la tarea se ha estado ejecutando: este tiempo será útil para poder calcular la capacidad de proceso que aun requiere la tarea.

El periodo, el plazo temporal y el peor tiempo de ejecución vienen dados por las características propias de cada tarea. El tiempo de activación, se calculará dinámicamente sumando al periodo el tiempo de activación anterior, y siendo igual a cero en el instante inicial. El plazo temporal absoluto, será el tiempo de activación más el plazo temporal relativo, y por último, el cálculo del tiempo que lleva ejecutado la tarea activa debe actualizarse siempre que haya una expulsión de esta tarea, o bien cuando se active la tarea que será igualado a cero.

El algoritmo de planificación puede implementarse en dos fases, la primera fase consiste en la elección de la tarea a ejecutar (tarea activa) y en la segunda fase, se realiza el cálculo de la velocidad de ejecución de la tarea activa con el consiguiente ahorro a nivel energético:

- La primera fase es exactamente la misma que podría usarse para implementar cualquier planificador expulsivo de prioridades (fijas o dinámicas).
  - **Elección**. La tarea que se elegirá para ser ejecutada (tarea activa) será la primera de la cola de tareas ejecutables (RDQ). La activación de una tarea τ_k consiste en mover τ_k de la DLQ a la RDQ, calculando el plazo temporal absoluto y su prioridad asociada. Siendo a partir de este instante elegible para ser ejecutada.
  - **Apropiación**. Al activarse una tarea τ_k el algoritmo de planificación tiene que comprobar la posible apropiación del procesador comparando la prioridad de τ_k con la prioridad de la tarea activa τ_i. Si la prioridad de la tarea recién activada τ_k es mayor, esta se apropiará del procesador expulsando del procesador a τ_i. A partir de este instante la tarea activa pasará a ser τ_k.
- La segunda fase es la que nos aporta el ahorro energético. El pseudocódigo del planificador se haya en la Figura 82.
  - Cuando exista slack disponible (producido por una tarea finalizada sin consumir todo el WCET o bien debido a la tarea libre), se calculará la velocidad de ejecución de esta tarea con la siguiente formula:

$$Velocidad = \frac{Falta_i}{\min(Falta_i + Slack, Td_i)} \qquad \text{(ec. 19)}$$

donde *slack* es todo el *slack* disponible para la tarea activa (tarea que se va a ejecutar), Td_i es el plazo temporal y F_i es el tiempo máximo de procesador a velocidad máxima que le falta a la tarea para finalizar su ejecución (WCET-tiempo ya ejecutado por la tarea).

- Una vez la tarea deja de ejecutarse por finalización o expulsión, el algoritmo actualiza la variable F, y la variable *slack* restándole el usado en la ejecución de la tarea y el no aprovechable y sumándole el *slack* generado por la propia tarea al finalizar antes de consumir todo su WCET.

```
L1 si no vacia(RDQ) entonces
L2      Tarea activa = RDQ.head;      // tarea activa=τi
L3            velocidad = Falta_i / min(Falta_i + Slack, Td_i)
L4 sino
L5      progr. Temporizador a (next Tak - wake up delay);
L6      poner el procesador en modo power-down;
L7 finsi
L8 Ejecutar la tarea activa a la velocidad calculada;
```

Figura 82: Pseudocódigo para el cálculo de la velocidad del procesador con el algoritmo de *Low Power Earliest Deadline First Scheduling* (LPEDF).

A continuación se muestra en un ejemplo como se planificarían las tareas de un conjunto de tareas muy simple especificado en la Tabla 1 usando el planificador "*Earliest Deadline First Scheduling Algorithm"* (EDF), y el planificador de "*Low Power Earliest Deadline First Scheduling* "(LPEDF).

La representación de la planificación se realizará durante todo un hiperperiodo (mínimo común múltiplo de todos los periodos de las tareas del sistema) y suponiendo que todas las tareas consumen el 100% de su WCET. Las flechas hacia arriba indican el instante de activación de cada tarea, las flechas hacia abajo indican el plazo temporal de la tarea. En la Figura 83 se muestra el comportamiento del EDF, y en la Figura 84 se muestra el comportamiento del LPEDF. En las figuras, los triángulos hacia abajo representan la activación de las tareas en el sistema y, al ser el periodo igual al plazo temporal, también representa el plazo temporal de la tarea activada con anterioridad, cada rectángulo representa la ejecución de 5 unidades de tiempo, y por último el espacio en blanco representa el tiempo libre del procesador.

En ambos casos, se ha supuesto que en el instante 0 (inicial) se activan todas las tareas del conjunto es por tanto el instante crítico del sistema.

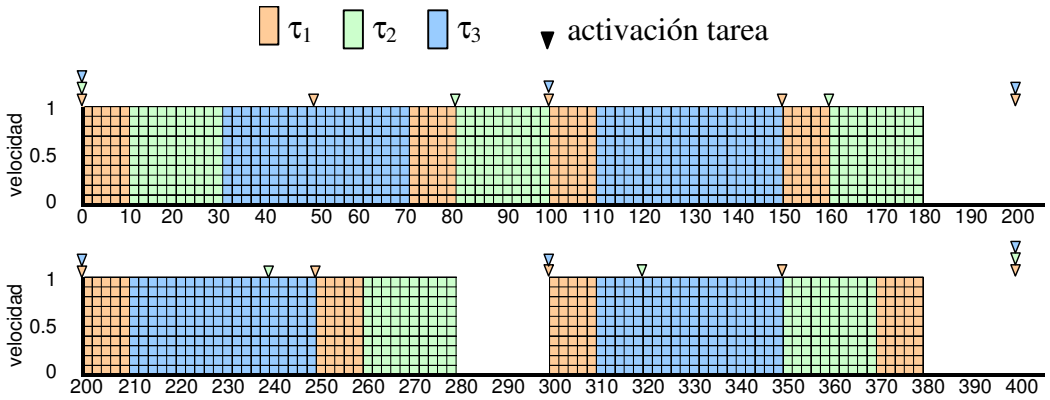

Figura 83: Diagrama de planificación de procesos usando el algoritmo EDF.

En la Figura 83 se puede observar el resultado de la planificación del conjunto de tareas que se muestra en la Tabla 1 cuando estas consumen el 100% de su peor tiempo de ejecución. En el instante t=0, llegan al sistema las tareas τ_1, τ_2 y τ_3. Empieza a ejecutarse la tarea τ_1 debido a que es la tarea con mayor prioridad, esta se ejecuta durante 10 unidades de tiempo y cuando finaliza, el algoritmo de planificación elige la τ_2 para su ejecución, ya que posee mayor prioridad que τ_3. τ_2 consume 20 unidades de tiempo y al finalizar el planificador elige τ_3 para su ejecución. Al cabo de 20 unidades de tiempo se activa τ_1 de nuevo, pero ahora tendrá menor prioridad, por tanto τ_3 sigue con su ejecución hasta que finaliza. En t=180 finaliza τ_2 que es la última tarea activa, dejando la libre al procesador hasta el instante t=200 en que llegan τ_1 y τ_3. Otro instante en el que el procesador queda libre es en t=280 y en t=380. En total, el procesador tiene 60 unidades de tiempo libre, que se hubieran podido aprovechar para reducir la velocidad de ejecución de las tareas, reduciendo la energía total utilizada.

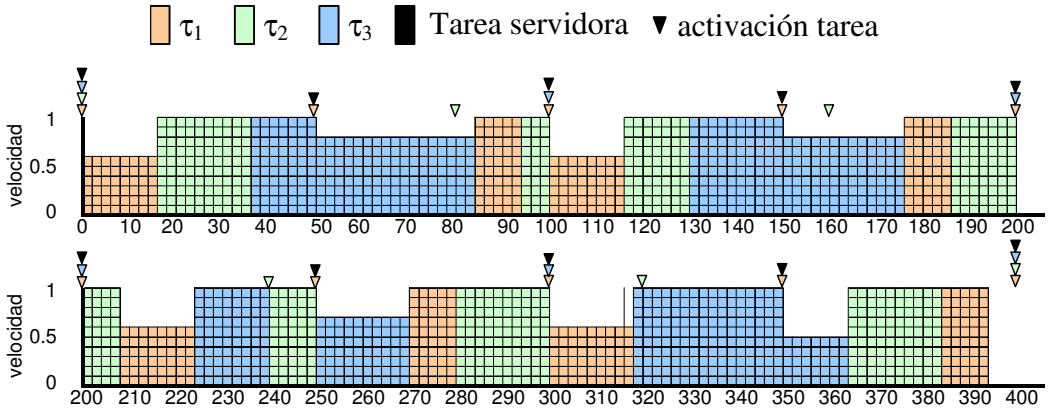

Figura 84: Diagrama de planificación de procesos usando el algoritmo LPEDF.

En la Figura 84 se puede observar el resultado de la planificación del conjunto de tareas que se muestra en la Tabla 1. La tarea libre que se ha añadido para proporcionar *slack* dinámico tiene periodo 50 y WCET de 7.5, sin embargo, su tiempo de ejecución real siempre será 0. En el instante t=0 se activan todas las tareas del sistema (instante crítico) encolándose todas ellas en la cola de preparados (RDQ) ordenadas según su prioridad (llegada del plazo absoluto), teniendo en cuenta que en caso de empate la tarea libre será la de mayor prioridad. Por tanto se escoge la tarea libre que finaliza sin consumir tiempo pero añade a *slack* su WCET, *slack* ahora vale 7.5 hasta que se consuma o hasta su plazo. A continuación, el planificador escoge la primera tarea de la cola τ_1 y le pasa el control para que se ejecute usando todo el *slack* disponible, se ejecuta consumiendo todo el $WCET_1$ a velocidad de procesador de 0,87, finalizando su ejecución en el instante t=17.5. Cuando finaliza, se inserta en la cola de suspendidos (DLQ) ordenada según el tiempo de activación absoluto calculado como la suma del tiempo de activación anterior más el periodo de la tarea. A continuación el planificador escoge a τ_2, que también se ejecuta hasta el final de su WCET, en este caso no hay *slack*, por tanto debe ejecutarse a máxima velocidad. Al finalizar, se encola en la DLQ y el planificador escoge a τ_3, en el momento en que llega la activación de la tarea libre y τ_1, y como estas tienen menor prioridad se continua con la ejecución de τ_3 hasta que esta finaliza. Y así se continua hasta la llegada del hiperperiodo en el que se volverá a repetir la misma historia de ejecución de tareas. En total, el procesador no ha tenido tiempo libre, puesto que todo se ha usado para reducir la velocidad de proceso, y con ello se ha reducido la energía consumida por el sistema.

- Para comprobar el ahorro energético que se consigue con este planificador, se comparará la energía necesaria para ejecutar varios conjuntos de tareas planificadas con el LFPS con la energía necesaria para ejecutar estos conjuntos de tareas planificados con los planificadores detallados en los anexos anteriores (EDF,LPFPS). Si se supone que siempre se detiene el procesador cuando no exista ninguna tarea lista para ser ejecutada, en el caso del conjunto de tareas de la Tabla 1 ejecutando siempre la totalidad del WCET, comparando la energía utilizada por las tareas según la planificación LPFPS/LPEDF es del 0,916, mientras que EDF/LPEDF es del 0,812. Los resultados obtenidos llevan a la conclusión de que la reducción energética que se obtiene con este algoritmo, es mejor que la obtenida con el LPFPS tal

como era de esperar. Y también es de implementación simple. Sin embargo, el algoritmo posee las desventajas de la planificación con prioridades dinámicas, por ejemplo, en caso de sobrecarga no se puede determinar que tareas perderán el plazo temporal.

GLOSARIO

Algoritmos de planificación expulsivos: Cuando se activa una tarea de mayor prioridad que la tarea que se está ejecutando la nueva tarea se apodera del procesador.

Breakdown utilization: Medida que indica el grado de planificabilidad de un conjunto de tareas. Se calcula de la siguiente manera: Cada $WCET_i$ del conjunto de tareas es multiplicado por un factor de escalado, mientras que los periodos y plazos temporales no se modifican. Todo el conjunto es escalado hasta el instante en que si hubiera un pequeño incremento adicional se perdería algún plazo temporal. El Breakdown utilization es la utilización del procesador en este punto $\Sigma_i (\alpha C_i/T_i)$.

Deadline Monotonic planificador de prioridades estáticas que asigna las prioridades en función del plazo temporal, de manera que las tareas con menor plazo temporal tendrán una prioridad mayor que aquellas tareas con mayor plazo temporal.

DVS (*Dynamic Voltage Scaling*) técnica que consiste en reducir la tensión suministrada junto con la velocidad del procesador.

Earliest Deadline First planificador de prioridades dinámicas que asigna las prioridades a las tareas dinámicamente en función de sus plazos temporales

Earliest Slack First planificador de prioridades dinámicas que asigna las prioridades dinámicamente a las tareas dependiendo del tiempo que le sobra para alcanzar el plazo temporal una vez descontado el tiempo que necesita para finalizar su ejecución.

Hiperperiodo: Máximo común divisor de todos los periodos de un conjunto de tareas. Si todo el sistema es periódico, al cabo de un hiperperiodo se vuelve a obtener la misma secuencia de activaciones y ejecuciones de las tareas.

Instante crítico: Instante en el que se activan todas las tareas del sistema, y por tanto todas las tareas tendrán la máxima interferencia por parte de las tareas de mayor prioridad.

Utilización del procesador:

$$U = \sum_{i=1}^{n} \frac{C_i}{D_i}$$

Peor tiempo de ejecución (WCET) máximo tiempo que tardará la tarea en realizar sus cálculos

Periodo de activación (T): separación temporal entre dos activaciones sucesivas de la tarea

Planificadores off-line: antes de empezar la ejecución de la aplicación, calculan en que instante se ejecutará cada instancia de las tareas y durante cuanto tiempo se ejecutaran.

Planificadores on-line: durante la ejecución de la aplicación, deciden que tarea se ejecuta y durante cuanto tiempo.

Plazo temporal (D): tiempo que tiene la tarea para emitir una respuesta después de activarse.

Prioridad: escogida por el diseñador de la aplicación o el resultado de la aplicación de los algoritmos de planificación Rate Monotonic o Deadline Monotonic.

Promoción, instante en que la tarea cambiará de cola desde LRQ a URQ. Se puede calcular off-line y es plazo temporal absoluto menos el peor tiempo de finalización.

Rate Monotonic planificador de prioridades estáticas que asigna las prioridades a la tareas dependiendo de su periodo de activación, de manera que una tarea será más prioritaria cuanto más pequeño sea el periodo de activación.

Sistema planificable: sistema que cumple todas las restricciones (plazos) temporales de las tareas.

Sistemas de tiempo real estricto: sistemas de tiempo real en los que es absolutamente imperativo el cumplimiento de los plazos temporales, debido a que su incumplimiento puede dar lugar a una catástrofe, en vidas humanas o económicas.

Sistemas de tiempo real mixto o firme: Sistemas de tiempo real en los que si se pierde algún plazo de vez en cuando, únicamente se produce una degradación de la cualidad del sistema.

Sistemas de tiempo real: Sistemas cuyo su correcto funcionamiento no depende únicamente del resultado lógico de la computación sino también del tiempo en el que se producen los resultados.

Slack: tiempo sobrante desde la finalización de una tarea hasta la llegada de su plazo temporal.

Tiempo de activación: instante de llegada del periodo de una tarea.

Tiempo de respuesta: tiempo que tarda la tarea desde que se activa hasta que empieza a ser ejecutada.

Tiempo de finalización: tiempo que tarda la tarea desde que se activa hasta que finaliza su ejecución.

Tiempo ejecutado: tiempo que la tarea se ha estado ejecutando: este tiempo será útil para poder calcular la capacidad de procesador que aun requiere la tarea.

WCET : (Worst Case Execution Time) cota superior del tiempo máximo de ejecución de una determinada tarea.

WCRT: (Worst Case Response Time) máximo tiempo que tardará la tarea en realizar sus cálculos, teniendo en cuenta que la tarea las posibles interferencias de tareas de mayor prioridad.

REFERENCIAS BIBLIOGRÁFICAS

[1] Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba, “Advanced Configuration & Power Interface ACPI.” http://www.acpi.info/index.html. Sept 2004

[2] Altenbernd, P. “On the false path problem in hard real-time programs”, Proceedings of the Eighth Euromicro Workshop on Real-Time Systems, Page(s): 102 –107, June 1996

[3] AMD “Athlon™ 64 Processor Power and Thermal Data Sheet.” http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/30430.pdf

[4] AMD “PowerNow!™ Technology. Advanced Micro Devices”, http://www.amd.com/usen/assets/content_type/DownloadableAssets/Power_Now2.pdf

[5] Aloul, F.A.; Hassoun, S.; Sakallah K.A. y Blaauw, D. “Robust SAT-Based Search Algorithm for Leakage Power Reduction”, Workshop on Integrated Circuit Design. Power and Timing Modeling, Optimization and Simulation. Pages: 167 - 177 , 2002. ISBN:3-540-44143-3.

[6] Audsley, N., Burns, A., Richardson, M., Tindell, K. and Wellings A.J.. "Applying new scheduling theory to static priority pre-emptive scheduling", Software Engineering Journal, pp 284-292, September 93.

[7] Aydin, H. Melhem, R. Mosse, D. y Mejia-Alvarez, P. “Determining optimal processor speeds for periodic real-time tasks with different power characteristics.” 13th Euromicro Conference on Real-Time Systems. 2001

[8] Aydin, H. Melhem, R. Mosse, D. y Mejia-Alvarez, P. “Dynamic and Aggressive scheduling techniques for power-aware real-time systems.” Proc. Real-Time Systems Symposium, pp. 95-105, 2001.

[9] Baker, T.P., “A stack-based resource allocation policy for realtime processes”, Real-Time Systems Symposium, 1990. Proceedings., 11th , 5-7 Dec. 1990. Pages:191-200

[10] Bernat, G. “Specification and Analysis of a Weakly Hard Real Time Systems” Tesis Universitat de les illes Balears, 1998

[11] Borkar, S. “Low Power Design Challenges for the Decade“, Proceedings of the ASP-DAC 2001, Asia and South Pacific Design Automation Conference 2001, Session C4, Low Power Design Methodology, pp 293-296.

[12] Burd, T.D. and Brodersen, R.W., “Energy Efficient CMOS Microprocessor Design”, Proceedings of the 28th Hawaii International Conferences on System Sciences, Vol.1, pages 288-297, Jan 1995.

[13] Burns, A y Wellings, A.J. “Dual Priority Assignment: A practical method for increasing processor utilization”, Proceedings of 5th Euromicro Workshop on Real-Time Systems, 48 (1993).

[14] Burns, A. y Wellings, A.J. “Sistemas de Tiempo Real y lenguajes de programación”, Ed. Addison Wesley. 3ª Edición. 2001

[15] Burns, A., Tindell, K. and Wellings, A. “Effective analysis for engineering real time fixed priority schedulers,” IEEE Trans. on Software Eng., vol. 21, pp. 475–480, May 1995.

[16] Buttazzo, G.C. “Rate Monotonic vs. EDF: Judgment Day”, Real Time Systems, 28, 1-22, 2004.

[17] Carpenter, G. “Low Power SSOC for IBM’s PowerPC Information Appliance Platform”. http://www.research.ibm.com/arl/projects/papers/405LP.pdf

[18] Cazorla, F.J.; Ramirez, A.; Valero, M.; Knijnenburg, P.M.W.; Sakellariou, R.; Fernandez, E.; “QoS for high-performance SMT processors in embedded systems”, Micro, IEEE, Volume 24, Issue 4, July-Aug. 2004 Page(s):24 - 31

[19] Chandrakasan, A. P., Sheng, S. and Brodersen, R. W. “Low-Power CMOS Digital Design”, IEEE Journal of Solid-State Circuits. Vol. 27, No 4, April 1992.

[20] Cheng, S.T., Chen, C.M., Hwang J.W. “Low-Power Design for Real-Time Systems”. Real-Time Systems 15(2): 131-148 (1998)

[21] Chetto, H.; Chetto, M.; “Some Results of the Earliest Deadline Scheduling Algorithm”, Software Engineering, IEEE Transactions on Volume 15, Issue 10, Oct. 1989 Page(s):1261 - 1269

[22] Davis, R.I., Tindell, K.W. and Burns, A., "Scheduling Slack Time in Fixed-Priority Pre-emptive Systems", Proceedings of Real-Time System Symposium, pp. 222-231, 1993.

[23] Davis, R. y Wellings, A.J. “Dual Priority scheduling”, Proceedings IEEE Real Time Systems Symposium, pp. 100-109, December 1995.

[24] “Dynamic Power Management”, sourceforge.net: http://dynamicpower.sourceforge.net

[25] Garey, M. R. y Johnson, D. S. "Computers and Intractability," Freeman, San Francisco, 1979.

[26] Gruian, F., “Hard Real-Time Scheduling for Low-Energy Using Stochastic Data and DVS Processors.” International Symposium on Low-Power Electronics and Design. August 2001.

[27] Gutiérrez, J.J. and González-Harbour, M., “Optimized Priority Assignment for Tasks and Messages in Distributed Hard Real-Time Systems”. Proceeding of the 3rd Workshop on Parallel and Distributed Real-time Systems, pp 124-132, April 1995.

[28] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., Toshiba Corporation, “Advanced Configuration and Power Interface Specification” September 2004, http://www.acpi.info/DOWNLOADS/ACPIspec30.pdf

[29] Intel Corporation y Microsoft Corporation, “Advanced Power Management (APM). BIOS Interface Specification”, Revision 1.2, February 1996

[30] Intel PXA250 and PXA210. “Applications Processors Design Guide”, Feb 2002.

[31] Intel “XScale™ Technology.”, http://developer.intel.com/design/intelxscale.

[32] Irani, S.; Shukla, S. y Gupta, R. “Algorithms for Power Savings” , Proceedings of the 14th Symposium on Discrete Algorithms, 2003

[33] Jejurikar, R. y Gupta, R. “Energy Aware EDF Scheduling with Task Synchronization for Embedded Real Time Operating Systems”, Workshop on Compilers and Operating Systems for Low Power(COLP'02), September 2002

[34] Jejurikar, R. y Gupta, R. “Energy Aware Task Scheduling with Task Synchronization for Embedded Real Time Systems”, International Conference on Compliers, Architecture, and Synthesis for Embedded Systems (CASES’02), October 2002.

[35] Jejurikar, R.; Pereira, C. y Gupta, R., “Leakage aware dynamic voltage scaling for real-time embedded systems”, ACM IEEE Design Automation Conference, Volume 00, Pages: 275 - 280. June 2004. ISBN:1-58113-828-8

[36] Joseph, M. y Pandya, P. (1986). “Finding response times in a real-time system”, British Computer Society Computer Journal, 29(5) Pages 390-395, 1986.

[37] Katcher, D., Arakawa, H. and Strosnider, J. “Engineering and analysis of fixed priority schedulers,” IEEE Trans. on Software Eng., vol. 19, pp. 920–934, Sept.1993.

[38] Kim N., Ryu, M., Hong, S., Saksena, M., Choi, C. y Shin, H. “Visual assessment of a real-time system design: a case study on a CNC controller”, Proceedings IEEE Real-Time Systems symposium. December 1996.

[39] Krishna, C.M. y Lee, Y.H. “Voltage-Clock-Scaling Adaptative Scheduling Techniques for Low Power in Hard Real-Time Systems”, IEEE Transactions on Computers, vol 52, No 12, December 2003.

[40] Kumar, P., Srivastava, M., “Predictive strategies for low-power RTOS scheduling”, Computer Design, 2000. International Conference on , 17-20 Sept. 2000 Page(s): 343 –348

[41] Lee, Y.H.; Reddy K. y Krishna, C. M. “Scheduling Techniques for Reducing Leakage Power in Hard Real-Time Systems”, Conference on Real-Time Systems (ECRTS’03). 2003.

[42] Lehoczky,J.P Sha,L and Strsnider,J.K, "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", Proceedings of Real Time Systems Symposium, pp. 261-270, 1987.

[43] Lehoczky, J., Sha, L. and Ding, Y. “The rate monotonic scheduling algorithm: Exact characterization and average case behavior”. In Proceedings of IEEE Real-Time Systems Symposium, pages 166–171. IEEE Computer Society Press, December 1989.

[44] Lehoczky,J.P and Ramos-Thuel, S., "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Pre-emptive Systems", Proceedings of Real Time Systems Symposium, pp. 110-123, 1992.

[45] Leung, J.Y; Whitehead, J., "On the complexity of fixed-priority scheduling of periodic, real-time tasks", Performance Evaluation, No.2, 1982, p.p. 237-250.

[46] Hai Li; Bhunia, S.; Chen, Y.; Vijaykumar, T.N.; Roy, K.; “Deterministic clock gating for microprocessor power reduction”, High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. The Ninth International Symposium on, 8-12 Feb. 2003 Page(s):113 - 122

[47] Lim, S.S., Bae, Y.H.; Jang, G.T.; Rhee, B.D.; Min, S.L.; Park, C.Y.; Shin, H.; Park, K.; Moon, S.M.; y Kim, C.S., ”An accurate worst case timing analysis for RISC processors”, IEEE Transactions on Software Engineering, Volume: 21 Issue: 7 , July 1995, Page(s): 593 –604

[48] Liu, C.L. and Layland, J.W., “Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment” Journal of the ACM 20(1), pp 46,61, 1973.

[49] Liu, J.W.S. “Real-Time Systems”, Prentice Hall PTR, Upper Saddle River, NJ, 2000

[50] Locke, C., Vogel, D. y Mesler, T. “Building a predictable avionics platform in Ada: a case study”, Proceedings IEEE Real-Time Systems symposium. (1991).

[51] Microsoft, “Descripción de los distintos estados de Administración avanzada de energía”, http://support.microsoft.com/kb/q308535/ , 25 septiembre 2001.

[52] Miyoshi, A., Lefurgy, C., Hensbergen, E.V., Rajamony, R. y Rajkumar, R.. “Critical power slope: Understanding the runtime effects of frequency scaling.”, Proceedings of the 16th Annual ACM International Conference on supercomputing, June 2002.

[53] Mok, A.K. y Dertouzos, M. L. “Multiprocessor scheduling in a hard real-time environment.”, Proceedings of the 7th IEEE Texas Conference on Computing Systems, November 1978.

[54] Moncusí, M.A., Banús, J.M., Labarta, J. And Llamosí, A, “The Last Call Scheduling Algorithm for Periodic and Soft Aperiodic Tasks in Real-Time Systems.”, V Jornadas de concurrencia. Universidad de Vigo, Junio 1997 y Report de Recerca DEI-RR-96-004, Universitat Rovira i Virgili.

[55] Moncusí, M.A., Banus, J.M., Labarta, J. y Arenas, A., "A New Heuristic Algorithm to Assign Priorities and Resources to Tasks With End-to-End Deadlines", International Conference on Parallel and Distributed Processing Techniques and Applications 2001 (PDPTA'2001), Vol. IV, pag. 2102-2108, June 2001.

[56] Moncusí M.A., A. Arenas and J. Labarta, “Power low approach in a modified dual priority scheduling for hard real-time systems”, Parallel architectures and compilation techniques, Compilers and Operating Systems for Low Power, pp 2.1-2.6, September 2001.

[57] Moncusí, M.A., Arenas, A., Labarta, J., "Improving Energy Saving in Hard Real Time Systems via a Modified Dual Priority Scheduling", ACM SigArch Computer Architecture Newsletter, Vol 29, No.5, pp 19-24, December 2001.

[58] Moncusí, M.A., Arenas, A., Labarta, J. "Power Low Modified Dual Priority in Hard Real Time Systems with Resource Requirements", Workshop on Compilers and Operating Systems for Low Power (COLP'02), September 2002.

[59] Moncusi M.A., Arenas A. and Labarta J., “A modified dual priority scheduling algorithm for hard real time systems to improve energy savings”, Compilers and Operating Systems for Low Power, 17-36, Kluwer Academic/Plenum Publishers, Norwell MA USA (2003).

[60] Moncusi M.A., Arenas A. and Labarta J, “Energy Aware EDF Scheduling in Distributed Hard Real Time Systems”, RTSS Work-in-Progress proceedings, pp 103-106, 2003.

[61] Moncusi M.A., Arenas A. and Labarta J, “Moving Average Frequency Reduction for Low Power in Hard Real-Time Systems”, Power-Aware Real-Time Computing Workshop (PARC 2005), N.J. USA, 2005.

[62] Mosse, D., Aydin, H., Childers, B. y Melhem, R. “Compiler-assisted power-aware scheduling for real-time applications.” Workshop on Compilers and Operating systems for Low Power COLP 2000.

[63] Nvidia. Sequence Design, Inc. “Leakage Power Reduction.” http://www.sequencedesign.com/images/success_stories/nvidia_ss.pdf

[64] Oh, S.H.; Yang, S.M; “A Modified Least-Laxity-First scheduling algorithm for real-time tasks“, Real-Time Computing Systems and Applications, 1998. Proceedings. Fifth International Conference on , 27-29 Oct. 1998 Pages:31 - 36

[65] Palencia, J.C. y González-Harbour, M., ”Offset-Based Response Time Analysis of Distributed Systems Scheduled under EDF”, 15th Euromicro Conference on Real-Time Systems, July 2003.

[66] Palencia, J.C., Gutiérrez, J.J. González-Harbour, M., “Schedulability Analysis with Static and Dynamic Offsets”, Proceedings of 19th IEEE Real-Time Systems Symposium, December 1998.

[67] Pering, T.A., Burd, T.D. and Brodersen, R.W.. “Voltage Scheduling in the lpARM Microprocessor System”, Proceedings of the 1998 International Symposium on Low Power Electronics and Design, pages 96-101, 2000.

[68] Pillai, P. y Shin, K.G. “Real-time dynamic voltage scaling for low-power embedded operating systems”, 18th ACM Symposium on Operating Systems Principles. 2001

[69] Pouwelse, J., Langendoen, k., Sips, H. “Dynamic voltage scaling on a low-power microprocessor”, Proceedings of the 7th annual international conference on Mobile computing and networking table of contents, Pp 251-259. 2001, ISBN:1-58113-422-3.

[70] Pouwelse, J. Langendoen, K., and Sips, H., “Application-directed voltage scaling”, IEEE Transactions on Very Large Scale integration (TVLSI)

[71] Pushner, P., and Burns, A. “A review of Worst-Case Execution-Time Analysis", The international Journal of Time-Critical Computing Systems, 18, 115-128, 2000.

[72] Quan, G. and Hu, X., “Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage Processors”, Proc. of Design Automation Conference, pages 828-833, 2001.

[73] Rabaey, J. and Pedram, M. (Editors). "Low Power Design Methodologies". Kluwer Academic Publishers, Norwell, May, 1996.

[74] Ripoll Ripoll,J.I. , "Planificación con Prioridades Dinámicas en Sistemas de Tiempo Real Crítico", Tesis Doctoral Dept de Ingeniería de Sistemas Computadores y Automática. Universidad Politécnica de Valencia. Junio 1996.

[75] Saewong, S. and Rajkumar, R. “Practical voltage-scaling for fixed-priority RT-systems.” The 9th IEEE Real-Time and Embedded Technology and Applications Symposium, pp106-115, May 2003

[76] Sha, L., Rajkumar, R. y Lehoczky, J.P. “Priority Inheritance Protocols: An approach to Real-Time Synchronization”, IEEE Transactions on Computers, 39, 1175. 1990.

[77] Shin, Y. y Choi, K. (1999). “Power conscious Fixed Priority scheduling in hard real-time systems”. DAC 99, ACM 1-58113-7/99/06.

[78] Spuri, M.. “Holistic Analysis of Deadline Scheduled Real Time Distributed Systems”, RR-2772, INRIA, France, 1996.

[79] Spuri, M. and Buttazzo, G., "Efficient aperiodic service under Earliest Deadline Scheduling", Proceedings of the Real-Time Systems Symposium, pp. 2-11, 1994

[80] Stankovic, J.A., Spuri, M., Ramamritham, K. and Buttazzo, G.C.. “Deadline Scheduling for Real-Time Systems EDF and related Algorithms” Kluwer Academic Publishers. 1998.

[81] Stankovic, J.A.; Spuri, M.; Di Natale, M.; y Buttazo, G. “Implications of classical scheduling results for real-time systems”. IEEE Computer, Vol 28, No. 6, pp. 16-25, June 1995

[82] Tia, T.S., Liu, J.W.S, Shankar, M., "Aperiodic Request Scheduling in Fixed-Priority Pre-emptive Systems", Technical Report UIUCDCS-R-94-1859, Dept. Computer Science, University of Illinois at Urbana-Champaign, 1994.

[83] Tindell, K. and Clark, J. ;“Holistic Schedulability Analysis for Distributed Hard Real-Time Systems”. Microprocessing & Microprogramming, Vol 50, pp 117-134, April 1994.

[84] Transmeta Corporation, “Crusoe Processor Specification”, http://www.transmeta.com/crusoe/specs.html

[85] Vajapeyam, S.; Valero, M.; “Early 21st Century processors” Computer, Volume 34, Issue 4, April 2001 Page(s):47 - 50

[86] Wang,Y.C., Lin,K.J, “Implementing a General Real-Time Scheduling Framework in the RED-Linux Real-Time Kernel”, Real-Time Systems Symposium, p. 246, Dec 1999.

[87] Zhu, Y. and Mueller, F., "Preemption Handling and Scalability of Feedback DVS-EDF", Workshop on Compilers and Operating Systems for Low Power(COLP'02), September 2002