Universidad de Cantabria
Facultad de Ciencias
Santander, Junio 2013

Implementación del protocolo DFP en MaRTE OS y propuesta para su incorporación al lenguaje Ada

Marina Gutiérrez López

Trabajo Fin de Máster
Máster Universitario en Computación

Director:
Mario Aldea Rivas
Codirector:
Michael González Harbour

Resumen

Las tareas en un sistema de tiempo real deben atenerse a unos requisitos temporales determinados. Para garantizar que estos requisitos se satisfacen se emplean técnicas de planificación, como por ejemplo, la política de planificación dinámica EDF. En un sistema EDF las tareas se ejecutan en orden de plazo absoluto creciente.

Además en cualquier sistema realista hay recursos compartidos entre las tareas. El acceso a estos recursos debe estar controlado para que no se produzcan accesos concurrentes a los mismos que puedan derivar en un funcionamiento erróneo del sistema. Existen diversos protocolos para proteger el acceso a recursos compartidos. Para sistemas EDF el más utilizado es el SRP.

El protocolo SRP introduce el concepto de nivel de expulsión en las tareas y recursos del sistema. Su uso requiere modificar las reglas de la planificación EDF, de forma que para establecer el orden de ejecución de las tareas debe tenerse en cuenta tanto el plazo absoluto de las tareas como su nivel de expulsión.

El DFP es un nuevo protocolo de sincronización de acceso a recursos compartidos en sistemas de planificación dinámica EDF que ha sido propuesto recientemente como alternativa al protocolo SRP. Se basa en la idea del suelo de plazo, que se asigna a los recursos del sistema. Tiene todas las propiedades fundamentales del SRP, pero presenta una serie de diferencias que pueden ser ventajosas. Este trabajo ha consistido en una evaluación completa del DFP.

En primer lugar se ha implementado el DFP en MaRTE OS con diferentes estructuras de datos para la cola de tareas ejecutables. Se han realizado pruebas para evaluar su rendimiento y se ha encontrado que el DFP obtiene mejores resultados que el SRP para la misma estructura de datos. Además existe una estructura de datos (el montículo binario) con la que se obtienen los mejores resultados y que solo puede utilizarse para implementaciones del DFP.

La segunda parte del trabajo ha sido la elaboración de una propuesta para la incorporación del DFP al lenguaje Ada. Se han analizado los aspectos del lenguaje directamente relacionados, es decir, las políticas de planificación y sincronización existentes. Se ha encontrado que los cambios que son necesarios realizar son pequeños y sencillos y no interfieren con ningún otro aspecto del lenguaje.

Índice general

Capítulo 1

Introducción

Sistemas de tiempo real

Sistemas de tiempo real son aquellos en los que el resultado satisfactorio de una operación no depende solo de la corrección del valor producido, sino también del momento en el que dicho valor se produce. Ejemplos de este tipo de sistemas pueden ser los controladores industriales o los sistemas de adquisición de datos en los que las actividades a realizar deben ajustarse a los requerimientos temporales impuestos por el entorno físico que pretenden controlar o monitorizar.

Así, un sistema de tiempo real está formado por un conjunto de tareas con fuertes requerimientos temporales que se deben satisfacer para garantizar su correcto funcionamiento. El análisis de planificabilidad del sistema es por tanto una parte crucial en el desarrollo de los sistemas de tiempo real. Cuando un sistema puede garantizar que todas las tareas cumplen sus requisitos temporales se dice que el sistema es planificable.

Políticas de planificación

Para que las tareas de un sistema de tiempo real puedan satisfacer sus requisitos temporales se emplean diferentes políticas de planificación. Existe una gran cantidad de políticas de planificación. La mayoría de los sistemas operativos en tiempo real utilizan algoritmos de planificación basados en prioridades. Pueden ser prioridades fijas o dinámicas dependiendo de si ordenan las tareas en base a parámetros que se mantienen constantes en todas las activaciones de una tarea o cambian en cada activación (o incluso varían en el transcurso de una activación).

Las políticas de planificación basadas en prioridades fijas están mucho más extendidas debido principalmente a su mayor sencillez y la menor sobrecarga que su implementación en el sistema operativo impone a las aplicaciones. La más extendida es FIFO (*First In First Out*) con prioridades que ejecuta las tareas de igual prioridad atendiendo al orden de llegada.

Las políticas de planificación basadas en prioridades dinámicas imponen una mayor sobrecarga a las aplicaciones que las basadas en prioridades fijas, pero a cambio presentan la ventaja de ser más potentes y adaptables. Así con ellas se

pueden planificar conjuntos de tareas que no son planificables con prioridades fijas.

La política de planificación EDF (*Earliest Deadline First*) [1] es la política de planificación dinámica más estudiada y utilizada para sistemas de tiempo real. Se trata de un algoritmo de planificación óptimo en monoprocesadores, en el sentido de que si un conjunto de tareas no es planificable con la política EDF, entonces no es planificable con ninguna otra política de planificación.

Se dice que una política de planificación es expulsora o no expulsora cuando permite o no que una tarea más urgente expulse a la tarea que está ejecutando en ese momento y empiece a ejecutar en su lugar. La política de planificación EDF es expulsora.

Políticas de sincronización para EDF

En un sistema realista las tareas no son independientes entre sí, sino que utilizan recursos compartidos con otras tareas. El acceso a estos recursos debe realizare de forma mutuamente exclusiva. Los recursos normalmente se protegen con semáforos o *mutex* propios del sistema operativo. Cuando una tarea de prioridad alta se suspende porque tiene que esperar a que una tarea de baja prioridad libere un recurso, se dice que la tarea de prioridad alta está bloqueada.

Para minimizar los tiempos de bloqueo se utilizan protocolos de sincronización de acceso a recursos compartidos. Para sistemas planificados con EDF el protocolo más utilizado es el SRP (*Stack Resource Policy*) [2] [3]. Se trata de una generalización del protocolo de techo inmediato de prioridad o PCP (*Priority Ceiling inheritance Protocol*) [4] que se utiliza con algoritmos de planificación de prioridades fijas y tiene sus mismas ventajas: evita la inversión de prioridad no acotada, evita interbloqueos en monoprocesadores y presenta un buen tiempo de bloqueo de peor caso. Además, al igual que ocurre con el PCP, las propias reglas del SRP aseguran la exclusión mutua en el acceso a los recursos compartidos sin necesidad de utilizar ningún tipo de primitiva bloqueante como los semáforos. Una explicación detallada del SRP puede encontrarse en la sección 2.1.2.

Recientemente un nuevo protocolo ha sido propuesto como alternativa al SRP, el DFP (*Deadline Floor inheritance Protocol*) [5]. Se trata de un protocolo que tiene todas las propiedades fundamentales del SRP pero que presenta una serie de diferencias que pueden ser ventajosas: es conceptualmente más sencillo que el SRP, no añade parámetros de planificación a las tareas y no modifica el criterio de ordenación de la política EDF. Su mayor desventaja es que necesita hacer una lectura del reloj del sistema cada vez que una tarea accede a un recurso. En la sección 2.1.4 se explica el funcionamiento completo del DFP.

Objetivos del trabajo

El presente trabajo persigue los siguientes objetivos: comparar las prestaciones de los protocolos SRP y DFP y realizar una propuesta de incorporación del protocolo DFP al estándar Ada.

El objetivo principal es evaluar las posibles ventajas y/o inconvenientes que puede presentar el DFP respecto al SRP. Así, se realiza una implementación completa del protocolo sobre un sistema operativo de tiempo real, MaRTE OS

y se llevan a cabo pruebas de rendimiento. Para realizar esta implementación es necesario entender en profundidad los protocolos SRP y DFP, así como la estructura del planificador de MaRTE y su implementación del SRP, ya que incluso, se ha modificado la actual implementación del SRP en MaRTE OS para probar sus prestaciones con colas de tareas ejecutables implementadas con diferentes estructuras de datos.

El segundo objetivo es la propuesta de incorporación del DFP al estándar Ada. Para ello, se estudia la actual definición de la política de planificación EDF en Ada, que a su vez, incluye la definición del SRP. Se pretende que la propuesta se integre lo mejor posible en el estándar, de forma que suponga el menor número de cambios y no suponga conflictos con otros aspectos del lenguaje.

Comparación de los protocolos DFP y SRP en MaRTE OS

MaRTE OS [6] es un sistema operativo de tiempo real desarrollado por el grupo de Computadores y Tiempo Real de la Universidad de Cantabria. MaRTE OS está escrito en Ada con algunas partes en C y código ensamblador y proporciona interfaces para aplicaciones Ada, C y C++. Además da soporte a la mayoría de funcionalidades de tiempo real definidas en el estándar Ada 2012 [7], en particular implementa la política de planificación EDF junto con el protocolo SRP. En la sección 2.2 se exponen las partes más interesantes de la implementación del SRP en MaRTE OS.

El capítulo 3 está dedicado a la evaluación del DFP en MaRTE OS y su comparación con el protocolo SRP. Para empezar se exponen los detalles de la implementación realizada de una manera análoga a como se hizo con la implementación ya existente del SRP. Además del propio protocolo también se han hecho diferentes implementaciones de MaRTE OS, con diferentes estructuras de datos para su cola de tareas ejecutables. Aunque en principio esto parezca no tener relación con la evaluación de los protocolos, en la sección 3.2 se explica el vínculo entre el protocolo y la estructura de datos, haciendo por tanto que sea un elemento significativo en su comparación. Por último se llevan a cabo diversas pruebas para evaluar la eficiencia de los protocolos en MaRTE OS cuyos resultados pueden verse en la sección 3.3.

Es importante resaltar que, dada la novedad del protocolo DFP, la implementación realizada como parte de este trabajo es la primera implementación de este protocolo a nivel mundial y la única de la se tiene noticia a fecha de la escritura de este trabajo.

Incorporación del DFP al lenguaje Ada

Una vez evaluado el rendimiento del DFP en comparación con el SRP, se estudia su posible incorporación al estándar Ada. La política de planificación EDF fue añadida al estándar Ada en el 2005. El estándar especifica además el SRP como el protocolo de sincronización para controlar el acceso a recursos compartidos en un sistema planificado con EDF. La incorporación del protocolo DFP al estándar, supondría no solo aprovechar sus ventajas frente al SRP, sino también simplificar el modelo de planificación que el estándar Ada propone.

Así, en el capítulo 4 se analizan los cambios que sería necesario llevar a cabo en las políticas de sincronización de acceso a los objetos protegidos y a la propia política de planificación EDF para incorporar el DFP a Ada.

Capítulo 2

Antecedentes

2.1. Políticas de sincronización

2.1.1. Modelo del sistema

Se parte de un sistema de tiempo real con n tareas $\{\tau_1, \tau_2, ..., \tau_n\}$ ejecutando en un monoprocesador. Cada tarea consiste de una serie de trabajos que han de ser completados antes de su plazo.

Cuando las tareas son periódicas todos sus trabajos tienen un mismo intervalo de llegada, el periodo T_i, de forma que si un trabajo de una tarea periódica τ_i se activa en t, el próximo trabajo de la tarea llegará en $t + T_i$. Para tareas no periódicas se considera que cada tarea tiene un tiempo mínimo entre activaciones al que también se llama T_i, de forma que se garantiza que el siguiente trabajo de la tarea τ_i no llegará antes de $t + T_i$.

La duración de la ejecución de cada trabajo de una tarea τ_i está acotado por el tiempo de ejecución de peor caso C_i. Cada trabajo de una tarea τ_i tiene un plazo relativo o *deadline* D_i que es el mismo para todos los trabajos de la tarea y es, por tanto, el propio plazo relativo de la tarea τ_i.

Si un trabajo de la tarea τ_i llega en el instante t debe finalizar su ejecución completa antes de que transcurra el tiempo equivalente a su plazo relativo D_i. Por tanto, el plazo absoluto de este trabajo es $d_i = t + D_i$.

El sistema de tiempo real consta también de m recursos $(r^1, r^2, ..., r^m)$. Las tareas acceden a los recursos bajo exclusión mutua, pero no se hace ninguna suposición sobre en qué momento de su ejecución lo hacen. Sí se supone que las tareas no se pueden suspender mientras están ejecutando dentro del recurso. Los recursos pueden hacer uso de otros recursos por lo que los accesos anidados a recursos están permitidos. Sin embargo los accesos solapados no se contemplan.

La inclusión de recursos compartidos en el sistema implica que las tareas pueden sufrir bloqueos que deben ser tenidos en cuenta en el análisis de planificación.

Earliest Deadline First (EDF)

De acuerdo con la política de planificación EDF (*Earliest Deadline First*) [1] las tareas activas del sistema se ordenan en una cola de tareas ejecutables de

acuerdo a su plazo absoluto. De forma que, en ausencia de bloqueos, el trabajo con el plazo absoluto más corto o cercano se ejecuta primero.

Se trata de una política de planificación expulsora y basada en prioridades dinámicas, ya que si bien el plazo relativo de cada tarea no cambia, el plazo absoluto de calcula nuevamente para cada activación de la tarea.

Si más de un trabajo tiene el mismo plazo absoluto, entonces se ejecutan en orden FIFO (*First In, First Out*), es decir, la tarea que lleve más tiempo activa en el sistema se ejecutará primero. Cuando un trabajo acaba su ejecución el trabajo con el plazo menor o más cercano y que más tiempo lleve en el sistema empezará a ejecutarse.

2.1.2. *Stack Resource Policy* (SRP)

Dada una aplicación definida por un conjunto de tareas $\{\tau_1, \tau_2, ..., \tau_n\}$ y un conjunto de recursos $(r^1, r^2, ..., r^m)$, de forma que $\mathcal{A}(r^i)$ es el subconjunto de tareas que pueden acceder al recurso r^i, el SRP [2] [3] se define de la siguiente manera:

1. A cada tarea, τ_i se le asigna un nivel de expulsión o *preemption level*, π_i. En un sistema planificado con EDF el nivel de expulsión tiene que ser inversamente proporcional al plazo relativo de la tarea, es decir:

$$\pi_i < \pi_j \Leftrightarrow D_i > D_j.$$

2. A cada recurso, r^j se le asigna un techo de nivel de expulsión, Π^j que es igual al máximo nivel de expulsión de las tareas que acceden al recurso.

$$\Pi^j = max\{\pi_j : \tau_j \in \mathcal{A}(r^i)\}.$$

3. Cuando una tarea τ_i accede a un recurso r^j se compara su nivel de expulsión con el techo de nivel de expulsión del recurso. Si el techo de nivel de expulsión es mayor que el nivel de expulsión de la tarea, la tarea hereda el nivel de expulsión del recurso, de forma que: $\pi_i \leftarrow max\{\pi_i, \Pi^j\}$.

4. Cuando la tarea libera el recurso su nivel de expulsión es inmediatamente restaurado al valor que tenía antes de tomar el recurso.

5. Se añade una nueva regla de planificación al algoritmo EDF: una tarea solo puede ejecutarse si su nivel de expulsión es estrictamente mayor que el máximo de los niveles de expulsión de la tarea actualmente en ejecución y de los recursos actualmente tomados en el sistema.

2.1.3. Planificación de un sistema EDF con SRP

En la sección anterior se ha expuesto la regla de planificación que añade el SRP a las reglas de planificación EDF ya existentes. Esta nueva regla produce un cambio importante en el orden en el que se ejecutan las tareas en un sistema EDF con SRP como protocolo de sincronización (EDF & SRP). Si en un sistema EDF "puro" las tareas se ejecutan en orden de plazo creciente, en un sistema EDF & SRP se debe tener en cuenta tanto el plazo como el nivel de expulsión

de las tareas. Así, una tarea τ_a se coloca por delante de otra τ_b en la cola de tareas ejecutables si es menor de acuerdo a la siguiente relación de orden:

$$\tau_a \leq \tau_b \Leftrightarrow \left\{ \begin{array}{ll} d_a \leq d_b & si\ \ \tau_b\ \ no\ \ tiene\ \ recursos\ \ tomados \\ d_a \leq d_b\ \ y\ \ \pi_a \geq \pi_b & si\ \ \tau_b\ \ tiene\ \ recursos\ \ tomados \end{array} \right\} \tag{2.1}$$

Orden total

Es interesante notar que esta relación de orden entre las tareas no corresponde a una relación de orden total.

Se dice de una relación que es de orden total cuando establece entre los elementos del conjunto un orden lineal único. Este orden es siempre el mismo independientemente de la secuencia en la que se hayan añadido los elementos al conjunto. Para que una relación sea de orden total ha de verificar las siguientes propiedades:

$$\begin{array}{ll} if\ \ a \leq b\ \ y\ \ b \leq a \Rightarrow a = b & (antisimetria) \\ if\ \ a \leq b\ \ y\ \ b \leq c \Rightarrow a \leq c & (transitividad) \\ a \leq b\ \ o\ \ b \leq a & (totalidad) \end{array} \tag{2.2}$$

Para demostrar que el SRP establece un orden no total entre las tareas se plantea el siguiente ejemplo. Sean dos tareas τ_a y τ_b con los parámetros de planificación expuestos en el Cuadro 2.1. Para averiguar cuál de las dos tareas ha de ejecutarse antes en un sistema EDF & SRP se aplica la relación de orden 2.1.

	d	π	*recurso tomado*
τ_a	11	4	$\surd$
τ_b	6	3	$\times$

Cuadro 2.1: Parámetros de planificación de las tareas τ_a y τ_b

1. ¿Es $\tau_a \leq \tau_b$? Como τ_b no tiene ningún recurso tomado de acuerdo con 2.1 solo hay que comparar los plazos de las tareas. Así se observa que $d_a > d_b$ y por tanto $\tau_a \leq \tau_b$ es falso.

2. ¿Es $\tau_b \leq \tau_a$? Como τ_a si tiene algún recurso tomado de acuerdo con 2.1 hay que comparar tanto los plazos de las tareas como sus niveles de expulsión. Así se observa que $d_b < d_a$ y $\pi_b < \pi_a$ por tanto $\tau_b \leq \tau_a$ es también falso.

3. En los puntos 1 y 2 anteriores se ha obtenido que tanto $\tau_a \leq \tau_b$ como $\tau_b \leq \tau_a$ son falsos lo que viola la propiedad de la totalidad expuesta en 2.2 y por tanto se puede concluir que la relación de orden 2.1 no es de orden total.

2.1.4. *Deadline Floor inheritance Protocol* (DFP)

Dada una aplicación definida por un conjunto de tareas $\{\tau_1, \tau_2, ..., \tau_n\}$ y un conjunto de recursos $(r^1, r^2, ..., r^m)$, de forma que $\mathcal{A}(r^i)$ es el subconjunto de

tareas que pueden acceder al recurso r^i, el DFP [5] se define de la siguiente manera:

1. A cada recurso, r^i se le asigna un suelo de plazo relativo o *deadlinefloor*, D^i que es igual al mínimo plazo relativo de las tareas que acceden al recurso.

$$D^i = min\{D_j : \tau_j \in \mathcal{A}(r^i)\}$$

2. Cuando una tarea τ_j que se activa en el momento a accede a un recurso r^i en el momento s (de forma que $a < s$) su plazo relativo se reduce inmediatamente a D^i y como resultado su plazo absoluto también puede cambiar de forma que: $d_j \leftarrow min\{s + D^i, d_i\}$ (Figura 2.1).

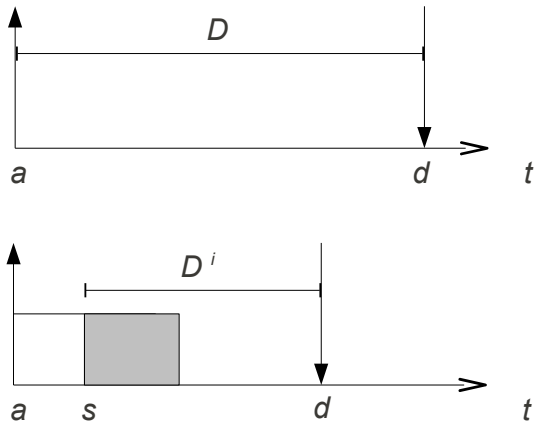

Figura 2.1: Reducción del plazo absoluto de una tarea al acceder a un recurso en un sistema con DFP.

3. Cuando la tarea libera el recurso su plazo relativo y absoluto son inmediatamente restaurados al valor que tenían antes de tomar el recurso.

Las reglas de planificación se mantienen invariables respecto a cualquier otro sistema planificado con EDF: en todo momento las tareas se encuentran ordenadas en la cola de tareas ejecutables en función de sus plazos absolutos, de manera que una tarea solo puede ejecutarse si su plazo absoluto es estrictamente menor o más próximo que el plazo absoluto de la tarea actualmente en ejecución.

2.1.5. Planificación de un sistema EDF con DFP

Tal y como se acaba de explicar el DFP no introduce ninguna regla de planificación nueva a las ya existente en cualquier sistema EDF y por tanto las tareas se ejecutan en orden de plazo absoluto creciente. Así, una tarea τ_a se coloca por delante de otra τ_b en la cola de tareas ejecutables si es menor, es decir, si satisface la siguiente relación:

$$\tau_a \leq \tau_b \Leftrightarrow d_a \leq d_b \tag{2.3}$$

A diferencia de lo que ocurría con la relación de orden que se establece en los sistemas con EDF & SRP, esta relación sí es de orden completo. En la sección 3.2 se puede ver como el hecho de que sea una relación de orden completo tiene importantes implicaciones en la eficiencia de la implementación.

2.2. Implementación del SRP en MaRTE OS

MaRTE OS es un sistema operativo desarrollado por el grupo de Computadores y Tiempo Real de la Universidad de Cantabria. En la Figura 2.2 se puede ver un esquema de su arquitectura interna.

El núcleo de MaRTE OS está escrito en Ada, pero es compatible con el perfil mínimo de sistema de tiempo real definido en el estándar POSIX.13. Así, presenta una interfaz POSIX sobre la cual se pueden ejecutar aplicaciones C.

En una capa superior a la interfaz POSIX se adapta el *Run-Time* de Ada (GNARL) y así se proporciona también una interfaz Ada, sobre la que poder desarrollar aplicaciones Ada.

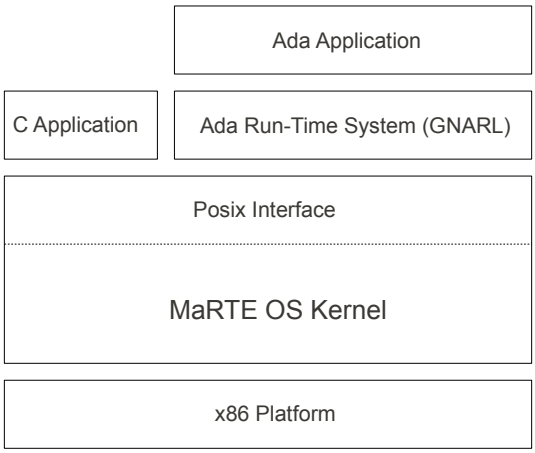

Figura 2.2: Arquitectura de MaRTE OS.

2.2.1. Visión general del planificador de MaRTE OS

MaRTE OS usa un planificador jerárquico con dos niveles. El planificador básico usa prioridades fijas, tal y como establece el estándar POSIX [8]. A cada tarea (llamadas hilos en POSIX) se le asigna un número entero llamado prioridad que se usa como parámetro principal en la planificación, de forma que una tarea de prioridad mayor siempre expulsa a una tarea de prioridad menor.

POSIX permite además asignar a cada tarea una política de planificación de manera individual, de forma que, dentro de un mismo nivel de prioridad pueden convivir hilos con diferentes políticas de planificación. El estándar POSIX define las políticas FIFO, *Round Robin* y Servidor Esporádico. Además de esas políticas, MaRTE OS define la política EDF.

La estructura principal que maneja el planificador de MaRTE es la cola de tareas ejecutables. Esta cola está implementada con un *array* de colas, una cola para cada nivel de prioridad (Figura 2.3). En cada una de estas colas se encuentra el conjunto de tareas con la misma prioridad y si se trata de tareas EDF se aplican sobre ellas las reglas de planificación propias de EDF.

Cuando una tarea se activa se añade a la cola que corresponde a su prioridad. La posición que ocupe la tarea dentro de esa cola depende de su política

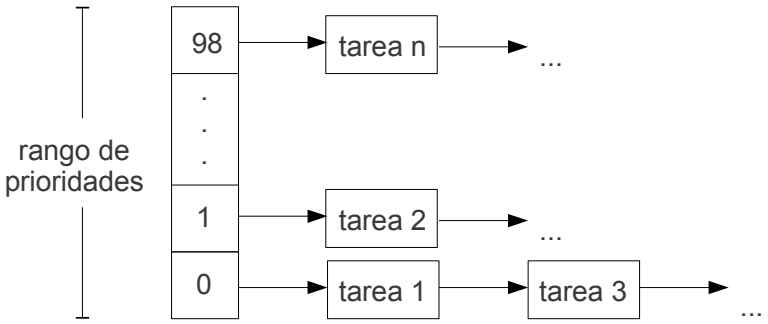

Figura 2.3: Cola de tareas ejecutables en MaRTE OS.

y parámetros de planificación. Por ejemplo, una tarea FIFO que se acaba de activar se colocará la última en la cola que corresponda a su prioridad.

A cada activación de una tarea EDF se le asigna un nuevo plazo absoluto (resultado de sumar su plazo relativo al momento de activación). Las tareas EDF se ordenan en la cola correspondiente a su nivel de prioridad en orden de plazo absoluto creciente.

La implementación de la política de planificación EDF hecha en MaRTE OS cuenta con el SRP como protocolo de sincronización para gestionar el uso de recursos compartidos. Así, además de la prioridad, las tareas EDF en MaRTE OS se necesitan tres parámetros más: los plazos relativo y absoluto y el nivel de expulsión.

Como en cualquier otro sistema operativo POSIX, los recursos en MaRTE OS están protegidos mediante *mutex*. Los *mutex* son los que implementan el protocolo de sincronización utilizado, en este caso el SRP y por tanto necesitan tener un campo que represente su nivel de expulsión.

En la Figura 2.4 pueden verse los tipos de datos y campos necesarios para la implementación del protocolo SRP en MaRTE OS. El tipo de dato entero Task_Preemption_Level se emplea para definir los niveles de expulsión de recursos y tareas. Se observa como los *mutex* tienen (entre otros) campos que representan su nivel de expulsión (Preemption_Level) y el nivel de expulsión original de la tarea que lo toma (Owner_Preemption_Level). En la siguiente sección se explica la necesidad de este último campo. De igual forma, las tareas en su bloque de control (*Task Control Block*, TCB) tienen los campos necesarios para funcionar en un sistema EDF & SRP: plazo absoluto (Deadline) y nivel de expulsión (Preemption_Level)

2.2.2. Acceso a un recurso con SRP

Los procedimientos que gestionan la toma y liberación de los *mutex* que, como se ha dicho, protegen a los recursos compartidos del sistema son Running_Task_Locks_Mutex y Task_Unlock_Mutex. Estos mismos procedimientos son los que invoca el Ada *Run-Time* para garantizar la exclusión mutua en el acceso a los objetos protegidos de las tareas Ada.

Para comprender mejor el funcionamiento del SRP en la implementación que se hace en MaRTE OS, a continuación se describe el proceso completo toma y liberación de un *mutex*.

```
type Task_Preemption_Level; --tipo entero

type Mutex is record
   ...
   Preemption_Level : Task_Preemption_Level;
   Owner_Preemption_Level : Task_Preemption_Level;
end record;

type TCB is record
   ...
   Deadline : HWTime := 0; --No especifico del SPR
   Preemption_Level : Task_Preemption_Level;
end record;
```

Figura 2.4: Campos necesarios para la implementación del SRP en MaRTE OS.

Cuando una tarea accede a un recurso protegido con un *mutex* se ejecuta el procedimiento Running_Task_Locks_Mutex (Figura 2.5). En primer lugar se incrementa el número de *mutex* tomados por la tarea y se almacena el valor original del nivel de expulsión de la tarea. Se compara el nivel de expulsión de la tarea con el nivel de expulsión del *mutex*. Si el nivel de expulsión del *mutex* es mayor que el de la tarea, se sube el valor del nivel de expulsión de la tarea al del *mutex*.

```
procedure Running_Task_Locks_Mutex (Mutex) is
   ...
   Task.Num_Mutex_Owned ++;
   Mutex.Owner_Preemption_Level := Task.Preemption_Level;
   if Task.Preemption_Level < Mutex.Preemption_Level then
     Task.Preemption_Level := Mutex.Preemption_Level;
   end if;
   ...
end Running_Task_Locks_Mutex;
```

Figura 2.5: Pseudocódigo de la función Running_Task_Locks_Mutex para el protocolo SRP.

Aunque el nivel de expulsión de la tarea, uno de sus parámetros de planificación, ha podido elevarse, no es necesario reordenar la cola de tareas ejecutables, ya que se trata de la tarea que está ejecutando y no puede expulsar a ninguna otra.

Cuando una tarea abandona un recurso protegido por un *mutex* se ejecuta el procedimiento Task_Unlock_Mutex (Figura 2.6). En primer lugar se decrementa el número de *mutex* tomados por la tarea. Se restaura el valor del nivel de expulsión de la tarea que se había almacenado en un campo del *mutex* y se ejecuta el procedimiento Reorder_and_Dispatch.

El nivel de expulsión de la tarea puede haber cambiado otra vez, en este caso puede haber disminuido y por lo tanto sí es necesario reordenar la cola de

```
procedure Task_Unlocks_Mutex (Task, Mutex) is
   ...
   Task.Num_Mutex_Owned --;
   Task.Preemption_Level := Mutex.Owner_Preemption_Level;
   Reorder_and_Dispatch (Task);
   ...
end Task_Unlocks_Mutex;
```

Figura 2.6: Pseudocódigo de la función Task_Unlocks_Mutex para el protocolo SRP.

tareas ejecutables ya que puede haber alguna otra tarea en la cola que pueda expulsarla. El procedimiento Reorder_and_Dispatch reordena la tarea actual en la cola de tareas ejecutables y en caso de que ya no sea la tarea más urgente, realiza un cambio de contexto, expulsando a la tarea que acaba de liberar el *mutex*.

```
function < (Left_Task, Right_Task) return Boolean is
   ...
   return Left_Task.Deadline < Right_Task.Deadline
     and then (not Mutexes_Owned or else
       Left_Task.Preemption_Level >
         Right_Task.Preemption_Level);
end <;
```

Figura 2.7: Pseudocódigo de la función < para el protocolo SRP.

Para ordenar la cola de tareas ejecutables el procedimiento Reorder_and_Dispatch llama a la función < (Figura 2.7) que implementa relación de orden 2.1. Esta función compara los parámetros de planificación de la tarea que está ordenando con el resto de tareas de la cola de tareas ejecutables. Comienza por la última tarea de la cola y va avanzando hasta encontrar su lugar.

2.2.3. Interfaz SRP

Aunque ni la política de planificación EDF, ni el protocolo SRP están incluidos en el estándar POSIX, MaRTE OS proporciona una interfaz tipo POSIX para que las aplicaciones puedan gestionar los parámetros específicos de estas dos políticas.

En la Figura 2.8 se puede ver la interfaz para el protocolo SRP. Se observa como hay funciones para establecer y leer el nivel de expulsión de las tareas y los *mutex* tanto dinámicamente como a través de su objeto de atributos.

```
int pthread_mutexattr_setpreemptionlevel
   (pthread_mutexattr_t *attr, unsigned short int level);
int pthread_mutexattr_getpreemptionlevel
   (pthread_mutexattr_t *attr, unsigned short int *level);

int pthread_mutex_setpreemptionlevel
   (pthread_mutex_t *mutex, unsigned short int level);
int pthread_mutex_getpreemptionlevel
   (pthread_mutex_t *mutex, unsigned short int *level);

int pthread_attr_setpreemptionlevel
   (pthread_attr_t *attr,
    unsigned short int preemptionlevel);
int pthread_attr_getpreemptionlevel
   (const pthread_attr_t *attr,
    unsigned short int *preemptionlevel);

int pthread_setpreemptionlevel
   (pthread_t thread, short int preemptionlevel);
int pthread_getpreemptionlevel
   (pthread_t thread, short int *preemptionlevel);
```

Figura 2.8: Interfaz del SRP en MaRTE OS.

2.3. EL lenguaje Ada y su soporte para EDF

2.3.1. El estándar Ada

Ada es un lenguaje de programación orientado a objetos y fuertemente tipado de forma estática. Es un lenguaje multipropósito, orientado a objetos y concurrente, pudiendo llegar desde la facilidad de Pascal hasta la flexibilidad de C++. Fue diseñado con la seguridad en mente y con una filosofía orientada a la reducción de errores comunes y difíciles de descubrir. Para ello se basa en un tipado muy fuerte y en chequeos en tiempo de ejecución (desactivables en beneficio del rendimiento). La sincronización de tareas se realiza mediante la primitiva *rendezvous* y el uso de objetos protegidos.

Es un lenguaje que continúa introduciendo cambios y nuevas características. La última revisión del estándar es del 2012 [7] y la próxima se espera para 2020. Tal y como se ha dicho en la Introducción, una parte importante de este trabajo consiste en la realización de una propuesta para incorporar el protocolo DFP al estándar Ada. Es por tanto interesante reseñar como es el proceso para la incorporación de un cambio al estándar, en particular en lo relativo a cambios relacionados con la parte de tiempo real del estándar:

1. Las propuestas se presentan y se discuten ante un equipo de trabajo que se reúne anualmente. En el ámbito del tiempo real las propuestas se presentan en el IRTAW (*International Real Time Ada Workshop*).

2. Una vez son aceptadas por el IRTAW, las propuestas se plantean como un *Ada Issue* [9] a los comités de estandarización ISO y se abre un periodo

de discusión pública.

3. Finalmente el *Ada Issue* se rechaza o se acepta para formar parte del estándar.

2.3.2. EDF & SRP en Ada

En la revisión del estándar Ada del año 2005 [10] se añadió la política de planificación EDF y el SRP como el protocolo de sincronización de acceso a recursos compartidos entre tareas EDF. Es interesante señalar que la definición inicial del protocolo SRP en el estándar sufrió un cambio puesto que resultó ser incorrecta [11] [12], lo cual da una idea de la complejidad del SRP.

La política EDF está definida en la sección D.2.6 del Manual de Referencia de Ada (ARM) con el identificador EDF_Across_Priorities. En la Figura 2.9 se pueden ver los detalles del paquete Ada.Dispatching.EDF que proporciona operaciones para manejar el plazo absoluto de las tareas. La funcionalidad del protocolo SRP se incorporó como una extensión del ya existente protocolo de techo de prioridad llamado Ceiling_Locking.

```
with Ada.Real_Time;
with Ada.Task_Identification;

package Ada.Dispatching.EDF is

   subtype Deadline is Ada.Real_Time.Time;
   Default_Deadline : constant Deadline :=
                                 Ada.Real_Time.Time_Last;

   procedure Set_Deadline
             (D : in Deadline;
              T : in Ada.Task_Identification.Task_Id :=
                     Ada.Task_Identification.Current_Task);

   procedure Delay_Until_And_Set_Deadline
             (Delay_Until_Time : in Ada.Real_Time.Time;
              Deadline_Offset : in Ada.Real_Time.Time_Span);

   function Get_Deadline
            (T : Ada.Task_Identification.Task_Id :=
                 Ada.Task_Identification.Current_Task)
            return Deadline;

end Ada.Dispatching.EDF;
```

Figura 2.9: Paquete Ada.Dispatching.EDF.

El modelo de planificación de Ada se basa en un planificador jerárquico, con un primer nivel basado en prioridades fijas y la posibilidad de definir planificadores de segundo nivel específicos para bandas de prioridad. En ese segundo nivel de planificadores se incorpora el EDF (con el protocolo SRP) al lenguaje

Ada que permite definir bandas (a las que hace referencia el identificador de la política EDF_Across_Priorities) en las que rige la política de planificación EDF.

Los niveles de expulsión de las tareas y los objetos protegidos se mapean como prioridades dentro de la banda especificada, de forma que la prioridad definida para una tarea que esté en la banda de prioridad EDF, será en realidad su nivel de expulsión y su prioridad será, mientras no se vea afectada por el uso de objetos protegidos, la prioridad más baja de la banda.

En la Figura 2.10 se puede observar un ejemplo de uso de la política EDF en el que se define una banda EDF entre las prioridades 20 y 25, dos tareas y un objeto protegido cuyas prioridades están en dicha banda y, por tanto, son planificados de acuerdo a la política EDF (y al protocolo SRP). Suponiendo que no haya más tareas en el sistema, inicialmente la tarea EDF _Task_A se sitúa en el nivel de prioridad base del rango EDF, es decir en el nivel de prioridad 20. Si la tarea EDF_Task_A accede al objeto protegido EDF_Object, entonces hereda su nivel de expulsión y EDF_Task_A se coloca ahora en el cola correspondiente a este nivel de expulsión, es decir, en la cola de prioridad 24.

Si en ese momento se activa otra tarea dentro del rango EDF, esta se situará en el nivel de prioridad más alto que no esté vacío y en la que se pueda poner a la cabeza. Para poder expulsar a la tarea que se encuentre a la cabeza de esa cola, debe satisfacer la relación 2.1, es decir debe tener un plazo menor o más cercano y un nivel de expulsión mayor.

Suponiendo que se mantiene la situación que se acaba de describir, solo se encuentra ocupado el nivel de prioridad 24 con la tarea EDF_Task_A. Al activarse la tarea EDF_Task_B ha de buscar su sitio en la cola de tareas ejecutables. Si el plazo de la tarea EDF_Task_B es menor que el de la tarea EDF_Task_A, la tarea EDF_Task_B se coloca la primera en el nivel de prioridad 24, ya que su nivel de expulsión es 25 y por tanto mayor que el de la tarea EDF_Task_A, que es 24. Si por contra, el plazo de la tarea EDF_Task_B es mayor que el de la tarea EDF_Task_A, la tarea EDF_Task_B no puede ponerse a la cabeza de la cola del nivel de prioridad 24 y por tanto, al no haber más niveles de prioridad ocupados, caerá al nivel de prioridad base.

Además de con el pragma Priority_Specific_Dispatching, el identificador EDF _Across_Priorities puede utilizarse con el pragma Task_Dispatching_Policy. En ese caso todo el rango de prioridades del sistema se planifica de acuerdo a la política EDF.

Este modelo, aunque pueda parecer estructuralmente algo complejo, presenta la ventaja de ordenar cada cola dentro de una banda de prioridad de acuerdo a un solo parámetro de planificación (el plazo absoluto, en el caso de la política EDF), simplificando así la función que se utiliza para ordenar las tareas dentro de la cola.

Tal y como se ha visto en la sección 2.2 MaRTE OS no utiliza bandas de prioridad para implementar la política EDF, sino que mantiene a todas las tareas EDF en una misma prioridad y dentro de ella las ordena de acuerdo a dos parámetros de planificación: el plazo absoluto y el nivel de expulsión. Al emplear dos parámetros la función de ordenación es más compleja, pero la estructura de la cola de tareas ejecutables es más sencilla.

Un análisis más profundo de las ventajas e inconvenientes de estas dos posibles implementaciones de la cola de tareas ejecutables y su relación con los protocolos SRP y DFP puede encontrarse en la sección 3.3.

```
pragma Priority_Specific_Dispatching
            (EDF_Across_Priorities, 20, 25);

task EDF_Task_A with Priority => 22;
-- la prioridad 22 esta en el rango EDF, por tanto:
-- la prioridad de la tarea es 20
-- el nivel de expulsion de la tarea es 22

protected EDF_Object with Priority => 24 is ...
-- la prioridad 24 esta en el rango EDF, por tanto
-- la prioridad del objeto protegido es 20
-- el nivel de expulsion del objeto protegido es 24

task EDF_Task_B with Priority => 25;
-- la prioridad 25 esta en el rango EDF, por tanto:
-- la prioridad de la tarea es 20
-- el nivel de expulsion de la tarea es 25
```

Figura 2.10: Ejemplo de asignación de prioridades en Ada con EDF & SRP.

Capítulo 3

Evaluación del DFP en MaRTE OS

3.1. Implementación del DFP en MaRTE OS

Tal y como se ha explicado en la sección 2.2, la política de planificación EDF ya está implementada en MaRTE, por lo tanto para poder evaluar el DFP solo hace falta añadir y modificar los aspectos propios de este protocolo.

El estándar POSIX define las políticas de sincronización de herencia y de techo de prioridad. Para incorporar el DFP a MaRTE OS hay que añadir una nueva política de sincronización.

En la sección 2.1.4 se puede ver como el DFP no añade ningún parámetro de planificación a las tareas (a parte de los propios de tareas EDF). Así, solo es necesario modificar los *mutex*, que como se ha explicado anteriormente, son los mecanismos con los que se protegen los recursos en MaRTE OS. Para el protocolo DFP, solo es necesario añadir el campo suelo de plazo (Deadlinefloor), que es del mismo tipo que los plazos de las tareas, y el plazo original de la tarea que toma toma el *mutex* (Owner_Deadline). En la Figura 3.1 se pueden ver los campos necesarios para la implementación del protocolo DFP en MaRTE OS.

```
type Mutex record
   ...
   Deadlinefloor : HWTime := 0;
   Owner_Deadline : HWTime := 0;
end record;
```

Figura 3.1: Campos necesarios para la implementación del DFP en MaRTE OS.

3.1.1. Acceso a un recurso con DFP

De la misma forma que se hace en la sección 2.2.2, se describe ahora el proceso completo de toma y liberación de un *mutex*. Así se pueden observar los cambios introducidos a los procedimientos Running_Task_Locks_Mutex y Task_Unlock_Mutex para implementar el DFP.

Cuando una tarea toma un recurso protegido con un *mutex* que implementa el protocolo DFP, se ejecuta el procedimiento Running_Task_Locks_Mutex (Figura 3.2). En primer lugar se almacena el valor original del plazo de la tarea. Se calcula el plazo que podría heredar la tarea del *mutex* (Heir_Deadline) como la suma del momento en el que la tarea accede al recurso más el suelo de plazo del *mutex*. Se compara este plazo con el plazo de la tarea y en el caso de que el plazo heredado fuera menor o más próximo en el tiempo, se cambia el valor del plazo de la tarea.

```
procedure Running_Task_Locks_Mutex (Mutex) is
   ...
   Mutex.Owner_Deadline := Task.Deadline;
   Heir_Deadline := Now + Mutex.Deadlinefloor;
   if Task.Deadline > Heir_Deadline then
     Task.Deadline := Heir_Deadline;
   end if;
   ...
end Running_Task_Locks_Mutex;
```

Figura 3.2: Pseudocódigo de la función Running_Task_Locks_Mutex para el protocolo DFP.

Aunque el plazo de la tarea ha podido cambiar, haciendo a la tarea incluso más urgente no es necesario reordenar la cola de tareas ejecutables, ya que se trata de la tarea que se está ejecutando, es decir, la primera de la cola, la más urgente.

Es importante observar que para calcular el valor del plazo que la tarea puede heredar es necesario conocer el momento actual (el momento en el que la tarea accede al recurso) y por tanto es necesario hacer una lectura del reloj del sistema.

Cuando una tarea abandona un recurso protegido con un *mutex* se ejecuta el procedimiento Task_Unlocks_Mutex (Figura 3.3). Se restaura el valor del plazo de la tarea que se había almacenado en un campo del *mutex* y se ejecuta el procedimiento Reorder_and_Dispatch.

```
procedure Task_Unlocks_Mutex (Task, Mutex) is
   ...
   Task.Deadline := Mutex.Owner_Deadline;
   Reorder_and_Dispatch (Task);
   ...
end Task_Unlocks_Mutex;
```

Figura 3.3: Pseudocódigo de la función Task_Unlocks_Mutex para el protocolo DFP.

El plazo de la tarea puede haber cambiado otra vez, pero en este caso haciendo a la tarea menos urgente, por lo tanto sí es necesario reordenar la tarea en la cola de tareas ejecutables. Igual que en el caso del SRP, el procedimiento Reorder_and_Dispatch realiza esta reaordenación llamando a la función < (Figura

3.4). En el caso de tareas EDF con el protocolo DFP se compara simplemente el plazo absoluto tal y como se expone en la relación de orden 2.3.

El orden en el que se va comparando la tarea actual con el resto depende fuertemente de la estructura de datos utilizada, en la sección 3.2 se explica esto con más detalle.

```
function <  Left_Task, Right_Task) return Boolean is
  ...
  return Left_Task.Deadline < Right_Task.Deadline;

end <;
```

Figura 3.4: Pseudocódigo de la función < para el protocolo DFP.

Se observa que en todo el proceso de toma y liberación de un *mutex* con DFP no ha sido necesario tener en cuenta en ningún momento si alguna de las tareas tiene algún *mutex* tomado o no.

3.1.2. Interfaz DFP

Además de implementar funcionalmente el DFP se ha generado una interfaz completa para gestionar los parámetros específicos de este protocolo (Figura 3.5). Se observa que hay funciones para establecer y leer el suelo de plazo de los *mutex* tanto dinámicamente como a través de su objeto de atributos.

```
int pthread_mutexattr_setdeadlinefloor
   (pthread_mutexattr_t *attr,
    struct timespec *deadlinefloor);
int pthread_mutexattr_getdeadlinefloor
   (pthread_mutexattr_t *attr,
    struct timespec *deadlinefloor);

int pthread_mutex_setdeadlinefloor
   (const pthread_mutex_t *mutex,
    struct timespec *new_deadlinefloor,
    struct timespec *old_deadlinefloor);
int pthread_mutex_getdeadlinefloor
   (const pthread_mutex_t *mutex,
    struct timespec *deadlinefloor);
```

Figura 3.5: Interfaz del DFP en MaRTE OS.

3.2. Estructuras de datos para la cola de tareas ejecutables

En la sección 2.2 se explica como MaRTE OS emplea un *array* de colas de tareas ejecutables (una por cada nivel de prioridad del sistema). En cada cola

se ordenan las tareas activas con esa prioridad en el orden que establezca su política de planificación.

Además en la sección 2.1 se han visto las relaciones de orden que establecen los sistemas EDF & SRP y EDF & DFP. Estas son las relaciones que implementan las funciones $<$ para los dos sistemas expuestas en las Figuras 2.7 y 3.4.

Lo que se expone a continuación es la justificación del uso de unas determinadas estructuras de datos para implementar la cola de tareas ejecutables para cada uno de los dos protocolos.

De la relación de orden 2.1 se deduce a simple vista que para una tarea en un sistema EDF & SRP es más complejo encontrar su lugar en cola de tareas ejecutables, ya que en cada comparación con otra tarea del sistema tiene que tener en cuenta dos parámetros de planificación: el plazo y el nivel de expulsión. Esta doble comparación es la que hace a esta relación de orden no total y obliga a empezar la búsqueda desde el final de la cola para poder asegurar que la cola cumple la relación de orden que impone el sistema EDF & SRP. Para demostrar esto, se plantea el siguiente ejemplo.

Sean las tareas TA, TB y TC cuya ejecución en un sistema EDF & SRP se puede ver en la Figura 3.6. La tarea TA accede al recurso R1. Las tareas TB y TC no acceden a ningún recurso.

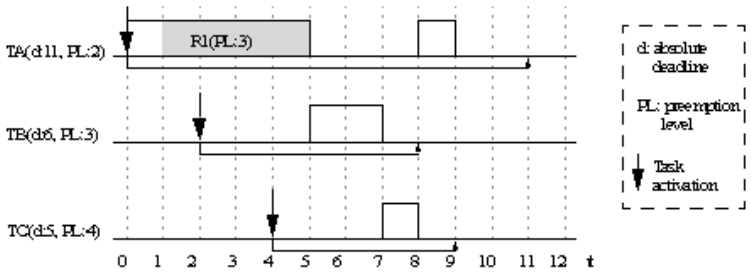

Figura 3.6: Orden de ejecución esperado para las tareas TA, TB y TC en un sistema EDF & SRP.

En la Figura 3.7 se puede ver el estado de la cola de tareas ejecutables con el paso del tiempo. La cabeza de la cola es el elemento más a la izquierda y el último elemento de la cola es el que está más a la derecha.

En el momento $t = 4$ se activa la tarea TC.

- Si se empieza a ordenar en la cola de tareas desde el final, la primera tarea con la que se encuentra es la tarea TB.

  ¿Es TC $\leq$ TB ? Como TB no tiene ningún recurso tomado de acuerdo con 2.1 solo hay que comparar los plazos de las tareas. Así se observa que $d_c > d_b$ y por tanto TC $\leq$ TB es falso, lo que lleva al orden de ejecución esperado expuesto en la Figura 3.6.

- Si se empieza a ordenar en la cola de tareas desde el principio, la primera tarea con la que se encuentra es la tarea TA.

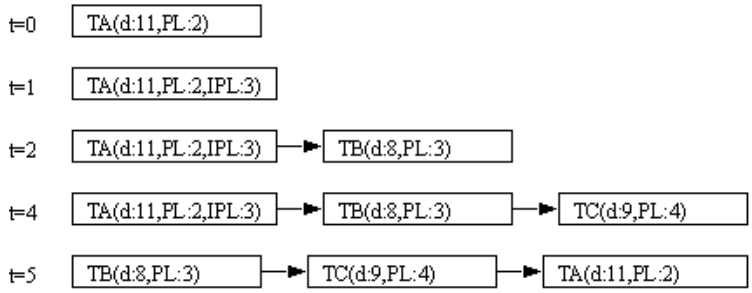

Figura 3.7: Estado de la cola de tareas ejecutables.

¿Es TC $\leq$ TA ? Como TA tiene un recurso tomado de acuerdo con 2.1 hay que comparar tanto los plazos de las tareas como sus niveles de expulsión. Así se observa que $d_c < d_a$ y $PL_c > PL_a$ y por tanto TC $\leq$ TA es verdadero, lo que hace que la tarea TC expulse a la tarea TA y empiece a ejecutarse en su lugar.

Esto es erróneo ya que entonces la tarea TC se ejecuta antes que la tarea TB en un claro caso de inversión de prioridad. Es otro ejemplo de que la relación de orden que impone un sistema EDF & SRP no es de orden total, ya que no verifica la propiedad transitiva expuesta en 2.2: TC $<$ TA y TA $<$ TB $\nRightarrow$ TC $<$ TB.

Queda demostrado que la relación de orden que establece entre las tareas un sistema EDF & SRP obliga a que, cuando una tarea busca su lugar en la cola de tareas ejecutables, empiece por el final de la cola. De otra forma no se puede garantizar que las tareas en la cola queden ordenadas de acuerdo a la relación 2.1. Esta limitación va a impedir utilizar algunas estructuras de datos para la cola de tareas ejecutables.

Por su parte, la relación de orden que impone un sistema EDF & DFP (2.3) sí es de orden total y no impone ninguna restricción de este tipo.

Así, se realizan tres implementaciones de MaRTE OS con tres estructuras de datos: listas doblemente enlazadas, montículos binarios y colas múltiples. A continuación se explican los detalles de cada una de las estructuras de datos utilizadas y se especifica para qué protocolo se han utilizado.

3.2.1. Lista doblemente enlazada

Una lista doblemente enlazada o *Doubly Linked List* (DLL) es una estructura de datos que consiste en una serie de nodos doblemente enlazados. Cada nodo contiene un enlace al nodo anterior y posterior de la cola. Una DLL puede utilizarse para implementar una cola de prioridad si se mantienen ordenados los elementos en la lista de acuerdo a la relación de orden deseada.

En esta estructura, el tiempo necesario para insertar o eliminar un elemento cualquiera crece linealmente con el número de elementos encolados, dicho de otra manera, las operaciones de inserción y extracción tienen una complejidad temporal de $O(n)$, siendo n el número de elementos en la cola. Por su parte, las operaciones para extraer el primer o el último elemento tienen una complejidad temporal de $O(1)$.

Una DLL puede utilizarse para implementar la cola de tareas ejecutables tanto de sistemas EDF & SRP y como EDF & DFP, ya que, para añadir un tarea a la cola, basta con con empezar desde el final y ir aplicando la relación de orden hasta encontrar el lugar apropiado de la tarea.

3.2.2. Montículo binario

Un montículo binario o *Heap* es una estructura de datos de tipo árbol binario con dos restricciones adicionales:

- Propiedad de montículo: Cada nodo contiene un valor superior al de sus hijos (para un montículo de máximos) o más pequeño que el de sus hijos (para un montículo de mínimos).
- Árbol semicompleto: Todos los niveles están completos, salvo quizá el último, que se llena de izquierda a derecha.

Esta estructura permite implementar colas de prioridad de una forma mucho más eficiente que utilizando listas doblemente enlazadas, ya que la inserción y extracción de un elemento es de $O(log(n))$.

Para implementar la cola de tareas ejecutables en un sistema EDF & DFP se utiliza un montículo de mínimos como el de la Figura 3.8, donde los nodos son tareas y los números representan los plazos absolutos de las tareas (que son el valor utilizado como criterio de ordenación).

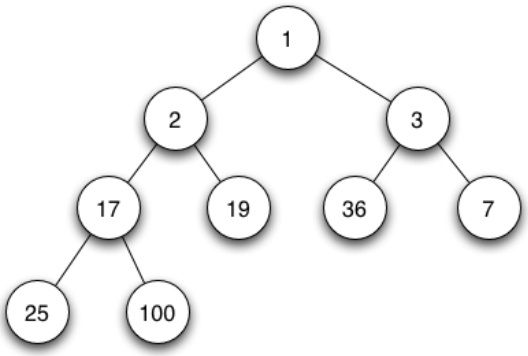

Figura 3.8: Montículo binario de mínimos.

La cola de tareas ejecutables de un sistema EDF & SRP no puede implementarse con un montículo binario, puesto que esta estructura requiere una relación de orden total, a continuación se explica porque.

Cuando se extrae un elemento, hay que reordenar los elementos de la cola para que no haya "huecos" en el árbol. Esto se hace colocando el último elemento de la cola en el lugar del elemento de extraído y comenzando el proceso de "hundimiento" del elemento hasta que encuentre su lugar. El proceso de "hundimiento" requiere que el elemento se compare con los demás en el orden de más a menos urgentes, lo que es contrario al orden de búsqueda de la posición de un elemento en la cola requerido por el algoritmo SRP. Una explicación más

detallada del funcionamiento de los montículos binarios puede encontrarse en [13].

3.2.3. Colas múltiples

La estructura de colas múltiples está inspirada por la implementación que sugiere el estándar Ada de un sistema EDF & SRP. Tal y como se explica en la sección 2.3.2, Ada incorpora el concepto de bandas de prioridad para cada política de planificación. Para una banda de prioridad planificada con EDF se define una cola de prioridad base en la que se sitúan las tareas cuando no hay recursos tomados en el sistema y colas de prioridades superiores donde se pueden situar las tareas cuando hay recursos tomados en el sistema.

Esta estructura no tiene sentido para sistemas EDF & DFP y solo es aplicable a una implementación de sistema EDF & SRP. Es interesante evaluar su rendimiento por tratarse de la estructura sugerida por el estándar de Ada.

Para la cola de prioridad base se decide emplear un montículo binario, ya que, como se ha visto, es una implementación eficiente de una cola ordenada. Sin embargo para las colas de prioridades superiores, se decide utilizar listas simplemente enlazadas.

Una lista simplemente enlazadas o *Singly Linked List* (SLL) es una estructura de datos que consiste en una serie de nodos simplemente enlazados. Cada nodo contiene un enlace al nodo posterior de la cola. La inserción y eliminación de un elemento cualquiera de la cola es $O(n)$ si se empieza la búsqueda por el primer elemento de la cola. La inserción y eliminación del elemento que está en la cabeza es $O(1)$.

Aunque, para un uso general, esta estructura es poco eficiente, su uso se justifica si se tiene en cuenta que, de acuerdo con el algoritmo propuesto en el estándar Ada, en las colas de prioridad superior al básico solo se insertan y se extraen elementos de la cabeza de la cola (la explicación puede encontrarse en la sección 2.3.2). En la Figura 3.9 puede verse un esquema de la estructura de la estructura de colas múltiples.

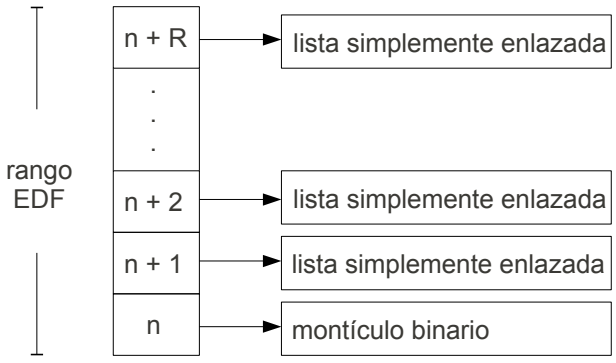

Figura 3.9: Estructuras de colas múltiples. El rango de prioridades EDF va desde la prioridad base, n, hasta la prioridad $n + R$

3.3. Comparación de los protocolos SRP y DFP en MaRTE OS

Una vez completada la implementación del protocolo DFP en MaRTE OS, se realizan las pertinentes comparaciones con el SRP. Se llevan a cabo diferentes pruebas para medir el rendimiento de las combinaciones elegidas de protocolo de sincronización-estructura de datos para la cola de tareas ejecutables.

Además se hace una comparación entre las propias implementaciones de los protocolos, ya que, como se viene diciendo en este trabajo, el DFP es un protocolo conceptualmente más sencillo que el SRP y esto debe notarse en el mismo código.

3.3.1. Complejidad de la implementación

Un indicador de la complejidad de un protocolo es el número de líneas de código que han sido necesarias para su implementación, así como el número de funciones y campos que ha sido necesario incorporar al sistema operativo.

En la Tabla 3.1 se puede observar el resumen de las implementaciones de los protocolos SRP y DFP en MaRTE OS. Es importante destacar que las partes de la implementación que son comunes a ambos protocolos no se han tenido en cuenta, como por ejemplo las estructuras de datos.

	SRP	*DFP*
Campos de los mutex	2	2
Operaciones de los mutex	4	4
Campos de las tareas	1	0
Operaciones de las tareas	4	0
Líneas de código	54	34

Cuadro 3.1: Resumen de las implementaciones del SRP y DFP en MaRTE OS.

Se observa como la implementación del SRP necesita añadir a las tareas un parámetro nuevo, el nivel de expulsión así como funciones para establecerlo y leerlo. El número total de líneas de código también es mayor para el SRP debido básicamente a la inclusión de estas funciones.

3.3.2. Pruebas de rendimiento

La comparación del rendimiento de los protocolos DFP y SRP se realiza sobre cuatro combinaciones diferentes de protocolo y estructura de datos:

- SRP & DLL: Protocolo SRP con listas doblemente enlazadas para la cola de tareas ejecutables. Es la implementación natural y más sencilla del SRP en MaRTE OS.
- SRP & Heap + SLL: Protocolo SRP con colas múltiples para la cola de tareas ejecutables. Es una implementación alternativa del SRP que resulta de interés por ser la sugerida por el estándar Ada.
- DFP & DLL: Protocolo DFP con listas doblemente enlazadas para la cola de tareas ejecutables. Es interesante para poder compararla con la

implementación SRP & DLL, es decir, comparar los dos protocolos sobre una misma estructura de datos.

- DFP & Heap: Protocolo DFP con montículos binarios para la cola de tareas ejecutables. Se trata de la implementación con la estructura de datos más eficiente. Es importante recordar que el montículo binario no puede ser utilizado con el SRP (ver sección 3.2).

Para evaluar el rendimiento del SRP y DFP se diseñan programas que se ejecutan sobre las cuatro implementaciones diferentes de los protocolos en MaRTE OS. Todos los experimentos se llevan a cabo sobre un Pentium III a 800 MHz. Son los siguientes:

- Test A: Un recurso en el sistema y muchas tareas. Una tarea con plazo relativo corto accede al recurso un millón de veces. El resto de tareas tienen un plazo relativo mucho mayor y no acceden al recurso. Se mide el tiempo de ejecución de la tarea de plazo relativo corto, lo que permite calcular el tiempo promedio en tomar y liberar un recurso.

- Test B: Un recurso en el sistema y muchas tareas. Una tarea con plazo relativo corto accede al recurso un millón de veces. El resto de tareas tienen un plazo relativo mucho mayor y no acceden al recurso. Es como el Test A, solo que no se mide el tiempo de ejecución total de la tarea, sino solo el tiempo de liberación del recurso.

- Test C: Ningún recurso en el sistema y muchas tareas. Una tarea con plazo relativo corto se activa periódicamente. El resto de tareas tienen un plazo relativo mucho mayor y un tiempo de ejecución infinito. En cada activación, la tarea de plazo relativo corto hace una lectura de reloj y se calcula la diferencia entre ese momento y el momento de activación teórico de la tarea. Esto da una idea del tiempo empleado en realizar el cambio de contexto.

Del análisis de los protocolos y sus implementaciones en MaRTE OS, se deduce que los factores más importantes en cuanto al consumo de tiempo son el reordenado de la tarea en la cola de tareas ejecutables cuando se libera el recurso (para ambos protocolos) y la lectura del reloj que necesita hacer el protocolo DFP al tomar un recurso. Los Test A y B permiten separar estos dos factores y así analizar su influencia individualmente.

Además, uno de los estos dos factores (la necesidad de encontrar la posición de una tarea en la cola) se produce también en muchas otras situaciones diferentes de la liberación de un recurso, como por ejemplo, cada vez que una tarea se activa (independientemente de que haya recursos en el sistema o no). Precisamente esta situación es la estudiada en el Test C, que permite analizar al influencia del protocolo de sincronización empleado en un sistema, incluso sin recursos en el mismo.

3.3.3. Resultados

La principal diferencia entre los Test A y B se puede apreciar en las Figuras 3.10 y 3.11 para menos de 10 tareas en el sistema, es decir, las gráficas de la izquierda.

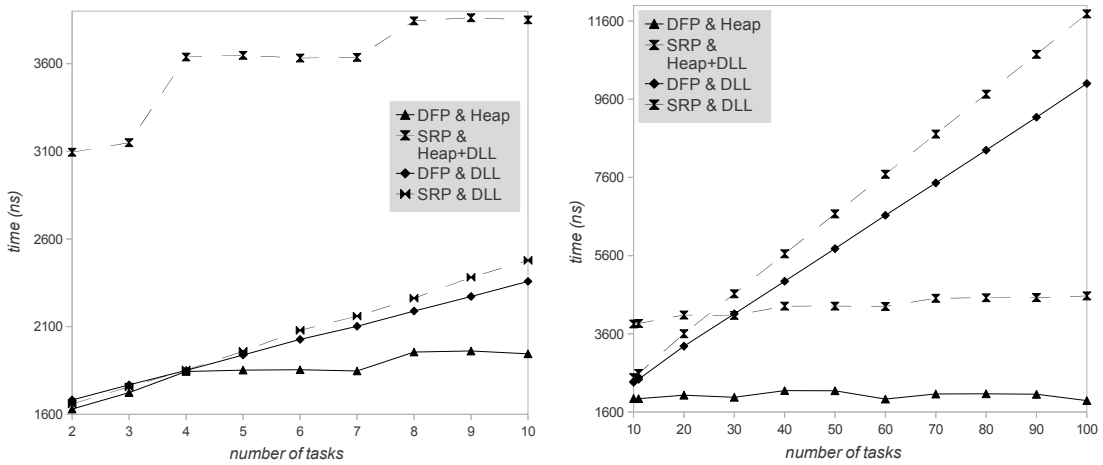

Figura 3.10: Resultados del Test A. Izquierda: hasta 10 tareas. Derecha: desde 10 a 100 tareas.

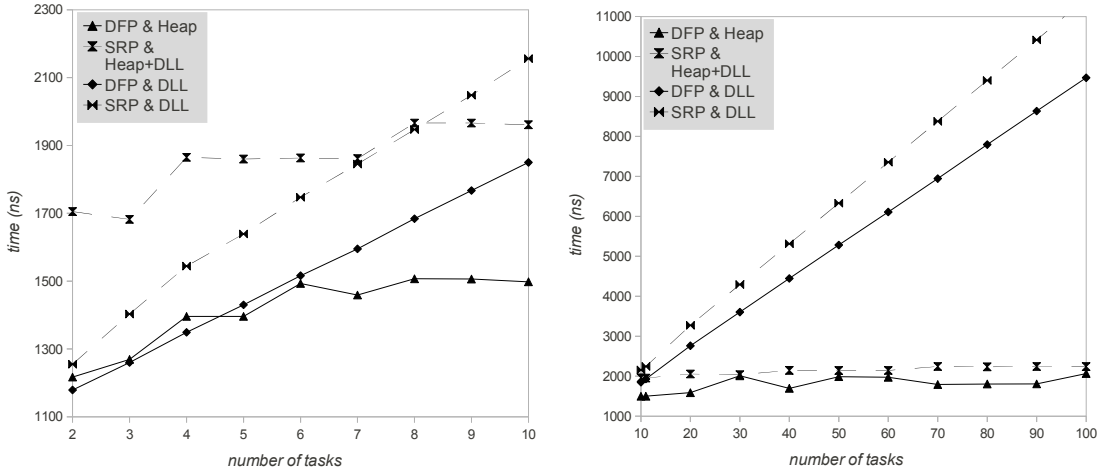

Figura 3.11: Resultados del Test B. Izquierda: hasta 10 tareas. Derecha: desde 10 a 100 tareas.

Así en la Figura 3.11 se puede observar como entre los dos sistemas que usan listas doblemente enlazadas, el que implementa el DFP (DFP & DLL) obtiene mejores resultados que el que implementa el SRP (SRP & DLL), sin embargo, en la Figura 3.10 se observa que, si bien el DFP sigue teniendo mejor rendimiento que el SRP, esta diferencia ha disminuido, lo que indica que la operación de toma del recurso es menos eficiente para sistemas con DFP que para sistemas con SRP. Esto es algo que ya se podía prever teniendo en cuenta que el DFP requiere una lectura de reloj cada vez que una tarea toma un recurso, tal y como se puede ver en la Figura 3.2.

Para más de 10 tareas en el sistema, es decir, las gráficas de la derecha de las Figuras 3.10 y 3.11 se observa como la diferencia entre las dos implementaciones va creciendo con el número de tareas. La razón se encuentra en la función de comparación empleada para ordenar las tareas en la cola de tareas ejecutables, que es más sencilla para el protocolo DFP (Figura 3.4) que para el SRP (Figura 2.7).

Es significativo que la implementación del protocolo DFP haya obtenido mejores resultados que la del SRP usando ambos la misma estructura para la cola de tareas ejecutables. Esto hace pensar que para otras estructuras de datos, tal vez más eficientes que las DLL, que pudieran ser usadas para ambos protocolos, el resultado sería el mismo, es decir, el DFP seguiría obteniendo mejores resultados que el SRP.

Por su parte, se observa como el sistema implementado con un montículo binario (DFP & Heap), obtiene los mejores resultados entre las cuatro implementaciones realizadas. Para menos de 5 tareas en el sistema, no hay demasiada diferencia con los sistemas implementados con listas doblemente enlazadas, pero conforme el número de tareas en el sistema va a aumentando, el comportamiento logarítmico del montículo binario empieza a mostrar sus beneficios.

La implementación del sistema con colas múltiples (SRP & Heap + SLL) también muestra un comportamiento logarítmico como consecuencia de haber usado un montículo binario para la cola de prioridad base. Sin embargo esta estructura solo es ventajosa respecto a las listas doblemente enlazadas cuando el sistema tiene más de 30 tareas, lo cual es ya un sistema bastante grande, y en ningún momento mejora al sistema DFP & Heap.

En la Figura 3.12 se pueden ver los resultados del Test C. Este test resulta interesante ya que presenta la situación de un sistema en el que no hay recursos compartidos. Se observa que el comportamiento de las diferentes implementaciones es el mismo que se aprecia en los Test A y B.

La implementación que mejor rendimiento presenta es la del DFP & Heap y entre los sistemas con listas doblemente enlazadas, el que implementa el DFP (DFP & DLL) vuelve a ser más eficiente que el que implementa el SRP (SRP & DLL). La diferencia de rendimiento entre las dos implementaciones con DLL se debe exclusivamente a la diferente función de comparación que emplean el SRP y el DFP, ya que, incluso sin recursos tomados en el sistema, la implementación SRP & DLL tiene que constatar ese hecho a cada paso en la operación de encolado de las tareas.

Es interesante señalar un efecto que se observa en las Figuras 3.10, 3.11 y 3.12 para menos de 10 tareas en el sistema (gráficas de la izquierda). Las implementaciones que usan un montículo binario (DFP & Heap y SRP & Heap + DLL) presentan unos saltos bruscos al pasar de 3 a 4 y de 7 a 8 tareas en el sistema. Esto se debe a los pisos de la estructura del montículo binario

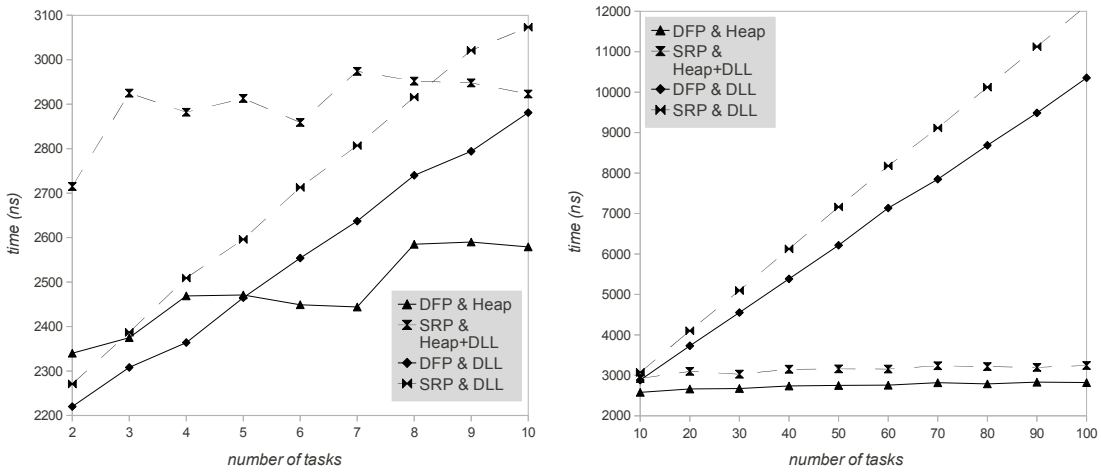

Figura 3.12: Resultados del Test C. Izquierda: hasta 10 tareas. Derecha: desde 10 a 100 tareas.

(Figura 3.8). Así cuando hay hasta 3 tareas en el sistema solo hay dos pisos del montículo ocupado y de 4 a 7 tareas en el sistema se ocupan tres pisos. Los siguientes cambios de piso no se aprecian porque son proporcionalmente menos significativos.

Capítulo 4

Incorporación del DFP al lenguaje Ada

La implementación y el análisis del DFP en MaRTE OS vista en el capítulo 3 pone en evidencia que se trata de un protocolo más sencillo y eficiente que el SRP. La implementación más eficiente que se ha encontrado del DFP obtiene incluso mejores resultados que la implementación del SRP con la cola de tareas ejecutables que propone el estándar Ada.

Así, se decide extender el análisis del DFP para evaluar su incorporación al estándar de Ada. En este capítulo por tanto, se analizan los cambios que sería necesario llevar a cabo en las políticas de sincronización de acceso a los objetos protegidos y a la propia política de planificación EDF. Se estudian diferentes alternativas para cada cambio necesario y se explican las ventajas e inconvenientes de cada una.

La mayor parte de esta propuesta ha sido presentada en el IRTAW 2013 [14], donde ha obtenido un visto bueno preliminar. Se han añadido detalles sobre algunos aspectos allí tratados y se pretende presentar una propuesta completa al IRTAW 2014.

4.1. Modificaciones a la política de planificación EDF

Tal y como se explica en la sección 2.3.2, la política de planificación EDF está incorporada al estándar Ada desde el 2005, con el identificador EDF_Across_Priorities y hace uso del protocolo SRP.

Se podría incorporar el protocolo DFP al estándar de tres formas diferentes:

1. Reemplazar la definición actual de EDF_Across_Priorities por una nueva que use el DFP en vez del SRP.

2. Añadir una nueva política de planificación llamada EDF_With_Deadline_Floor que defina las reglas para un sistema EDF con el protocolo DFP. Declarar la política EDF_Across_Priorities obsoleta.

3. Añadir una nueva política de planificación llamada EDF_With_Deadline_Floor que defina las reglas para un sistema EDF con el protocolo DFP.

Mantener la política EDF_Across_Priorities en el estándar y que cada implementación decida cual de las dos quiere usar.

La primera opción es muy radical y haría muy complicado el paso de un protocolo a otro para las aplicaciones ya existentes. Teniendo en cuenta que no se han encontrado ventajas del SRP sobre el DFP se considera que la mejor opción es la segunda, algo en lo se estuvo de acuerdo en el IRTAW 2013.

El nuevo identificador propuesto EDF_With_Deadline_Floor puede ser usado con los pragmas Task_Dispatching_Policy y Priority_Specific_Dispatching que aparecen en la Figura 4.1.

```
pragma Task_Dispatching_Policy
                              (EDF_With_Deadline_Floor);
pragma Priority_Specific_Dispatching
                              (EDF_With_Deadline_Floor,
                               First_Priority,
                               Last_Priority);
```

Figura 4.1: Uso del identificador EDF_With_Deadline_Floor.

El pragma Priority_Specific_Dispatching establece la política de planificación para la banda de prioridades acotada entre First_Priority y Last_Priority. Aunque tal y como se ha explicado, el protocolo DFP no necesita bandas de prioridad para su implementación, es interesante mantener este pragma tanto para incrementar la compatibilidad con aplicaciones existentes como para mantener la analogía con otras políticas de planificación.

Así, en el caso establecer la política de planificación EDF_With_Deadline_Floor en un rango de prioridades, el manual de referencia de Ada debería establecer que la prioridad de todas las tareas en ese rango caiga a la prioridad base del rango.

4.1.1. Plazo relativo de las tareas

Actualmente el plazo relativo aparece en el lenguaje Ada como un aspecto de las tareas (Relative_Deadline). Se utiliza solo para establecer el primer plazo absoluto tras su activación y no existen funciones o procedimientos para leer el plazo relativo de una tarea. En cada activación posterior es necesario llamar al procedimiento Delay_Until_and_Set_Deadline (ver Figura 2.9) que como su nombre indica, suspende la tarea hasta el momento indicado por el parámetro Delay_Until_Time y le asigna un plazo absoluto igual a la suma de Delay_Until_Time y Deadline_Offset. Este último parámetro funciona como plazo relativo, aunque su valor no se almacena en ninguna parte y por tanto no puede ser recuperado.

Hasta ahora no existía la necesidad de que el plazo relativo fuera un parámetro fundamental de las tareas, ya que la planificación EDF se hace en base al plazo absoluto. Sin embargo, el protocolo DFP necesita comprobar los plazos relativos de las tareas para verificar que no se produzcan violaciones del mismo, es decir, que ninguna tarea acceda a un objeto protegido con un suelo de plazo menor que su plazo relativo.

Así otro cambio necesario para incorporar el DFP al lenguaje Ada es introducir el plazo relativo como un parámetro fundamental para las tareas EDF. Para ello se crea un nuevo subtipo, Relative_Deadline del tipo Ada.Real_Time.Time_Span y funciones para leerlo y establecerlo dinámicamente. Además, es necesario modificar ligeramente el procedimiento Delay_Until_and_Set_Deadline de forma que el parámetro New_Relative_Deadline que se le pasa se almacene como el plazo relativo de la tarea (por eso se cambia su nombre). Además, como es posible que en la mayoría de las activaciones no se desee cambiar el plazo relativo de la tarea se crea un nuevo procedimiento con el mismo nombre, pero que no requiere de un segundo parámetro, sino que usa el plazo relativo de la tarea para calcular su siguiente plazo absoluto. En la Figura 4.2 se pueden ver las modificaciones propuestas al paquete Ada.Dispatching.EDF.

```
with Ada.Real_Time;
with Ada.Task_Identification;

package Ada.Dispatching.EDF is
   ...

   subtype Relative_Deadline is Ada.Real_Time.Time_Span;

   procedure Delay_Until_And_Set_Deadline
             (Delay_Until_Time : in Ada.Real_Time.Time;
              New_Relative_Deadline : in Relative_Deadline);

   procedure Delay_Until_And_Set_Deadline
             (Delay_Until_Time : in Ada.Real_Time.Time);

end Ada.Dispatching.EDF;
```

Figura 4.2: Modificaciones del Paquete Ada.Dispatching.EDF.

4.1.2. Cambio dinámico de plazo

En la Figura 4.3 pueden verse los detalles del paquete Ada.Dispatching.EDF .Dynamic_Relative_Deadlines, que se propone incorporar al estándar Ada. En él se introducen procedimientos para establecer y leer el plazo relativo de las tareas. Como se ha dicho anteriormente la inclusión de este paquete resulta fundamental para la incorporación del DFP al lenguaje Ada, sin embargo su uso podría llevar a violaciones del protocolo DFP.

	a	D	d
τ_a	0	10	10
τ_b	2	5	7
τ_c	2	10	12

Cuadro 4.1: Parámetros de planificación de las tareas τ_a, τ_b y τ_c

Por ejemplo, sean las tareas τ_a y τ_b cuyos parámetros aparecen en el Cuadro

```
with Ada.Real_Time;
with Ada.Task_Identification;

package Ada.Dispatching.EDF.Dynamic_Relative_Deadlines is

   procedure Set_Relative_Deadline
             (D : in Real_Time.Time_Span;
              T : in Ada.Task_Identification.Task_Id :=
                     Ada.Task_Identification.Current_Task);

   function Get_Relative_Deadline
             (T : Ada.Task_Identification.Task_Id :=
                  Ada.Task_Identification.Current_Task)
            return Real_Time.Time_Span;

end Ada.Dispatching.EDF.Dynamic_Relative_Deadlines;
```

Figura 4.3: Primitivas para el aspecto Relative_Deadline de las tareas.

4.1. Un esquema de su ejecución puede verse en la Figura 4.4. Se observa como la tarea τ_a accede durante su ejecución a un recurso cuyo suelo de plazo es 10. Cuando la tarea τ_b se activa expulsa a la tarea τ_a mientras esta se encuentra ejecutando dentro del recurso. Si la tarea τ_b quisiera entonces acceder al recurso se produciría una violación del suelo de plazo y sería detectado por el protocolo DFP y evitado.

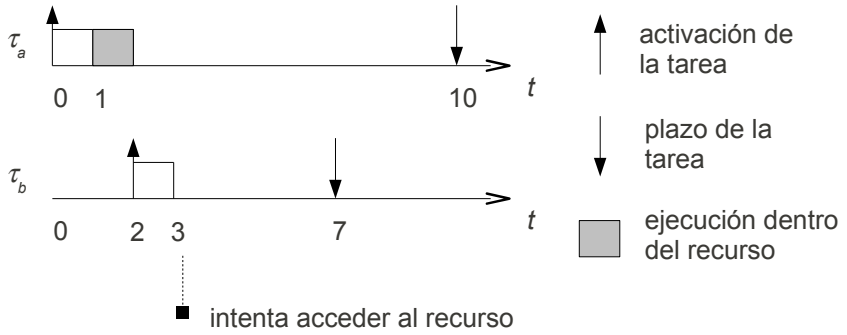

Figura 4.4: Ejemplo de ejecución de las tareas τ_a y τ_b cuyos parámetros de planificación pueden verse en el Cuadro 4.1

Sin embargo, cabe la posibilidad de que haciendo uso del procedimiento Set_Relative_Deadline el plazo relativo de la tarea τ_b se hubiera cambiado, por ejemplo a 10, con lo que podría acceder al recurso. En ese caso, si el plazo absoluto de la tarea τ_b no hubiera cambiado seguiría expulsando a la tarea τ_a y se daría la situación de que haya dos tareas al mismo tiempo ejecutando dentro del recurso.

Una situación semejante se podría dar con el procedimiento Set_Deadline (ver Figura 2.9). Por ejemplo, sean ahora las tareas τ_a y τ_c cuyos parámetros

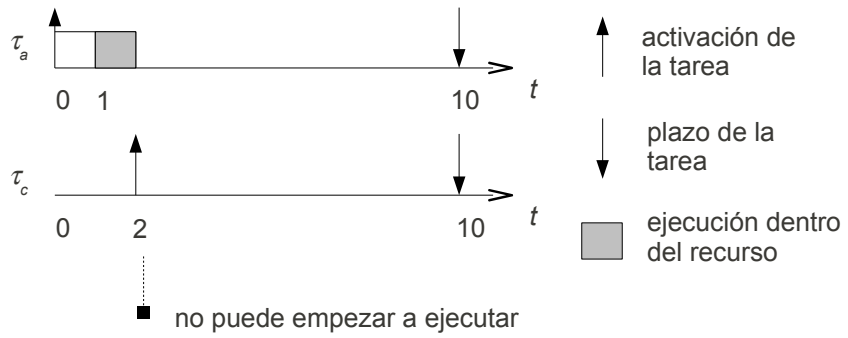

Figura 4.5: Ejemplo de ejecución de las tareas τ_a y τ_c cuyos parámetros de planificación pueden verse en el Cuadro 4.1

de planificación aparecen en el Cuadro 4.1. Un esquema de su ejecución puede verse en la Figura 4.5. Igual que antes, la tarea τ_a accede durante su ejecución a un recurso cuyo suelo de plazo es 10. Cuando la tarea τ_c se activa, su plazo es mayor que el de la tarea τ_a y por tanto no la puede expulsar.

Sin embargo, cabe la posibilidad de que haciendo uso del procedimiento Set_Deadline el plazo absoluto de la tarea τ_c se hubiera cambiado a 7, con lo que al activarse, expulsaría a la tarea τ_a. Entonces la tarea τ_c podría acceder al recurso ya que su plazo relativo es igual que el suelo de plazo del recurso. De esa forma se daría otra vez la situación de un acceso concurrente al recurso.

Estos comportamientos no son deseables y se podrían evitar de las siguientes maneras:

1. Proteger los recursos con primitivas de acceso exclusivo, como por ejemplo semáforos. Esto no es la solución ideal, ya que el protocolo DFP es potencialmente suficiente para asegurar que no se producen accesos concurrentes a recursos.

2. Modificar los procedimientos Set_Relative_Deadline y Set_Deadline para que, en el momento en el que son invocados, s, hagan las siguientes comprobaciones:

   - El procedimiento Set_Relative_Deadline antes de establecer el nuevo plazo relativo D_{nuevo}, debería comprobar que:

     $$d_a \geq D_{nuevo} + s \tag{4.1}$$

     donde d_a es el plazo absoluto actual de la tarea. En el caso de que 4.1 no se cumpliera el plazo relativo de la tarea no cambiaría.

   - Análogamente, el procedimiento Set_Deadline antes de establecer el nuevo plazo absoluto d_{nuevo}, debería comprobar que:

     $$d_{nuevo} \geq D_a + s \tag{4.2}$$

     donde D_a es el plazo relativo actual de la tarea. En el caso de que 4.2 no se cumpliera el plazo absoluto de la tarea no cambiaría.

Se observa que las comprobaciones que tienen que hacer los procedimientos Set_Relative_Deadline y Set_Deadline son la misma, teniendo en cuenta en cada caso el parámetro que modifican. Estas comprobaciones deberían ser suficientes para evitar los accesos concurrentes a los recursos compartidos. Sin embargo los plazos relativo y absoluto quedan desvinculados. Lo que lleva a la tercera opción.

3. Hacer que el plazo absoluto y relativo sean siempre dependientes el uno del otro. De la siguiente forma:

   - Si se desea cambiar el plazo relativo, el procedimiento Set_Relative _Deadline modifica también el plazo absoluto de la tarea de la siguiente manera:

     $$d_{nuevo} = d_a + (D_{nuevo} - D_a) \tag{4.3}$$

     donde d_a y D_a son respectivamente el plazo absoluto y relativo de la tarea antes su modificación.

   - Análogamente, si se desea cambiar el plazo absoluto, el procedimiento Set_Deadline modifica también el plazo relativo de la tarea de la siguiente forma:

     $$D_{nuevo} = D_a + (d_{nuevo} - d_a) \tag{4.4}$$

Nuevamente la primera opción no es recomendable, ya que se pierde una de las ventajas del protocolo DFP. La segunda y la tercera opción parecen igualmente válidas, pero la opción tres es más limpia y sencilla de comprender, por lo que esta será la opción que se propondrá en el IRTAW 2014.

4.2. Modificaciones de los protocolos de sincronización existentes

El protocolo de sincronización se establece para una partición entera con el pragma Locking_Policy. El identificador existente en el lenguaje Ada llamado Ceiling_Locking integra el protocolo de techo de prioridad y el SRP, que como ya se ha dicho, es una generalización del protocolo de techo de prioridad.

Para la incorporación del protocolo DFP al lenguaje Ada se propone mantener el identificador Ceiling_Locking. Esta decisión se justifica si se tiene en cuenta que el funcionamiento del DFP es análogo al del protocolo de techo de prioridad, solo que las tareas heredan el plazo además de la prioridad.

Aunque el identificador no cambie, si es necesario reescribir su definición para añadir la funcionalidad propia del DFP. Para ello hay que incorporar de alguna manera el parámetro del suelo de plazo para los objetos protegidos. Se plantean dos posibilidades:

1. Introducir un nuevo aspecto para asignar suelo de plazo a los objetos protegidos llamado Deadline_Floor tal y como se propone en la Figura 4.6.

```
protected Object with Deadline_Floor =>
                  Ada.Real_Time.Milliseconds(23) is ...
```

Figura 4.6: Aspecto Deadline_Floor para los objetos protegidos.

2. No introducir ningún aspecto nuevo. Reutilizar el aspecto ya existente para el plazo relativo de las tareas Relative_Deadline y aplicarlo a los objetos protegidos. Sería análogo al uso que se hace del aspecto que asigna la prioridad a las tareas (llamado Priority) que funciona como techo de prioridad cuando se le asigna a los objetos protegidos.

4.2.1. Cambio dinámico de suelo de plazo

Al introducir un nuevo aspecto en los objetos protegidos, el suelo de plazo (ya sea como un aspecto totalmente nuevo o reutilizando el aspecto Relative_Deadline de las tareas), hay que proporcionar mecanismos para establecerlo y leerlo dinámicamente sobre un objeto protegido ya existente.

Desde Ada 95 se permite a las tareas el cambio de prioridad dinámico con las operaciones que provee el paquete Ada.Dynamic_Priorities y en Ada 2005 se incluyó el atributo Priority para los objetos protegidos y se permitió a las aplicaciones cambiar dinámicamente su techo de prioridad.

De forma análoga se debe permitir que el suelo de plazo de un objeto protegido pueda ser cambiado dinámicamente definiendo el nuevo atributo Deadline como se muestra en la Figura 4.7.

Se observa que el suelo de plazo solo puede ser cambiado por una tarea que ejecuta dentro de una operación protegida. El suelo de plazo del objeto protegido cambia inmediatamente, pero la tarea que está ejecutando esa operación no cambia su plazo. El nuevo suelo de plazo afectará a las siguientes tareas que ejecuten operaciones dentro del objeto protegido.

```
protected body PO is
  procedure Change_Relative_Deadline
                          (D: in Real_Time.Time_Span) is
  begin
    ...
    -- El suelo de plazo tiene el valor antiguo
    PO'Deadline := D;
    -- El suelo de plazo tiene el valor nuevo
    ...
  end Change_Relative_Deadline;
  -- El cambio de suelo de plazo tiene efecto aqui
  ...
end PO;
```

Figura 4.7: Cambio de suelo de plazo para un objeto protegido.

4.2.2. *Rendezvous*

En Ada, para implementar las comunicaciones asíncronas se utiliza una primitiva de sincronización asimétrica llamada *rendezvous*. El *rendezvous* permite que dos tareas concurrentes intercambien datos de forma coordinada. La tarea que solicita el *rendezvous* debe esperar en el punto de reencuentro hasta que la tarea llamada llegue allí. Igualmente la tarea llamada puede llegar al *rendezvous* antes que la solicitante y debe esperar a que esta llegue al punto de encuentro para poder continuar ejecutando.

En el caso de que las dos tareas no tengan la misma prioridad, el *rendezvous* se ejecuta a la prioridad más alta de las dos tareas involucradas. Es decir, si la tarea solicitante tiene mayor prioridad que la tarea llamada, cuando la tarea solicitante llegue al punto de encuentro y la tarea llamada empiece a ejecutar, esta lo hará a la prioridad de la solicitante.

El estándar Ada en la sección D.1 establece que durante un *rendezvous* la tarea llamada hereda la prioridad de la tarea solicitante. Tal y como se ha explicado en la sección 2.3.2, para tareas EDF, esta prioridad es en realidad su nivel de expulsión.

Para incorporar el protocolo DFP es necesario que en un *rendezvous* las tareas hereden, además de la prioridad, el plazo absoluto. Así si el plazo absoluto de la tarea solicitante es más corto que el de la tarea llamada, cuando la tarea solicitante llegue al punto de encuentro y la tarea llamada empiece a ejecutar, esta debe hacerlo con el plazo absoluto de la solicitante. Este comportamiento debe ser incluido en la definición de la política EDF.

Además y tal y como se ha visto en la sección 4.1.2, al cambiar el plazo absoluto de una tarea, debe cambiarse también su plazo relativo, de acuerdo con la expresión 4.4.

4.3. Compatibilidad con aplicaciones existentes

Para que la adaptación de aplicaciones que usen la política EDF_Across_Priorities a la nueva EDF_With_Deadline_Floor no sea muy costosa, se propone permitir que, aunque no lo necesite, la política EDF_With_Deadline_Floor se defina en una banda de prioridades. Así solo sería necesario un cambio en el identificador de la política de planificación para que las aplicaciones existentes funcionaran correctamente. Eso sí, en este caso la prioridad de todas las tareas dentro de la banda caería hasta la prioridad más baja de la banda.

Además, tampoco sería estrictamente necesario asignar suelo de plazo a los objetos protegidos ya que por defecto el plazo para un objeto protegido sería cero. No obstante sí sería deseable hacerlo ya que así mejorarían los tiempos de respuesta de la aplicación al reducirse los bloqueos introducidos por los objetos protegidos.

4.4. Convivencia con otras políticas de planificación

En un sistema realista, es normal que existan tareas con diferentes políticas de planificación en diferentes bandas de prioridad. Además, estas tareas pueden

compartir datos a los que deben acceder de forma exclusiva. Para ello se protegen los datos con un objeto protegido, cuyo techo de prioridad debe estar correctamente asignado de acuerdo con el protocolo de techo inmediato de prioridad, PCP. Es decir, el techo de prioridad del objeto protegido debe ser la prioridad de la tarea de mayor prioridad que acceda al objeto protegido. Además, las políticas deben estar correctamente definidas para que el comportamiento del sistema sea el correcto, es decir, que no se den interbloqueos, ni inversiones de prioridad no acotada.

Se ha analizado la convivencia de la nueva política de planificación EDF_With_Deadline_Floor con las ya existentes: FIFO_Within_Priorities, Round_Robin_Within_Priorities. Incluso se han estudiado los casos particulares de la convivencia con otra franja EDF_With_Deadline_Floor o EDF_Across_Priorities.

- EDF_With_Deadline_Floor y FIFO_Within_Priorities: A las tareas FIFO se les asigna un plazo relativo infinito (Ada.Real_Time.Time_Span_Last).

  Si la banda EDF está a una prioridad más alta que la FIFO, una tarea de la banda FIFO puede acceder a recursos con prioridad EDF heredando tanto el techo de prioridad del recurso como su suelo de plazo. Así, la ausencia de interbloqueos y/o inversiones de prioridad queda garantizada por la propia definición del DFP.

  Si por el contrario la banda FIFO está a una prioridad más alta que la EDF, una tarea EDF puede acceder a recursos con prioridad FIFO heredando el techo de prioridad y en ese caso se mantienen las propiedades del protocolo de techo de prioridad.

- EDF_With_Deadline_Floor y Round_Robin_Within_Priorities: Se trata de la misma situación que la vista anteriormente para una banda FIFO, ya que las tareas *Round Robin* se comportan como tareas FIFO mientras están dentro del recurso.

- EDF_With_Deadline_Floor y EDF_With_Deadline_Floor: Cuando una tarea EDF accede a un recurso de otra banda de prioridad superior en la que también se aplica la política EDF, las tareas estarán compitiendo con plazos de diferentes bandas de prioridades. Es cuestionable si un sistema de estas características puede resultar de utilidad, pero en cualquier caso, la política propuesta funciona correctamente y es responsabilidad del usuario asignar correctamente los plazos a las tareas y suelos de plazo a los recursos del sistema para que no se produzcan interbloqueos o inversiones de prioridad.

- EDF_With_Deadline_Floor y EDF_Across_Priorities: Suponiendo que se permita coexistir a estas dos políticas en el mismo sistema, se trataría de la misma situación vista anteriormente para un sistema con dos bandas EDF del tipo EDF_With_Deadline_Floor.

Para cada caso se han elaborado programas de prueba en Ada que verifican el correcto funcionamiento de esta nueva política. Estos programas se han hecho usando los pragmas EDF_Across_Priorities y Ceiling_Locking. No se ha modificado el compilador para añadir la política de planificación EDF_With_Deadline_Floor ya que es una tarea de una complejidad que excede las dimensiones de

este trabajo y además se ha encontrado una manera alternativa de realizar las pruebas necesarias. Se explica a continuación.

MaRTE OS lleva una lista de los *mutex* tomados por una tarea en orden LIFO (*Last In, First Out*), es decir, el último tomado, se coloca el primero en la lista. Cuando una tarea accede a un objeto protegido, toma el *mutex* que lo protege y este se coloca el primero en la lista de *mutex* tomados por la tarea. El procedimiento Set_Deadlinefloor_Last_Mutex_Locked (Figura 4.8) cambia el protocolo del último mutex tomado por la tarea para que use el DFP y le asigna suelo de plazo. Así, si se ejecuta este procedimiento al principio de la aplicación y dentro del objeto protegido se consigue que el *mutex* que protege el objeto protegido use la implementación del DFP realizada en MaRTE OS vista en 3.1.

```
procedure Set_Deadlinefloor_Last_Mutex_Locked
                              (Deadlinefloor : in Duration) is
  M := Mutex_Lists.Head(Self.Mutex_Owned));
begin
  M.Policy := DEADLINE_FLOOR;
  M.Deadlinefloor := Deadlinefloor;
end Set_Deadlinefloor_Last_Mutex_Locked;
```

Figura 4.8: Procedimiento Set_Deadlinefloor_Last_Mutex_Locked.

Se han realizado pruebas de convivencia de todas las políticas y el comportamiento es el esperado en todos los casos.

Capítulo 5

Conclusiones

5.1. Rendimiento del DFP

La implementación del DFP realizada en MaRTE OS, es la primera implementación del protocolo DFP y la única de la que se tiene noticia a la fecha de escritura de este trabajo. Gracias a esa implementación se ha podido evaluar su rendimiento en comparación con el protocolo SRP, previamente implementado en MaRTE OS.

De acuerdo con los resultados expuestos en la sección 3.3.3 se puede concluir que el DFP es un protocolo de sincronización de acceso a recursos compartidos en sistemas EDF más eficiente que el SRP.

Se ha encontrado que, usando la misma estructura de datos para su implementación (una cola de listas doblemente enlazadas) el DFP consume menos tiempo en la toma y liberación de un recurso que el SRP. Incluso sin recursos en el sistema, una tarea en la implementación con el SRP tarda más tiempo en encontrar el lugar en la cola de tareas ejecutables que la misma tarea en la implementación con DFP. Esto ocurre porque el protocolo SRP tiene que comprobar si hay recursos tomados en el sistema en cada paso del proceso de encolado.

Además la implementación realizada del DFP con una cola basada en un montículo binario como estructura para la cola de tareas ejecutables obtiene los mejores resultados de todas las implementaciones comparadas, siendo mucho más eficiente que el resto para sistemas con más de cinco tareas. Tal y como se ha explicado en la sección 3.2 esta estructura de datos no puede utilizarse para implementar el SRP y supone por tanto una gran ventaja del DFP sobre el SRP.

Estos resultados son la consecuencia de una las características fundamentales del DFP: no añade parámetros de planificación a los ya existentes en cualquier sistema EDF y por tanto no modifica las reglas de planificación. Esto garantiza la existencia de un orden total entre las tareas en la cola de tareas ejecutables. Además, el hecho de que el DFP sea conceptualmente mas sencillo que el SRP permite una implementación mucho más sencilla, como se ha podido comprobar al comparar sus implementaciones en MaRTE OS.

5.2. Propuesta de incorporación del DFP a Ada

El segundo objetivo de este trabajo es la realización de una propuesta para la incorporación del DFP al lenguaje Ada. El análisis realizado del protocolo DFP en el contexto de Ada ha demostrado que el DFP presenta suficientes ventajas sobre el SRP como para ser incorporado al estándar.

Se ha propuesto la incorporación de un nuevo identificador de política de planificación que defina las reglas de un sistema EDF & DFP y que se comporte de forma análoga al resto de políticas de planificación definidas en Ada. Se ha visto la necesidad de que en un sistema EDF & DFP, el parámetro de las tareas que marca su plazo relativo se convierta en un parámetro fundamental para la planificación y por tanto se proponen cambios en las operaciones existentes para sistemas EDF, así como la incorporación de operaciones para la gestión dinámica del plazo relativo.

En cuanto a las políticas de sincronización, se propone incorporar la funcionalidad del DFP a la política de sincronización ya existente Ceiling_Locking. Así solo sería necesario añadir un aspecto a los objetos protegidos que represente su suelo de plazo y un atributo para cambiarlo de forma dinámica.

Por último, todas las pruebas realizadas para comprobar la convivencia de la nueva política EDF & DFP con el resto de políticas soportados por el lenguaje Ada, han sido convenientemente analizadas y arrojan resultados positivos.

Por tanto, la propuesta que se ha elaborado es sencilla, no requiere cambios radicales en el estándar y no modifica ninguna de las otras políticas soportadas por el lenguaje Ada.

5.3. Trabajo futuro

El trabajo pendiente que hay que hacer en un futuro más inmediato es la redacción de la propuesta completa de incorporación del DFP al lenguaje Ada. Si bien, como se ha dicho, en este trabajo se abordan todos los aspectos fundamentales, sería necesario escribir la propuesta completa tal y como habría de aparecer en la próxima revisión del estándar Ada, prevista para el 2020.

También es un objetivo a corto plazo el incorporar el DFP a la versión pública de MaRTE OS ya que ha demostrado ser más eficiente que el protocolo actual, el SRP. A la luz de los resultados obtenidos en este trabajo, también sería conveniente cambiar la cola de tareas ejecutables de MaRTE OS para que utilice la estructura que ha probado ser más eficiente: el montículo binario.

Ni en este trabajo ni en la propuesta original del DFP [5] se han considerado sistemas con más de un procesador. Por tanto, otra línea interesante de trabajo futuro puede ser la extensión o adaptación del protocolo DFP a sistemas multiprocesadores. Sería interesante analizar las posibles adaptaciones del DFP a multiprocesadores y comparar su rendimiento con otros protocolos existentes en la bibliografía.

Bibliografía

[1] C.L. Liu and J.W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *JACM*, 20(1):46–61, 1973.

[2] T.P. Baker. A Stack-Based Resource Allocation Policy for Realtime Processes. In *Proceedings IEEE Real-Time Systems Symposium (RTSS)*, pages 191–200, 1990.

[3] T.P. Baker. Stack-Based Scheduling of Realtime Processes. *Journal of Real-Time Systems*, 3(1), March 1991.

[4] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. Comput.*, 39(9):1175–1185, September 1990.

[5] Alan Burns. A Deadline-Floor Inheritance Protocol for EDF Scheduled Real-Time Systems with Resource Sharing. Technical Report YCS-2012-476, Department of Computer Science, University of York, UK, 2012.

[6] M. Aldea Rivas and M. González Harbour. MaRTE OS: An Ada kernel for real-time embedded applications. In *Reliable Software Technologies, Proceedings of the Ada Europe Conference, Leuven*. Springer Verlag, LNCS 2043, 2001.

[7] ISO/IEC 8652:2012(E). Ada 2012 Reference Manual.

[8] POSIX: IEEE 1003.1-2008. *Standard for Information Technology - Portable Operating System Interface (POSIX) Base Specifications, Issue 7.*

[9] http://www.ada-auth.org/ais.html.

[10] ISO/IEC 8652:2007(E). Ada 2005 Reference Manual.

[11] A. Zerzelidis, A. Burns, and A.J. Wellings. Correcting the EDF protocol in Ada 2005. In *Proceedings of IRTAW 13, Ada Letters, XXVII(2)*, pages 18–22, 2007.

[12] Mark Louis Fairbairn and Alan Burns. Implementing and verifying EDF preemption-level resource control. In *Proceedings of the 17th Ada-Europe international conference on Reliable Software Technologies*, Ada-Europe'12, pages 193–206, Berlin, Heidelberg, 2012. Springer-Verlag.

[13] Michael T. Goodrich and Roberto Tamassia. *Data Structures & Algorithms in JAVA*, chapter 8, pages 320–321. John Wiley & Sons, Inc, 2006.

[14] Mario Aldea, Alan Burns, Marina Gutiérrez, and Michael González. Incorporating the Deadline Floor Protocolo in Ada. In *Proceedings of IRTAW 16, A la espera de publicación en Ada Letters*, 2013.